CS12320 Main Assignment: Patience: a card game

Gwion Hughes; gwh18@aber.ac.uk

Handed in: 19th May 2022

Percentage of overall module marks: 50%

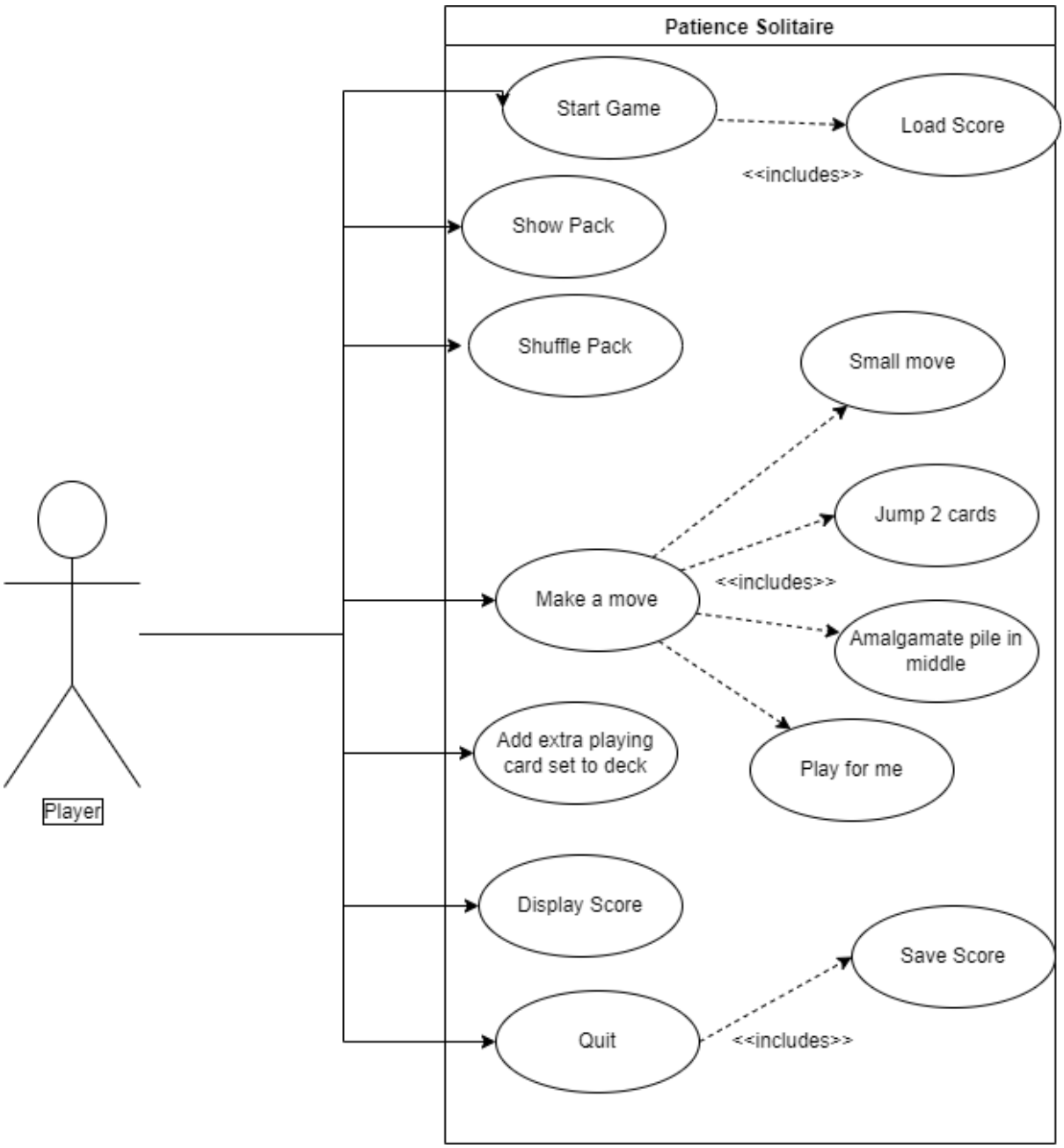
Table of Contents

Introduction

The purpose of the application is to provide light entertainment through a version of Solitaire. The game is implemented through methods allowing the player to interact through a CLI to interact with collections of cards. The application also scores the player and the top 10 of these scores are stored in a text file. Further below I have created a Use Case Diagram to illustrate the expected functionality of the program.

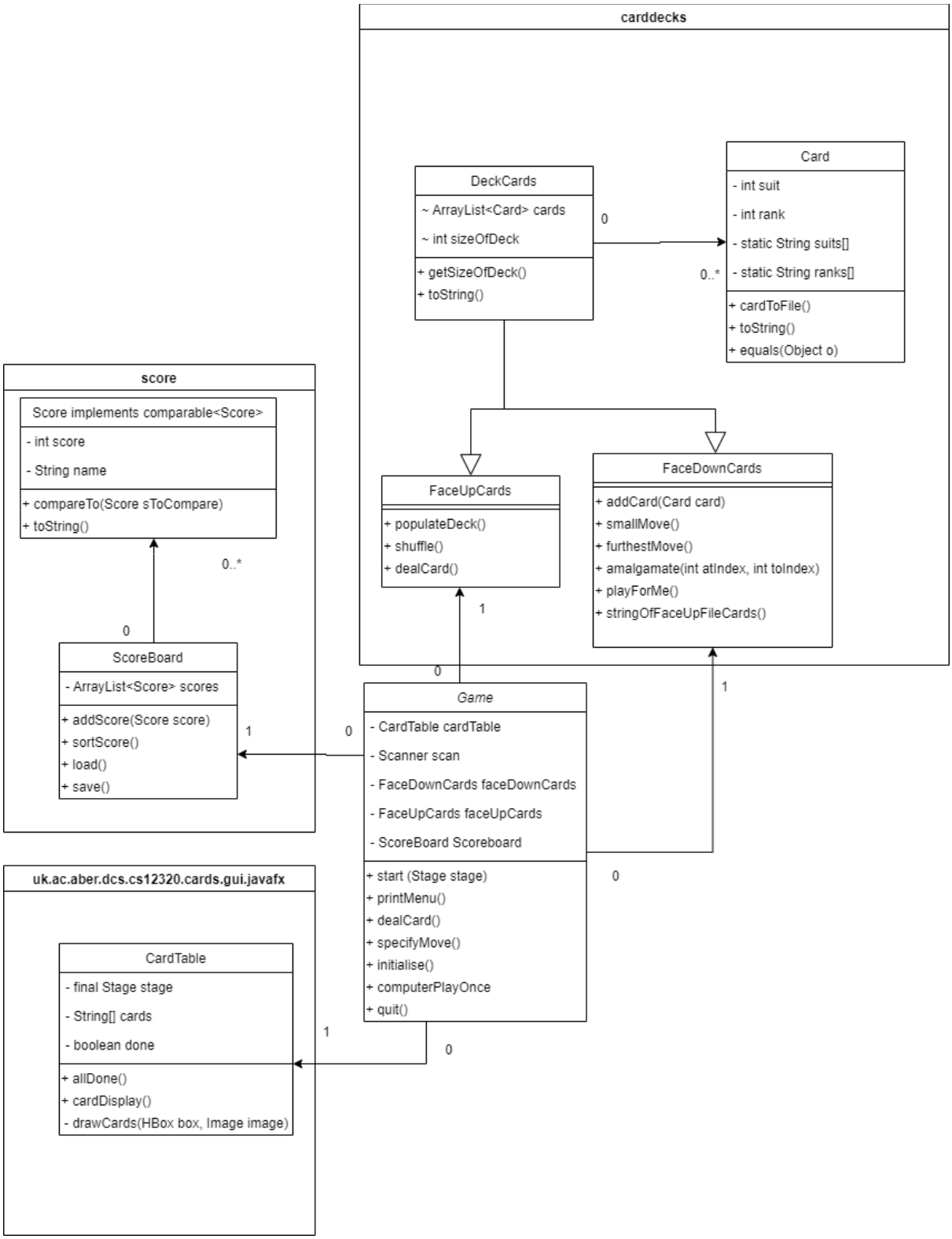
For the graphical user interface, I used the template and package provided to begin the game.

Use Case Diagram



Design

Class Diagram



Pseudo-Code

Amalgamate is a method to merge cards in the middle of the deck by giving the position of the card itself and the position the player wants the card to move to. There are only two possible kinds of moves in this game and the method checks to see whether either would be a legal move with the given input using helper methods. If the move is not legal, or the inputs are not valid, the program returns to the menu. Otherwise, it commits the move.

Function amalgamate with parameters card at index 1, card at index 2

If the remainder of subtracting index 2 from index 1 is 3 and both are an actual position of face-up cards

  If cards at index 1 and index 2 have a matching rank or suit

    Merge the card at index 1 over the card at index 2

Else if index 1 is greater by 1 than index 2 and both are actual positions of face-up cards

  If cards at index 1 and index 2 have a matching rank or suit

    Merge the card at index one over its neighbour at index 2

Else

  Do nothing

End if

End Function

Class Descriptions

Game

This class is the interface of the application and starts a thread and stage through which most user-program interaction is done. It prints to the console through the System Object, contains two decks of cards(face up and down), and collects inputs through a Scanner Object. The following is a list of its major methods:

- Main – Program start point
- Start – Starts a thread for the GUI and contains the navigation through the menu for user-program interaction.
- printMenu – Prints menu options. Called in Start.
- dealCard – Takes a card from the face-down deck and places it in the face-up deck.
- specifyMove – The user inputs the positions of two cards to merge in the middle of the deck. Inputs passed to amalgamate.
- computerPlay – The computer will make a valid move if there is one or call dealCard.
- computerPlayMore – Loops over computerPlay a player specified number of times.
- Initialize – Prepare scoreBoard. Called in start().
- Quit – Saves scoreBoard to file. Called when breaking out of Start.

CardTable

This class was provided by Aberystwyth University. It is used to modify the Javafx stage into a graphical representation of the game. It uses the names of the cards converted to string and the corresponding graphic to card type in a collection of .gif.

- cardDisplay – This method updates the Javafx stage. This has been modified to expand the window with the number of cards drawn.
- drawCard – This method is called in cardDisplay to fetch a card image.

Scoreboard

This class is used to contain and modify previous scores. Its sole attribute is the ArrayList of scores it needs to handle. When the object is outputted, it's done so in the form of a leaderboard with the lowest score placed highest out of 10. We're only concerned with dealing with the top 10 scores so no more than that 10 is loaded in and saved out of the game.

- addScore – Adds a score to the arraylist.
- sortScore – Sorts the scores in reverse order.
- load – Load in scores from Scoreboard.txt
- save – Save only the top 10 scores into Scoreboard.txt

Score

This class is a template of a score and implements comparable so that an ArrayList of this object can be sorted in reverse order by its points and that every point has a name that is proudly displayed next to its abysmal record.

- getScore
- getName
- compareTo – Used to return the lower score when sorting the arraylist in scoreboard.
- save

DeckCards

This is an abstract class which all classes that handle collections of cards inherit from. It's a generic version of a card collection not supposed to be created. It's supposed to provide some semblance of a shared standard between FaceUpCards and FaceDownCards whilst making sure that the two classes don't have methods beyond the functionality of the classes.

- getSizeOfDeck
- toString() – Prints the cards stored in the Array.

FaceDownCards

This class inherits from DeckCards and represents the cards that haven't been drawn yet. Encapsulating these functions in a separate class from the collection of cards that have been drawn protects the integrity of the collection in this class.

- populateDeck – Add the 52 playing cards to the collection. Called in the object constructor. Every time the game starts, the full deck is remade.
- Shuffle
- dealCard – This is the start of the one-way road in which cards go from the FaceDownCards cards arraylist to the FaceUpCards arraylist.

FaceUpCards

This class also inherits from DeckCards but represents the cards "in play". It is the collection of cards in this class which the player interacts with the most.

- addCard – Add a card to the card ArrayList that hopefully came from dealCard in FaceDownCards.
- smallMove – This function moves the last card in collection over the previous card if such a move is allowed.
- furthestMove – This function moves the last card in the collection onto the third previous cards if such a move is allowed.
- amalgamate
- playForMe – This function is called in the Game class' computerPlayOnce. It'll go through the card Arraylist from the start looking for a valid move to make before it returns.
- makeAMoveOne – This is a private helper method called to shift a card onto the previous one by removing the previous card.

- makeAMoveTwo – This is a private helper method called to move a card over onto the third previous card.
- isMove1Possible – This private helper method is called to check whether moving the card over the previous is a legal move.
- isMove2Possible – This private helper method is checks whether moving a card onto the third previous card is allowed.
- stringOfFaceUpFileCards – This is used to print an ArrayList of filenames for each faceup card for the cardTable class to update the GUI with.

Card

This is a template for the card class. The constructor takes two parameters as ints which it uses to reference positions on a static final list of suits and ranks. Doing it this way avoids complications with loading the cards in from a file when the file inevitably goes missing.

- cardToFile – Prints the card into a string useful for referencing a file location.
- toString
- equals – This method returns true if two cards share the same suit or rank.

Testing

Introduction

The areas of this application to be tested are as follows:

1. Navigating via command line menu
2. Print FaceDownDeck
3. Shuffle cards
4. Deal a card
5. Move the last card onto the previous card
6. Move the last card onto the third previous card
7. Amalgamate piles in middle by giving their numbers
8. Print FaceUpDeck
9. Play for me once
10. Load Scoreboard
11. Save Scoreboard

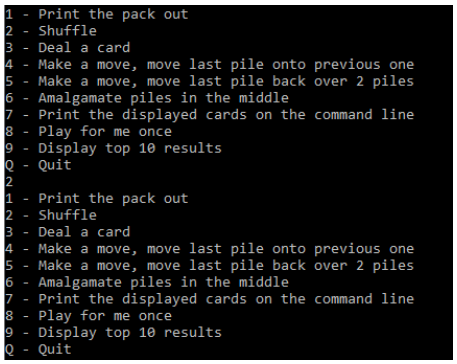
Test Table

ID	Requirement	Description	Relevant Test Data(Input)	Expected Outcome	Pass / Fail	Comments
1.1	That the menu handles a variety of inputs.	Enter a menu choice and press enter.	Menu Choice= “2” (Valid)	Screenshots 1-3 In “test 1”	P	
1.2			Menu Choice= “Q” (Valid)		P	
1.3			Menu Choice= “SIRPR1Z3!!”(Invalid)		P	
2.1	The FaceDownDeck populates cards ArrayList properly and prints contents correctly	Print card deck.	“1”	Screenshot 1 in “Test 2”	P	
3.1	faceDownDeck.shuffle() shuffles cards	Shuffle card deck.	“2”	Screenshot 1 in “Test 3”	P	
4.1	dealCard() takes a card from faceDownDeck and gives it to faceUpDeck and updates GUI.	Deal a card	“3”	Screenshots 1-2 in “Test 4”	P	
5.1	faceUpCards.smallMove() merges last card with previous card if valid move.	Small move	Attempt merge as with 3s (Valid)	Screenshots 1-2 in “Test 5”	P	
5.2			Attempts merge js with ac (Invalid)	Screenshot 3 in “Test 5”	P	
6.1	faceUpCards.furthestMove() merges last card with third previous card if valid move.	Jump 2 cards.	Attempt to merge qs with ac (Invalid)	Screenshot 1 in “Test 6”	P	
6.2			Attempt merge 7c with qc (Valid)	Screenshots 1-2 in “Test 6”	P	
7.1	Amalgamate moves card from a specified position to another specified position if allowed	Merge cards from the middle of the deck	Attempt merge 6c with 7c from the middle deck (Valid)	Screenshots 1-2 in “Test 7”	P	
7.2			Attempt 2 card skip move with 9c and 6c from the middle deck (Valid)	Screenshots 3-4 in “Test 7”	P	
7.3			Attempt to merge kc and 9h from the middle (Invalid)	Screenshot 5 in “Test 7”	P	
7.4	Catch exception from input		Attempt to merge two cards at position “a” and “not a card” (Invalid)	Screenshot 6 in “Test 7”	P	
8.1	faceUpDeck.toString prints all the face-up cards	Print cards in play	“7”	Screenshot 1 in “Test 8”	P	
9.1	computerPlaysForMeOnce doesn’t do illegal moves	Make sure the computer follows the rules	“8”	Screenshots 1-4 in “Test 9”	P	

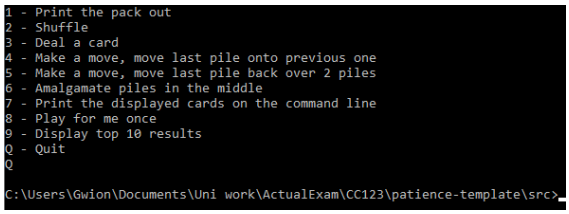
10.1	Exception FileNotFound from load() caught	Proofing the Load function	No file (Invalid)	Screenshot 1 in “Test 10”	P	
10.2	Loads sample file and sorts	Display ranked scores	An unsorted file of scores (Valid)	Screenshots 2-3 in “Test 10”	P	
11.1	Save new score to folder correctly	Top 10 scores stored and sorted correctly	Score(5, “BeepBoopSupremacy”)	Screenshots 1-2 in “Test 11”	P	

Test 1

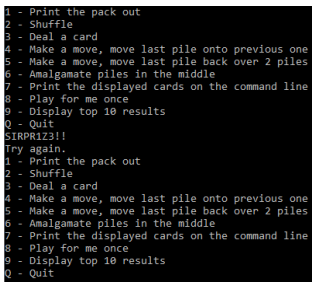
Screenshot 1



Screenshot 2



Screenshot 3



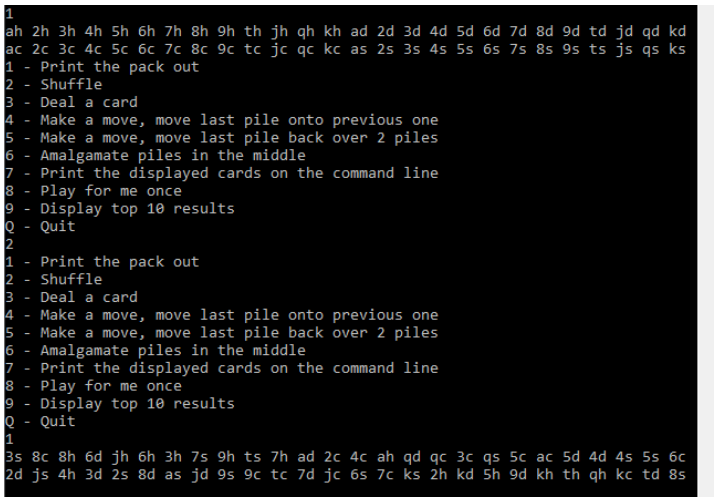
Test 2

Screenshot 1

```
1 - Print the pack out
2 - Shuffle
3 - Deal a card
4 - Make a move, move last pile onto previous one
5 - Make a move, move last pile back over 2 piles
6 - Amalgamate piles in the middle
7 - Print the displayed cards on the command line
8 - Play for me once
9 - Display top 10 results
Q - Quit
1
ah 2h 3h 4h 5h 6h 7h 8h 9h th jh qh kh ad 2d 3d 4d 5d 6d 7d 8d 9d td jd qd kd
ac 2c 3c 4c 5c 6c 7c 8c 9c tc jc qc kc as 2s 3s 4s 5s 6s 7s 8s 9s ts js qs ks
```

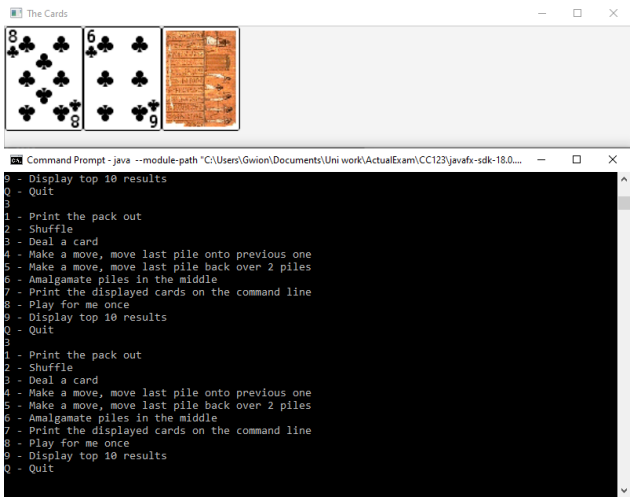
Test 3

Screenshot 1

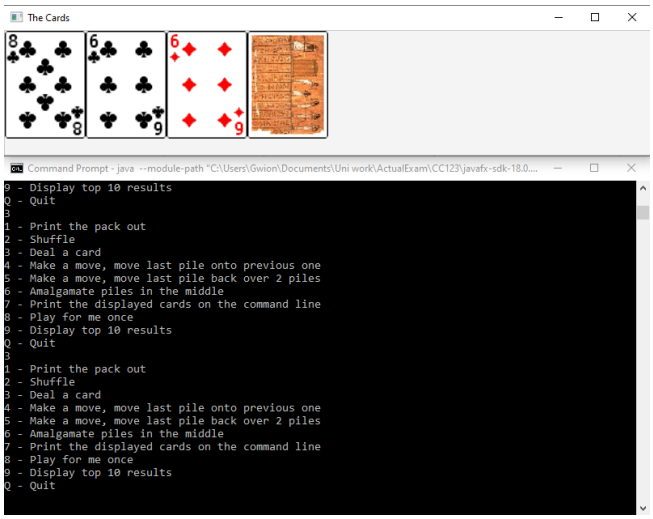


Test 4

Screenshot 1



Screenshot 2



Test 5

Screenshot 1

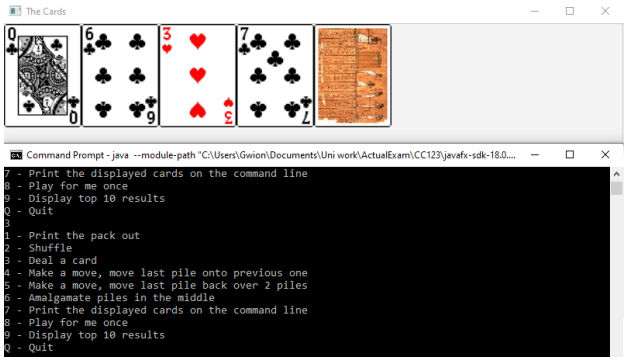
Screenshot 2

Screenshot 3

Test 6

Screenshot 1

Screenshot 2

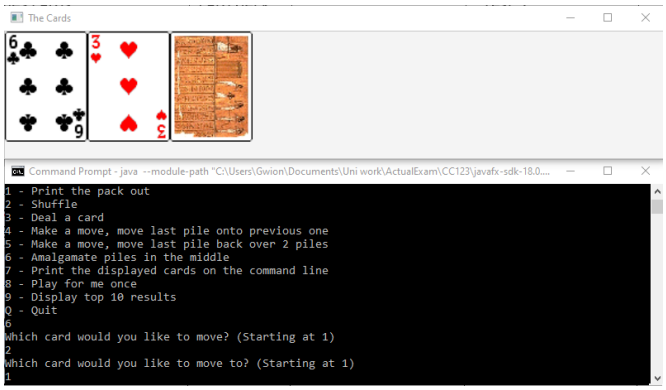


Screenshot 3

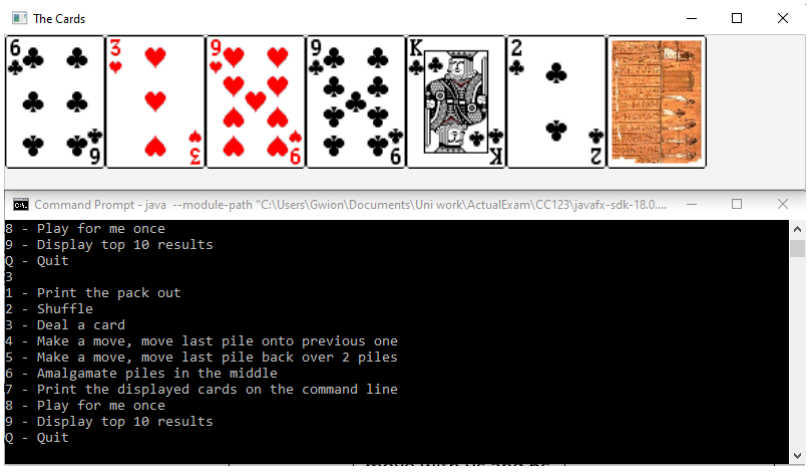
Test 7

Screenshot 1

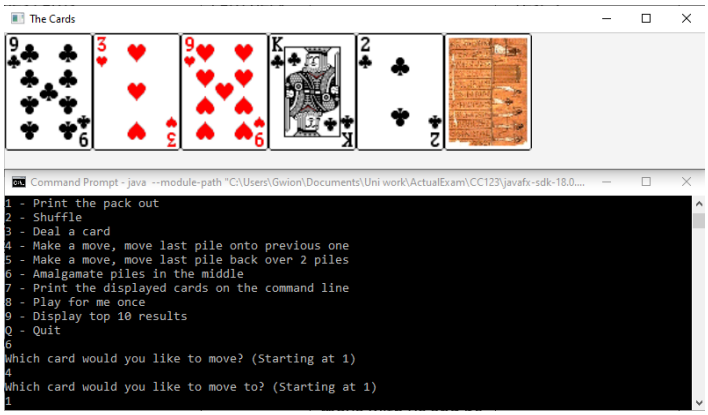
Screenshot 2



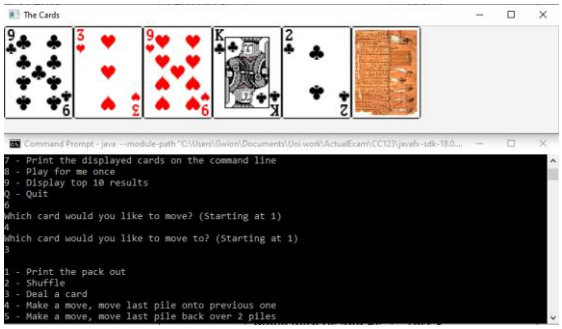
Screenshot 3



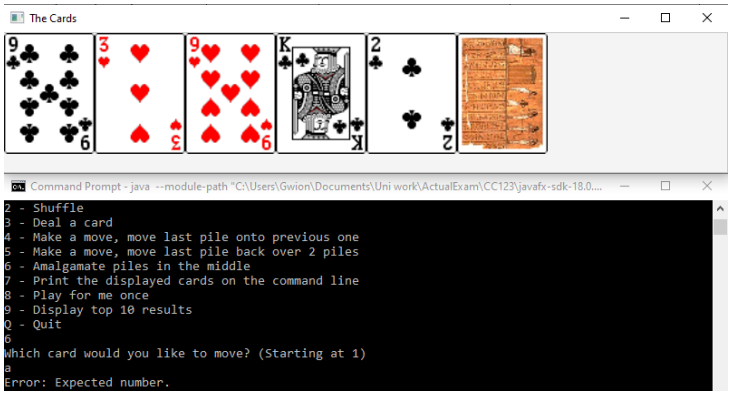
Screenshot 4



Screenshot 5

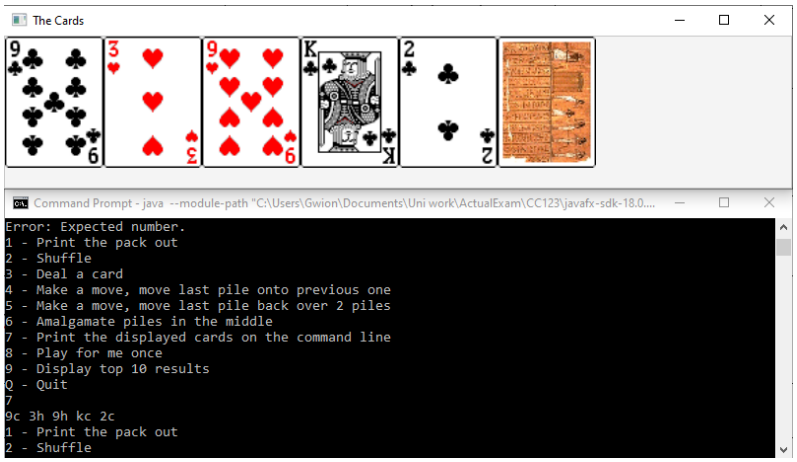


Screenshot 6



Test 8

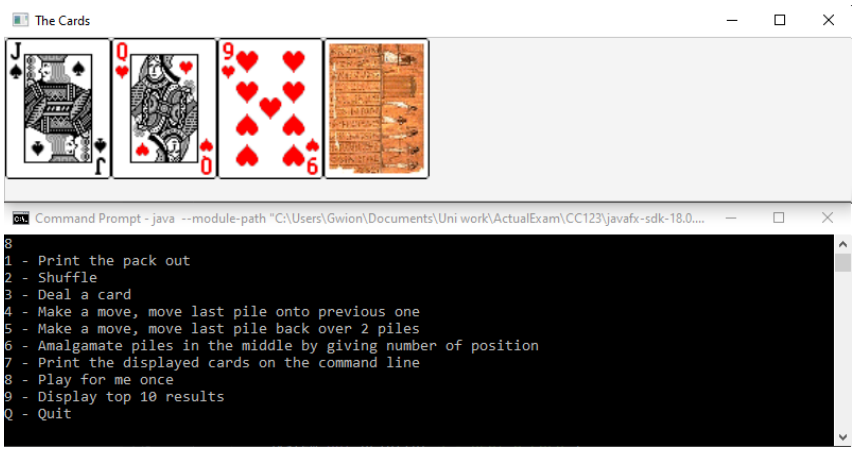
Screenshot 1



Test 9

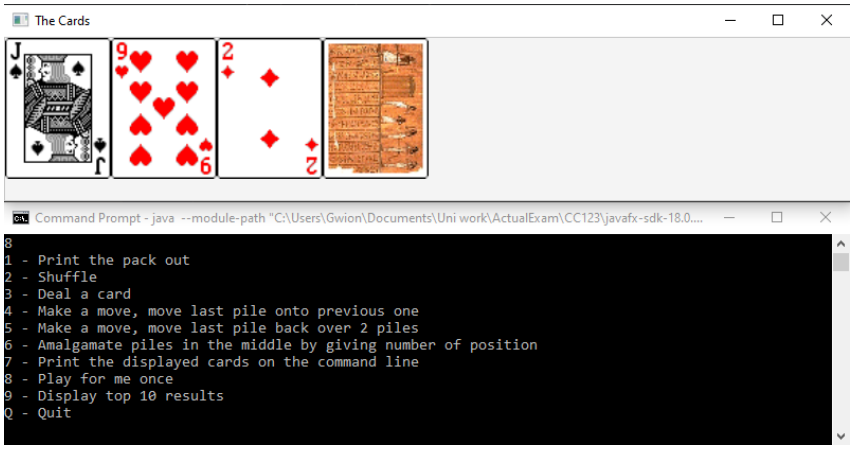
Screenshot 1

Screenshot 2



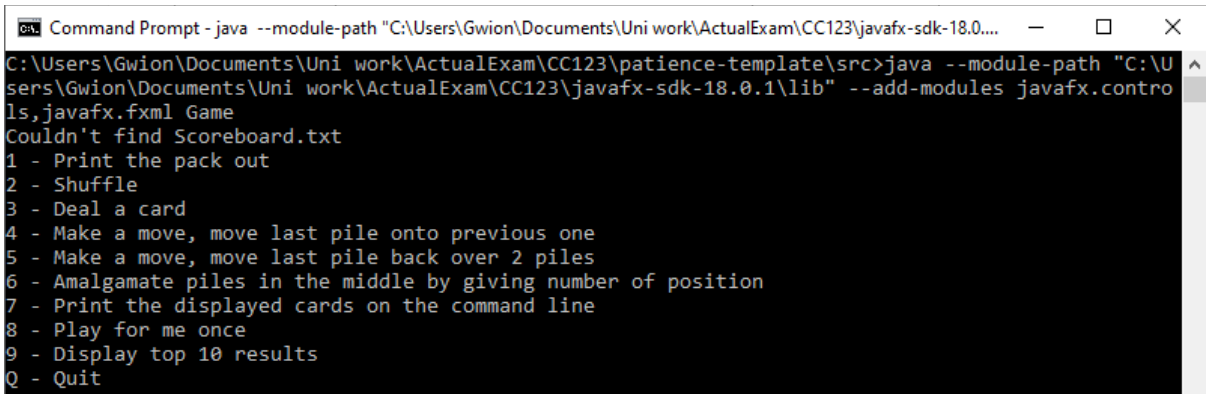
Screenshot 3

Screenshot 4

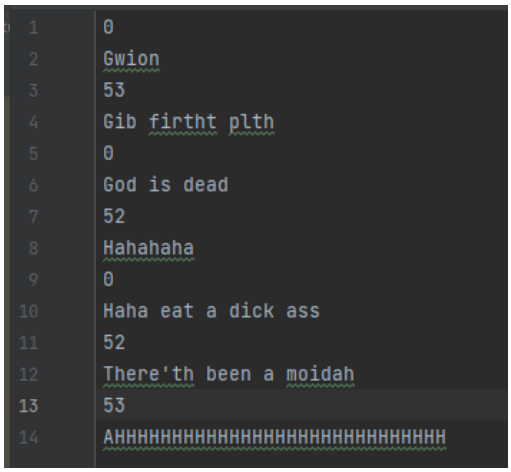

The Cards
Command Prompt - java --module-path "C:\Users\Gwion\Documents\Uni work\ActualExam\CC123\javafx-sdk-18.0....
8
1 - Print the pack out
2 - Shuffle
3 - Deal a card
4 - Make a move, move last pile onto previous one
5 - Make a move, move last pile back over 2 piles
6 - Amalgamate piles in the middle by giving number of position
7 - Print the displayed cards on the command line
8 - Play for me once
9 - Display top 10 results
Q - Quit

Test 10

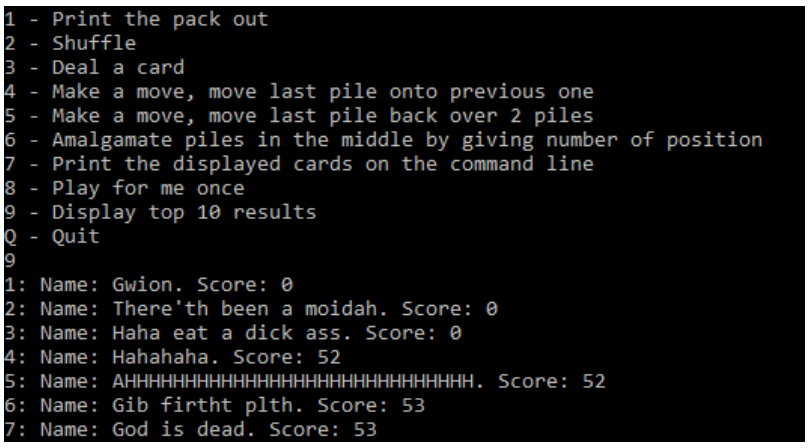
Screenshot 1



Screenshot 2



Screenshot 3



Test 11

Screenshot 1

```
1 - Print the pack out
2 - Shuffle
3 - Deal a card
4 - Make a move, move last pile onto previous one
5 - Make a move, move last pile back over 2 piles
6 - Amalgamate piles in the middle by giving number of position
7 - Print the displayed cards on the command line
8 - Play for me once
9 - Display top 10 results
Q - Quit
Q
Please input name for score.
RobotRebellion1337

C:\Users\Gwion\Documents\Uni work\ActualExam\CC123\patience-template\src>
```

Screenshot 2

```
0
Gwion
0
There'th been a moidah
0
Haha eat a dick ass
5
BeepBoopSupremacy
5
Hahahaha
8
RobotRebellion1337
52
AHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
53
Gib firtht plth
53
God is dead
```

Evaluation

I believe that I have properly and robustly implemented the core functional and non-functional requirements of the application. I have tried to validate input as best as I can, creating helper methods inside the FaceUpDeck to dismiss poor inputs before they cause OutOfBounds Exceptions. Using two static lists to index card ranks and suits instead of reading from a file mitigates the risk of the program dying before even properly starting because the card file goes missing. I've encapsulated classes in packages to make sure they only interact with other classes through classes in which they are imported and no others.

Some other pieces of flair I added to the program include...

- Option to add an extra set of playing cards to the deck.
- Specify a number of times(up to 100) that the player wants the computer to make a move.
- Increasing the width of GUI to match the number of cards drawn to a width bound only by the bit length of integer type.

These extra pieces of "flair" were added in consideration that the goal of this application is to provide light entertainment and that at some point, playing sensibly isn't going to be as fun for the player.

There are portions of my code which could be better structured. There are methods in the Game class that are, in my opinion, too tightly coupled with the sizeOfDeck attribute for my two Cards classes. Changing how this attribute is calculated would break the behaviour of computerPlay method since it is dependent on this attribute to know if a valid move existed. And I've largely left the GUI untouched lack of knowledge with Javafx.

To conclude my evaluation, whilst I believe I have fulfilled the demands of the brief, my lack of flair in alterations to the GUI alongside issues with structuring my code to be inflexible and more difficult to rewrite than it would otherwise need to be, I would reward myself at most 55% for the project.