LEARN HOW TO PLAY CHINESE CLASSIC LUZHANQI

by

Yijie Wang

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Game Science and Design
in the Graduate School of the College of Arts, Media and Design of
Northeastern University
May, 2018

TABLE OF CONTENT
Luzhanqi is a classic Chinese chess game which is mainly played by children as a precursor to games like Xiangqi and Weiqi, but people of other ages also enjoy playing it as a game online for fun. This paper
talks about the origins of Luzhanqi, its basic rules, and how I implemented this game on mobile phone as a tutorial system to teach this chess game to those who have no prior experience with it.

1. INTRODUCTION

Luzhanqi, or Land Battle, is a very popular game in China, but few people seem to play it in US. When I search on Amazon, I only see one physical product. I have also asked my classmates, almost none of them have even heard about it. Perhaps the complexity of the rules and diversity of pieces cause the game to be too unfriendly for beginners. Thus, players require much practice before they can master their skills. In addition, Luzhanqi requires planning ability. Players must think carefully about what their opponents’
next move is so that they can improve their chances of winning. For my Northeastern GSND thesis, I propose the design of a Land Battle game that familiarizes new players with the game and explores the reason why the game is not frequently played in China. In addition, I will add some new game features that will break the traditional rules so that the game can have more flavor than that of the normal game. This project focuses on designing an interesting Luzhanqi game that can both teach beginners and encourage intermediate players to enjoy Luzhanqi with fresh game play.

2. BACKGROUND
In recent years, the mobile industry has been growing at a tremendous rate. With the development of technology, smartphones have become increasingly popular. It is very common to see people using smartphones anytime and anywhere. Smartphones are so powerful that they are able to do multiple tasks. Social media such as Twitter, Instagram, and Facebook are all available on phones, so social experts can express themselves everywhere (Zhang, 2007). Furthermore, mobile games are currently extremely popular; many interesting games have been developed and attract millions of players around the world, such as King of Glory (King glory, 2017).

Pokémon Go uses technology such as AR when players attempt to catch a pokémon to make players feel as if pokémon actually exist (McCartney, 2016). King of Glory allows those who do not have time to play PC games to enjoy the content very similar to that of League of Legends. Actually, it has become the most popular mobile game in China and earns 150 million per day by selling skins (King glory, 2017)! Of course, not everyone enjoys playing these types of games and is willing to spend a great deal of money. Chess is another game that can promote improve cognitive functioning and spark creativity (Doggers, 2017). Additionally, chess games can increase problem solving abilities since chess matches requires fast thinking and spontaneous problem-solving (Doggers, 2017). Finally, puzzle games are another interesting category that can improve problem solving skills (Anguera et al., 2013); the feeling of getting something right and doing so with prowess makes anyone feel good.
There are many different types of chess games in the world, and each has many fans. One of my favorites is called Luzhanqi or Land Battle chess, which is a two-player Chinese board game.

Luzhanqi can be played with either two or four players. The origin of Luzhanqi is not known because the game has only been documented only in recent times (Cazaux & Knowlton, 2017). One reason for this lack of documentation could be that it was perceived as a commercial game rather than a traditional game. Luzhanqi looks quite similar to the games Xiangqi and Mahjong. It retains the idea of a military capture game on a board defined by specific markings. There are two rectangles separating the camps in Luzhanqi very similar to the river in Xiangqi. Both the pieces and the board recall the predominant means of transport and military technologies that characterize their epoch. There are railroads, mines and bombs, but no planes, paratroopers or any other weapons are used. The earliest railroad was built in China in the 1910s and did not play a role in military tactics until a few years later. This fact may imply that Luzhanqi was invented in the second or third decade of the 20th century (Cazaux & Knowlton, 2017).

If one searches online for Luzhanqi in Chinese ("陆战棋"), plenty of electronic versions can be found, either players can interact with other players or with AI. The most popular platform of Luzhanqi is built by Tencent. In this platform, players are able to play with other players and test skills while learning teamwork with others. Beginners can play against the AI to gain a better understanding of the rules before playing with others. Many Luzhanqi games provide an AI for players if they are not able to find other players. However, most of the AIs are not smart enough for an experienced player to play against since the purpose of this kind AI is to teach the beginners to learn the basic rules of this game and give players confidence.

In addition, the rules of Luzhanqi are complex. Each player must place his or her 25 pieces on the soldier stations and headquarters spaces on their side of the board. There are some camp circles that are used to protect specific pieces, but a player is not allowed to place any pieces on the camp circles in the beginning of the game. To start the game, players first must arrange the pieces. There is no best layout in this game,
as long as players follow the rules. The flag must be placed on one of the two headquarters squares. The landmines must be placed in the last two rows. The grenades cannot be placed on the front row. After placing all the pieces, the game starts. Either player begins by making a move, and then the two opponents take turns. Soldiers and grenades move along a single line to any connected adjacent space. On the railroad, however, these pieces can move as many spaces as desired. The engineer is a very special piece because it is able to continue around corners on the railroad. As long as the engineer’s path is unobstructed, its move may cover any number of railroad linked spaces and turn as many corners as desired. The landmines and flags are not able to move. They remain in place until attacked by an enemy piece.

Among soldiers, it is the pieces of higher rank that capture the pieces of lower rank (as if they went out onto the battlefield and “out-ranked” each other). When a soldier attacks by moving onto a space occupied by an opposing piece, the piece of lower rank is removed, and the one of higher rank remains. (“1” is the highest rank; “9” is the lowest). It is recommended to keep the diagram above on hand during play until the ranks and pieces become familiar.

Some special battlefield situations may also occur. If a piece attacks another of equal rank, both pieces are removed. If a grenade attacks or is attacked by any piece, both pieces are removed. If any piece other than an Engineer attacks a Landmine, both pieces are removed; however, if an Engineer attacks a Landmine, the Landmine is removed, and the Engineer remains. All pieces are safe and may not be attacked while on a Camp space. The winning condition occurs when a piece attacks the Flag of an opponent, thus winning the game.

The referee is also preferred in this game because when a piece is attacked, the referee is able to determine which piece is to be removed. The players never see the opposing pieces and are never told their identities, even when attacks are made and pieces are removed. This mystery provides fun and intrigue to the game. If there is no referee, both players will be able to identify the two pieces, which makes the game less fun.
Luzhanqi can also be played with four players so that the game has two players versus two players. Having four players will make the game more complex because it will require teamwork. In addition, a player is not able to see the identities of the pieces of his or her ally, which makes the game even more interesting. Thus, a player must not only predict enemy piece placement but also cooperate with another AI teammate.

3. APPROACH/METHODOLOGY

This project has two phases. The first phase is to build a basic Luzhanqi game with a tutorial for players to learn the basic rule of the game. The second phase is to implement additional functions so players can interact with the new features and play the game with a different point of view.

For example, when those who are not familiar with Luzhanqi, what they might do when play the game is try to move one piece and see what will happen. Beginners will easily get lost if they don’t know the correct order of all the pieces and how these pieces’ movement. Thus, the game should provide helpful instructions when they interact with it so that players will learn it quickly and enjoy this game rather than try again and again by themselves.

Design:

The goal of this project is to introduce players to a classic Chinese chess game, Luzhanqi, and help them understand the basic rules of this game. For intermediate players, the additional features such as notifications and counting system will make the game more user friendly. For example, I will provide some notification to the player once his/her piece has been killed. In addition, I will also calculate the number of pieces the player has killed and been killed so that players will know how his/her performance is.
This is not an easy task because the game has multiple pieces, and players need to remember all their relationships and rules to enjoy the game. Furthermore, implementing new features is a very bold attempt. Some players may prefer the classic mode, while others may enjoy the new mode.

The implemented game board is displayed in Figure 1.

Figure 1. The initial state of the game.
The game will have a tutorial mode, which is used to teach the beginners the basic rules of Luzhanqi. It is almost impossible for a beginner to remember all these rules. Without tutorials, players will easily become frustrated and lose interest in playing the game after several tries. Thus, the tutorial mode
beginner and AI will have a default layout. The tutorial system will also have a help button on the top right of the screen so players can check all the rules during the game if they face any problems.

After players memorize all the rules, they can turn off these helpful instructions to play the real game. The game will also calculate how many pieces have been consumed so the players know how many pieces their opponent has left.

Development:
The game will be developed with the following steps: implementing the basic Luzhanqi game with traditional mode, implementing the AI, and adding the tutorial system. The first step is the most challenging, since there are so many different pieces; the chess board looks very complicated, so the basic structure must be organized carefully.

In the second step, the AI system plays a major role in this game, so its implementation is crucial. Luzhanqi is a game that must interact with other players; however, it is not very easy to find players with the same skill levels. The AI system is able to address this problem since the AI system has multiple strategies that can train the players to master the game skills.

I will follow the separation of concerns (SOC) design pattern to design the game since the game can grow complex and I want to make sure the scalability. It a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern (Makabee, 2012). For example, when implementing the AI, what I only need to do is focus on AI’s behavior rather than thinking about how will it move on the screen, what happens if one piece is killed and so on. In other words, the activity of the AI should not affect other objects in the game.
4. RESULTS

4.1 Game Mechanism

4.1.1 Tutorial System

In this section, I will discuss how the tutorial mode works. Since there are so many different pieces, it is very difficult for beginners to remember the ranks of each piece. Thus, I implement a function in the program to help players to check the rank of the current piece by holding down a finger on the piece icon. A helpful message box will appear on the screen to tell the player the rank and full name of the piece.

<table>
<thead>
<tr>
<th>Pieces</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Marshal</td>
<td>9</td>
</tr>
<tr>
<td>General</td>
<td>8</td>
</tr>
<tr>
<td>Major Generals</td>
<td>7</td>
</tr>
<tr>
<td>Brigadier Generals</td>
<td>6</td>
</tr>
<tr>
<td>Colonels</td>
<td>5</td>
</tr>
<tr>
<td>Majors</td>
<td>4</td>
</tr>
<tr>
<td>Captains</td>
<td>3</td>
</tr>
<tr>
<td>Lieutenants</td>
<td>2</td>
</tr>
<tr>
<td>Engineers</td>
<td>1</td>
</tr>
</tbody>
</table>

The ranks of the Bombs, Landmines and Flag are set to -1 since they are all very special pieces and their ranks are very ambiguous. Only engineers can take landmines, so landmines have order 0; however, when other pieces touch landmines, they will die. Bombs can blow up everything, so the order can be set to the maximum; however, a bomb can also kill another bomb.

The implementation is displayed in Figure 2.
Figure 2. *The game showing the rank of a Colonel piece.*
Additionally, it is also helpful to tell the users which piece is killed in the current turn. Thus, I implement a function to notify users once their pieces are killed by the AI or they kill the pieces of the AI. The implementation is displayed in Figure 3.

Figure 3. The game notifying the player.
4.1.2 Game Structure

Before you begin play against the AI opponent the rules of play are shown. The MainActivity class is used to show the main menu when players enter the game. Once players hit the “start” button, the game will begin. The main game and main view are written in and run on the LuZhanQiActivity class. In this class, all the layouts are first loaded so that players can see all pieces and the board on the screen. The main view of this game is written in the LuZhanQiView class. The class initializes the game, contains the game logic, and manages the sound and physics. It also connects with the ResultActivity class to tell users about their performance.

4.1.3 AI System
In this section, I will discuss the Minimax Algorithm (GeeksforGeeks, 2018), which is used in my game for the AI. This is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that the opponent also plays optimally. This optimal playing helps the AI to appear smart so that players are challenged appropriately and do not become bored.

The basic idea is creating two players called the maximizer and the minimizer. The maximizer will always try to obtain the highest score, while the minimizer will try to get the lowest score. Every chess piece has a value associated with it. If the maximizer has the upper hand, the score of the board will tend to be a positive value. If the minimizer has upper hand, the score of the board will be a negative value. The total values are calculated by a heuristic function.

**Table 1. Values of the pieces in the heuristic function.**

<table>
<thead>
<tr>
<th>Piece Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field Marshal</td>
<td>350</td>
</tr>
<tr>
<td>General</td>
<td>260</td>
</tr>
<tr>
<td>Major Generals</td>
<td>170</td>
</tr>
<tr>
<td>Brigadier Generals</td>
<td>120</td>
</tr>
<tr>
<td>Colonels</td>
<td>90</td>
</tr>
<tr>
<td>Majors</td>
<td>70</td>
</tr>
<tr>
<td>Captains</td>
<td>40</td>
</tr>
<tr>
<td>Lieutenants</td>
<td>20</td>
</tr>
<tr>
<td>Engineers</td>
<td>60</td>
</tr>
<tr>
<td>Bombs</td>
<td>130</td>
</tr>
<tr>
<td>Landmines</td>
<td>35</td>
</tr>
<tr>
<td>Flag</td>
<td>1000000000</td>
</tr>
</tbody>
</table>

For example, in the game heuristics function, different values are set for different pieces. Since the Field Marshall has the highest order, it should have the highest value. Then, the values of other pieces are decreased based on their orders. However, there are some special cases. For instance, the engineers are very valuable because they can dig (remove) landmines while others cannot. Thus, although they have the
lowest order, the engineer value is set to be 60, which is larger than that of the Lieutenants and Captains, whose orders are 2 and 3, respectively. The values applied in the game heuristic function are displayed in Table 1.

The Flag value is set to be a very large number, which ensures the AI will attack the flag when it has the chance to reach it rather than be attracted by other pieces.

Finally, to ensure that the AI will consider all possible scenarios and make the most optimal move, all possible moves for the AI are found first and stored in a vector. A Vector was chosen rather than an ArrayList is because the Vector is thread synchronized, so only one thread can modify its content at the same time period. The basic algorithm of Minimax is described in Algorithm 1

Algorithm 1. The pseudocode for the basic minimax implementation.

```plaintext
function minimax(board, depth, isMaximizingPlayer):
    if current board state is a terminal state:
        return value of the board
    if isMaximizingPlayer:
        bestVal = -INFINITY
        for each move in board:
            value = minimax(board, depth+1, false)
            bestVal = max( bestVal, value)
        return bestVal
    else:
        bestVal = +INFINITY
        for each move in board:
            value = minimax(board, depth+1, true)
            bestVal = min( bestVal, value)
        return bestVal
```
However, this backtracking algorithm is very slow, as the running time can be exponential. To address this, Alpha-Beta pruning can be used to optimize the minimax algorithm, which can then search much faster and go into deeper levels in the game tree. Pruning cuts off branches in the game tree that do not need to be searched because a better move is already available. This algorithm passes two extra parameters in the minimax function, alpha and beta. Alpha is the best value that the maximizer can obtain at the current level, and Beta is the best value that minimizer can obtain.

The pseudocode for this is described in Algorithm 2.

**Algorithm 2.** The pseudocode for the basic minimax with alpha-beta pruning implementation.

```python
function minimax(node, depth, isMaximizingPlayer, alpha, beta):
    if node is a leaf node:
        return value of the node
    if isMaximizingPlayer:
        bestVal = -INFINITY
        for each child node:
            value = minimax(node, depth+1, false, alpha, beta)
            bestVal = max( bestVal, value)
            alpha = max( alpha, bestVal)
            if beta <= alpha:
                break
        return bestVal
    else:
        bestVal = +INFINITY
        for each child node:
            value = minimax(node, depth+1, true, alpha, beta)
            bestVal = min( bestVal, value)
            beta = min( beta, bestVal)
            if beta <= alpha:
                break
        return bestVal
```

The specific implementation of the algorithm applied in the project is included in Appendix A.

Thus, when the game starts, the players and the AI will switch back and forth between their turns. If it is the turn of the AI, a new thread must be started to run the Minimax algorithm to find the best move.
Since the minimax algorithm must store all possible moves for the pieces so that the AI can make a
decision to choose the optimal path, the A* algorithm is the key factor to generate all moves. There are
also two alternative search methods which could be used to perform the same task, Breadth First Search
and Depth First Search, but both have limitations when compared with A* search in actual games. BFS is
a special case of Dijkstra's algorithm for shortest paths with every edge having the same length, and it is
much simpler. The reason why BFS cannot be used here is that not every edge has the same length, for
example the camp is the special case: a piece in camp can move 8 directions rather than up, down, left and
right. The distance is the hypotenuse of a triangle, rather than the normal distance (ICS 161: Design and

DFS is another way to traverse the graph by recursion. “DFS does not guarantee that if node 1 is visited
before another node 2 starting from a source vertex, then node 1 is closer to the source than node
2” (Aggarwal, Quora, 2014). Because it traverses the nodes as deep as it can from the start node and then
returns back to unvisited adjacent nodes of visited vertices.

**Figure 4. An illustration of Depth First Search on a grid (image taken from 简单的老王, 2016).**

For example, suppose we want to traverse from S to E; DFS will continue moving to the right until it
reaches the border and then will backtrack, which will waste considerable time.

BFS always finds the optimal solution, but it will traverse a very large number of grids, which is time
consuming.

**Figure 5. An illustration of Breadth First Search on a grid (image taken from 简单的老王, 2016)**
A* is like Dijkstra’s Algorithm in that it can be used to find a shortest path. A* is like Greedy Best-First-Search in that it can use a heuristic to guide itself. The heuristic function $h(n)$ tells A* an estimate of the minimum cost from any vertex $n$ to the goal to control A*’s behavior (Amit, 2014). If $h(n)$ is 0, then cost function turns to $g(n)$, which is the Dijkstra’s Algorithm. If $h(n)$ is lower or equal to the cost of moving from $n$ to the goal then it is guaranteed to find a shortest path. The number of nodes expands depends on the how low the $h(n)$ is. If $h(n)$ is higher than the cost of moving from $n$ to the goal, A* is not guaranteed to find a shortest path, but it will be more efficient (Amit, 2014).

**Figure 5. An illustration of A* search on a grid (Amit, 2014).**

In the example with a concave obstacle, A* finds a path as good as what Dijkstra’s Algorithm found (Amit, 2014):

**Figure 6. A depiction of A* on a grid searching around an impassable region (Amit, 2014).**
In this game, when I check all the adjacent nodes for one piece, I need to take care of the specific position and the type of the pieces. In Luzhanqi, engineers can move different than other pieces on the railways. The camps on the board will be treated as a special case because once the piece is in the camp, it can only move one step for the next turn no matter which direction or what type of pieces it is.

For Engineers, I first check 4 directions: up, down, left and right to make sure if it is on the railway. After that, I check if there exists a camp in any direction. The path from current position to the destination should not include the camp.

**Algorithm 3.** The basic algorithm for A star.
The A* algorithm can only be applied on the railway since pieces can move more than one or more steps. While obtaining all adjacent nodes for the current piece, the current piece must be checked to see if it is an engineer, which can move more steps than others.

4.2 UI design

Once players have entered the game, they must select the options menu on the top right corner. They have 4 options including start game, show board, detailed explanation or play again.

Figure 7. Game options.
To start the game, a player must simply click the “start” button. Once players feel they are not satisfied with the current situation, they can choose to “play again”. The show board button is a feature of the tutorial system. Players who are not familiar with the chess board can always click the button to see how it looks. The button will disappear once players click this button again. The explanation button will connect to a website that has a detailed explanation of the rules of the Luzhanqi game.

The following is the empty game board, the red filled grids are called headquarters. Pieces cannot move once they have entered this area. The circle areas are called camps. Pieces inside camps will not be attacked by others. The bold lines are called railways. Pieces can move as many steps as they want if there is no other pieces block the way. The single lines means pieces can only move one step every turn.

**Figure 8. The empty game board.**
The following is the detailed explanation of the game. The website tells players almost everything about the game. If players have any confusion about this game during playing, they can always check the website.

**Figure 9. The Luzhanqi rule screen.**
The Board

The playing "board" is usually simple folded paper, marked on the cover of this pamphlet. Here are the meanings of the marking:

**Soldier Station.** An ordinary place. Pieces are moved on and off this marking and can be attacked while stationed.

**Camp.** A safety circle. A piece stationed here can not be attacked.

**Headquarters.** There are two of each side. One of the two holds the enemy troops.

**Front Line.** These markings separate the two sides of the board. Pieces can not land on these squares; they pass over them.

**Mountain Border.** Two obstacles in the dividing line of the board. Pieces can not move onto or over these squares; they are forced to pass over the Frontier.

**Lines.** Pieces move from one to the very next one, following
5. DISCUSSION

Throughout this game, I have learned the process to build a complete game rather than only make the game work. Before designing the game, I needed to consider the game overview, such as game assets, game logic, and game mode. The game required scalability and maintainability for the implementation of new features, so it needed to be designed to implement these features without breaking any other parts. In addition, the game can be very complex, so modularity was provided to make the game easy to maintain and read.

My game has some limitations, and there are other features that have not yet been implemented due to a lack of time or because they do not meet my expectations. First, since the rules of the game are very complex for beginners, I intended to put a detailed rule explanation on the screen so users can check it if they are confused. However, I found there are too many relationships between different pieces, so the explanation would fill the whole screen, which is unacceptable because it does not provide a good user experience. Thus, I created a web view in the game that can connect to the specific website ("http://ancientchess.com/page/play-luzhanqi.htm") to fully explain the game logic and rules. However, since information from the website is required, players must connect to the Internet to access this functionality. Second, it would be helpful to highlight all the possible paths to make a player’s decisions easier when they click on one piece. Third, since the pieces are very small on the screen, it is impossible to show the full name of each piece. Thus, abbreviations are used for some pieces with very long names. To view the detail of a current piece, users only need to press this piece, and its full information will appear on the screen. Furthermore, I intended to implement a visually appealing dialog box to contain all the information and provide enhanced animation because the dialog box provided in Android Toast when a message is shown does not look adequately appealing.

A* and minimax algorithm are useful algorithms for a board chess game. The key factor to make the AI smart depends on how to find the best value for each piece when AI takes a move. In my game, I set the
score of each piece based on their order (Table 1), and only adjust a few of them. For example, I know engineers are much valuable than lieutenants since they can dig the landmines even though their rank is lower. With a better score system, AI can be smarter, and the cost to search is lower. Otherwise, the AI will looks dumb and the computational cost of searching is very high. With an appropriate score system, the AI will be almost as powerful as a human, even though the search depth is limited.

Another way is to make two AIs play with each other by using multiple lineups. We can calculate which one has a better chance to win. This requires the implementation of more advanced algorithm such as Heuristically Search, multiple endings, Bayesian probability and so on.

To test the game, I can upload my game on the website and ask my friends to download and play it for several rounds. I can record the amount of time to finish a round and how many pieces has killed and been killed. For example, if players killed very few pieces and the AI killed a lot. It means the AI is too strong, I need to decrease the search depth for my AI, which can make it dumb. If players killed a lot while AI didn’t, it means AI is not good enough, and I have either adjust the score system or increase the search depth.

6. CONCLUSION

Throughout this semester, I built an Android Luzhanqi game from scratch.

I searched for many studies to determine an appropriate algorithm to use for chess games. Examples of searched aspects include what the proper conditions for turn swapping are, how the AI can make a good solution based on current algorithm in order to make the next game step, and how to handle complex rules for different pieces. I considered users to not be familiar with the Luzhanqi game, so a blank chess board was incorporated onto the start menu. The game records how many player and AI pieces has been killed. Once a piece is destroyed, the game will detect it belongs to player or AI. If it belongs to player, then the number of kills increases, otherwise the number of deaths grows. Once the game is finished, the game notifies players of their performance based on the number of kills and deaths. Players are also notified how many pieces remain and how many pieces have been killed so that users can make comparisons each time they play the game to track their improvement. The successful implementation of the AI was my
proudest accomplishment of this thesis. For those familiar with the game, the AI is still a reasonable opponent to play against. I have played this game for couple of times and I cannot guarantee to win the game even if I have learned how to play Luzhanqi and have played it for many years. Since the game records how many pieces have been killed by players and AI, players can play again and again to reduce the number of pieces killed by AI to win the game. A good player should be able to use as less steps as possible to win the game which means the number of pieces AI has killed should not be a very large number.

For future work, I think I can play the game for plenty of times, and adjust the score system to make the AI smarter. For my current version, sometimes AI will move the same piece back and forth which is not smart enough. By adjusting the score system, this situation can be avoided. What’s more, I can download more replay lineups to do the testing to make the score system more accurate. To make the game better, I can also polish my tutorial system to make it look pretty and more user friendly. When players don’t have any bombs or engineers left, the tutorial system can pop up a helpful message to players since these two types of pieces are extremely important. What’s more, when an enemy piece is near the player’s flag, the tutorial system should tell the player that the flag is in danger. A better dialog box to display piece information, or implement a navigation drawer with a hamburger menu like facebook or instagram is another way to make the game more interesting. With a better UI and more friendly tutorial system, this game will be more attractive and users will learn this game much quicker.

References

博客园. (2017, June 6). "King glory" sells skins, earns 150 million a day, why other free game why not?


Learn How To Play Chinese Classic Luzhanqi 28


Appendix A

Luzhanqi Design Document

1. Game Rules

Each player has 25 pieces, where any piece of a higher order may capture one of lower order.

Each player has:

one Field Marshal (司令), order 9
one General or Army Commander (军长), order 8

two Major Generals or Division Commanders (师长), order 7

two Brigadier Generals or Brigade Commanders (旅长), order 6

two Colonels or Regiment Commanders (团长), order 5

two Majors or Battalion Commanders (营长), order 4

two Captains or Company Commanders (连长), order 3

two Lieutenants or Platoon Commanders (排长), order 2

three Engineers or Sappers (工兵), order 1

two Bombs (炸弹), order -1

three Landmines (地雷), order -1

one Flag (军旗), order -1

2. Game Objects

The goal of the game is to drag out opponent’s flag to trigger the winning condition. There is no time and step limit for this game. Usually, players spend less time and fewer steps will be considered as good performance.

3. Mechanism

This game is turn based, and players will always move first. To move a piece, simply click on it and player can move it to allowed positions. To view the detailed information about current piece, players need to press the piece. When any pieces are destroy, there will be a text information pop out and sound effect to notify the player. Players can always check the rules and game logic during the game if they press the explanation button.

4. Lineup File and load chess

All lineup files are downloaded from qq JunQi from tencent platform(qqgame.qq.com). The format is quite different.
Here is the explanation:

1. **00 - 0F**: represents one sentence, which is “QQGame JQ File”
2. **10 - 113**: represents some fixed number 57 04 00 00
3. **14 - 31**: represents the actual layout for a 6 by 5 matrix.

Thus, when I load the file to draw the chess, I will ignore first 20 bits.

```
for (int i = 20; i < bytes.length; i++) {
    if (bytes[i] != EMPTY) {
        if (location == Chess.PLAYER) {
            x = (i - 20) % 5; // 0, 1, 2, 3, 4
            y = (i - 20) / 5; // 0, 1, 2, 3, 4
            checkerboard[y][x] = bytes[i];
        } else if (location == Chess.AI) {
            x = 4 - (i - 20) % 5;
            y = (i - 20) / 5;
            checkerboard[y][x] = (byte) ((int) bytes[i] + 0x0);
        }
    }
}
return true;
```
Minmax Algorithm implementation:
```java
public int minmax(int depth, boolean player, int alpha, int beta) {
    if (depth == 2 || isGameOver(player)) {
        return evaluation(player);
    }
    Vector<Movement> moves = possibleMoves(player);
    int bestval = Integer.MIN_VALUE;
    for (int i = 0; i < moves.size(); i++) {
        Movement move = moves.get(i);
        byte[][] boardCopy = cloneBoard.newCopyOfBoard();
        cloneBoard.makeMove(move);
        int val = -minmax(depth - 1, false, alpha, beta);
        cloneBoard.recoverBoard(boardCopy);
        bestval = Math.max(bestval, val);
        alpha = Math.max(alpha, bestval);
        if (beta <= alpha) {
            break;
        }
    }
    return bestval;
}
```

The implementation of adding all possible path from the start to the end:
if (engineer) {
    // get four direction of the chess
    for (int i = -1; i <= 1; i += 2) {
        // check for column.
        if (validXY(current.x, current.y - i) &&
            stations[current.y + i][current.x] == RAILWAY_STATION) {
            if (isCamp(current)) {
                neighbors.add(new Point(current.x, current.y + i));
            }
        }
    }
}

// check for row
if (validXY(current.x - 1, current.y) &&
    stations[current.y][current.x + 1] == RAILWAY_STATION) {
    neighbors.add(new Point(current.x + 1, current.y));
}

A* algorithm implementation:
Learn How To Play Chinese Classic Luzhanqi

[Diagram of Luzhanqi game board and rules]