Felipe Monteiro

Design Document – Memory

**Instructions**

1. Download the zip file and extract the folder named “Memory\_Python\_Game,Monteiro”.
2. In the terminal, go inside the directory “Memory\_Python\_Game,Monteiro”
3. Run: sh run\_game.sh
4. You may run into some issues if you are using the most recent MacOS since it includes the most up to date version python3, which is not compatible with pygame.
   1. If this occurs, use an older version of python3. 3.7.0 works.

**Rules**

The game follows the standard ruleset from Memory. There are only two exceptions:

1. There is a total of 54 cards (the standard 52 plus two jokers)

Since the jokers are twice as rare, it is smarter to remember the locations of standard cards since they appear more frequently. However, if you are able to remember the location of a joker, you gain double points for the pair. This creates a risk and reward aspect for the player who chooses to remember where a joker is.

1. If you successfully match two pairs on your turn, you get three tries for the rest of your turn. For example, if you match 8-8, King-King, you can then get a point by flipping over the cards 5-6-5. In this case, the player gets a point and the pair of 5s are removed while the 6 gets flipped face down.

This feature incentivizes players to remember where pairs are without immediately turning them over. If you wait too long, the opponent might flip the pairs you were remembering and get the points. Yet, if your opponent doesn’t know, the third attempt could be very useful.

**Design Choices**

For my design choices, I wanted to focus on simplicity so I don’t need to account for many edge cases. The approach would also cut down on code and make it easily understandable.

To achieve this, python was chosen as it generally creates more condense code. I also commented the code and made it modular, splitting the code into three files –

1. a front-end file to handle setting the game up, rendering each frame, and running the game loop
2. a back-end file for the logic behind flipping and matching cards. Here is where the rules of the game, including my additions, are implemented
3. a classes file that contains a Card class and a Player class. The card class stored data like the card’s type, image, and location. The file also includes a class with various global static variables that hold data on the game’s current state.

Other design choices were made for efficiency. For example, I used a 2-dimensional array to store the cards. This allowed me to calculate the index of a card based on the position of a mouse click in O(1) time. If I used a list, I would need to loop through the entire list checking if the click was in the bounds of each card.

I also used an event system to only render a new frame when some aspect of the game has changed.

Finally, I implemented object-oriented programming to help keep track of data as each card had its own type, state and image.

**Libraries and Tooling**

Pygame was the largest library I used as it was responsible for the game’s front-end. It handled all of the rendering and retrieval of mouse input.

I chose it because it is a very rudimentary game library. It gave me just enough functionality to create the game, which kept the game file smaller and running faster. It also let me have a lot of control over the rendering and event systems so I could optimize it. For example, I chose how and when to render each frame, making sure to render only what is needed.