CS143 - Professor: Anderson
G. Sattelberg, V. Bun, Y. Hammat, S. Cha
December 7th, 2021

Hi Chew - Project Report

Overview

The Great Sacrifice is a text-based turn by turn dueling card game. Simply, it is a two player game where each player draws cards and then selects the ones they will play each turn. This card duel battle system is inspired by the game Hearthstone. Two players choose cards and use them strategically to inflict damage to their opponents. There is a bit of luck involved in getting cards from the deck, but overall it requires a significant amount of strategy from the player. Each player will start with a deck filled with 20 cards. Players will have the same cards in their deck as the other player, but the decks will be in a randomized order. Each player will start with 15 health, and 5 cards in their hand at the beginning of the match. A dice will be rolled to determine who goes first and the player who rolls the highest number will go first. There are 3 different classes of cards: attack, defend, and heal. Every turn a player chooses the 2 cards they will use. A player will draw 3 cards after their turn. After each player selects their cards they enter the battle stage. Attack card deals to the enemy player. Heal card's will heal the player. Defend cards neutralize all damage regardless of how many attack cards the enemy used. Psychological warfare between players, who are going to use attack and defense cards. Players who run out of defense cards first are at greater risk of losing the game. As the battle progresses the amount of cards a player has in their hand will continually increase. If the cards run out before any player's health reaches zero, the player's health will lose 1 health per round. As if a defender type was on the playing field. You are not able to attack player health. If a health type card is on the playing field. Their player health increases by one per card per start of their turn.

Requirements

- Game State
    - Is maintained by a while loop
    - After each player's turn, the program will check and print out the status of the game to the console until a player's health reaches zero

- Exception Handling
    - Most importantly is the FileNotFound Exception. With the use of CSV file
- Create a list of cards
    - The cards data type will be Strings and will be stored in an ArrayList
    - The quantity of cards will be limited to 20
- Major Rule
    - The program will track the quantity of cards in hand and the number of health cards each player has.
    - Player loses the game If he reaches zero health cards first.
    - If the cards run out before any player's health reaches zero, player's health will lose 1 health per round
    - if a defender type was on the playing field. You are not able to attack player health. If a health type card is on the playing field. Their player health increases by one per card per start of their turn.
- Game text template
    - Using recursive methods to recursively check the winner and cards in hands.

Our method will be created based on the actions of the games. They might include (but are not limited to) attack, heal, defense, draw, play_cards, track a players health. Those decisions are totally on the Players.

Implementation

Our team used Repl.it to work on code simultaneously. Repl.it allows users to write code and create apps and websites using a browser. Group members can share the code in real time by registering on the site by email.

- In CardStats Class at Line 28

```
if (!row[0].equals("") && row.length >= 1) {
    card.push(cardName);
    cardCount++;
}
```

The “cardName“ from the CardStats.csv files is push(added) onto the Stack.

- At line 48 an array is created.

```
    }
}
this.deck = new String[cardCount];
for (int i = cardCount - 1; i >= 0; i--) {
    deck[i] = card.pop();
}
```

The Stack is then pop (remove) and then added to the array starting at the index at the end of the array. (To keep ordering). This array is used when passed into the Deck class.

- In the Deck Class line 9

```
public Deck(String[] card) {
    size = card.length;
    for (int i = card.length - 1; i >= 0; i--) {
        deckOfCard.push(card[i]);
    }
    this.cardName = new String[size];
    for (int i = 0; i < cardName.length; i++) {
        cardName[i] = deckOfCard.pop();
    }
    for (int i = cardName.length - 1; i >= 0; i--) {
        deckOfCard.push(cardName[i]);
    }
}
```

The array is passed down to this class and it is then push(added) starting at the index at the end of the array to maintain ordering. Index 0 will be at the top of the deck. Meaning it is on top of the stack.

- On line 23 in the deck class. The method shuffles change the ordering.
    - Basically, the Deck class and CardStats Class uses Stack to create the deck of cards from the CardStats.cvs file.
- The CardStats class is more FOCUS on creating HashMap and keeping track of the card stats.
- The HandAndField Class uses LinkedList so when a card is pop(remove) from the top of the deck. It is added to the Linklist. This is use to keep track of the player hand and player field

- Player Class just uses all the classes created and I create some methods to help keep track of the game.
- TheGreateSacrifice class is just the while loop of the entire game.

So in all we use array, Stack, Hashmap, and Linklist.

- For the Exceptions handling ... Can be found on Line 176 and 270 in Player Class.

```
if (playerField.size() < 3 && !cardIndex.equals("")) {
    try {
        int handIndex = Integer.parseInt(cardIndex) - 1;
        playerField.addCard(playerHand.chooseCard(handIndex));
        counter++;
    } catch (IndexOutOfBoundsException e) {
        System.out.println("Index enter was invalid");
    } catch (NumberFormatException e) {
        System.out.println("Index enter was invalid");
    } finally {
        getHandStats();
    }
}
```

Because we don't want the user to enter value, we don't want like picking an index that doesn't exist in the linked list we use an index out of bounds exception. Also, the number format exception is used just in case the user input doubles as an input or letters or blanks space. This then calls the method again like a recursion.

Testing

Each group member's personal schedule was different, so it was very difficult for us to have time to discuss or work together. repl.it is perfect for combining code we've worked on individually, without having to work together. After the code was completed, the program was executed several times and various variables were applied, but no errors or bugs were found.

Conclusions

More often than not, when entering the programming workforce it is required to work with others. That usually entails simultaneously working on a project with other programmers. When creating this game we explored different options to asynchronously work. At times it was difficult to understand another member's program. Although writing comments takes more time, it comparatively saves time in the future.