CS4303 P4 Group Report

Introduction

This project aims to develop a card-based role-playing game. Players will embark on adventures in a randomly generated endless world, encountering a variety of events and utilizing their resources to overcome them. The purpose of this report is to provide a comprehensive analysis and interpretation of the game's content, examining it from five distinct perspectives: Design, Implementation, Context, Evaluation, and Critical Appraisal.

Design

Event and Card System - 190003505

The game title is "Deckmaster's Adventure", a single-player role-playing game that is based on card mechanics and resource management. Players assume the role of an adventurer in a medieval fantasy world, utilizing magic, items, contacts, and even life as resources to resolve various events.

In this game, the adversaries that players need to face are the events encountered on the map, which may consist of any scenes that can be encountered in the real or fantasy world, including battles, conversations, trades, and other events. The succeeding sections will provide an in-depth explanation of the event encounter process and the information contained in the various game components.

In this game, the players' cards possess distinct properties that represent the card's functionalities and meanings. For instance, a sword represents the ability to cause damage and engage in combat and also signifies a certain level of danger. When players encounter an event, a user interface will appear on the screen with several empty card slots for them to fill in the cards. These cards will be removed once the event is concluded. Each event will display different outcomes depending on the total value of the properties of the cards filled in by the player. For instance, a player facing a conflict can use a sword card to defeat the enemy or use a serpent card to silently evade the battle. It is important to note that some events have their own subsequent events. After resolving an event, a new event will be added to the world, replacing the original event.

Meanwhile, in this game, players are allowed to actively utilize their cards to achieve desired effects. The most common example is that players can use ritual cards or contact specific characters at any time. To use cards proactively, players can trigger a special event in the world by pressing the "T" key and fill in the cards following the normal event process.

Apart from the rewards and results after the event (results may increase or decrease specific cards such as reducing life or obtaining random cards), players can also search for treasure chests in the world to obtain random card rewards. It should be noted that special cards, such as most special NPCs, cannot be found in treasure chests.

The ultimate objective of this game is to defeat a creature known as the "Dusk Dragon," which can be regarded as the final boss of the game.

World Generation - 190002809

The world generation was a vital part of this game as the game was imagined from the start to have an infinite but engaging world. The world needed to spawn some loot as cards, have events, and buildings to explore.

Initially, the world was planned to be generated with Wave Function Collapse (WFC) [1]. WFC uses techniques from Constraint Programming to take a bitmap and generate similar bitmaps. It works by taking NxN sections of the bitmap and randomly assigning one of these sections on the generated bitmap which restricts the number of possible sections the surrounding tiles can take. This is continued until all variables are assigned a valid bitmap and if a variable has no possible remaining bitmaps then the solver backtracks to a previous state. Since it creates a similar bitmap this meant that custom structures could have been added to the bitmap which would make it populate the world with an infinite variety of similar structures.

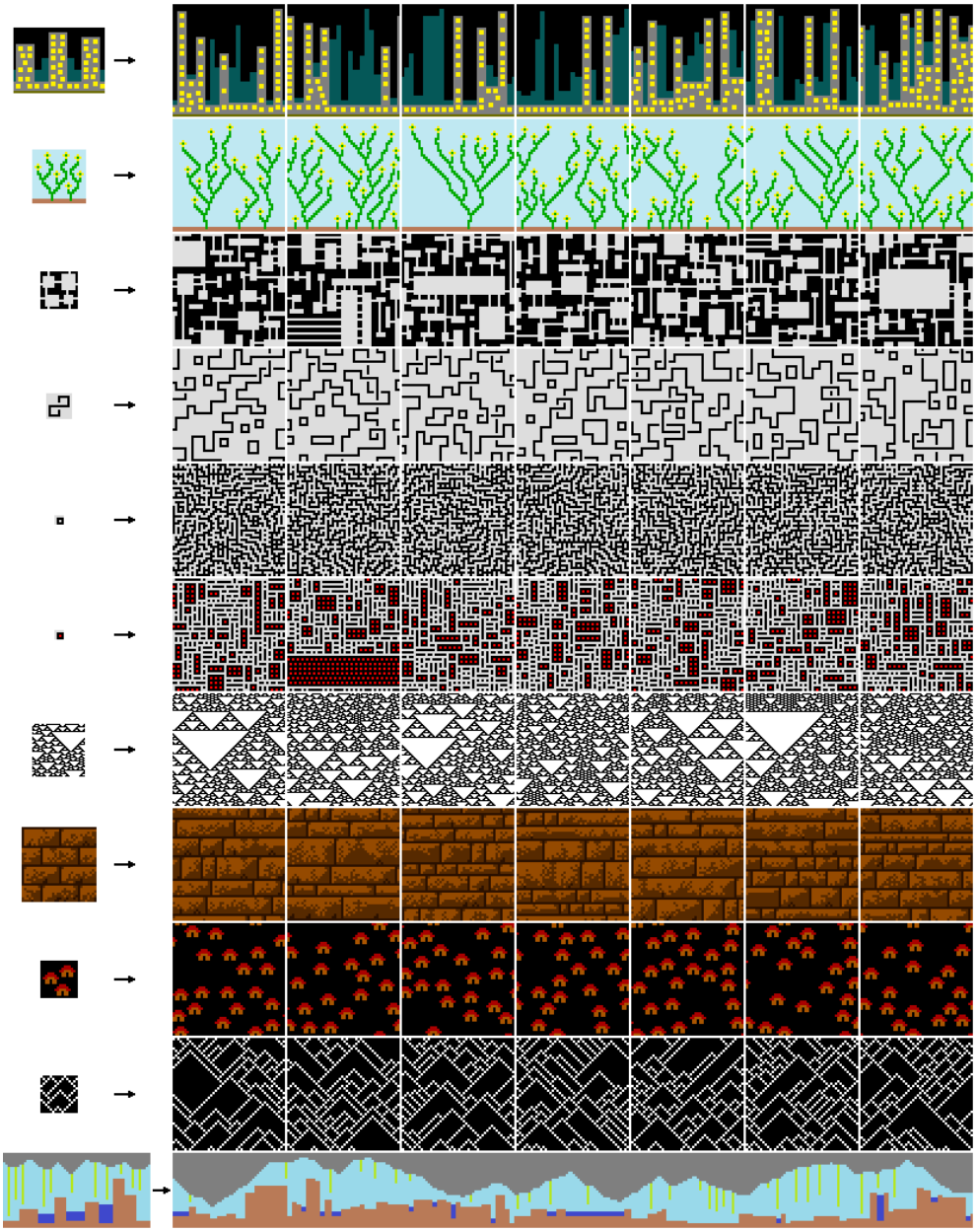

Figure: Examples of bitmaps that Wave Function Collapse can generate from the input bitmap on the left. https://github.com/mxgmn/WaveFunctionCollapse

However, given the nature of the problem and its complexity it is hard to make it solve and expand a world in real time. There is also the possibility of the entire world changing while the player is travelling around the world as the currently assigned tiles might restrict the world from expanding which could in theory backtrack to the initial node. Because of these reasons WFC was ruled out even though it would have resulted in a more engaging world and diverse world.

In the end, the default noise function from the processing library (Perlin Noise) was used to create a 3rd dimension on the 2D world. This 3rd dimension is used to colour the terrain in different values to create a visually appealing world. This height value was also initially planned to be used to restrict the player from climbing steep hills but was later discarded

as it was very confusing. Communicating the height information to the player from a top down perspective was not possible. Later this height value was used to give the effect of 3D in a 2D game.

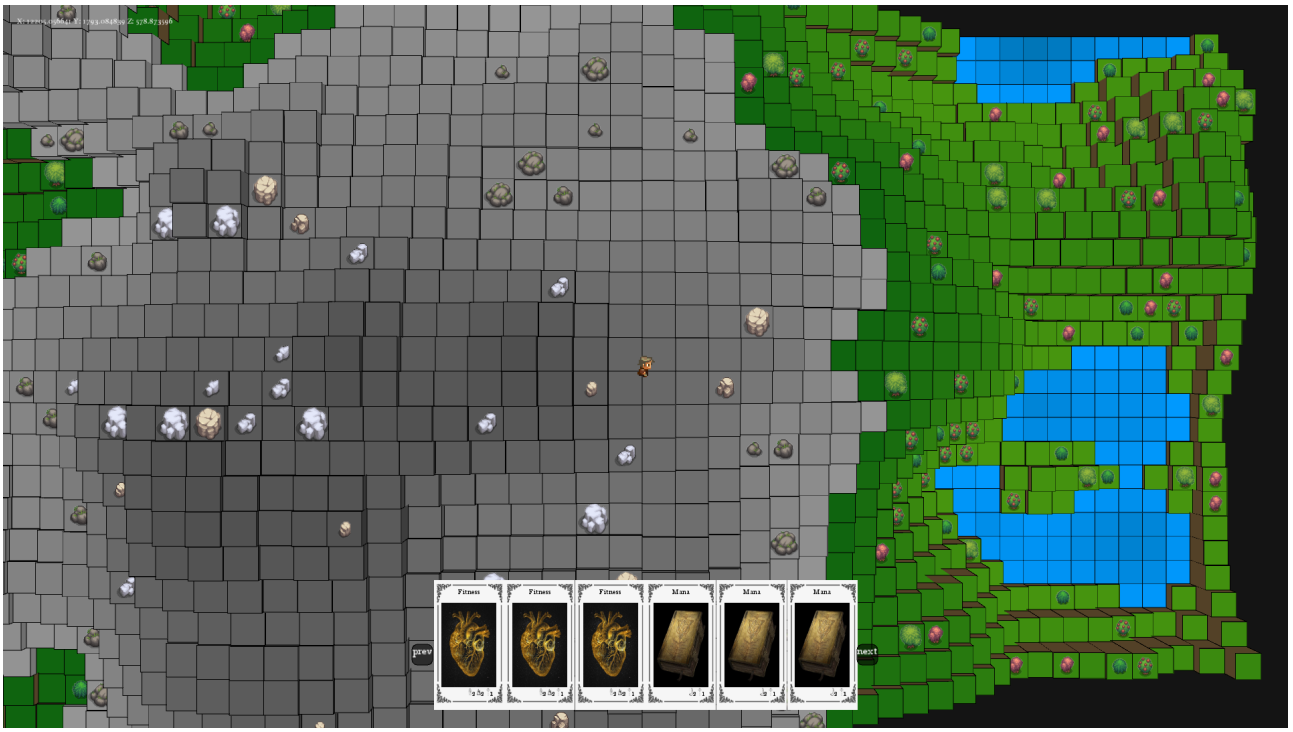

Figure: Example of the implemented world generation

The entire game is implemented using the P2D renderer. So, the 3D looking terrain is created by translating and scaling everything according to the height generated. This made the height clearly visible to the player on the edges of the screen as the camera is located on the center. However, the height difference is still not clear at the center of the screen as we are looking at it straight on.

Implementation

Event and Card System - 190003505

The following section provides a detailed description of the methods utilized in the Card and Event systems.

When implementing the card system, in addition to storing data, it is imperative that players can drag a card at any time. To ensure that the dragging behavior is visually seamless, the card should not move instantly while being dragged. Consequently, the relative position of the card and mouse should remain constant during dragging. Prior to

dragging, the position of the card and mouse are recorded, and the card moves relative to the mouse as it continues to move. These functionalities are implemented in the Card.drag() method.

Simultaneously, to implement the operation of placing a card into a card slot, the card's position needs to enter the range of the card slot. Once in the slot's range, it should be marked as paired with that slot, and the position of the card should be adjusted to match the card slot. To accomplish this, a dedicated card slot class is created, which stores the card instance paired with it. When the card is placed within the slot range and the mouse is released, the card pairs with the target card slot. When leaving the range, the card is unpaired and returned to its original position in the deck.

Furthermore, the game requires that when the mouse is placed on a property icon, a user interface should display the introduction of that property for players to understand its information. To achieve this, each card is equipped with a hashmap that saves the position of each icon and the corresponding property. When the mouse is placed on the icon, the Card object prompts the Inventory Screen instance to create a new user interface and display the corresponding property information on it (which is stored in an enum class named "property"). Originally, the display UI function was executed by the Card class. However, it was found during testing that UI displayed by a Card could be blocked by other Cards later on. Therefore, I decided to move this function to the Inventory Screen class and draw it after all Cards have been drawn.

When implementing the event system, the initial design was for each event to have only one possible success condition, and any card combination that did not meet this condition would be considered a failure. Therefore, the event class only stored a hashmap of the required properties and their values for success, as well as a list of corresponding rewards. However, testing revealed that having only one success condition could not adequately reflect the diversity of an event and did not allow players to fully immerse themselves in the game. As a result, in the final version of the game, the event class expanded its storage structure to include another hashmap that saved all possible success conditions, along with their corresponding text, rewards, and consequences.

In addition, in the initial design, after an event ended, the card would be directly added to the player's inventory without any corresponding prompt. In the final version of the game, the Event Screen class has a state system that adjusts the displayed content based on the current state (event state or result state). Therefore, any obtained cards will be shown after the event ends.

Thanks to the state system mentioned above, the end-of-game prompt in this game is also completed by events. When special conditions are met, such as defeating the final boss or the player losing all their lives, the corresponding event will be triggered. These special events have no event state and do not allow players to insert cards but directly enter the result state, achieving the effect of the prompt. It is worth mentioning that special events will not enter the list of random events.

Finally, taking into consideration the potential requirement for a large number of events and cards in this game, their specific information is stored in files and converted into instances using the Content Loader class. This class serves two functions: it converts the names of events or cards into the corresponding file names, and reads the corresponding instances from the files. The methods of this class rely on the java.util.Scanner class to read each line of the file, split the String, and create a new instance with the corresponding information. As a result, developers only need to write the game content in a .txt file in a specific format and can use this class to incorporate all the content into the game.

World Generation - 190002809

The world consists of MapTiles that are stored in a HashMap with their coordinates. The HashMap is used to quickly access the tiles that are already generated or to generate the tiles if they are not contained inside the HashMap. MapTiles hold the position they are located in, their height, and their state. This state allows the game to place the correct sprites on each tile and it also marks if the player can walk over the tile. Some tiles like events and chests can spawn an event and award the player a card respectively. These tiles get activated when the player enters their boundary and their states change to reflect that they have been used. The tiles get coloured according to their height. This height is generated by using Perlin Noise. As a rule of thumb as the player gets higher the tiles get darker. The exceptions to this is snow which is always white, and water which gets darker as the water deepens. Apart from the tiles that are marked as being unwalkable the water also cannot be entered by the player which can create massive oceans that the player must move around.

More complex structures like buildings are also located on the map, these structures can span across multiple tiles and in the case of buildings they can be entered by the player if they step on the correct tiles. These tiles contain the building as a HashMap and when the player steps on it instead of the map the building is drawn. These buildings have strict restrictions on where they can spawn to blend them in with the terrain. For example, they can only spawn on the grass and only if there is a flat enough patch. The insides of the building are generated randomly and they are more likely to contain a large number of chests so, they are landmarks that the player should look out for.

Figure: Example of a building sprite located on the map.

Figure: Example of a building interior

The fake 3D effect is achieved by scaling the tiles down according to their height difference from the player. This initially created believable terrain but there were massive holes between tiles when they were far from the player and had a large height difference. To fix this custom quads had to be drawn on screen. Drawing these quads were quite tough as each tile has a different transform matrix. For this 3D effect to look right, the order the tiles are drawn in is also important. The tiles had to be drawn from lowest to highest so

that the higher tile can obstruct a lower one but not the other way around. To achieve this a priority queue was used. At each frame the tiles that were going to be drawn get added to a priority queue and then after all of them are added the priority queue is used to draw the tiles one by one.

Context

The events system in this game are similar to those in many other games, with the closest comparison being Hand of Fate [2]. In Hand of Fate, players embark on adventures in a world composed of a deck of cards, with each card representing an event. This design effectively utilizes game text to create content-rich adventures at a relatively low cost, while retaining the freedom of traditional role-playing games.

In contrast, this game uses a traditional map and narrative style similar to Hand of Fate. However, the method of handling events is significantly different. Rather than using cards as a resource to resolve events, Hand of Fate uses dice or guessing cards to provide players with a more random way to resolve events and gain rewards. This approach results in a fixed difficulty for resolving events, which changes only with the player's dynamic vision and has little to do with the resources the player possesses. This undoubtedly weakens the player's motivation to collect resources and goes against the core gameplay of role-playing games. Therefore, this game has decided to use another method to interact with events.

The resource management system implemented in this game draws inspiration from Cultist Simulator [3]. In this game, players do not possess any attributes, such as health or money; instead, they use cards to represent all resources. This design provides players with a larger pool of resources, thereby increasing the game's freedom. For example, players can invest their health cards in a particular event. Additionally, unifying all resources into a single class simplifies game content development, making it easier to create more complex effects. For instance, in this game, the logic behind contacting NPCs is the same as that for resting. This approach undoubtedly reduces the game's development cost. However, Cultist Simulator's storytelling performance is relatively poor as players are limited to only a few types of actions, which hinders the inclusion of a significant amount of plot in the game. Consequently, the plot of the game is often obscure and straightforward. To address this issue, this game chooses to employ a traditional form of role-playing games and utilizes only its resource management system.

Evaluation

The testing methodology employed in this project involves the developer personally playing through the current state of development after each phase to identify any

shortcomings. The subsequent paragraphs outline the changes that were implemented during the course of development.

Initially, events only had one success condition, but in later stages, more success conditions were added to provide players with additional strategic options. Additionally, an interface was added in subsequent development stages to display the results and rewards of events, as there was no dedicated prompt screen in the original design.

Furthermore, special events, such as those resulting in death or triggered by the player, were occasionally added to the game world during the process of introducing random events. To prevent specific events from being added to the game world, each event was assigned a boolean attribute.

In the original design phase, the game was supposed to include a game-over screen to display relevant information. However, after careful consideration, it was decided that displaying the game-over information on the event screen would enhance player immersion. Therefore, the game over screen was removed, and players were directed back to the main menu after an event was triggered.

Lastly, the game was designed to return the card to the player in the event of failure. However, given that it is relatively easy for players to obtain cards from treasure chests, the decision was made to remove the card used in the event at the conclusion of the event, regardless of whether it was successful.

Critical Appraisal

As noted in the previous comparative analysis between this game and others on the market, this game leverages the element of cards to comprehensively simulate the interaction between players and the game world. This enables players to freely engage in a variety of real-world activities, including combat, communication, rest, and stealth, resulting in a higher degree of player freedom than traditional role-playing games. Additionally, due to the low development costs of this system, it is relatively easy to develop new behavior systems. For instance, if one intends to add horror elements to the game and introduce a sanity attribute, only a new card and corresponding events and outcomes for that card need to be developed.Furthermore, the game's pseudo-3D map also enhances player immersion.

However, the game's UI design is rather crude. The most significant design flaw in the UI after playing the game is that event results only show rewards and do not provide a means to display unique results, such as reduced health. This forces players to actively check their inventory to see the consequences of events. Moreover, the player's inventory can only display a maximum of eight cards at a time, making it challenging to check items when the player has numerous cards.

Conclusion

This project has effectively developed a role-playing game based on card mechanics and resource management. A comprehensive analysis has been conducted on its design and development, followed by a thorough comparison with other games available in the market to assess its strengths and weaknesses.

References

[1] Gumin, Maxim. n.d. “mxgmn/WaveFunctionCollapse: Bitmap & tilemap generation from a single example with the help of ideas from quantum mechanics.” GitHub. Accessed April 15, 2023. https://github.com/mxgmn/WaveFunctionCollapse.

[2] Defiant Development, “Hand of Fate” https://store.steampowered.com/app/266510/Hand_of_Fate/ , accessed:May 9, 2023

[3] Weather Factory, “Cultist Simulator” https://weatherfactory.biz/cultist-simulator/ , accessed:May 9, 2023