International Journal of Computer Science and Information Technology

ISSN: 3005-9682 (Print), ISSN: 3005-7140 (Online) | Volume 4, Number 3, Year 2024
DOI: https://doi.org/10.62051/ijcsit.v4n3.06
Journal homepage: https://wepub.org/index.php/IJCSIT/index

Digitising a Turn-Based Strategy Board Game: Implementation of Spellcaster Using the Mini-Max Algorithm

Yukai Wang

[1] School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK

[2] School of Computer Science and Technology, Ocean University of China, Qingdao 266100, China

ABSTRACT

Spellcaster is a turn-based strategy board game for two players, involves players casting spells by making sequences of gestures, aiming to defeat their opponent through different kinds of spells within a turn-based system. The research succeeded in the implementation of Spellcaster as a 2-player game system, with an AI opponent who can make reasonable decisions. The digitised game is developed using the LibGDX game engine because of its cross-platform features and high editability; and implemented in the Java language, using java's object-oriented features to improve programming efficiency. The game's AI opponent is implemented using the Mini-Max ideas, with an evaluation function that focuses on the gesture selection of the current turn.

KEYWORDS

AI Strategy Games; Mini-Max Algorithm; Board Game Digitisation; Spellcaster; LibGDX

1. INTRODUCTION

1.1. Aims and Objectives

Spellcaster [1] is a turn-based strategy board game for two players. In Spellcaster, the players, as wizards, can choose different gestures each turn and have these gestures combined into a series of gesture sequences to form spells. The goal of the game is to defeat an opponent through different types of spells within the framework of a turn-based game. The game features a variety of spells, such as summoning Monsters, applying effects such as blindness and paralysis, and direct attacks, each of which requires the player to plan a specific sequence of gestures in advance.

The project requires digitising the Spellcaster, to implement its corresponding functions through a programming language, and to develop an artificial intelligence AI for the game, able to act as a competent opponent to play Spellcaster with a human player.

beyond delivering the digital game as a product and evaluating its usefulness, the implementation of a board-game and an AI is a very useful learning process in the design of complex systems, supported by an AI component. Lean et al. points out that, from an academic perspective, the process of digitising board games is a good mode of learning, providing engaging research cases for university courses [2]. By implementing a game, it is possible to synthesize the study of algorithms, software engineering, artificial intelligence and other areas of teaching. This fits the motivation of this project.

The learning objective of the project is to learn and understand in-depth the idea of building Artificial Intelligence (AI) in strategy games and the methods of game state evaluation through the digital implementation of Spellcaster. Through the programming implementation of the game rules and the development of AI opponents, the project explores a learning process to become more proficient in the use of programming languages and the writing of AI algorithms. It is also an opportunity to explore playful learning, particularly in terms of developing players' logical thinking and decision-making skills, although this is not currently the focus of this project.

Building up to the project aim, this project will be mainly divided into the following objectives:

(1) Design an intuitive user interface, including a clear menu layout that shows the current game status and turn history to ensure the player's gaming experience.

(2) Implement all the basic rules of Spellcaster game logic, including turn-based game loop, interaction between spells, spells triggered by different gesture sequences, etc.

(3) Develop an effective artificial intelligence. Using the Mini-Max ideas [3], analyse the current state of the game and develop AI opponents that exhibit a reasonable level of difficulty in the game to make plausible decisions.

1.2. Research Results

Based on the learning and discussion, the Java language and LibGDX game engine were chosen as the main development tools for this project in the digital realisation of the Spellcaster board game. Java language was chosen for its powerful object-oriented programming language features that can well help to decompose the various elements of the game, while LibGDX was chosen for its highly cross-platform nature and suitability for prototyping from the ground up.

In AI selection, Mini-Max ideas [3] are mainly used in the project, because for a two-player turn-based game like Spellcaster, where the game history information is transparent, the Mini-Max can analyse the state of the game and performs well. Analogy is used to try to abstract the Spellcaster game as a kind of Chess Game like Gomoku, and the evaluation function can be applied to the AI of the Spellcaster.

Design patterns are introduced throughout the implementation, such as Observer Pattern and Factory Pattern, for better compliance with OOP design principles, a more stable code structure, making future modifications and extensions easier, and making modularity based on abstract classes makes code more stable.

2. BACKGROUND

2.1. Motivation

As pointed out by Lean et al., digital board games have good appeal from an academic perspective, motivating and enhancing the learning experience [2]. Chen also stated that with appropriate learning strategies, the development of digital computer games can provide a more interesting and challenging learning environment for education [4]. Digital board game platforms not only serve as entertainment tools, but also power educational and research tools. Therefore, in this project, the digitisation of Spellcaster is a good learning process to learn computer aspects such as programming and AI techniques in an integrated way.

2.2. Spellcaster Game Rule Explanation

In Spellcaster, the player needs to choose gestures for both hands each turn, and use the sequence of gestures combined over multiple turns to form a spell. Deal damage and effects by casting these spells on the opponent to win.

2.2.1. Sub-section Headings

Below is an explanation of the basic elements in Spellcaster:

Wizard: The player takes on the role of a wizard in the game, with an initial 15 health point. As a wizard, the player can cast spells independently with each hand, as well as powerful spells that require both hands to work together. During each turn, the player can make a gesture with each of the left and right hands, directly stab opponents (physically deal 1 damage) or choose to do nothing.

Gestures: There are three types of gestures that the player can perform, including five one-handed gestures: fingers (F), palm (P), snap (S), wave (W) and digit pointing (D); one two-handed gesture (i.e., both hands must make the same choice): clap (C); and stab (a physical attack that causes 1 damage), and skip (doing nothing).

Different combinations of gesture sequences can cast different spells, for example: three consecutive turns of fingers (F) can cast ParalysisSpell (F-F-F), which restricts the opponent's gesture choices for the next turn.

Spells: In Spellcaster, spells are classified into 4 types:

PROTECTION SPELL: This class of spells is mostly defensive effect, capable of applying shields, healing or counter damage.

SUMMON SPELL: This is a special class of spells that summon different Monsters.

DAMAGING SPELL: This type of spell is capable of dealing damage (to a wizard or a monster). Some damaging spells have special effects, such as: FireBallSpell instantly kills Ice Elemental (a monster) in addition to dealing 5 points of damage.

ENCHANTMENT SPELL: These spells have the ability to apply a variety of effects, such as limiting the opponent's gesture choices, gaining an extra turn, and so on.

Monsters: Monsters are summoned by Summon spells and have their own attack power and health point; their target is chosen by the controller.

2.2.2. Game Mechanics

Turn-based: Spellcaster is a turn-based game. Each player takes a turn. During each turn, the wizard can choose one gesture for each of the left and right hands. After several turns, the left and right hands will each form a different sequence of gestures, which may form a spell. An example of a spell formation is shown below in Figure 1:

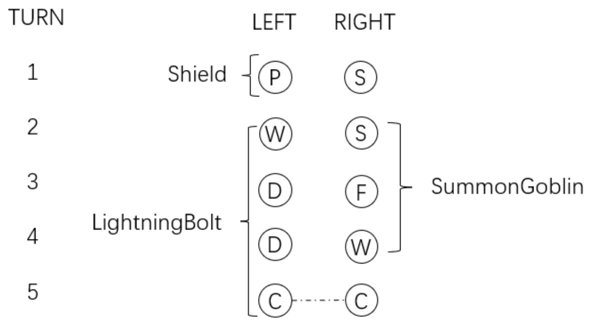


Figure 1. Turn-based gesture inputs and form spells

Spell Interaction: Spells can interact with each other in different ways. For example, most spell damage can be defended by ShieldSpell, or can be countered by CureLightWoundsSpell. Also, some of the Enchantment Spell's effects are invalidated by the RemoveEnchantmentSpell. The following Figure 2 shows the ShieldSpell interaction example:

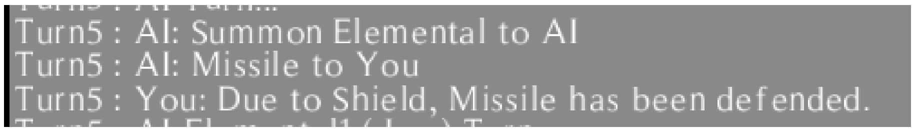

Turn5 : AI: Summon Elemental to AI
Turn5 : AI: Missile to You
Turn5 : You: Due to Shield, Missile has been defended.

Figure 2. ShieldSpell defends damage from MissleSpell

(Figure from a later implementation)

Spell Overlap Mechanism: This is one of the core mechanics in Spellcaster and can be very strategic. Each spell requires a different sequence of gestures. If the gesture sequences required for two spells have overlap, players can share these gestures when casting the spells consecutively without needing to input them again. That is, certain parts of a gesture sequence can simultaneously belong to the sequences of different spells.

For example, ConfusionSpell sequence is D-S-F, while SummonGoblinSpell sequence is S-F-W, then by choosing the W gesture (which creates D-S-F-W) on the next turn after casting ConfusionSpell, SummonGoblinSpell can be cast directly. The following figure 3 shows the effect of Overlap.

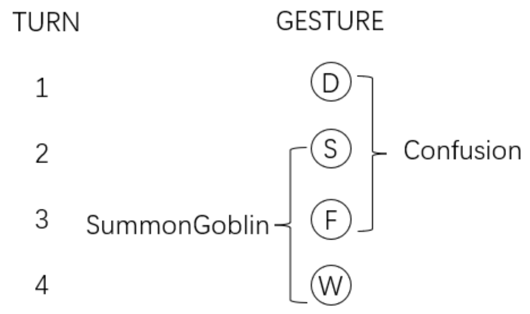


Figure 3. Overlap in gesture sequence

2.2.3. Minor Revisions

In the original version of the game rules, there are some spell interactions are not clearly defined. Therefore, during the actual implementation process, the following changes were made to ensure proper interaction in the game:

(1) On the turn a monster is summoned, the target specified by the right hand is the target of Monster's attack (except Elemental).

(2) Elemental (a monster) deals damage to all characters (including its controller).

(3) When both players cast AntiSpell at the same turn, the effects counter.

(4) The RaiseDeadSpell effect has been modified to: released target removes all negative effects and restore all health points.

**This is because in the original rules, RaiseDeadSpell effect is to resurrect a friendly character, but the rules do not define a character's "near-death state". When a player dies, the game ends. Therefore, in the original rules, this spell has no valid target to cast.

2.3. Strategy Game Digitization

The implementation is undoubtedly the key to the process of strategy game digitisation. When digitising these games, the key challenge is how to accurately and effectively represent the original strategy on a computer platform. Therefore, the key to solving this challenge lies in two questions: firstly, how to represent the rules of the game on a computer? And secondly, how to encode game strategies in AI?

2.3.1. Strategy Game Rules and Balance

The translation of rules to digital platforms is not just a simple translation process but requires consideration of how these rules can be redefined and adapted in the new environment to maintain the core character and appeal of the game [5]. For Spellcaster, the implementation of the spell interaction system within the rules was a key factor in ensuring game balance. An example is the interaction between the Protection Spells and the Damaging Spells, ShieldSpell needs to effectively block damage from MissileSpell. Good implementation of interactions can be very effective in digitising the rules of the game.

Zin [6] provides another major element of the conversion of a board game to a digital game that the complex game rules embedded in board games and the interaction between different objects, cards, etc. is one of the areas in which computers specialise. For Spellcaster, the key to the conversion of the rules is to allow the programme to better understand the spells corresponding to the gesture and the gesture sequence, and to understand the connection between the two. It is necessary to use sensible data structures to store the gesture and the Spell, such as using Map's key-value pairs to store both.

2.3.2. Strategy Game AI

Robertson and Watson [7] noted that a key challenge for AI setups in strategy games is the ability to accurately recognise the possible strategies of the opponent. This ability is crucial for AI; by analysing the opponent's behaviour and strategy, AI can adjust its own strategy more accurately. The key for Spellcaster AI lies in recognising the opponent's current gesture sequence and predicting the spells that the opponent is ready to release and dealing with them in a timely manner based on this information. For example, if recognising the current sequence under which the opponent's next turn can release CauseLightWoundsSpell, ShieldSpell needs to be released in time to defend it.

Hoang, Lee-Urban and Muñoz-Avila, on the other hand, introduced the application of the Task-Method-Knowledge (TMK) model [8] in coding strategy game AI. According to the model, AI for strategy games hinges on three steps: understanding the tasks that need to be accomplished in the current game state, mastering the specific methods for performing the tasks, and acquiring the relevant knowledge needed to perform the tasks. For Spellcaster's AI, there is only one specific way to perform a task - choosing a gesture. The key is to understand the current game state, understand the spells that your opponent may release, and counter or attack.

2.4. Playful Learning and Serious Games

2.4.1. Playful Learning

Playful learning is an important direction in the field of education that increases motivation and engagement in learning by incorporating play elements into the learning process. The Harvard Graduate School of Education states in its "Askwith Forum" [9] that playful learning is effective in integrating exploratory learning into the educational environment, and while it is not always equally fun for all learners, it provides a freer and more creative learning space for learning.

This is reflected in this project. Digitising the Spellcaster game is not simply a matter of converting the rules and gameplay into an electronic format, but more importantly it opens a creative learning space through this process, where both the player and the implementor can learn and innovate. Within the learning space, users are free to experiment with different strategies in an environment free from external pressure. This way of learning encourages players to be creative and learn strategic thinking and decision-making skills through practice. At the same time, for developers, the process of realising this project is a creative learning journey. implementor, through the project, could learn and understand programming techniques through practice, and to explore the use of AI in strategy games. It is a good expression of playful learning.

2.4.2. Serious Games

Similar to playful learning, the use of serious games in education is gaining increasing attention. Dicerbo notes that they are not only used to develop knowledge and skills in specific areas, but also help to form professional identities and values [10]. Interest in learning games has continued to grow over the years, with the initial focus being on how to engage and sustain students' attention through new methods. Yu also notes that as people began to explore the potential of games for student engagement, they found that games fit well with many theories of learning [11]. Serious games collect large amounts of data about player behaviour and analyse it through sophisticated statistical models, which allows teachers and students to obtain reports on knowledge, skills, and other attributes based on behavioural patterns in games.

For this project, the implementation of digital version of the strategy board game can have a positive educational impact on the implementor in a traditional educational setting. By applying the idea of serious games, implementors are not only able to learn and apply technologies such as artificial intelligence and programming languages in the game design and development process, but also gain a deeper understanding of game mechanics and the importance of strategic thinking.

2.5. Game Mechanics

2.5.1. Core Game Mechanics in Digital Strategy Games

Digital strategy games provide a rich depth of strategy through a complex set of rules and mechanics that push players to think, plan, and make decisions. And good game mechanics can greatly deepen the depth and complexity of strategy games. It allows players to feel the importance of strategy development.

Turn-based Process: many digital strategy games use a turn-based process in which players take turns executing actions, a mechanic that emphasizes the importance of forward thinking and strategic planning [12]. Turn-based processes provide a clear rhythm and structure to the game, allowing players time to make thoughtful decisions and anticipate their opponents' likely actions. While this loop promotes deep strategic thinking, it also provides the game with the advantage of being easy to manage and track. One of the core mechanics of Spellcaster is the turn system. In particular, Spellcaster is relatively transparent about the information in each turn. During each player's turn, they must strategies based on the current state of the game, choose gestures to combine into spells, and try to anticipate and interrupt their opponent's plans.

Variety of Choices: One of the core mechanics of digital strategy games is the provision of complex decision trees that require players to make a series of choices that will directly affect the course of the game and the final outcome [13]. Similar to many strategy games, Spellcaster features a complex spell system that provides players with a wide range of strategic choices. Each spell requires a specific series of gestures to cast, and these spells include, but are not limited to, summoning monsters, attacking opponents, defences, and special enchantments. The variety of spells requires players to have an in-depth understanding of each spell's effects and gestures in order to make the best choices based on the matchup. Additionally, the effects of certain spells can be stacked or counteracted by other spells, adding to the complexity and interactivity of the game.

Multiple Bonus for A Single Action: at the same time, Arnab et al. points out that many digital strategy games employ efficiency-maximizing strategies that add strategic depth by allowing a single player action to trigger multiple effects [14]. Overlapping of gestures is a core mechanic of Spellcaster, which allows the player to trigger multiple spells in several continuous turns with only one or two gestures. A detailed description of overlapping will be discussed later. This mechanic successfully enhances the strategic nature of the game, as players need to consider not only the strategic value of individual spells, but also how to maximize the efficiency of their actions by combining and "overlapping" gestures during gameplay.

2.5.2. Simplify Game Mechanics through Computer Games

Zagal, Rick and Hsi noted that through computers, automated processes such as resource management, damage calculations, and automated application of status effects reduce the cognitive load on players [15], allowing them to focus more on strategic decisions. At the same time, in the process of board game digitization, some of the "metaphors" that exist in board games should be implemented, a simple example being that when the game interface gives the image of a dice, the player should understand that a new turn has been reached [16], the realization of these metaphors helps the player to be able to get into the atmosphere of the game better.

Optimized user interface design also plays a crucial role in streamlining game mechanics. Intuitive user interface design effectively reduces the difficulty for players to learn the rules of the game, while good user interface design improves overall player satisfaction and enhances player immersion [17]. These optimizations not only make the game more accessible, but also provide an easier entry point to the game for all types of player groups.

2.6. Game Engines

2.6.1. Unity

Unity (Unity3D) is one of the most popular game production engines on the market today. The framework of Unity engine is mainly constructed by several basic frameworks, including the core system, scripting system, scene management module, etc. The working mode of Unity engine is mainly composed of four layers: the application layer, the game scene, the game object and the game script to work together to complete the development of the game [18]. Unity engine has a comprehensive API documentation and an active developer community with easy access to materials, rendering, and other resources [19]. Unity's high degree of flexibility and ease of use makes it ideal for rapid prototyping and iteration. Its visual editor and rich set of prefabricated components (Prefabs) greatly simplify the game design process, especially for board games, which are a relatively simply structured type of game. However, for Spellcaster, which focuses on game logic and AI development, there are some advanced aspects of Unity that will not be used in this project, which may lead to a waste of resources.

2.6.2. LibGDX

LibGDX is not a direct game development software, but rather a Java-based game development framework. It has highly flexibility in programming compared to other engines with high integration. Although the high integration of other engines reduces the learning curve of programming, LibGDX flexibility allows for more freedom and fine-grained control over the code framework. The overall architecture of LibGDX is modular and is made up of a number of components that are independent of each other, but work in concert, including the core libraries, the back-end interfaces, and the platform-specific modules [20].

Meanwhile, LibGDX is developed based on the Java language, which allows for easy use of JVM features, such as Code Hot Swapping and WYSIWYG (What You See Is What You Get) at runtime. Overall, LibGDX modular architecture is well suited for the development of strategy board games, which often require a flexible combination of components to simulate different game rules and logic; however, LibGDX, as a coding-dependent framework, is not easy to learn.

2.6.3. Godot

One of the core features of the Godot engine architecture is the combination of a scene system consisting of nodes. In Godot engine, nodes are the basis of all functions, each node can exist separately and perform certain simple functions. A node tree, which is a combination of multiple nodes in the form of a tree, is called a scene. a scene is the basic unit in the Godot engine: an object, a physical value UI can all be called a scene [21]. Godot's scenarios can be saved as reusable templates, which facilitates modular game design for developers and the creation of complex game structures through scenario instantiation. The engine's lightweight nature and flexibility make it ideal for independent developers and for educational purposes, providing a practical and powerful toolset for game development. The Godot engine has certain advantages in digital board game development. Its node and scene system greatly simplifies the creation and management of game elements. In a board game, each piece or game block can act as a separate node, which is easy to program and reuse. Godot's community and resource base is relatively small compared to some of the more established commercial engines, which may mean that finding solutions or getting support for specific problems encountered during development may be more challenging.

2.6.4. Unreal

The Unreal Engine, a more powerful game development environment, is built on the C++ language and is designed with a highly integrated architecture that includes a number of modules that work together, such as the core system, animation mechanisms, and blueprint visual scripting. The Unreal Engine is based on the C++ language, which not only brings performance benefits, but also reduces the difficulty of development through tools such as the blueprinting system, allowing developers to achieve intuitive editing and immediate feedback. However, Sharif et al. also mentions that although the Unreal engine is more advanced in many aspects such as visual effects and terrain creation [19], its overly large engine system is not well suited to the development of smaller games. The Unreal Engine's blueprinting system simplifies the programming process and is an advantage for implementing complex game rules and interactions, but the drawbacks of the Unreal Engine are also obvious, and the overhead associated with learning and using such a powerful engine can result in a project that doesn't end up being completed successfully.

2.6.5. Comparison and Summary

A comparison of game engines is necessary, table 1 below shows a comparison of some of the features of the above different engines. The choice of a game engine should not only take into account whether it supports the required technical features, such as AI algorithm support and programming support capabilities, but also factors such as the engine's development efficiency, and community support. These considerations are especially critical for the digitising of strategy board games, which not only need to reproduce the depth of decision-making, but also bring good experiences to the player in terms of visual presentation and interaction.

Ali and Usman point out, a particular concern when developing complex strategy games is the ability of the engine to support complex AI algorithms as well as efficient rendering [22]. The LibGDX engine, for example, is based on a Java package and is able to provide an atmosphere of prototyping and the ability to iterate quickly [19]. This means that implementor can quickly build and test game prototypes and easily make modifications and optimisations, thus effectively shortening the development cycle.

Table 1. Comparison of features of the selected engines

Features/Engine	Unity	LibGDX	Godot	Unreal
Programming Languages	C#	Java, C++	GDScript, C#, C++	C++, Blueprint
Development Environment	Unity Editor	any Java IDE	Godot Editor	Unreal Editor
Community Support	very active	active	very active	very active
AI Algorithms	Good support	More basic support, easy to extend	basic AI function	advanced AI function
Algorithm Efficiency	good	moderate	moderate	good

Based on the comparison before, LibGDX is relatively cross-platform, and it has a high flexibility. Therefore, LibGDX engine is used in this project.

2.7. Game Computing Platforms

Cross-platform is a very important requirement. It is necessary to analyse and discuss the advantages and disadvantages of different platforms.

2.7.1. Desktop

For strategy games like Spellcaster, the Desktop platform is an important implementation platform. Based on its powerful processing capabilities and higher display resolution, the Desktop platform is able to provide a richer representation of complex game logic and detailed graphics. In addition, players on the Desktop platform tend to favour deeper and more strategic games, which is highly compatible with the characteristics of Spellcaster. The development environment on the Desktop platform is also mature and widely supported, with a variety of development tools and libraries to meet the needs of the game development process [23]. Especially for this project, which uses the Java language and the LibGDX framework, these tools not only support cross-platform development, but also make it easy to extend the game from Desktop to mobile devices, thus achieving true multi-platform compatibility.

2.7.2. Mobile Devices

Mobile platforms such as mobile phones and other portable devices are also a good choice considering the ease of development and play. Perrin et al. point out that on mobile platforms, although iOS has a more stable development environment while Apple has been increasing the openness of its interfaces in recent years, its closed-source nature and higher development costs still pose limitations to development [24]. Moreover, Android is a good choice due to its open-source nature and wider market share. nature and higher development costs will still impose limitations on development; while Android is a better choice due to its open-source characteristics, diverse device configurations and wider market share. In addition, the Android platform has a rich community environment, which is especially important for small-scale game development.

2.7.3. Online Platform

BoardGameArena: BoardGameArena's strength as a browser-based online digital board game platform lies in its rich library of games and the ability for online instant play. During COVID-19, when many players were unable to meet face-to-face to play traditional physical board games, some turned to the fully digital versions of board games available on these platforms to stay socially connected with others [25]. This exemplifies BoardGameArena's ease of play, as players can experience the game directly in their browser without having to download and install it. However, it still needs to be discussed to directly choose an online platform as a development goal for this project, BoardGameArena its performance in terms of graphic rendering capabilities and sound customisation is poor.

Tabletopia: Tabletopia, also a browser-based online board game platform, offers a richer 3D online board game experience that recreates the immersion of tabletop gaming for players. Compared to BoardGameArena, Tabletopia is closer to traditional tabletop gaming in terms of visual presentation and gaming experience. However, while there are advanced digital solutions like Tabletopia on the market, they still have limitations. For example, Dive et al. notes that Tabletopia, while offering a rich set of features and game editors, may not be as powerful as a desktop application when compared to a robust Desktop platform [26], for example, in terms of complex game logic and detailed graphical presentations.

2.7.4. Comparison and Summary

There are differing views on Online Computing Platforms. According to Sparrow and Rogerson, while platforms such as BoardGameArena and Tabletopia offer a convenient way to play board games remotely, the "digital" approach ignores some of the core qualities of board games [27], particularly the lack of physical components such as dice and pieces, which are highly valued by board game enthusiasts. However, Luiz et al. also points out that, despite some criticism of the combination of digital platforms and board games, digital board games offer directions for applying AI techniques to enhance the gaming experience as well as strategy scores [28]. For example, algorithms can be used to automate game processes, manage resources, calculate scores, and check winning conditions.

There are also many different advantages and disadvantages for different Game Computing Platforms. The first is cross-platform compatibility: while LibGDX provides true cross-platform support, enabling games to run seamlessly on Desktop and Android platforms, online platforms, while offering broad accessibility, are limited in terms of performance and freedom. In terms of development efficiency, Dive et al. indicates to the emergence of online platforms such as Tabletopia that use computer technology to bring the classic board game experience into the digital realm [26]. The platform's unique editor allows users to create new games from scratch without programming knowledge. However, BoardGameArena and Tabletopia, while providing certain development tools and interfaces, may not be as flexible in terms of actual development and performance optimisation as using a desktop framework such as LibGDX directly. The popularity of the Java language and the community of the LibGDX framework can provide an efficient development process and strong technical support.

The comparative analysis shows that although online platforms provide a convenient way to promote digital board games, the choice of the Java language and the LibGDX framework based on the Desktop and Android platforms brings significant technological advantages to the project when developing board games with complex logic, high-quality graphics and powerful AI.

For this project, developing in Java on the Desktop and Android platforms and using the LibGDX engine is a good choice.

2.8. Game AI Algorithms

In the process of digitisation of board games, it is extremely necessary to discuss the ai algorithms that need to be used in the research, and the following section explains some of the common ai algorithms related to the research. When choosing an algorithm, the complexity of the algorithm, the availability of resources and the characteristics of the game itself are all factors to consider. Table 2 provides a brief comparison of these algorithms.

2.8.1. Mini-Max and Alpha-Beta Pruning

The core idea of the Mini-Max is to maximise one's minimum gain. In the game tree, it considers all possible moves and reactions of the opponent to predict the possible game outcomes [29]. The algorithm evaluates all leaf nodes of the game tree by recursively evaluating them and then selecting the best move from them. The algorithm can be formalised as:

$$\text{Mini-Max(node)=maxchild} \in \text{children(node): Mini-Max (child)} \qquad (1)$$

The Max player tries to maximise the gains in the game, while the Min player tries to minimise the maximum player's score. By assuming that the Min player will play optimally in each state, the Max player needs to choose the available actions that will eventually give the optimal result to both players. See figure 4.

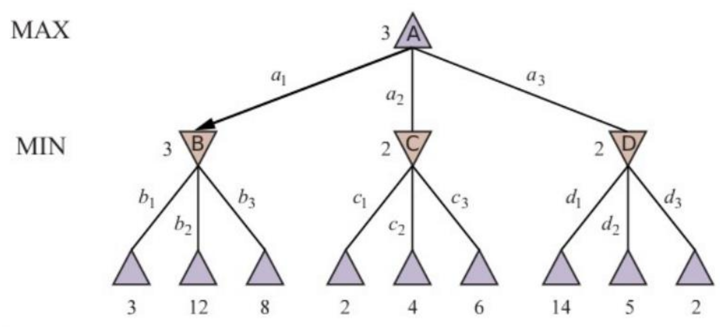


Figure 4. How a Mini-Max proceeds [3]

Alpha-Beta pruning, as shown in figure 5, reduces computation by stopping checking the rest of the children of the current node when the algorithm determines that one of the children of the current node is the best. However, Dreżewski et al. found that during real-world play, the intentions of both players are not fully visible, and thus many actions may be possible in each state [30]. In this case, search methods usually use some speculative pruning to reduce the number of options to be considered. The main advantage of both algorithms is that they are effective in finding the best move, especially in complex games with many possibilities. However, their disadvantage is that they require significant computational resources, especially when considering deep game trees.

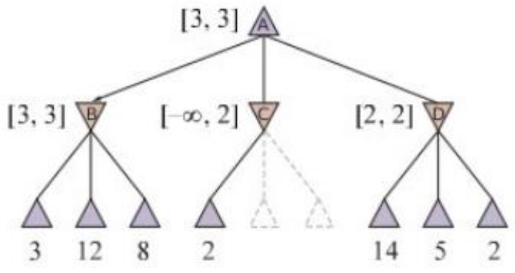


Figure 5. Result of Alpha-Beta pruning [3]

2.8.2. Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) is a search method that combines the accuracy of tree search with the generality of random sampling. As figure 6 shows, its core algorithm constructs the search tree incrementally and asymmetrically, using a tree strategy that strikes a balance between exploring regions that have not yet been adequately sampled and exploiting previous regions [31]. Whereas Ontanon found in their experiments, by combining machine learning with the strategy of Monte Carlo tree search, that the MCTS algorithm initially does not know the information about the expected reward for each action [32], and needs to make a judgement call by repeatedly trying different actions to make the optimal choice in the end. Browne et al. also state that: the advantage of MCTS is that it is an anytime algorithm, meaning that its performance usually improves to some extent as computational resources are increased [31].

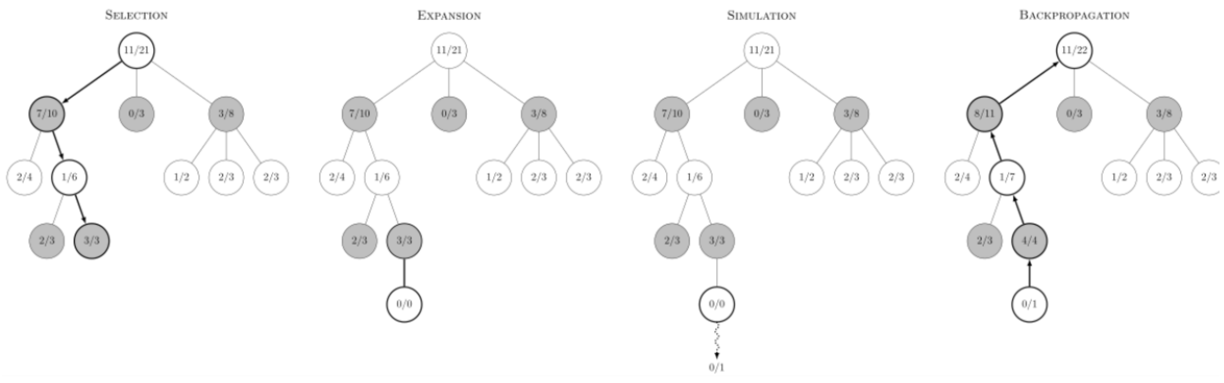


Figure 6. Step of MCTS [33]

However, Baier and Winands pointed out that: MCTS builds a tree with a high degree of selectivity [34]. Such high degree of selectivity means that the algorithm would like to prioritise those options that seem to be the most promising, and focusing its resources and computational power on exploring paths that are most likely to lead to a victory, so as to find a better solution in a limited time. However, a high degree of selectivity also brings limitations, especially in strategy games, where the AI can easily miss critical choices and fall into traps prepared by the opponent in tactical situations if the opponent intentionally misleads the AI.

2.8.3. Machine learning in Game AI

Neural Networks (NN): Neural Networks have developed very rapidly in recent years and are a powerful AI tool. Marius Marinel Stănescu et al. point out that for strategy games, games between strategies can be solved by applying a continuous recursion through the neural network until a termination state is reached [12]. Meanwhile, a study by Yakovenko et al. found that CNNs have the advantage of being able to make optimal choices in a finite set of data and improve low-level strategies made by Mini-Max based on the choices made [35].

Genetic Algorithm: Riechmann suggests that the steps of a genetic algorithm can be viewed as a two-part process: the population first moves to a steady state [36], once it reaches a steady state, a learning process takes place to move out of the steady state and back again. Board type games can be analogized to the steps indicated by the authors, where each player's turn is actually a learning process and affects the whole game, and at the end of the player's action, the whole game will return to a steady state.

2.8.4. Comparison and Summary

Considering the resource and time constraints, the Mini-Max algorithm and the basic Monte Carlo Tree Search (MCTS) algorithm have certain advantages. Whereas the strategic and decision-making process of Spellcaster games requires the AI to be able to handle different game situations efficiently. See table 2 for a detailed comparison:

Table 2. Comparison of different algorithms in game AI

Algorithm	Application	Responsiveness	Time Complexity	Space Complexity	Limitation
Mini-Max	Two-player turn-based game	medium	O(b^d)	O(b*d)	Only more applicable to games with full information and limited state space
MCTS	Complex, highly randomized games	Change (dependence on size of problem)	Depends on the realization	Depends on the realization	Computationally intensive, need to balance exploration and utilization
NN	Complex pattern recognition, prediction	medium to high	Change (Dependent on network size)	Change (Dependent on network size)	High data volume requirements and long training time
GA	Optimization of game decisions	low to medium	Change (Dependence on size of problem)	Change (Dependence on size of problem)	Good adaptive functions are required

Considering that the Mini-Max idea is widely used in strategy board games and is especially applicable to turn-based strategy systems, it has been chosen as the implementation of the AI for this project.

3. METHODOLOGY

3.1. Research Method

In order to realise the Spellcaster game and implement an AI for it, the Methodology of this project will be elaborated separately around the following different phases, which are arranged in the order of the initial implementation plan of this project.

3.1.1. Design Phase

Main Game Framework: Due to the complexity of Spellcaster's rule system, it is essential to design a well-developed game flow framework to accurately handle the interactive effects of various types of spells. The initial plan is to divide each turn into multiple phases within the game, first dealing with the continuous effects triggered by the previous turn. Subsequently, the player selects the gesture for the current turn, a process that takes into account gesture selection restrictions caused by some spells. Next, the system parses the player's chosen gesture, executes the various spells in turn, and performs a preliminary damage settlement for damage. Then an action is performed by Monster. Finally, the overall damage settlement is finally performed.

Technology selection: In making the technology selection, three aspects will be focused on: platform, game engine and programming language. First, Desktop was initially considered as a good platform for game testing and iteration due to its broad user base and high accessibility, while the project also considered extending the program to the Android platform at a later stage. The Desktop platform was chosen due to its ability to provide a richer user-interaction experience and higher game performance, while the inclusion of the Android platform was intended to expand the potential user base, utilizing its openness and high accessibility. In addition, the development process must follow the Apache License 2.0, which eliminates the need to consider licensing issues during the development process, an advantage for smaller games such as Spellcaster.

For the game engine, LibGDX was chosen, which is an open-source framework based on the Java language environment, and is very suitable for the development needs of this game due to its lightweight and powerful functions. Based on Java language, LibGDX can easily utilize the features of JVM (Java Virtual Machine), which is very convenient for instantly checking the effect of the game during the programming process.

As for the programming language, LibGDX uses Java as the game engine, so Java is adopted as the main development language. At the same time, Java is given object-oriented (OOP) characteristics, which are very suitable for modular board game elements, making program writing and subsequent calls more convenient.

Interface Design: Design an intuitive interface based on the rules of the game Spellcaster. The initial idea is to present the game using a vertical screen interface. The different gestures of each wizard will be set in the form of buttons at the bottom of the screen, to activate different spells through gesture combinations, to summon followers on the scene, etc. The interface should also have clear metrics, including health points, previous gestures and the duration of the opponent's spells. During the actual Implementation, the final game interface was made in landscape, leaving more room for the display of game states with target selection, spell selection, and other actions.

3.1.2. Implementation Phase

Game Logic Programming: At the beginning of the implementation phase, the first task is to translate the design phase game framework into actual game logic code. This involves the programming implementation of the core rules of the Spellcaster game, including but the sequential processing of player actions, the logic of the gesture and spell system, and the management of game states. In order to ensure the accuracy and efficiency of the game logic, the initial plan is to adopt a modular programming approach, where each game component such as Spell, Effect, Monster, etc. is designed as an independent module, which is easy to develop and test separately. Among them, for the design of Monster class, it is planned to make it inherit part of the functions of Player class, so that Monster's normal attack is also regarded as "releasing a damage spell without special effect", which is conducive to simplifying the complexity of the code.

At the same time, due to the convenient cross-platform nature of LibGDX, it is possible to program directly in the IDE, and the final exported project can be run on both Desktop and Android platforms, so there is no need to think too much about cross-platform code porting.

AI algorithm: Use the Mini-Max as the basic decision-making framework of the AI, which helps the AI to judge the next gesture to be used by using the very large and very small values, which will be the main algorithm to be used by the AI in the whole game. The algorithm is able to provide the AI with an effective strategy formulation mechanism by modelling the possible actions of the opponent [30]. The Mini-Max takes into account the best- and worst-case scenarios when evaluating the potential value of each move, thus enabling the AI to select the optimal move (i.e., the gesture in the game). This approach is particularly effective for predicting opponents' moves and developing its own response strategies.

To evaluate the effectiveness of the Mini-Max, it is planned to play different configurations of Mini-Max instances against each other. This involves using different pruning settings to see how the AI performs under different strategies. In this way the efficiency and effectiveness of the Mini-Max can be compared under different parameters. Such an evaluation will help tweak and optimize the algorithm to be applied more effectively throughout the game.

In the actual implementation, the design idea of AI mainly refers to the use of Mini-Max ideas in Gomoku, hoping to map the elements of Spellcaster game with Gomoku, so as to facilitate the application of Mini-Max in non-chess strategy games such as Spellcaster.

3.2. Limitation

Limitations of algorithm selection: The Mini-Max algorithm selection, while suitable for most board decision games, may not be able to handle extremely complex decision trees or accommodate extreme game strategies for Spellcaster. In addition, these algorithms may require long computation times at runtime, which may affect the smoothness of the game and the user experience.

Platform and technology limitations: Using LibGDX may limit the end result of the game. LibGDX, as an open-source game development framework, may not be as powerful as professional game engines in some respects, although it provides a range of tools that facilitate the development of 2D and 3D games. Also, LibGDX is not as well-known as engines such as Unity or Unreal, and community support may not be as strong as other engines, which may limit the ability to find related resources.

Limitations of data collection and analysis: reliance on social media for user testing participant recruitment may have resulted in biased samples. For example, the sample may be biased towards players of a particular age group or level of gaming experience. Large-scale data collection and complex data analysis may not be possible due to resource and time constraints. This may affect the overall evaluation of AI performance and game evaluation. There may be subjective judgements when evaluating the game evaluation at the same time.

4. IMPLEMENTATION

This section mainly describes the specific implementation ideas and process of Spellcaster, and will introduce the game main loop design, interface design, code framework design and AI agent design respectively. All the code mentioned in this section will be submitted together with the Dissertation.

In the implementation, the specific environment configuration version is as follows:

IDE: IntelliJ IDEA Community Edition 2023.3

JDK version: OpenJDK 17.0.7

Gradle version: 7.5.1,

LibGDX engine version: 1.12.1,

API used for the Android virtual machine: API 34.

The implementation of Android version of the game benefits from the excellent cross-platform features of the LibGDX framework. This means that there is no need to write different code for different platforms during development. The final code implementation is done in a single IDE and can be seamlessly applied to both desktop and mobile. This greatly simplifies development by eliminating the need to wrestle with issues such as Camera placement in LibGDX, allowing development to focus on refining the core functionality of the game.

The implementation of HTML version of the game is also included in the project package built with the LibGDX engine. Due to the cross-platform nature of LibGDX, the HTML version of the game can theoretically be played in a browser as well, but this was not the goal of this project and therefore no specific testing was done.

4.1. Game Main Loop Design

In the digitisation of Spellcaster, the design of the main loop of the game is particularly critical. Different settlement processes lead to different spell interactions. Therefore, it is necessary to determine the main loop and the order in which the spells will take effect before the specific implementation can be carried out in the subsequent process of writing the code.

The core of the main game loop is based on Spellcaster's original game rules. The overall game loop is divided into four steps: resolving the effects of the previous turn, selecting gesture, resolving the effects of the spells in that turn, and settling the overall damage. This process design can well simulate the turn-based characteristics of board games, keep the pace of the game and increase the strategy.

Of these four steps, spell effect resolution is particularly important. In particular, the Spells in the game are classified into four categories: Summon, Protection, Enchantment, and Damaging, and the order in which these spells are resolved has a significant impact on the outcome of the game. The original game rules did not directly state the order in which the spells interact, so in the actual implementation of this project, the order in which the spells are settled was additionally specified, i.e., Summon, then Protection, then Enchantment, and finally Damaging. The reasons behind this decision are:

(1) Except for RemoveEnchantmentSpell, there are no spells that affect the Summon type spell. Therefore, Summon spells are placed at the top

(2) To ensure that Protection spells work correctly, they need to always take effect before Enchantment spells take effect

(3) Enchantment spells usually cause some negative effects, and most often affect the next turn, so they don't need to be considered too far ahead.

(4) To ensure that Damaging spells are properly affected by other spells, they should always be resolved last.

In Spellcaster's original game rules, there are some rule-specific aspects. For example, the special rule for the Monster elemental - it does not attack when it is about to die, while other types of Monsters are still able to attack normally before death. Other details such as these need to be considered in the development of the game loop. After these considerations, the final game loop is shown below in figure 7:

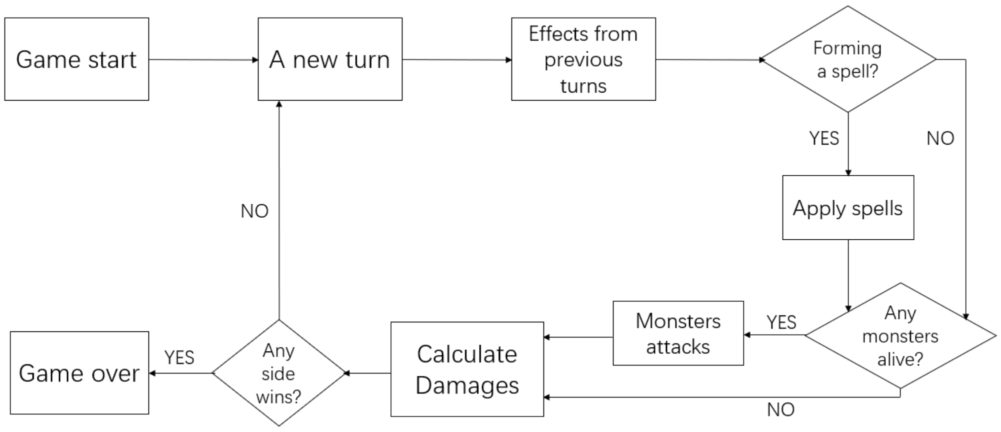


Figure 7. Spellcaster game loop

4.2. User Interface Design

The game interface is the core point of interaction between the player and the game. For this part of the code implementation, Observer Pattern is used to update the game interface and game status.

Desktop Interface Design: The left and right sides of the interface are the player and AI areas, designed to mimic an information board, where the player can recognise their own status and that of their opponent at a glance. The player can select gestures by clicking on several buttons directly below. When there are spells that can be casted, a drop-down menu appears on the left and right for the player to select from.

In terms of button design, it was chosen to use the first letter of each gesture directly, such as palm (P) and fingers (F). Using the actual images of these gestures as the material for the buttons might be a good idea, as it visually enhances the immersion of the game and gives the player the experience of casting a spell themselves. However, the reason for not using pictures as button texture is based on two main reasons. Firstly, for convenience of programming, modifying the material images of the buttons individually is cumbersome, and modifying them uniformly requires regenerating the LibGDX material packs. Secondly, for ease of memorisation. It is easier to memorise a sequence of letters than it is to memorise a sequence of gestures hieroglyphically. At the same time, when consulting the spell list, it is easier to find the corresponding spells according to the letters.

The basic interface design is shown below in figure 8:

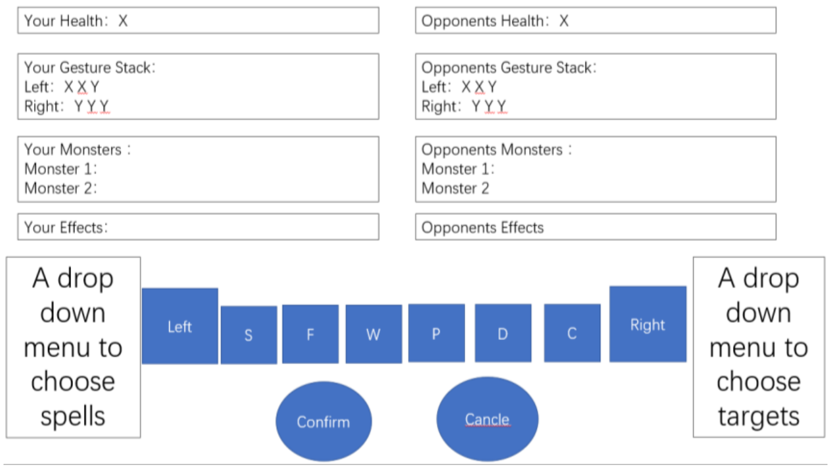
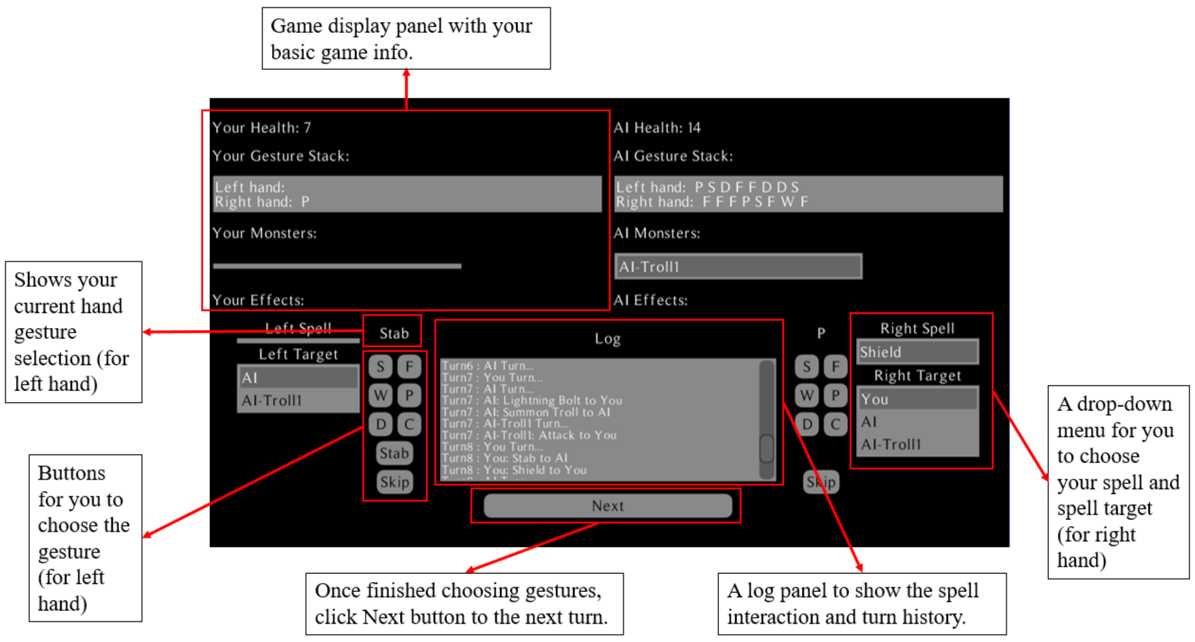


Figure 8. Early ideas of the game UI

In addition, in the actual development, an area was added for displaying the game log, which makes it easy for the player to track the historical dynamics of the game in real time, and also makes it easier to understand what exactly is happening in the game and how the spells are interacting. Also separating spell and target selection for left and right handers prevents the player from making separate selections. The final game interface is shown in figure 9:


Game display panel with your basic game info.
Shows your current hand gesture selection (for left hand)
Buttons for you to choose the gesture (for left hand)
Once finished choosing gestures, click Next button to the next turn.
A log panel to show the spell interaction and turn history.
A drop-down menu for you to choose your spell and spell target (for right hand)
Left Spell
Stab
Left Target
S
F
W
P
D
C
Stab
Skip
Next
P
Right Spell
Right Target
Skip
Log

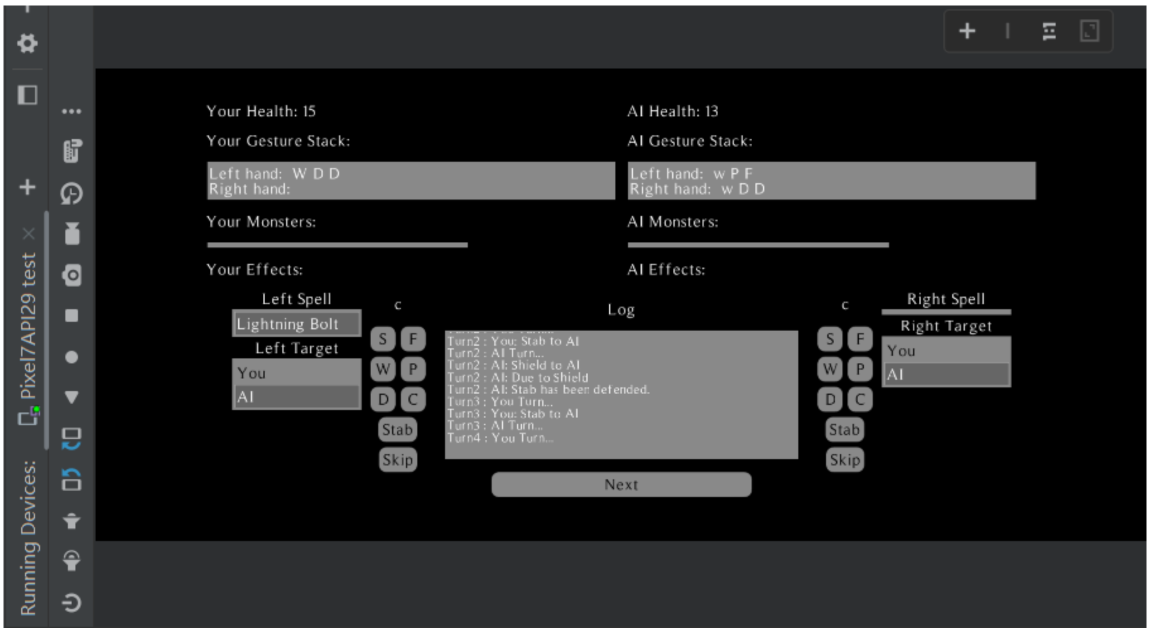
```
Your Health: 7                 AI Health: 14
Your Gesture Stack:            AI Gesture Stack:
Left hand:                     Left hand:  P S D F F D D S
Right hand:  P                 Right hand:  F F F P S F W F
Your Monsters:                 AI Monsters:
                               AI-Troll1
Your Effects:                  AI Effects:

Left Target: AI / AI-Troll1
Right Spell: Shield
Right Target: You / AI / AI-Troll1
```

```
Turn6 : AI Turn...
Turn7 : You Turn...
Turn7 : AI Turn...
Turn7 : AI: Lightning Bolt to You
Turn7 : AI: Summon Troll to AI
Turn7 : AI-Troll1 Turn...
Turn7 : AI-Troll1: Attack to You
Turn8 : You Turn...
Turn8 : You: Stab to AI
Turn8 : You: Shield to You
```

Figure 9. Final Game UI on desktop

Android Interface design: Thanks to the excellent cross-platform feature of LibGDX framework, the game can run on Android directly. The interface of the game on the Android virtual machine is shown below in figure 10:



```
Your Health: 15                AI Health: 13
Your Gesture Stack:            AI Gesture Stack:
Left hand:  W D D              Left hand:  w P F
Right hand:                    Right hand:  w D D
Your Monsters:                 AI Monsters:
Your Effects:                  AI Effects:

Left Spell: Lightning Bolt
Left Target: You / AI
Right Target: You / AI
```

```
Turn2 : You: Stab to AI
Turn2 : AI Turn...
Turn2 : AI: Shield to AI
Turn2 : AI: Due to Shield
Turn2 : AI: Stab has been defended.
Turn3 : You Turn...
Turn3 : You: Stab to AI
Turn3 : AI Turn...
Turn4 : You Turn...
```

Figure 10. Final Game UI on Android virtual machine

4.3. Game Framework Design

4.3.1. Overall Design

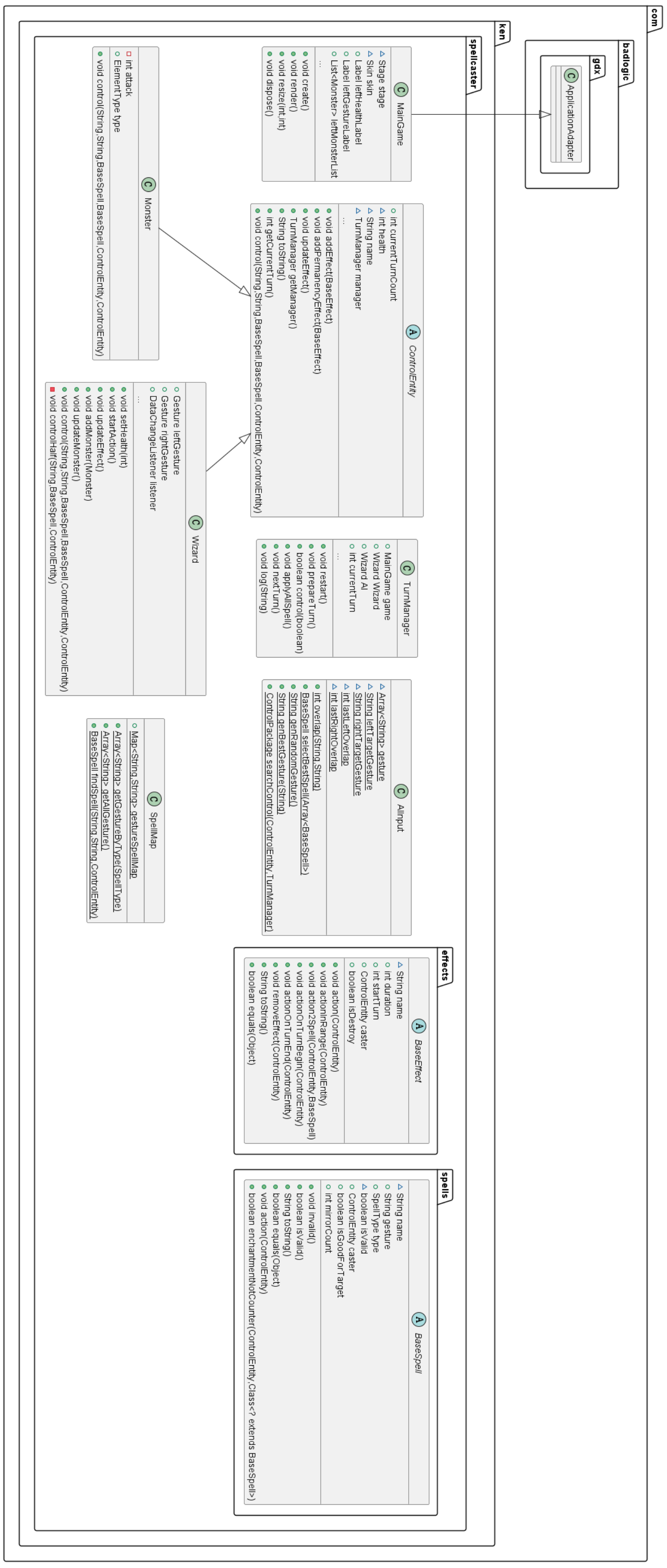


Figure 11. Overall UML class diagram

In the early stages of implementation, the initial design idea was to create a unified Spell class, where each specific spell acts as a method in the class. However, during actual development, this idea was found to be infeasible because it violated the basic principles of object-oriented programming (OOP). Such a design requires a large amount of code stacking for interactions between spells, similar to casually connecting lines between spells. This resulted in extremely poor maintainability and extensibility of the interaction results, and even small changes could require a great deal of work.

Therefore, in the subsequent improvement, the main modularity and object-oriented approach is adopted. For example, creating a base class for inheritance, which is convenient with the maintenance and expansion of the code later. An overall class diagram is shown below in figure 11.

In the framework, BaseSpell and BaseEffect are mainly defined as abstract base classes, which respectively define the templates and behaviours that all spells and effects must follow, ensuring that different types of spells and effects can be handled in a unified way while maintaining their independence. This design makes adding new spells and effects simple and straightforward. During subsequent development, it is only necessary to define the specific behaviour of the new spells, and by inheriting from BaseSpell or BaseEffect, it is easy to integrate the new spells into an existing game.

Additionally, the SpellMap class acts as a registry for spells, using Map<String, String> to store spells, with the key being the sequence of gestures, and the value being the name of spells. It is possible to map all spell objects to the corresponding symbols and gestures. This mapping ensures that no new spells are added without interfering with the existing system, and it is especially easy to call new spells in the game logic.

The ControlEntity class also serves as an abstract base class for managing the underlying properties and behaviour of all interactable entities, including variables such as health, controller, and created abstract methods such as addSpell for releasing spells.

4.3.2. Class Design

Since the code implements a very large number of classes, the main discussion here will focus on the core classes.

BaseSpell, as an abstract class, is at the heart of Spell design. This class contains the basic elements of a spell, such as the action method that is executed when the spell takes effect, and the method that generates the log. All concrete spell classes inherit from BaseSpell and implement their own unique effects by overriding methods such as action.

Writing each spell as a separate class also makes it easier to filter the spells in the game logic later, which can be done with the Java instance of keyword. A UML diagram of BaseSpell is shown in figure 12.

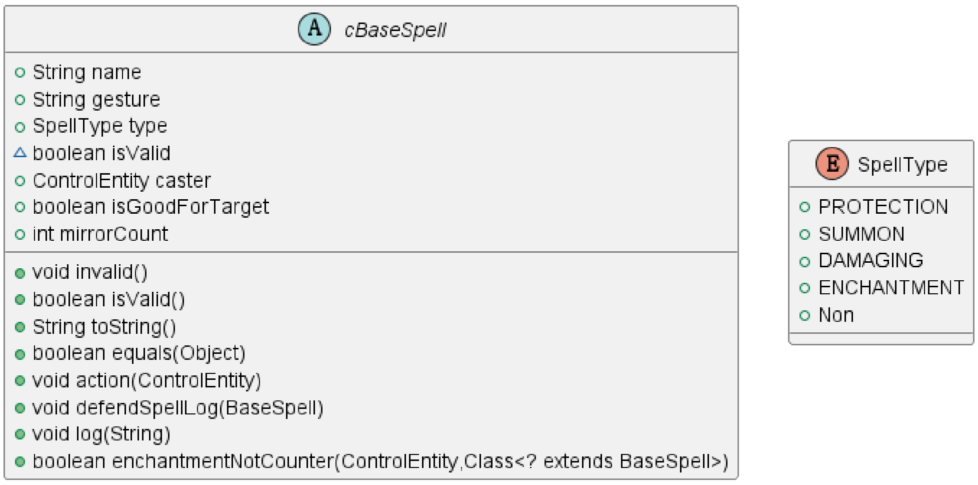


Figure 12. UML class diagram of BaseSpell

Similar to BaseSpell, as shown in figure 13, the BaseEffect class also serves as an abstract class that provides the basis for all possible effects in the game. It defines the actionOnTurnBegin and actionOnTurnEnd methods for effects to take effect at the beginning and end of the turn, which are used to ensure that the timing of the Effect's effectiveness is correct.

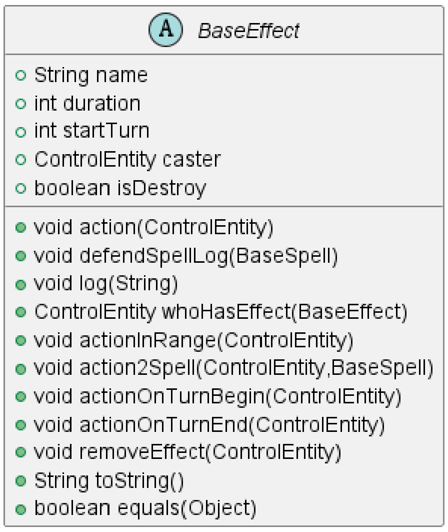

A
BaseEffect
String name
int duration
int startTurn
ControlEntity caster
boolean isDestroy
void action(ControlEntity)
void defendSpellLog(BaseSpell)
void log(String)
ControlEntity whoHasEffect(BaseEffect)
void actionInRange(ControlEntity)
void action2Spell(ControlEntity,BaseSpell)
void actionOnTurnBegin(ControlEntity)
void actionOnTurnEnd(ControlEntity)
void removeEffect(ControlEntity)
String toString()
boolean equals(Object)

Figure 13. UML class diagram of BaseEffect

There is more to the game than just the player and the AI, there is also the Monster. If Monster was really implemented as just an appendage of Wizard (a variable in the Wizard class), many of the spells that affect Monster would be very difficult to implement then.

Therefore, it was decided that Monster, like Wizard, would be treated as a separate game entity, with its own turn, health attributes, etc. To achieve this, the ControlEntity class was designed as an abstract class containing basic properties such as Attack, Health and Effects. As shown in figure 14, this allows the Wizard and Monster classes to implement their specific behaviours by inheriting from ControlEntity and overriding the corresponding methods.

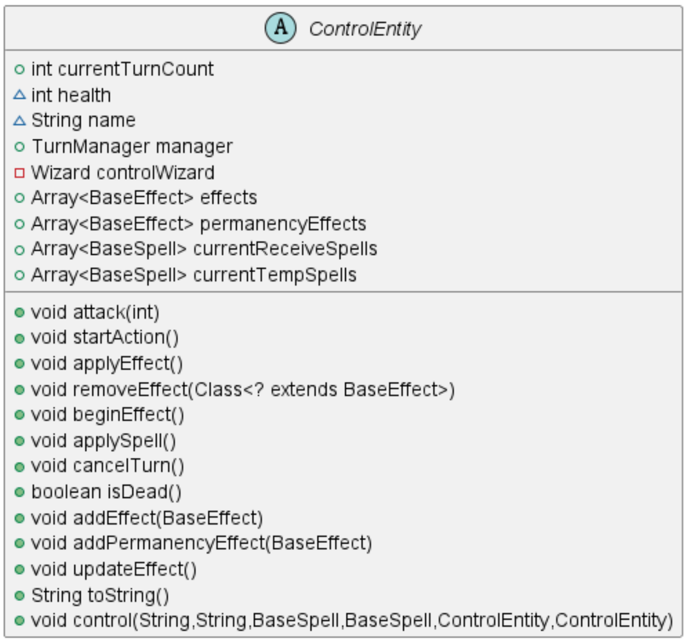

A
ControlEntity
int currentTurnCount
int health
String name
TurnManager manager
Wizard controlWizard
Array<BaseEffect> effects
Array<BaseEffect> permanencyEffects
Array<BaseSpell> currentReceiveSpells
Array<BaseSpell> currentTempSpells
void attack(int)
void startAction()
void applyEffect()
void removeEffect(Class<? extends BaseEffect>)
void beginEffect()
void applySpell()
void cancelTurn()
boolean isDead()
void addEffect(BaseEffect)
void addPermanencyEffect(BaseEffect)
void updateEffect()
String toString()
void control(String,String,BaseSpell,BaseSpell,ControlEntity,ControlEntity)

Figure 14. UML class diagram of ControlEntity

4.3.3. Design Patterns

Observer Pattern: In the implementation, design patterns were tried. Among them, Observer Pattern is used, which is mainly used to implement the information update of UI interface. In Observer Pattern, the Wizard class and TurnManager class act as Subject, while the UI components of the game play the role of Observer. when the game state changes, such as the update of the Log, the Health, Effects and the gesture stack, the UI components will be notified and update their state accordingly.

The ChangeListener interface in the TurnManager class is used to listen for updates to the game log, and as soon as a new log is generated, it notifies the registered listener - the Log panel UI component - and updates the display of the game log. Here, the TurnManager is the Subject, and the UI component is the Observer.

The use of the DataChangeListener in the Wizard class also follows the principles of the Observer pattern; the Wizard class is responsible for managing key information such as the health, gestures, Monster list, and Effect list, etc., and when this information changes, the Wizard notifies the registered DataChangeListener UI component to update the interface accordingly in the panel. In this scenario, the Wizard acts as a Subject, controlling the key states of the game, while the UI component acts as an Observer, updating the player view based on the Wizard's state changes.

Factory Pattern: Although no traditional factory classes or methods are used in the Spellcaster project, the BaseSpell and BaseEffect base classes are designed and applied using the concept of Factory classes. These two base classes provide a unified creation and management framework for different types of spells and effects, making the process of adding new spells and effects easier.

In the SpellMap class, the implementation of the findSpell method fully demonstrates the application of the Factory method pattern. The method dynamically determines and creates the corresponding BaseSpell subclass instances according to the incoming name, which is a process of selecting specific products and instantiating them on demand.

4.4. Game AI Design

4.4.1. Overview

Initially the traditional recursive Mini-Max algorithm [3] was used. Attempts were made to set up a function to evaluate all states of the game: health differences, restrictions between spells (e.g., damage spells are restricted by Shield), etc., trying to consider ALL possible scenarios. However, this resulted in an evaluation function that was too large and complex, making AI consume too much time to generate a bad result. Subsequently, an attempt was made to simplify the evaluation function by considering only the difference in life values for evaluation, but this method was out of touch with the actual game state and failed to achieve effective results.

From the previous attempts, it can be concluded that the idea of Mini-Max is effective for Spellcaster, but the game tree of Spellcaster is more complex, and the traditional search of the whole game tree is not applicable. Therefore, an attempt is made to start with the sequence of gestures corresponding to each spell. The sequence required by each spell is constant, so assume a child node A. Node A is the turn in which a particular spell is just cast. Starting from the child node A, work backwards through the entire game tree to reach the purpose of pruning. Using this advantage, the state space of the AI can be limited, so that the AI always generates the optimal solution that reaches the sub-node A quickly. To make a basis for the evaluation function setting mentioned later.

An interesting idea is to try tuning the evaluation function on an abstract level. In the preliminary AI tuning, most of the focus is on the figurative representation of the evaluation function, including the evaluation content, changing parameters of scoring calculation. After many attempts, it is realised that the implementation of Mini-Max idea can be adjusted from the abstract level, and trying to extract the features of Spellcaster, analogising Spellcaster to Gomoku, such as regarding the gestures in

Spellcaster as the black and white stones in Gomoku. It is also considered the setting of Spellcaster's evaluation function from the perspective of Chess Game, and analogy mainly focuses on the setting of the evaluation function and the determination of the game state, such as analogising "generate a gesture" as "place a stone". Compared to the previous tuning, this analogy provides more insight into the mechanism of the game.

4.4.2. An Abstract Strategy and Tuning the Evaluation Function

Application of Mini-Max ideas: For traditional chess games, such as Gomoku, the evaluation function set by the Mini-Max is often designed to evaluate a game of "three stones in a row" or "four stones in a row” and set up an "expected value" - i.e., "five stones in a row". Mini-Max will keep advancing to this expected value in each evaluation, which is the process of approaching Max value and Min value from the principle of the algorithm.

Analogising Spellcaster with Gomoku, each time the player chooses a gesture, it can be modelled as a "drop", with different gestures "connecting" to form different spells - These soon-to-be-completed spells become the AI Agent's "expected value".

A simple schematic is shown below in figure 15:

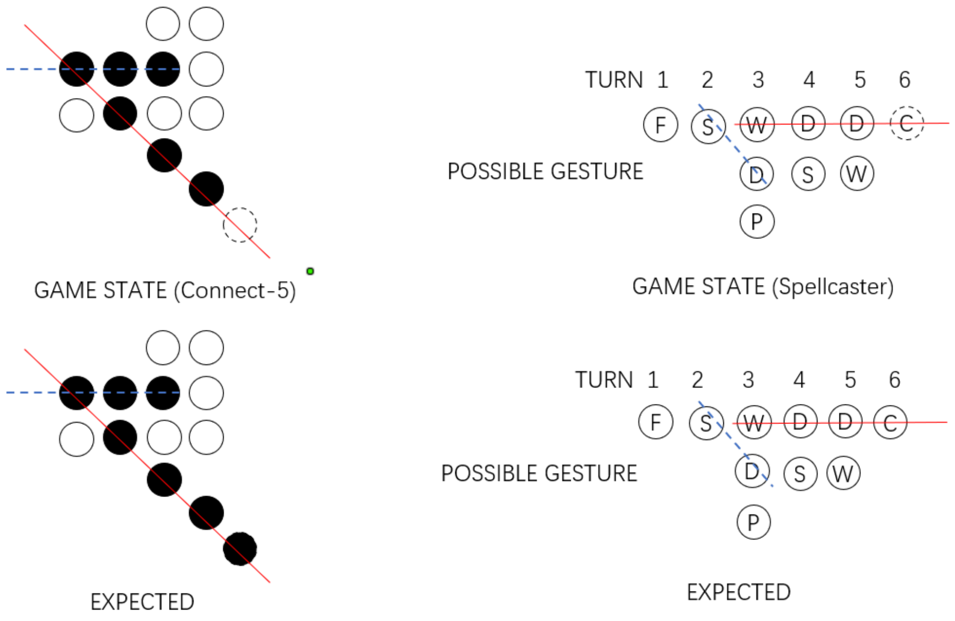


Figure 15. How Spellcaster analogies Gomoku

(WDDC – LightingBoltSpell, SD – MissileSpell)

By this analogy, the Mini-Max algorithm in Gomoku can be modelled on Spellcaster's AI implementation.

Spellcaster can be converted into an abstract board game: each time the AI Agent chooses a gesture it is simulating the process of making a move, and the goal of the AI is to make another kind of "five in a row" - a gesture sequence that creates a spell. In this case, an evaluation function and a sorting function are essential. The evaluation function is used to evaluate the current state of the game, while the ranking function is used to filter the AI's goals so that the AI always chooses spells that are favourable to the game as its expectations.

Evaluation Function: In Gomoku, the case of five in a row is regarded as an "expectation", and the AI makes decisions by aiming for it. Similarly, in Spellcaster, the evaluation function introduces the completion of a gesture sequence as an "expectation" for evaluation, and the AI makes decisions in the direction of achieving the particular spell. This expectation setting allows the AI to select gestures with a goal in mind. The key to the accuracy of the evaluation function is the setting of the scoring mechanism. Overlap Function scores the overlap between the current gesture and the target gesture sequence. The higher the overlap, the higher the score, and the AI always tends to make decisions with high scores. Based on Sorting Function, gesture length is used here as the main criterion for evaluating the priority of spells. An assumption is made here that longer gestures usually mean more powerful or effective spells. Sorting Function below is used to help choose the best target.

Sorting Function: In Spellcaster, the sorting function scores the gestures based on their contribution to the completion of the spell, which is used as a basis for decision making. This approach allows the AI's selection process to favour spells with high gesture consumption and complexity readings and make the next move that is most conducive to victory.

Overlap Calculation: Similar to evaluating the potential for connecting stones on the board in Gomoku, Spellcaster's AI evaluates the contribution of each gesture to completing the target spell by calculating the overlap between gestures (Number of overlapping gestures between the previous spell and the sequence of gestures consumed by the next spell). This calculation allows the AI to identify the optimal choice of gesture in the current situation, thus improving the efficiency and effectiveness of spell completion.

The Introduction of Randomness: when the AI generates gestures in the way described above, the rule set to the effect that longer gesture sequences consume higher spell scores leads to the possibility that the AI may always choose those spells that are the longest (such as FingerOfDeathSpell), which can lead to the AI's actions becoming predictable. Therefore, the improvement is to introduce randomness in the evaluation function and let the AI randomly choose the top three ranked spell gesture sequences as expected values, which can make the AI's decisions more unpredictable.

5. CONCLUSION

This project successfully digitised the game Spellcaster using the Java language, within the LibGDX framework. A basic user interface was implemented, and AI opponents were implemented within the game using Mini-Max ideas. Through this project, implementor has gained following insights into the construction of AI in strategy games and the evaluation of programs.

5.1. Implementation

The AI is implemented based on Mini-Max ideas, including the evaluation function, to generate the optimal solution for decision making. The sequence of gestures required for each spell fix can be seen as a pruning of the current search space in advance. Taking advantage of this, the search space can be converted into a planar searcher, limiting the state space of the AI. Therefore, the decision sources of the evaluation function are mainly: the current gestures made, target gesture sequence length, and the overlap numbers. Each turn generates a current optimal solution based on the current state, and then makes a decision according to this optimal solution.

Analogy with Gomoku help form Spellcaster's evaluation function. By analogising Spellcaster's evaluation function to Gomoku at an abstraction level, the gestures chosen for each turn are analogous to the drop of a piece on a chessboard. The evaluation function enables the AI to make more targeted advances towards the optimal solution.

Cross-platform is achieved through LibGDX. This project has achieved cross-platform implementation, mainly due to the excellent cross-platform nature of the LibGDX engine. By simply writing the source code in the core folder, LibGDX projects built with Gradle can be deployed and run normally on Desktop and Android respectively.

5.2. Limitation

Lack of Agile development: During the initial project planning, the use of Agile development was not considered to plan the project process in a specific way. In the early stages of development, when implementing spell interactions, it was often found that there were still problems with the initially declared spell variables, and then went back to rewrite them, which in turn led to other problems. Although the development was planned more scientifically at a later stage, the time wasted at the beginning still led to a slower actual process.

Limitations of the AI decision-making process: The decision-making process set by the current version of the AI is still relatively homogeneous, with the expected value set to work towards the longest sequence of gestures, which leads to the direct result that the AI has a high probability of working towards the longest gesture in the list of spells, which makes the AI's behaviour completely predictable. Although subsequent improvements to the AI take into account the introduction of randomness, such a decision-making process will still have limitations, especially in the early stages of the game, where the AI's subsequent decisions have a higher probability of being predicted by the player if the randomly generated gestures are identical.

Limitations of the game interface: the game interface is designed to be relatively homogeneous and suffers from too many buttons, which may cause some degree of annoyance to the player during actual play. These design limitations of the interface may affect the player's gaming experience.

5.3. Future Work

5.3.1. AI Performances

Add "gesture table": Since Spellcaster has many game states, in order to further optimise the performance of the current version of the summary AI, it is planned to introduce the concept of "gesture table", which is similar to the Chinese Chess or Chess game list. That is, Spellcaster's AI can evaluate the value of a particular game state (i.e., a particular sequence of gestures by the opponent and by itself), just like chess, and the "gesture table" should be able to pre-set the optimal gesture choices of the AI in the current situation for this game state. Selection. By assigning appropriate values to specific situations and spell combinations in Spellcaster, the AI should be able to more accurately assess the favourable level of the current situation and make more reasonable decisions based on the pre-assigned decisions in the "gesture table".

Implement AI based on MCTS algorithm: In order to further improve the AI's decision-making quality and its adaptability to the game state, it is planned to try to implement AI based on Monte Carlo Tree Search (MCTS) algorithm. The MCTS evaluates the expected value of each possible action through a large number of stochastic simulations and handles the situation of a large decision space that is not completely certain. better fits the Spellcaster game. This applies Spellcaster.

5.3.2. User Interface

Adding more interfaces: It is planned to add a basic game start panel in the future, providing options including starting a new game, adjusting settings, and viewing game rules. At the same time, considering that the player's understanding of the rules of the game is crucial to enhancing the gaming experience, future work plans to add a detailed rules narrative panel. This panel will provide players with the basic rules of Spellcaster and a guide to playing the game, helping players quickly grasp the essentials of the game and lowering the learning threshold.

Add a tutorial: To help new players be familiar with the game's interface and operations, an interactive tutorial is planned to be added the first time a player enters the game. The tutorial should be able to guide players through the functions and meanings of the various panels and buttons, ensuring that players can start their gaming experience more smoothly.

REFERENCES

[1] R. Bartle: Rules for the Spellcaster Game [Online]. Available: https://www.andrew.cmu.edu/user/gc00/reviews/spellcaster.html [Accessed 29 Nov. 2023].

[2] J. Lean, J. Moizer, M. Towler and C. Abbey: Simulations and Games, Active Learning in Higher Education, Vol. 7 (2006) No. 3, pp.227–242. doi: https://doi.org/10.1177/1469787406069056

[3] S.J. Russell and P. Norvig: Artificial Intelligence: A Modern Approach [Online]. Pearson. Available: https://thuvienso.hoasen.edu.vn/handle/123456789/8967 [Accessed 24 Mar. 2024].

[4] N.-S. Chen and G.-J. Hwang: Transforming the Classrooms: Innovative Digital Game-based Learning Designs and Applications, Educational Technology Research and Development, Vol. 62 (2014) No. 2, pp.125–128. doi: https://doi.org/10.1007/s11423-014-9332-y

[5] Rogerson, M.J., Gibbs, M. and Smith, W.: 'I Love All the Bits': The Materiality of Boardgames, Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (2016), Vol. 1, No. 1, pp.3956-3969. doi: https://doi.org/10.1145/2858036.2858433

[6] Zin, Yue, W.S., & Jaafar, A.: Digital Game-Based Learning (DGBL) Model and Development Methodology for Teaching History, WSEAS Transactions on Computers, Vol. 8 (2009) No. 2, pp.322-333. [Online]. Available: https://www.researchgate.net/profile/Nor-Azan-Mat-Zin/publication/282054005_Digital_game-based_learning_DGBL_model_and_development_methodology_for_teaching_history/links/02e7e529e7d96311f1000000/Digital-game-based-learning-DGBL-model-and-development-methodology-for-teaching-history.pdf [Accessed 03 Nov. 2023].

[7] Robertson, G. and Watson, I.: A Review of Real-Time Strategy Game AI, AI Magazine, Vol. 35 (2014) No. 4, p.75. doi: https://doi.org/10.1609/aimag.v35i4.2478

[8] Hoang, H., Lee-Urban, S., and Muñoz-Avila, H.: Hierarchical Plan Representations for Encoding Strategic Game AI, Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Vol. 1 (2021), No. 1, pp.63–68. doi: https://doi.org/10.1609/aiide.v1i1.18717

[9] Butterfield, T.: The Power of Playful Learning | Harvard Graduate School of Education [Online]. Available: https://www.gse.harvard.edu/ideas/askwith-education-forum/19/10/power-playful-learning [Accessed 27 Nov. 2023].

[10] Dicerbo, K.: Taking Serious Games Seriously in Education [Online]. Available: https://er.educause.edu/articles/2015/7/taking-serious-games-seriously-in-education [Accessed 27 Nov. 2023].

[11] Yu, Z.: A Meta-Analysis of Use of Serious Games in Education over a Decade, International Journal of Computer Games Technology, 2019, pp.1–8. doi: https://doi.org/10.1155/2019/4797032

[12] Marius Marinel Stănescu, Barriga, N.A., Hess, A., and Buro, M.: Evaluating Real-Time Strategy Game States Using Convolutional Neural Networks, 2016 IEEE Conference on Computational Intelligence and Games (CIG), 2016, pp.1–7. doi: https://doi.org/10.1109/cig.2016.7860439

[13] Smith, G., Whitehead, J., and Mateas, M.: Tanagra, Proceedings of the Fifth International Conference on the Foundations of Digital Games - FDG '10, Vol. 1 (2010) No. 1, pp.209–216. doi: https://doi.org/10.1145/1822348.1822376

[14] Arnab, S., Lim, T., Carvalho, M.B., Bellotti, F., de Freitas, S., Louchart, S., Suttie, N., Berta, R., and De Gloria, A.: Mapping Learning and Game Mechanics for Serious Games Analysis, British Journal of Educational Technology, Vol. 46 (2014) No. 2, pp.391–411. doi: https://doi.org/10.1111/bjet.12113

[15] Zagal, J.P., Rick, J., and Hsi, I.: Collaborative Games: Lessons Learned from Board Games, Simulation & Gaming, [Online] Vol. 37 (2006) No. 1, pp.24–40. doi: https://doi.org/10.1177/1046878105282279

[16] Rogerson, M.J., Gibbs, M., and Smith, W.: Digitising Boardgames: Issues and Tensions, Proceedings of DiGRA 2015; Diversity of Play [Online]. Available: https://web.archive.org/web/20211025230459id_/http://www.digra.org/wp-content/uploads/digital-library/66_Rogerson-etal_Digitising-Boardgames.pdf [Accessed 23 Oct. 2023].

[17] Nicholson, S.: Making Gameplay Matter: Designing Modern Educational Tabletop Games, Knowledge Quest, Vol. 40 (2011) No. 1, pp.60-65. [Online]. Available: https://www.proquest.com/docview/912513342?pq-origsite=gscholar&fromopenview=true&sourcetype=Scholarly%20Journals [Accessed 24 Mar. 2024].

[18] Haas, J.: A History of the Unity Game Engine, Digital WPI Interactive Qualifying Projects (All Years) Interactive Qualifying Projects [Online]. Available: https://core.ac.uk/download/pdf/212986458.pdf [Accessed 09 Nov. 2023].

[19] Sharif, K.H., and Yousif Ameen, S.: Game Engines Evaluation for Serious Game Development in Education, 2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM), 2021, pp.1–6. doi: https://doi.org/10.23919/softcom52868.2021.9559053

[20] Kosidło, P., Kowalczyk, K., and Badurowicz, M.: Comparison of Capabilities of the Unity Environment and LibGDX in Terms of Computer Game Development, Journal of Computer Sciences Institute, Vol. 21 (2021), pp.324–329. doi: https://doi.org/10.35784/jcsi.2735

[21] Bradfield, C.: Godot Engine Game Development Projects: Build Five Cross-platform 2D and 3D Games with Godot 3.0, Packt Publishing Ltd. [Online]. Available: https://books.google.com/books?hl=en&lr=&id=KMNiDwAAQBAJ&oi=fnd&pg=PP1&dq=Godot+engine&ots=z10EYUIrXZ&sig=6LZ9TK3aQAf3eFx1pPIh-ZuZ8-Y [Accessed 13 Nov. 2023].

[22] Ali, Z., and Usman, M.: A Framework for Game Engine Selection for Gamification and Serious Games, 2016 Future Technologies Conference (FTC), 2016, pp.1199–1207. doi: https://doi.org/10.1109/ftc.2016.7821753

[23] Fun Man Fung, Lam, Y., Yap, J., Dawoud Abdullah Musalli, Jia Yi Han, Kenzo Aki Togo, and Kim, Y.-B.: ChemPOV: Digitising an Organic Chemistry Boardgame to Support Online Learning, 2021 IEEE International Conference on Engineering, Technology & Education (TALE), 2021, pp.905–909. doi: https://doi.org/10.1109/tale52509.2021.9678765

[24] Perrin, A.-F., Ebrahimi, T., Zadtootaghaj, S., Schmidt, S., and Moller, S.: Towards the Need Satisfaction in Gaming: A Comparison of Different Gaming Platforms, 2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX), 2017, pp.1–3. doi: https://doi.org/10.1109/QoMEX.2017.7965641

[25] Sparrow, L.A., and Rogerson, M.J.: Lessons from Homebrewed Hybridity: Designing Hybrid Digital Boardgames for Distanced Play, Proceedings of the ACM on Human-Computer Interaction, Vol. 7 (CHI PLAY) (2023), pp.45–72. doi: https://doi.org/10.1145/3611022

[26] Dive, K., Ruplag, A., Rane, P., Gawali, P., and Diwakar, S.: Design and Development of Digital Board Gaming System, 2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA), Vol. 17 (2017) No. 8, pp.1–5. doi: https://doi.org/10.1109/ICCUBEA.2017.8463662

[27] Sparrow, L.A., and Rogerson, M.J.: Lessons from Homebrewed Hybridity: Designing Hybrid Digital Boardgames for Distanced Play, Proceedings of the ACM on Human-Computer Interaction, Vol. 7 (CHI PLAY) (2023), pp.45–72. doi: https://doi.org/10.1145/3611022

[28] Luiz, J.P., Charikova, M., Sales, J.E., Smirnov, V., and Thapaliya, A.: Towards a Game-Independent Model and Data-Structures in Digital Board Games: An Overview of the State-of-the-Art, Proceedings of the 14th International Conference on the Foundations of Digital Games, Vol. 19 (2019) No. 93, pp.1–8. doi: https://doi.org/10.1145/3337722.3342238

[29] Plaat, A., Schaeffer, J., Pijls, W., and Bruin, A.: Best-First Fixed-Depth Mini-Max Algorithms, Artificial Intelligence, Vol. 87 (1996) No. 1-2, pp.255–293. doi: https://doi.org/10.1016/0004-3702(95)00126-3

[30] Dreżewski, R., and Solawa, J.: The Application of Selected Modern Artificial Intelligence Techniques in an Exemplary Strategy Game, Procedia Computer Science, Vol. 192 (2021), pp.1914–1923. doi: https://doi.org/10.1016/j.procs.2021.08.197

[31] Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S.: A Survey of Monte Carlo Tree Search Methods, IEEE Transactions on Computational Intelligence and AI in Games, Vol. 4 (2012) No. 1, pp.1–43. doi: https://doi.org/10.1109/tciaig.2012.2186810

[32] Ontanon, S.: Informed Monte Carlo Tree Search for Real-Time Strategy Games, 2016 IEEE Conference on Computational Intelligence and Games (CIG), 2016, pp.1–8. doi: https://doi.org/10.1109/cig.2016.7860394.

[33] Wikipedia: Monte Carlo Tree Search [Online]. Wikipedia, 2019. Available: https://en.wikipedia.org/wiki/Monte_Carlo_tree_search [Accessed 16 Jan. 2024].

[34] Baier, H., and Winands, M.: Monte-Carlo Tree Search and Mini-Max Hybrids, 2013 IEEE Conference on Computational Intelligence in Games (CIG), 2013, pp.1–8. doi: https://doi.org/10.1109/cig.2013.6633630

[35] Yakovenko, N., Cao, L., Raffel, C., and Fan, J.: Poker-CNN: A Pattern Learning Strategy for Making Draws and Bets in Poker Games Using Convolutional Networks, Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 30 (2016) No. 1. doi: https://doi.org/10.1609/aaai.v30i1.10013

[36] Riechmann, T.: Genetic Algorithm Learning and Evolutionary Games, Journal of Economic Dynamics and Control, Vol. 25 (2001) No. 6-7, pp.1019–1037. doi: https://doi.org/10.1016/s0165-1889(00)00066-x