Johnny Tran, Trevor Conway, Tristin Harvell, Travis Kruse

We designed a program that used a single client to launch four larger programs, each their own independent game, based on whichever the user chose. When one game is launched, the client sits in the background until the game is closed or another game is selected. The games rely upon two classes: a Card class and a Deck class (based on queue). These classes are used to control the cards in the various games.

A Card object contains four data members: a value, a suit, whether the card is face up or face down, and an image that is what the player sees in game.

A Deck contains 52 Cards, with each card having its own unique identity (suit and value combination) and .png image associated with it. Choosing what to base the Deck on was a challenge that we encountered, as we weren't sure at first what data structure type best behaved how we wanted. We pondered over stacks, queues, deques, and arraylists before deciding that queue had the best combination of simplicity and function.

A difficulty chooser frame was added to add more depth and variety to our games. Although only one game implements different difficulties, the difficulty chooser frame demonstrates how a universal intermediate frame can be used to affect each game differently.

Player statistics were incorporated to express how data can be serialized and utilized over time. The program allows for player creation, which leads to various stats being tracked in the background about the current player. The players and their statistics are loaded automatically upon reentering the program if the user chooses to save.

There was difficulty with Simon Says when trying to focus on a card briefly, wait, then remove the focus from it, and repeat for a different card. The timing was challenging, especially with little experience using action listeners. To solve this issue, we decided to have a pop-up message dialogue box that expresses the order needed to solve.

Adding to the previous idea, there was also difficulty with getting Memory Match to work as we had anticipated. Instead of having an incorrect pair of cards pop up briefly and then turn over to show the back again, we simply just chose to have a both cards flipped on their back immediately if the cards didn't match. Furthermore, the correct cards previously flipped up all get turned over when an incorrect card is chosen. This adds some additional difficulty for the user considering there are very few cards to begin with anyway.

We added audio files to the program to add additional challenges and experience for each of us.