Geoff Ribu - grib784 - 926605515

Index:

What I added to the original game:

Saving Functionality:
I added saving to the game so that if the application is closed mid-game the game state is saved (current move, amount of card piles and the states of each respective card pile). Upon reopening the application, the game prompts the user asking if they would like to continue from the previously saved game or start a new game.

Replay Functionality:
Upon starting the game, if there was no previously saved state, the game prompts the user for a deck size to use. This deck size is then turned into a list, randomised and then the game is played. When the game is over and the player either wins or loses, the game asks the user if they would like to continue playing. If the user chooses to continue playing the game is looped and the game prompts the user for a deck size to use.

Deck Randomising:
The deck size can be any size between 1 and 100 cards, once a size has been selected the number is converted into a list, randomised and passed into the Solitaire() object to create an instance of the game.

Changes to the original classes:

Solitaire:

New Variables:

- self.loaded()
    - Boolean variable to check whether or not the game will be loaded or new

- self.move_number()
    - Integer variable to hold the current move in the game

__init__()
I modified the initializer to check whether or not there is data in the save file or not. If there is no data in the save file, the game runs as usual. If there is data in the save file, the game sets a variable named self.loaded to True to tell the self.play() function how to start the game. This allows the game to be loaded from the save file.

play()
I modified the play() function to check whether or not the game is to be loaded from the save file or whether it's a new game to be started.
I put the self.save() function inside the play() loop to save the game after every move.

save()
The save function saves the data of the game in the "save.txt" file in the following format:
Line 1) Current move
Line 2) Card pile amount
Line 3) Amount of total cards
Lines 4 - 104) Cards in each pile

The save function only writes the cards for the used piles; if a used pile is empty, the line is filled in as "EMPTY".

load()
The load function reads each line of the "save.txt" file and turns it into a list. Each line is assigned to their respective variables (as formatted in the save() explanation). The card piles are then filled in using the add_bottom function from the CardPile class.

Functions:

clear_save()
The clear_save function opens the "save.txt" file and clears it, replacing the text in the file with "NO DATA".

prompt_user()
This function asks the user whether or not they want to continue playing the game. This function is used both in checking if the user wants to continue their old save or just continue the game after finishing. This function returns True if the user wants to continue and False if they don't.

ask_for_size()
This function asks the user how many cards they want to use to play the game. This input has to be an integer and between 1-100. This function returns an integer value of what the user inputs.

start_game()
This function is used to loop the game. While the game_continue variable remains True, the game loops. The start_game() function checks whether or not there is a save in the “save.txt” file.

If there is no save the game asks the user for a deck size using the ask_for_size() function. This integer is then used to create a list of numbers that size, this list is then shuffled using random.shuffle(). Then a Solitaire() object is created using the randomised deck and the play() function from the Solitaire() object is called. When the game is completed the data is cleared and the user is prompted whether or not they want to keep playing the game using the prompt_user() function.

If there is a save, the game asks the user whether or not they want to use it using the prompt_user() function. If the user decides they don’t want to continue, the data is cleared and the loop repeats, this time using the code for when there is no saved data. If the user chooses to continue, the Solitaire() object is created with an empty list and the save is loaded from where it left off and the game starts. When the game is completed the data is cleared and the user is prompted whether or not they want to keep playing the game using the prompt_user() function.

If the user chooses to end the game the game loop stops.