Kevin Duong, Maddy Kalaigian , John Klopcic , Clea Ramos
Professor Romano, Stough
CSCI 205- Software Engineering and Design
Apr 29, 2024

Design Manual

Introduction

For this project we wanted to recreate the game and be able to play its basic functionalities by using an MVC model to practice using industry software engineering practices and object oriented design techniques. We determined that the fundamental game play was achievable for the time frame of the project, so we omitted the more complex features of the game when creating the minimum viable product (MVP).

When starting the analysis phase of the project, we conducted background research on how the game works, how to play it, and by identifying the core components and actions of playing the game which corresponded to the classes and methods within our program. Afterwards, when completing our CRC cards and UML diagrams, we made sure to think about the different ways to implement high cohesion by ensuring that the actions or methods are appropriately associated with each of the corresponding classes. Structuring these components like so made the process of creating the controller more straightforward, streamlined, and modular, since we thoughtfully planned this process. This also resulted in high cohesion and low coupling among our classes. By following the MVC model, we are ensuring that our game is modular and compatible with added game functionalities and features.

Our system uses the Model View Controller layout to organize the structure of everything. Contained within the view is the raw images and UI elements we are showing to the user. In the model we have all of our “functional” classes. That is to say anything that requires very complicated unseen functionality relies on a class from the model. The controller is what connects our view to the model, it takes values from various objects in our model and connects them to the relevant UI elements on the screen.

Overall, we have five main classes that work together to give the vast majority of the game functionality. We have the Card class which has all the relevant fields to track important actions for the player and monster. The Deck class is composed of cards and has a few methods that can perform operations on the cards. The Player class has a Deck and interacts with the monster. The monster has a set of cards to represent its attack patterns and interacts with the Player. Finally, the game model is composed of a Player and Monster and keeps track of a variety of important attributes. A lot of the functionality is built upon the Cards and how their fields interact with other classes.

User Stories

Janet Jorell - Killer:

In our game, we did not implement many features that would appeal to Janet's likings. She would prefer the game to be a head on head challenge with another player or a game that is conducive to speedrunning. So while we did make sure our animations didn't slow down the gameplay too much, there aren't many features that she would like. We also implemented a reset button catered to speedrunners, so they could continuously reset their game to achieve an optimal hand.

Jean Jacques - Aesthetics:

We have quite a few things in our game that would appeal to Jean. Jean prefers games that are aesthetically pleasing, such as *Ori and the Blind Forest* and *Abzu.* To appeal to this user, we integrated unique images for each card. This requires a specific aspect of our card objects that knows where its image should be in the resource folder. This is used by the controller to connect what the cards look like to the view. Furthermore, we designed all the animations for both the player and the monster, and based them on the characters in the actual game. All of this so that Jean will be happy with how pretty the game is.

Jimothy Jorkins - Achiever:

We have a variety of methods that supports Jimothy's ideal gameplay. There's a few easter eggs that rely on various aspects of the different classes. For instance, the score is a compilation of a variety of fields from the player, turn count, and monster. The turn count has to be manually incremented with an interaction between the controller and the view. While the player, monster, and the model have to collaborate to get the score. This score is then used to give Jimothy something to work towards in his play throughs.

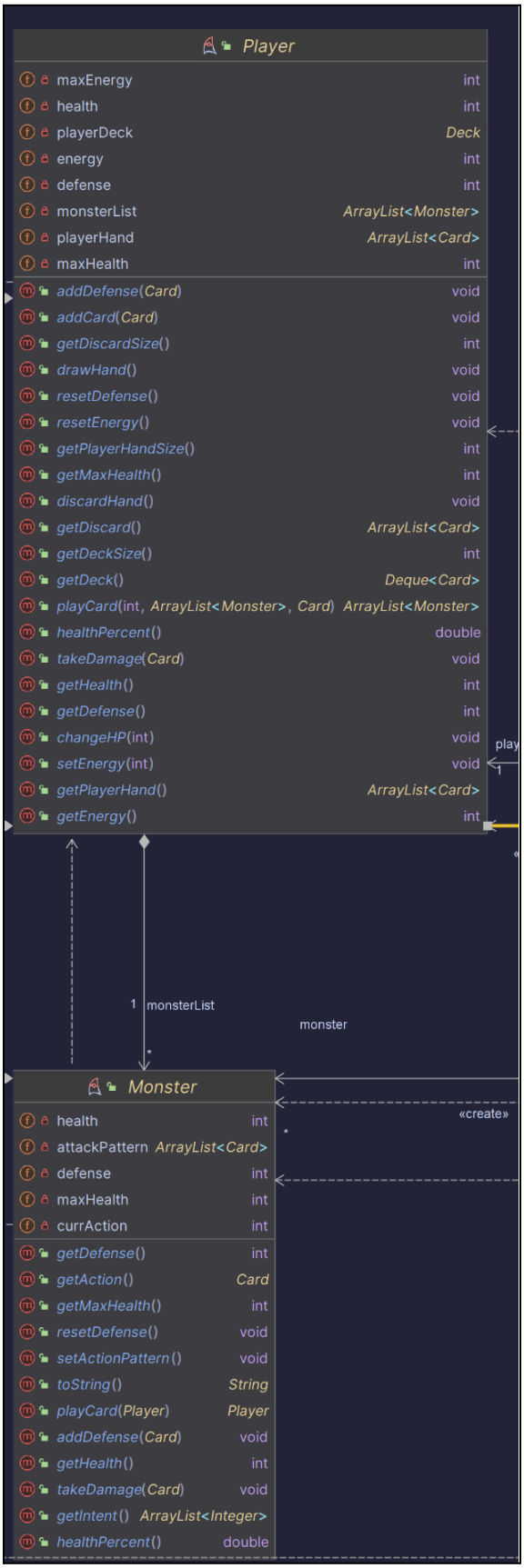
Jorg Jorggenson - Newbie:

While we have no implementation that works towards Jorg, we would be able to add some in the future with a little more time. As somebody new to games like this, Jorg would benefit greatly from a tutorial or help screen, which we can add by creating another screen similar to how we created the end screen. This new screen would contain information on how to play the game and what the goal is. It would require using the existing controller and view and simply adding a little more onto it.

Justine Jorbs - Expert Nerd:

As mentioned above, the score is a conglomerate of various fields from our game. As an expert in the game, Justine would know exactly how the score is calculated and would know how to manipulate it to get various easter eggs we had sprinkled throughout the game. She would also enjoy the various card interactions and ideal card plays. The cards themselves are supported as a distinct object that we can easily customize by simply importing in a new CSV file. So adding more cards and more functionality is not too difficult due to the modular design of our game.

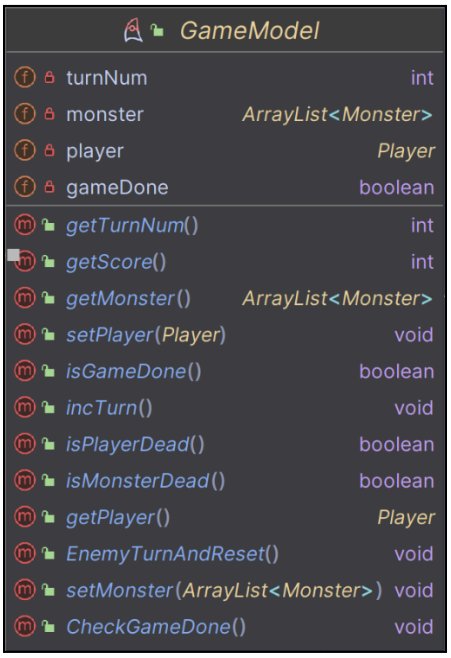
Object Oriented Design

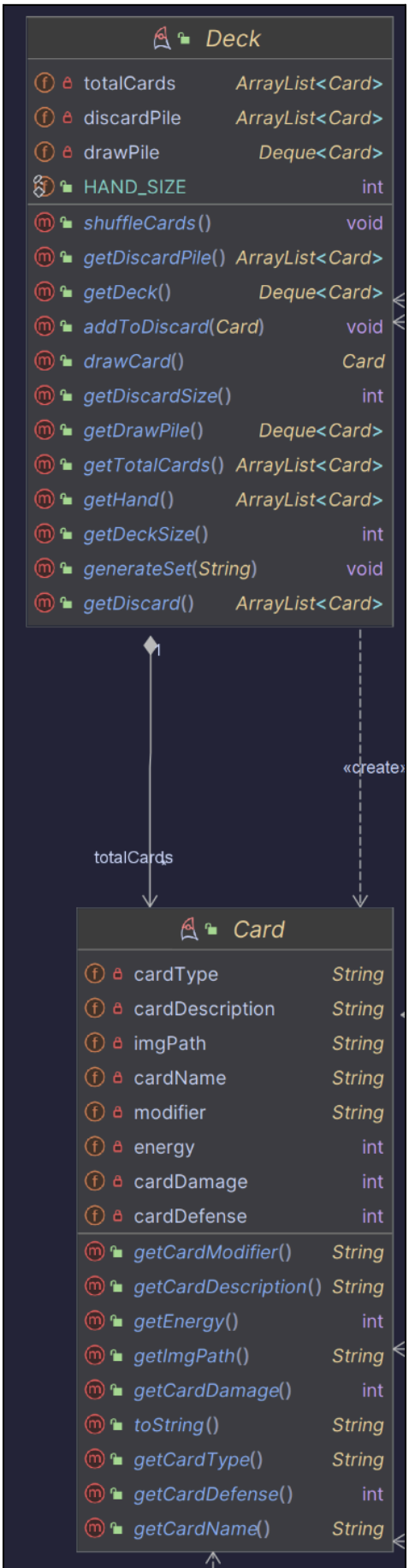


The design of our project relies on the interaction between the Player and the Monster. The Player and Monster both share common methods to change the values of the common variables such addDefense(), resetDefense(), healthPercent(), takeDamage(Card), etc. The key function relies on our playCard() in each Class, the Player's playCard() takes in the index of the monster being targeted, the array of monsters, the card being played, and will return the updated array which overwrites the previous array. Similarly, the Monster's playCard() takes in a player and returns the updated Player object. These objects are created and used in the GameModel class containing a score calculated based on the status of the variables in the Player and Monster objects.

The Monster class contains an array of cards, which are the actions that the monster would perform on the Player object during each of its turns. Cards are inserted into the array with setAttackPattern() and everytime playCard() is called, it loops it's actions.

The Player class contains a Deck object which contains a queue of several Card objects. The drawHand() method would add 5 cards to the Player's hand array, via the view, these cards can be selected which would call the playCard() and perform the action by calling their respective method on the Monster object(s) or the Player. These actions can be performed as long as the Player's energy variable can pay the cost of the Card's energy attribute.

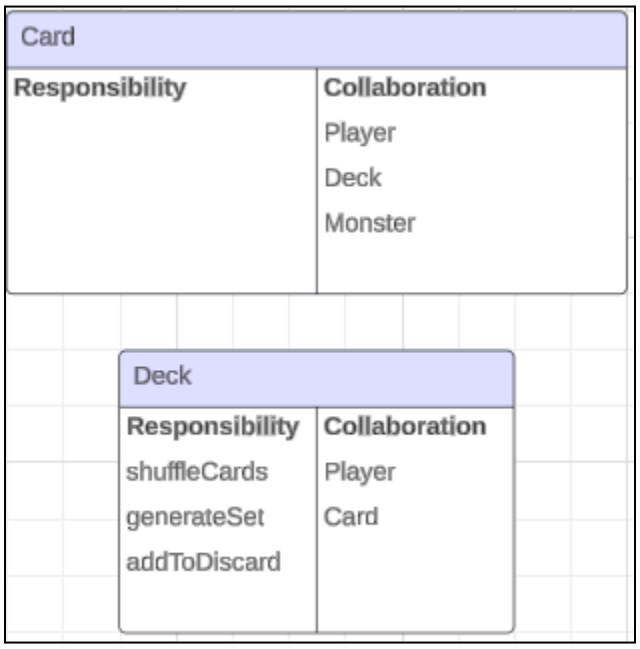




Moving onto the Deck and Card Class. The Deck is an object that contains any number of cards. The number of cards in the hand is currently limited to 5 with the potential for more cards.

The Deck class takes in the path to the file in the resources folder which contains the .csv of the deck where every line, excluding the first, should be a different card. The generateSet() would take that file and generate each Card object from the .csv.

The Card class is a class that only contains attributes and getters of the cards gathered through the .csv. Some important notes about some of our variables is that the modifier attribute is used for additional effects on cards outside of the standard “Inflict X damage” or “ Block for X” and the imgPath is the path in the resource folder to the card itself.



Now that the Deck class contains every card in totalCards, it then takes those cards and generates a queue of Cards in a random order. When the Player calls its drawHand(), it would call the getHand() function to take cards from the queue and pop them into the player’s hand. Upon attempting to draw 5 cards, it reshuffles the discard pile back into the deck in a random order and draws 5 cards.

There are very particular implementations which contribute to the user personas that we have established. Some of the main things within our Object Oriented Design which appeal to the user stories are our score attribute in GameModel, the easter eggs we put into the end screen in gotoEndScreen() within gameViewController, the gotoMainScreen() binded to reset button also within gameViewController. To elaborate further, the score printed at the end screen would be benefitting Jimothy Jorkins because he thrives on getting better and better scores. The different end screen text appeals to Justine Jorbs due to having unique messages for completing specific conditions upon finishing the game. The reset button present in the view allows for optimal speedrunning capabilities for Janet Jorell because it allows more efficient resetting for the “optimal” hand. We built our MVC with the user personas in mind and designed it to be very modular with a lot of room to add more functionality on top of what we had. This allowed us to add things as we desire such as if we wanted to add more easter eggs, more cards, more unique interactions. With further updates we decide to add to the game, we can add more to appeal to different user personas and potentially expand on our user stories.