314 Final Project Report

Group Members: **Austin Bennett, Brett Conetta and Stephen Prospero**

Goal: **Create a three card game library that uses a GUI to improve the gameplay experience**

Motivation

When class started back in January, we were excited to begin learning Python. None of us had any Python experience at that point, but we recognized the language due to its popularity among fellow programmers and in the software industry. Completing the first homework assignment eased our transition into Python by giving us our initial experience. This experience combined with Python's favorable reputation made it a natural choice to be the language of choice for our project.

With a language picked, we decided to focus on creating GUI card games for two reasons: completing homework 4 (Hangman) sparked our interest in game programming and taking Discrete Structures II simultaneously with this class has exposed us to various card games. To improve the user experience, we decided that our card games would take place in a GUI. Our language of choice, Python, supports GUI programming with the use of tkinter. With a language decided and a concept devised, we started programming!

Overview

Our card game library consists of three favorites: Blackjack, War and Old Maid. All of these games are two player, where one of the players is the computer and the other is the user. The rules we followed for Old Maid and War were explicitly defined by Bicycle, the playing card company. The rules followed for Blackjack are enclosed within that game and can be viewed using the Rules button. Our program begins with a selection window where the user chooses which game he or she would like to play.

Old Maid

In Old Maid, three of the Queens have been removed from the standard deck of 52 cards, leaving just the Queen of Hearts (or broken hearts, depending on your view). The remaining 49 cards are split among the user and the computer. Pairs of two and four cards are removed from the deck. During each turn, either the computer or the user picks a single card from the other's hand in hopes of forming additional pairs. If a pair(s) exists, it is removed and then the turn changes. All pairs will eventually be removed, thus leaving one player with the Queen of Hearts. The winner is the player who was successfully able to rid their hand.

Blackjack

[Rules are enclosed within the game.]

314 Final Project Report

War

In War, the deck is shuffled and then distributed evenly among the computer and the user, with each deck placed face down. The deck is shuffled before it is split among the two players, and then each smaller deck is shuffled again to ensure randomness. Each player flips their top card and whoever has the larger card takes both and adds it to the bottom of the deck. In the case of both players having the same card, an additional card is placed face down, followed by a card placed face up. This new card will be the new card being used to determine who has the higher card. The process of flipping an additional card can be repeated until one player has a higher card than the other. If both players have the same card, and one of them does not have two additional cards to place down, that player automatically loses. The winner is determined by who has all the cards. Due to the nature of this game, cards are constantly being taken by both players, leading to a potentially long game, lasting many turns.

Approach

Creating the games in our library was a multi-step process. First, we devised the logic necessary for each game and translated our plan to code. At this point, our games were functional and ran on the command line (CLI). Our next step was to introduce graphics and better user interface controls to improve the experience of our games. This sub-process consisted of stages: applying graphics/images to frames to visually represent the game as it was played on the CLI and then adding widgets that users can interact with to replace the CLI. With the game logic coded and the GUI present, our focus shifted to further polishing our interface. This involved selecting appropriate fonts and colors and readjusting the location of certain frames throughout our game's progression. Lastly, we restructured the code within each game to make it successfully import to our startup.py file that serves as the opening window for our game library.

Challenges: What Didn't Work and their Respective Solutions

- Python's Garbage Collector deleted images that did not have an explicit reference. Once the GUI window appeared, locations where card images should have appeared where just blank spaces since Python automatically removed the image references while constructing the labels that held the images.
  - ***Solution***: Explicit references to the card images were declared so that Python's garbage collector would not delete them.
- Adding a graphical interface meant its state had to be aligned with the game's state. Initially, we encountered an obstacle trying to make the GUI represent the current state of the game because the loops that advanced the games continued using default values, not user entered values.
  - ***Solution***: tkinter widgets have a wait_variable(var) method that waits for a user to change the value of the specified variable, var. This method allows tkinter widgets to operate similarly to the input() method that waits for user input on the CLI.
- Our code references images that are stored within a system folder, but the path to this folder varies according to operating system. Without the correct path, our games fail to load their images.
  - ***Solution***: Accessing the sys.platform variable allows our games to recognize which platform they are running on, meaning they can actively adjust the path to the card images.
- tkinter offers several geometry managers with varying levels of customizability. In our early stages of development, many of our graphics and widgets overlapped each other and/or did not appear on screen, leading to a poor user experience.
  - ***Solution***: Using the appropriate geometry manager per frame, such as grid(), allowed us to make the necessary adjustments to our widgets and properly place card images on the screen.

314 Final Project Report

Knowledge Applied from Class

- **List comprehensions to form the deck for Old Maid and War**

  At first, creating a deck of 52 cards seemed like a grueling task. However, using a list comprehension allowed us to create the deck within a single line of code.

- **Dictionaries to form the decks within Blackjack**
  We found that using dictionaries also made the task of creating a deck painless. Specifically, different cards, no matter their suit, can be mapped to their corresponding values using a dictionary.

- **Anonymous functions in widgets**
  Several of our tkinter widgets are associated with simple actions, such as changing the value of a variable. Instead of creating separately defined functions, anonymous functions allowed us to easily state our desired action within a widgets declaration.

  Ex: `command=lambda: var2.set(1)`

- **Programming Paradigm: Procedural vs OOP**
  Our games were initially developed procedurally, meaning code placement played a significant role in our game's operation. Nonetheless, this approach was not best suited for developing a library that imports its games, so we transitioned our games to being contained within a class. Our experience shifting between different programming paradigms for our homework assignments made this process a breeze.

- **eval() function to reference image variables**
  With the issue of Python's garbage collector deleting images solved, we needed a way to dynamically reference each of the image variables, which followed the naming convention of suit then value. For example, "spades2" references the 2 of spades card. To access variables named in this manner, the string concatenation of the current card was evaluated using the eval() function, thus allowing the image variable to be referenced.

  For example, `image=eval("self."+p.suit+str(p.value)))`, where `p` is the current card