Jack’s Arcade: Design

By Sahil Kapur

1. How to run code

After downloading/cloning the repository onto the local machine, navigate to the root of the project directory.

Type “./build.sh” and the arcade will load. Players pick the game by entering “tokens” (typing in a number). 1 token will run Slapjack. Two tokens will run Magic Jacques. All other inputs will end the session.

Project must be executed from Linux terminal.

1. Introduction

Background

As part of the application process for the 2020 Kleiner Perkins Fellowship, applicants are tasked with creating “a console-based (i.e. runs in terminal) unix-compatible interactive card game.”

Desired Outcomes

The goal of this projects is to showcase engineering talent and creativity during the application review process. Key stakeholders include the reviewers of the application.

Brainstorming Process

During ideation, I began by simplifying the question to it’s core: to create an interactive experience using playing cards, through a console-based medium. Starting from this, I was able to generate a mind map of all activities a player can use cards for. This mind map included card games, throwing cards, magic, etc. I narrowed it further to activities that fit the prompt best, then did a second round to narrow activities that were feasible through a console.

As I continued to brainstorm terminal-based games, I remembered the classic “Hitchhiker’s Guide to the Galaxy” and wondered if there was way to rebuild a story-like experience, while fitting the prompt of building “interactive card games.”

Game Selection

I first developed “Slapjack,” as it was a simple game to understand for new users, and relatively easy to develop compared to most card games.

Once this was developed, I began crafting an adventure game, centered around the theme of “cards.” I recognized this might not be viewed as a *card* game, however, part of the prompt assesses candidates on creativity on their submission, therefore I concluded this would stretch this prompt just enough to showcase creativity.

1. Playing Games

**Slapjack**

Time to play: ~3 min

The goal of Slapjack is to **win over all the cards**. The game finishes when all other players run out of cards. The player who holds all the cards at the end wins.

Each player is given a pile of cards, face down, and is not allowed to look at their cards. Players take turns placing their top card in a center pile, then revealing the card. If the player is not a Jack, nothing happens and the pile grows. Otherwise, the card is a Jack, the first player to slap the Jack wins the round and collects all the cards in the pile. As the pile grows larger, the stakes grow larger.

In our modified version, players play against a CPU. The program will automatically place the cards in the pile for each player and keeps track of how many cards each player has. The task of the player is to quickly respond and “slap” the card before the CPU responds. The CPU gets faster as the game progresses.

**Magic Jacques**

Time to play: 5 - 7 min

One of the most fascinating uses of playing cards is magic. Though re-creating the experience through a computer is challenging, Magic Jacques explores a different kind of magic: the magic of story.

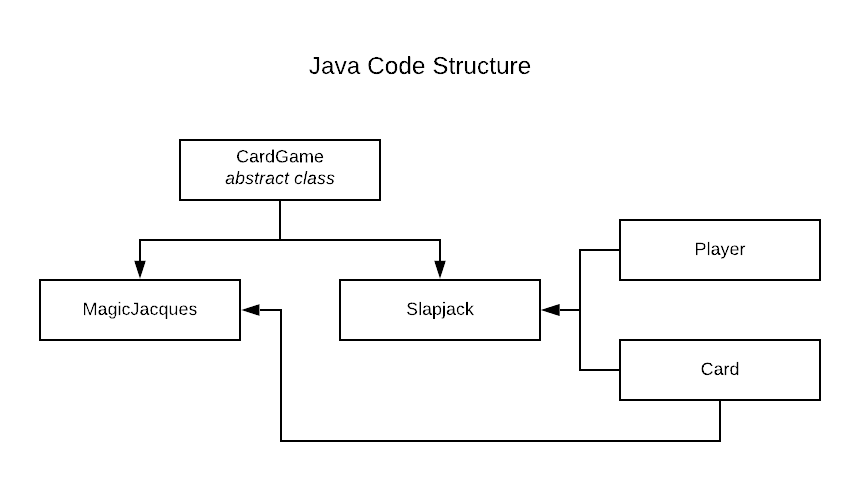
Magic Jacques is an adventure-based story game where users explore a new world they wake up into. They must make decisions as they progress, and use playing cards to stay alive. A comprehensive rule list is below:

1. Do not die

If players fail the quest, they restart from the last stage they passed. Part of the game is figuring out the available moves and looking critically for hints in the game. At any stage, users can ask for:

1. Options (all the possible moves)
2. Hints
3. Design Choices
   1. Language

Java was a simple choice, as I had the most experience with data structures and application design in Java. Java’s class structure simplified the design process, as I could factor repeated code across all games into a parent class. Finally, processing objects in games allowed for cleaner code.

* 1. Code Structure  
     
     1. CardGame: contains methods and implementation common across card games, including if I were to make new games
        1. Invoked to create deck of cards
     2. Slapjack: defines the slapjack game
     3. Player: Defines a game player with cards, name, etc.
        1. This is not used in MagicJacques because it is a single player game. There’s no need to track number of cards, name, etc.
     4. Card: Defines a game card. Has info about name, ASCII image.
     5. Magic Jacques: defines the MagicJacques game
  2. Data Structures
     1. Magic Jacques uses no data structures
     2. Slapjack uses a Stack to contain each player’s cards and an ArrayList to contain the pile of cards in the center.
  3. Algorithm Choices/Code Factoring
     1. Slapjack uses no “special” algorithms, but is cleanly factored to reduce repetition at every stage.

At each round:

* Code puts down the next player’s card, increasing the pile size
  + Pulls the top card from player’s Stack of cards and appends to common pile
* Times the user’s response (pass/slap)
  + If the user slapped within time and the card is a Jack, transfer pile to their deck
  + If the user slapped too late and the card is a Jack, CPU gets the pile
  + If the user slapped and the card is not a Jack, CPU gets the pile
  + If the user did not slap but the card is a Jack, CPU gets the pile
  + Print number of cards for each player
* Change the current player and repeat
  + 1. Magic Jacques uses a “stage” sequential code flow. Most of the code is defining the story for the game and checking user inputs against “correct” moves. The most common function is lineOut(), which prints the next line of the game, sets the hint, and sets all the possible options.

1. Edge Cases
   1. Slapjack
      1. **English**

In effort to reduce code repetition, the same functions are used to print updates. However, the English language has specific syntax that depends on the noun. For example:

* + - 1. “CPU **puts** down one card and **has** 1 **card** remaining.”
      2. “You **put** down one card and **have** 2 **cards** remaining.”

The same function is used to print both statements, so we must check the subject and verb agreement with every print statement and set the correct tenses.

* 1. Magic Jacques
     1. **English**

Similar to Slapjack, English makes the game a bit difficult. The game allows users to enter virtually anything, and we must decide whether their input has the intention of the correct move.

* 1. Runtime Environment
     1. Though we know we must run on a Linux machine, there are many variations. These include differences in screen sizes, resolutions, Unicode outputs, etc. The games use ASCII “images” and emojis to create a better experience, but these may not render the same across systems.
        1. Screen sizes: build.sh checks screen size before it runs to make sure the terminal session can fit the images used in the games.
        2. Emojis: Mac terminals support emojis, but other Linux systems may not. They render them as boxes with a Unicode representation. A design decision was made to display the emojis anyway, as I made the assumption the program would mostly run from Macs. In the case they run from other systems, there may simply be boxes. This is fine, because the emojis are not critical to the games (cards still have ASCII letters and numbers).