**Solitaire**

We would like one class called Solitaire to be defined and that class has to have the following

• An initial deck, waste, foundations and tableaux. These four things are the main reason for having a class since at point in the time they represent the state of the game. All of these will be defined in the init method

• A play() method which calls into other methods to play the game.

• A method called getUserInput() which asks the user for a move. The user is supposed to only respond in a certain manner which is listed below.

• A method called display() which provides a nice display of the game to the user.

The one place where we want to enforce things is the user interaction piece. Since we do not have any fancy mouse drag and drop abilities yet, we have to do all of this in a text based manner.

We are going to call the 7 stacks at the bottom half of the display - tableau (dictionary with 4 keys each having a list as the value)

We are going to call the 4 stacks at the top half of the display (where you build from ace upwards) - foundations (dictionary with 7 keys each having a list as the value)

We are going to call the main deck of cards - mainDeck (list)

We are going to call the place where you place 3 cards from the deck - waste (list)

Since a picture is worth a thousand words, just use the attached powerpoint file. It shows you the conventions and the numbering as well. So here is the set of legal commands from the user. Assume all stacks, foundations etc are numbered from 1. CS thinks it is so cool doing 0 indexing but we do not want to annoy the user with our geekiness.

1. m t[i,j] t[k] - Move the i th card in the j th tableau to the k th tableau. for clarification on the numbering within the tableau see the powerpoint.

2. m t[n] f[m] - Move the card from tableau n to foundation m.

3. g - Give the user one card and place them face up on the waste.

4. m w t[n] - Move the card from the waste to tableau n.

5. m w f[m] - Move the card from the waste to foundation m.

6. r - restart the whole game.

Classes: Solitaire.py, Cards and Deck

Challenges

You have to validate the user’s inputs. This is one of the tougher things to do since this time we actually allow the user to enter any string and we want you to be robust to it. We do guarantee that the user will be entering some string though. Strongly suggest you break this check up several small components and write some smaller functions with unit tests.

The following function may help

def isInt(arg):

try: arg = int(arg)

except ValueError, e:

return False

else: return True

• Even if the user provides a valid input, they may not be able to make the move because of the rules of the game. So you need some kind of method in the solitaire class which checks that. Come up with your own version of an isLegalMove function.

• You need to display the tableaux in the right manner. For this one try and write a function that given a dictionary of lists can display them in the right manner.

def tableauPrint(self):

'''prints the rank and suit of each card in each key of the tableau dictionary in a vertical fashion'''

lengths = []

for cardList in self.tableau.values():

length = len(cardList)

lengths.append(length)

longest = max(lengths)

for i in range(0,longest):

print "\n"

for cardList in self.tableau.values():

length = len(cardList)

if length < i + 1:

print " ",

else:

card = cardList[i]

print str(" ") + card.\_\_str\_\_(),

**Class:**

Card

**Fields:**

self.rank = r

self.suit = s

self.hidden = False

**Methods:**

has\_same\_color(self, other)

set\_hidden(self, val=True)

get\_hidden(self)

set\_suit(self, value)

get\_suit(self)

set\_rank(self, value)

get\_rank(self) -> Rank can be a string or a no. but get rank will convert the string to a number and always return a no.

show\_card(self)

\_str\_(self)

**Class: Deck**

**Fields:**

self.deck= a list =[]

**Methods:**

shuffle(self)

deal(self) –pop top most card

top()-> [-1]

bottom()-> [0]

add\_top()

add\_bottom()

\_str\_()

cards\_left()-length

discard(n) -> Remove top n cards

empty()

**Class:**

Solitaire

**Fields:**

self.mainDeck= Deck()

self.mainDeck.shuffle()

self.waste=[]

self.foundation={1:[],2:[],3:[],4:[]}

self.tableau = {} -> can be less than 7

**Methods:**

**Helper Methods:**

isInt(character)

removeSpaces(string) -> remove all spaces

getLetters(string)-> removes nos., comma; returns a list

getNumbers(string)-> gets all numbers AND commas

**Actual Moves:**

moveWasteToFoundation(self,m)

moveWasteToTableau(self, n)

moveTableauToFoundation(self,n,m)

moveBetweenTableau(self,i,j,k)

**Check if moves are legal:**

checkLegalWasteFoundationMove(self, m)

checkLegalWasteTableauMove(self,n)

checkLegalTableauFoundationMove(self,n,m)

checkLegalTableauMove(self,i,j,k)

**Execute moves acc to move type , 1= t-t, 2=t-f, 3=w-t, 4=w-f**

executeMove(self,moveType,values)

**Get the correct positions or numbers from parsing the user Input string:**

getValues(self,string,moveType)

getValuesTableauMove(self,string)-> extracts the three numbers from the tableau move input that has been checked and returns them as a list, dealing appropriately with double digit numbers

getValuesWasteMove(self,string)-> deals both w-f and w-t

getValuesFoundationMove(self,string): deals with t-f

**Check if user Input strings are valid:**

checkFoundationMoveValid(self,string)

checkWasteMoveValid(self,string)

checkTableuMoveValid(self,string)

checkMoveInputValid(self,string,moveType)

determineMoveType(self, string)-> determine move types

determineFunctionType(self, userInput)->'''returns a value corresponding to the type of input the user entered in response to the move prompt - whether g, r, a move, or none of the three'''

**Check if we have won:**

checkIfWon(self)

**Display:**

display()

tableauString()

foundationString()

deckString()

wasteString()

draw\_from-deck()-> top most card from deck, unhide it and add it to the waste

flipWaste(self): -> '''takes the cards in the waste list and places them face down in the deck'''

play()

printIntroMessage()-> in Main

**Challenges:**

Parsing user inputs and making sure they are valid

Check if moves are legal

Display the tableau

Structuring the program

**Learnt:**

Python classes, association, dictionaries, lists, parsing strings

Unit tests

DocStrings

Pair programming