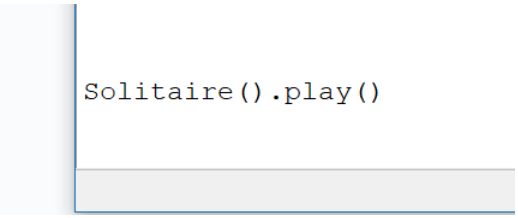
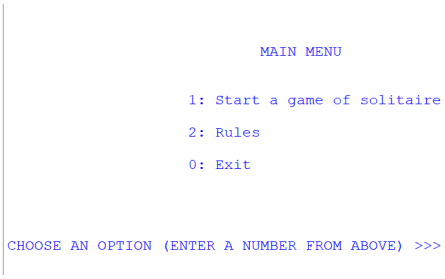
How to play:

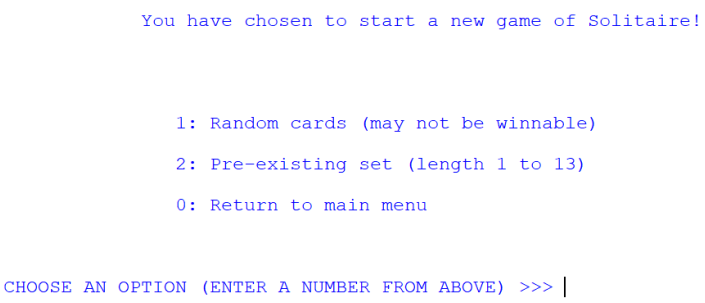
You do not need to provide a set of cards in AssCool.py when running the Solitaire() class. At the bottom of the file, after the code, simply call the Solitaire class's "play" method as such: Solitaire().play() and you will enter the main menu for the game when you run the file. You can browse throughout your experience by selecting the options you want that will be shown to you on the console.



All you need to run the game ^



<< The main menu



<< The game menu

The card hand's maximum length is 13 if you choose a pre-loaded set or any computable and realistically playable integer length if you want a longer one. if going with the random route, you could potentially play a card hand that has for example fifty thousand cards, although it would hardly be playable (mainly because the rate at which piles are presented is artificially slowed down, under the assumption that players mainly want to use the game with smaller sized hands and not with fifty thousand cards. The UI of any computer is also unsuited to such hands)

Novel Contributions Compared to more simple CLI version

Additions to playability were made visually:

- via the use of ASCII art to provide distractions from the monotonous UI of the console
- via carefully tuned insertions of the time.sleep() function to provide rhythm, movement to the UI.
- via formatting of the text via the triple quotation mark function (print(“”” ”””))

And logically:

- By letting the user select whether they wanted a predefined set of cards or a random set of cards with the length of their choosing.
- By creating a semblance of a user interface in the console by nesting different menus and having a navigational flow. This enhances replayability as the user does not have to exit the program in order to load the game with a new hand. Games with different hands can be set up without leaving the program. The user can also quit a game in progress.
- At all steps where inputs from the user are required, try and except blocks catch invalid inputs and prompt the user to enter correct values.

One main change would be that doing a ‘useless’ move (i.e moving from an empty pile to an empty pile) now does not consume a round. It instead raises a reminder that this move is not valid. The other arguably bigger change is that the player can now choose the length of their hand (code in the game_menu() method) inside the game and not as a setup in an IDE. This was achieved with the use of the random module.

Technical Explanations

The time module and the random module have been imported and used.
The CardPile class has not been modified.
The Solitaire class has received numerous modifications
All along the route, modularity and clarity were kept in mind so methods were added as pre existing ones became obviously clogged and too wide in scope.

Modified functions:

__init__(self, cards=[1,2,3,4,5,6])

This function now contains a few placeholder card hands stored in the self.inventory dictionary. This is useful if during the game_menu() set up of the game the user chooses to go with a predefined set of cards for their game.

play(self)

This function now simply is a logical centre from which other functions are called depending on the input of the user. If the user wants to read the rules, the play function will call the newly created rules() method. If the user wants to play a game, the play function will call the newly created game() function in which the game logic is stored. This play function is now the backbone of the newly made game.

Newly made functions:

main_menu(self)

An important function, this method sets the tone visually for what the rest of the game will be like. It offers a selection menu, and a way to exit. For increased playability, it contains a link to the rules of this game that are stored in the rules() function that the user can go to by selecting 2 on the main menu. It also allows the user to start and set up a new game.

game_menu(self)

While the game() method contains the logic of what happens during a game, the work of setting up the game comes in the game_menu() method. It contains the menu for the user to choose how they want to set up their game, and contains the code to build the card hand that the user wants to play with, whether it be random or predefined. This method then passes that information on to the game() method.

game(self)

This method is passed the card hand that the user selected and that was built in the previous game_menu() function. For the purpose of legibility/clarity, the Ass.py's play() code for the game has been made here into its own game() function. This allows more scope to

expand on the game logic without clogging up the play() function which as previously described now serves as a hub to route the user through to their requested sections.

reset_pile(self)

This method is important for playability. It acts on the desire to set up a new game by resetting the class's parameters stored in the __init__ method. This reset_pile() method is called after the game_menu() screen, when the user selects which hand they now want to play, thus overwriting the previous hand and setting them up with a new hand.

rules(self)

This method shows the rules of the game to the user.