Señor Power APCS2 pd 4
Nicolas Naing, Chris Yu 5/18/17

Hearthstone

Overview

Hearthstone is a popular player vs. player card game made by Blizzard Entertainment. Players use cards to reduce their opponent's health to 0. Players can choose from 3 different "heroes," each with their own unique cards and abilities. We will try to implement as many features and cards from the original game that we can.

User Experience

Players interact with the hand/board, choosing targets for their card effects and declaring attacks with monsters already on the board.

Minimum Viable Product

Our MVP would be a board where we can click and drag cards.

Showcase

Stacks

The decks will be implemented as stacks. Drawing a card from the deck will utilize pop(), and returning cards to the deck will utilize push(). There are card effects that reveal the next card in the deck but doesn't remove them, which can be implemented using peek(). We can also keep track of the number of cards in a deck by calling size().

Inheritance

There are several hero classes in Hearthstone, each with different decks. However, some heroes have cards in common, so we could have each hero class be a subclass of a general Character class and save time by implementing the methods in the superclass.

Processing

We plan to use Java Processing to implement this game. Some programs that we would like to use are
LoadDisplayImage - This would put the background board and the available cards on the screen
MouseFunctions - Cards will be moved by the mouse, and most of the MouseFunctions methods will be useful for this
fill() - We can use fill() to give the currently selected card a gold border, which makes it much easier to see which card is about to be played
rect() - this can be used to create the end turn button in the original Hearthstone

ArrayList

ArrayLists would be useful because they can easily add and remove elements. This is necessary for card games like Hearthstone, where cards are being added and removed from the current hand each turn.

Polymorphism
Spells and weapons both attack, but they have different effects on the player(s) afterwards. This calls for polymorphism because we could have the signature of both arguments be the same while having different arguments and different code blocks.

Queue
A queue is necessary to keep track of which card's effect is currently taking place and which effect has to take place immediately after. For example, if Card A's effect destroys two random monsters, and Card B's effect activates when one of those monsters is destroyed, Card A's effect must finish before Card B's effect activates, and Card B's effect must take place immediately after Card A is finished so that another card cannot be activated until Card B is finished.

Priority Queue
A priority queue is also necessary because a player's health getting reduced to 0 has a greater priority than any card effect. If a player queued a bunch of card effects, but the first one reduces the opponent's health to 0, the game will just end instead of going through the other effects.

Extensions
- Win/Loss percentage for each Hero
- Playing another game after one is over