THESIS TITLE: A Skill-Based Framework to Analyze the Generation and Presentation of Instructional Videogame Content

AUTHOR: Britton Horn

Ph.D. Thesis Approved to complete all degree requirements for the Ph.D. Degree in Computer Science,

Thesis Advisor

Thesis Reader

Thesis Reader

Thesis Reader

Thesis Reader

4/22/2019
Date

4/23/2019
Date

4/22/2019
Date

4/22/2019
Date

4/22/2019
Date

GRADUATE SCHOOL APPROVAL:

Director, Graduate School

4/24/2019
Date

COPY RECEIVED IN GRADUATE SCHOOL OFFICE:

Recipient's Signature

4/23/2019
Date

Distribution: Once completed, this form should be scanned and attached to the front of the electronic dissertation document (page 1). An electronic version of the document can then be uploaded to the Northeastern University-UMI website.
A Skill-Based Framework to Analyze the Generation and Presentation of Instructional Videogame Content

Author: Britton HORN
Advisor: Dr. Seth COOPER

Committee Members:
Dr. Gillian SMITH (Co-Advisor)
Dr. Michelle BORKIN
Dr. Magy SEIF EL-NASR
Dr. Santiago ONTAÑÓN (External)

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in the
Khoury College of Computer Sciences

April 24, 2019
Abstract

A Skill-Based Framework to Analyze the Generation and Presentation of Instructional Videogame Content

Britton HORN

Video games are complex mental tasks that require mastery of numerous individual and interconnected skills that map onto the component mechanics of a game. Basic skills and mechanics typically build and depend on each other in a nested learning hierarchy, which game designers have modeled as skill chains of skill atoms. For players to optimally learn and enjoy a game, it should introduce skill atoms in the ideal sequence of this hierarchy or chain. Though players learn new skills from any game, instructional games explicitly attempt to guide a player through the acquisition of skills through the careful organization of levels and content. However, game designers typically construct and use hypothetical skill chains based solely on design intent, theory, or personal observation, rather than empirical observation.

Additionally, instructional games have begun to incorporate procedural content generation (PCG) into their design since it can offer several advantages over human-authored content including: personalized content, the inability to share answers between students, and endless practice. Existing PCG methods to analyze content typically focus on physical properties of the content in a synchronic point-in-time manner rather than a diachronic view of experience as players obtain new skills over time. A lack of rigorous PCG evaluation techniques makes understanding the effect of PCG on players a challenging task. Currently, no quantitative methods exist to analyze content based on the skills players should learn during gameplay without subjecting players to burdensome playtesting in order to gather data. Even when data is available, these methods struggle to adapt during the design and development of a game as changes may fundamentally alter the game’s mechanics and required strategies.

In this thesis, we move to understand three critical aspects of PCG in educational games: the effect of PCG on players’ gameplay behavior, the formalization of skill chains, and the automated analysis methods of content based on game skills to create level orderings. To evaluate the effect of PCG on players, we develop GrACE, a computer science educational game for middle-school students that incorporates PCG as a key mechanic. Additionally, we implement a mixed-methods approach to analyzing player behavior which sheds light on limits of current PCG metrics which failed to generate an adequate level progression. Next, we address the challenge of formalizing the creation of skill chains through an adapted cognitive task analysis method incorporating player feedback. Included is a critical reflection of our method to determine its strengths and weaknesses. Finally, we address the lack of methods for skill-based automated playtesting of content through the development of StrataBots which encapsulate the suite of player-acquired skills. We initially develop StrataBots for GrACE to improve the game’s level progression, but also demonstrate the generalizability of StrataBots with Monte Carlo Tree Search by applying this technique to the analysis of a human computation game called Foldit, thus allowing us to evaluate the difficulty curves of games with a more nuanced view of difficulty based on player understanding and skills.
Acknowledgements

First and foremost, I want to thank all of my friends and family who have helped me get to this point. Without their unending love and support, I would not have had the courage to pursue my dreams. I thank my brothers and parents for beginning my pursuit of a computer science degree from an early age. All those years of gaming, hacking, and tinkering developed my desire to learn about computers and the world at large.

I was very fortunate to have so many professors help me succeed and take me under their wing at some point during my academic career. Thank you to my original advisor, Gillian Smith, who helped me focus my research direction and taught me more about games, PCG, and being a good student than I could have ever imagined. To my advisor, Seth Cooper, thank you for finding a way for me continue my research interests in the realm of human computation games, supporting me as I completed my degree, and not letting me slip through the cracks after I felt lost. Thank you Casper Hartevedt for pulling me into the GrACE project, teaching me how to carefully design research projects, being a great friend at all those meetings and game nights at Punter’s, and teaching me the intricacies of Dutch puns. Thank you Michelle Borkin for always supporting me even though I wasn’t one of your students, and giving me the chance to realize how much I enjoy teaching. Without the classroom experience you gave me, I wouldn’t have pursued the career path I did. I also must thank Sebastian Deterding and Magy Seif El-Nasr for their support in my research career. Their guidance, direction, and knowledge has been invaluable.

I will never have enough time and room to write about all the things Amy Hoover has done for me in the short time I’ve known her. Amy has been my research partner, friend, confidant, protector, and teacher who constantly looked out for me no matter what. From the trips to Dunkin’, conversations on evolutionary algorithms, and advice on my personal life, I have never felt more support from one person than I have from Amy.

I am incredibly blessed to have been around amazing people in the Northeastern University community and the Playable Innovative Technologies Lab. Thank you Elín Cartensdóttir for keeping me (in)sane and grounded throughout the years. I couldn’t ask for a better cubicle partner and friend to talk to, bounce research ideas off of, take coffee breaks with, and tell me how bad my Icelandic pronunciation is. Thank you Anurag Sarkar for being my traveling partner at conferences and letting me practice talks with you. Thank you to everyone else in the PLAIT lab who has affected me in one way or another during my grad school years.

Without the hard work, dedication, and commitment of the entire GrACE team, much of my work would not have been possible. Thank you Oskar Strom, Hilery Chao, and Amy Stahl for your wonderful work and long hours. Also, I must thank Christopher Clark for putting up with my constantly changing parameters and interfaces for the GrACE puzzle generator. I hope I was more than merely a burden and helped you learn something new.

Thank you Santi Ontañón and Jichen Zhu for giving me a home away from home and a place to work when I moved to Philadelphia. Without a quiet space to think and write, I wouldn’t have been able to complete my work.

Thank you Thomas Wielemaker for continuing to be a great friend and always being available to chat. I have to thank all of the other people who gave me an outlet when I needed it: Jesse St. Clair, Katie Zarroli, Kyle Trecartin and Amy Trecartin. Thank you for the get-togethers, meals, games, vacations, and conversations that let me know how blessed I was to move to Boston and meet the greatest people.
Finally, and most importantly, I want to thank my wife for all of her love, support, and encouragement. Her constant companionship gave me the strength to move across the country and pursue my dreams while being as happy as I could possibly be. She is the hardest working, strongest, most supportive person I have ever met and reminds me how lucky I am every day.
# Contents

Abstract

Acknowledgements ii

1 Introduction 1
   1.1 Instructional Games 2
   1.2 Procedural Content Generation 3
   1.3 Player Modeling 4
      1.3.1 Analyzing Play 5
   1.4 Contributions 5
      1.4.1 Design Insights into Computational Puzzle Games 6
      1.4.2 Mixed-Methods Playtrace Analysis 6
      1.4.3 Skill Chains 7
      1.4.4 Hierarchical Game AI 7
   1.5 Game Descriptions 7
      1.5.1 GrACE 8
      1.5.2 Paradox 8
      1.5.3 Foldit 10
   1.6 Thesis Overview 11

2 Relevant Background 13
   2.1 Skill Chains 13
      2.1.1 Game Atoms 13
   2.2 Player and Student Modeling 16
      2.2.1 Student Modeling in Intelligent Tutoring Systems 16
      2.2.2 Player Modeling 18
      2.2.3 Modeling Skill and Challenge 20
      2.2.4 Simulated Play 21
   2.3 Game Analytics for Instructional and Educational Games 21
      2.3.1 Tools and Methods for Playtrace Analysis 22
   2.4 Creating and Analyzing Difficulty Progressions 23
      2.4.1 Human Computation Games 25
   2.5 Procedural Content Generation 25

3 Educational Game Design 28
   3.1 Computer Science Educational Games 28
   3.2 Related CS Educational Games 29
   3.3 Existing Cross-Cultural Game Analysis 30
   3.4 Methods 31
      3.4.1 Participants 32
      3.4.2 Nigerian Case Study 32
      3.4.3 Materials 32
      GrACE 32

iv
# Skill-Based Level Analysis

6.1 Introduction ................................................. 73
6.2 Existing Approaches to Difficulty Assessment ......................... 75
6.3 GrACE .................................................. 77
   6.3.1 Graphs and Minimum Spanning Trees ............................. 77
   6.3.2 Game Design ........................................... 77
   6.3.3 Level Generator ......................................... 77
6.4 Computationally Modeling Strategy .................................. 78
   6.4.1 StrataBot Design ......................................... 78
   6.4.2 StrataBot Descriptions .................................... 79
   6.4.3 Puzzle Classifications ..................................... 80
6.5 Experiments .................................................. 81
6.6 Results ..................................................... 82
   6.6.1 Player-Specific Results .................................... 86
6.7 Discussion ................................................... 87

# A Monte-Carlo Approach to Skill-Based Automated Playtesting

7.1 Generalizing Stratabots with MCTS .................................. 88
7.2 Related Approaches .............................................. 89
7.3 Human Computation Games ......................................... 90
7.4 Foldit ....................................................... 90
7.5 Study Methodology ............................................... 91
7.6 Results ....................................................... 92
7.7 Discussion ..................................................... 95

# Future Work

8.1 GrACE Design ................................................ 97
8.2 Progression Creation ............................................. 98
8.3 Stratabots ...................................................... 98

# Conclusion and Summary of Contributions

9.1 GrACE ....................................................... 99
   9.1.1 Cross-Cultural Evaluation ................................... 100
   9.1.2 Mixed-Methods Player Analysis ................................ 100
9.2 Learning Hierarchies ............................................. 100
9.3 Stratabot Implementation ......................................... 101
9.4 From Skill Chains to Stratabots ................................... 101

Bibliography ..................................................... 102
List of Figures

1.1 Screenshots of the games used throughout this thesis: (top) GrACE, (middle) Paradox, and (bottom) Foldit. ........................................ 9

2.1 An example skill chain. .......................................................... 14

2.2 Skill chain for Tetris, simplified from Cook (2007). .................. 15

3.1 Screenshot of a GrACE level where the player has fully explored each node and has returned the mouse to the starting node indicated by the bunny. The player has marked the minimum spanning tree by placing three flags on the correct edges. ........................................ 33

4.1 Every connected graph has a variety of spanning trees. A selected graph is shown in (a) with one of its spanning trees shown in (b) and its minimum spanning tree shown in (c). The minimum spanning tree is the set of edges that connects all nodes without cycles and with the minimum possible total edge weight. ........................................ 46

4.2 Screenshot of GrACE. This puzzle currently shows three nodes (as other nodes may be revealed later) with two edges, both with a cost of one, indicated by the arrow and number of rocks. Flagging involves literally placing a flag in the middle of an edge. On the top right are Hopper’s and Scout’s energy levels, to the left are the level select options, and at the bottom the submit button. ...................... 47

4.3 Example Player Progressions. Progression 1 shows a player progressing from level 1 to 2 to 3, repeating level 3 a few times, then progressing from 3 to 4 to 5, repeating level 5 many times, and finally going back to retry level 2. Progression 2 shows a player progressing from level 1 all the way to level 11 without repeating any levels. Arcs connecting dots from above indicate forward progression and arcs below the dots indicate a player going back to a previous puzzle. Loops above a single dot indicated retries of the same level. ......................... 50

4.4 Progression visualizations over the 11 levels by player, where Level 1 is represented on the leftmost dot, and Level 11 the rightmost. In a set, the topmost progression is a player’s first playthrough. The bottom progression is the final playthrough. .............................. 53

4.5 Clustering of successful playtraces for Level 1. Colored labels on the right indicate the player ID and run number. Some players completed one run (Blue and Violet) while others completed many (Green). .......................... 54

4.6 Graph depictions of levels in GrACE. Levels in GrACE can be abstracted into graphs, where each node represents a burrow for the character to investigate for vegetables and each edge the path to a particular burrow. The significant Levels 1, 3, 4, and 6 are depicted. ............... 54

5.1 A screenshot of the game Paradox analyzed in our case study. ........ 68
5.2 Detail of Paradox skill chain produced by our method. . . . . . . . . . . . 69
5.3 Simplified Paradox skill chain hand-authored by one of the game’s designers. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70

6.1 Puzzles from Difficulty-Stage Five. Four puzzles are shown from difficulty-stage five of the original GrACE implementation, chosen because they are each classified with different StrataBot labels. Players begin at node 3 on each of these puzzles. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 80
6.2 Player Progressions by StrataBot Classification and Success. Each puzzle attempt is represented by the x-axis while the particular difficulty-stage the puzzle is in is represented on the y. StrataBot classifications are represented by colors as shown in the legend, while success or failure on a puzzle is indicated by the ‘X’ and circle symbols respectively. 85

7.1 Hierarchy of Stratabot skills for Foldit. Each Stratabot has the skills it is connected to on the left. Shake is duplicated for layout purposes. . . . . 92
List of Tables

3.1 Comprehension test improvement across conditions, in $M (SD)$. The test is separated into three representations (original game aesthetic, road network metaphor and abstract) and three concepts (edge addition, edge deletion and completed tree correction). Conditions are broken down by Individual (Ind.) vs Collaboration (Coll.) and PCG vs No-PCG. .................................................. 37

3.2 Overview of comprehension test scores across the USA and Nigerian groups, in $M(SD)$. .................................................. 39

4.1 The nine participants and their questionnaire results from the pilot study and predominant strategy from analyzing their gameplay. Questionnaire scores include pretest (Pre) and posttest (Post) scores (each ranges from 0 to 9), enjoyment (Fun) of GrACE (composite score ranging from 4 to 28), how difficult (Diff) they perceived GrACE (also from 4 to 28), how often they played games (Freq), and their predominant strategy (Strategy) during their initial playthrough of the game. Enjoyment and difficulty are listed in terms of Low < 14 (L), Medium = 14-20 (M), or High > 20 (H). Freq of game play ranges from 1 to 6 (“Never or Almost Never” to “Several Times a Day”). Freq is listed by Low = 1 or 2 (L), Medium = 3 (M) or High = 4 or 5 (H). There are three strategies: deliberate (D), exploratory (E), or iterative testing (IT). ......................... 52

6.1 StrataBot Classification per Puzzle Difficulty-Stage. All 368 puzzles played by the students from the study are classified by the StrataBots. The particular stage of the puzzle is shown in the leftmost column while the total number of puzzles that fit a particular classification is shown in the rightmost column. In the middle are the number of puzzles in the particular difficulty-stage that StrataBots of given types can solve. The percentage of puzzles that each solves is in parentheses. 83

6.2 Success Rates of Human Players by StrataBot Classification. The player rates of success on puzzles of different StrataBot classifications are split into those played by people in the PCG and Static conditions and totaled at the bottom. Overall, puzzles classified as PrimsOnly presented the most challenge for players. ......................... 84
7.1 List of concepts taught in each of the first 16 tutorial levels of Foldit as outlined in Andersen et al. (2012). Some skills are not applicable to Stratabots since they are specific to human players such as translating and rotating the camera or color perception. Others require proxies since the mechanics provided by the Foldit scripting language do not match one-to-one with the concept (e.g. pulling sidechains). Repeated concepts are marked with a (+). We create each Stratabot from Foldit’s skill chain by combining one or more designer-specified skills along with all prerequisite skills.

7.2 Table indicating the set of skills modeled by each Stratabot. Skills present in a Stratabot are represented by a ●. Some Stratabots incorporate all the skills of other bots plus additional skills making them categorically superior when playing Foldit.

7.3 Results of each Stratabot playing the introductory levels of Foldit. If a bot can solve a level, it is indicated by a ●. Levels solvable only by more complex bots indicate that the levels may require additional skills present in some complex Stratabots but not others. For the skill makeup of each bot, see Table 7.2. Horizontal lines separate the four sections of tutorial levels. In general, each section begins with an easier level as indicated by more successful Stratabots, and gradually gets more difficult until the next section where it starts easy again. We found no differences between the hand-authored Stratabots and their Monte Carlo counterparts.

7.4 A listing of player success rates on the first 16 tutorial levels of Foldit separated by tutorial section. Player data gathered during this specific timeframe included players with sessions already in progress. This allows the successful attempts for one level to be lower than the total attempts for the next.
Dedicated to my wife Sarah and our growing family.
Chapter 1

Introduction

Playing a video game and many other everyday activities require engaging a complex set of interdependent skills (Thompson et al., 2013). Complex skills integrate a network of more basic skills: driving, for instance, requires independently mastering braking, steering, and switching gears, as well as executing and fluently switching between them (Merriënboer, Clark, and Croock, 2002). These constituent basic skills hang together in a learning hierarchy: the logical order in which they build and depend on each other and therefore, in which they are ideally learned. For example, we have to learn counting before we can learn addition and subtraction, and it is easier to learn these before multiplication (Baralt, Gilabert, and Robinson, 2014; Gagne, 1968; White and Gagné, 1974). Identifying the learning hierarchy of a to-be-taught complex skill is therefore a key task in designing interactive systems, including both serious (educational and citizen science) and entertainment-oriented game design (Jonassen, Tessmer, and Hannum, 1998). No matter if designers want to create good tutorials and level progressions for a game (Butler et al., 2015; McMillan, 2013); balance level difficulty or procedurally generate levels that fit player skills (Linehan et al., 2014); create instructional games whose mechanics train targeted capacities (Echeverría et al., 2012); or gamefully restructure everyday activities (Deterding, 2015)—they are faced with the question of what component skills a given game entails or ought to entail, and in what order the game should introduce these skills to players. More colloquially, if games are learning machines players enjoy to master (Gee, 2003; Koster, 2004), it stands to reason they benefit from a well-designed sequence of learning. If challenges are too difficult or too easy, the player can become frustrated, bored and disinterested (Csikszentmihalyi, 1990).

As a result, game design practitioners and theorists have developed a range of formal models that describe games as nested networks of feedback loops which revolve around specific actions or skills (Deterding, 2015). One particularly popular model, developed by Dan Cook, describes games as skill chains, directed graphs of skill atoms or core loops that logically build on each other—mirroring learning hierarchies in everything but name (Cook, 2007). At the base of Cook’s model is his definition of skill: “a behavior that the player uses to manipulate the world.” This broad definition allows a skill to be comprised of a single action, batched set of actions, or conceptual actions such as navigating a map.

Originating as war terms, tactics and strategies comprise the organization of an army and a plan of actions for military success (Von Clausewitz, 1940). In games, actions lie at the lowest level of interaction for players, including pressing individual buttons, moving a computer mouse, or moving the player’s body such as is necessary in virtual reality and gesture-controlled games. Tactics incorporate a series of actions to achieve local goals, whereas strategies are concerned with linking tactics on a global scale to enact long-term goals (Bakkes, Spronck, and Lankveld, 2012). An individual atom of Cook’s model may include any or all of these—actions, tactics, and strategies.
Work contained in this thesis primarily focuses on actions and strategies, though the delineation between the areas is often ambiguous and ill-defined in practice allowing one to argue tactics is included as well since some game strategies focus on local information.

Additionally, there are many design methods such as *Rational Level Design* for prospectively deriving optimal level progression sequences from a given atom model (McEntee, 2012; McMillan, 2013). Unfortunately, these models and methods provide little if any guidance on how to reliably deduce the skill chain or learning hierarchy of a given game. Models are either sketched as blueprints for a new game or based on a designer’s or researcher’s individual reading of a game. Scarcely any game research methods exist to empirically deduce the skill chain or learning hierarchy of a game from actual player experience, assess to what extent the skills and ideal sequencing order predicted by a model matches the skills it requires from players, assess the efficient acquisition of those skills by players, or the optimal learning hierarchy. This risks overlooking essential skills, not introducing them to players, or introducing them in a sub-optimal sequence.

Unsurprisingly, designers often require extensive playtesting or, through machine learning approaches, extensive data to accurately understand players. Constantly recruiting playtesters throughout the design of a game can be prohibitively expensive, especially for smaller games companies. Content changes so often that new testers must be continuously brought in if designers want to understand the effect of a specific change or risk their intuition being wrong at the game’s release. It stands to reason, the ability to automatically evaluate content from the player’s perspective could drastically reduce the testing burden on designers, both in time and money.

Amplifying the issue further, the addition of procedural content generation (PCG)—that is, content created through algorithmic means—generally removes the ability for designers to test specific content before releasing it to players. While players progress through a game, it is vital for designers to be assured that the dynamic content they present to players strictly adheres to the educational goals, aesthetic constraints, and learning hierarchy of the overall game. A common workaround is to seed the algorithm that generates content in order to consistently generate the same content. This severely limits the expressive range of a PCG system and removes the often touted benefit of PCG systems: replayability. Ideally, designers should understand the effects of content on players throughout the development of a game as changes occur regardless of the inclusion of procedurally generated content and without the manual inclusion of specific PCG content.

### 1.1 Instructional Games

Instructional games are a specific type of software that aims to teach concepts and knowledge to players through an entertaining platform (Doering and Veletsianos, 2009). Instructional and educational games focus on teaching players knowledge from a curriculum that, upon completion, makes them competent in the desired material. While educational games traditionally focus on material found in the classroom (but do not have to), they mirror instructional games in purpose and design thus becoming synonymous. Instructional games come in diverse forms with examples including typing drill-and-practice software *Mavis Beacon* (Cannon, 1999), math education software *Math Blasters* (Eckert and Davidson, 1987), the logic game *Logic Quest 3D* and various intelligent tutoring systems that simulate human tutors through the implementation of Artificial Intelligence (Kulik and Fletcher, 2016). Typically used for
teaching new skills rather than practicing existing skills or altering one’s perception of a concept (Dempsey et al., 1993), instructional games have the ability to reach a wider audience than traditional educational settings and potentially help more people learn a given topic. However, to take advantage of this expanded audience, designers must create emotionally appealing, appropriately challenging, logically structured and educationally effective games (Hays, 2005).

Both the creation and evaluation of instructional games pose challenges to designers for understanding the effect of content on players and its ramifications on knowledge acquisition (Harteveld et al., 2014). Assessing the effectiveness of a game requires more than simply evaluating gameplay (Harteveld, 2011)—we must understand the extent knowledge transfers outside the game to conventional environments. Games come in many genres with different target populations and various research goals such as promoting learning (Gros, 2007), engagement (Girard, Ecalle, and Magnan, 2012), enjoyment (Malone, 1980), or diversity (López and Cáceres, 2010). Evaluation and assessment techniques and materials that work for one game may not be appropriate for others (Munson et al., 2011). Additionally, there are implications to the order of skill introduction and assumption of pre-existing skills on player behavior and performance in instructional games (McEntee, 2012; McMillan, 2013).

1.2 Procedural Content Generation

Algorithmically and automatically producing material—including game levels, text, art, and data—is known as procedural content generation, or PCG for short. Often cited as a way to increase replayability in games (Smith, 2014), PCG allows for seemingly never-ending content. In fact, as games become more complex and detailed, content production by designers has become a major bottleneck in the development cycle of games (Smelik et al., 2009; Kelly and McCabe, 2006), creating an opportunity for PCG systems to lower the authorial burden on designers and decrease production costs. However, generators require an understanding of countless additional contexts including design constraints, aesthetics, and player experience, in order to produce quality content.

In regards to educational and instructional games, an often overlooked benefit of PCG is the ability to prevent sharing answers and solutions, or at least to make it more difficult. When two students are given identical problems, it is very easy for one student with more knowledge to explicitly tell the other student the answer. On the other hand, if the students always receive different problems, they must discuss how to solve them. Coupled with replayability benefits, this allows students to practice skills with a wider array of variety. It also requires content to be thoroughly understood—that is, algorithmically defined—in regards to skills required and difficulty. Smith (2014) states one capacity of PCG in games is practicing in different environments; that is, generating content in a variety of related contexts allowing for the player to practice game strategies outside a specific instance. What related means is left up to the designer. In instructional games, we view related contexts as those which teach common material at a similar difficulty. Thus, understanding what skills a level requires and the difficulty of that level is vital to the application of PCG in instructional games.

Analyzing procedurally generated content is still a challenging problem, especially concerning the relatedness of individual content. Typically, content analysis focuses on the physical features of content (Shaker, Yannakakis, and Togelius, 2011; Horn et al., 2014; Smith and Whitehead, 2010; Shaker, Smith, and Yannakakis, 2016;
Chapter 1. Introduction

Liapis, Yannakakis, and Togelius, 2013) rather than how the player experiences the content (Holmgård et al., 2014). Common markers delineating procedurally generated game content are level features such as number and placement of hazards, locations of resources, optimal paths, and success rates. These features remove the individuality of players and how each may experience content in a personal way. Instead, instructional game designers require a method to analyze procedurally generated content from the perspective of the player which takes their individual skills into account.

1.3 Player Modeling

Players are unique individuals of varied backgrounds, pre-existing knowledge, and mastery of skills who may behave drastically different when presented with the same situation. Particularly important for procedural content generation, game AI researchers model player skill to predict player performance in new environments and challenges. This thesis is primarily concerned with player modeling as it pertains to procedural content generation. In this field, accurate player modeling can help create more engaging games by generating content tailored for a specific player at a particular point in time (Bakkes, Whiteson, and others, 2014). While not exhaustive, Smith et al. (2011) provide a taxonomy of some current player modeling techniques, showing four major aspects that differentiate methods:

- **Scope**—intended audience for the model
- **Purpose**—whether the model is meant to generate player data or describe existing players
- **Domain**—if the model describes game actions or human behaviors outside the game
- **Source**—where data comes from to create the model

We use player models throughout this dissertation to evaluate the effect of content on players and call each model a *Stratabot*. In regards to **Scope**, StrataBots are applicable to a class of players—namely those with particular sets of skills—rather than one player or all players. However, our models could also be seen as hypothetical since certain cases may arise that do not resemble any human players. If there are sets of skills contained in the player model that a player never attains due to the game’s skill progression, there will be no players that reflect that specific model. The **Purpose** and **Domain** of our model is to generate game actions in the same format as human traces of actions. Finally, we use a synthetic **Source** for our model via skill chains rather than machine learning or other model derivations. While much research has commenced since the introduction of Smith et al.’s taxonomy, none of the included examples fall into the same category in each of the four facets as our model. More recent player models (such as *Procedural Personas* in (Liapis et al., 2015)) show more similarity to our method, however none to date generate hypothetical simulated players based on theorized accumulated game skills without the need for existing player data. Creating player models with unique combinations of facets may seem interesting in and of itself to some, yet others want to see the applicability of such a model to existing games. For this, we turn to one particular analysis method called playtrace analysis that helps better understand player and bot solutions to puzzles.
1.3.1 Analyzing Play

Analyzing a player’s solution to a puzzle can uncover potential strategies used and gaps in understanding (Harpstead and Aleven, 2015). The approach and act of interpreting a player’s actions during gameplay is called playtrace analysis. Many methods have been developed to trace and analyze a player’s gameplay (Fullerton, 2008) including traditional observational studies (Hilbert and Redmiles, 2000) as well as videotaping play and interview sessions (Ambinder, 2009); all of which are qualitative in nature, span various game genres, and are difficult to use in analyzing large-scale playtest data (Andersen et al., 2010). In contrast, statistical and machine learning techniques have been used to track and categorize players in, among others, Bioware (DeRosa, 2007), Forza Motorsport (Romero, 2008), and World of Warcraft (Duchenaut et al., 2006) by leveraging raw game data. For these quantitative approaches, scholars have developed and validated aggregate metrics from the raw data to measure game states or player behavior (Seif El-Nasr, Drachen, and Canossa, 2013), such as the number of attempts or level difficulty. Each of these methods have at their core an attempt to gain insight into player understanding and the effect of games on each individual. As part of this thesis, we outline and implement a mixed-methods approach to playtrace analysis through quantitative hierarchical clustering, qualitative playtrace assessment, and think-aloud player data.

1.4 Contributions

Due to the benefits and challenges of analyzing players, designers would benefit from an automated method for analyzing the effect of PCG on various types of players in games during game development. In furtherance of this goal, this thesis demonstrates the design, implementation and assessment of a computer science educational game with a PCG-based level progression, a formal method for creating skill chains, and a novel approach to simulating players at various points along a game’s learning progression. These stratified AIs, formed from qualitative user feedback, can play single-player computational puzzle games to reveal flaws in existing difficulty progressions and can predict player performance on unseen puzzles. In so doing, this thesis primarily contributes to the fields of procedural content generation and game design by developing and assessing an educational game with procedurally generated puzzles, producing a framework implemented in multiple games for extracting the individual skills required for a game and their relationships, designing AIs with those skills, identifying optimal orderings of challenges that promote mastering a particular set of skills, and evaluating player performance and learning outcomes.

In this thesis, we describe: 1) the creation and assessment of an educational game called GrACE with PCG as a core mechanic, 2) a formalization for skill chain elicitation, and 3) the implementation of skill chains using simulated players called “StrataBots” which can help designers automatically evaluate level progressions, even if they contain procedurally generated content. Each StrataBot represents one strategy; or, the algorithmically-defined rules for the application of a subset of skills present in a particular game. Specific games used in our work are GrACE (designed as part of this thesis), Foldit (Cooper et al., 2010b; Cooper et al., 2010a), and Paradox (Dean et al., 2015; Sarkar et al., 2017)—educational and citizen science games that have puzzle-like qualities. We detail methods for creating StrataBots, their applications in both understanding player behavior and learning in game contexts, as well as analyzing level progressions as learning progressions.
Chapter 1. Introduction

While defining an automated method for analyzing the effect of PCG on various types of players in games during game development, this thesis aims to answer three main research questions:

1. How does the introduction of procedural content generation in an educational game affect player behavior and performance?

2. How can the skills necessary to complete a video game be extracted and organized by dependencies?

3. How can the logical organization of video game levels be analyzed by required skills in order to promote learning without relying on existing player data?

To answer these three research questions, the work presented in this thesis outlines the analysis of player performance in an educational game containing procedurally generated levels, a method for skill extraction based on cognitive task analysis in order to develop StrataBots, and novel methods for level progression analysis based on expected player skills. By answering these questions, this thesis makes contributions to the three game research areas of game design, playtrace analysis, and artificial intelligence.

1.4.1 Design Insights into Computational Puzzle Games

In Section 1.1, we outline the difficulty in assessing the effectiveness of instructional or educational games, especially those which rely on PCG to form smooth learning progressions. To address our first research question, we discuss in Chapter 3 the importance of game design and its effect on players in educational games by creating GrACE, a computer science education game. Namely, we want to understand how the introduction of procedural content generation in the level design process changes players’ perception, experience, and mastery of a game. We expand this evaluation to an additional, non-western culture in order to identify specificities of cross-cultural game experience.

1.4.2 Mixed-Methods Playtrace Analysis

In furtherance of our attempt to answer our first research question, Section 2.3 gives examples of existing playtrace analysis methods and how they are important to understanding player behavior in games. We outline a mixed-methods approach to analyze player strategies in GrACE in Chapter 4 that highlights how different players progress through the game and what they do when they come across challenges too difficult for their current skill level, including the exploitation of GrACE’s PCG system. Our mixed-methods approach gains insight into deficiencies in the GrACE level progression, game bugs and inconsistencies, and various ways players deal with frustration. In this research we present a strategy for gameplay analysis that triangulates data from three perspectives: think-aloud voice recordings, qualitative playtrace analysis we call retrospective player sense-making, and quantitative clustering on player actions. We show how this mixed-methods approach can be applied to an educational game and how it helped us gain insights that may have been overlooked or incorrectly inferred without game analytics or using a single method to analyze playtraces. Finally, data-driven design and evaluation of an educational game allows us to detect problem areas early in the design phase where steps can still be taken to correct shortcomings.
1.4.3 Skill Chains

In Chapter 5 we demonstrate a formal method for skill chain creation in response to our second research question. This method formalizes the elicitation of skill chains for a computational puzzle game from qualitative player interviews to help designers understand whether or not skills introduced in a game match the order in which they should be ideally learned. Game designers often have an idea of the skills required for a game and an order that players acquire those skills. However, intuition can be misleading and qualitative user data may help gain insight to additional skills required, incorrect or alternate acquisition orderings, and challenging situations where users fail to acquire expected skills (Crandall, Klein, and Hoffman, 2006). This work shows that the same holds true when utilizing this method for games research.

1.4.4 Hierarchical Game AI

As shown in Chapter 4, players can get stuck at various points in a game’s learning progression and it can be difficult to understand exactly what caused the most struggles. The addition of procedural content generation into a learning system makes this even more challenging as the levels to be analyzed may only be seen once and are not played by every player. With the addition of a PCG system for puzzle generation, we see behavioral differences in players presented with procedural content generation available during gameplay versus those who do not. While the inclusion of procedural content generation into an educational game lets players see additional puzzles, it also allows players to take advantage of the generator by requesting new levels over and over until they get an “easier” one. Chapter 6 answers our final research question by describing the novel implementation of AI bots with varying levels of knowledge about GrACE puzzles in order to give designers additional insight into why players struggle on various levels even when they may seem identical in difficulty at first glance through level geometry. This allows for a player-centric focus on level difficulty in procedurally generated puzzles which designers can use to organize levels according to particular player skills.

Not only does this work contribute to the novel use of AI as analysis tools, it also contributes a novel method for the construction of AI bots with varying degrees of skill based on a game’s given skill chain. Bots are crafted through the careful selection of subsets of related skills in a skill chain and leveraged for the analysis of tutorial levels in a human computation game called Foldit. Crafting bots for computationally challenging problems such as protein folding, while novel in itself, opens the way for future research into player modeling and prediction. Additionally, we find that leveraging existing gameplaying algorithms and modifying them for StrataBot creation can reduce development time without the loss of predictive power. In so doing, we outline a framework to automatically assess procedurally generated content on the basis of player skill.

1.5 Game Descriptions

This thesis presents a framework that helps designers and researchers outline expected incoming knowledge and the desired learning outcomes of a game as well as methods to create content based on those outcomes and evaluate if the process is successful. The insights gained from this thesis may apply to various genres, but we focus on single-player, turn-based, goal-driven computational puzzle games for three reasons:
• player actions are discrete which heavily reduces the complexity required for analysis,
• single-player games allow the player full control over the state of each puzzle, and
• presented content is assumed to be structured in such a manner to promote the linear acquisition of game skills and knowledge.

Simplifying the problem context for novel Artificial Intelligence (AI) creation methods helps prove the validity of the approach by reducing possible mediating factors. We first explore computational puzzle games in two-dimensional graph-based puzzles and later expand to three-dimensional spatial puzzles.

We use three different instructional games throughout this thesis for educational game design, skill chain elicitation, and as test-beds for StrataBots that meet the game selection criteria outlined above: one educational game (GrACE), and two citizen science games (Paradox and Foldit).

1.5.1 GrACE

Created as part of this dissertation, GrACE is a puzzle game with a vegetable-collecting narrative designed to encourage algorithmic thinking. Algorithmic thinking involves reasoning about problems abstractly, identifying common traits so that they can be treated as a class of problems instead of a single instance, and building sequences of instructions as solutions. It is a logical method of problem solving that can also be applied in areas both inside and outside computing (Tucker et al., 2006). Specifically, GrACE aims to illustrate the potential of teaching algorithmic thinking through puzzles analogous to finding the minimum spanning tree (MST) of a graph, a core problem in computer science.

To avoid computer science jargon and create an interesting and relevant context, solving the MST-based puzzles centers around a narrative with two characters, Scout the mouse and Hopper the rabbit, whose combined goal is to collect all possible vegetables while expending the least amount of energy. In this narrative, burrows represent the graph nodes and each burrow contains a vegetable. The graph edges are tunnels that Hopper can dig to collect the vegetables. The cost of each tunnel is represented by the number of rocks through which Hopper must dig. A screenshot of GrACE is shown in Figure 1.1(a).

We crafted GrACE in order to evaluate the efficacy of procedural content generation and collaboration in an educational game. We hypothesize that algorithmically generating puzzles, coupled with a pair of students playing similar but different levels together, enhances a student’s ability to think about a problem abstractly and more easily synthesize a general procedure to solve any future GrACE puzzle.

Additionally, we evaluate StrataBots with GrACE because the player makes one discrete move at each game state, is a single-player game where no outside agents affect the game, and players complete the game in a linear fashion. The limited action space and simple graphs allow the straightforward application of simulated AI bots. As this is the least complex game environment, we select GrACE as the first application of StrataBots.

1.5.2 Paradox

Paradox was originally developed for crowd-sourced formal verification of software (Dean et al., 2015), in which players would assist in producing proofs of correctness
Figure 1.1: Screenshots of the games used throughout this thesis: (top) GrACE, (middle) Paradox, and (bottom) Foldit.
for computer programs. The game is designed as a 2D puzzle game in which each level represents a maximum satisfiability (MAX-SAT) problem. Players can use a combination of manual and automated tools to assign values to variables in the underlying MAX-SAT problem, and are scored based on how many clauses they satisfy. A player completes a level by reaching a pre-determined target score. Some levels in the game are not fully solvable, and in general we may not know if levels are fully solvable or not (i.e. if all clauses can be satisfied). Still, even partial solutions can potentially be useful. Paradox has successfully been used in several studies of player and level modeling (Sarkar et al., 2017; Williams, Sarkar, and Cooper, 2017; Sarkar and Cooper, 2017).

Similar to GrACE, Paradox meets our selection criteria stated previously yet is slightly more complex since each action the player takes has considerably more variables on a much larger graph. A screenshot of Paradox is shown in Figure 1.1(b).

1.5.3 Foldit

Foldit is an online game where players compete and collaborate to find well-folded protein structures. Foldit makes extensive use of the Rosetta biochemical software suite (Leaver-Fay et al., 2011; Rohl et al., 2004) for scientific accuracy in calculating energies and performing structural optimizations. In the game, players directly manipulate the protein by pulling on it, freezing regions, adding rubber bands, and numerous other actions. They have access to automated Rosetta algorithms including continuous optimization of energy, discrete rotamer optimization, and fragment insertion. In this way, the human players do high level restructuring of the protein and can allow the automated tools to refine the structures they find promising. Players compete on a leaderboard (ranked by the energy of the best structure they have found) with other players who are working on the same structures, and can form groups to work together on challenging problems. Foldit introduces the gameplay and scientific concepts necessary to succeed in the game through a series of tutorial level puzzles, so that a background in biochemistry is not necessary to play (Cooper et al., 2010b). The game is highly instrumented to gather and log data on player actions; the game is continually evolving and changing based on both gameplay and scientific data generated by players. A recent screenshot of Foldit is shown in Figure 1.1(c).

Thus far, Foldit players have: outperformed state of the art protein structure prediction methods (Cooper et al., 2010a); created models of sufficient quality for successful molecular replacement and subsequent structure determination of a monomeric retroviral protease (Khatib et al., 2011b; Gilski et al., 2011); participated in extensive backbone remodeling of a computationally designed bimolecular Diels-Alderase, increasing the activity of the enzyme (Eiben et al., 2012); discovered structure prediction algorithms that outperformed previously published methods (Khatib et al., 2011a); and fit electron density data to produce structural models of higher quality than trained crystallographers (Horowitz, Koldewey, and Bardwell, 2014).

While multiplayer components exist in Foldit, such as leaderboards and groups, a single player can complete each level without other players impacting the level state at any point during gameplay. Additionally, though player actions are significantly more complex with the introduction of continuous variables, each action is still separate and discrete from other actions made. Finally, we assume Foldit tutorial levels are in linearly increasing difficulty, however we hold no such assumption with community crafted content. Each of these discrepancies shows the increased challenge applying StrataBots to Foldit as compared to GrACE. The application of
StrataBots to Foldit demonstrates their broad applicability to instructional games in general.

1.6 Thesis Overview

The work outlined in this dissertation begins in Chapter 2 with an overview of existing research in relevant and related areas to the research contained herein including skill chains, player and student modeling, game analytics, and procedural content generation. Next, in Chapter 3, we describe the creation of GrACE—an educational computer science game for young students where PCG is a core feature allowing for the assessment of knowledge acquisition by players to determine the extent to which the inclusion of PCG and collaborative play effects player behavior and performance—as well as novel tools and methods for evaluating the efficacy of the game including whether knowledge transfers outside the game context. We analyze both player performance and the quality of our assessment materials in an attempt to evaluate the efficacy of GrACE for teaching a particular CS concept and lay the foundation for how to answer our first research question: How does the introduction of procedural content generation in an educational game affect player behavior and performance? Challenges encountered throughout this process highlight the need for automated analysis tools for procedurally generated content with a diachronic, player-centric focus. While we strive to make GrACE appealing to a broad audience including under-represented and disadvantaged students as recommended by Hays (2005), few educational games attempt to reach these groups across cultures. We complement the initial assessment of GrACE with a cross-cultural evaluation to understand player performance and behavioral differences across backgrounds and aptitude.

In Chapter 4, we answer our first research question by analyzing player performance from our initial GrACE study in detail using a mixed-methods approach we call retrospective player sense-making that includes quantitative hierarchical player clustering, qualitative playtrace analysis, and think-aloud player commentary. We identify three broad strategies players employ during gameplay: iterative testing, deliberate, and exploratory; however, these strategies do not correlate with player performance within the game or on the knowledge transfer assessment materials. Because of this, we thoroughly evaluate puzzle complexity and progression to identify deficiencies in the puzzle generator and re-evaluate our understanding of puzzle complexity. Additionally, we focus on players who weren’t presented with PCG puzzles due to the absence of an existing player-centric method for the automated assessment of procedurally generated content which would allow us to evaluate puzzles from the perspective of players with varying degrees of concept mastery. The rest of this thesis outlines a formalized method for such evaluation in order to close the existing gap in analysis strategies for PCG.

In response to our second research question, How can the skills necessary to complete a video game be extracted and organized by dependencies?, Chapter 5 lays out a formal process to elicit the skill chain of an existing game by appropriating cognitive task analysis methods from psychology for the interpretation of complex mental tasks. We describe some similarities between skill chains and concept maps which allow us to leverage common techniques from cognitive task analysis typically used in the construction of concept maps for the purpose of creating skill chains. Player interviews coupled with video-aided recall sessions enable us to delve into the inner workings of players’ thought processes while attempting to solve puzzles. As a case study, we put our formal method into action with Paradox—a citizen-science
game with a graph-based layout similar to GrACE, but moderately more complex allowing for the potential of additional optimal strategies. To evaluate the success of our method, we compare the skill chain developed with our formal process to that of a skill chain developed by one of Paradox’s designers.

Chapter 6 details our initial creation of StrataBots for GrACE in order to automatically assess which skills are required for each puzzle and classify puzzles based on those findings in response to our third research question: How can video game levels be logically organized by required skills to promote learning? Analyzing player success rates in parallel with StrataBot classifications shows a strong correlation between classification and how likely players were to complete a given puzzle thus demonstrating the validity of our approach. However, our initial approach creates StrataBots specifically for GrACE and are not easily extensible or transferable to other games. We detail obstacles, challenges, and benefits of the StrataBot creation process as well as necessary improvements to the GrACE puzzle generator for future releases based on our findings.

Based on the formalization found in Chapter 5 and to ground the development of StrataBots to formal skill chains in response to our third research question, How can video game levels be logically organized by required skills to promote learning?, we construct a skill chain for Foldit and develop StrataBots in Chapter 7 that incorporate various combinations of atoms from the skill chain. Requiring many more StrataBots than GrACE for adequate skill coverage, we find the development of additional Foldit bots to be prohibitively time-consuming. We move to create a general approach for the development of StrataBots by leveraging a popular general game-playing algorithm known as Monte Carlo Tree Search (MCTS) which provides a game-agnostic approach to competent simulated players. We find that development time of additional MCTS bots is substantially reduced without losing predictive power.

Finally, in Chapter 8, we describe potential directions for future research including new avenues for StrataBot application, GrACE design updates, and coupling StrataBots with the procedural generation of levels. In Chapter 9, we conclude this dissertation with a summary of the work presented herein and the contributions it provides to the game design and artificial intelligence communities.
Chapter 2

Relevant Background

Outlined in section 1.4 are the three main research questions this dissertation aims to answer. This section covers the relevant background from related fields for each of these questions in order to situate the contained work. As such, we first cover a core game design methodology known as Skill Chains—their formalization, elicitation, and application to games. Next is a review of major player modeling techniques for personalized content in games, including applications to simulate player behavior, as well as similar student modeling techniques in the related field of intelligent tutoring systems. This is followed by an overview of game analytics and their application to educational game design. Next, we review difficulty progression creation and analysis techniques which aim to pair players with appropriately challenging game content. Finally, we detail a brief history of procedural content generation in games and some current PCG analysis techniques and their limitations.

2.1 Skill Chains

Church (1999) initiated contemporary work on “formal abstract design tools”: developing grammars and tools to describe, analyze, and design the structural components of a game (for recent reviews, see (Almeida and Silva, 2013; Dormans, 2012)). Following Almeida and Silva (2013), one can roughly distinguish (a) broad models like the MDA framework (Hunicke, LeBlanc, and Zubek, 2004), (b) collections of descriptive terms and patterns (e.g. (Björk and Holopainen, 2005)), (c) design guidelines such as playability heuristics (Koeffel et al., 2010a), and (d) modeling languages and tools of the core mechanics of a game, such as Machinations (Dormans, 2012; Adams and Dormans, 2012). Game mechanics describe the “core verbs” or “methods” by which players change the game state, such as moving, shooting, or trading (Sicart, 2008). They form part of game atoms (Koster, 2013) or game loops (Sicart, 2015) – feedback loops between player input (invoking a particular mechanic, e.g. shooting), rules processing (e.g. adjudicating whether the shot hit), and computer output. A game atom is the smallest indivisible functional unit of a game. However, games are usually composed of nested networks of interlinked atoms: In a cover shooter, the “shooting” atom is part of a larger “defeating enemies” atom, which also entails a “cover” atom and may connect to an “upgrade weapon” atom, etc.

2.1.1 Game Atoms

As noted, game atom modeling is highly similar to modeling the learning hierarchies of complex skills. Both capture nested relations of basic to complex capacities, mechanics here, skills there – with one crucial difference: most game atom models concern themselves with a synchronous overview of the game and how the outputs of one atom (e.g. in-game resources like health or experience points) feed into others
Chapter 2. Relevant Background

FIGURE 2.1: An example skill chain.

(Almeida and Silva, 2013; Dormans, 2012). They do not capture the diachronic sequence in which players (ought to) acquire proficiency in each atom. The exception is the skill atom model first articulated by Cook (2007) and since extended by Deterding (2015). It expressly models game mechanics and their relation from the perspective of player learning. A skill atom describes a game loop between player and game comprising five elements:

1. **action:** the player invoking a mechanic (e.g. shooting);

2. **simulation:** the game processing the action according to rules and changing its internal game state (adjudicating whether the shot hit, changing the location and health score of the hit enemy);

3. **feedback:** the game informing the player (displaying an animation of the hit enemy);

4. **challenge:** the parameters that make executing this particular action differently easy or difficult; and
5. **synthesis**: the player incorporating the feedback, adjusting their mental model of the game state and improving the skill(s) required to master this particular atom (e.g. fast hand-eye coordination to aim and shoot).

Skill atoms exist in linked **skill chains**: directed graphs of the order in which skills build on each other and in which players necessarily or ideally acquire them (Cook, 2007). For instance, a player has to know how to **equip** a gun before learning how to **aim** and **shoot** with it. Skill chains bottom out in **pre-existing skills**: capacities game designers can assume players already bring to the game. Most PC games assume that players know how to move and click a mouse, for instance. Figure 2.1 presents a simple skill chain of one pre-existing skill, two basic skills, and one advanced skill that builds on them. Figure 2.2 shows a skill chain for the game **Tetris**.

Cook’s model has found rich practical application, particularly in applied game design. For instance, Echeverría et al. (2012) used it to improve an educational physics game. They analyzed which physical concepts the game ought to teach and which concepts it actually incorporated as skill atoms. Redesigning the game to incorporate previously missing skill atoms led to statistically significant learning improvements. Deterding’s (2015) method for gameful design similarly uses skill atoms to tease out the latent ‘mini-games’ of existing real-life activities and then redesign these to make them more explicitly and enjoyably game-like.

While not using Cook’s explicit articulation, Rational Level Design (RLD) (McEn- tee, 2012; McMillan, 2013) has brought game atom analysis to broad use in entertainment game design, chiefly for difficulty balancing. Following flow theory (Csikszentmihalyi, 1990), RLD assumes that players have an optimal or “flow” experience when the difficulty of challenges presented matches players’ skill. As player skill grows over time, games have to increase difficulty in lockstep to avoid frustrating or boring players. This raises the question how to systematically design the **difficulty curve** of a game – the rate at which it increases difficulty. To this end, RLD suggests to identify (a) the atoms of a given game and (b) the parameters which affect the challenge of each game atom. For instance, the difficulty of “shooting” may be affected by parameters like enemy distance and speed. Designers should then craft a level sequence that systematically (a) introduces and involves new mechanics and (b) varies and increases the difficulty of the parameters of each atom. RLD essentially translates a synchronic map of game atoms into a recipe for diachronic
level progression. However, RLD is chiefly interested in difficulty as an aggregate effect of the number of atoms involved and the configuration of their parameters. Unlike skill chains, it doesn’t concern itself with logical or pragmatic dependencies – how skills build on each other.

In summary, skill atom chains formally model the component mechanics and skills of a game and their logical dependencies. Thus, they lend themselves readily to map a game’s learning hierarchy. Current game user research, applied gaming, and intelligent tutoring research provide no reliable empirical method to identify the learning hierarchy or skill chain of a given game – the actual skills a player needs to acquire to master a game, and the actual order in which they build on each other and therefore should be introduced. Existing methods are limited to either (a) charting what mechanics a given game practically includes and introduces in what order (rather than should include or sequence to match the empirical learning hierarchy), (b) generating, testing, and optimizining level progression in terms of difficulty given an initial model, or (c) testing the statistical fit of a given model.

2.2 Player and Student Modeling

Due to substantial overlap of goals, methods, and models, no overview of player modeling in educational games is complete without some background on student modeling in intelligent tutoring systems (ITS). Each of these fields strive to guide knowledge acquisition and understand users better by directly observing them or analyzing data after a user interacts with a system.

2.2.1 Student Modeling in Intelligent Tutoring Systems

The application of artificial intelligence to the field of education dates back to at least the 1970s. Intelligent Tutoring Systems (ITS) specifically attempt to replace or augment a human tutor by providing material to teach, when to teach it, and how it should be taught (Nwana, 1990). There are numerous types of ITS falling into various categories such as constraint-based modeling, content sequencing tutors, Item Response Theory, and Bayesian Knowledge Tracing. The continued reception of all these model types comes down to ITS giving immediate, individualized feedback and instruction to a wide range of students (Desmarais and Baker, 2012). Critical to their success, ITS attempt to detect the moment students learn a particular concept as well as detecting off task behavior (Baker, 2007; Baker, Goldstein, and Heffernan, 2010). All of these tasks are traditionally done by a human tutor, however time constraints and availability mean that not all students have access to a human tutor. Merrill et al. (1992) compare human and computer tutors and find that both are good at keeping students on track when they begin to detect a student’s problem solving is going wrong. Human tutors focus student attention on incorrect parts of a solution without verbalizing what went wrong thereby letting the student figure it out. ITS instead point out what the student did wrong. Essentially, human tutors perform a diagnosis of students but do not communicate that diagnosis to the student.

While we’ve seen many advances in ITS throughout the decades (an in-depth list of existing systems, categorizations, and design tradeoffs can be found in Murray, 1999), modeling student performance remains difficult for three reasons:

1. We infer student knowledge from observation with no way to directly see what a student knows
2. Student knowledge does not reflect perfectly in student performance due to guessing or knowledge slipping (forgetting information)

3. A student’s knowledge changes over time (Chang et al., 2006).

One approach to solve these problems is Dynamic Bayes Networks which reason about uncertainty (student knowledge) in time series data (period of interaction between a student and an ITS). Bayesian networks have been used in math, computer programming, and reading (Baker, Corbett, and Aleven, 2008). Chang et al. (2006) create a toolkit that allows users to input their data as well as an XML-specified model of learning and train and test this model with their data. The input model is theory-based from researchers and describes the causal relationship between student knowledge and behavior. The model developed in this research evaluated their toolkit on 360 students between six and eight years old using a reading tutor and compared their results to the corresponding Knowledge Trace (KT) model. They found their model to more accurately predict student performance than the KT model. One particularly challenging part of using this toolkit is creating the relationship between student knowledge and behavior. In a game setting, this generally is not straightforward.

Similarly, Baker, Corbett, and Aleven (2008) use Bayesian Knowledge Tracing to represent student knowledge. By creating a continuous assessment of student knowledge for a particular skill through probabilities, Bayesian Knowledge Tracing estimates that a student knows a skill by updating a probability based on student behavior. In Bayesian Knowledge Tracing, each problem step in the tutor is associated with a single cognitive skill. Knowledge of a skill is binary (either a student knows it or not). A student who does not have a skill will answer a given question that requires that skill incorrect—with a guess parameter that adjusts this probability above zero. This research updates Bayesian Knowledge Tracing with contextual estimation of guess and slip (rather than fixed probabilities) so that the model does not succumb to model degeneracy. A model becomes degenerate when students who know a skill are more likely to get the item in question wrong or students that don’t know a skill are more likely to guess a question correctly.

González, Burguillo, and Llamas (2006) compare Bayesian network models with case-based reasoning (CBR) systems to determine their effect on student performance as well as development effort. While Bayesian networks are a common and broad approach to ITS with numerous successes, they are highly complex in their design, computational resources, and knowledge acquisition or definition process. CBR systems help evaluate student performance to decide when to give hints, retry strategies, and change teaching plans. The main advantages of CBR systems include:

• easier to update and maintain the student model,

• promotes student reflection through reporting misconceptions and why they occurred, and

• facilitates continuous tutor supervision of students by letting tutors access both quantitative and qualitative information.

In addition to the methods already stated, some approaches to ITS appear similar to skill chains used in games as described in Section 2.1. Partial Ordering Knowledge System (POKS) (Desmarais and Baker, 2012) and conceptual map models (Hwang, 2003) are two particularly relevant cases where each bit of knowledge a student should learn is organized graphically to indicate how a student should acquire and
progress through all concepts of a subject. Given the relevance of these approaches to games, it is interesting to note the limited adoption for use prior to this thesis (O’Rourke et al., 2015).

2.2.2 Player Modeling

Each player is unique and reacts to various gameplay situations in different ways. Player modeling attempts to understand, describe or predict player behavior when faced with diverse game states (Yannakakis et al., 2013; Smith et al., 2011a). Commonly used during content design or generation, player models can predict player behavior on levels without subjecting the player to countless playthroughs (Togelius, De Nardi, and Lucas, 2007; Smith et al., 2011b; Liapis et al., 2015; Shaker, Yannakakis, and Togelius, 2010), yet AI-based solutions are difficult to design because modern games can be complex with many strategies (Jaffe et al., 2012). While player modeling is often analogous to opponent modeling, it can be used in single player games to adapt to players as well as smoothing learning curves for improved game experience (Charles et al., 2005a).

Two major surveys (Bakkes, Spronck, and Lankveld, 2012; Smith et al., 2011a) outline common approaches to modeling players. Bakkes, Spronck, and Lankveld (2012) describe player modeling from the perspective of the player and divide existing approaches into three major categories: player strategies, tactics and actions. They assemble current research to convey possible applications of player modeling as well as various approaches to attain particular goals based on the categorization of a model. Strategies are the highest level player model and focus on long-term goals. These feed into player tactics which are goals on a local scale. Finally, game actions are the lowest level and are the direct inputs a player has to the game environment. Modeling can be done at any of these levels and commonly dictates the approach. In addition, this survey briefly outlines how player profiling may enhance existing player modeling approaches by providing a psychologically or socially motivated cognitive model to complement the player model. While a player model attempts to model specific game behavior and actions, player profiling models external behavioral attributes. Smith et al. (2011) take a somewhat different approach by focusing on the overall process rather than specifically looking at game behavior creation. Their taxonomy divides player modeling approaches into four areas: scope, purpose, domain and source. Scope indicates to whom the model is applicable—individuals, classes of players, all players or hypothetical players. Purpose denotes the use of the model either for generating or describing players. Domain describes whether the model regards game actions or human reactions outside the games such as emotional and gaze responses. Bakkes, Spronck, and Lankveld (2012) largely ignore this distinction and concern themselves with game actions. Finally, source describes where the data comes from—either empirically or theoretically. Empirical sources include actual game data which lends itself well to automated approaches like machine learning, or outside observational data which requires some form of human intervention to map the existing data to informative descriptions. Theoretical sources include both analytic and synthetic methods. Analytic methods extract game knowledge by leveraging game rules or systems, and include approaches like theorem proving, optimization, search or other rule-based systems. Synthetic sources leverage some concept outside the game including fun, challenge, experience or knowledge. Based on the taxonomy given by Smith et al. (2011), the StrataBot framework outlined in this dissertation describes a hypothetical, generative, synthetic, action model.
Common player modeling applications involve adapting game content to fit a particular player or class of players (Togelius, De Nardi, and Lucas, 2007; Liapis et al., 2015; Smith et al., 2011a; Bakkes, Sprock, and Lankveld, 2012; Togelius, Shaker, and Yannakakis, 2013; Burelli and Yannakakis, 2015). Charles et al. (2005) define adaptive game design as dynamically changing a game through changes to: a player’s character, NPCs, the game environment, or game state. Additionally, Charles et al. (2005) overview various ways that player modeling, user testing and user-centered HCI combine to benefit adaptive game design. Proposed in this overview is a framework for adaptive game design that begins with a model of players and their preferences which feeds into a system that monitors player performance, adapts the game to the player, measures the effectiveness of that adaptation, and updates player models where appropriate based on effectiveness.

Understanding player behavior remains a difficult research problem. Drachen, Canossa, and Yannakakis (2009) attempt to understand player data in Tomb Raider: Underworld through k-means clustering, hierarchical clustering, and a type of neural network called self-organizing maps. Results show four clusters of players: players that die rarely and complete the game quickly, players that die often and complete the game slowly, players that do not request help often and mainly die to opponents, players that are fast to complete the game and request help often. Though these methods can be applied to other games, extensive player data is required and the results do not shed light on why players fall into one of these categories.

Togelius, De Nardi, and Lucas (2007) evolve player models of individual players in order to procedurally generate racetracks using certain aspects of player style such as average driving speed or deviation from a racing path with fitness functions for a player-centric model of gameplay experience. They utilized a particular evolutionary algorithm called cascading elitism that handles multiple fitness functions to generate tracks and evolve controllers. In addition, this research classifies the player modeling approaches into two major categories: indirect and direct. Direct modeling uses supervised learning to directly associate car state with the actions the player takes. This research uses two direct modeling approaches: multilayer perceptrons and nearest neighbor classification. Both models failed to adequately model players due to numerous unseen states that the training game data did not cover. They also employ indirect modeling by combining a robust controller with player style data such as average driving speed and racing line deviation. Based on these controllers, tracks were then generated to “challenge” the player based on which controllers struggled to make quick progress through the tracks. Though this research begins to portray how different players with different styles experience the same content, it requires extensive data to train the models. Thus, it is not adaptive to future design updates (e.g. altering physics constants to make cars more/less stable, adding new vehicles, modifying input mechanics).

Holmgård et al. (2014) extend this method by modifying the fitness functions to represent player goals rather than playstyle. They develop procedural personas that play levels from classic-style 2D dungeon crawlers and predict how players experience levels based on the goals they have while playing (e.g. collecting coins, finishing the level quickly, killing monsters, etc.). Rather than developing personas to predict player behavior, Liapis et al. (2015) incorporate personas into the dungeon creation process as level critics. Not only are levels generated to maximize the utility for a particular persona, they are also generated to maximize risk/reward in order to create decision points during playthroughs rather than allowing trivial playthroughs by the agents. This addition created instances where monsters guard the treasures and potions rather than them being safely alone. Additionally, Holmgård et al. (2015)
develop a method to create procedural personas with a Monte-Carlo Tree Search (MCTS) algorithm where the various goals are encoded by replacing the standard UCB1 search criterion with an algorithm obtained through evolutionary computation for each persona. As playthroughs of dungeons can quickly become infinitely long due to repeated cyclic actions, simulated playthroughs are stopped after 10 actions before back-propagating scores. It is unclear whether the evolved personas converge with traditional MCTS as simulation time increases which would mean the differences in personas can be attributed to exploitation of a limited search space.

In regards to skill-based player modeling, we build on the concept of procedural personas by developing a series of hierarchically organized AIs that model players at various points along a learning progression. We manually deduce major points in an educational game’s learning progression and craft a specific player model for that point in time. Furthermore, our use of MCTS limits possible actions taken rather than defining high-level goals by modifying the search criterion though this could be expanded on in the future.

2.2.3 Modeling Skill and Challenge

Creating appropriately challenging content is generally approached from one of three directions: modeling the overall aptitude of players, modeling individual skills, or ignoring players and focusing on specific metrics of a level (e.g. number of enemies, length of level). Specifically related to modeling player aptitude, Avontuur, Spronck, and Van Zaanen (2013) attempt to classify players of Starcraft 2 into their respective league (i.e. aptitude level). Numerous attributes from game logs inform the classification such as actions per minute, number of units built, hotkeys used, and minerals mined. Results show fairly good classification with misclassification distance being very low meaning that when a player was not accurately identified for the right league, the classifier generally put them into a neighboring league.

Zook et al. (2012) design a mission generator that takes into account author pre-defined goals including pacing and difficulty curves to generate missions in a game tailored to a particular difficulty curve. Tensor factorization helps inform the model of player aptitude which is then combined with the generator to construct missions for the desired curve as well as expected player performance. Player modeling is done at a high level and does not model individual features or skills that may attribute to the overall difficulty.

Sorenson, Pasquier, and DiPaola (2011) attempt to maximize fun and create oscillating periods of high and low challenge in Super Mario by generating levels in rhythm groups (Smith et al., 2011b). The evolution-based generator predicts the entertainment value of a level based on the presence of rhythm groups as well as a desired difficulty curve. Combined with a constraint satisfaction algorithm, the generator ensures the level is playable and certain design constraints are satisfied (using feasible-infeasible evolution and A* search). While player experience is a central focus of this research, it fails to describe how players of different skills experience the same content since difficulty is ascertained through level features.

Harpstead and Aleven (2015) use machine learning techniques to develop player models based on the skill chain concept (Cook, 2007) in order to inform designers what skills are exercised by players in an educational game. They create models ranging from only one skill synonymous to “playing the game well”, to individual models for single levels assuming that each level teaches new independent skills, and finally to granular models more representative of composite knowledge we would expect from traditional educational games (Linehan et al., 2014). This research
critically identifies how players with different skills experience the same educational content but requires extensive user data to develop each model.

Finally, Togelius, Shaker, and Yannakakis (2013) argue for the use of “active player modeling” which is where the learning algorithm attempts to find curious and undiscovered places in the search space to label during the training process. The selection is generally done based on which next instance the algorithm will learn the most from (probabilistic learning algorithms). This is put into example with Super Mario where the generator has seemingly infinite level parameters to use during level generation of online, real-time levels. It selects a set that it will learn the most from and generates a level with those parameters to present to the player. Upon completion of the level by the player, the player model is updated and a new level is generated.

2.2.4 Simulated Play

Chaslot et al. (2008) state that game AI requires domain knowledge and a long development timeframe, but implementing a Monte Carlo-based solution can reduce both of these. Monte Carlo solutions rely on simulated playouts to evaluate game moves rather than domain specific heuristics creating a flexible yet powerful technique. Successfully implemented for a range of single-player and adversarial games, including Go, Scrabble, Solitaire, and Settlers of Catan (Kocsis and Szepesvári, 2006; Browne et al., 2012; Chaslot et al., 2008), Monte Carlo implementations generally focus on creating optimal AIs. Varying the performance of a Monte Carlo-based AI through decreasing the simulation depth or overall runtime allows players the opportunity to play against sub-optimal opponents (Zook, Harrison, and Riedl, 2015; Browne et al., 2012; Baba Satomi, Iwasaki, and Yokoo, 2011). However, players vary in skill by more than the time they think about a problem or how far in advance they plan moves, players also differ in how well they grasp the mechanics of a game. Restricted Play analysis (Keehl and Smith, 2018; Jaffe et al., 2012) lets designers see general trends in gameplay, alter game mechanics, and evaluate the effect those changes have on players. Keehl and Smith (2018) created a Unity tool to streamline this process along with a proof-of-concept and let designers analyze the effect of design changes on players with three distinct playstyles. This is the first MCTS solution we found varying playstyle through more than computational depth or runtime, though the playstyles are game specific and focus on when the simulated player performs a specific action (collecting a game piece) rather than what actions they perform.

2.3 Game Analytics for Instructional and Educational Games

As already stated in Section 1.1, instructional and educational games come from various genres and teach a wide range of topics. Although there is a long and rich history of developing games for education, such games often lack rigorous evaluation (Harteveld, 2011). While used in game development, evaluating learning using game analytics is relatively new. Given time constraints of educators or researchers to distribute and evaluate traditional assessments, or code interview data, game analytics is a promising alternative to efficiently and automatically evaluate performance and learning. In fact, some scholars argue that game data itself is an assessment of learning, provided that appropriate metrics have been defined (Shute and Ventura, 2013). Shaffer and Gee (2012) even state that “We have been designing games for
learning when we should have been designing games for testing” (p. 3). As evidence of how game data can act as an assessment of learning, one recent study showed a strong correlation between game performance and external test measures (Harteveld and Sutherland, 2015).

Further, analyzing game data should not only determine whether a design works, but also how it works. For example, in a study that analyzed both efficacy measures as well as game data (Harteveld, 2012), it was found that the game was an effective training intervention, but looking closely at the playtraces—or, sequences of actions a player performs during gameplay—indicated that players consistently made the same errors, suggesting the design itself could be improved. Asking only efficacy questions can discount other aspects of player experience and potentially result in incorrect conclusions. Given this, game analytics can facilitate data-driven game development, which can expose design problems at an early stage. Although such data-driven design is becoming increasingly common in the industry (Seif El-Nasr, Drachen, and Canossa, 2013), it is not yet a common practice for educational games. Data-driven game development has the potential to assist in the formative evaluation of achieving learning objectives (Harteveld, 2011).

2.3.1 Tools and Methods for Playtrace Analysis

Many methods have been used for playtrace analysis (Fullerton, 2008), including traditional observational studies (Hilbert and Redmiles, 2000) as well as videotaping play and interview sessions (Ambinder, 2009), all of which are qualitative in nature and are difficult to use in analyzing large-scale playtest data (Andersen et al., 2010). In contrast, statistical and machine learning techniques have been used to track and categorize players in, among others, Bioware (DeRosa, 2007), Forza Motorsport (Romero, 2008), and World of Warcraft (Duchenaut et al., 2006) by leveraging raw game data. For these quantitative approaches, scholars developed and validated aggregate metrics from the raw data to measure game states or player behavior (Seif El-Nasr, Drachen, and Canossa, 2013), such as the number of attempts or level difficulty.

Using such metrics to describe player behavior has also been applied to educational games. For example, Serrano-Lugano et al. (Serrano-Laguna et al., 2014) introduce a scalable two-step approach where they first define simple generic metrics that could be applied to any educational game and then build game-specific assessment rules based on combinations of these metrics. They found that their approach is valid for identifying where players get confused, but it does not give insight into how to address player confusion, likely due to a lack of understanding of why players behave in a certain way (Hazan, 2013).

In addition, to better understand player decisions during gameplay and how player choices differed from what designers expected, a playtrace of an educational game has been done using conceptual feature extraction methods drawn from log data (Harpstead and Aleven, 2015). Researchers created various models that break gameplay down into individual cognitive tasks (or knowledge components) and assess which of these models is the best descriptor of player learning throughout the game. Though this work aims to gain insights into the dynamics of learning in an educational game, it relies solely upon logged game data and performing statistical regressions to better understand predictor variables.

To decode how players play games, tools have been developed to visualize player trajectories. In fact, two existing efforts, Playtracer (Andersen et al., 2010; Liu et al., 2011) and Glyph (Nguyen, Seif El-Nasr, and Canossa, 2015), have both supported the qualitative and quantitative analysis of an educational puzzle game through state
graphs to identify common play patterns. Playtracer calculates the distance of states to the goal, whereas Glyph uses defined player actions as the edges connecting the game states. Even with these tools, it is a challenge to decide how to define game states, their relationships to one another, and to interpret the data being visualized.

Historically, playtrace analyses have developed from small-scale qualitative studies to large-scale quantitative studies. The latter was enabled with the arrival of game analytics and the need to study larger data sets; however, this came at the cost of meaning, which has been the strength of traditional playtrace studies. In our work, we pursued a mixed-methods approach to address the issue of interpretation in analyzing player strategies in GrACE while maintaining as many benefits as possible from both quantitative and qualitative approaches.

2.4 Creating and Analyzing Difficulty Progressions

A traditional approach to assessing difficulty progressions in games is through player-oriented testing, a method of developing games that focuses on iteratively improving them based on player-feedback (Isbister and Schaffer, 2008; Kim et al., 2008). Often, developers begin by implementing a prototype of a game, and then have many people play it to evaluate how the game is likely to be experienced. The designer then considers whether the goals of the game are being realized through play, and redesigns aspects of it based on data gathered from the iterative design and testing procedures. While this process helps fine-tune the difficulty progression and provides ultimate control to the designer, it requires significant resources from the designer and the test-players.

One way to alleviate the demand for time from the designer is by first determining what exactly the player is expected to learn through play, and then automatically analyzing the effectiveness of different progressions. Often these follow a model similar to Elaboration Theory by Reigeluth et al. (1980) that argue the simplest form of a task should be introduced first, then more complex tasks which build on the first. This concept has been encapsulated by Cook’s (2007) skill atoms game design theory, which provides a framework for describing the challenges and skills that are being mastered in the game over time. In formalizing skill atoms, Cook describes a model comprising of action (taken by the player), simulation (by the game), feedback (from the game to the player as a result of simulation) and modeling (updating user’s mental model as a consequence of feedback received). Deterding (2013) uses this model as an approach to define player’s challenges as they pursue their goals in a game. This approach provided a method called the “lens of skill atoms”, which allows the perception of any interactivity from the point of view of game design by concentrating on the users’ goal (Deterding, 2012). A problem with this approach is that the model typically puts analysis of the game systems at the forefront, rather than taking a more player-centric approach. Therefore, this approach helps to analyze (and design) the underlying game system but does not focus on how players interact with that system. Our StrataBots, in contrast, are an idealized interpretation of how players of various skill levels engage with the game.

In terms of evaluating how players interact with systems, Linehan et al. (2014) qualitatively analyzed the order in which skills are introduced in four existing puzzle games. They extracted the order and method of novel skill introduction in each game and found that solution length increased until a new skill is introduced; at which point the levels return to short traces and build back up to longer traces using the newly acquired skill. Each main skill is introduced separately through simple puzzles.
that require only the basic performance of the new skill. After a skill is introduced, the player is given ample opportunity to practice the skill and combine it with other previously learned skills. Puzzles that use the player’s existing skills increase in complexity until a new skill is introduced.

Harpstead and Alevan (2015) attempt to quantify player strategy evaluation by applying skill atoms to the empirical analysis of difficulty progressions in order to predict player success in levels. This research used fundamentals of intelligent tutoring systems to analyze how well hypothesized models of required skills fit collected player data. Each predictive model was trained on player data and built on varying numbers of knowledge components or procedural skills. Overall this method reasonably predicts player success. Levels with higher error rates were easier for players and could be solved using a rote strategy (not one of their original models). However, this method requires extensive player data and is not conducive to procedurally generated levels as there will not be enough player data for each level. In addition, while the method tries to predict success and is useful for evaluating existing difficulty progressions, it does not indicate which strategies are usable on each level.

Not only is it useful to analyze existing progressions, but it is convenient during the design process to have tools that help craft the introduction of new skills. Andersen, Gulwani, and Popovic (2013) run an algorithm (e.g., addition) and log the order different steps are executed producing a solution trace with the skills necessary to complete a problem. The introduction of new steps can be done in a smooth fashion and situations where this does not occur can be highlighted. Algorithms presented in Chapter 6 focus on strategies rather than individual game mechanics which allows us to analyze level progressions after players master each mechanic. Also, the algorithms used by Andersen, Gulwani, and Popovic (2013) are deterministic so do not have situations where the algorithm must make a choice between two equally good options and follow each to completion. This is an important distinction because two solutions to a problem may have very different traces and the comparison of difficulty between multiple puzzles with this property becomes ambiguous. It is unclear if puzzles should be compared using the simplest solution, the most complicated, or some sort of hybrid method.

Similarly, Butler et al. (2015) created a mixed-initiative tool that automatically ensures a difficulty progression is followed by verifying a particular level feature is used or unused in the solution and that the introduction of new features is done at an appropriate pace. This tool was produced for a puzzle game called Refraction where players learn mathematical fractions by splitting a laser beam. Level features include using laser bending or splitting pieces, leaving some laser beams unused, and laser beams being unavoidably crossed. Our approach is similar but rather than looking at particular solution features and whether or not they are required to find a solution, we take a strategy-centric approach to determine available strategies that can be successfully applied to find the correct solution of a puzzle. In further research, Butler and Banerjee (2014) turn to visualizing progressions to help analyze and evaluate them. They show “ideal” progressions produced by humans or a progression generation tool. We adopt this approach to create user-specific progressions that help us understand how players are affected by various orderings of puzzles with different difficulty properties.
2.4.1 Human Computation Games

Across domains including computer science (Sarkar et al., 2017; Dukes, 2013), biology (Barone et al., 2015; Lee et al., 2014), medicine (University of Oxford, 2014; Coburn, 2014), astronomy (Lintott et al., 2008) and psychology (Computing, 2015) to name a few, HCGs give players tools and mechanisms to perform gamified, real-world, domain-specific tasks that computers cannot computationally solve due to complexity or lack of data. Prevalent tasks in HCGs include data classification (e.g. image labeling or sentence transcription) and common sense activities such as identifying color differences (Computing, 2015) or image labeling (Ahn and Dabbish, 2004).

Due to the complexity of tasks and inability to computationally model solutions, HCG designers often don’t know the skills their game must teach or the appropriate order in which to teach them, resulting in poor player retention—perhaps due to poor learning progressions or insufficient tutorials (Andersen et al., 2012; Sarkar et al., 2017)—suggesting most players do not acquire the full suite of skills game designers intended (Sauermann and Franzoni, 2015). Without these skills, players are unable to meaningfully contribute to the scientific research contained within an HCG, limiting the power of that game. In this thesis, we evaluate difficulty progressions in two HCGs through the application of skill-based AIs.

2.5 Procedural Content Generation

Some of the first known instances of PCG are in Rogue-like games such as Beneath Apple Manor (Worth, 1978) and, of course, Rogue (Toy et al., 1980). These early examples used PCG to create the game layout by defining monsters, treasures, rooms, and hallways allowing for a complex environment with no need for a designer to develop individual levels. Similarly, before graphically oriented games became mainstream and text-based games were state-of-the-art, Maze Craze (Atari, 1978) used an algorithm to generate a maze that constituted one game level. The random maze generation process allowed each play through to be unique. This was necessary due to the relatively short duration each maze took to complete and the ability for players to memorize puzzles.

As players and designers wanted more intricate environments, storage space was often a limiting factor (Amato, 2017). PCG allowed for the runtime generation of content to save storage space. The Sentinel (Crammond, 1986) contained roughly 10,000 different levels in just 64KB of storage. This was required since that’s all the BBC Micro, the system it was designed for, had at the time (Martínez Vilar, 2015). Elite (Braben and Bell, 1984) originally contained more than 250 trillion galaxies with 256 solar systems in each; however, reservations by the game’s publisher forced the designers to choose just eight of those galaxies for inclusion (Spufford, 2003). In addition to generating entire worlds and levels, PCG has been used to create textures, terrain, characters, quests, and simulate particle systems like smoke, fire, and water (Müller, Charypar, and Gross, 2003; James, 2001; Ebert, 2003). PCG has now become ubiquitous and is included in major titles from large corporations including top-sellers Diablo (North, 1996), Sid Meier’s Civilization VI (Games, 2016), Far Cry 5 (Montreal, 2018), and The Elder Scrolls II: Daggerfall (Softworks, 1996), among countless others.

Early examples of PCG in games and graphics took a constructivist approach—generating content from an initial set of parameters without analyzing or checking the

---

1Portions of this section previously appeared in a FDG 2014 conference paper evaluating content generators for 2D platformers (Horn et al., 2014)
content during the process—with techniques including grammars, fractal methods such as binary partitioning and quadtrees, and cellular automata (Shaker et al., 2016). More recent PCG techniques include a broad category known as search-based PCG (Togelius et al., 2011). In search-based PCG, content is created through a generate-and-test method where content is evaluated after generation and accepted or rejected based on some criteria. Notable techniques include: genetic algorithms, exhaustive search and simulated annealing.

Until search-based techniques became prevalent, few quantitative methods of analyzing PCG content, if any, existed and instead relied on human analysis of small generated samples. For example, Björk and Holopainen (2005) catalog recurring patterns that can be found across many games, while others have examined level design-specific patterns in domains such as first-person shooters (Hullett and Whitehead, 2010), 2D platforming games (Smith, Whitehead, and Mateas, 2011; Dahlskog and Togelius, 2012), and role-playing games (McNaughton et al., 2004; Smith et al., 2011c). While some of these patterns are largely intended as qualitative descriptions of game properties, others take the view that design patterns can be solutions to specific design problems. Most of the aforementioned work in understanding game and level design is based in qualitative analysis and theories.

Recently, there has been some work in trying to automatically and quantitatively measure aspects of game quality. Within search-based procedural content generation there is a need for evaluation functions, and for this reason several researchers have tried to quantitatively capture what they deem to be crucial aspects of game quality. This includes Browne’s various metrics for board games, such as drawishness, length, drama and outcome uncertainty (Browne and Maire, 2010), and Togelius and Schmidhuber’s learning-based metric (Togelius and Schmidhuber, 2008). There has also been some work on trying to measure the quality of platform game metrics with Smith and Whitehead defining two key metrics—linearity and leniency—as well as introducing a method for visualizing the expressive range of a generator (Smith et al., 2011b). Shaker et al. followed up this work by introducing further metrics, some of them based on theories of player experience and others based on data mining (Shaker et al., 2012).

While the breadth of evaluative techniques in PCG has been increasing for years, player-centric analysis methods remain few and far between. Where they exist, as in Togelius et al. (2007) and Shaker et al.’s (2012) player experience metrics, large amounts of player data are required for learning controllers or the AIs that assess content lack nuance by typically having one “optimal” player. Not until persona-based player modeling (Holmgård et al., 2015) was there a formalized method for evaluating content from the perspective of multiple (human-like) types of players without needing vast quantities of game data from which to learn. However, personas are differentiated not by their ability to play the game, but rather their goals while playing. We are still left with no quantitative, data sparse, skill-based technique to evaluate game content, especially content that is procedurally generated.

More broadly than PCG for games, there is some work in evaluating computationally creative systems; Jordanous provides a survey of current evaluation methods (Jordanous, 2011). It is important to note that these evaluation criteria are being used to answer a different, though related, question: computational creativity evaluation asks the extent to which a system is creative, while PCG evaluation asks how expressive and controllable the system is. For example, Pease et al. incorporate an evaluation of the process that the generative system follows as well as rating the product produced by the system (Pease, Winterstein, and Colton, 2001). Similar evaluations have been performed on platform game level generators (Cook, Colton,
and Pease, 2012). While we recognize the importance of process in understanding creativity, and feel that such discussions would be of great value to PCG researchers, it lies outside the scope of this dissertation.
Chapter 3

Educational Game Design

We begin by creating a computer science (CS) educational game called GrACE which allows us to toggle the presentation of procedurally generated puzzles to players in order to answer our first research question of Section 1.4—*How does the introduction of procedural content generation in an educational game affect player behavior and performance?* Currently, there is no game available that allows the study of educational material in both a PCG and non-PCG environment so we are forced to develop one in order to adequately assess the effect of PCG content on players. We find that PCG allows for more material to be seen and practiced by players resulting in improved knowledge acquisition when playing alone.

Additionally, we develop a series of assessment materials to determine the extent to which players learn during gameplay and evaluate them across two studies—one with students from the USA and one with Nigerian students. We evaluate the broader appeal of the current design of GrACE by analyzing player performance among two groups of Nigerian students—one from a free-of-charge summer camp aiming to broaden technology education to disadvantaged groups, the other an all-girls, STEM-focused boarding school of high-performing students. From this additional study, we identify future design considerations for both GrACE and the assessment materials.

In Chapters 4 and 6, we analyze the level progression of GrACE to determine drawbacks of our approach. A major finding is that traditional metrics used in PCG research fail to adequately explain the difference in player performance on some levels and so we develop a skill-based approach to better predict player performance.

### 3.1 Computer Science Educational Games

The computer science education community has spent much of its effort on introducing CS education earlier in the curriculum, in hopes to better prepare students for a world where procedural literacy and computational thinking is increasingly important (Guzdial, 2008; Mateas, 2008) as well as improve chronic issues of a lack of diversity in STEM fields, including CS at the university level and beyond (United States Department of Commerce, 2011). Many of these efforts to reach younger students are done in informal contexts such as classroom visits, summer camps, and after-school programs (Denner and Campe, 2008; Doerschuk, Liu, and Mann, 2007; Frieze, 2005). The ACM has also recently proposed a set of computing education guidelines that can be integrated into a standard K-12 curriculum (Tucker et al., 2006), and in the UK there are already efforts in place to teach computational thinking in primary and secondary schools (Computing, 2015).

---

1The original GrACE study as well as the Nigerian study that follows were previously published as conference papers at SIGCSE 2016 and GLS 2016, respectively (Horn et al., 2016a; Folajimi et al., 2016).
Games have been a promising and popular component of many efforts to teach CS and broaden participation. At the middle school level, there are programs that encourage students to create games using simplified programming environments such as Alice (Cooper, Dann, and Pausch, 2000) or Scratch (Resnick et al., 2009). Others are educational games specifically designed to teach CS. However, the vast majority of these game-related outreach efforts tend to focus on the craft and practice of programming, rather than higher-level CS concepts (Harteveld et al., 2014).

With this in mind, we have developed GrACE, a puzzle game that aims to teach computational thinking—specifically, concepts of abstraction and algorithms (Wing, 2006). The game teaches neither the structure nor syntax of programming. Instead, it is designed such that players must solve CS puzzles in an algorithmic manner, thus fostering computational thinking.

Puzzles in GrACE are built around the common CS problem of finding a graph’s minimum spanning tree (MST). This problem was chosen due to the relative ease of mapping graphs onto spatial puzzles, the existence of many algorithms to solve the problem, and its property that even finding an incorrect solution (such as finding a spanning tree that is not minimal) still can involve computational thinking. Thompson and Bell also find the MST problem useful for mapping to a CS educational game due to the need for players to identify algorithms (Thompson and Bell, 2015).

With this game, we explore the use of procedural content generation (PCG) to aid in computational thinking. Computer generated puzzles bring advantages including rapid puzzle creation and a guarantee all puzzles meet desired graph structure and aesthetic design constraints. Additionally, the incorporation of PCG means players can request new puzzles on demand increasing the amount of in-game content available to them to practice different strategies (Smith, 2014). We conjecture that by seeing a larger variety of content, players receive a better understanding of the underlying concept (Smith and Harteveld, 2013). We further believe that seeing content variety in a collaborative setting will push discussions among players toward abstracted solutions.

Our long-term goals are to come up with an appropriate puzzle design to foster computational thinking in middle school students and to study the educational benefit of including PCG in educational games. In this chapter, we present the evaluation instruments and results of an experimental pilot study of the current design of GrACE as well as an evaluation in two contexts with Nigerian students. Our contribution is a set of design insights into how to construct and evaluate educational games that do not aim to specifically teach programming, but rather focus on high-level concepts.

3.2 Related CS Educational Games

Many game initiatives for teaching CS exist. Existing CS games frequently focus on teaching programming, either using actual code or a simulated programming-like set of tasks (Harteveld et al., 2014). Those that do support multiplayer activities are typically competitive rather than collaborative, and rhetorically the games often position each player as a solitary character working alone towards a goal. In contrast, GrACE casts the player as a helpful character who needs to use their reasoning skills to help a friend, encouraging collaboration even though it is a single-player game.

The focus on competition in educational CS games is seemingly at odds with the value of collaboration in other aspects of CS education. Pair programming (Williams and Kessler, 2000) is a common practice in introductory programming courses, with
benefits to student learning arising from the ability to share strategies and point out potential errors during collaboration. Research in educational games in other fields, such as mathematics, shows competition can be more effective for achieving learning outcomes, but collaboration is still effective and can lead to greater student interest (Ke and Grabowski, 2007; Plass et al., 2013).

Most educational games use content that was scripted by the human authors of the game. Procedural content generation (PCG) is the practice of “creating game content automatically, through algorithmic means” (Togelius et al., 2011). Refraction (Smith et al., 2012) is an example of a math education game that has procedurally generated puzzles. It uses PCG to create its puzzles both to ease authoring effort (by having the computer take on this design role) as well as make hard guarantees about the properties of levels from both an educational and aesthetic perspective. However, Refraction simply replaces a typical human-designed level with a computer-designed level; it does not take advantage of the existence of an on-demand puzzle designer during gameplay. GrACE uses similar technology to Refraction to generate puzzles, but also allows players to request new puzzles at the same difficulty level on demand throughout the game.

Our work examines differences in performance based on whether players had exposure to many different levels, as well as the impact on the use of PCG when players collaborate but are seeing different puzzles from each other. It has been argued that PCG may be useful for educational games through allowing players to explore alternate problems, and communicate abstracted strategies with each other (Smith and Harteveld, 2013).

### 3.3 Existing Cross-Cultural Game Analysis

Although much emphasis has been put on broadening CS to girls and underrepresented groups specifically through games (Kafai et al., 2008), initiatives have not been pursued to broaden CS cross-culturally, that is, to make a CS teaching game that would be appealing to two or more different cultures or countries. This chapter initiates this type of broadening in addition to the ones frequently advocated for, with a focus on broadening a game from a developed country towards a developing country. The entire world would benefit from a population that is more computer literate and building a completely new game tailored to each culture is not efficient. Additionally, in the case of developing countries, it allows for accessibility to technologies that may help them in their development and prevent a larger gap between nations from occurring.

Because games are designed to tell a specific story or describe specific situations with substantial cultural artifacts and direct relationship between the game storyline and its audience, most educational games are designed and evaluated for certain groups of people with similar culture and interests, or within the same geopolitical areas. The implication of this is that players will not easily understand nor appreciate the design for a game that is contrary to their attitudes, customs, and beliefs, and this may happen within the same country because cultures are not necessarily bound by borders. This lack of cross-cultural usage is representative for the field of educational games in general and not specific to CS teaching games alone. To be validated as being truly cross-cultural, a game should be designed and evaluated among groups of people across varying cultures.

Existing efforts thus far have been sparse. Kam et al. (2007) evaluated the efficacy of eight educational games among rural children in India to identify the role of
contextual factors, and later research (Kam et al., 2009) conducted an exploratory study to inform the design of a new videogame that rural children in India found to be more intuitive and engaging. Although this work specifically considered culture in designing an educational game, the work still focused on a specific group and, therefore, is not cross-cultural. By studying electronic gaming machines and gambling disorder, Medeiros et al. (2015) did a cross-cultural comparison between treatment-seeking subjects from Brazil and the United States but their aim was not to evaluate an educational game for intercultural impact. In terms of truly cross-cultural efforts on educational games, Khaled et al. (2006) developed two versions of a persuasive game to educate citizens about smoking cessation: one for New Zealand Europeans and one for the Maori. They concluded it is best to design with the cultural background of the intended audience in mind. This conclusion is aligned with the findings of Folajimi, Istance, and Rolfe (2012). After implementing a game for educating children about sickle cell disease and implemented with UK and Nigerian children, it was concluded that educational games need to be built with a view to varying cultural backgrounds. Other efforts have involved using existing theories about cultures as input to the design and implementation of games, such as Hofstede’s cultural dimensions (Hofstede and Pedersen, 1999). Additionally, for entertainment games specifically there have been various practical examples as well as documented ones (Fernández Costales, 2014; O’Hagan, 2009) about the need for game localization, which mostly involves cosmetic adjustments to a game (e.g., changing from red to green blood for games distributed in Germany).

In this chapter, we contribute to the design of cross-culturally relevant educational games, and examine this in the particular context of CS education. Acknowledging that varying designs may be necessary based on the work in this area thus far, our efforts focus on first establishing the differences between two clearly different cultures in order to make design recommendations for future work. For this effort, we investigate the cross-cultural use and impact of GrACE. The choice of Nigeria for our case study was majorly because of convenience due to opportunities provided to the researchers, but we see it as a step towards expanding the reach of the game to more developing and developed nations in the world. The results of this Nigerian implementation and the comparison with the US implementation are reported in this chapter.

3.4 Methods

Based on our ideas of the potential for PCG for education and the possible mediating role of collaboration, we came up with the following three exploratory propositions:

**Proposition 1.** Experiencing variety leads to increased learning gains in computational thinking compared to no variety.

**Proposition 2.** Working in a collaborative context leads to increased learning gains in computational thinking than in an individual context.

**Proposition 3.** Experiencing variety in a collaborative context leads to increased learning gains in computational thinking compared to no variety in an individual context.

To explore these propositions and, more broadly, our game and evaluation design, we implemented an independent 2x2 factorial design as part of a pilot studies. The two 2-level independent variables (IVs) are PCG (PCG vs. No-PCG) and Collaboration (Individual vs. Collaborative).
3.4.1 Participants

Three separate groups of students were selected for the experiments. The initial pilot was implemented as part of a two-week summer program at Northeastern University. This program selects 48 middle school talented students each year, and focuses on STEM content. The program historically supports underserved and underrepresented students with limited opportunities and is free of charge.

During the day of implementation, 43 students participated with the consent of their parents, 22 identifying as female and 21 as male. Ages ranged from 10 to 13 ($M = 11.9$, $SD = 0.85$). Four participants identified as Hispanic or Latino, 12 reported to be Asian, nine Black or African American, 12 White, and six as “other” (with four participants preferring not to answer).

We applied the same setup from the USA pilot study to two Nigerian groups and made modifications where necessary in the procedure and material. For example, we modified terms such that it would be comprehensible and familiar to Nigerian students (e.g., changing Mayor to Governor).

3.4.2 Nigerian Case Study

The pilot study was replicated in two contexts: the Nigeria Geek Girls Collaborative Camp (Summer Camp) and the Ogunsanya Girls Science Academy (Academy). The first context is a computer training and mentoring camp for Nigerian secondary school girls (ages 10-16 years) with the aim of enhancing the talent and skills needed to fuel technological and economic growth. To reach out to the best talents, notwithstanding their social or economic background, the camp is free of charge. The 2015 camp consisted of 40 participants, with 36 opting to participate in the pilot, including four male non-campers who indicated interest in evaluating the game. Of these 36, one student was excluded for neither having survey nor game data. Parental consent and school approval were received from each participant.

The second Nigerian context is an all-girls secondary school with a special focus on preparing girls for STEM careers. Of the 30 students that participated from the Science Academy, two students were excluded for lack of survey and game data thereby leaving us with 28 participants. By default, all participants from this group are female boarding house students of the same school and study science-related subjects. Computer Science is one of the major subjects taught in the school and almost all the students have personal computers.

3.4.3 Materials

All materials mentioned in this section are publicly archived. The USA pilot and Nigerian pilot studies will be reported separately.

GrACE

GrACE has a nature-based theme, chosen as a result of focus testing with middle school students to determine the appeal of a variety of metaphors. The game is centered on two characters—a mouse and a rabbit—who are collecting vegetables. Vegetables (nodes) in the ground are connected by cracks (edges) that only the mouse can fit through. Edge weight corresponds to the amount of bunny energy needed to dig along a crack. Players control the mouse as it explores the map and flags cracks

2http://hdl.handle.net/2047/D20199448
for the rabbit. To minimize the bunny’s digging energy, players must find and flag the MST. An example can be seen in Fig. 3.1

Initially, players can only see the starting node and the nodes connected to it. The player chooses an edge to traverse at which point the next node and its connections are revealed. This mechanic discourages players from solving the puzzle visually by examining the entire graph at once. Instead, we designed the game around limited information exploration to encourage stepwise thinking and mimicry of MST solving algorithms.

We used two game versions in accordance with the PCG manipulation: one version with PCG and one without. The standard version contains 11 difficulty levels, each with a single associated puzzle pre-generated by the computer and the same across all instances of the game. These puzzles were each chosen at random from the set of levels used in the PCG version, to limit accidental biasing from including human-authored puzzles. Players can restart their puzzle and access previously played puzzles once completed. In addition, the PCG version allows for the possibility at anytime to press a “random” button to generate a new puzzle with the same difficulty level. We generated 100 puzzles for each level in advance, making the PCG version contain 1,100 puzzles in total. Level one contained two nodes and one edge to demonstrate basic game mechanics. Level 11 contained nine nodes and 16 edges.

Maps were created using a constraint-based, Answer Set Programming (ASP) method (Smith and Mateas, 2011). Graph constraints included the number of nodes and edges, minimum and maximum weights for edges, node connectivity, and a valid MST. Aesthetic constraints were the map size, distance between nodes, valid edge intersection angles, and various art tile layout rules. Constraints were separated

---

**Figure 3.1:** Screenshot of a GrACE level where the player has fully explored each node and has returned the mouse to the starting node indicated by the bunny. The player has marked the minimum spanning tree by placing three flags on the correct edges.
so that future art implementations could have their own set of constraints swapped with the current set and not impact the core map generation.

The game was instrumented to track all player actions (e.g., placing or removing a flag, submitting an answer) and game states (e.g., mouse and bunny energy). We used USB sticks to retrieve this data from the personal and lab computers used in the studies. Unfortunately, a significant number of Nigerian files got corrupted, leaving us with eight complete data sets for the Summer Camp and 21 for the Science Academy implementation.

**Questionnaires**

The pre-questionnaire captured game attitudes and the post-questionnaire demographics and game experience. For game attitudes, we aimed to measure the constructs of *Liking* (3 items; $\alpha_{US} = .76$, $\alpha_{NIG} = .54$) and *Leisure* (4 items; $\alpha_{US} = .85$, $\alpha_{NIG} = .75$). *Liking* measures how much the students like playing video games; *Leisure* measures the degree to which games are incorporated in the leisure time of participants. Regarding their game experience, we aimed to measure how much participants liked playing with the construct *Enjoyment* (4 items; $\alpha_{US} = .93$, $\alpha_{NIG} = .69$) and had trouble with playing with the construct *Difficulty* (3 items; $\alpha_{US} = .87$, $\alpha_{NIG} = .32$). For both game attitude and game experience, we used 7-point Likert items adapted from the New Computer Game Attitude Scale (NCGAS) (Reece and Gable, 1982) and the Game Experience Questionnaire (GEQ) (IJsselsteijn et al., 2007), respectively. Based on factor analyses (principal axis factoring, varimax) and a comparison of similarity indices (after procrustean rotated loadings), and specifically Tucker’s coefficient of congruence (Lorenzo-Seva and Ten Berge, 2006), we concluded that our instruments have been applied to a different population reasonably well but need to be viewed with caution due to the low internal consistency with the Nigerian sample. The *Difficulty* construct seems not to generalize and for this reason we exclude it from our analysis.

**Comprehension Test**

The comprehension test was developed to measure conceptual understanding of the MST. Previous studies on computational thinking have included tests that made effective use of measuring change in students’ conceptual understanding through design scenarios (Brennan and Resnick, 2012) or concept exams (Munson et al., 2011) specific to the software being tested. To allow for objective comparison across students, we provided a multiple choice assessment along the lines of the concept exams. In contrast to previous studies, we opted for questions not only about solving MST problems in the context of the specific software but also about solving MST problems with another metaphor and in an abstract manner. The goal with this is to measure knowledge transfer from the game to other contexts.

We created two tests, each with a different but similar puzzle for all three contexts: game, metaphor, and abstract. Each puzzle has three associated questions with four choice options for each question. Consistently across the puzzles, one question concerned the problem of adding a single edge, one about removing a single edge, and one about correcting an incorrect spanning tree. The questions were developed independently, validated by the three test question authors, and error-checked by a fourth researcher on the team. As during the Nigerian Camp implementation about half of the participants did not receive the house subtraction question, we excluded
this question from our analysis. We counterbalanced the test distribution similarly to the US Pilot and did not find a significant difference between the two versions.

**Game.** For these questions, we edited screenshots of random puzzles from the PCG version and constructed questions using similar language to that used in the game.

**Metaphor.** Here we adapted the “Muddy City” exercise from CS Unplugged (Bell, Witten, and Fellows, 2010). In our adaptation, we simplified the language of this roads network problem because the students had to read it to themselves, and redid the artwork to make it clearer and easier to modify.

**Abstract.** These puzzles were based on a typical graph with nodes and edges that we referred to as circles and lines. This representation is the common format for Computer Scientists.

### 3.4.4 USA Procedure

The USA study was run as part of a “computational thinking” activity in the summer program. We randomly assigned students to a computer, and evenly distributed them across the conditions; however, due to a slightly lower turnout and one student opt-out, we had fewer students in both PCG conditions. We counterbalanced the comprehension test to rule out the difficulty of the test as a bias.

The experiment started in a single classroom where students received their pre-questionnaire, pre-test and lab assignment. Instructions for the game were then provided by a facilitator without mentioning the MST concept or the game’s relationship to CS. One lab held all students in the individual condition, the other had students in the collaborative condition. We divided both spaces such that one half was assigned to the PCG condition and the other half to the No-PCG condition to minimize interference. Assigned pairs played next to each other and were encouraged to communicate and look at each other’s screens. In the individual condition, students were discouraged from interacting. All facilitators followed a script instructing them how to respond during the questionnaires, tests, and gameplay.

Students completed the post-questionnaire and post-test after the game, followed by a semi-structured discussion on the topic of CS and the MST concept in particular. The experiment took three hours, with one hour of gameplay.

### 3.4.5 Nigerian Procedure

At the Nigerian Summer Camp, the study was implemented as part of an educational games activity; for the Science Academy implementation the team was able to organize an informal, extracurricular activity on a Saturday. In both cases, the experiment followed the same pattern as the USA implementation. For the Summer Camp, the study took approximately 3.5 hours, with one hour of gameplay, while in the case of the Science Academy, the experiment took about 2.5 hours to complete.

### 3.5 USA Results

For the purposes of this section, we limit ourselves to reporting the main results that we derived on demographics, gameplay, and the performance on the comprehension test for the USA group.
3.5.1 Demographics

The majority (61.9%) reported playing games more than 1-2 days a week. There were no strong preferences or dislikes of genres. Based on the preferences for school subjects, with 32 (74.4%) indicating a liking for math and 33 (76.6%) a liking for science, it is clear that this is a group with a bias towards STEM content. In terms of gender differences, boys reported to play more than girls, $\chi^2(1) = 4.39, p = .036, V = .371$. Boys further reported to like playing adventure and action games more so than girls, for both $\chi^2(1) = 10.5, p = .001, V = .495$. For school subjects, girls liked art and language and literature better, respectively $\chi^2(1) = 7.21, p = .001, V = .409$ and $\chi^2(1) = 6.89, p = .007, V = .400$. Boys, on the other hand, report liking technology better, $\chi^2(1) = 4.06, p = .044, V = .309$. No racial differences were observed in game and school subject preferences.

The results regarding game attitude suggest that the majority like games but are more spread in their opinion when it comes to the role of games in their leisure time. The variety in Leisure is partially a result of a difference in gender, $t(40) = -5.67, p < .001, r = .73$, with girls disagreeing more about its importance. This finding is consistent with the gender difference in frequency of playing.

3.5.2 Gameplay

Investigating the items associated with Enjoyment, it becomes clear that a strong majority agreed or strongly agreed they had fun and majorities further indicated they would recommend this to a friend, play it at home, and learned from it. Despite the enjoyment, in exploring the items associated with Difficulty, it shows that majorities thought it was frustrating, challenging, and hard. In contrast, the majority disagreed that they felt bored. No differences are found on gender and race, except for the Enjoyment item “I thought it was fun”. It turns out girls ($Mdn = 5, IQR = 4-6$) agreed with this less than boys ($Mdn = 6.5, IQR = 5-7$), $U = 144, p = .047, r = .31$.

Both Liking and Leisure correlate with Enjoyment, respectively $r = .40, p = .009$ and $r = .309, p = .049$, which further suggests an initial predisposition is of influence in how the game is experienced. It is then not surprising to find that students who play more than 1-2 days a week enjoyed the game more than those who play less, $t(39) = 2.91, p = .006, r = .42$. Liking, leisure, and frequency of play do not relate to Difficulty.

There are aspects that do relate to Difficulty. Those who like puzzle and strategy games perceived less difficulty than those who do not, respectively $t(41) = 2.31, p = .026, r = .34$ and $t(40) = 2.73, p = .009, r = .40$. Interestingly, the same is true for students with a preference for sports games and physical education, respectively $t(21) = 2.54, p = .019, r = .48$ and $t(41) = 2.11, p = .041, r = .31$. This may suggest that these students like a challenge and have a different perception of difficulty. Surprisingly, it is students with an interest in simulation and role-playing games that scored higher on Enjoyment, respectively $t(40) = 2.41, p = .021, r = .36$ and $t(40) = 2.22, p = .032, r = .33$.

Regarding the conditions, it appears students in the PCG conditions agreed more with the Difficulty item “I felt frustrated”, $t(41) = 2.47, p = .018, r = .36$. For the collaborative conditions it is interesting to note that it approaches significance for agreeing more on the Enjoyment item “I would recommend this game to a friend”($p = .061$) and on the Difficulty item “I thought it was hard”.

The game data revealed that students in the collaboration condition took longer to play the game with an average level time of 111 seconds ($SD = 33.3$) compared to
Table 3.1: Comprehension test improvement across conditions, in M (SD). The test is separated into three representations (original game aesthetic, road network metaphor and abstract) and three concepts (edge addition, edge deletion and completed tree correction). Conditions are broken down by Individual (Ind.) vs Collaboration (Coll.) and PCG vs No-PCG.

<table>
<thead>
<tr>
<th>Test Measure</th>
<th>Ind. x No-PCG (N=12)</th>
<th>Coll. x No-PCG (N=12)</th>
<th>Ind. x PCG (N=8)</th>
<th>Coll. x PCG (N=10)</th>
<th>Overall (N=42)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>1.08 (1.44)</td>
<td>0.67 (1.30)</td>
<td>1.13 (1.73)</td>
<td>-0.30 (2.31)</td>
<td>0.64 (1.73)</td>
</tr>
<tr>
<td>Game</td>
<td>0.50 (1.09)</td>
<td>0.17 (0.94)</td>
<td>0.63 (1.51)</td>
<td>0.30 (1.49)</td>
<td>0.38 (1.21)</td>
</tr>
<tr>
<td>Roads</td>
<td>0.50 (0.52)</td>
<td>0.33 (0.78)</td>
<td>-0.25 (1.04)</td>
<td>-0.30 (1.33)</td>
<td>0.12 (0.97)</td>
</tr>
<tr>
<td>Abstract</td>
<td>0.08 (1.00)</td>
<td>0.17 (1.03)</td>
<td>0.75 (1.03)</td>
<td>-0.30 (0.95)</td>
<td>0.14 (1.03)</td>
</tr>
<tr>
<td>Addition</td>
<td>0.17 (0.86)</td>
<td>-0.08 (0.90)</td>
<td>0.50 (0.76)</td>
<td>0.00 (0.94)</td>
<td>0.12 (0.86)</td>
</tr>
<tr>
<td>Deletion</td>
<td>0.42 (0.90)</td>
<td>0.42 (0.90)</td>
<td>0.25 (1.16)</td>
<td>-0.10 (0.99)</td>
<td>0.26 (0.96)</td>
</tr>
<tr>
<td>Correction</td>
<td>0.50 (0.80)</td>
<td>0.33 (0.89)</td>
<td>0.38 (1.30)</td>
<td>-0.20 (0.92)</td>
<td>0.26 (0.96)</td>
</tr>
</tbody>
</table>

individual participants’ average of just 85 seconds (SD = 28.5), $t$(40) = 2.67, $p$ = .01, $r$ = .39. Somewhat related and nearing significance is that students in the individual condition (Mdn = 11, IQR = 9-11) were more likely to complete the final level than the collaboration condition (Mdn = 9, IQR = 7.5-10.5), $\chi^2(1) = 3.93, p = .056$. Every student was able to complete at least level 5 (4 nodes and 5 edges).

The “random” button was used a total of 125 times by the 19 students in the PCG group ($M = 6.58, SD = 6.74$). No statistically significant difference was seen between the collaboration and individual groups. The button was used most on Level 6 at 31 times (24.8%), followed by Level 8 at 25 (20%) and level seven at 23 (18.4%). The “random” was not used by seven students and—surprisingly—on Level 11.

3.5.3 Comprehension

Independent Samples T-tests show that there is no significant difference between the two versions on the total scores before and after the game, suggesting that the test versions are equal in difficulty. Test scores were calculated by providing one point per correctly answered question. Thus, with nine questions students could get a maximum score of nine points. Even though MST problem was likely unfamiliar to students, they performed quite well on the pre-test ($M_{pre} = 5.71, SD_{pre} = 2.13$) and marginally but significantly improved their performance on the post-test ($M_{pre} = 6.37, SD_{pre} = 2.33$), $t$(41) = 2.40, $p$ = .021, $r$ = .35. Of note is that the number of students with a perfect score increased from two (4.8%) to nine students (21%).

Table 3.1 provides an overview of the average improvement across conditions on the overall test and on the specific puzzles as well as kinds of questions. The table implies there are differences across the conditions. The collaborative conditions seem to improve less so than the individual ones, and this seems especially true for the collaborative condition with PCG, suggesting an interaction effect exists. No strong significant difference appears at first; however, in exploring the data a surprising finding was that students who liked the action genre ($M_{pre} = 5.61, SD_{pre} = 1.97$; $M_{post} = 6.88, SD_{post} = 1.92$) improved more than students who did not ($M_{pre} = 5.84, SD_{pre} = 2.36; M_{post} = 5.74, SD_{post} = 2.68$), $t$(40) = 2.73, $p$ = .009, $r$ = .40. None of the other demographics had such an influence. As boys liked the action genre more so than girls, we performed 2x2 ANCOVA analyses with PCG (PCG vs. No-PCG) and Collaboration (Ind. vs. Coll.) as between-subjects factors and with gender and action
as covariates. On the total improvement this revealed a main effect of action, $F(1, 36) = 10.1, p = .003, \eta_p^2 = .22$, and Collaboration, $F(1, 36) = 4.69, p = .037, \eta_p^2 = .12$. The effects of gender ($p = .20$) and PCG ($p = .67$) were insignificant; however, the interaction effect between PCG and Collaboration approximates significance, $F(1, 36) = 3.64, p = .064, \eta_p^2 = .09$. The interaction effect is significant on the abstract puzzles specifically, $F(1, 37) = 5.36, p = .026, \eta_p^2 = .13$. As exemplified in Table 3.1, we can see that students in the individual condition with PCG improved more so than in the No-PCG conditions, whereas in the collaborative condition with PCG the performance is less than in the No-PCG conditions.

### 3.6 Nigerian Results

In detailing our results, we focus on the main outcomes from the survey, comprehension test, and the game data in each Nigerian group. We make specific mention whenever we found a relevant difference compared to the USA pilot study.

#### 3.6.1 Survey

The majority of the Nigerian students (52.5%) reported to play more than 1-2 days a week (as opposed to 61.9% of US students). Interestingly, most (39.3%) reported to play games multiple times a day, which is more than what the US students reported (16.7%) and interesting for the fact that US boys play more than girls. Despite seemingly playing more games on a daily basis, the Nigerian students disliked the more typical game genres more so compared to the US students, with an almost entire dislike for strategy, role-playing, and simulation games, which are maybe games that this population does not have much exposure to. Except for the adventure genre, which was more favored by the Science Academy students (81.9% vs. 55.9%), there were no differences between the two Nigerian groups, suggesting that they were homogeneous in terms of what games they play and like. Similar to game genres, the Nigerian students liked fewer school subjects than the US students.

Regarding game attitudes, the majority of students indicated a liking of games, similar to the US students. When it came down to the degree to what games are part of their leisure time it becomes clear that for Nigerian girls games play overall a more important role than for US girls, $t(77) = −3.24, p = .002, r = .35$, but less so than compared to US boys, $t(75) = 2.68, p = .009, r = .29$. Interestingly, the statements where the Nigerian girls ranked similarly to the US girls, it is where the US boys scored higher, indicating more of a gender difference rather than a cross-cultural one. The US boys scored higher on thinking about games while not playing, considering games to be part of their life, and spending their free time on playing games. For the latter statement, the Nigerian girls did differ compared to the US girls but this is attributed to the Academy girls. They are spending more of their free time playing games compared to the Camp girls, $U = 281, p = .005, r = .36$. Overall, the Academy girls indicated a more important role for games in their leisure compared to the Camp girls, $t(50) = −2.82, p = .007, r = .31$.

As for how the participants experienced the game, it is clear that the Nigerians enjoyed the game, $t(77) = −2.11, p = .038, r = .23$. They would recommend this game to their friends, want to play it at home, and have learned from it more than the US participants. No differences were noticeable in terms of the difficulty of the game. However, among the two Nigerian groups, it is noticeable that Academy girls found
Chapter 3. Educational Game Design

### Table 3.2: Overview of comprehension test scores across the USA and Nigerian groups, in \(M(SD)\).

<table>
<thead>
<tr>
<th>Item (max)</th>
<th>US Pilot (N=43)</th>
<th>Nigerian Camp (N=35)</th>
<th>Nigerian Academy (N=28)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pre</td>
<td>Post</td>
<td>Pre</td>
</tr>
<tr>
<td>Game (3)</td>
<td>1.70(1.12)</td>
<td>2.11(1.05)</td>
<td>0.67(0.77)</td>
</tr>
<tr>
<td>House (2)</td>
<td>1.41(0.74)</td>
<td>1.06(0.90)</td>
<td>0.62(0.74)</td>
</tr>
<tr>
<td>Abstract (3)</td>
<td>1.98(0.90)</td>
<td>2.12(0.85)</td>
<td>1.06(0.89)</td>
</tr>
<tr>
<td>Addition (3)</td>
<td>2.14(0.78)</td>
<td>2.26(0.85)</td>
<td>1.03(0.76)</td>
</tr>
<tr>
<td>Subtraction (2)</td>
<td>1.09(0.76)</td>
<td>1.33(0.78)</td>
<td>0.44(0.56)</td>
</tr>
<tr>
<td>Correction (3)</td>
<td>1.88(1.13)</td>
<td>2.12(1.15)</td>
<td>0.67(0.82)</td>
</tr>
<tr>
<td>Total (8)</td>
<td>5.07(2.03)</td>
<td>5.67(2.16)</td>
<td>2.35(1.18)</td>
</tr>
</tbody>
</table>

it more challenging, \(U = 330, \ p = .030, r = .28\), and experienced more frustration, \(U = 331, \ p = .025, r = .28\).

With the US participants a predisposition was noticed in how the game is experienced. For the Nigerian sample, however, no differences were found regarding their preferences in game genres and school subjects. Unlike the US audience, both Liking and Leisure do not correlate to their Enjoyment as well as how frequently they play games.

#### 3.6.2 Comprehension

In order to calculate the performance on the comprehension test, we calculated in addition to the total scores the scores per puzzle (Game, House, and Abstract) and per type of question (Addition, Subtraction, and Correction). Table 1 provides an overview of the average scores per implementation on all these items. Immediately apparent is that large disparity between the US participants and the Nigerian participants. Where the US participants answered on average 60% of the test correct already, and moderately improved after playing, the Nigerian students answered 30% or less on average correct, and there is no noticeable improvement. In fact, for the Summer Camp participants a decrease is almost observable, \(t(33) = -1.87, \ p = .071, r = .31\). Specifically, they seemed to have trouble with the addition questions, \(t(33) = -2.54, \ p = .016, r = .40\). When looking at the maximum total scores, we see that with the US pilot a perfect score increased from four to 12 participants. With the Nigerians, on the other hand, three students received a score of five points at the start. After playing, one participant received a score of six points and none got five points.

The results may have been due to a possible floor effect. The test may have been too challenging, especially considering that the Nigerian students themselves expressed to have learned from it, and more so than the US participants. In exploring possible relationships with the test performance, we found that age may be a factor in this group, \(r = .29, \ p = .023\). In addition, and much surprisingly, it seems that player enjoyment is negatively correlated with how much they improved, \(r = -.27, \ p = .043\). Further exploration shows that in particular participants with positive improvements agreed less with the statement “I thought it was fun”. Based on this, we made a new dichotomous variable distinguishing participants who agreed and disagreed on this statement, and ignored those who chose neither disagree nor agree. We then compared this new variable separately for each Nigerian group because a decrease was almost observable in the Summer Camp participants. For the Summer Camp participants, the “disagreeers” (\(N = 7; M_{pre} = 1.43, SD_{pre} = 0.53\);
Mpost = 2.14, SDpost = 1.21) improved more so than the “agreers” (N = 22; Mpre = 2.73, SDpre = 1.24; Mpost = 1.63, SDpost = 1.40), t(27) = 2.71, p = .014, r = .45; likewise, for the Academy participants the “disagreers” (N = 9; Mpre = 0.78, SDpre = 0.67; Mpost = 2.33, SDpost = 1.41) improved more so than the “agreers” (N = 19; Mpre = 1.95, SDpre = 1.27; Mpost = 1.95, SDpost = 1.03), t(16) = 2.71, p = .016, r = .56. From these results it is noticeable that the disagreers scored lower on the pre-test than their peers. When only considering the disagreers, the game did have a significant impact, t(15) = 3.45, p = .004, r = .67.

3.6.3 Game Data

For the game data we considered what difficulty levels participants played (1 to 11), the time it took them to complete a level, how many times it took them to complete a level, how many actions they needed to take within a level to complete it, and—when playing the PCG version—when participants requested a new level. Regarding the latter, none of the 17 participants made use of the “random” button to get a new puzzle. If this can be generalized to all of 34 participants in this condition then it explains why the PCG manipulation may not have had any effect. For the US pilot study about a third in the PCG condition did not make use of the “random” button. As for the levels played, compared to the 17 (41%) US participants who completed the final level, only two (7%) did for the Nigerian students. Moreover, the US participants were able to complete at least Level 5, for the Nigerian students this was Level 3. In fact, only six (21%) were able to complete a level higher than six compared to the 38 (90%) of the US participants.

In total, the Nigerians (M = 33.4, SD = 25.3) played on average as many levels as the Americans (M = 31.9, SD = 13.0); they just played the earlier levels more so and also needed more time to play these levels. For example, it took the US participants a little over a minute on average to play Level 1 whereas it took the Nigerian participants more than four minutes. A difference in performance is not clearly noticeable until Level 4, where the Americans fail about half as much as the Nigerians, t(69) = 2.34, p = .024, r = .27. Interestingly, and unlike with the US pilot, no difference is noticeable in the time to play for the collaborative condition.

3.7 Discussion on USA Results

Our experience with designing the game, constructing instruments to evaluate it, and performing a pilot study has led to a number of design insights and areas for future study. It should be noted that the current study is limited by using a biased group of diverse but well performing students who are interested in STEM.

The potential of PCG and collaboration. With the results in this instance of GrACE, we must outright reject Proposition 2 since the opposite happened. The students in the collaboration conditions improved less. The possibility to discuss with partners does not outweigh playing more exercises, and it was noticeable that pairs played less and got less far. Future research should look into the communication between pairs and ways to better support collaboration through the game. Although an interaction effect seems to happen, it is also different than expected. PCG is only beneficial for those who play alone. For that reason we need to reject Proposition 3 about the multiplier effect of PCG and Collaboration too. However, we can partially accept Proposition 1. The current data provides an indication that when PCG is used in an individual context, increased learning gains are a result. The fact this is
especially noticeable for the abstract puzzles is a promising result. Teaching abstract thinking is a difficult yet important skill to master, especially for students aspiring to a CS career.

**Designing for collaboration.** In keeping with other findings for educational games in non-CS STEM fields (Ke and Grabowski, 2007; Plass et al., 2013), our findings suggest that players are more likely to recommend the game to a friend in the collaboration condition, yet perform better in the game in the individual condition. The goal of informal learning interventions is to be both engaging and educational. The Collaboration condition in GrACE does not involve students collaborating on the same puzzle, but rather collaborating to help each other with their own puzzles outside the game. In future versions of the game, we intend to explore ways to make this strategy-level collaboration more explicit and occur within the game in order to achieve a positive collaborative learning effect.

**Placing content generation in context.** A common argument in the game design and PCG community is that additional content leads to replayability and enjoyment (Smith, 2014). However, our findings show that playing in the PCG condition leads to greater frustration. One possible explanation for this is that students who are struggling with one particular puzzle may generate new content hoping for an easier puzzle, but instead find one at the same difficulty level that is also challenging. While frustration may be linked to game enjoyment overall, this effect is something that requires more dedicated study. It is clear that the context in which PCG is used and how it is integrated into the game’s overall mechanics must be carefully considered.

**Fun vs. frustration** Students reported a high rate of enjoyment for the game but also found it frustrating, challenging, and hard. This is perhaps tied to Papert’s concept of “hard fun” in educational games: not that players find a game fun despite it being hard, but that they find it fun because it is hard (Papert, 1998). While many educational games work to simplify concepts and offer extremely gentle introductions, with GrACE it seems that its difficulty is important for the enjoyment of the game. Our future work will include examining what specifically it is about the game that students find frustrating by looking at patterns in behavior from the game data, analyzing player demographics more in-depth, and testing additional iterations of the game.

**Beyond gender: Understanding player profiles.** Researchers have often discussed the importance of designing games that are accessible to girls, either by attempting to understand the design preferences exhibited by girls (Dickey, 2006; Denner et al., 2005) or by incorporating stereotypically feminine play styles and preferences into a game’s design to make it “gender-neutral” (Ray, 2004). However, our results show that while gender identity can influence player enjoyment, it is not alone responsible. Indeed, genre preferences (such as a preference for puzzle games, action games, or competitive sports games) have an impact on both experience and outcome performance, independent from gender. This points to the need for a deeper understanding of a target audience, refocusing design efforts away from looking at gender alone.

### 3.8 Discussion on Nigerian Results

This study has helped to understand the differences in how students of different cultural backgrounds perceive, interact with, and learn from an educational game. After the implementation, the Nigerian students revealed that they wanted inclusion of hints, which is understandable considering their difficulty with progressing in the
game. They also wished for a mobile version of the game. This is also not surprising as smartphones have rapidly penetrated Africa, and is arguably a very important tool in the future of education in Africa (Brown, 2003). However, the main takeaways from this study are the performance gap and the performance paradox. Based on these takeaways and the other results, we discuss the implications, also considering the limitations of this study.

3.8.1 Performance Gap
It is clear from this study that the USA students performed much better than the Nigerian students in terms of test scores. However, it is worthy to note that due to the program’s selectivity, there is an inherent bias which may be responsible for the wide margin. The USA students were selected based on high academic performance while the Nigerian students consist of a blend of students with varying academic status. With the Science Academy, more than 50% of the participants have their own laptops. The majority of the participants from the Summer Camp are from a humble background with very limited or no access to computers. This tacitly suggests that their level of proficiency with computers is less compared to USA students and this may have impaired their ability to learn from the game, as they have more trouble to learn how to play it. Additionally, the Nigerian students have increased variance of game experience compared to the USA students which may have played a role too. Of course, a possible bias may have been the research instruments that we used. Although we modified the language of the test, the test itself may have been too difficult to measure improvement. Regardless of all these possible biases, the performance gap is so significant that it would need to be considered in designing educational games for cross-cultural impact.

3.8.2 Performance Paradox
Overall, the game did not have a significant educational impact on Nigerian students, which is in contrast to the moderate improvement of the USA students. However, a performance paradox became apparent. Those who did not enjoy the game actually made drastic improvements. It should be noted that these students were the lowest performing students at the start, suggesting that they are academically either uninterested or have more difficulty in performing well. This outcome is promising but also discouraging. On the one hand, the game seems to impact those who need it; on the other hand, those who need it may not make use of it as they do not find it enjoyable.

3.8.3 Implications for Cross-Cultural Design
It is clear from the study that the USA and Nigerian students experienced the game in similar ways. The increased enjoyment expressed by the Nigerian students may be the result of these students receiving less exposure to innovative educational technologies in their classrooms such as games. In fact, in Nigeria a game is still not considered a learning tool in most schools. The idea of a game is even considered “taboo” in some classrooms. An interesting finding is that Nigerian girls seem more open to games than US girls, which may be telling about the role of games in both cultures. However, in general it is clear that the Nigerian students experience and play different games, which may have hindered them in accessing the learning content of the game. Both the performance gap and paradox are issues for further investigation, but it seems
clear that varying levels of difficulty may be necessary to increase cross-cultural usage.
Chapter 4

Analyzing the Behavioral Effect of PCG on Players

Once we have established the overall performance of GrACE players and validated our assessment materials, we move to understand individual players and the behavioral effect procedurally generated content has on them in response to our first research question stated in Section 1.4: How does the introduction of procedural content generation in an educational game affect player behavior and performance? In this chapter, we analyze individual players to gain insights into instances of frustration, evaluate our PCG system, and differentiate player strategies.

4.1 Game Analytics

Through the emerging field of game analytics (Seif El-Nasr, Drachen, and Canossa, 2013)—analogous to learning analytics (Siemens and Baker, 2012)—methods for both evaluating gameplay and analyzing player behavior have been dramatically expanding (Loh, Sheng, and Ifenthaler, 2015). Rather than evaluating learning outcomes through external evaluations alone (e.g., questionnaires and tests), recent approaches investigate player actions logged during gameplay to provide insights into the learning processes of players (Linehan et al., 2014; Harpstead and Aleven, 2015; Harpstead et al., 2015a). A significant challenge in game analytics and educational game design lies in successfully combining traditional methods of assessing knowledge with tracking and analyzing behavioral telemetry for the purposes of both assessing learning and improving the design of educational games.

This chapter contributes to analyzing educational games with game analytics by exploring player strategies in GrACE, an educational puzzle-based game for middle school aged students (11-13 years old) that is designed to support algorithmic thinking. While many educational games such as CodeCombat (CodeCombat, 2016), Robocode (O’Kelly and Gibson, 2006), and Robozzle (Li and Watson, 2011) also focus on supporting algorithmic thinking, they often emphasize the structure and syntax of computer programming rather than addressing the core planning and strategy development processes. GrACE on the other hand, supports algorithmic thinking by presenting players with a puzzle analogous to solving a typical computer science problem (i.e., finding the minimum spanning tree). Through navigating and solving multiple puzzles, the goal is for players to learn both the data structure and the step-wise algorithm for solving it.

One obstacle in game analytics is that it is often difficult to correctly interpret the meaning of player data (Hazan, 2013). Recent work on Data-Driven Retrospective

\[^{1}\text{Work presented in this chapter previously appeared in the Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play (Horn et al., 2016b).}\]
Chapter 4. Analyzing the Behavioral Effect of PCG on Players

Interviewing (DDRI) demonstrates a method to evaluate game data and interview users to understand how a game is received by players (El-Nasr et al., 2015). DDRI merges quantitative and qualitative data so that designers can make appropriate changes to their game based on user feedback and behavioral game data. To interpret the meaning of player actions in GrACE, we take a similar approach of triangulating findings by combining quantitative hierarchical cluster analysis of player actions with a qualitative analysis of playtraces supported by concurrent think-aloud data and progression visualizations. We find that this mixed-methods approach helped discover emergent player strategies and how the game mechanics may have supported such strategies, findings that may not have been visible through traditional assessments or game analytics alone. Our work unveils prevalent player strategies by “opening the black box of play” using this triangulation approach, building a deeper understanding of how players learn and progress in the game, and aiding decisions in the (re)design process of building an effective educational game. Included is an in-depth analysis of the GrACE level progression and how it may help or hinder players at specific points in time.

4.2 GrACE Design

GrACE is a puzzle game with a vegetable-collecting narrative centered on two characters named Scout and Hopper designed to encourage algorithmic thinking. Algorithmic thinking involves thinking about problems abstractly, identifying common traits so that they can be treated as a class of problems instead of a single instance, and building sequences of instructions as solutions. It is a logical method of problem solving that can also be applied in areas outside computing (Tucker et al., 2006). Because GrACE is being developed in part to interest girls in computer science, the puzzle genre is chosen based on research suggesting it is appealing to girls (Greenberg et al., 2010; Phan et al., 2012). Furthermore, the puzzle genre is one of the top two genres played by teens (Lenhart and Project, 2008) and like much of the computer science discipline, is about logically solving complex problems (Harteveld et al., 2014).

Specifically, GrACE aims to illustrate the potential of teaching algorithmic thinking through puzzles analogous to finding the minimum spanning tree (MST) of a graph, a core problem in computer science. These MST-based puzzles were chosen over other possibilities for their ease of visual representation, the existence of many algorithms to solve the problem, and that even finding an incorrect solution (i.e., a spanning tree that is not minimal) involves algorithmic thinking.

Abstractly, the MST-based puzzles in GrACE are represented as graphs, which are a collection of nodes connected by edges. Each edge has an associated cost as shown in Figure 4.1(a), which shows an abstract MST puzzle. Players travel from one node to another along the edges that connect them. The cost of including an edge in the solution is represented by the number above the traveled edge. For example, in Figure 4.1(a), the cost of the edge between Node 0 and Node 2 is three.

In these graphs, players can travel along any edge connected to the node they are currently visiting. To indicate a particular edge should be considered in the solution of the puzzle, players can flag the edge when visiting the nodes connected to it. The darkened edges in Figure 4.1(b) like the edge connecting Node 0 and Node 4 represent such flagged edges.

A “spanning tree” is formed when a player visits all of the nodes and flags edges such that each node is only connected to the others once (as shown in Figure 4.1(b)). While a graph can have many different spanning trees, those with minimal cost are a
Figure 4.1: Every connected graph has a variety of spanning trees. A selected graph is shown in (a) with one of its spanning trees shown in (b) and its minimum spanning tree shown in (c). The minimum spanning tree is the set of edges that connects all nodes without cycles and with the minimum possible total edge weight.
Figure 4.2: Screenshot of GrACE. This puzzle currently shows three nodes (as other nodes may be revealed later) with two edges, both with a cost of one, indicated by the arrow and number of rocks. Flagging involves literally placing a flag in the middle of an edge. On the top right are Hopper’s and Scout’s energy levels, to the left are the level select options, and at the bottom the submit button.

MST (shown in Figure 4.1(c)). The abstract goal in GrACE is for players to find a MST by traveling along and flagging the least costly path where each node is connected to the others exactly once.

Once players think they have explored the graph sufficiently and found the best possible solution, Hopper will dig tunnels along the flagged path, and collect the vegetables. The total cost of the flagged paths represents the amount of work for Hopper to dig tunnels, and if this is more than Hopper’s available energy (represented as an energy bar) then players have not found an MST, which is necessary to successfully complete the puzzle and proceed to the next level.

Scout also has an energy bar which decreases with every action the player takes (moving, flagging, unflagging). For completing the level, Scout’s amount of energy is not relevant except if she runs out, because in that case players will have to try again. Scout’s energy has been included to encourage players to explore puzzles efficiently, and thereby algorithmically.

To simulate the way computers “think” through the problem node by node while keeping track of discovered edges and their weights, players initially can only see the starting node and the nodes which are directly connected to it. Players choose an edge to traverse at which point the new node and its connections are revealed. By slowly revealing the entirety of the graph, players are encouraged to develop an algorithmic approach which helps them solve more complex problems that would be challenging with only visual inspection.

Once all nodes have been explored and desired edges flagged, the player may submit their answer by clicking the “Submit” button. A correct answer sends the player to the next level which will contain more nodes and edges than the previous, while an incorrect answer leaves several options: the player may edit their submitted
answer (which will continue to reduce Scout’s available energy), restart the level, or replay any previously successfully submitted puzzles.

4.3 Method

To both explore how well GrACE encourages algorithmic thinking and to evaluate the ability of players to solve MSTs, this chapter analyzes game data collected from a pilot study with GrACE as outlined in Chapter 3. Results from traditional external evaluation measures indicated a moderate improvement in scores after playing. Gameplay metrics were also considered (e.g., time for completion of each level, the number of correct and incorrect submitted solutions, and the number of levels completed), and indicated that participants tended to stumble at particular levels, with some never completing a level past the fifth (out of 11). Because some participants completed all eleven puzzles and some did not, a difference in strategies (and learning) may explain the variety in progress. To analyze players’ strategies, we selected a small sample of participants and performed an in-depth analysis using qualitative and quantitative techniques.

Our approach combines a quantitative hierarchical clustering of player actions over the course of the game with a qualitative analysis of playdata that we call retrospective player sense-making collected through concurrent think-aloud data. Retrospective player sense-making involves reconstructing what a player did in the game by examining their playtrace step-by-step, noting observations along the way. Cluster analysis in particular is susceptible to issues of interpretation and trial-and-error (Sugar and James, 2003; Thorndike, 1953), so the advantage of this combined approach is that findings can be triangulated. Also, the think-aloud data may provide insights into players’ in-the-moment perceptions and goals. Playtraces alone only help infer what players did rather than why they did it. Overall, our goal was to understand the divergence of player trajectories beyond quantitative game data. That is, we wanted to get an idea of why some players got stuck on certain levels and others were able to progress easily.

4.3.1 Study Context

The pilot study (Chapter 3) was implemented as a programmed three-hour activity in a two-week summer program at Northeastern University that focuses on STEM education and retention for underserved and underrepresented minorities. Talented middle school students (ages 11 through 13) are selected to participate in this free of charge program. The original study was a 2x2 experimental design, where students received a prequestionnaire and pretest at the start. Following an explanation of how to play the game (without mentioning MSTs or the game’s relationship to computer science), participants were randomly assigned to one of the conditions and played the game for about an hour. If students finished playing all levels, they were encouraged to repeat past levels and attempt to minimize the energy used by Scout the mouse. The activity ended with a postquestionnaire and posttest, followed by a group discussion.

Selection of Case Studies

Out of the 43 students who participated with the consent of their parents in the pilot study, we selected nine. These nine students participated in the same condition and played the same puzzles making it possible for us to perform an in-depth comparative analysis on how they played. For these selected students, ages ranged from 11 to
13 ($M = 12.2$, $SD = 1.06$). Four students identified themselves as female and five as male. Additionally, one student identified as Hispanic or Latino, three as Asian, one as Black or African American, two as White, and one as “other” (with one participant preferring not to answer). Each student indicated interest and experience playing games (e.g. video games, board games, sports), with some indicating that they played games several times a day while others only played every few weeks. Therefore, the selected nine students were diverse in terms of their socio-demographics and frequency of play (see also Table 4.1).

**Data Collection**

Of relevance to this chapter is that players were asked to complete a postquestionnaire on their game experience with *GrACE* as well as a pretest and posttest to assess their algorithmic thinking. The postquestionnaire measured play experience with four 7-point Likert items focused on enjoyment and four on difficulty. The resulting composite scores range from 4 to 28, and are both reliable and valid as seen in Chapter 3.

The tests were developed to measure conceptual understanding of the MST and involved a multiple choice assessment of nine test questions with four choice options. All test questions were phrased in the context of a specific MST puzzle such as deciding what edge to add, what edge to remove, and what needs to be corrected. Test scores were calculated by providing one point per correctly answered question. Thus, with nine questions students could get a maximum score of nine points. The pre- and posttest differed in the exact questions asked, but were the same type.

The game data logged for each player was collected and each player also received a USB voice recorder to record their talk while playing. We encouraged students to think-aloud while playing so we could infer from their talk why they are playing in a certain way.

**4.3.2 Analysis**

The analysis considers players’ strategies and progression through cluster analysis and retrospective player sense-making. For the cluster analysis, each playtrace was first converted to a string representing the sequence of actions performed. For instance, if a player successfully completed a level by starting at Node 0, moving to Node 1, flagging that edge, and then submitting their answer, these actions would be represented as “Start Node0, Move Node0 Node1 Edge1, Flag Edge1, Submit Correct.” These strings are then compared by calculating the Levenshtein string edit distance between them (Levenshtein, 1966; Osborn et al., 2014), which counts the number of additions, substitutions, and deletions necessary to convert one string to the other. This number indicates the similarity between strings of playtraces.

For the retrospective player sense-making, we considered player strategies for each level and how these strategies evolved as players attempted each new level. We also used level progression visualizations to observe where players got stuck and puzzle features that may explain why. Once patterns emerged, we worked to better understand these patterns by supplementing initial findings with in-the-moment audio recordings of players during the game experience.
Chapter 4. Analyzing the Behavioral Effect of PCG on Players

Figure 4.3: Example Player Progressions. Progression 1 shows a player progressing from level 1 to 2 to 3, repeating level 3 a few times, then progressing from 3 to 4 to 5, repeating level 5 many times, and finally going back to retry level 2. Progression 2 shows a player progressing from level 1 all the way to level 11 without repeating any levels. Arcs connecting dots from above indicate forward progression and arcs below the dots indicate a player going back to a previous puzzle. Loops above a single dot indicated retries of the same level.

Strategy Analysis

For the strategy analysis, playtraces are compared to a standard algorithm for finding the MST of a graph called Prim’s algorithm (Prim, 1957), which maintains and grows a tree by incrementally selecting the lowest cost edge encountered. Only edges that connect previously unconnected nodes are added to the MST solution and the process is repeated until all of the nodes are connected. If players solve the puzzles according to Prim’s algorithm, it shows that GrACE helps to solve computational problems algorithmically. Other algorithmic approaches are possible, but Prim’s algorithm was chosen as the golden standard algorithm because it directly aligns with our game mechanics.

We first performed a hierarchical cluster analysis to tease out different play styles used by GrACE players. Because the number of emergent strategies was not known a priori and hierarchical clustering operates without a predetermined number of clusters, each player run is hierarchically clustered based on the edit distances from one another and from the solution found by Prim’s algorithm. Since sorting through qualitative data is time consuming, the idea is that clustering helps target the runs and level structures that need additional analysis with retrospective player sense-making.

For the retrospective player sense-making individual strategies were analyzed by retracing player steps by hand. This analysis involved redrawing on paper how players played each puzzle. Although this exercise seems tedious compared to watching a video of a player, the actual act of replaying the player steps on paper encourages us as researchers to imagine how players were making sense of the puzzle while playing, and therefore what strategies they were employing. In analyzing, the player steps were compared to a single run of Prim’s algorithm on the same level.

Progression Analysis

Levels in GrACE are designed to increase in difficulty through the progression of the game, where difficulty is based on increasing numbers of nodes and edges.
Levels with many failed player submissions indicate the challenge it posed to players. Progression visualizations can help evaluate whether the puzzles are progressively more difficult and where players first display some misunderstanding. We used a similar progression visualization method as Linehan et al. (2014) who used it to detail how novel skills are introduced in a progressive manner.

An example of our player progression visualization method is shown in Figure 4.3. Each progression visualization represents a single playthrough of the game, but may not necessarily include completion of each level. For instance, Progression 1 in Figure 4.3 is an incomplete playthrough where the player never reached Level 6. Each dot represents a particular puzzle. The dot on the far left indicates Level 1 and on the far right Level 11. Lines connecting puzzles that arc upwards show forward progression in the game (e.g., moving from Level 1 to Level 2). Lines that arc downward are backward progressions (e.g., moving from Level 5 to Level 2) and indicate a replay of a previously completed puzzle. Loops starting and ending at the same dot are replays of the current level. The arc size (height and line width) and the boldness of the line are proportional to the number of times an answer for a particular level is submitted. For example, Progression 1 in Figure 4.3 has a large, wide loop on Level 5 illustrating that the level was attempted many times. Progression 2 in Figure 4.3 illustrates a player who progresses through the levels from Level 1 to Level 11 without replaying a current or previous level.

**Qualitative Analysis of Think Aloud Data**

To better understand the player experience and address the emergent questions that arose from the aforementioned analyses, we analyzed the approximately hour-long recordings for each of the nine participants. Any audible utterances were transcribed and time-stamped, to later compare with game actions taken during the same time. The transcripts were investigated in two ways: 1) the talk was searched for evidence of strategy articulation during the trajectory of the players’ experience and 2) the transcripts were used to align utterances to player action patterns at specific points of gameplay.

### 4.4 Results

Results in this section explore the emergent strategies players exhibited and how well they can solve and learn MSTs through GrACE. Each player is represented by one of the following colors: Red, Orange, Yellow, Green, Blue, Indigo, Violet, Brown, Maroon. Each player’s pretest, posttest, enjoyment of the game, assessment of its difficulty, frequency of game playing in general, and strategy is reported in Table 4.1. All but two individuals showed improvement from pretest to posttest (Green and Indigo). The table further shows diversity in how players experienced the game with no clear relationships between any of the measures, which indicates that these external measures may not provide the necessary insight to explain for these results, and that their actual play needs to be explored. The only interesting observation that can be inferred from the table is that it seems that players who play few games (Red, Orange, Brown, and Maroon) in their free time seem to find the game more fun and less difficult.

A broad overview of how players progressed through the game can be inferred from the progression visualizations in Figure 4.4. For instance, it is apparent that
Chapter 4. Analyzing the Behavioral Effect of PCG on Players

<table>
<thead>
<tr>
<th>Participant</th>
<th>Pre</th>
<th>Post</th>
<th>Fun</th>
<th>Diff</th>
<th>Freq</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>NA</td>
<td>7</td>
<td>H</td>
<td>M</td>
<td>L</td>
<td>D</td>
</tr>
<tr>
<td>Orange</td>
<td>5</td>
<td>6</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>IT</td>
</tr>
<tr>
<td>Yellow</td>
<td>8</td>
<td>9</td>
<td>M</td>
<td>M</td>
<td>H</td>
<td>D</td>
</tr>
<tr>
<td>Green</td>
<td>8</td>
<td>8</td>
<td>L</td>
<td>H</td>
<td>M</td>
<td>IT</td>
</tr>
<tr>
<td>Blue</td>
<td>5</td>
<td>8</td>
<td>L</td>
<td>M</td>
<td>H</td>
<td>E</td>
</tr>
<tr>
<td>Indigo</td>
<td>7</td>
<td>5</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>D</td>
</tr>
<tr>
<td>Violet</td>
<td>3</td>
<td>4</td>
<td>H</td>
<td>H</td>
<td>M</td>
<td>D</td>
</tr>
<tr>
<td>Brown</td>
<td>8</td>
<td>9</td>
<td>M</td>
<td>L</td>
<td>L</td>
<td>D</td>
</tr>
<tr>
<td>Maroon</td>
<td>5</td>
<td>7</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>D</td>
</tr>
</tbody>
</table>

Table 4.1: The nine participants and their questionnaire results from the pilot study and predominant strategy from analyzing their gameplay. Questionnaire scores include pretest (Pre) and posttest (Post) scores (each ranges from 0 to 9), enjoyment (Fun) of GrACE (composite score ranging from 4 to 28), how difficult (Diff) they perceived GrACE (also from 4 to 28), how often they played games (Freq), and their predominant strategy (Strategy) during their initial playthrough of the game. Enjoyment and difficulty are listed in terms of Low < 14 (L), Medium = 14-20 (M), or High > 20 (H). Freq of game play ranges from 1 to 6 (“Never or Almost Never” to “Several Times a Day”). Freq is listed by Low = 1 or 2 (L), Medium = 3 (M) or High = 4 or 5 (H). There are three strategies: deliberate (D), exploratory (E), or iterative testing (IT).

Green experienced difficulty grasping the game concepts as he completed only up to Level 5, and made multiple attempts from higher levels to go back and repeat lower levels. That idea is supported by the player’s audio data expressing frustration and his expressed low enjoyment and high difficulty in playing the game. Similarly, though Blue never attempted to solve levels before her current level, her highest completed level is Level 9 and has a low enjoyment too. Blue, however, did seem to make the most visible improvement on the tests.

Evaluating successful players is less straightforward. For instance, because Red, Orange, Yellow, Indigo, Brown, and Maroon completed the game multiple times, they appear to have implemented an efficient strategy; however, further analysis of each player on each level leads to three emergent strategies called the 1) deliberate strategy, 2) exploratory strategy, and 3) iterative testing strategy. These strategies are indeed similar to previous research that grouped player strategies in response to challenge in the process of learning in single player games (trial and error, experiment, repetition, stop and think, and take the hint) (Iacovides et al., 2014). A deliberate strategy typically indicates the player has a clear understanding of how to solve the puzzle by flagging as they move, and an iterative testing strategy typically indicates player experimentation or that the player is having difficulty solving the puzzles. An exploratory strategy is often an intermediate strategy where players explore large portions of the map before flagging any edges. The following analysis is broken up into analyzing these player behaviors at each level.

4.4.1 Play Analysis in Level 1

To help players understand basic gameplay in GrACE, Level 1 is based on the simple graph illustrated in Figure 4.6(a). It has two nodes, called Node 0 and Node 1 and one connection with a weight of one. Depending on their current position, players
can either move left or right, and flag or unflag the path until their energy runs out. Solving this puzzle indicates a basic understanding of game mechanics.

All players eventually solve Level 1 with the least amount of moves, most on their first try. In fact, in the clustered representation of successful Level 1 attempts shown in Figure 4.5, each player has at least one run of Level 1 clustered with the computer’s solution obtained through Prim’s Algorithm. In this level, however, many also experiment with the mechanics. For instance, Yellow whose first run is solved near perfectly, begins experimenting with Level 1 on the second run. While in the first run, Yellow solves the puzzle by moving from Node 1 to Node 0, flagging the path and then submitting on the first try. In subsequent attempts, the player tests the game functionality figuring out whether it is necessary to move to each node and if a path can be flagged before moving to it.

Several users did have difficulty understanding these basic game mechanics, only reaching a solution after several attempts. Only three actions are necessary to solve this puzzle, yet Orange performed ten actions in the first two runs of the level, both unsuccessful. On the third try of Level 1, the correct solution is found in nine tries, seemingly through trial and error. Eventually on the fourth run the puzzle is solved in three actions: move, flag, and submit.

### 4.4.2 Play Analysis in Level 3

Level 2 mostly reinforces the concepts that nodes must be visited, edges flagged and new information about a node’s connections is discovered by visiting it. Level 3, on the other hand, challenges players to selectively choose edges to flag, important for identifying MSTs. Shown in Figure 4.6(b), a correct solution is moving between all the nodes, flagging connections between Nodes 0 and 1 and Nodes 1 and 2.

Though Red and Green start with a deliberate strategy, seemingly understanding flagging edges with minimum weights, once both encounter an “Out of Energy” error they retry the puzzle. During the second try in the same run, Red explores crossing the highest weight edge several times before running out of energy again,
Chapter 4. Analyzing the Behavioral Effect of PCG on Players

Figure 4.5: Clustering of successful playtraces for Level 1. Colored labels on the right indicate the player ID and run number. Some players completed one run (Blue and Violet) while others completed many (Green).

Figure 4.6: Graph depictions of levels in GrACE. Levels in GrACE can be abstracted into graphs, where each node represents a burrow for the character to investigate for vegetables and each edge the path to a particular burrow. The significant Levels 1, 3, 4, and 6 are depicted.
but eventually submits the correct answer. Similarly, Green submits the correct solution the second time Level 3 is played, but does not experiment with crossing the higher weight. Audio data suggests that Green grasps the goal of efficiency when exploring the graph. While playing Level 3, he utters “I’m not even sure how you do it...But, I don’t need to go over there. Then, I run out of energy.” Though he is likely still learning navigational affordances in the game, he appears to be attending to the energy constraints encountered by moving. Similarly, despite his actions Red acknowledges energy constraints, saying “…Too much time exploring."

Yellow, Brown, Orange, and Indigo also all execute deliberate strategies while Violet and Maroon who had previously executed deliberate strategies start each run in Level 3 by exploring all of the nodes and then flagging them in an exploratory style. Blue previously implemented an iterate and test strategy, but for this level switched to exploratory.

This is the first level where players rerun levels to achieve the most efficient solution (i.e., one completed in the least amount of steps). Only Yellow and Green find the solution on their first attempts. Most players eventually converge on the Prim’s solution, yet Orange and Blue never find a solution comparable to Prim’s.

### 4.4.3 Play Analysis in Level 4

With a total of four nodes and four edges compared to Level 3’s three nodes and three edges, Level 4 shown in Figure 4.6(c) produces even greater challenges to the players. Furthermore, in this level it is possible to select two edges not connected to the same node. It is evident from the progression visualizations that Level 4 presents the first big hurdle for players with most having an increased number of attempts on their first playthrough of the game. The correct solution involves flagging a path of weight 4 between Nodes 1 and 2, and choosing not to connect Nodes 0 and 3 with weight 2. The other two paths with weight 1 should be flagged instead.

Initially, players struggle to only connect the lowest weight edges. For example, Blue and Violet begin by flagging all of the edges, while Maroon connects all of the nodes and excludes an edge, but does so without regard for which has the lowest weight. Interestingly, on a second playthrough of the game, Maroon solves the puzzle in a number of steps similar to that calculated by Prim’s.

Perhaps because all lowest cost weights in previous levels were 1, it is possible that some players faced difficulty flagging the only connection to Node 2, which has a weight of 4. Everyone eventually solves Level 4, however, only Yellow, Brown, and Maroon present solutions comparable to Prim’s. Of these players, only Yellow solves it like Prim’s on the first try. These players perform the best throughout the remaining levels, with Yellow and Brown scoring perfectly on the posttest, and Maroon improving more than others (except for Blue). Interestingly, Maroon found it more enjoyable than Yellow and Brown, possibly because she learned more.

### 4.4.4 Play Analysis in Level 6

Pictured in Figure 4.6(d), Level 6 is a graph with six nodes and eight edges, and is the first to present a node connected to four other nodes, testing the player’s ability to choose which edge for the node should be flagged. Level 5, the last that all players were able to solve, presented at most three edge selections at any given node and had two fewer nodes overall.

Even Yellow who had previously solved each level easily and optimally took three tries to solve the puzzle. Starting at Node 2, on her first try, Yellow moved along a
shortest path to Node 4, flagged it, and then moved to Node 5 across a path of weight 4. She flagged the higher weight path after visiting Node 5, meaning this weight was chosen after seeing the cheaper connection to Node 5 from Node 3. From Node 5, she went to Node 3, flagged the path and then moved to Node 1. After moving from Node 1 to Node 0, Yellow ran out of energy and restarted the level.

Since Yellow previously seemed to understand the necessity of choosing the lowest cost path, it is notable that she chose the weight path of value 4 connecting Nodes 4 and 5. Though she eventually finds the correct answer, the confusion is reflected across other players. For instance, although Violet starts deliberately moving and flagging, after the first submission fails, he explores and iterates and tests until six submissions later he finds the right answer.

When comparing to the answer found by Prim’s algorithm, for this level the edit distance from any given solution is greater than zero, with Brown’s second try and Maroon’s second and third tries being the closest. However, Orange, Yellow, Indigo, and Brown were all eventually able to complete Level 6 with only one more action than Prim’s algorithm.

### 4.4.5 Play Analysis Levels 7 through 11

Levels 7 through 11 continue to reinforce the concepts introduced by previous levels, challenging the players’ existing strategies on more difficult puzzles. Players who have already discovered the deliberate strategy described in Table 4.1 have little trouble completing these levels, while those with exploratory and iterative testing strategies struggle with the additional nodes and edges.

Level 9 is the last level Blue completes, perhaps because Level 10 jumps to eight nodes, 13 edges, and a maximum connectivity of six, versus the six nodes, ten edges, and maximum connectivity of five seen in Level 9. The average connectivity of a node decreases slightly (3.30 to 3.25), but the highest connectivity of a node increases from 5 to 6.

Inspecting the clusters indicates that Red, Orange, and Violet begin to diverge from Prim’s algorithm at Level 7, and Red and Orange continue this pattern until the end. Interestingly, Red and Orange sat next to each other while playing the game and were frequently talking to each other during game play, although they were instructed to solve the problems on their own—a fact highlighted only through audio data. Importantly, their similarity in play was detectable in their game actions. Their conversation included critique of the game—first Red then Orange posited that they found “a bug.” Red explained that the game “says I’m wrong when in actuality I was right.” Later, he repeated “the last few levels I had the right answer and it said try again.” Occasionally Orange asks Red for help and Red is continuously vocal in discussing his critiques with researchers. Through clustering of their playtraces and retrospective player sense-making, the difficulty they faced solving levels in GrACE is evident.

Indigo, Maroon, Brown, and Yellow continued to solve the puzzles in a manner similar to Prim’s algorithm. Interestingly, Violet seemed to be implementing a deliberate strategy followed by an amount of guessing and checking, and by Level 11 the strategy was clearly cemented. Violet’s progression in Level 10, began to hint that the main ideas were being learned.

In fact, our analysis suggests that over the course of the game, most students learn to visit all of the nodes in the MST, prefer traveling on shorter paths to avoid running out of energy, the concept of a spanning tree, and how to find the MST.
4.5 Discussion

Our results highlight that including game data in analysis can shed light onto findings from traditional measures in addition to giving a better understanding of what players actually do. However, in some cases it raises more questions such as in the interesting case of Blue. In this section, we focus on discussing our findings for what made levels more or less difficult and the emergent player strategies and their implications for educational game design.

Although this work represented analysis of data for a particular game, future work will focus on its scalability to larger data sets and different games. For instance, with data from more players, an archetype clustering (Cutler and Breiman, 1994) can be performed helping sort the data into small enough subsets to perform meaningful hierarchical clustering.

4.5.1 Level Complexity and Progression

To avoid boredom and frustration, and to support learning effectively, it is important that the difficulty of levels increases steadily over time. However, through the progression analysis in this chapter, it is apparent that Level 6 was the first to present such a significant challenge to players that some could not successfully complete it. By tracing through individual player actions and examining the level structure (i.e., its underlying graph), part of the challenge appeared to be the increase in the number of nodes and edges from that in Level 5 (i.e., from four nodes to six nodes and five edges to seven).

Interestingly, while Level 11 also increased the number of nodes and edges from that in Level 10, this increase posed less of a challenge to players. While an explanation for this difference could in part be that those who were struggling most never reached Levels 10 and 11, further analysis across all levels in the progression reveals that in fact the biggest predictor of challenge lies in the number of choices available to players at a given node (i.e., the number of edges connected to a node). The number of edges connected to the node with the greatest amount of choices represents the maximum connectivity of the underlying graph. Although increasing the number of nodes and edges can lead to increased maximum connectivity, alone it is only a secondary indicator of difficulty. In fact, Levels 4, 6, 8, and 10 all saw increased maximum connectivity and were the levels players struggled with the most. Before our analysis, it was thought level difficulty could be sufficiently represented by the number of nodes and edges; however, a detailed trace analysis revealed the primary indicator of level difficulty as the graph’s maximum connectivity.

Although the issue of maximum connectivity of a graph is specific to graph-based games, the process of discovering this phenomenon is not. We suggest that designers of educational games add into their development cycle a phase exploring where players get stuck, identifying the factors that contribute to challenges for those players, and then modifying and retesting the level progression. Through application of this procedure, designers can adjust the progression in an effort to reduce player frustration and increase learning.

That said, it is an open question to designers if reducing frustration needs to be aimed for. Blue is an interesting case. She did not enjoy the game and did not progress as far in the game as others, yet made the most visible improvement. Although Blue is a single case, in our other analyses with GrACE (Chapter 3) we found a similar paradox: that players who did not enjoy it seemed to be the ones most benefiting
from it. Regardless of how to deal with frustration, it will be beneficial to designers to understand what makes their game really difficult.

4.5.2 Player Strategy

Player strategies were determined through the mixed-methods approach described in this chapter. We discuss how we determined such strategies and what they mean in the context of other findings. In addition, we discuss the implications for educational game design, and focus on the importance of game mechanics in encouraging (or hindering) learning and determining the appropriate measures of success.

Determining Strategies

Whether a player solved a level is important information in analyzing an educational game, but to explore learning it is crucial to understand how players solved the problem. Although player strategies are often discerned through laborious qualitative studies (Iacovides et al., 2014), the mixed-methods approach presented in this chapter reduced time spent analyzing the qualitative data by allowing us to focus on critical game events and problematic levels. Clustering players’ game actions combined with playtrace and audio analysis led us to determine that six of the nine players employed a deliberate strategy while two used an iterative testing method and one used an exploratory strategy.

When characterizing player strategies, it is difficult to conclusively state why particular actions were taken, but some evidence exists as to why player experiences diverged. For example, there appear to be trends related to player engagement as shown in Table 4.1. Player strategy appears independent to either their enjoyment or perception of difficulty. Looking at Orange and Green, who both took iterative approaches, Orange had a low rating of difficulty with a high rating of enjoyment. Green had a high rating of difficulty and a low rating of enjoyment. At least for the iterative approach, strategy development does not appear related to enjoyment or difficulty; however, both players had a gain of either 0 or 1, and so it may be the case that iterative strategies did not support learning. These are potential trends that need to be explored further.

Although surprising, improvement in score does not appear to be driven by enjoyment of the game. Counter-intuitively, those who played games more often rated the game at a medium or high difficulty, and medium and low enjoyment. One might expect that frequent gamers might enjoy a new game, and would find it less difficult than other players. These patterns require further investigation, but one hypothesis is that the players may have perceived the game as an educational task, rather than a true game, and the very logical nature of the task compared to the games they usually play, may have affected their engagement, learning (e.g., despite low engagement, Blue was one of two participants with high frequency of gaming, but was also the participant who gained most from pre to post), and potentially strategy development.

Interleaving Game Mechanics, Strategy, and Learning

Another important finding is the impact of game mechanics on learning. For example, Violet initially began with an efficient strategy, but was often confused by the game mechanic that required visiting all burrows before submitting a correct answer. Even though he sometimes began with a deliberate and correct strategy, when he failed to
visit all burrows, his answer was judged as incorrect leading him to doubt his original strategy. Educational games that allow multiple strategies to lead to success should carefully consider their game mechanics’ interactions with player strategies and how those impact learning. It is important to support all valid strategies and give proper feedback at failure points, which may vary based on strategy.

**Metrics for Success**

Another issue is developing appropriate metrics for success. As seen in previous playtrace research (Andersen, Gulwani, and Popovic, 2013), trace length (number of actions) was insufficient to determine difficulty (or player success) of levels. In GrACE, some players successfully completed levels in as few actions as possible whereas others adopted a more exploratory strategy and would successfully complete levels using more actions than required. Neither of these strategies coincided with a more successful approach as they both successfully completed the levels.

Additionally, players could be equally efficient but have playtraces clustered separately. This was seen on Level 6 between Orange, Yellow, Indigo, and Brown. Only two were grouped with Prim’s algorithm even though they all had identical efficiency and trace length. Although our original intent was to allow players to find their own algorithm since many exist for solving the MST problem, it became evident upon playtrace analysis that an additional measure of success (e.g., number of actions taken) is required during gameplay to encourage a more algorithmic approach than guess-and-check. In our game, the measure of success for efficiency—Scout’s energy—was not reinforced enough since we did not want to limit players to only one algorithm. Because of this, players would hardly ever run out of energy even with an iterative testing strategy. We encourage other algorithmic thinking game designers to implement similar measures of success to ensure a player must be algorithmic to be successful. In fact, this approach can be extended to any game where designers want players to exhibit a particular behavior.
Chapter 5

Eliciting the Skill Chain of a Game

In order to create automated analysis tools for designers to understand game content from the perspective of players with varying levels of concept mastery, we must define a generic approach to list and organize the skills required of a game. As outlined in Chapter 1, games are complex activities requiring aptitude in numerous interrelated skills. Skill chains (Section 2.1) define, organize, and visually present these skills allowing for a better understanding of the required skills in a game. Unfortunately, no formal method exists for the extraction and creation of skill chains for existing games. In this chapter, we formalize the skill chain creation process by leveraging cognitive task analysis methods. By doing so, we answer our second research question outlined in Section 1.4: How can the skills necessary to complete a video game be extracted and organized by dependencies?

In Chapter 7, we demonstrate how to implement AIs based on skill chains produced from the following method in this chapter. In this manner, we define a novel skill-based method for the automated analysis of level progressions without requiring prior game data to build player models. This allows designers to analyze their game during the design and development phases as they make significant changes.

5.1 Instructional Design

In instructional design, cognitive task analysis (CTA) is a well-established family of methods to identify the skills and knowledge involved in a given task based on empirical observation and interviewing of experts (Crandall, Klein, and Hoffman, 2006; Clark et al., 2008). This includes methods for eliciting and modeling learning hierarchies (Wei and Salvendy, 2004; Jonassen, Tessmer, and Hannum, 1998), which makes CTA an ideal candidate for identifying the learning hierarchies or skill chains of games. Although CTA techniques have existed for decades, to our knowledge, they haven’t been adopted in games research and design for this purpose. Hence, this chapter presents an adapted cognitive task analysis method for extracting the skill chain of a game from empirical gameplay. Akin to prior work on method development (Khaled and Ingram, 2012), we conducted a case study in a mode of critical reflective practice (Schon, 1984), continually documenting and reflecting on our method design process to assess its utility, identify future improvements, as well as surface more general issues in eliciting the knowledge and skills involved in playing a particular game.

1Work presented in this chapter was previously published in the 2017 Proceedings of the Annual Symposium on Computer-Human Interaction in Play (Horn, Cooper, and Deterding, 2017).
The next section reviews existing work in games research related to modeling and identifying the component skills of games and introduces CTA. We then lay out our adapted CTA method, including its underlying rationale and a concrete step-by-step procedure for interested users. Our case study – using the method to identify the skill chain of the human computation game Paradox – illustrates the method in use and provides material for emerging observations and challenges. We discuss the contribution and limitations of the presented work and derive ramifications for future research.

5.2 Methods for Atom Identification

Both skill chain mapping and RLD require means to elicit the actual skill atoms a game consists of. Cook (2007), Echeverría et al. (2012) and Deterding (2015) are notably silent about how they arrived at the skill atoms and chains they discuss. Where they mention the underlying process, it essentially bottoms out in “expert evaluation”. This is a common issue of formal game analysis methods: most are some form of expert interpretation whose content and quality hinges on the unvalidated and tacit expertise of the reviewer. Guidance only concerns the format of the presented result, not the review process, leading to low replicability (Lankoski and Björk, 2015). RLD (McEntee, 2012; McMillan, 2013) similarly provides no method how to initially identify the atoms of a game and its parameters. Only once designers prototype actual levels based on hypothesized atoms and parameters does RLD loop in playtesting to assess the actual difficulty of each level as a player fail rate. From there, RLD focuses on iteratively understanding and tweaking the impact of atom parameters (enemy speed and distance) on difficulty. It offers no similar process to identify the game atoms themselves.

Existing playtesting and game user research methods are likewise of little help. No matter if based on player self-report, observation, psycho-physiological measures, game telemetry, or a mixture thereof, they revolve around capturing constructs of player experience (like flow or immersion) and how game features affect these (Koeffel et al., 2010b; Desurvire and Seif El-Nasr, 2013; El-Nasr, Drachen, and Canossa, 2013; Choi et al., 2016; Kivikangas et al., 2011). Closest to our concerns are heuristic analyses of game approachability – how easy a game is to learn (Desurvire and Wiberg, 2015) – and methods to balance game difficulty (Hunicke, 2005). Yet again, both revolve around approachability or difficulty as aggregate results, not the underlying required skills. For example, Linehan et al. (2014) charted difficulty curves for four popular puzzle games by coding Let’s Play videos for the number of actions required to solve a given level and when the game required a new skill. This provides an aggregate measure of difficulty and a description of the sequence in which skills are introduced by the game, but not the learning sequence in which these build on each other and should be introduced to players. The same holds for applied game design (Nacke, Drachen, and Gobel, 2010; Wouters, Spek, and Oostendorp, 2009). A number of recent methods support the capture of a given game’s game and learning mechanics, but chart them in the actual game design sequence, not the ideal learning sequence (Arnab et al., 2015; Carvalho et al., 2015).

A final source of potential methods is recent work merging intelligent tutoring systems with educational games. Butler et al. (2015) for instance present a system for automatic game progression design for a game teaching fractions. The system models the algorithm required to solve all possible basic fraction problems, generates a large number of game levels, assesses each level’s complexity on the number and kind of
involved solution features (substrings of the total algorithm required to solve it), and serves players levels matching their measured mastery of solution features. While promising, this approach by definition only works for skills that are easily formalized into an algorithm, and offers no means of empirically identifying what skills a game requires and therefore needs to formalize. Harpstead and Aleven (2015) use empirical learning curve analysis, a performance data analysis method from intelligent tutoring systems, to evaluate how well hypothesized models of player skills predict player success in an educational game. While this method does help assess whether there are hidden, non-modeled skills, again, it provides no means to empirically develop initial models.

5.3 Cognitive Task Analysis

Faced with the same question—how to identify the skills involved in a domain—instructional design has developed a cluster of methods called Cognitive Task Analysis (CTA). CTA involves a variety of interview, observation, and modeling techniques to elicit and describe the knowledge and skills experts use to solve complex tasks (Cooke, 1994). CTA is the currently prevalent method for determining how people solve complex problems and for eliciting their learning hierarchies, forming the bedrock of any instructional design (Jonassen, Tessmer, and Hannum, 1998). Recent systematic reviews suggest that basing instruction on CTA has strong positive effects on learning outcomes (Tofel-Grehl and Feldon, 2013).

That said, there is no one single CTA. With over 100 CTA techniques available (Cooke, 1994), choosing an appropriate method is challenging. A review by Wei and Salvendy (2004) distinguishes four families of CTA methods and derive guidelines when to apply which: (1) more informal observations and interviews are advisable when the domain in question is very broad, ill-defined, or ill-understood; (2) more rigorous process tracing captures the actual steps and involved knowledge and skills of performing a given task through think-aloud or stimulated recall techniques, and is advised when exemplary tasks are easily identified; (3) conceptual techniques generate structured representations of domain concepts and their relations and are used to either analyze and represent data collected through other techniques, or when the domain in question mainly involves conceptual knowledge; (4) computer simulations testing formal models are used when task models already exist and quantitative predictions or measures are required. Combining multiple techniques is generally recommended to reduce errors and improve validity.

No matter what technique, CTA generally involves an iterative five-step process (Clark et al., 2008):

1. Collect preliminary knowledge to identify learning goals, tasks and subjects: The analyst familiarizes themselves with the domain and desired learning outcomes to identify tasks to analyze and experts to recruit through e.g. document analysis, observation, or initial interviews.

2. Identify knowledge types: The analyst determines what kind of knowledge and skills the given tasks comprise and therefore, what specific elicitation, analysis and representation techniques are best suited (e.g., cooking a meal is a highly sequential task involving lots of tacit skills around preparation techniques, suggesting close observation and a flow chart as a representation).

3. Apply focused knowledge elicitation methods: The analyst uses the chosen techniques to elicit the knowledge and skills involved in the observed tasks. These
typically involve some form of verbal report by the expert to surface covert cognitive processes.

4. **Analyze and verify data**: The analyst codes the generated data following the chosen method and produces initial representations of the involved skills and knowledge. Data and representations are cross-checked with the involved experts for potential errors and unclear points, and compared and contrasted between multiple elicitations to arrive at a final, integrated model.

5. **Format results for intended application**: The analyst prepares a formal presentation of the resulting model depending on the intended purpose of the CTA.

## 5.4 Adapting CTA for Skill Chain Elicitation

Given the maturity of CTA as a means for eliciting the skills involved in a given task, we decided to develop an adapted CTA method to identify the skill chain of a game. We were especially encouraged in this as the skill atom model frames gameplay as a learning process of moving through an implicit learning hierarchy (Cook, 2007), and CTA is recommended specifically to identify learning hierarchies (Jonassen, Tessmer, and Hannum, 1998). For each of the five steps of CTA methods, we will first explain why we chose specific techniques and adaptations. We will then give idealized step-by-step instructions for our final procedure to allow easy replication.

### 5.4.1 Method Development and Rationale

1. **Collect preliminary knowledge to identify learning goals, tasks and subjects.** In the case of game analysis, domain and learning goals are determined by the game in question and what counts as successfully completing it. The task is naturally a stretch of gameplay, which should be long enough for players to demonstrate the skills in question without putting undue hardship on subject or analyst. Depending on the size of the game, analysts may therefore want to focus on a particular aspect or stretch of the game. For instance, to analyze end-game raiding in an online role-playing game, which players often only access after dozens or hundreds of hours of gameplay, it may be advisable to focus on the first half an hour of an exemplary raid.

   In terms of subject recruitment, most CTA techniques rely on subject matter experts. However, especially basic, low-level gameplay (like using controls) is a highly automated skill (Clark, Tanner-Smith, and Killingsworth, 2016) that experts are rarely able to consciously explicate. A proven technique for foregrounding these skills is comparing novice and expert performance (Seamster, Redding, and Kaempf, 2000). We conclude that unless the focus is on ‘high-end’ gameplay (such as end game raiding), recruiting a diverse set of expert and novice players of the game in question is a preferable strategy. As regards sample size, CTA gives no hard recommendations beyond recruiting more than one expert (Clark et al., 2008). Given most CTA techniques are qualitative suggests adopting sampling criteria from qualitative research paradigms, most notably theoretical saturation: data collection should cease when additional data doesn’t challenge the developed model anymore. Since a recent meta-analysis of qualitative interview methods suggest theoretical saturation is typically reached at around 12 or more participants (Guest, Bunce, and Johnson, 2006), we chose 12 participants as our lower bound.
2. Identify knowledge types: Playing any game is a well-defined task that usually involves complex problem-solving with a wide variety of required skills and knowledge types (Clark, Tanner-Smith, and Killingsworth, 2016), suggesting a process tracing technique (Wei and Salvendy, 2004). Given our particular interest in skills, we chose Seamster and colleagues’ (Seamster, Redding, and Kaempf, 2000) skill-based CTA (SBCTA) as a starting point. SBCTA combines a number of specific techniques to identify five types of cognitive skills that capture the range of skills required by video game play well: automated (e.g. hand-eye coordination), procedural (e.g. how to open menus), representational (mental models like predictions of enemy movement patterns), decision-making, and strategies. Wei and Salvendy (2004) and Seamster, Redding, and Kaempf (2000) suggest to elicit automated and procedural skills through process tracing combined with verbal reports such as think-aloud. However, gameplay is highly cognitively involving, making parallel think-aloud problematic (Hoonhout, 2008). We therefore chose to use stimulated recall, likewise a common process tracing technique in CTA (Crandall, Klein, and Hoffman, 2006). Here, the subject is video-recorded while performing the task in question. Afterwards, the analyst replays the video to the subject, stopping the video at relevant moments to ask the subject to explicate their thoughts and decision-making processes at the recorded time. This method allows the subject to perform tasks without interruption in a more natural setting while also cueing fresh memories and double-checking recall against actual recorded behavior, reducing false memories and post-rationalization (Dempsey, 2010). For these and other reasons, variants of stimulated recall have been in active use in game research for some time (Pitkanen, 2015; Borderie and Michinov, 2016; Kirschner and Williams, 2013; Kirschner and Williams, 2015). Following SBCTA, representational and decision-making skills are captured through the critical decision method (Klein, Calderwood, and MacGregor, 1989) and error analysis, focused interview probing of moments in task performance when subjects made decisions or errors. Finally, strategy skills are likewise elicited with structured interview probing on decision points and/or scenarios (Seamster, Redding, and Kaempf, 2000).

3. Apply focused knowledge elicitation methods: Each subject is video-recorded playing the gameplay stretch investigated. Since gameplay occurs both on and in front of the screen, both should be captured and merged into a single picture-in-picture or picture-next-to-picture video file for replay and analysis (Pitkanen, 2015; Stevens, Satwicz, and McCarthy, 2008). We decided to instruct players to think-aloud while they play to the extent possible, since think-aloud data provides additional cues and checks on the player’s memory during stimulated recall (Someren, Barnard, and Sandberg, 1994; Tjerdnsma, 1997). To elicit representational and decision-making skills via critical decisions and errors, the analyst watches the unfolding play and makes time-stamped notes on these incidents for focused follow-up. Indicators for relevant incidents are moments such as the player taking additional time to figure something out; struggling, pausing, or making errors; expressing an “aha” moment verbally or through body language; making a decision; or deviating from expected gameplay.

The play session is followed by a video-aided recall session that is also recorded. These generally follow a semi-structured interview pattern of initial scripted questions to elicit the subject’s thinking at a given point, followed up by further, more open probing (Lyle, 2010). Concretely, we decided to show the player the record of each point in gameplay marked by the interviewer, and ask them (a) what elements of the game they interacted with or paid attention to, (b) what they were thinking at this point, and (c) why they took the action they took. These questions try to elicit procedural and automated knowledge around low-level gameplay as well as
representational decision-making and strategy skills. Finally, subjects are asked what aspects of the game made it more or less difficult to complete the particular game goal at that point in order to identify the “challenge” component of the skill atom.

4. Analyze and verify data: Following standard procedures for stimulated recall analysis (Pitkanen, 2015), the recall session record is transcribed as a structured, time-coded script of (a) the recall dialogue and (b) recorded gameplay and think-aloud verbalizations it refers to. To conduct analysis and cross-check transcripts against video data, we suggest using a computer-aided qualitative data analysis (CAQDA) software that can code and display text and video data. As skill atoms already prescribe a clear unit of analysis, we adopted a directed qualitative content analysis method (“Three approaches to qualitative content analysis” 2005): each unit of the first transcript is parsed for any actions the player takes in the game at that point. These actions are then contextualized in the video record and transcript to assess whether it forms part of a skill atom, meaning it involves some simulation, game feedback, challenge, and player synthesis.

The simulation portion of a coded atom can be determined by observing audiovisual feedback indicating a game state change, or additional knowledge of the game itself. Feedback is determined by observing the audiovisual record of gameplay and analyzing a subject’s statements directly after an action is performed to see what feedback (if any) they noticed and (rightly or wrongly) interpreted as a result of their action. Challenge is explicitly derived from subject’s statements about what makes a given moment of gameplay hard or easy to master, and implicitly from moments of pausing or failures at performing a given action. Synthesis can be derived from moments where the player explicitly voices a particular “aha” moment or demonstrates newly competent performance of an action. Each instance showcasing all five elements is coded as a skill atom. A second pass through the transcript codes for further instances of the identified skill atoms or their components, e.g. additional dimensions of challenge.

Dependencies between atoms in the skill chain are discovered through (a) analyzing the transcript for the sequence in which players showed or reported to learn a given atom, and (b) subsequent logical challenging whether the observed sequence expresses a necessary dependency or not. After analyzing the first transcript, transcripts of additional subjects are coded for the already-identified and additional skill atoms, also revising or refining prior skill atoms as needed.

5. Format results for intended application: Wei and Salvendy (2004) recommend conceptual CTA techniques such as visual diagramming to articulate and present the structure of knowledge of a domain (Wei and Salvendy, 2004). Cook (2007) already provides a visual diagramming language for skill chains, which we chose to adopt. Interestingly, skill chains parallel the graphical structure of concepts maps, a common conceptual tool for diagramming results of a CTA (Novak and Cañas, 2008): both are constructed of nodes representing a specific concept and edges connecting nodes that represent their relationships. We took this as convergent support for our choice. We programmed a script to automatically generate a visualization from a simple JSON file. True to the iterative nature of CTA (Clark et al., 2008), we found it useful to already sketch and iteratively revise and refine a draft diagram skill chain in parallel to data analysis.

5.4.2 Method Procedure

The following is a streamlined set of instructions for replicating the final methodology of our adapted CTA.
1. **Identify analysis goals, tasks and subjects.** Determine which game and particular aspect of its gameplay you wish to map as a skill chain. Choose a portion of gameplay that requires players to learn and/or demonstrate mastery of the focused aspect and does not overburden subjects – assume that interview sessions last at least double the time of the recorded gameplay stretch plus 20 minutes of briefing and debriefing. Unless you focus on a particular audience or gameplay aspect (e.g. end-game content), recruit a diverse pool of 12+ subjects that involves both novices and experts at the game.2

2. **Elicit knowledge.** Instruct subjects to play the selected stretch of gameplay, verbalizing what is going through their head as they do so. During gameplay, audio-visually record both on-screen game events and off-screen player activity and take notes including time stamps on critical moments when players (a) seem to make a decision; (b) struggle, pause, or make an error; (c) express an “aha” moment; or (d) deviate from expected gameplay. After the play session, replay the video recording to the subject. Fast forward to and play each critical moment you noted and ask the subject to verbalize (a) what game elements they were paying attention to or interacting with, (b) what was going through their mind at that point, (c) why they took the action they took, and (d) what aspects of the game made it more or less difficult to complete the particular game goal at that point.

3. **Analyze data.** Transcribe all stimulated recall sessions with time codes, noting (a) the recall dialogue and (b) recorded gameplay and think-aloud verbalizations it refers to. Upload video data and transcript to a CAQDA software that can display and code both. For analysis, parse each unit of the first transcript for actions the player takes. Contextualize each action in the video recording and transcript to determine whether it forms part of a skill atom comprising:

- an action,
- simulation or rule processing and game state change, based on recorded game screen feedback,
- game feedback, based on recorded game feedback and subject statements directly after an action indicating whether they (in)correctly perceived a game state change as feedback on their action,
- dimensions of challenge, based on subject statements about what makes a given moment of gameplay hard or easy, as well as play pauses or failures at performing a given action,
- moments of synthesis where the player voices insight into or demonstrates competent enactment of some required knowledge or skill connected to the action.

Code each instance showcasing all five elements as a skill atom and label it based on the main synthesis knowledge or skill. Cross-validate player-derived simulation with the actual game rules, code, and/or game designer to ensure these aren’t player misconceptions. In a second pass, code the transcript for further instances of the identified skill atoms or their components. After identifying skill atoms, parse the transcript for dependencies between atoms expressed in when and/or order players showed or reported to learn a given atom. Challenge each derived dependency by questioning whether the documented order is an incidental result of the game’s design, or a logically necessary dependency. After analyzing the first transcript, code

---

2Our results indicate a smaller $n$ of 5+ may be sufficient, see below.
transcripts of additional subjects for already-identified and additional skill atoms, iteratively revising or refining prior skill atoms and re-coding prior transcripts as needed.

4. Visualize skill chain. Already during data analysis, draft a first skill atom list and skill chain diagram as a reference and cross-check for coding. Once all transcripts are analyzed and have informed the draft skill chain, draw up a final clean skill chain diagram using the provided script.

5.5 Case Study

We developed and tested the above CTA method by eliciting the skill chain of the human computation game (HCG) Paradox (Figure 5.1). This was part of a larger project aimed at developing automatic level progression algorithms for HCGs that crowdsource scientific tasks like classifying images of galaxies. Notably, HCGs suffer from poor player retention at least partially due to poor progression design: instead of sequencing tasks in an order matching the learning curve of players, they predominantly serve tasks at random, risking both player frustration and boredom (Sarkar et al., 2017). To inform machine learning algorithms that would automatically assess the difficulty of each task in Paradox, we wanted to get a grounded understanding of what component skills are required to play the game and thus, how difficult each Paradox task would be, depending on what skills it required (akin to (Butler et al., 2015)). Paradox was designed to crowdsource software verification, checking how many given conditions a given piece of code could satisfy. Each level or task presents players with a visualization of an underlying code piece as a graph of variables (displayed as nodes that can take different states) and conditions pertaining to variables (visualized as edges). Players can manually click individual variables to change their states or use various “brushes” to select subsets of variables to be modified. Different brushes trigger different approaches to modifying variables, from simple brushes that immediately set all to a certain value, to brushes that run specific optimization algorithms. The player’s goal is to configure variables so that the highest possible number of conditions are satisfied. To complete a level, a player must reach a given target score or percentage of satisfied conditions. In general, it is not known in advance whether the target percentage of conditions (let alone all conditions) of a level can be satisfied.

5.5.1 Procedure, Observations and Reflections

In the following, we report on how we concretely implemented our method step by step and what generally relevant observations we made for that step.

Identify analysis goals, tasks and subjects. To make sure subjects were exposed to the same gameplay, we used a stable local version of Paradox that featured seven tutorial levels introducing gameplay, followed by a fixed series of 20 challenge levels, which were generally larger, more open-ended and more difficult than the tutorials. Levels were chosen to cover a range of level sizes and likely solution strategies. Tutorial levels were gated: players had to complete a level by reaching its target score before being able to proceed to the next tutorial level. In contrast, players were able to skip challenge levels without completing them if they desired. We asked participants to play the game for 30 minutes, immediately followed by a 30 minute stimulated
recall session. Gameplay length was determined by estimating how long players typically take to get through the tutorial and five challenge levels, which we assumed sufficient for novice players to acquire and demonstrate basic gameplay skills and for expert players to be challenged in the breadth of their expertise.

To record at least 12 subjects and ensure theoretical saturation (Guest, Bunce, and Johnson, 2006), we recruited 15 subjects, preparing for a number of no-shows. We recruited 5 “expert” subjects who had played Paradox extensively in the past and 10 “novice” players who had never seen Paradox before, otherwise aiming for maximum diversity in gender, age, and socio-economic background.

Observations and Reflections. Novices are more valuable than experts. Interestingly, novices proved much more valuable for discovering low-level interface and gameplay skills than expert players. Indeed, analyzing expert gameplay only added minor refinements to the emerging skill chain. This somewhat contradicts standard CTA philosophy to rely on experts, but may be at least partially due to the relatively simple gameplay of Paradox or the fact that expert traces were analyzed last.

Quick saturation. We identified the vast majority of skill atoms during analysis of the first five recall transcripts, with subsequent transcripts adding only about one additional skill atom (3 percent of all codes) each. This suggests that future analyses may work sufficiently with a smaller number of subjects than we used – although this has to be tested with larger, more complex games.

Recognize and bracket shortcuts. At the conclusion of the first three recall sessions, we noticed that players heavily relied on the so-called optimizer brush. This brush automatically maximized satisfied conditions in a given graph area. While generating a good first score, the global maximum of possible satisfied conditions cannot usually
achieved with the optimizer; it requires deeper analysis and probing of the total graph. However, since the optimizer brush was introduced early on in the tutorial and was enough to complete early tutorial levels, novices tended to learn the heuristic to simply use the brush to clear each level, rather than learning how the constraint satisfaction mechanic worked and how to manually analyze and manipulate the graph. Hence, they would often become frustrated in later challenge levels when the brush alone didn’t suffice, and were not able to switch to manual optimization. (One participant even said that the optimizer felt “like cheating” because it would do all the work without players understanding how.) In terms of CTA, this highlights that the availability of “power tools,” “exploits,” or “shortcuts” as part of the analyzed task can prevent certain procedural skills from being actively performed and thus made observable. Observed tasks should therefore ideally be trialled in advance of actual analysis to check for and eliminate undesired shortcuts. In our case, we later on manipulated the game and restricted two novices and one expert from using the optimizer brush at all, requiring them to manipulate each variable of a level individually. This helped discover particular skill atoms for novices as well as challenge features of graph layouts we hadn’t observed before.

**Elicit knowledge and analyze data.** We began stimulated recall sessions with novice players as we assumed that their play would feature many critical moments foregrounding basic *Paradox* skills which expert players had already perfected and would therefore be hard to notice. Gameplay sessions were captured using *Morae*\(^5\), which allows to make categorized notes during screen and camera recording that are logged on the recording timeline. We used this to log critical moments we would then replay to subjects to stimulate recall after the play session. Stimulated recall sessions were recorded with *Camtasia Studio*\(^6\), as this program allowed to display and screen capture play session video and audio in addition to the camera video and audio of analyst and subject conversing. Stimulated recall sessions were transcribed and coded using *MaxQDA*\(^7\).

**Observations and Reflections.** *Skill dependencies are unclear and confounded by level design.* We found it hard to identify clear dependencies between skill atoms and to disentangle (a) the order in which the game’s progression design required certain skills, (b) the order in which players developed insights, and (c) the ideal learning

---

\(^5\)https://www.techsmith.com/morae-features.html

\(^6\)https://www.techsmith.com/camtasia.html

\(^7\)http://www.maxqda.com/
hierarchy in which both should occur. Instead, the order in which the game tutorial introduced skill atoms strongly shaped player perceptions and analyst coding: both tended to state that the order in which the game presents skill atoms is the order in which players learned them. This indicates a strong limitation in using qualitative analysis of fixed games with fixed progressions for eliciting skill chains. Ideally, players would be exposed to a randomized multivariate ordering of skill atoms to observe which order empirically produces the fastest learning gains.

**Strategy skills are unvalidated.** In alignment with Seamster and colleagues’ (Seamster, Redding, and Kaempf, 2000) hierarchy of cognitive skills, we found strategies to sit at the ‘highest’ level of our skill chain. The distinction between procedural and strategy skills is fuzzy. We found a good indicator for strategy skills to be that players consciously identified different approaches or composite applications of procedural skills and chose between them based on context. For example, players chose from what location to begin solving Paradox levels and in which order to move through the graph depending on level geometry. Sometimes a player would work from the periphery to the center, other times a player would choose to start in an area that had the most conflicts, and players would generally verbalize that and why they chose this particular strategy. That said, it is hard to tell from our data whether and how optimal any of these strategies in a particular context actually are. (Notably, the same holds for CTA, which simply assumes that expert practice is self-validated as best practice.) It would be good to triangulate our qualitative data with quantitative data on the relative performance of different strategies, in the same way players and teams in eSports analyze the performance of different characters or items.

Also, conceptual CTA techniques focus on mapping the cognitive skills and knowledge of a task rather than individual strategies, meta-strategies, and conditions when to employ them (Wei and Salvendy, 2004). To a certain extent, one could argue that the three strategy skill atoms we mapped are really on the skill atom “choosing optimal strategies”. Hence, CTA may be less apt at analyzing and visualizing strategies and strategy-heavy games.

**Visualize skill chain.** A detail of the final skill chain we created for Paradox can be seen in Figure 5.2. The full skill chain is given as supplementary information. To assess the produced skill chain, we asked one of the designers of Paradox to draw a skill chain based off of their understanding of the skills necessary for the game, which can be seen in Figure 5.3.

---

8See e.g. https://www.dotabuff.com/heroes/winning
Observations and Reflections. Skill chain analysis surfaces low-level and pre-existing skills. While both designer and CTA-derived skill chain covered the same basic mechanics, the CTA-generated chain is far more detailed and comprehensive. First, it entails many required pre-existing skills. For instance, we discovered that the game was not accessible for people with blue-yellow and green-red color blindness, who had difficulty recognizing the color-coded states of the variables. Second, comparing novice and expert gameplay brought to light that experts used certain low-level skills that were not explicitly conveyed to players in the tutorial and therefore not used by novice players. One example is changing brush sizes. Because expert players had explored Paradox and its interface more deeply, they had discovered how to change the brush size, which allowed them more control over the variables selected. This observation arguably demonstrates the most direct value of our method: the tutorial, based on the designers’ hypothetical skill chain, overlooked parts of the actual empirical skill chain of players – quite possibly since designers are expert players who therefore have difficulty recognizing the low-level, highly automated skills they possess but novices don’t. That said, it is an open question whether the same insights could not be generated more efficiently through standard usability and playability testing.

Skill chains run together in a core mechanic. Interestingly, unlike the designer-generated skill chain, the CTA-generated chain eventually runs together in one central node, “efficiently reduce number of conflicts”, which then branches out into strategies for achieving such efficient reduction. Discussions with the game’s designers and our own gameplay experience suggest that this central node is indeed the “core mechanic” of Paradox (Sicart, 2008). We take this as further validation of our method and find it suggestive for formal game analysis more broadly: core mechanics or loops are the graph-theoretically most central nodes in which all dependencies and subskills run together.

Skill chains remain flat. Overall, the skill chain we elicited had a flat, “pancake” quality: it consists of many fundamental skills around using the interface without many dependencies between or beyond them. The same flat structure can be seen in Cook’s skill chain of Tetris (Cook, 2007), while the skill chain of Pac-Man shows greater depth (firecorth, 2013). This may be due to many things: the relative simplicity of Paradox and Tetris compared to a greater gameplay depth of Pac-Man; the subjectivity of involved analysts; or the general complexity of the underlying genre. It is worth noting that all published uses of skill atoms cover relatively simple puzzle and action games. Hence, it is an open question whether (a) different game genres and more complex games would produce different skill chains, and whether (b) skill chain mapping is feasible or productive for more complex games or whether graphs become too unwieldy to be of much use.

5.6 Discussion

Reflecting on our case study, we think it warrants the conclusion that our method worked: we were able to elicit a skill chain from gameplay that roughly mapped the understanding of one of the game’s designers, identified the correct “core loop”, and produced useful design insight. Particularly, it surfaced a range of overlooked prerequisite and low-level skills that had eluded the designer’s attention and made the game more challenging to learn and play for novices. A second main observation vis-à-vis CTA is that observing novices learning how to play a game proved potentially more valuable than observing smooth expert performance.
That said, our case study also surfaced a series of major challenges and limitations. First, it is unclear whether standard usability and playability testing methods wouldn’t be more efficient in producing the same insights into overlooked low-level gameplay skills. Although our case study suggests that skill chains can be elicited with a smaller \( (5+) \) \( n \) than we used, the method remains quite involved, using about 10 hours per subject (1 interview, 6 transcription, 3 analysis). Likewise, it is unclear how approachable our method is for designers and game researchers. In future work, we would therefore like to run method variants with other analysts to assess ease of use, approachability, perceived efficacy, and outcomes, and to trial the method with smaller \( n \)'s and with direct video analysis that doesn’t rely on transcription.

Now standard playability and usability testing don’t provide learning hierarchies to inform e.g. level design or procedural content adaptation or generation. However, this arguable main differentiator of our method – identifying the dependencies between skills – also proved the most elusive. Actual ideal dependencies were hard to ascertain, and the resultant skill chain featured little depth. We assume this is partially due to the simplicity of Paradox and the fact that the game’s actual fixed progression sequence strongly biased observation. In future work, we would therefore want to use process tracing on experimental variations of skill sequencing, and replicate our method with more complex games, which brings us to a further limitation: We tested the method with a very simple puzzle game. It is unclear whether it would work with different genres or more complex games. Action games rich in automated ‘twitch’ skills may prove harder to analyze, and ‘big’ games like the MMORPG EVE Online may involve so many skill atoms that eliciting them in short gameplay sessions or mapping them in a single chain could prove unwieldy.

A final challenge and limitation concerns strategy skills. While we could elicit a number of emergent strategies actively used by players, our method cannot speak to how empirically optimal these strategies actually are. Here, combinations of our qualitative analysis and quantitative game analytics would be useful. Indeed, we view novel mixed method combinations of qualitative ‘thick data’ methods like CTA with ‘big data’ analytics to be a highly promising future direction, as many data-driven methods of tutorial or progression design and analysis (Thompson et al., 2013; Butler et al., 2015; Harpstead et al., 2015b) in turn lack exactly the specificity and insight into what particular skills to analyze for or generate from that qualitative data provides.
Chapter 6

Skill-Based Level Analysis

A critical component of an engaging game is delivering content to players with incrementally increasing challenges (Koster, 2013). In puzzle games in particular, the idea is that early levels provide the foundation players need to develop skills and strategies that will help them solve later levels (Schell, 2014). However, a significant challenge in designing games lies in assessing the difficulty of individual levels and whether the level progression promotes learning the goals of the game (Koster, 2013; Butler et al., 2015; Butler and Banerjee, 2014; Kumar, 2012; Alonzo and Gotwals, 2012).

This chapter details the initial design process behind our framework for an automated analysis of level progressions. The expansion of this framework to a more generalized approach involving a Monte-Carlo based method is further outlined in 7. In combination, these methods answer our third research question as stated in 1.4: How can the logical organization of video game levels be analyzed by required skills in order to promote learning without relying on existing player data?

6.1 Introduction

Often game designers have an intuitive idea of an ideal difficulty progression and test these ideas by recruiting players to play their games. Based on player feedback, designers then iteratively refine their levels and progressions until satisfied (Salen and Zimmerman, 2004). While a positive aspect of iterative testing is that it provides complete control to the designers, the process is resource intensive and intractable when levels are procedurally generated rather than hand-designed. Furthermore, relying on surveys or informal feedback provides only a post-play report on player performance, which may overlook the intricacy of how player strategies develop over time.

One way to capture player strategies is by logging all player actions and then automatically assessing difficulty progressions by computationally exploring the data (Butler and Banerjee, 2014; Butler et al., 2015; Holmgård et al., 2014; Nielsen et al., 2015). Some approaches to automatically assess player strategies compare human playtraces with those of AI-based bots (Holmgård et al., 2014; Smith et al., 2012; Nielsen et al., 2015). However, because most games are complex and can be solved through a number of different strategies, AI-based solutions are difficult to design (Jaffe et al., 2012). Often these approaches focus on ways players can play the game rather than on whether the particular strategy is showing evidence of learning. For example, Holmgård et al. (2014) design AI players to capture different player goals rather than their competence in developing strategies whereas Smith

1Work presented in this chapter was previously published at the Foundation of Digital Games conference in 2017 (Horn et al., 2017).
et al. (2012) focus on how different game elements affect difficulty while assuming perfect playing skill. An exception is Nielsen et al. (2015) who explores AI agents with varying intelligence on generated and human-designed games to determine the difference in agent performance. Agents performed worse on human-designed games, however results do not show performance differences within games on individual levels making it challenging to understand how players would progress through each game. Therefore, it is a significant challenge to model AI-bots that reflect sub-optimal strategies that help analyze if a level progression promotes learning.

To address this challenge, in this chapter we contribute to the automatic assessment of level progressions using AI-based bots with a different approach, by analyzing which optimal and sub-optimal strategies can complete a series of levels with AI-based bots, called “StrataBots.” The aim is to assess how player strategies perform throughout gameplay as level difficulty changes. We assess this gradient bot fitting approach by exploring gameplay in GrACE, a restricted-world puzzle-based educational game that encourages players to learn a strategy that on replay could also solve earlier levels (Chapter 3). By starting with simple puzzles, players are allowed to layer specific sub-strategies to discover the optimal strategy. We model each substrategy with one of four StrataBots. Each StrataBot represents one of the strata (i.e., layers of strategy) in a learning curve or skill chain and models a particular stage of the designer-specified learning process of the player from basic to advanced strategies. These bots allow for the simultaneous evaluation of a level’s difficulty according to many possible strategies players could employ as well as the evaluation of how players progress through the stages of learning to successfully solve the puzzles. This method alleviates the necessity for large amounts of existing data to train models and avoids the introduction of human players at each testing phase.

To evaluate whether the StrataBots reflect how players perceive each level, we compare the StrataBot data with existing player data from a past pilot study of GrACE (Chapter 3). Players spent 30 minutes playing GrACE and encountered a variety of puzzles. StrataBots then played each puzzle encountered by a player and we classified the puzzles based on which strategy is able to successfully complete it. Results indicate higher failure rates on puzzles that could only be solved through the more advanced strategies, thereby validating the StrataBot gradient. We also evaluate individual player progressions to determine how variations in the presentation order of each puzzle classification affects player behavior.

With this chapter, we contribute a method for analyzing the difficulty progression of a puzzle game through automated evaluation. We aim to capture a more nuanced definition of difficulty, cast in terms of the strategies or “complete set of instructions” for how to play a game (Lantz et al., 2017) players must learn and master in order to complete specific puzzles. Our approach is particularly useful for procedurally generated puzzles, since determining puzzle difficulty cannot be performed by hand and there are many potential progressions for a player to take through the game. Although we present our work in the context of an explicitly educational game, where there is a strong need for understanding player progression in order to assess the effectiveness of the game, this work is also relevant in non-educational game contexts.

The next section introduces relevant background information, while Section 6.3 discusses GrACE and how puzzles are procedurally generated in the game. Section 6.4 fully describes the implementation of each StrataBot and Section 6.5 describes an experiment performed to rate the difficulty of each level based on which StrataBots can solve it. The results, discussion, and future work follow.
6.2 Existing Approaches to Difficulty Assessment

A traditional approach to assessing difficulty progressions in games is through player-oriented testing, a method of developing games that focuses on iteratively improving them based on player-feedback (Isbister and Schaffer, 2008; Kim et al., 2008). Often, developers begin by implementing a prototype of a game, and then have many people play it to evaluate how the game is likely to be experienced. The designer then considers whether the goals of the game are being realized through play, and redesigns aspects of it based on data gathered from the iterative design and testing procedures. While this process helps fine-tune the difficulty progression and provides ultimate control to the designer, it requires significant resources from the designer and the test-players.

One way to alleviate the demand for time from the designer is by first determining what exactly the player is expected to learn through play, and then automatically analyzing the effectiveness of different progressions. Often these follow a model similar to Elaboration Theory by Reigeluth et al. (1980) that argue the simplest form of a task should be introduced first, then more complex tasks which build on the first. This concept, encapsulated in skill atoms game design theory (Cook, 2007), provides a framework for describing the challenges and skills that are being mastered in the game over time. In formalizing skill atoms, Cook (2007) describes a model comprising of action (taken by the player), simulation (by the game), feedback (from the game to the player as a result of simulation) and modeling (updating user’s mental model as a consequence of feedback received). Deterding (2013) uses this model as an approach to define player’s challenges as they pursue their goals in the game. This approach provided a method called the “lens of skill atoms”, which allows the perception of any interactivity from the point of view of game design by concentrating on the users’ goal (Deterding, 2012). A problem with this approach is that the model typically puts analysis of the game systems at the forefront, rather than taking a more player-centric approach. Therefore, this approach helps to analyze (and design) the underlying game system but does not focus on how players interact with that system. Our StrataBots, in contrast, are an idealized interpretation of how players of various skill levels engage with the game.

In terms of evaluating how players interact with the system, Linehan et al. (2014) qualitatively analyzed the order in which skills are introduced in four existing puzzle games. They extracted the order and method of novel skill introduction in each game and found that solution length increased until a new skill is introduced; at which point the levels return to short traces and build back up to longer traces using the newly acquired skill. Each main skill is introduced separately and are introduced through simple puzzles that require only the basic performance of the new skill. After a skill is introduced, the player is given ample opportunity to practice the skill and combine it with other previously learned skills. Puzzles that use the player’s existing skills increase in complexity until a new skill is introduced.

Harpstead and Aleven (2015) attempt to quantify player strategy evaluation by applying skill atoms to the empirical analysis of difficulty progressions in order to predict player success in levels. This research used fundamentals of intelligent tutoring systems to analyze how well hypothesized models of required skills fit collected player data. Each predictive model was trained on player data and built on varying numbers of knowledge components or procedural skills. Overall this method reasonably predicts player success. Levels with higher error rates were easier for players and could be solved using a rote strategy (not one of their original models) were easier for the players. However, this method does require extensive player data.
and is not conducive to procedurally generated levels as there will not be enough player data for each level. In addition, while the method tries to predict success and is useful for evaluating existing difficulty progressions, it does not indicate which strategies are usable on each level.

Regarding procedurally generated levels, player models have been developed as a means of evaluating them in the context of maze-based games (Liapis et al., 2015). The original implementation of these models relies on discrete and separate personas (i.e., coin collector, monster killer, speed runner, etc.). These “Procedural Personas” attempt to model player goals and strategies as an evolvable controller composed of linear perceptrons which allows it to be used as critics of newly created puzzles. Our implementation builds upon this idea and orders personas into a hierarchy that represents a player at various points of the learning curve. Additionally, they evaluated procedural personas through repeated fixed-number trials. There are multiple ways a persona may traverse a level if at any point, more than one equally good opportunity arises. As more and more of these situations present themselves, there is a lower likelihood that a fixed trial count approach will find all possible paths. We rectify this situation by exhaustively producing all possible playtraces each persona could create.

Not only is it useful to analyze existing progressions, but it is convenient during the design process to have tools that help craft the introduction of new skills. Andersen, Gulwani, and Popovic (2013) run an algorithm (e.g., addition) and log the order different steps are executed producing a solution trace with the skills necessary to complete a problem. The introduction of new steps can be done in a smooth fashion and situations where this does not occur can be highlighted. The algorithms we present in this chapter focus on strategies rather than individual game mechanics which allows us to analyze level progressions after players master each mechanic. Also, the algorithms used by Andersen, Gulwani, and Popovic (2013) are deterministic so do not have situations where the algorithm must make a choice between two equally good options and follow each to completion. This is an important distinction because two solutions to a problem may have very different traces and the comparison of difficulty between multiple puzzles with this property becomes ambiguous. It is unclear if puzzles should be compared using the simplest solution, the most complicated, or some sort of hybrid method.

Similarly, Butler et al. (2015) created a mixed-initiative tool that automatically ensures a difficulty progression is followed by verifying a particular level feature is used or unused in the solution and that the introduction of new features is done at an appropriate pace. This tool was produced for a puzzle game called Refraction where players learn mathematical fractions by splitting a laser beam. Level features include using laser bending or splitting pieces, leaving some laser beams unused, and laser beams being unavoidably crossed. Our approach is similar but rather than looking at particular solution features and whether or not they are required to find a solution, we take a strategy-centric approach to determine available strategies that can be successfully applied to find the correct solution of a puzzle. In further research, Butler and Banerjee (2014) turn to visualizing progressions to help analyze and evaluate them. They show “ideal” progressions produced by humans or a progression generation tool. We adopt this approach to create user-specific progressions that help us understand how players are affected by various orderings of puzzles with different difficulty properties.
6.3 GrACE

Level difficulties in this chapter are analyzed from a puzzle game called GrACE. Puzzles are designed around the computer science concept of finding the Minimum Spanning Tree (MST) of a graph, described below. Also described are the design and mechanics of GrACE and how puzzles are created in the game. Section 6.5 explores a method for assessing difficulty of these puzzles.

6.3.1 Graphs and Minimum Spanning Trees

Graphs in computer science are abstract data types that represent objects (i.e., nodes) and the relationships between them (i.e., edges) shown in Figure 4.1(a). Edges can be directed, which means that the relationship goes specifically from one node to the next or undirected, implying a bidirectional relationship between nodes. Edges can also be weighted, where the weight represents some cost between the objects. Puzzles in GrACE are represented as undirected graphs, where each edge is weighted by a positive integer. A spanning tree of a graph is a set of edges such that each node is connected to the other nodes exactly one (i.e., no cycles: A-B, B-C, C-A). While there can be multiple spanning trees of a graph, a minimum spanning tree (MST) is a spanning tree such that the sum of edge weights contained in it are equal to or less than all other spanning trees. Players successfully complete a puzzle only if they find a minimum spanning tree.

6.3.2 Game Design

StrataBots are designed to reflect human strategies to solve puzzles in GrACE, which was originally created to help players implicitly learn an algorithm for finding an MST. Each puzzle in GrACE is a different graph with weighted edges. Nodes are represented as burrows that contain a vegetable and are connected to other burrows by edges whose weights indicate the cost necessary to move from one burrow to another.

Edges in the puzzle are traveled by a player-controlled mouse named Scout, who expends an amount of effort equivalent to the edge weight traveling between two nodes. Scout’s goal is to identify the least-cost or least-effort way to dig up all the vegetables in a puzzle. The path should be selected such that each node is connected exactly once with the least amount of combined weight, equivalent to solving the MST.

Once the player has performed a series of actions indicating the edges that he or she believes comprise the MST, the player then clicks the submit button to check whether the solution is correct. With successful solutions, the player continues to the next puzzle and again tries to find its MST. Players with incorrect solutions can either tweak their previously submitted solution, reset the current puzzle, or request a new puzzle of the same difficulty.

6.3.3 Level Generator

Because puzzle difficulty was originally conceived primarily as its number of nodes and edges, puzzles in GrACE are generated using the number of nodes and edges of a graph as input parameters. Puzzles are procedurally generated with a constraint satisfaction paradigm in an answer set programming package known as clingo

\[^2\text{http://potassco.sourceforge.net/}\]
(Gebser et al., 2016). While some constraints are defined based on the aesthetic and mechanics of the game, only the number of nodes and edges are variable inputs.

Initially, 100 puzzles were generated for each set of input parameters to create a difficulty progression from puzzles with two nodes and one edge to those with nine nodes and 16 edges arranged in the game as eleven difficulty-stages. By increasing the number of nodes and edges in a puzzle, the graph becomes increasingly visually complex, thus requiring the player to choose more edges correctly. This study explores the 368 unique puzzles played by students in a study presented in Chapter 3.

### 6.4 Computationally Modeling Strategy

Considering strategy as the “complete set of instructions” for how to play a game (Lantz et al., 2017), each StrataBot is a set of instructions operationalized as a well-defined algorithm. Algorithmically analyzing games offers several benefits to studying human playtraces including that algorithms are repeatable, static, testable, fully examinable and possibly more diverse than existing human strategies (Lantz et al., 2017). Unlike human players that may become distracted by individual features of a level, algorithms can make consistent decisions at any point in a level allowing for reliable analysis across diverse levels.

While consistent, sometimes these choices are arbitrary. For instance, when the algorithm dictates travel along the lowest cost edge but two such choices are available, an algorithm may choose one at random. However, because algorithms can be run on a puzzle exhaustively, after enough runs if an algorithm can solve the puzzle, it will. The hierarchical StrataBots who execute increasingly complex computational models of strategy are described in the following section.

#### 6.4.1 StrataBot Design

We designed StrataBots by first determining the set of content or skills that players should master, and then what it means to master them. Mastery in GrACE is defined as the player finding an MST (section 6.3.1) in a minimal amount of steps for an arbitrarily large and complex graph. Similar to Prim’s algorithm (Prim, 1957) that finds the MST for any connected graph, players start with knowledge of which node they are on and the values of the edges connected to it. Like Prim’s, the player selects a lowest cost edge to travel to a new node, and then discovers new connections for that node. Both the player and Prim’s are expected to only travel along the lowest cost edges that it has encountered until each node is connected exactly once. While Chapter 3 propose that introducing players to an increasingly complex set of graphs is sufficient for players to discover Prim’s algorithm, 4 suggest that more scaffolding is necessary for many players to independently discover this algorithm.

StrataBots are a series of bots that solve puzzles based on the three main algorithms the designers expect players to incrementally master by the end of the game. The most complex of these algorithms is an optimal bot called PrimsBot, which is a game-specific implementation of Prim’s algorithm (Prim, 1957). PrimsBot is guaranteed to make globally-focused decisions that eventually lead to the correct answer. Another globally-conscious bot is SearchBot, which performs game-specific variations of two common search algorithms for graphs: breadth-first search (BFS) and depth-first search (DFS). This bot performs a breadth-first and depth-first search through the puzzle while ignoring the cost of any particular edge. It is important to note that unlike PrimsBot, SearchBot is not guaranteed to find a correct answer.
Finally, two greedy bots called LocalBot and BacktrackBot both make locally greedy choices, but differ in their behavior when no choices are available at the current node. When no other choices are available BacktrackBot can move backward one node to a previously visited node and continue playing the puzzle, while LocalBot ceases playing regardless of the completeness of the current solution.

Each StrataBot attempts to complete all of the 368 unique puzzles by applying its own strategy. At each game state the StrataBot makes decisions with respect to its internal algorithms, resulting in a strategy-specific game-tree that represents its entire range of playtraces. Because there may be multiple possible correct answers, it is important that each StrataBot explores this space exhaustively. For example, a greedy bot can potentially encounter two equally low-cost edges; the StrataBot must explore both selections because one choice may eventually result in a correct solution while the other may not. While a bot may find many different paths that complete a puzzle, if even a single play-through results in a successful solution, it will be found.

6.4.2 StrataBot Descriptions

The following section describes each StrataBot and how the strategy-specific game trees are created for them.

**PrimsBot** PrimsBot implements a slight modification of Prim’s algorithm (Prim, 1957). In contrast to the traditional Prim’s algorithm, game mechanics require PrimsBot to consider its current location. Like Prim’s, PrimsBot begins by setting the label of the node at the player’s current location to connected. Throughout the playthrough, PrimsBot maintains a set of connected nodes. At each iteration of the algorithm, the bot chooses the lowest cost edge that spans from a connected node to an unconnected node and updates the unconnected node to connected by adding it to the set of connected nodes. Once all nodes are discovered and connected, the bot submits its answer, which is guaranteed to be an MST. When multiple edge selections are equally good at a given game step, the algorithm runs each to completion to determine if either will result in a successfully completed puzzle.

**SearchBot** Two iconic iterative search algorithms for searching a graph are breadth-first search and depth-first search. These search algorithms combine to form SearchBot, who runs both of these algorithms. BFS and DFS are almost identical in implementation, but BFS stores nodes in a queue while DFS stores nodes in a stack, affecting the order of the nodes visited. Both BFS and DFS also include a set of visited nodes to prevent re-visiting the same node. In the context of GrACE, these algorithms are modified to flag the edge they just traveled along whenever visiting a new node.

**LocalBot and BacktrackBot** At each step, these locally greedy bots choose the lowest cost local edge that connects the already connected nodes to an unconnected node. If there are no unconnected nodes adjacent to the current node, BacktrackBot returns to the previously visited node and again chooses the lowest cost edge that connects that node to an unconnected node whereas LocalBot halts and submits its current solution. If BacktrackBot needs to backtrack, LocalBot will submit an incorrect solution. After backtracking one step, if no unconnected edges exist in the puzzle, BacktrackBot stops and fails to complete the level because the bot would have to make non-local decisions otherwise. This feature means that any puzzle solvable by LocalBot is also solvable by BacktrackBot. In cases where the locally greedy choice is always the globally greedy choice, this bot behaves exactly like PrimsBot. Also, it is worth noting that LocalBot and BacktrackBot produce solutions that are a subset of a DFS algorithm which means any level that LocalBot or BacktrackBot can complete successfully, SearchBot also can complete successfully.
Chapter 6. Skill-Based Level Analysis

6.4.3 Puzzle Classifications

To determine the difficulty of puzzles in GrACE, each is classified by the StrataBots who can successfully solve them. For instance, if a puzzle is only solvable by PrimsBot it is classified by the label PrimsOnly. However, if a puzzle is solvable by both PrimsBot and SearchBot, its label is PrimsSearch. Recall from the bot descriptions that LocalBot produces solutions that are a subset of BacktrackBot, and BacktrackBot produces solutions that are a subset of SearchBot. So while it may seem that there are many combinations of bots for solving the puzzles, there are actually only four classifications: PrimsOnly, PrimsSearch, PrimsSearchBacktrack, and AllBots.

All levels are solved by PrimsBot, but only some are exclusively solved by it. Some are solvable by only PrimsBot and SearchBot and labelled PrimsSearch. PrimsSearchBacktrack indicates that only PrimsBot, SearchBot and BacktrackBot can solve the puzzle, while AllBots is solvable by all four.

Figure 6.1 shows four puzzles originally classified as difficulty-stage five by the designers (out of eleven). The puzzle in 6.1(a) is only solvable by PrimsBot because locally-greedy solutions (BacktrackBot and LocalBot) necessarily choose the edge from Node 3 to Node 1 first, then either the edge connecting Node 1 to Node 0 or Node 1 to Node 2. From there, BacktrackBot and LocalBot make a non-optimal choice to connect Node 0 to Node 2 since it is the only edge locally available that connects an already connected node to an unconnected node. While SearchBot can perform both depth-first and bread-first search, neither will find the correct solution. With

Figure 6.1: Puzzles from Difficulty-Stage Five. Four puzzles are shown from difficulty-stage five of the original GrACE implementation, chosen because they are each classified with different StrataBot labels. Players begin at node 3 on each of these puzzles.
the BFS strategy, SearchBot first chooses the edges connecting Node 3 to Node 1 and then from Node 3 to Node 2 (an incorrect choice). No more choices will be explored by SearchBot’s BFS strategy because the incorrect choice renders a correct solution impossible. The closest to correct DFS strategy chooses edges connecting Node 3 to Node 1 then Node 1 to Node 0 followed by Node 0 to Node 2. Therefore it is only PrimsBot who can find the correct solution (edges connecting Node 3 to Node 1, Node 1 to Node 2, and Node 1 to Node 0).

The puzzle in figure 6.1(b) is solvable by PrimsBot and SearchBot. Both LocalBot and BacktrackBot will either choose the edge connecting Node 3 to Node 1 first or the edge connecting Node 3 to Node 0 first. Either choice forces the bot to make a locally greedy yet globally sub-optimal choice next. In the case the bot chooses the edge connecting Node 3 to 1, the next choice is connecting Node 1 to 0. If the bots first choose the edge connecting Node 3 to Node 0, the next edge choice could connect from Node 0 to either Node 1 or Node 2. Connecting Node 0 to Node 2 is so far correct. At this point, however, LocalBot and BacktrackBot cannot make anymore local choices from Node 2 so LocalBot fails. BacktrackBot can return to the previous node (Node 0) and make the only locally greedy choice available which is to connect Node 0 to Node 1 (an incorrect choice). SearchBot, on the other hand, can successfully complete this puzzle by performing a breadth-first strategy. The bot first chooses the edge connecting Node 3 to Node 1, then Node 3 to Node 0, and finally Node 3 to Node 2.

LocalBot is the only bot that cannot solve the third puzzle shown in figure 6.1(c). SearchBot finds an MST through DFS by choosing the edge from Node 3 to Node 0, then either the edge from Node 0 to Node 1 or from Node 0 to Node 2. Because there are no deeper nodes to find, the bot returns to Node 0 and then chooses the other edge. This strategy is coincidentally identical to a locally greedy strategy that requires backtracking so BacktrackBot can also solve this puzzle.

The final puzzle shown in figure 6.1(d) is solvable by all StrataBots because any bot can find the solution that begins by choosing the edge connecting Node 3 to Node 2, then choosing the edge from Node 2 to Node 0, and finally choosing the edge from Node 0 to Node 1.

6.5 Experiments

The experiment in this chapter is designed to test whether StrataBots accurately capture the difficulty that humans face when solving puzzles in GrACE. First, it is hypothesized that certain classes of puzzles will pose more difficulty to players than others. Then, we suggest an ordering of difficulty based on how difficult we think the strategies will be for players to develop. Puzzle difficulty is then tested against human playtraces to determine whether the proposed ordering reflects puzzle difficulty in practice.

First from a given set of puzzles, each is classified and labeled based on the particular StrataBots that can solve it described in section 6.4.3. Then, the four puzzle classifications are ranked by what the designers think are the complexity of the strategies required to solve them.

Puzzles only solvable by bots that make choices based on global information (i.e., PrimsBot and SearchBot) are expected to pose the most challenge for players as it requires them to make decisions based on more information than if only local decisions were being made. An important distinction between the two bots that make global decisions is that SearchBot selects edges based only on the order of the nodes
Chapter 6. Skill-Based Level Analysis

82

it sees and the connections between them. PrimsBot, on the other hand, evaluates edge weights while it is making global traversal choices. Therefore, puzzles labeled PrimsOnly (i.e., only PrimsBot can solve) are expected to pose the most difficulty for players. These puzzles require the player to make global decisions with respect to the weights of the puzzle. Because the SearchBot like PrimsOnly makes global choices, it is expected to be the next most difficult type of puzzle for players to solve.

Puzzles solvable by bots that can only make local decisions (i.e., LocalBot and BacktrackBot) are expected to be the easiest for players. In contrast to PrimsBot and SearchBot, the locally greedy bots only select the edges to which they are connected. Unlike SearchBot, both select the edge with the least cost. In contrast to LocalBot, BacktrackBot can revisit the most recently visited node to continue the algorithm if it reaches a dead-end. The expected result is that puzzles that can be completed by BacktrackBot but not LocalBot will be more challenging for players, because it forces players to recognize a problem and take a step to fix it. Ultimately difficulty classification is first predicted to be based on whether players will need to make global versus local traversal choices. Global strategies are expected to be more difficult for human players to make.

To explore whether the ordering of these labels matches difficulty players face when solving puzzles, data is analyzed from a previous pilot study where 42 middle school students played GrACE during a STEM-based summer camp, their ages ranging between 10 and 13 years old (\( M = 11.9, \ SD = 0.85 \)). Twenty students played a version of GrACE where they all received the same puzzle for each difficulty-stage, while twenty-two students played unique levels. Those who played unique levels also were given the ability to generate new puzzles when the current puzzle proved particularly challenging, but they could also generate a new puzzle for practicing a puzzle of similar difficulty. While players individually encounter a limited number of the puzzles in gameplay, they collectively played a total of 368 puzzles in the pilot study. The pilot study was originally designed to test the effects of collaboration and procedurally generated levels on player learning 3, however this chapter is intended to gain further insight into why some players performed better than others.

Players who could generate new puzzles are referred to as those in the procedurally generated content condition (PCG-condition), while the others are referred to as the static condition. For every puzzle, each student’s playtrace was logged and includes the puzzle layout, all player actions, correctness of the solution, and whether the level is retried or a new one is generated. The number of successful and unsuccessful attempts for each puzzle in each StrataBot classification are then aggregated and sorted by difficulty-stage. Unsuccessful attempts contain more than failed submissions. We also include instances where players request a newly generated puzzle prior to completing their current puzzle. Finally, the StrataBot classifications are ranked based on the success rate across all difficulty-stages.

6.6 Results

Results in Table 6.1 show the eleven stages of difficulty GrACE players encounter and how puzzles from these difficulty-stages are classified by the StrataBots. Puzzles were first sorted by difficulty-stage and then classified as one of four categories based on the StrataBots that find MST solution. Of the 368 puzzles, 20.9% are classified as PrimsOnly, 18.8% as PrimsSearch, 12.0% as PrimsSearchBacktrack, and 48.4% as AllBots. The PrimsOnly classification only applies to puzzles above difficulty-stage
Chapter 6. Skill-Based Level Analysis

Stage | Number of PrimsOnly Puzzles (%) | Number of PrimsSearch Puzzles (%) | Number of PrimsSearchBacktrack Puzzles (%) | Number of AllBots Puzzles (%) | Total
--- | --- | --- | --- | --- | ---
1 | 0 (0%) | 0 (0%) | 0 (0%) | 34 (100%) | 34
2 | 0 (0%) | 0 (0%) | 14 (43.8%) | 18 (56.2%) | 32
3 | 0 (0%) | 5 (16.1%) | 0 (0%) | 26 (83.9%) | 31
4 | 0 (0%) | 7 (21.9%) | 8 (25.0%) | 17 (53.1%) | 32
5 | 7 (25.0%) | 7 (25.0%) | 1 (3.6%) | 13 (46.4%) | 28
6 | 18 (38.3%) | 13 (27.7%) | 6 (12.8%) | 10 (21.3%) | 47
7 | 11 (26.2%) | 13 (31.0%) | 6 (14.3%) | 12 (28.6%) | 42
8 | 9 (25.7%) | 7 (20.0%) | 5 (14.3%) | 14 (40.0%) | 35
9 | 14 (38.9%) | 1 (2.8%) | 0 (0.0%) | 21 (58.3%) | 36
10 | 12 (31.6%) | 14 (36.8%) | 3 (7.9%) | 9 (23.7%) | 38
11 | 6 (46.2%) | 2 (15.4%) | 1 (7.7%) | 4 (30.8%) | 13
Total | 77 (20.9%) | 69 (18.8%) | 44 (12.0%) | 178 (48.4%) | 368

Table 6.1: StrataBot Classification per Puzzle Difficulty-Stage. All 368 puzzles played by the students from the study are classified by the StrataBots. The particular stage of the puzzle is shown in the leftmost column while the total number of puzzles that fit a particular classification is shown in the rightmost column. In the middle are the number of puzzles in the particular difficulty-stage that StrataBots of given types can solve. The percentage of puzzles that each solves is in parentheses.

five, showing that earlier levels are always possible to solve by multiple, often simpler strategies.

In support of the hypothesis, players find PrimsOnly puzzles the most difficult with a success rate of 13.0%. The next most difficult puzzles are PrimsSearch at 16.8% and PrimsSearchBacktrack at 19.3%. Finally, there is a big jump in success rate for AllBots levels with player success rate increasing to 38.0%. The full attempt counts and success rates broken down by condition are in Table 6.2.

Interestingly, a Friedman test shows a statistically significant difference in overall player success rate by puzzle classification, $\chi^2(3) = 9.699, p = .021$. To determine what made the difference, post hoc analysis with Wilcoxon signed-rank tests was conducted with a Bonferroni correction applied, resulting in a significance level set at $p < 0.008$. From this analysis it is not clear that any pairings show significant difference, although the AllBots-PrimsOnly pairing ($Z = -2.401, p = .016$) and the AllBots-PrimsBacktrackSearch pairing ($Z = -2.490, p = .013$) are nearing significance. In total, these results indicate that players are challenged when encouraged to devise a more globally oriented strategy.

However, the static condition shows PrimsSearch puzzles have a surprisingly high success rate at 46.2% making it appear as though puzzles with this classification are the easiest to complete. However, this anomaly can partially be explained by the low number of overall attempts on puzzles with this classification. These players played only one PrimsSearch puzzle, and because it was the last puzzle, fewer players had the opportunity to try it. Furthermore, those that are able to attempt the puzzle are probably performing better than those who drop out earlier considering that puzzle difficulties must be solved in-order.

While some of the challenge that players encounter can be explained by the
### Chapter 6. Skill-Based Level Analysis

<table>
<thead>
<tr>
<th>PCG Condition</th>
<th>Successful Submits</th>
<th>Failed Submits</th>
<th>Success Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimsOnly</td>
<td>42</td>
<td>335</td>
<td>11.1%</td>
</tr>
<tr>
<td>PrimsSearch</td>
<td>40</td>
<td>267</td>
<td>13.0%</td>
</tr>
<tr>
<td>PrimsSearchBacktrack</td>
<td>37</td>
<td>120</td>
<td>23.6%</td>
</tr>
<tr>
<td>AllBots</td>
<td>156</td>
<td>295</td>
<td>34.6%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Static Condition</th>
<th>Successful Submits</th>
<th>Failed Submits</th>
<th>Success Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimsOnly</td>
<td>54</td>
<td>308</td>
<td>14.9%</td>
</tr>
<tr>
<td>PrimsSearch</td>
<td>18</td>
<td>21</td>
<td>46.2%</td>
</tr>
<tr>
<td>PrimsSearchBacktrack</td>
<td>67</td>
<td>314</td>
<td>17.6%</td>
</tr>
<tr>
<td>AllBots</td>
<td>197</td>
<td>282</td>
<td>41.1%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Total</th>
<th>Successful Submits</th>
<th>Failed Submits</th>
<th>Success Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimsOnly</td>
<td>96</td>
<td>643</td>
<td>13.0%</td>
</tr>
<tr>
<td>PrimsSearch</td>
<td>58</td>
<td>288</td>
<td>16.8%</td>
</tr>
<tr>
<td>PrimsSearchBacktrack</td>
<td>104</td>
<td>434</td>
<td>19.3%</td>
</tr>
<tr>
<td>AllBots</td>
<td>353</td>
<td>577</td>
<td>38.0%</td>
</tr>
</tbody>
</table>

**Table 6.2: Success Rates of Human Players by StrataBot Classification.** The player rates of success on puzzles of different StrataBot classifications are split into those played by people in the PCG and Static conditions and totaled at the bottom. Overall, puzzles classified as PrimsOnly presented the most challenge for players.

Increasing numbers of nodes and edges in each successive difficulty-stage, it may not be the only indicator of challenge. Perhaps because players may need new strategies to solve puzzles with new StrataBot classifications, success rates indicate that some challenges players experience are correlated with whether the player has previously encountered a puzzle requiring particular StrataBot classification. Each time players encounter a puzzle classified by a new type of StrataBot, success rates drop. For example, the AllBots classification is first encountered on difficulty-stage 1 where players have a 72.7% success rate. PrimsSearchBacktrack is seen as early as difficulty-stage 2 and player success rates drop to 48.5%. PrimsSearch is introduced on difficulty-stage 3 and the success rate drops to 25.0%. PrimsOnly is first encountered on difficulty-stage 5 where players have a 20.0% success rate.

Although the sample size is only 164 total attempts compared to the overall puzzle attempts of 2553, results indicate players found puzzles classified as PrimsOnly and PrimsSearchBacktrack the most difficult with a 20.0% success rate (40 attempts and 5 attempts respectively), followed by PrimsSearch with a 28.6% success rate (14 attempts) and puzzles classified as AllBots the easiest with a 46.7% (117 attempts). The low numbers of attempts for PrimsSearchBacktrack is because only two players encountered this type of puzzle on difficulty-stage 5. One player completed the puzzle after three attempts and the other failed the first attempt and generated a new puzzle.
Figure 6.2: Player Progressions by StrataBot Classification and Success. Each puzzle attempt is represented by the x-axis while the particular difficulty-stage the puzzle is in is represented on the y. StrataBot classifications are represented by colors as shown in the legend, while success or failure on a puzzle is indicated by the ‘X’ and circle symbols respectively.
6.6.1 Player-Specific Results

Interestingly, it is not until about half-way through the game at difficulty-stage 5 that any player can possibly encounter a PrimsOnly puzzle. In the PCG-condition players may never be forced to solve a PrimsOnly puzzle as puzzles solvable by all StrataBots are presented to players from difficulty-stage 1 through 11. Therefore it is possible for a player to complete the entirety of the game without ever seeing a puzzle that would encourage them to develop more sophisticated strategies. The majority (58.3%) of puzzles played on difficulty-stage nine, only two puzzles from the end of the game, are classified as AllBots.

When a player in the PCG-condition solves a puzzle unsuccessfully, they have the option to play a new procedurally generated puzzle with a similar number of nodes and edges. The hope was that seeing puzzles with diverse layouts would help cement the game concepts through repetitious yet varied play. However, for some players this feature helped them advance difficulty-stages by asking the game for less challenging puzzles. For example, Figure 6.2(a) shows the StrataBot classifications of each level played by Player 198 and whether or not the puzzle is solved successfully. Unsuccessful attempts are indicated by an ‘X’ while successful attempts are indicated by a circle and the particular classification is indicated by color. On difficulty-stages seven through ten, Player 198 attempts a variety of puzzles, but the only ones completed successfully are classified as AllBots. Counter to what was hoped through the design of GRACE, players were not always encouraged to refine their strategies toward a Prim’s-like solution. Instead, after completing two PrimsOnly puzzles the player completes lower difficulty puzzles until the end of the game.

The reverse situation also occurs. Players play a PrimsOnly puzzle after exclusively playing AllBots puzzles. When players encounter the PrimsOnly puzzle the significant increase in challenge causes many players to get stuck and frustrated. Prior results showing difficulty-stage four as the first major hurdle for players in the static condition are corroborated by the classification results revealing this as the first puzzle where players are abruptly required to advance to an optimal strategy (Chapter 4).

Player 105 is presented with a relatively good progression the first time through the game. They are given four graphs of the easiest difficulty in the first five levels (the only other one is Level 2 which is 3 nodes, 2 edges, players starts in the middle so they have to use backtracking). After these five levels, the player is presented with a slightly more difficult level with the classification PrimsBacktrackingSearch. The player completes this on the first attempt, moves to a PrimsSearch level and completes it on their first attempt again. At this point, the player struggles on a slightly more complex puzzle of the same PrimsSearch classification. After four attempts, the player completes the puzzle and moves to a PrimsOnly level. This level takes even more attempts (six) before the player solves it. Levels 10 and 11 are PrimsSearch and PrimsOnly levels which the player solves with little trouble. The second and third time through the game, Player 105 is served with slightly easier levels in the later half of the game. As one would expect of a player who has completed multiple highly complex levels requiring a strategy comparable to PrimsBot, the easier levels presented in difficulties 7 through 11 are completed with little challenge. This trend continues on the third time through the game when the player completes every puzzle on the first attempt.

Player 106 is presented with a difficulty progression that never includes a puzzle where PrimsBot is the only StrataBot which can solve it. This means that the player
Chapter 6. Skill-Based Level Analysis

6.7 Discussion

Results from the experiment are promising, suggesting that player success is in part based on the StrataBot classifications presented in this chapter. However, it is curious why some players solved puzzles of a particular classification yet later struggled with those of the same class. One idea is based on Elaboration Theory (Reigeluth et al., 1980). It may be that the complex tasks are not appropriately introduced though simpler tasks. In addition, perhaps GrACE players need exposure to the same level classification multiple times before mastery is assumed.

Another and more likely reason is that players appear to have mistakenly mastered a particular StrataBot strategy. Facilitated by the design features in GrACE such as the button for generating new puzzles, players can complete a puzzle with a guess-and-check strategy by simply asking for a new puzzle when the old strategy stops working. Occasionally, this strategy may on the surface appear to be one captured by the StrataBots, but ultimately only illustrates that a player is having difficulty developing a more efficient strategy. It would then make sense that a player continued to struggle with this type of difficulty level later in the game. By developing more StratBots in the future, it could be possible to correctly classify the example strategy as guess-and-check rather than belonging to a StrataBot class. Developing more StrataBots can potentially facilitate the discovery of new strategies.

To fully explore the similarities between a player’s strategy and a StrataBot’s, we also propose an edit distance based approach as outlined in previous research (Holmgård et al., 2014; Osborn and Mateas, 2014). This method allows the analysis of players based on how much their playtrace differs from a given playtrace (e.g., an optimal bot trace).

Puzzles analyzed for this research had discrete actions and were relatively small making the exhaustive production of playtraces possible. This method is not feasible for every game, including ones with continuous actions or excessively long solution traces. We update this method in the following chapter to accommodate some of these cases. In combination with edit distance, we suggest a sliding window approach based on the current game state for a player instead of start-to-end playtrace generation. This would require reframing what success means for a given window but would enable the ability to estimate how closely a player is following a given strategy and for how long during lengthy play sessions where exhaustive playtrace generation is infeasible.
Chapter 7

A Monte-Carlo Approach to Skill-Based Automated Playtesting

In Chapter 6, we answered our final research question: *How can the logical organization of video game levels be analyzed by required skills in order to promote learning without relying on existing player data?* However, this method was specific to GrACE and required laborious hand-authored StrataBots which are not easily implemented in other games. To address this issue in furtherance of our final research question, this chapter describes and implements a method for StrataBot creation that utilizes Monte-Carlo Tree Search—a general game-playing algorithm—allowing for a substantial reduction in development time for additional StrataBots.

7.1 Generalizing StrataBots with MCTS

Interactive digital games require numerous skills—implicitly and explicitly—from players (Thompson et al., 2013). Often these skills are seen as direct game mechanics such as jumping in a 2D platformer or steering in a driving simulator. However, games can require many additional skills that do not directly tie to game mechanics yet are critical to gameplay, such as map navigation in an MMORPG or multi-tasking and microing units in a real-time strategy game (Stroback, Frensch, and Schubert, 2012; Gee, 2003; Dye, Green, and Bavelier, 2009). To prevent frustration, level designers pay close attention to the skills they require from players during gameplay to make sure they do not introduce challenges the player cannot or should not complete yet, a concept concretely outlined in Rational Level Design (RLD) (McMillan, 2013; McEntee, 2012). Interestingly, traditional player models attempt to mirror player behavior without tying performance back to individual skills. New player modeling techniques have started incorporating expected skills or goals into their models in an effort to better reflect their human counterparts (Smith et al., 2011a; Liapis et al., 2015; Bakkes, Whiteson, and others, 2014; Harpstead and Aleven, 2015). While promising, it remains difficult for designers to use player models to understand how players learn and acquire skills as they progress through a game (Charles et al., 2005a).

Learning progressions guide how a player advances through a game, what they learn along the way, and the rate of difficulty increase—often dictating the experiential success of that game (Butler et al., 2015; Harpstead and Aleven, 2015; Andersen et al., 2012). By teaching and allowing the mastery of individual game skills through successively more difficult challenges, learning progressions lead a player from novice

---

1Work presented in this chapter was published and presented at the Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference in 2018 (Horn et al., 2018).
to expert (Deterding, 2015). Often, explicit guidance comes in the form of tutorials or help tips while implicit instruction stems from the careful design of content by giving the player increasingly difficult challenges where they must use particular skills (McMillan, 2013; Koster, 2004). Quantitative approaches develop metrics for difficulty (e.g. number of enemies or reaction time required), but this ignores player experience and focuses solely on the content itself, yet how the player goes through the level may change their experience, skills used and—ultimately—perceived difficulty.

In this chapter, we extend Stratabots to a more complex game environment known as human computation games (HCGs) and compare hand-authored implementations to a Monte Carlo based approach. HCGs leverage people’s problem solving skills where computational models fail to adequately perform a task on their own. We analyze the performance of Stratabots on a series of tutorial levels of the HCG Foldit to determine the extent to which introductory levels require the designer-intended skills. We compare hand-designed Stratabots to a Monte Carlo Evaluation approach using restricted actions to understand if we can decrease design time of Stratabots by leveraging existing general game-playing algorithms. We then compare these bots to human players to identify similarities in performance and efficiency. Our research contributes to AI research by demonstrating how skill chains can be integrated during design to create novice and expert AIs used for the analysis of levels. Additionally, we highlight the applicability of Stratabots to games that are not training players toward a specific known algorithm for solving levels, by analyzing the difficulty curve of a tutorial sequence in Foldit, indicating that even sub-optimal AIs lend useful information to level designers.

7.2 Related Approaches

Chaslot et al. (2008) state that game AI requires domain knowledge and a long development timeframe, but implementing a Monte Carlo based solution can reduce both of these. Monte Carlo solutions rely on simulated playouts to evaluate game moves rather than domain specific heuristics creating a flexible yet powerful technique. Successfully implemented for a range of single-player and adversarial games, including Go, Scrabble, Solitaire, and Settlers of Catan (Kocsis and Szepesvári, 2006; Browne et al., 2012; Chaslot et al., 2008), Monte Carlo implementations generally focus on creating optimal AIs. Varying the performance of a Monte Carlo based AI through decreasing the simulation depth or overall runtime allows players the opportunity to play against sub-optimal opponents (Zook, Harrison, and Riedl, 2015; Browne et al., 2012; Baba Satomi, Iwasaki, and Yokoo, 2011). However, players vary in skill by more than the time they think about a problem, players also differ in how well they grasp the mechanics of a game.

Restricted Play analysis (Keehl and Smith, 2018) lets designers see general trends in gameplay, alter game mechanics, and evaluate the effect those changes have on players. Keehl and Smith (2018) created a Unity tool to streamline this process along with a proof-of-concept and let designers analyze the effect of design changes on players with three distinct playstyles. This is the first MCTS solution we found varying playstyle through more than computational depth or runtime, though the playstyles are game specific and focus on when the simulated player performs a specific action (collecting a game piece) rather than what actions they perform.
Chapter 7. A Monte-Carlo Approach to Skill-Based Automated Playtesting

<table>
<thead>
<tr>
<th>Concept/Skill</th>
<th>Level</th>
<th>Concept/Skill</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clashes</td>
<td>1-1</td>
<td>Rubber Bands</td>
<td>3-2</td>
</tr>
<tr>
<td>Pulling Sidechains</td>
<td>1-1</td>
<td>Camera Translation</td>
<td>3-3</td>
</tr>
<tr>
<td>Camera Rotation</td>
<td>1-2</td>
<td>Rubber Bands (+)</td>
<td>3-3</td>
</tr>
<tr>
<td>Score</td>
<td>1-2</td>
<td>Freeze</td>
<td>3-4</td>
</tr>
<tr>
<td>Shake</td>
<td>1-3</td>
<td>Backbone Color</td>
<td>3-5</td>
</tr>
<tr>
<td>Pull Backbone</td>
<td>2-1</td>
<td>Remix</td>
<td>3-5</td>
</tr>
<tr>
<td>Undo</td>
<td>2-1</td>
<td>Hydrophobics</td>
<td>4-1</td>
</tr>
<tr>
<td>Voids</td>
<td>2-2</td>
<td>Exposeds</td>
<td>4-1</td>
</tr>
<tr>
<td>Reset</td>
<td>2-2</td>
<td>Tweak Rotate</td>
<td>4-2</td>
</tr>
<tr>
<td>Wiggle</td>
<td>2-3</td>
<td>Tweak Shift</td>
<td>4-3</td>
</tr>
<tr>
<td>Hydrogen Bonds</td>
<td>3-1</td>
<td>Tweak Rotate (+)</td>
<td>4-4</td>
</tr>
<tr>
<td>Wiggle (+)</td>
<td>3-1</td>
<td>Secondary Structure</td>
<td>4-5</td>
</tr>
</tbody>
</table>

Table 7.1: List of concepts taught in each of the first 16 tutorial levels of Foldit as outlined in Andersen et al. (2012). Some skills are not applicable to Stratabots since they are specific to human players such as translating and rotating the camera or color perception. Others require proxies since the mechanics provided by the Foldit scripting language do not match one-to-one with the concept (e.g., pulling sidechains). Repeated concepts are marked with a (+). We create each Stratabot from Foldit’s skill chain by combining one or more designer-specified skills along with all prerequisite skills.

7.3 Human Computation Games

Across domains including computer science (Sarkar et al., 2017; Dukes, 2013), biology (Barone et al., 2015; Lee et al., 2014), medicine (University of Oxford, 2014; Coburn, 2014), astronomy (Lintott et al., 2008) and psychology (Computing, 2015) to name a few, HCGs give players tools and mechanisms to perform gamified, real-world, domain-specific tasks that computers cannot computationally solve due to complexity or lack of data. Prevalent tasks in HCGs include data classification (e.g., image labeling or sentence transcription) and common sense activities such as identifying color differences (Computing, 2015) or image labeling (Ahn and Dabbish, 2004).

Due to the complexity of tasks and inability to computationally model solutions, HCG designers often don’t know the skills their game must teach or the appropriate order in which to teach them, resulting in poor player retention—perhaps due to poor learning progressions or insufficient tutorials (Andersen et al., 2012; Sarkar et al., 2017)—suggesting most players do not acquire the full suite of skills game designers intended (Sauermann and Franzoni, 2015). Without these skills, players are unable to meaningfully contribute to the scientific research contained within an HCG, limiting the power of that game.

7.4 Foldit

Foldit is an HCG where players compete and collaborate to fold and pack protein structures efficiently. Numerous biochemistry-specific mechanics exist within Foldit and gameplay is very different from traditional games, meaning players cannot rely on previous game experience to understand how to play (Andersen et al., 2012). Conversely, designers cannot assume extensive knowledge from players when designing the levels and user interface. This means designers must be careful when introducing content to new players. Players begin with a series of tutorial levels designed to teach the main game mechanics. Each tutorial level displays a protein on the screen that
is not yet folded well. Players must then decide which action(s) to perform on the protein, as well as where and how to perform them. As players alter the protein structure, their score gets higher if they improve the structure by lowering its energy. If players increase the energy of the protein structure, then their score drops. Once a score threshold has been met in a tutorial level, the player can continue playing the same level and attempt to improve their score or move on to the next level.

Foldit allows many interactions with proteins including moving, freezing, replacing and auto-organizing individual segments of a protein. Some operations can be done globally such as “Wiggle” which attempts to optimally situate the protein’s core and peripheral components in relation to one another. Others are done on specific segments and may not move the protein at all. For example, adding a band between two segments sets the attraction between them but does not have a visible effect until a corresponding move action is performed (e.g. Wiggle). Actions that auto-organize attempt to automatically (that is, Foldit does the computation rather than the player) position parts of the protein to find an optimal state. Players could perform the same actions manually by clicking and dragging each protein component but this would be tedious and time-consuming.

While multiplayer components exist in features such as leaderboards, online contests, and user-created puzzles, each playthrough of a tutorial level is completed independently. We use only the tutorial levels in this study, since their organization and design are meant to increase in difficulty and target specific skills. The Foldit tutorials are an initial linear sequence of 16 tutorial levels, after which the tutorial branches and introduces more advanced and specialized concepts. Table 7.1 shows a list of concepts taught in each of the first 16 levels. Recent iterations of Foldit add more tutorial levels, however these are not the focus of our research since they generally teach specific one-off concepts.

7.5 Study Methodology

The release of Foldit\textsuperscript{2} used in this work contains 36 tutorial levels—of which the initial 16 were used by previous studies with concepts logged by designers—followed by competitive online science puzzles where players vie for leaderboard positions specific to each puzzle. We focus our analysis on the first 16 tutorial levels, since these arguably teach the most important or commonly used mechanics. The early tutorial is divided into four sections each containing three to five levels. Also, we expect levels to increase in difficulty and have lower human success rates as players advance through the tutorial while gaining and mastering new skills.

In order to create Stratabots that play the tutorial levels, we first produce a game-specific skill chain for Foldit. Foldit designers and other game design researchers developed this skill chain through their experience with Foldit as well as ad hoc discussions and viewings with other players. Based on these experiences, we choose skills that either repeated throughout many tutorial levels, were designer specified by Andersen et al. (2012), or seemed crucial one-offs. Additionally, we create some Stratabots with a combination of skills that are not dependent upon each other to determine how the mastery of separate independent skills affects success. A list of all Stratabots as well as their included skills can be seen in Table 7.2; the hierarchical nature of the bots is represented in Figure 7.1. Every Stratabot includes the ability to understand their current score and goal score for a level, undo actions they have taken (in order for a bot to explore the search space), and the ability to select part or all of a

\textsuperscript{2}https://fold.it/
protein, though some can only select all and others can only select parts. In general, our Stratabots take greedy approaches to score improvement. When faced with a series of actions and parameters (e.g. degrees of rotation or number of iterations), they choose the action and set of parameters that will increase their score the most.

After selecting crucial combinations of skills from the skill chain and previous designer-stated intentions, we craft 11 Stratabots with the built-in Lua scripting language provided by Foldit (Khatib et al., 2011a) and run them on all 16 of the introductory puzzles, logging whether the bot can successfully complete a given puzzle or not.

As stated previously, crafting each Stratabot can be a time consuming process. To identify if modern general game-playing algorithms can alleviate some of this demand, we implement Monte Carlo evaluation (Chaslot et al., 2008) using Foldit’s Lua scripting API and compare its results with the hand-authored Stratabots. Similar to restricted play Monte Carlo Tree Search (Keehl and Smith, 2018), we restrict the available game actions to those only a particular Stratabot can use for the entirety of one Foldit playthrough. We repeat this process for each Stratabot to get a set of Monte Carlo simulations that mirror each Stratabot. Due to restrictions in Foldit’s Lua interface (e.g. memory and limited save game slots), we cap simulations at 500,000 nodes and must re-generate the nodes after each action is taken with the new game state as the root node.

We complete this research by analyzing which bots could complete each level to determine if the existing learning progression roughly follows the success rates of Stratabots. Additionally, we analyze if the skills present in the successful Stratabots mirrors those expected by the designers as outlined in Table 7.1. To corroborate results, we include success rates from human players.

### 7.6 Results

Throughout the following section, we refer to Stratabots by the title entry in Table 7.3. For specific skills of each bot, see Table 7.2. Results in Table 7.3 show the breakdown of the 16 introductory Foldit levels and the success or failure of each Stratabot on that puzzle. Additionally, levels are broken down into four sections corresponding to the menu layout in Foldit. During gameplay, only certain actions are available to players based on the level they are playing whereas the Foldit scripting API allows all
operations on every level. This potentially means Stratabots are more powerful than their human counterparts and we will indicate this throughout the remainder of the section where applicable. Finally, player performance is shown in Table 7.4.

**Levels** Perhaps unsurprisingly, every level could be solved by one or more Stratabot. This shows the range of bots we created sufficiently cover the introductory puzzles indicating that one bot is at least as complex as the designers expect for each level. Additionally, we found no differences in levels completed between manually-created Stratabots and their Monte Carlo equivalent indicating a possible reduction in Stratabot design time without loss of power.

Overall, 3 of 16 levels require the one specific skill that the level is intended to teach. Levels 3-3, 3-4, 3-5 and 4-4 were completed by only one Stratabot. Bot7 which corresponds to the Band skill completed levels 3-3 and 3-4, though 3-4 is targeting Freeze. Bot9 completed level 3-5 which includes the Remix skill that no other bots possess, and only Bot11 completed level 4-4 which includes the unique Tweak skill. These roughly reflect the skills that designers want players to learn for those individual levels. Although Tweak is introduced prior to Level 4-4, this is the level where players must really understand the Tweak tool in order to complete the puzzle.

Only Bot4 could not complete every level in the first tutorial section. *Foldit* designers allow players to perform only certain actions in the first section by disabling others in the game interface. In Level 1-1, players must manually click and drag parts of the protein rather than using the corresponding button introduced in subsequent levels. The Stratabots have no such restrictions in the Lua interface, but the ability for Stratabots with very few skills to complete this level indicates that it is still a good starting point in the game’s progression.

With some exceptions, results are consistent with previous research into learning progressions in educational and instructional games (Linehan et al., 2014). Each tutorial section generally begins with an easier puzzle followed by more challenging puzzles where the player has a chance to hone their mastery of the newly presented skills. The Stratabots show *Foldit* roughly follows this model even though the bots are potentially more powerful than their human counterparts on many levels. The

### Table 7.2: Table indicating the set of skills modeled by each Stratabot.

Skills present in a Stratabot are represented by a •. Some Stratabots incorporate all the skills of other bots plus additional skills making them categorically superior when playing *Foldit*.
Table 7.3: Results of each Stratabot playing the introductory levels of Foldit. If a bot can solve a level, it is indicated by a •. Levels solvable only by more complex bots indicate that the levels may require additional skills present in some complex Stratabots but not others. For the skill makeup of each bot, see Table 7.2. Horizontal lines separate the four sections of tutorial levels. In general, each section begins with an easier level as indicated by more successful Stratabots, and gradually gets more difficult until the next section where it starts easy again. We found no differences between the hand-authored Stratabots and their Monte Carlo counterparts.

Surprisingly, Bot2 is able to complete most levels (12 of 16) even though it is one of the more simple bots. This is probably due to the power available through the scripting tool but not to players during tutorial gameplay (players have this power later in the game). For example, Bot2 can be very selective in what parts of the protein it wiggles while players are only given the option to wiggle everything at once. However, this still illustrates the importance of the Wiggle tool to the manipulation of proteins in Foldit. Mastering this one skill has tremendous benefit for players throughout the rest of their gameplay experience.

Player Data To add context to the results presented so far, we also present player
Chapter 7. A Monte-Carlo Approach to Skill-Based Automated Playtesting

<table>
<thead>
<tr>
<th>Level</th>
<th>Total Attempts</th>
<th>Successful Attempts</th>
<th>Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>8784</td>
<td>8534</td>
<td>97.2%</td>
</tr>
<tr>
<td>1-2</td>
<td>8363</td>
<td>8196</td>
<td>98.0%</td>
</tr>
<tr>
<td>1-3</td>
<td>8111</td>
<td>7818</td>
<td>94.4%</td>
</tr>
<tr>
<td>2-1</td>
<td>7725</td>
<td>7523</td>
<td>97.4%</td>
</tr>
<tr>
<td>2-2</td>
<td>7481</td>
<td>6575</td>
<td>87.9%</td>
</tr>
<tr>
<td>2-3</td>
<td>6513</td>
<td>6296</td>
<td>96.7%</td>
</tr>
<tr>
<td>3-1</td>
<td>6316</td>
<td>5960</td>
<td>94.4%</td>
</tr>
<tr>
<td>3-2</td>
<td>5950</td>
<td>5542</td>
<td>93.1%</td>
</tr>
<tr>
<td>3-3</td>
<td>5435</td>
<td>4571</td>
<td>84.1%</td>
</tr>
<tr>
<td>3-4</td>
<td>4631</td>
<td>3982</td>
<td>86.0%</td>
</tr>
<tr>
<td>3-5</td>
<td>3930</td>
<td>3410</td>
<td>86.8%</td>
</tr>
<tr>
<td>4-1</td>
<td>3371</td>
<td>3296</td>
<td>97.8%</td>
</tr>
<tr>
<td>4-2</td>
<td>3342</td>
<td>3127</td>
<td>93.6%</td>
</tr>
<tr>
<td>4-3</td>
<td>3218</td>
<td>2660</td>
<td>82.7%</td>
</tr>
<tr>
<td>4-4</td>
<td>2708</td>
<td>2484</td>
<td>91.7%</td>
</tr>
<tr>
<td>4-5</td>
<td>2454</td>
<td>2281</td>
<td>93.0%</td>
</tr>
</tbody>
</table>

Table 7.4: A listing of player success rates on the first 16 tutorial levels of Foldit separated by tutorial section. Player data gathered during this specific timeframe included players with sessions already in progress. This allows the successful attempts for one level to be lower than the total attempts for the next.

success data for the same levels. Since the release version of Foldit is still under continuous development, we only include player data where the score threshold for each puzzle is the same as thresholds given to the Stratabots and the initial protein structure is the same. Puzzle success rates for players obtained from Foldit’s data repository can be seen in Table 7.4.

A Spearman’s rank-order correlation was run to determine the correlation between player and Stratabot success rates on each level. There was a strong positive correlation between these success rates, which was found to be statistically significant ($\rho = .754$, $p < .001$). While players are generally successful, five levels have sub-90% success rates. Of these five, three were completable by only one Stratabot, indicating the usefulness of Stratabots to identify levels that may be unusually difficult for players.

7.7 Discussion

As indicated in our results, Stratabots possessing the Wiggle skill can complete a large majority of levels indicating the power of this tool throughout Foldit. This tool is first introduced to players in Level 1-3 yet it is available to Stratabots through the exposed scripting language of Foldit on every level. Additionally, the scripting language provides more power over this tool to bots compared to human players during tutorial levels. Due to the importance of this tool and the extensive capabilities it possesses, it may be beneficial to focus more time on teaching this skill to players. We realize designers removed specific functionality to force players to use the to-be-taught skill rather than allowing them to solve puzzles however they please, and we acknowledge this is a common practice in tutorial design; however, we believe more research could go into the effect this has on players and whether or not it is beneficial.
to limit player capabilities—especially limiting the most powerful and useful tools at the player’s disposal in future levels.

Some level transitions in the tutorials of Foldit don’t seem to require any additional skills and the difficulty of the levels themselves does not increase; however, some of these levels are meant to introduce concepts and terminology not directly reflected in the actions that players take. For example, Level 1-1 introduces the concept of clashes yet this is not required to computationally manipulate proteins and improve one’s score in Foldit. It is merely a graphical indicator of where it might be most beneficial to focus attention. The Stratabots appear to have the most trouble near the end of the third section of the tutorial levels. This is mirrored by the player data, and may indicate that section of the game needs attention.

In our research, we find Stratabots still present useful information to the analysis of level progressions even when no known optimal player exists as was the case in Chapter 6. This shows that even when player models or profiles don’t exist for every player type, Stratabots allow designers to understand the effect of a level progression based on subsets of skills. As human computation games increase in number, this may be a promising direction for designers to understand how players will view their level progression and which skills designers may want to focus on during tutorial design. To understand the extensibility of this method, additional research is needed into gameplay requiring more strategic skills (such as chess), rather than gameplay that is reliant on mechanics-based skills.

Finally, we found that a Monte Carlo based approach gave us the same performance for each bot without the need to individually create each one. This drastically reduced design time while allowing us to perform the same analysis. With the tremendous research going into Monte Carlo approaches to game AI, we foresee this method becoming increasingly viable for skill-based analysis of games and players.
Chapter 8

Future Work

There are many directions for future work that builds on the ideas presented in this dissertation. These include additions and improvements to the StrataBot framework, updates to GrACE, and additional avenues for analysis of play.

8.1 GrACE Design

In future iterations of GrACE, we want to implement the framework outlined in this dissertation as an additional constraint on the procedural puzzle generator. We will then focus on testing the new progression to explore if a more appropriate progression affects the learning outcomes of the game. Since the goal of GrACE is to teach players about minimum spanning trees, these future tests will continually evaluate players on their mastery of MSTs outside of the game. However, simulations of players through levels can add significant time to the generation process. It remains an open problem how to add player simulations into this process in an efficient manner. One potential direction could be to train a machine learning algorithm on the StrataBot classifications and use that classifier rather than the direct simulations during generation. This would substantially reduce simulation time, however it could remove the guarantee that levels are solvable by particular StrataBots. Once levels are generated, an end-stage check is necessary to ensure the content is classified appropriately by the StrataBot framework.

Creating additional bots will help fill the gaps in strategy that the current set of StrataBots have. Strictly positive edge weights in GrACE means the player can select all edges with weight one (provided they create no cycles) and guarantee the correct solution will contain them. If the player repeats this process with edges weighted 2, eventually they will arrive at the correct solution. In fact, if the player does this, they perform Kruskal’s algorithm (Kruskal, 1956). Additional StrataBots that run other minimum spanning tree algorithms can help future design iterations of GrACE focus on requiring a player learn a specific algorithm or give them the option to complete levels with any correct algorithm. In addition, StrataBots for GrACE were created by hand rather than using the general solution outlined in Chapter 7. To cope with the changing design and mechanics of future updates, moving the general solution would help ensure we could continue to use StrataBots as analysis tool throughout development.

Furthermore, we want to evaluate players in their progression as well as the difficulty of the levels. Baker, Goldstein, and Heffernan (2010) demonstrate a method for detecting when players learn a particular skill and whether learning occurs gradually or as a “eureka” moment. Baker et al.’s method requires large quantities of player data in order to detect these occurrences, however we think a method similar to our work presented in Chapter 4 that quantifies the difference between a player’s actions and given player model could achieve the same goal without the player
data. This means we need a better understanding of how far players diverge from specific algorithmic strategies. Initially, we plan to use a window-based approach as outlined in Chapter 6, but the extension of this framework to additional game genres may require further changes. Additionally, quantifying when a player has sufficiently demonstrated they learned a skill in order to advance them through a level progression remains an open question: How many times must a player do something successfully before it is considered learned?

8.2 Progression Creation

The analysis completed in this research is a starting point for future progression generation—whether through testing manually created progressions or as part of a generation tool. This research provides some insight into what makes puzzles difficult but more is necessary to understand the optimal introduction of levels to maximize player learning. We do not believe presenting levels exclusively in increasing difficulty is sufficient. Rather, some sort of player performance metric should be introduced into the equation to give players more personalized and effective content based on their current position in a game’s learning curve. Even with an optimal level progression, players may not progress through the game and learn everything designers hoped. It could be that outside factors pose additional challenges or some players aren’t fully invested in learning. Understanding why some players advance to complex strategies and others struggle requires qualitative feedback from players including surveys and interviews.

The presented work is based on educational and instructional puzzle games; however, we argue that the approach of our gradient AI-bot fitting approach is of benefit to both educational and entertainment, and, in fact, is applicable to any game where a strategy can be defined algorithmically.

8.3 Stratabots

So far, Stratabots have shown promise in progression analysis for puzzle-based games, but we believe there are still contexts and situations that could pose significant challenges. We would like to apply Stratabots to more games to understand if and where this framework breaks down. In those cases, we expect Stratbots to provide less useful information on level progressions. Currently, StrataBots have not evaluated progressions in games that require dexterity-based skills such as real-time strategy games, first-person shooters, platformers, and action role-playing games.

Additionally, there is ample opportunity to improve the runtime performance of Stratabots. Some levels required almost 48 hours of simulated play to complete. While this is may be less than the time it takes to do some types of large-scale human playtesting, it is still a bottleneck many designers could be discouraged by. Giving the Lua scripting interface to Foldit more available memory and save slots as well as applying MCTS performance enhancements (Keehl and Smith, 2018) should drastically reduce overall runtime. Furthermore, most of the skills enacted by StrataBots in Foldit are relatively low-level or direct game mechanics. With the inclusion of higher level skills as was done in GrACE, we can begin to analyze strategies that players use instead of merely conceptual understanding of the game. Conceptual understanding is important for the analysis of tutorials, but as we look to analyze more challenging Foldit levels, we need to understand how different strategies affect player behavior and solution traces.
Chapter 9

Conclusion and Summary of Contributions

The work presented in this dissertation focuses on the design of educational games and automated skill-based methods to analyze the content contained within an educational game. Specifically, we make three main contributions:

- **How does the introduction of procedural content generation in an educational game affect player behavior and performance?** Insights into the design, development, and evaluation of an educational game to understand how the introduction of PCG into a game’s level progression alters player behavior and affects learning. This includes a novel mixed-methods approach that incorporates hierarchical clustering, player think-aloud data, game data, and surveys introduced in Chapter 4.

- **How can the skills necessary to complete a video game be extracted and organized by dependencies?** A formal process for determining the skills a game entails, extracting those skills from player interviews, and organizing skill relationships and dependencies shown in Chapter 5.

- **How can the logical organization of video game levels be analyzed by required skills in order to promote learning without relying on existing player data?** A novel method for automated analysis of level progressions based on designer-specified skills we implement in Chapter 6 as well as a novel modification to MCTS for skill-based player simulations to determine which skills are required to complete specific levels implemented in Chapter 7.

9.1 *GrACE*

In Chapter 3, we discussed the results from an experimental pilot study of *GrACE*, a game designed to foster computational thinking through PCG and collaboration, with the aim of retrieving design insights into the creation and evaluation of this CS educational game. While the game achieved a moderate improvement in the conceptual understanding of the MST problem, multiple unexpected insights were found such as the need for designing explicitly for collaboration, and the need to more deeply understand player profiles. These design insights are guiding our future work with *GrACE*, and can serve as guidelines for others interested in developing CS educational games.
Chapter 9. Conclusion and Summary of Contributions

9.1.1 Cross-Cultural Evaluation

By investigating the same game among middle school students in the USA and Nigeria, this study has enabled us to perform a cross-cultural evaluation of a CS teaching game. Findings from the study have led us to conclude that there is a need to introduce a cross-cultural perspective to educational game design and its evaluation in order to broaden a game’s impact. Our results will be used to improve future iterations of the game and achieve our ultimate goal of broadening the game to be both inclusive to girls as well across varying cultures. We intend to achieve this by integrating adaptive models that will be able to give personalized interactions to players notwithstanding their cultural background.

9.1.2 Mixed-Methods Player Analysis

In Chapter 4, we detail the analysis from nine individuals playing GrACE, an educational puzzle game emphasizing algorithmic thinking. Our research outlines insights gained through the use of a mixed-methods strategy for gameplay analysis that triangulates data from four perspectives: think-aloud voice recordings, qualitative playtrace analysis or retrospective player sense-making, level progression visualizations and quantitative clustering on player actions. We have shown how a mixed-methods approach can be applied to an educational game and how it helped us gain insights that may have been overlooked or incorrectly inferred without game analytics or using a single method to analyze playtraces. Finally, data-driven design and evaluation of an educational game has allowed us to detect problem areas early in the design phase where steps can still be taken to correct shortcomings. We hope that future game designers—educational and otherwise—take into consideration the benefits of mixed-methods approaches to game analytics and develop appropriate metrics for success that will ensure the game designer’s goals are met.

9.2 Learning Hierarchies

Like designing good instruction, designing an enjoyable, easy to learn game requires understanding what skills the game’s mechanics require, how these build and depend on each other, and thus, in what order to introduce them to players. Learning hierarchies in instructional design and skill chains in game design are common formal models to map these relations. Yet where instructional design can rely on cognitive task analysis to empirically identify the learning hierarchy of a task, game design so far relied on expert interpretation to identify the skill chains of games. Given that experts are typically blind to the full range of tacit skills they have mastered, this risks overlooking crucial skills that novices need to be taught. In Chapter 5, we therefore developed and presented an adapted cognitive task analysis method to elicit a game’s skill chain from empirical player observation and interviewing. The method combines stimulated recall interviews on targeted stretches of gameplay with directed qualitative content analysis of the generated data. We demonstrated and critically reflected on the method through a case study use on the game Paradox. The skill chain elicited for Paradox with our method indeed proved aligned with but more comprehensive than a designer-crafted skill chain produced without empirical player observation. Specifically, it included critical missing pre-requisite and low-level skills.

While principally effective in our case study, the method also showed major limitations and open questions regarding its efficiency, generalizability across genres and more complex games, ability to reliably elicit skill dependencies, and validity of
captured emergent player strategies, which we hope to address in future applications with replications and mixed method approaches.

### 9.3 Stratabot Implementation

Chapter 6 explores level difficulty in a puzzle-based educational game called GrACE. To determine puzzle difficulty, we propose a set of incrementally complex AI-bots designed to solve these puzzles that are called StrataBots, named because they each occupy different points on a skill chain or strata of the optimal strategy. Each puzzle is classified by the StrataBots that can solve the puzzle. When compared with human playtraces collected from a previous pilot study, results suggest that human players find puzzles that require more complex strategies more difficult to solve, and—in the procedurally content generated condition—can manipulate the levels they play to unintentionally receive easier puzzles in order to progress through the game. Future work will address how these results can help automatically design better difficulty progressions for players to promote learning.

### 9.4 From Skill Chains to Stratabots

In Chapter 7, we demonstrated the applicability of skill chains to the design and production of hierarchical skill-based AIs to model players of varying skill levels. A series of bots created for Foldit, a human computation game in the field of biology, shows that introductory levels require the skills that designers expect with a few exceptions. These findings have the potential to improve tutorial level design by ensuring the skills needed to complete a level are those desired by the designer—no more and no less. Bot performance was also compared to human players showing that levels perceived as more difficult by players (i.e. lower success rate) were those that fewer bots could solve. Similarly, bots that could solve more difficult levels generally had more skills at their disposal implying that advanced players could solve those levels while novice players could not. Future work will center on expanding the application of Stratabots to other games and improving runtime performance.
Bibliography


Ambinder, Mike (Mar. 2009). Valve’s Approach to Playtesting: The Application of Empiricism. Published: Game Developer’s Conference.


Avontuur, Tetske, Pieter Spronck, and Menno Van Zaanen (2013). “Player Skill Modeling in Starcraft II.” In: AIIDE.


Browne, Cameron and Frederic Maire (2010). “Evolutionary game design”. In: IEEE Transactions on Computational Intelligence and AI in Games 2, pp. 1–16.

Browne, Cameron B et al. (2012). “A survey of monte carlo tree search methods”. In: IEEE Transactions on Computational Intelligence and AI in games 4, pp. 1–43.


Chaslot, Guillaume et al. (2008). “Monte-Carlo Tree Search: A New Framework for Game AI.” In: AIIDE.


Cook, Daniel (July 2007). *The Chemistry Of Game Design*.

Cook, Michael, Simon Colton, and Alison Pease (2012). “Aesthetic considerations for automated platformer design”. In: *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*.


Dahlskog, Steve and Julian Togelius (May 2012). “Patterns and Procedural Content Generation”. In: *Proceedings of the Workshop on Design Patterns in Games (DPG 2012)*, co-located with the Foundations of Digital Games 2012 conference. Raleigh, NC.


DeRosa, Phillip (2007). “Tracking player feedback to improve game design”. In: *Gamasutra*.


Doerschuk, Peggy, Jiangjiang Liu, and Judith Mann (June 2007). “Pilot summer camps in computing for middle school girls: from organization through assessment”. In: SIGCSE Bull. 39, pp. 4–8.


firecorth (Nov. 2013). Game design alchemy the second.
Bibliography

Gilski, Miroslaw et al. (Nov. 2011). “High-Resolution Structure of a Retroviral Protease Folded as a Monomer”. In: Acta Crystallographica D67, pp. 907–914.
Bibliography

Harteveld, Casper et al. (June 2014). “A Design-Focused Analysis of Games Teaching Computer Science”. In: Proceedings of Games+Learning+Society 10. Madison, WI.


Holmgård, Christoffer et al. (2014). “Evolving personas for player decision modeling”. In: Computational Intelligence and Games (CIG), 2014 IEEE Conference on. IEEE, pp. 1–8.

— (2015). “Monte-carlo tree search for persona based player modeling”. In: Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference.


Horn, Britton et al. (2018). “A Monte Carlo Approach to Skill-Based Automated Playtesting”. In: Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference.


Kam, Matthew et al. (2007). “Mobile Gaming with Children in Rural India: Contextual Factors in the Use of Game Design Patterns.” In: DiGRA Conference.


Kirschner, David and J Patrick Williams (2013). “Experts and Novices or Expertise? Positioning Players through Gameplay Reviews”. In: DiGRA 2013: DeFragging Game Studies. DiGRA.


Lorenzo-Seva, Urbano and Jos MF Ten Berge (2006). “Tucker’s congruence coefficient as a meaningful index of factor similarity”. In: Methodology 2, pp. 57–64.


Novak, Joseph D and Alberto J Canhas (2008). “The theory underlying concept maps and how to construct and use them”. In:


Osborn, Joseph C. et al. (2014). “Evaluating play trace (dis)similarity metrics”. In: 10th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment.


Romero, Ramon (2008). “Successful instrumentation: Tracking attitudes and behaviors to improve games”. In: *Game Developer’s Conference*.
Sarkar, Anurag and Seth Cooper (2017). “Level difficulty and player skill prediction in human computation games”. In: *Proceedings of the 13th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.
Shaker, N. et al. (2012). “Evolving Levels for Super Mario Bros Using Grammatical Evolution”. In: *IEEE Transactions on Computational Intelligence and Games (CIG)*.


Smith, Gillian et al. (Mar. 2011b). “Launchpad: A Rhythm-Based Level Generator for 2D Platformers”. In: IEEE Transactions on Computational Intelligence and AI in Games 3, pp. 1–16.


