Automated Playtesting of Platformer Games using Reinforcement Learning

A thesis presented to the academic faculty in partial fulfillment of the requirement for the Degree
Masters of Science in Game Science and Design in the College of Arts, Media and Design

by
Varun Sriram

Thesis Committee
Thesis Advisors: Dr. Giovanni Troiano
Committee Members: Dr. Christoffer Holmgård, Dr. Casper Harteveld, Dr. Magy Seif El-Nasr

Northeastern University
Boston, Massachusetts
December 2019
Abstract

Platformer games are popular in the video game industry and their design require much efforts from game companies. As part of the design process, playtesting is key for improving the gameplay before game release. Playtesting is the quality assurance phase of the game development cycle where people are hired to play the game, report bugs and provide feedback regarding the playability of the game. This feedback could be used for game balancing (process of tuning game rules to prevent them from being ineffective or provide undesirable results). However, playtesting may be expensive if done manually and may require several iterations, resulting in high budget requirement and time for game companies. In this thesis, we investigate a way to automatically playtest 2D platformer levels using a combination of deep reinforcement learning and curriculum learning, for both quality assurance and game balancing. Deep Reinforcement Learning has contributed greatly in playing games (Atari and Dota 2) and in this thesis, we will try to replicate the results to playtest games. Curriculum learning is an approach that has shown promising results thus we will explore it to derive useful results. We develop our APT tool by training an artificial intelligence (AI) agent on several different platformer levels following a curriculum, and use the trained agent to playtest newly-created levels. Our APT is able to identify areas of the level that needed design improvements and further gameplay balancing. We contribute a reliable APT tool for designers that wish to easily design 2D platformer games and a discussion of how our results extend to APT at-large.

Keywords: 2D Platformer Games, Quality Assurance, Automated Playtesting, Deep Reinforcement Learning, Curriculum Learning

Northeastern University
Boston, Massachusetts
December 2019
# Table of Contents

List of Tables

List of Figures

List of Equations

Acknowledgements

1. Introduction 7

2. Background 8

3. Methodology 9

4. Results 15

5. Discussion 25

6. Conclusion 31

References 35

Appendix 37
List of Tables

Table 1. Hyperparameters for Neural Network
Table 2. Reward Structure for the agent
Table 3. Damage taken by agent
Table 4. Agent Death

List of Figures

Figure 1. Huge map system in Ori and the Blind Forest
Figure 2. Turrets and Spikes Pits in Ori and the Blind Forest
Figure 3. Markov Reward Process
Figure 4. A Neuron
Figure 5. A Neural Network
Figure 6. Raycasting
Figure 7. Level with spike pits
Figure 8. Level with enemy AI
Figure 9. Level with both, spike pits and enemy
Figure 10. Unseen level for testing
Figure 11. Unseen level with player heatmaps
Figure 12. Heatmaps for random agent on unseen level
Figure 13. Coins in Sonic the Hedgehog
Figure 14. “Spirit light” is the equivalent to coins and acts as a currency to unlock new abilities
Figure 15. Special moves in Ori and the Blind Forest

Figure 16. Ability tree in Ori and the Blind Forest

List of Equations

Equation 1. Equation for Markov Reward Process

Equation 2. Bellman Equation
ACKNOWLEDGEMENTS

Firstly I would like to thank my committee members Dr. Christoffe Holmgard, Dr. Casper Harteveld, Dr. Giovanni Troiano and Dr. Magy Seif El-Nasr for their guidance and help which eased the thesis writing process. Thank you Christoffe for introducing me to artificial intelligence and guiding me, answering my endless list of queries and providing constructive criticism wherever needed.

I would like to thank my family for believing in me that games science is an important field regardless of its application and helping me pursue my dreams. Special thanks to all my friends living here (USA) and in India, who helped me throughout my Master's journey with their constant support.
INTRODUCTION

Playtesting is an important part of the game development cycle. It provides feedback regarding the “playability” of the game. In the game industry, the quality assurance (QA) process involves hiring human playtesters to play the game, report bugs and provide feedback regarding the playability of the game.

Game development is an iterative process and a game is released when it is balanced and almost has no bugs (it is difficult to assume that a game will have no bugs). For this iterative process to function, there is constant requirement of human playtesters which cost money and time. One solution to cut down on QA cost is by automating the playtesting process resulting in a minimal need for human playtesters.

Machine learning (branch of artificial intelligence) is used in play-testing applications (PTA; (Gudmundsson et al., 2018); such approach is often referred to as automated play-testing (APT). In APT, pre-trained artificial intelligence (AI) agents will “play” the game, test for bugs and provide feedback to game designers regarding balance in game mechanics, and provide QA (Pfau, Smeddinck, & Malaka, 2017).

Platformer Games (PG) are amongst the most popular type of video games (Galyonkin S., 2019) and include games like Mario Bros, Sonic The Hedgehog, and Crash Bandicoot. In PG games, the main task of players is usually to jump between obstacles, move and jump from one platform to another, and avoiding and or shoot enemies. A PG is a combination of various design patterns,
including collectibles, mechanics, and power-ups (Khalifa, de Mesentier Silva, & Togelius, 2019), which developers can use to create a vast number of unique levels. However, analyzing design patterns (Smith, Cha, & Whitehead, 2008) and their combinations for balancing the gameplay (Spencer, 1977) is hard. Depending on the dimensionality of the game objects used, 3D games have x, y, and z axes can and ignoring the z axis will result in 2D PGs. In this thesis, we will be focusing on 2D PGs.

This thesis uses APT in the context of PGs, with the end goal of helping game designers playtest and design their PGs effectively and rapidly. We develop a PTA, which automatically plays and tests premade PG levels. The PTA developed for this thesis will provide other developers with feedback about the difficulty and the degree of game balance in their game design. The thesis will help developers to playtest their games for feedback regarding the attributes mentioned above. Additionally, this system could be applicable to students who aspire to develop platformer games, independent game developers, and for research in game artificial intelligence.

BACKGROUND

This thesis investigates APT in PGs using machine learning and deep reinforcement learning to be precise. There has been extensive research conducted in the field of APT with some of them done by King (Gudmundsson et al., 2018) using supervised learning, EA (Zhao et al., 2019) using inverse reinforcement learning and researchers (Mugrai, Silva, Holmgard, & Togelius, 2019) use monte carlo tree search with genetic evolution, (García-Sánchez, Tonda, Mora,
Squillero, & Merelo, 2018) use genetic evolution. All the mentioned papers have provided good results to the field of APT.

Curriculum learning (CL) (Bengio, Louradour, Collobert, & Weston, 2009) is defined as a way of training a machine learning model where more difficult aspects of a problem are slowly introduced to challenge the model/agent optimally. This way, I could train AI models well versed with different aspects of its environment as the problems are presented to the agent following a proper difficulty curve (Aponte, Levieux, & Natkin, 2009).

Deep reinforcement learning has been used to play different games like Atari (Mnih et al., 2013) by DeepMind, Dota 2 (OpenAI, 2018) by OpenAI etc. DeepMind’s AI agents have successfully completed the Atari games and OpenAI’s Dota team has defeated the current world champions (Peng, Sarazen, 2019) too. Therefore, I could use deep reinforcement learning not just to play, but to playtest games as well. To explore new possibilities and set baselines, in this thesis, I position my work in the area of APT, with machine learning as an approach and deep reinforcement learning in particular with curriculum learning.

Next, I briefly review previous work in both areas, and discuss challenges that earlier work encountered when developing PG levels, and how APT can help tackle such challenges.
Level Design in Platformer Games

Consider a 2D PG like *Ori and the Blind Forest* (*Moon Studios*, 2015). It consists of a complex map system, which includes PG mechanics such as jumping to and from platforms, as well as solving puzzles. Ori, the protagonist can faces enemies (e.g., turrets, melee frogs, porcupines with long ranged projectiles), collect health shards, energy shards and many other special items (e.g., snow orb, key for doors). This presence of multiple elements results in the formation of multiple mechanics, setting no limit to the number of unique levels one can design. Thus, by automating the testing process, I am able to test a vast number of levels, in a short period of time, and make playtesting more scalable, less expensive, and requiring only minimal human supervision.
Fig 1. An example of map system from *Ori and the Blind Forest*
Automated Play-Testing in Platformer games

APT is used in games to replace human testers with pre-trained AI agents. The agent will provide QA regarding the playability of the level. The advantages of APT is that it reduces time and money expenses compared to human testers. In this thesis, we explore the use of APT for the design of PGs.

Reinforcement learning is an algorithm used in machine learning to train AI agents, making them learn from their own “experience”. In absence of training data, the agent learns empirically from “direct” experience while collecting data via training examples (e.g. tagging actions as
good actions or bad actions) through trial-and-error as it attempts its task, with the goal of maximizing long-term reward.

Reinforcement learning relies on rewarding the AI agent based on its actions known as *Markov Reward Process* (MRP). The MRP involves value judgement, which is the cumulative reward through some particular state sequence of the agent.

An MRP is a tuple \((S, P, R, \gamma, \pi)\) where \(S\) is a finite state space, \(\pi\) is the policy/model adopted by the agent, \(P\) is the state transition probability function, \(R\) is a reward function where, \(Rs = \mathbb{E}[R_{t+1} | S_t = S]\), it tells the agent how much immediate is expected to get from state \(S\) at the moment.

To sum it up,

![Figure 3. Markov Reward Process](image-url)
\[
\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]
\]

Equation 1. Equation for Markov Reward Process

**METHODOLOGY**

**Tools**

There are several tools for game development, including Unity, Unreal, and Game Maker Studio 2. Unity3D has a grid system, an asset package named Corgi Engine (MoreMountains, 2015) that helps create PGs with ease and an external Machine Learning API named *Unity-ML Agents*. The API can be combined with Unity to train reinforcement learning agents in the PG levels created.

**Algorithm**

**Deep Learning Algorithm and Neural Networks**

“It is learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations. In deep learning, these layered representations are (almost always) learned via models called neural networks, structured in literal layers stacked on top of each other.” (Chollet, 2018)

Before I discuss about a neural network’s structure, let us see what a neuron looks like
Figure 4. A neuron

Where $x_1$, $x_2$, $x_3$ are inputs and “output” is the output or result of the neural network.

The circle is what is called an *activation function* and it introduces non-linearity to the output of the neuron. There are several functions used for activation like sigmoid, tanh, ReLU, etc. (Olgac & Karlik, 2011).

A collection of these neurons make up a neural network:
The leftmost layer is called the input layer, which serve as input to a neural network. For the APT agent I developed, the inputs are determined by raycasting, where the rays of fixed length originate from the AI agent radially outward and when obstructed, detect objects. These objects could be platforms, enemies, spikes pits, lava, coins or any interactable object in the scene. The objects detected are the agent’s intractable environment and serve as input to our model.
**Fig 6. Raycasting**

The hidden layers are the layers where the calculations happen, and in my PTA, it determines the best possible action to achieve the highest reward for the state depending on the inputs for that agent state.

The rightmost layer, the output layer is the result of all the calculations in the hidden layer. The output is the action that agent performs which can be one out of a set of three actions namely, move left, move right and jump.


**Curriculum Learning**

*Curriculum learning (CL)* is a concept coined as early as 1993 (Elman, 1993) which is defined as a way of training a machine learning model where more difficult aspects of a problem are slowly introduced to challenge the model/agent optimally. CL can be explained with the following analogy: as children, we were educated with an order of classes and topics, where arithmetic is taught before algebra, and algebra before calculus; as such, the skills we learn earlier act as the basis (or support) for the learning of more advanced skills. The reports in the paper *Curriculum Learning* (Bengio et al., 2009) speak positively about CL. The model trained using CL had a lower error rate compared to that of the model trained without a curriculum. At the end of their training, CL based model had an average test error of approximately 15% compared to non-CL’s 30%.

In this thesis, I use platformer levels as curriculum starting with a simple difficulty level aimed to familiarize the mechanics to the agent. As the game progresses, new mechanics are introduced in succession (as individual levels) and the difficulty increases. As a result, I create a gradual difficulty curve (Aponte et al., 2009) for the agent, so that it can understand each and every mechanic of the game.

**The Design**

In this thesis, the curriculum is a set of levels for agent training. The preliminary levels will consist of a single unique mechanic for the agent followed by levels having a combination of the previous mechanics. The curriculum comprises of 7 levels and the order of the first three levels in the curriculum are spike pits, Enemy AI and a combination of spike pits and enemy AI (Fig 4,
Fig 5, Fig 6, see appendix for the whole curriculum). Unless the curriculum follows a smooth difficulty curve, the number of levels in the curriculum does not play a substantial role in creating a well trained agent. The same reasoning is applicable with the order of the curriculum. In our curriculum, changing the order of the first three levels from spike pits, enemy AI and both (spike pits and enemy) to enemy AI, lava and both (enemy and lava) would provide the same results.

Figure 7. A level with spike pits

Figure 8. Level with Enemy AI
I set the “max_steps” value as 150,000 since any value above it resulted in overfitting (a case when the agent is too familiar with the training environment that it underperforms in the testing environment). Overfitting is an important issue to keep in mind while training. With longer training time, the agent will become too familiar with the level, underperforming on unseen levels. After 150,000 steps, the next level in the curriculum is loaded and the agent continues to train in the new environment. During the CL, the agent converged around 135,000 steps after which it successfully completed the levels for the rest of 15,000 steps. *Unity ML-Agents* API comes with a configuration file which stores all the hyperparameters (variables) required for the training process (Table 1). Variables like *lambda* and learning rate are variables for the Bellman Equation, where \( V(s) \) - The value of a given state, \( R(s, a) \) - Reward of the action a for state s, \( \gamma \) (gamma) - Discount factor, \( V(s') \) - The value for the following state.
\[ V(s) = \max_a (R(s, a) + \gamma V(s')) \]

**Equation 2. Bellman Equation**

The network developed in this thesis consists of 2 hidden layers with 128 units each since the inputs for the network will not be too many. Using a very deep network for fewer inputs will result in overfitting. I settled for this structure after trial and error.

“Exploration vs Exploitation tradeoff” (Ecoffet, Huizinga, Lehman, Stanley, & Clune, 2019) is a well known problem in reinforcement learning where a learning system has to choose in an environment with uncertainty pay-offs; it is a dilemma for a decision making system with incomplete knowledge of the environment where it has to choose between repeating decisions that have worked before (exploiting) or to make newer decisions, hoping to achieve even greater rewards (exploring). The use_curiosity variable will help us train an agent capable to adopt different paths while looking to explore the environment for a high rewarding path. This gives game designers diverse data for analysis.

**Table 1. Hyperparameters**

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>hidden_units</td>
<td>128</td>
</tr>
<tr>
<td>lambda</td>
<td>0.95</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>learning_rate</td>
<td>3.0e-4</td>
</tr>
<tr>
<td>max_steps</td>
<td>5.0e5</td>
</tr>
<tr>
<td>memory_size</td>
<td>256</td>
</tr>
<tr>
<td>normalize</td>
<td>false</td>
</tr>
<tr>
<td>num_epoch</td>
<td>3</td>
</tr>
<tr>
<td>num_layers</td>
<td>2</td>
</tr>
<tr>
<td>time_horizon</td>
<td>128</td>
</tr>
<tr>
<td>sequence_length</td>
<td>64</td>
</tr>
<tr>
<td>summary_freq</td>
<td>2000</td>
</tr>
<tr>
<td>use_recurrent</td>
<td>true</td>
</tr>
<tr>
<td>use_curiosity</td>
<td>true</td>
</tr>
<tr>
<td>curiosity_strength</td>
<td>0.15</td>
</tr>
</tbody>
</table>
Reward Structuring

As mentioned earlier, reinforcement learning follows the MRP. For every important interaction that the agent has with objects (lava, enemy, spikes and coins) in the environment, it is rewarded points accordingly. Reward restructuring is the process of tuning reward values (Dayan & Balleine, 2002). For every action, the AI agent is awarded points as a feedback for its quality. Consider an environment with spikes pits that the agent has never encountered. The agent is bound to fall and die in the spike pits in the first encounter, receiving negative points, affecting the overall reward value for that play-through iteration. In the next iteration, the agent will refrain from jumping into the spike pits. Similarly, positive points are awarded for completing a level and picking up coins. Given below is the reward structure for our agent.

Table 2. Reward Structure for the agent

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Coins</td>
<td>+0.1f</td>
</tr>
<tr>
<td>LevelEnd (Win)</td>
<td>+1f</td>
</tr>
<tr>
<td>Damage Taken</td>
<td>-0.07f</td>
</tr>
<tr>
<td>Player Death</td>
<td>-0.15f</td>
</tr>
</tbody>
</table>
Data Acquisition

To evaluate the PTA’s success, I logged the agent’s movement data and event data in two respective files to conduct qualitative and quantitative analysis.

File 1. Movement Data

<table>
<thead>
<tr>
<th>X coordinate</th>
<th>Agent’s x coordinates every second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y coordinate</td>
<td>Agent’s y coordinates every second</td>
</tr>
</tbody>
</table>

File 2. Event Data

<table>
<thead>
<tr>
<th>X coordinate</th>
<th>Agent’s x coordinates every second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y coordinate</td>
<td>Agent’s y coordinates every second</td>
</tr>
<tr>
<td>Event</td>
<td>Event that occurred during the playtest which could be one of: Player Death, Coin Pickup, Level End ( Reached Final Flag), Damage Taken (from Enemy or Lava), Environment Reset (Resets the environment if the agent does not receive a reward for a certain amount of time)</td>
</tr>
<tr>
<td>Interaction</td>
<td>Noting the object name the agent interacted with</td>
</tr>
</tbody>
</table>

RESULTS

The agent tested an unseen level for fifteen minutes and the results are given below. Along with the pre-trained agent, I also logged the movement of a random baseline agent which has basic knowledge of movements, the being walking and jumping. Now I will compare the two agents’ performance in the unseen level.
Qualitative Analysis

With the movement data logged in file one, I create heatmaps of the points the agent has visited in the fifteen minute playtest on tableau (Fig 11). I analyze the heatmap to identify any reason different interactions of the agent with the environment.

Fig 10. Unseen level

Fig 11. Unseen level with agent heatmaps
In this level, the blue creatures are the AI Enemies who on being touched, will damage the agent, and the same goes for the red lava tiles. The spikes on the ground will kill the agent instantly upon interaction. The flag is the end point and the coins are the currency in game. I have placed the coins in a path from start to goal flag since coins in a path have a similar effect on player paths if there were no coins present (Sarkar, Sriram, Padte, Cao, & Cooper. 2018 ). I decided to place them because it is present in many PGs in some or the other form as currencies.
Fig 13. Coins in Sonic the Hedgehog (Sega, 1991)
Fig 14. “Spirit light” is the equivalent to coins and acts as a currency to unlock new abilities

Upon analysis, evidence suggests that the agent successfully reached the end goal a number of times. The red spots point of areas where the agent has visited the most. Red spots B, C, and E in Fig 11 are a result of the agent jumping from the wrong position and ending up dead. This will give game designers insight regarding the spacing of platforms. Instances where the agent dies as a result of taking damage from enemies and lava are present, which will help designers balance damage values for the Enemies or lava in the game. Visualization also gives subtle insights. For instance, the distance between platforms containing lava F and G in figure 11 is not enough resulting in the agent receiving damage from “lava1” from figure 6 as a result of jumping on platform containing F (Figure 11). This information will help designers prevent unnecessary interactions between the agent and environment, providing a clean experience for the players after release.

When compared with the random baseline agent, it is evident that the trained agent has performed better. The random agent never made it ahead of “spike C” proving that the pre-trained agent has working knowledge of all the mechanics and dynamics of the game which is the result of CL.

Quantitative Analysis
Quantitative analysis proves that the PTA is resource efficient performing 434 (for the pre trained agent) and 340 (for the random agent) playthroughs in fifteen minutes, saving time compared to hiring human play-testers and receiving feedback from them.

161 out of 184 times, the agent has received damage from “Enemy1” indicating that 88% of times it will interact (attack) the agent. Game designers depending on their design choice, could reduce the interaction probability by reducing the raycast length of that Enemy, or by increasing the platform length giving more space for the agent to jump without being attacked.

Interactions with various enemy objects (Enemy, Lava, Spikes) also portray the difficulty curve of the level. 227 instances of “DamageTaken” out of 268 occurred in the first half of the level unlike the second half. Adding objects to the second half or removing objects from the first half is a solution to increase or decrease the overall difficulty of the level respectively.

As I inferred from the qualitative analysis using heatmaps, the baseline agent barely never made it past “Spike C” thus providing little to no information regarding the quality of the level.

Table 3. Damage Taken Instances

<table>
<thead>
<tr>
<th></th>
<th>Trained Agent</th>
<th>Random Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Damage Taken Instances</td>
<td>268</td>
<td>104</td>
</tr>
<tr>
<td>Damage from Enemy AI</td>
<td>184</td>
<td>104</td>
</tr>
<tr>
<td>Damage taken from Lava</td>
<td>84</td>
<td>0</td>
</tr>
<tr>
<td>Lava 1</td>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>Lava 2</td>
<td>37</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Trained Agent</td>
<td>Random Agent</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Lava 3</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>Enemy 1</td>
<td>161</td>
<td>104</td>
</tr>
<tr>
<td>Enemy 2</td>
<td>23</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4. Death

<table>
<thead>
<tr>
<th></th>
<th>Trained Agent</th>
<th>Random Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Death by Spikes Pits</td>
<td>329</td>
<td>224</td>
</tr>
<tr>
<td>Spike 1</td>
<td>27</td>
<td>194</td>
</tr>
<tr>
<td>Spike 2</td>
<td>22</td>
<td>14</td>
</tr>
<tr>
<td>Spike 3</td>
<td>146</td>
<td>16</td>
</tr>
<tr>
<td>Spike 4</td>
<td>102</td>
<td>0</td>
</tr>
<tr>
<td>Spike 5</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>Death by losing Hit Points</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>Lava 1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Lava 2</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>Lava 3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Enemy 1</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>Enemy 2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

DISCUSSION

The PTA developed in this thesis provides insights, like potential unnecessary interactions in APT (e.g., agent colliding with “lava1” while jumping from a platform below) and regarding game balancing, for instance, the agent has a pretty easy path in the second half of the level showing irregularities in the game’s difficulty. Next, I discuss the lessons learned from applying
the APT tool to the design of PG levels, and discuss the broader implications of this thesis to game design and AI development at large.

**Impact of APT in Games**

At present, PTAs represent a cost-effective, go-to playtesting solution, particularly for independent game developers, or for AAA game studios who want to rely on fast feedback in the iterative game development processes. APT is gaining recognition as King (Gudmundsson et al., 2018), a company is using PTA in their game development process. They use *supervised learning* as their approach which involves logging player moves and using them to train a machine learning model. Research on APT is also a topic of its own. Research papers like “Automated Playtesting with Procedural Personas with Evolved Heuristics” (Holmgard, Green, Liapis, & Togelius, 2018) could provide game designers to playtest levels with agents having different personalities. Thief (Square Enix, 2014) for instance, is a game that could be played stealthily (silently assassinate enemies) or by not fighting/killing enemies at all. Players can complete levels by opting any of the either mentioned ways as their playstyle. Using the approach mentioned in the paper, game designers could playtest their games using two custom personas namely “stealthy assassin” (killing enemies silently) or “stealthy pacifist” (not killing enemies at all) gaining feedback regarding the playability of the level separately from the two perspectives.
Limitations

My work presents limitations. First, in PTA the agents do not follow paths that are similar to a human playtester. For instance, in the PTA, the agent tries to maximize the distance covered in the shortest period of time, thus jumping continuously. However, that is not how a human playtester would approach a game. Human playtesters usually have a streamlined movement path, which are more linear or “obvious” of the ones performed by an AI agent, and thus offering a human perspective to the quality assurance process (Sarkar et al., 2018).

The PTA is also limited by the number of mechanics considered for playtesting. While the agent has the knowledge of only some mechanics like moving, jumping, evading enemies, spike pits and lava. 2D PGs have several mechanics forcing developers to retrain the agent for each new environment. Ori and the Blind Forest, for instance, is a game that consists of several special moves or mechanics.

Figure 15. Special moves in Ori and the Blind Forest
Future work should compensate for the above shortcomings by mixing *supervised learning* with *deep reinforcement learning* to achieve a trained agent capable following a human-like path. This PTA could also be used to playtest *procedurally generated* levels for QA of the automatically generated levels.

**Future Work**

Mixing methods like deep reinforcement learning and supervised learning will solve the problem of agents not playtesting the game like a human playtester. Using human playtesting data as a reference, the AI agent can train itself using Deep Reinforcement Learning to gain a playtesting perspective of a human. Remember that this is not *imitation learning* (Thurau, Bauckhage, &
Sagerer, 2004) since it not copying the human playtesters action completely. It only uses the human’s actions as a reference and trains the same way our agent trains via deep reinforcement learning.

Procedurally Generated Levels using *Procedural Content Generation* (PCG) for Super Mario levels is a popular research topic. It involves the automatic generation of mario levels using machine learning; it is done by developing a machine learning model that recognizes patterns in existing mario levels to automatically generate new ones. Many researchers have worked on procedurally generating levels. Some of them being using *Hidden Markov Models* (Snodgrass & Ontanon, 2017), representing map-tiles with a letter and using strings of these letters by n-gram-based text generation (Dahlskog, Togelius, & Nelson, 2014), using Long Short Term Memory recurrent neural networks (LSTMs) for generating levels trained from a corpus of Super Mario Brothers levels (Summerville & Mateas, 2016). The PTA could be used to playtest the levels generated by the methods mentioned above, provide quality assurance, and information regarding the “playability” of the level. It will fasten the iterative cycle of level generation and quality assurance.

**CONCLUSION**

This thesis explored deep reinforcement learning via curriculum learning as a method to develop automated playtesting (APT) systems and it demonstrated its ability to playtest platformer game levels and provide quality assurance, helping game designers to tweak levels for better game balance and playability.
In this work, I only looked at curriculum learning. The algorithm itself was guided by heuristics defined by curriculum learning and by us, but other heuristics could do better, possibly impacting the agent’s paths; meaning that diversity of information game designers acquire from the playtest is directly proportional to the diversity of paths taken by the agent in the testing process.

Finally, future work could adopt methods that combine supervised learning (using human playtest data for training) with deep reinforcement learning. Using a small set of human playtest data we could develop an agent uses this data as a reference to train itself on levels. Considering the human playtest data as a basis, the agent will learn to follow a human-like path.
REFERENCES


Ori and the Blind Forest, Moon Studios, 2015.


IEEE Transactions on Computational Intelligence and AI in Games, 9(4), 410–422.

https://doi.org/10.1109/TCIAIG.2016.2623560


https://doi.org/10.1016/0095-8956(77)90057-0


APPENDIX A

Screenshot of the levels

Level with spike pits

Level with only platforms
Level with only AI

Level with spike pits and AI
Level with Lava

Level with lava, spike pits and AI
A level with more platforms, spikes pits and AI