SIMPLE SOLITAIRE

This game has the same underlying principles as the classic card game Solitaire but is simplified in many ways. Instead of playing with a deck of 52 cards, the player can choose to play with any number of items. In this context, ‘items’ is referring to the ‘cards’ in the game. Generally, the more items, the longer and more complex the game is. The ultimate aim of the game is to build up one single stack of cards in descending order, from the number the player has chosen, down to 1.

To achieve the user interaction with the game, I have utilised Tkinter to create the GUI. The visual planning of the game can be found at the end of this document.

A few different frames were set up to better organise the game window visually.

- Option frame
    - This frame contains all the buttons involved in the game. This includes the help button, ranking button and the start button
    - There is also an entry box where the user can enter the number of items they would like to play with. There will be text around the entry box to indicate its purpose.
- Moving frame
    - This frame contains the entry boxes for the user to enter where they would like to move the items from and to. There will be text around the entry boxes to indicate their purpose.
- Game frame
    - This frame is where the actual game progress is displayed.
- Move count frame
    - This frame displays the number of moves the user has used and the number of moves available in the game.

Modifying the original Solitaire Class

The initial solitaire class from previous questions has been modified in order to better facilitate the needs of creative extension.

- A move counter, game_iter, has been taken out of the play function, to count the moves that the player has used. This variable is set to 0 for every new game, and increments by 1 after each move, including invalid moves. This variable is globalised in many functions.
- The original play function displays “x out of y rounds” as an indication of x moves has been used out of y moves in total. This has been moved out of the play function, and is displayed separately at the bottom of the game screen (move count frame) in the form of “x out of y moves”.
- A both_valid() function has been created. This function is called when both user inputs are valid. It clears the entry boxes upon being called and calls the play function.
- Two functions (check_c1() and check_c2()) were set up to check the validity of the user’s entry. The functions are called when the player hit the return key after an entry. If the entry is valid, the cursor will be moved to the next input box, otherwise, it will stay in the same entry box until the user enters a valid input. An invalid input will

also result in the move count to increment. Additionally, in check_c2, if the input is valid, the both_valid function will also be called.

- The play function now takes the two valid row numbers, (c1, c2) from user input as parameters. The game_iter variable is also globalised in this function. The basic structure of the function has stayed the same.
    - As long as the game_iter is smaller than the total moves allowed, the move function will be called, taking the user inputs as arguments.
        - Inside the move function, the display function is called after each move to update the state of piles
    - If the game has not finished after the move, the game continues as the user enters their input
    - If the game has finished after the game, the appropriate output will be printed depending on the outcome of the game. The cursor is then set to the number of items entry box, for the user to start a new game.
        - If the play has won the game, a window will pop up and prompt the user to enter their name to store their score. (more on this in ranking)

The ranking class

A feature that has been added to the game is to store past scores of different players (playing on the same computer) and display them. This feature is implemented through a JSON file. The JSON file is used so that it is stored locally on the computer, therefore when we exit the program and play later on, the history scores are still stored. The scores are stored in the form of a python dictionary, where the keys are the number of items the game was played with, each key's value is a nested dictionary. The nested dictionary's key is the player's name and the value is the player's score in the game.

- Every time the program is run, it first checks if the score.txt file exists already. If it does not exist, the program will create a text file to store the scores locally.
- Every time a player wins a game, a window pops up and asks the player to enter their name. Once the player has entered their name and pressed the enter button, the add_ to_ranking() function is called.
- The add_to_ranking function first checks the number of items played with against the past records. If no one has played with the specific number of items before, the score with the player's name is added to the score history. Otherwise, the function checks if the player name already exists in the dictionary for this number of items played with. If the player has played before, the function keeps the lower score. Otherwise, the function adds the score into the file. The pop-up window is destroyed in this function, and the into_json(data) function is also called.
- The into_json(data) function is where the local dictionary that was modified in add_to_ranking function is added to the JSON file.

The show ranking function

There is a ranking button in the option frame, and once it is pressed, the show ranking function called. The function opens the score.txt file and reads its content. It then reads from the number of items entry.

- If no score exists for the specific number of items, then a message is displayed to notify the player that no one has played with that many items before

- If the number is 0, then the show_all_ranking() function is called. That function follows the same concept as the show_ranking function but displays all rankings present in the history of the game on this computer, in ascending order of the number played with.
- The function shows the ranking for the specific number of items played with that has been entered in the entry box. It sorts the ranking in ascending order of move count (lower move count ranks higher).

The new game function

This function is called when the START button is pressed.

- The function sets the move count to 0 and moves the cursor to the first entry box
- It gets the number of items playing with input from the entry box then prints out appropriate warning messages if the user input is not valid.
- The function then calculates the available moves for the game and update it.
- The number of items is then shuffled in a python list. This will be the foundation pile that the user draws from.
- The piles are then displayed and the game proceeds as the player provides inputs.

The help function

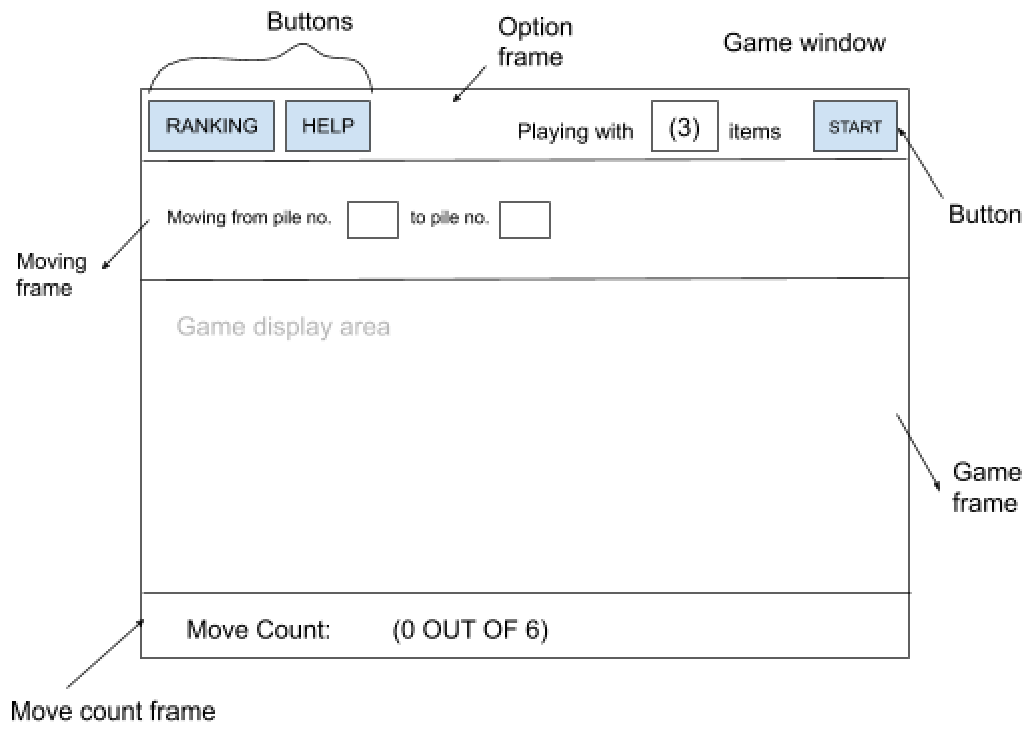
Once the help button is pressed, a message box will pop up with all the instructions necessary for the player to proceed with the game

Redirector

A redirector class was established. This class moves the standard output and standard error to the Tkinter window instead of the console/shell.

- The write function first enables the game window to be editable, then prints to it then disable the window so that the player cannot write to it.

The visual planning of the game



The visual planning of the ranking output

RANKING WITH (NUM) ITEMS

RANKING	NAME	MOVES
1	coolkid101	2
2	cs130	3
3	blah	4
…	…	...