SQA Advanced Higher

Computing Project

Joe Dobson, March 2022

Content

[Analysis 5](#_Toc98421955)

[Description of problem 5](#_Toc98421956)

[Scope, Boundaries and Constraints 5](#_Toc98421957)

[Scope 5](#_Toc98421958)

[Boundaries 6](#_Toc98421959)

[Constraints 6](#_Toc98421960)

[Requirements 7](#_Toc98421961)

[End-User Requirements 7](#_Toc98421962)

[Functional Requirements 7](#_Toc98421963)

[Persona, User Stories and User Scenarios 8](#_Toc98421964)

[Persona 8](#_Toc98421965)

[User Stories 8](#_Toc98421966)

[User Scenarios 8](#_Toc98421967)

[User Survey 9](#_Toc98421968)

[Survey Analysis 19](#_Toc98421969)

[Project plan 20](#_Toc98421970)

[UML Use Case Diagram 21](#_Toc98421971)

[Design 22](#_Toc98421972)

[Solitaire Gameplay and Terminology 22](#_Toc98421973)

[Wireframes and UI design 25](#_Toc98421974)

[Detailed UI designs 29](#_Toc98421975)

[Rounded edges 31](#_Toc98421976)

[Software Design 32](#_Toc98421977)

[Model-View-Controller 32](#_Toc98421978)

[UML class diagram 35](#_Toc98421979)

[Pseudo-code 36](#_Toc98421980)

[Insertion Sort 36](#_Toc98421981)

[Card class 37](#_Toc98421982)

[Card Deck class 38](#_Toc98421983)

[Card Stack class 41](#_Toc98421984)

[Suit Stack class 43](#_Toc98421985)

[GameState class 44](#_Toc98421986)

[Playing Area class 51](#_Toc98421987)

[Game Controller class 59](#_Toc98421988)

[Database class 63](#_Toc98421989)

[Implementation 66](#_Toc98421990)

[Programming Language 66](#_Toc98421991)

[User Interface 66](#_Toc98421992)

[Graphics Library 66](#_Toc98421993)

[Playing Card Images 67](#_Toc98421994)

[Card Size Problem 68](#_Toc98421995)

[Integration 70](#_Toc98421996)

[Structure 70](#_Toc98421997)

[Log of Ongoing Testing 73](#_Toc98421998)

[Blank Boxes 73](#_Toc98421999)

[CardStack class 74](#_Toc98422000)

[Middle of card 76](#_Toc98422001)

[Displaying deck discard 76](#_Toc98422002)

[Finding the clicked card 77](#_Toc98422003)

[Dragging in a drag 79](#_Toc98422004)

[Drawing error 81](#_Toc98422005)

[Using a threading lock 83](#_Toc98422006)

[Delayed drop 86](#_Toc98422007)

[Dropping at the bottom of a stack 87](#_Toc98422008)

[Row number not correct 90](#_Toc98422009)

[Displayed too many entries 92](#_Toc98422010)

[Testing 94](#_Toc98422011)

[Component testing 95](#_Toc98422012)

[Insertion sort testing 95](#_Toc98422013)

[Card Deck testing 101](#_Toc98422014)

[Game State testing 105](#_Toc98422015)

[Integrative testing 107](#_Toc98422016)

[Mid\_drop testing 108](#_Toc98422017)

[King\_blank testing 108](#_Toc98422018)

[Win testing 108](#_Toc98422019)

[End-user testing 109](#_Toc98422020)

[Evaluation 112](#_Toc98422021)

# Analysis

## Description of problem

I intend to create a project which will be a game that can be played by users with a variety of skill levels. The game will be designed to be played on a desktop computer. I intend to write this game in Python as I am familiar with the language. At the start of the project, I hadn’t actually decided on which game to implement but I decided that it would include the following features:

* A home screen showing a leaderboard with data from a database, that will be sorted using an insertion sort
* A game screen in which the game will be played
* A database containing the scores of all previous winners
* An on-screen timer to time how long the player takes to complete the game
* An on-screen score which will tell the player how well they are doing

The main end-users of my game will be 13-18 year-olds who would use the game for entertainment purposes and in order to test their own logic and speed. It would also be used to compare times and high scores in a competitive setting.

## Scope, Boundaries and Constraints

### Scope

The scope of my project will include:

1. A completed design using pseudocode and wireframes showing the intended user interface of the game
2. A description of the project and the AH concepts that will be included
3. A UML use case diagram showing the processes that the program will undergo, featuring actors, use cases and relationships
4. Completed test plan with descriptions of a test persona, test cases and expected outputs
5. A working game
6. An integrated leaderboard of times and scores drawn from a database which is updated will new winners on game completion
7. The results of final testing of each individual element
8. A report describing the entire process of developing my game

### Boundaries

There are several restrictions that will apply to my project.

1. The leaderboard will only contain the nicknames of the players of the game, this is because storing real names would violate GDPR
2. I will implement a version of an existing game because the research and development involved in creating a new game is outside the scope of the project
3. The leaderboard will only display the information for the top 5 players, because of the limited screen real estate
4. The project will be completed using Python, as that is the language that I am most comfortable in
5. All user inputs will need to be validated to ensure the user doesn’t give invalid inputs which might crash the game or ruin the logic of the game
6. The program is not designed for use on phones, or other mobile devices as it was built in python

### Constraints

1. The game will not have a choice of difficulty
2. Only valid moves will be available in the game
3. It won’t be possible to save a game which is part-way-through
4. It won’t be possible to undo a move that has been made

## Requirements

### End-User Requirements

* The user should be able to access a leaderboard of people, scores, moves and times
* The user should be able to enter their nickname and submit their own results of a completed game to the leaderboard and this new information should be displayed on the leaderboard if they have one of the top five all-time scores
* The user-interface should be clearly displayed on a variety of desktop computer screen sizes
* The user should be able to easily navigate around a concise user-interface
* The user should be able to quickly and easily understand how to play the game and be able to play the game without difficulty

### Functional Requirements

* The database will store data about high-scoring games that have been played (this will include nicknames, scores, moves, times)
* The program will validate all user inputs to the game
* The home screen will be easy to navigate with a link to the screen where the user can play the game
* The database will include data from all winners that is updated every time a user gets a new score
* Moves in the game will be made by a combination of mouse and keyboard inputs
* The display of the user interface will be clear and easy to understand, even for a novice player

## Persona, User Stories and User Scenarios

### Persona

Jerry is a 17-year-old student in his fifth year of high school, he plays competitive video games with his friends every day, however his laptop is old and slow so often he can’t play very high-resolution games. He enjoys the competitive aspect of video games and likes games that involve thinking and problem solving. He is very loyal to his games and doesn’t have much time for anything new. He finds it hard to play together with his friends as he often has volleyball practise, and his friends are very busy.

### User Stories

* As a student, I don’t have much time for new games, so I want an easy to understand game so that I don’t have to spend ages learning how to play
* As a competitive person, I want to be able to compete with my friends and other people in a leaderboard so I can improve in relation to others

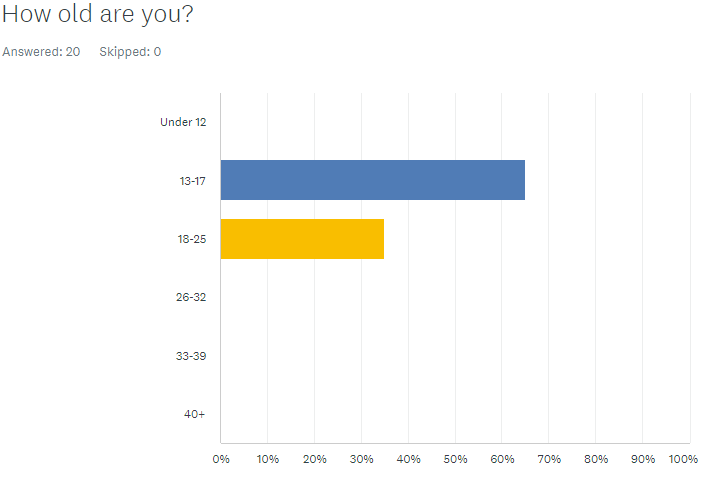
### User Scenarios

I am studying for my higher examinations this year. I don’t have much time to spend learning and downloading a new game. I need a game that is fast to understand and easy to play. I want something that can help me to relax and take a break from my studies.

### User Survey

I devised a survey intending to find out what kind of game might be most interesting to develop for my target audience, namely people around my age with an interest in games. I was particularly interested in finding out which kinds of games people wanted to play and how important aspects of those games were, so I could allocate resources towards the development of those areas.

#### Age:



Unsurprisingly, since I send it to all my friends, all my responses are in the 13-25 categories. This means that the data will be highly skewed towards this age group but that was my intention as I wanted a game that would appeal to that age group.

#### Frequency of playing:

I had a wide variety of frequencies however most people played computer games multiple times a day, this means that they could be very willing to try and play new games.

#### Reason for not playing:

Again, there was a lot of variety but most people said that they didn’t have enough time to play games, so it looks as though a game that is quick to play would be a good idea

#### Genre of game:

Even though most people would have enjoyed a shooter style game, I felt that it would be too difficult to develop something like that and that something more straight-forward would be more sensible.

#### Text Description automatically generated with medium confidenceMulti-player or Single player:

There wasn’t a very strong preference between multi player and single player games and I decided to create a single player game but create a leaderboard so that people can compete with each other.

#### Competitive:

There was an even split between competitive and casual games, so I added a leaderboard so that there was a competitive element if people wanted to compare their own scores and times to other people.

#### Features:

A vast majority of people said that the gameplay was the most important element of any game for them, so most of effort in implementation will go into making the game as enjoyable as possible, mainly by having useful features and making the game easy to understand and play.

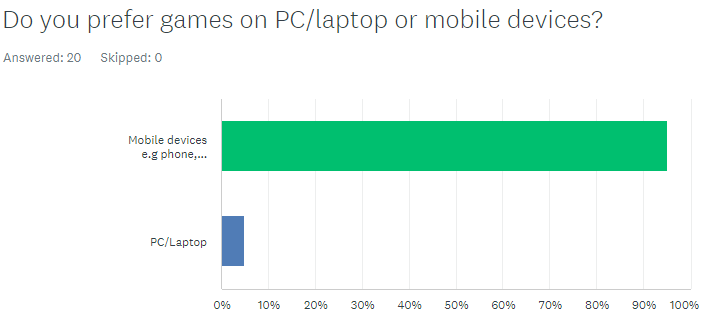
#### Difficulty:

Most respondents said they enjoyed games that were well balanced between too easy and too difficult, so the game that I create will need to be difficult enough to pose an interesting challenge (that will need to be thought through and planned out in order to win) but easy enough to play purely for entertainment.

#### User interface:

A majority of people said that they found the interface to be somewhat or extremely important, so I will make sure that the UI is easy to understand and use, while also looking nice. I will do this by getting user feedback on the wireframes and UI designs that I create.

#### PC or mobile preference:



A vast majority of people said that they would prefer a game on a mobile device instead of on a computer, however this is very difficult as I will be coding in Python for the implementation of my project, which will require being run on a computer.

### Survey Analysis

The main reason that respondents cited for not playing more games was lack of time (50%) – so I am going to create a game that is easy to understand and quick to play. I am going to re-create a pre-existing game instead of developing a brand-new game. I’m thinking of something like Tetris or Solitaire.

Most people would like to play a shooter. However, I don’t think this is a good option as creating an environment to play a shooter would be a lot more difficult than anything I have done in the past, and learning how to do it would be very time consuming.

Most people who took my survey would like a multi-player game. And 50% of the respondents said that including competition would make my game better. Connecting to other people or playing against bots would be difficult, but a leaderboard (which allows for some element of competition) could be a good inclusion.

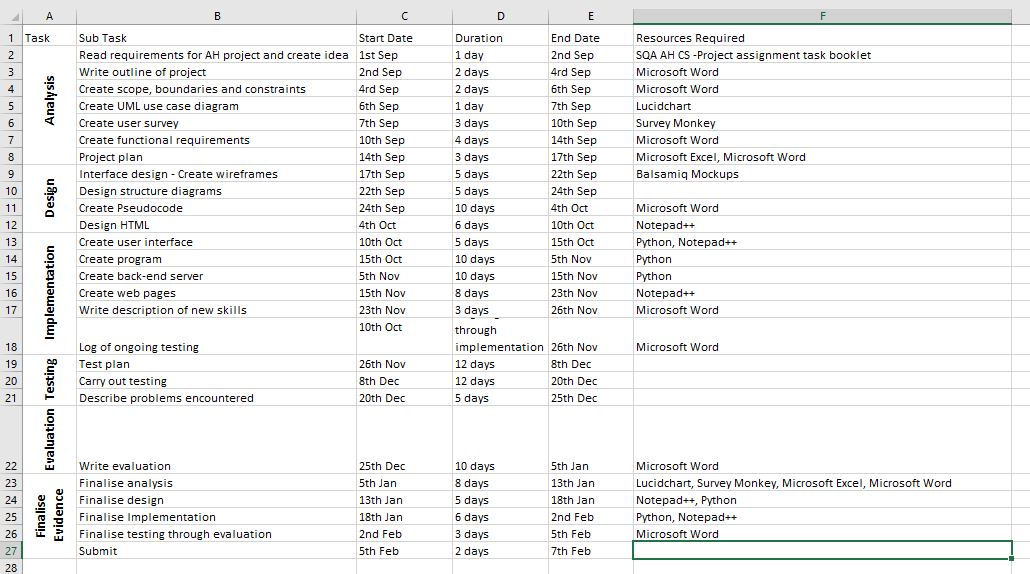
A majority of respondents (65%) who took the survey think that gameplay is the most important aspect of a game, so most of the time creating the game will be spent on making the game as smooth and enjoyable as possible.

Many people said that the interface (what the player interacts with on the screen) is important in their enjoyment of the game.

An overwhelming majority said that they would prefer a game on mobile device such as a phone, tablet or switch rather than a either a laptop or desktop computer. However, I feel that this would be too difficult to achieve so I will make the game to work on a desktop computer.

As a result of this survey feedback, I decided adapt the game Solitaire to include what the features that were asked for in this survey.

### Project plan



## Diagram Description automatically generatedUML Use Case Diagram

The user will be initially presented with a home screen which will show the current state of the leaderboard, which will have been populated from the database with a sorted list of highest scoring players. Also on this screen will be a button which allows the user to start the game.

Once in the game, the user will be able to move cards (by dragging and dropping) and will be able to turn over cards in the deck. Each time a card is moved, a check will be done to see if the game is over. In the case that the user has won, they will be asked to enter a nickname and this, along with the score will be inserted into the database.

# Design

## Solitaire Gameplay and Terminology

Throughout this project I decided to use specific words to describe different aspects of the game. These words aren’t commonly used in Solitaire so here is a terminology section to help make my code and my descriptions more understandable.

I used these terms consistently throughout my write-up and in my code, to be able to quickly and easily refer to different areas of the game.

These words are:

* Stack / Card Stack – this refers to one of the seven sets of cards in the middle of the screen
  + These cards are dealt in increasing piles (one in the first stack, two in the second stack etc.)
  + When the game starts, the bottom card of the stack is initially turned over
  + When all the face-up cards are removed from the stack, the next face-down card will turn over
  + The stacks can hold many cards and any card placed on a stack must be the opposite colour and one less in rank than the bottom card of the stack.
  + A special case is that a king can be dropped on an empty stack
  + All cards in the stack are visible and all face-up cards can be interacted with and moved
  + If a card in the middle of the stack is moved, then that card and all the cards below it will be able to be dragged around the screen and dropped on another stack
  + Multiple cards can be dropped on a stack, it is the only area of the screen where multiple cards can be dropped
  + For my game, the stacks define the win condition, as when the face-down cards in all the stacks are gone, the player has won
* Suit stack – this refers to one of the four piles in the top right of the screen
  + Initially the suit stacks are all empty, and need to be filled throughout the game
  + The only card that can be placed on a suit stack is the card that is the same suit and one more than the current card on the top of the suit stack.
  + An ace of any suit can be placed on an empty suit stack, however once a two is placed on the ace, that suit stack must be that suit and cannot be changed
  + Only the top card of the suit stack is visible at any time, and if there is nothing in the suit stack then it will display an empty box
  + Once placed on the suit stack, cards can be moved off to be put in a stack instead, or another suit stack if the card is an ace and the suit stack is empty. When the card is moved, the card below it in the suit stack will be revealed instead. E.g., if the three of clubs is moved from the suit stack on to a stack, then the two of clubs will be revealed at the top of the suit stack
* Deck – this refers to the single face-down card in the middle right of the screen
  + The only thing visible for the deck is a single face-down card, the player cannot see what is in the deck
  + When the deck is clicked on, three cards will be moved from the deck to the deck discard, or the rest of the cards in the deck if there are less than three, i.e. is there are only two cards in the deck when it is clicked, then only two cards will be put in the deck discard.
  + The deck cannot be dragged or dropped on, it can only be clicked on
  + When empty, the deck will display a single blank card instead of a face-down card, this will indicate to the user that the deck is empty and needs to be refilled
  + When the blank card is clicked on, the deck discard will empty and all the card from it will be placed into the deck instead, so that the player can continue to look through the deck and see new cards
* Deck discard – this refers to the set of three cards below the deck
  + When empty, the deck discard will display nothing and will just display the background and not any blank cards
  + The deck discard will display three cards, or the number of cards in the deck discard at the time, if that is less than three, i.e. if the deck discard has only two cards in it, only two cards will be displayed
  + Only the top card of the deck discard is able to be interacted with, however the top three cards can all be seen, this is so that the player can decide whether they want to uncover cards in the deck discard to use in the suit stacks or the stacks
  + Cards can be dragged from the deck discard, however once remove, they cannot be put back

## Wireframes and UI design

I decided that the game should have two screens:

* A home screen which will display:
  + The game titles
  + A leaderboard containing the top 5 players
  + A button that will take you to the game screen
* A game screen which contains:
  + Seven interactable stacks of cards, which are dealt from the shuffled pack at the beginning of a new game
  + Four interactable suit stacks that will be filled with cards over the course of the game
  + A pile containing the rest of the deck, which can be interacted with to turn over a set of three cards at a time
  + The deck discard, containing all the cards which have been turned over from the deck, the top card of the deck discard is able to be used
  + Counters that track the time (in seconds) from the start of the game and the number of moves the player has made
  + Once the user has won the game, a pop up will appear and ask for the user’s name, which will be inserted into the database with the time and moves.

#### Diagram Description automatically generatedHome screen wireframe:

The home screen will be displayed when the game is first started and will display the leaderboard, which will be obtained from a database stored on the user’s computer.

The home screen will also be shown after the user has won and entered their name, at which point the database will update the home screen leaderboard table with the information of the new winner, if their score is in the top 5.

#### Shape Description automatically generatedGame screen wireframe:

The game screen will be shown after the user clicks on the start game button of the home screen. In this screen the user will be able to freely move cards around the screen in order to attempt to win the game.

The timer will start counting from 0 after the user has made their first move and will continue until the game ends. The moves counter will count from 0 every time the user makes a move.

#### Diagram Description automatically generatedGame screen win pop up wireframe:

The win box will appear when the user has successfully completed the game, i.e. when they have turned over all cards in the stacks. Once the win pop up appears, the user won’t be able to interact with the cards or the game anymore and the moves counter and timer will stop running.

The user is then asked for their name, which can be inputted into the white box shown above. Once this information is inputted and the enter button is clicked, the database will be updated with their name, moves, time and score, which is calculated from the moves and time.

## Detailed UI designs

#### Graphical user interface Description automatically generatedDetailed home screen design:

I wanted to keep the design simple but containing a lot of information about the past winners and their performance. I decided to include the moves and time headings in my leaderboard so that new players knew what times and moves made for good scores.

I decided that a high score would be worse than a lower score, as the more moves and time that a user took to complete the game, the higher, and worse the score. This is similar to how scoring works in golf.

#### A picture containing graphical user interface Description automatically generatedDetailed game screen design, including example cards:

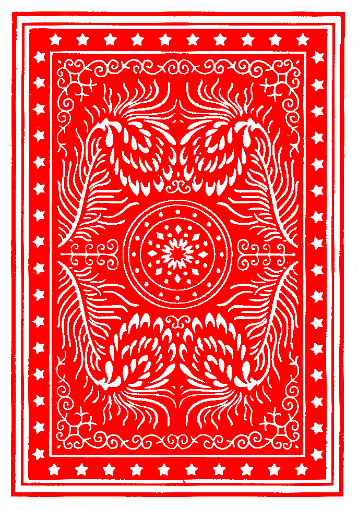
Here I have mocked up how the game might look mid-way through a game, as you can see not all the card are filled in however this give a good sense of how the game might look.

I did this because I wanted to see how the game would look and whether I needed to change the layout of the game, I was most worried about the cards in the stacks going off the bottom of the screen, however, after testing it, no matter what the user does, this can never happen.

## Rounded edges

The image for the card back that I decided to use didn’t have rounded edges – so I needed to manually edit it in paint 3d to the same size and shape as the cards with transparent background. The quality of the image was significantly worse after I had edited the image, mainly because I needed the card back to be the exact same shape and size as the cards, so it would look natural, which is why the card after the edit is more pixelated and low quality.

#### A picture containing text, furniture, rug Description automatically generatedCard before: Card After:



## Software Design

### Model-View-Controller

To implement the game software, I needed to work out how the different parts of the game would work together. I decided to use the model-view-controller design to implement my game. This is a design pattern that is built around the interconnection of three components, the model, the view and the controller.

#### Model:

The model handles the state of the game, in my game this will include where each of the cards are in the different areas of the game, which cards are in the stacks, suit stacks, deck and deck discard (see terminology section).

The model in my game:

* Moves the cards, with the information form the view and after the move has been validated by the controller
* Checks if the game is over, if the user has won
* Finds out how much time has elapsed since the start of the game, to be used in the timer

#### View:

The view handles the user interface of the game, the cards displayed on the screen. The view uses the data from the model to draw the cards on the screen in the correct place.

The view:

* Gets the user inputs, click, drag and drop,
* Finds where the user did the input, giving the data to the controller to be validated so that a move can be made by the model
* Contains the locations of everything on the screen, the stacks, suit stacks, deck and deck discard (see terminology section)
* Undraws and then redraws everything on the screen after a move is made, to update the user interface to the new game state, that is stored in the model
* Displays the move counter and timer

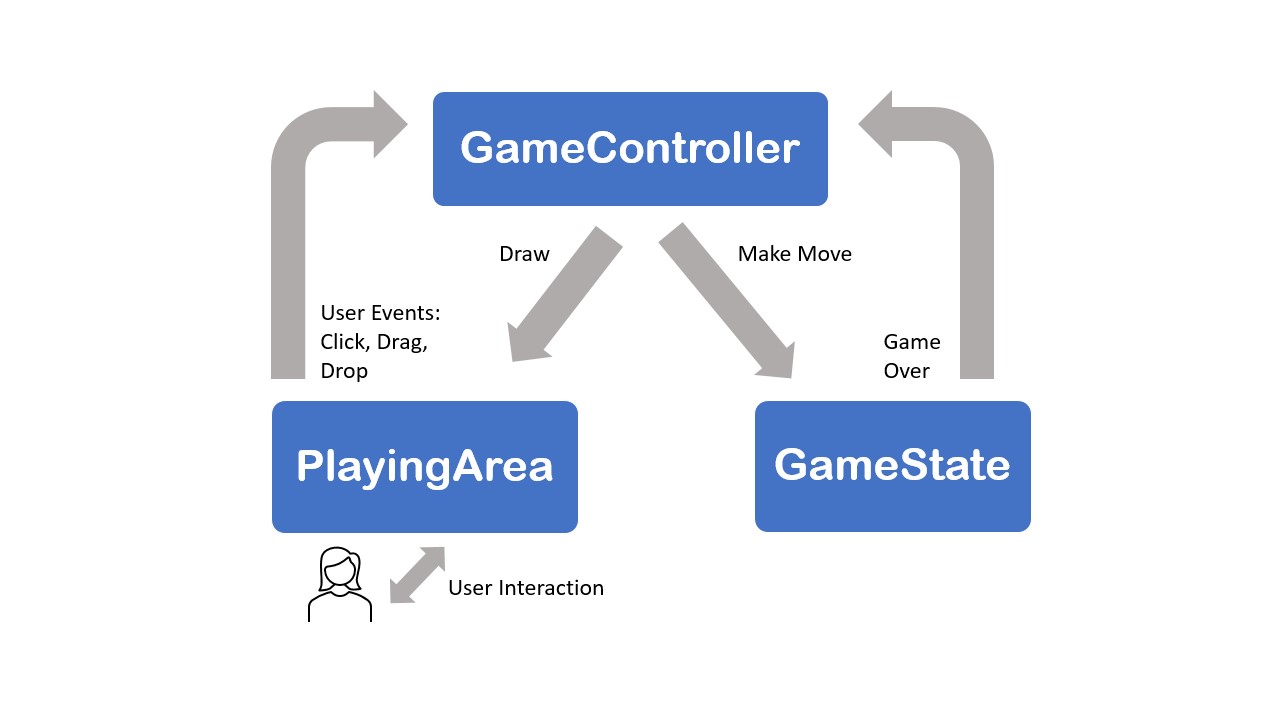
#### Controller:

The controller handles the logic of the game, in my game this will include validating the users inputs, clicks, drops and drags, and making sure that the cards are being placed in correct places.

The controller:

* Uses the information from the view to validate the user input, then tells the model to make the move
* After a move has been made, the controller tells the view to redraw the screen
* Handles the interaction between the deck and deck discard, i.e. when the user clicks on an empty deck, the deck discard is copied onto the deck and then emptied (see terminology section)

The interactions are shown in the diagram below:



I decided to use MVC with the following objects:

|  |  |
| --- | --- |
| MVC Object | Solitaire Game Class |
| Model | GameState |
| View | PlayingArea |
| Controller | GameController |

### Diagram Description automatically generatedUML class diagram

## Pseudo-code

### Insertion Sort

// Implements an insertion sort on a given list

FUNCTION insertionSort (data)

RETURNS NOTHING // Edits the passed in list as the insertion

// sort wont create a copy but edit the original

// Loops through the data list

FOR i = 1 TO data LENGTH

j = i

// Iterate backwards through the previous list elements,

// swapping with the previous element if it is smaller,

// until the start of the list

WHILE data[j] < data[j - 1] and j > 0 DO

// Swaps the current element with the previous element

temp = data[j]

data[j] = data[j – 1]

data[j - 1] = temp

j -= 1

END WHILE

NEXT

END FUNCTION

### Card class

// Card is a class that represents a single playing card

CLASS Card

// Class constructor

#CONSTRUCTOR

FUNCTION Constructor (suit, number, colour, filename)

THIS.suit = suit

THIS.number = number

THIS.colour = colour

THIS.fileName = fileName

END FUNCTION

// Get the card name

FUNCTION cardName ()

RETURNS STRING // returns the card name

RETURN THIS.suit + (THIS.number AS STRING)

END FUNCTION

// Gets the file name

FUNCTION getFileName ()

RETURNS STRING // returns the name of the image file

RETURN THIS.fileName

END FUNCTION

END CLASS

### Card Deck class

// CardDeck is a class that creates a deck of cards

CLASS CardDeck

#CONSTRUCTOR

FUNCTION Constructor ()

// Defines the cards array which will store all the cards

THIS.cards = LIST OF CLASS Card

THIS.suits = ARRAY OF STRING INITIALLY ["D", "C", "H", "S"]

THIS.colours = ARRAY OF STRING INITIALLY

["R", "B", "R", "B"]

THIS.cardNames = ARRAY OF STRING INITIALLY

["A", "2", "3", "4", "5", "6", "7",

"8", "9", "10", "J", "Q", "K"]

// Initialises the objects from the card class

FOR i = 1 TO 4

FOR j = 1 TO 13

// Format the file name of the card image

suit = THIS.suits[i]

name = THIS.cardNames[i]

colour = THIS.colours[i]

filename = suit + name + ".png"

card = NEW Card (suit, j, colour, filename)

// Appends card to the cards list

THIS.cards APPEND card

NEXT

NEXT

// Shuffle the deck

THIS.shuffleDeck ()

END FUNCTION

FUNCTION shuffleDeck ()

RETURNS NOTHING

// Shuffle the cards in-place

// This can be achieved in Python with the shuffle function

RANDOM SHUFFLE (THIS.cards)

END FUNCTION

// Returns the card

FUNCTION getCard (suit, number):

RETURNS Card OBJECT

FOR EACH card in THIS.cards

IF card.suit == suit and card.number == number THEN

RETURN card

END IF

NEXT

END FUNCTION

// Removes the top card

FUNCTION removeTop ()

RETURNS NOTHING

THIS.cards.remove (THIS.topCard ())

END FUNCTION

// Gets the top card

FUNCTION topCard ()

RETURNS Card OBJECT

RETURN THIS.cards FIRST

END FUNCTION

// Checks if the cards list is empty

FUNCTION isEmpty ()

RETURNS BOOLEAN // True if empty

RETURN THIS.cards IS EMPTY

END FUNCTION

END CLASS

### Card Stack class

// CardStack class defines a card stack ( see terminology section )

CLASS CardStack

#CONSTRUCTOR

FUNCTION Constructor (backNum)

// Sets the list of cards in the stack

THIS.cards = LIST OF Card OBJECT

// Sets the number of cards that are face down (cannot be

interacted with)

THIS.backNum = backNum

END FUNCTION

// Returns the index (in the Card Stack list) of the

specified card

FUNCTION find (card)

RETURNS INTEGER

RETURN INDEX OF card IN THIS.cards

END FUNCTION

// Removes all cards below the specified index

FUNCTION removeBelow (idx)

RETURNS NOTHING

templist = THIS.cards ABOVE INDEX idx

THIS.cards = templist

END FUNCTION

// Gets all cards below the specified index

FUNCTION getBelow (idx)

RETURNS LIST OF Card OBJECT

RETURN THIS.cards BELOW INDEX idx

END FUNCTION

FUNCTION length ()

RETURNS Integer // Length of cards list

RETURN cards LENGTH

END FUNCTION

END CLASS

### Suit Stack class

// SuitStack class defines a suit stack ( see terminology section )

CLASS SuitStack

#CONSTRUCTOR

FUNCTION Constructor ()

// Sets the list of cards in the suit stack

THIS.cards = LIST OF Card OBJECT

END FUNCTION

// Checks if the cards list is empty

FUNCTION isEmpty ()

RETURNS BOOLEAN // True if empty

RETURN True IF THIS.cards IS EMPTY

END FUNCTION

FUNCTION length ()

RETURNS Integer // Length of cards list

RETURN cards LENGTH

END FUNCTION

END CLASS

### GameState class

// GameState class defines the positioning and layout of the cards

// at all stages of the game

CLASS GameState

#CONSTRUCTOR

FUNCTION Constructor ()

// Creates an instance of the CardDeck class

THIS.cardDeck = NEW CardDeck ()

// Sets the time and the moves at the start

THIS.startTime = 0

THIS.moves = 0

// Sets the number of stacks and suit stacks

THIS.numCardStacks = 7

THIS.numSuitStacks = 4

// Creates the cardStacks as a ist of objects

THIS.cardStacks = []

FOR I = 0 TO THIS.numCardStacks

// Instantiates a new card stack object with an

increasing backnum

cardStack = NEW CardStack (i)

FOR j = 0 TO i + 1

// Gets the top card of the deck

topCard = THIS.cardDeck.topCard ()

// Adds it to the card stack

cardStack.cards APPEND topCard

// Removes it from the deck

THIS.cardDeck.removeTop ()

NEXT

// Adds the newly created card stack to the list of card

stacks

THIS.cardStacks APPEND cardStack

// Creates the suit stacks

THIS.suitStacks = []

FOR i = 0 TO THIS.numSuitStacks

// Instantiates a new suitStack

suitStack = NEW SuitStack ()

THIS.suitStacks.cards APPEND suitStack

// Creates the interactive discard of the deck

THIS.deckDiscard = []

END FUNCTION

// Makes the move

// e contains the cards which are to be moved,

// there may be more than one as multiple cards can be moved

// from the card stacks

// It also contains the name of the area that we are dropping

// the card(s) on e.g. SuitStacks

// and the index of the specific stack/suitStack that

// is being dropped on

FUNCTION makeMove (e)

RETURNS NOTHING // Edits the gameState

// Starts the game timer if it is the first move

IF THIS.startTime IS 0 THEN

THIS.startTime = CURRENT TIME

END IF

// Adds one to the moves

INCREASE THIS.moves BY 1

// See if changing deckDiscard

IF FIRST OF e.clickedCards IN THIS.deckDiscard THEN

// Gives us the top card of the deck discard

sourceCards = LAST OF THIS.deckDiscard

// Removes the card selected from the deck discard

REMOVE sourceCards FROM THIS.deckDiscard

ENDIF

// See if changing suitStacks

FOR suitStack IN THIS.suitStacks

// If the clicked card is in the suit stack

IF FIRST OF e.clickedCards IN suitStack.cards THEN

// Gives the top card of the selected suit stack

sourceCards = LAST OF suitStack

// Removes the card from the suit stack

REMOVE sourceCards FROM suitStack

END IF

NEXT

// See if changing cardStacks

FOR stack IN THIS.cardStacks

// For the stacks we need the actual location of the

// selected card in the stack because you can select a

// card that is midway up the stack

// Sets the index for the selected card

cardStackIdx = FIND FIRST OF e.clickedCards IN stack

// If the card actually was in the stack

IF cardStackIdx IS FOUND THEN

// Get all cards below the index of the clicked card

// So it can move multiple cards at the same time

sourceCards = stack.getBelow (cardStackIdx)

// Turns over the next card if the index is the same

as the number of back up cards

// i.e. if the clicked on card is the first after

the set of back up cards

IF cardStackIdx IS stack.backNum THEN

// Decreases the backnum by one, which will draw

the next card when the screen redraws

DECREASE stack.backNum BY 1

END IF

// Removes the cards from the stack

stack.removeBelow (cardStackIdx)

END IF

NEXT

// Find drop location

IF e.dropName IS "SuitStacks" THEN

// Adds the card to the top of the suit stack

THIS.suitStacks (e.dropIdx) APPEND FIRST OF sourceCards

ELSE IF e.dropName IS "Stacks"

// Loops through as you can drop multiple cards on the

stacks

FOR card IN sourceCards:

// Adds those cards to the stack

THIS.cardStacks[e.dropIdx] APPEND card

NEXT

END IF

END FUNCTION

// Checks if the game is over

FUNCTION isGameOver ()

RETURNS BOOLEAN // True if the player has won

// Sets the condition to be where all cards in the card

stacks are turned over

FOR stack IN THIS.cardStacks

// If the backnum of any stack is not 0

IF stack.backNum IS NOT 0 THEN

// The player hasnt won

RETURN False

END IF

NEXT

// Sets the total time, which will be inserted into the

database

THIS.totalTime = TOTAL TIME ELAPSED

// If the function doesnt return before this point, the

player has won

RETURN True

END FUNCTION

END CLASS

### Playing Area class

CLASS PlayingArea

#CONSTRUCTOR

FUNCTION Constructor (gameState, window)

// Creates the timer

THIS.gameTimer = NEW Timer ()

// Sets the window

THIS.window = window

END FUNCTION

// Draws the screen

FUNCTION draw ()

RETURNS NOTHING // Edits the UI

// Wipes the screen

THIS.window.clear ()

// Changes the background colour to green

THIS.window.fill ("Green")

// Starts the timer

THIS.gameTimer.start ()

// Calls the display functions for everything in the game

THIS.displayEverything ()

END FUNCTION

FUNCTION displayEverything ()

THIS.displayStacks()

THIS.displaySuitStacks()

THIS.displayDeckDiscard()

THIS.displayDeck()

END FUNCTION

// Displays the stacks

FUNCTION displayStacks ()

RETURNS NOTHING // Edits the UI

// Loop through the card stacks

FOR i = 0 TO gameState.cardStacks LENGTH

// Loop through the cards in the stacks

FOR j = 0 TO gameState.cardStacks[i] LENGTH

// Defines the location of the current card

x = leftMargin + i \* stackWidth

y = topMargin + j \* cardYDist

// If the current position is less than the number

of backwards facing cards

IF gameState.cardStacks[i].backNum > j THEN

// Instead of displaying the image, it displays

the card back instead

THIS.window.draw ("cardBack.png", x, y)

ELSE

// Displays a face-up card

filename =

gameState.cardStacks[i].cards[j].getFileName()

THIS.window.draw (filename, x, y)

END IF

NEXT

NEXT

END FUNCTION

// Displays the suit stacks

FUNCTION displaySuitStacks ()

RETURNS NOTHING // Edits the UI

FOR i = 0 TO 4

x = suitStackLeft + (i MOD 2) \* cardXDist

y = topMargin + j / 2 \* cardYDist

// If the suit stack is empty then display a blank card

IF THIS.gameState.suitStacks[i].isEmpty () THEN

THIS.window.draw ("cardBack.png", x, y)

ELSE

// Display the card at the top of the suit stack

card = END OF gameState.suitStacks

THIS.window.draw (card.getFileName (), x, y)

END IF

NEXT

END FUNCTION

// Displays the deck discard

FUNCTION displayDeckDiscard ()

RETURNS NOTHING // Edits the UI

// Finds the minimum of 3 and the length of the deck discard

num = MIN OF 3, gameState.deckDiscard LENGTH

FOR i = 0 TO num

x = deckDiscardLeft + i \* cardXDist

y = deckDiscardTop

filename = gameState.deckDiscard[i].getFileName ()

THIS.window.draw (filename, x, y)

NEXT

END FUNCTION

// Displays the deck

FUNCTION displayDeck ()

RETURNS NOTHING // Edits the UI

// Checks if the deck is empty

x = deckLeft

y = deckTop

IF gameState.cardDeck.isEmpty () THEN

THIS.window.draw ("blankCard.png", x, y)

ELSE

THIS.window.draw ("cardBack.png", x, y)

END IF

END FUNCTION

// Called when the player drags a card

FUNCTION drag (clickImgs, x, y)

RETURNS NOTHING

// Gets the image and the index for each element that we are

dragging - which can be multiple cards

FOR i = 0 TO clickImgs LENGTH

// Undraw the card

clickImgs[i].undraw ()

// Redraws the image at the new location

THIS.window.draw (clickImgs[i], x, y)

NEXT

END FUNCTION

// Called when the player drops a card

FUNCTION drop (x, y)

RETURNS NOTHING

IF NOT gameState.isGameOver () THEN

// Find the drop location

dropLocation = THIS.findEventLocation (x, y)

// Validate the drop

gameController.validateMove (dropLocation)

END IF

END FUNCTION

// Called when the player clicks on a card

FUNCTION click (x, y)

RETURNS NOTHING

IF NOT gameState.isGameOver () THEN

// Find the click location

clickLocation = THIS.findEventLocation (x, y)

IF THIS.clickLocation.area IS "Deck" THEN

THIS.turnCards ()

END IF

END IF

END FUNCTION

// This is used to set variables for both a drop and a click

FUNCTION findEventLocation (x, y)

RETURNS EventLocation // location of event

// If we clicked on the deck

IF THIS.isClicked (x, y, DECK) THEN

// The deck cannot be dropped on

RETURN NEW EventLocation ("Deck")

ELSE IF THIS.isClicked (x, y, DECKDISCARD) THEN

RETURN NEW EventLocation ("Discard", END OF

gameState.deckDiscard)

ELSE IF THIS.isClicked (x, y, SUITSTACKS) THEN

i = THIS.clickedStack (x, y)

RETURN NEW EventLocation ("SuitStacks", END OF

gameState.suitStacks[i], i)

ELSE IF THIS.isClicked (x, y, STACKS) THEN

i = THIS.clickedStack (x, y)

j = THIS.clickedStackCard (x, y, i)

RETURN NEW EventLocation ("Stack", END OF

gameState.stacks[i], i, j)

END FUNCTION

FUNCTION isClicked (x, y, area)

RETURNS BOOLEAN // True if clicked

// This is a point in rectangle function

IF x, y IN area THEN

RETURN True

ELSE

RETURN False

END IF

END FUNCTION

FUNCTION clickedStack (x, y)

RETURNS INTEGER // No. of clicked stack

FOR i = 0 TO 4

IF x, y IN gameState.suitStacks[i] THEN

RETURN i

END IF

NEXT

FOR i = 0 TO 7

IF x, y IN gameState.stacks[i] THEN

RETURN i

END IF

NEXT

RETURN 0

END FUNCTION

FUNCTION clickedStackCard (x, y, i)

RETURNS INTEGER // Location of selected card in the stack

FOR j = 0 TO gameState.stacks[i] LENGTH

IF x, y IN gameState.stacks[i][j] THEN

RETURN j

END IF

NEXT

RETURN 0

END FUNCTION

// Displays the score

FUNCTION showScore ()

RETURNS NOTHING

THIS.window.text (THIS.gameTimer, THIS.gameState.moves)

END FUNCTION

END CLASS

### Game Controller class

CLASS GameController

#CONSTRUCTOR

FUNCTION Constructor (playingArea, gameState)

END FUNCTION

FUNCTION validateMove (eventLocation)

RETURNS NOTHING

// Checks if the drop is valid

dropValid = THIS.isValid (eventLocation)

IF dropValid THEN

// Makes the move

gameState.makeMove(eventLocation)

END IF

// Redraws the screen

playingArea.draw ()

END FUNCTION

// Passes in an eventLocation object as e

FUNCTION isValid (e)

RETURNS BOOLEAN // True if the event is valid

// If we are dropping on a suitstack and the length of the

cards we picked up is exactly 1

IF e.name IS "SuitStacks" and e.clickedCards LENGTH

IS 1 THEN

// If dropping on an empty suitstack and the picked up

card is an ace then return True else return False

IF e.onCard LENGTH IS 0 THEN

RETURN e.clickedCards[0].number IS 1

END IF

// If we have one card picked up, the suit is the same

// and the number is one more then return True else

return False

END IF

// If you are dropping on the same suit and the number

// is one more than the top card then the drop is valid

IF e.clickedCards[0].suit IS e.onCard[0].suit AND

e.clickedCards[0].number IS e.onCard[0].number + 1

// If we are dropping on a stack and we are dropping at the

end of a stack

IF e.name IS "Stacks" AND e.stackLocation IS 0 THEN

// If the stack is empty

IF e.onCard LENGTH IS 0 THEN

// Return true if the top of the clicked cards is a

king

RETURN e.clickedCards[0].number IS 13

END IF

// Else return true if the colour is not the same and

the number is one less

RETURN e.clickedCards[0].colour IS NOT

e.onCard[0].colour AND e.clickedCards[0].number

IS e.onCard[0].number - 1

END IF

// If we don’t drop in any of these locations, return false

RETURN False

END FUNCTION

// Puts the deck discard back into the deck

FUNCTION turnCards ()

RETURNS NOTHING

// If the card deck is empty

IF gameState.cardDeck.isEmpty () THEN

// Sets the card deck to the deck discard

gameState.cardDeck.cards = gameState.deckDiscard

// Empties the deck discard

SET gameState.deckDiscard TO EMPTY LIST

ELSE

// Gets the length of the deck if it is less than 3,

otherwise gets 3

numToMove = MIN 3, gameState.cardDeck.cards LENGTH

// Adds those cards to the deck discard

cardsToMove = LAST numToMove OF gameState.cardDeck

gameState.deckDiscard.cards APPEND cardsToMove

// Removes those cards from the deck

REMOVE cardsToMove FROM gameState.cardDeck

END IF

END FUNCTION

END CLASS

### Database class

CLASS GameDb

#CONSTRUCTOR

FUNCTION Constructor ()

// Creates the connection

THIS.conn = CONNECT TO DATABASE "Solitaire.db"

// Query to create table

tableQuery = CREATE TABLE SCORES WITH FIELDS

(Name, Moves, Time, Score)

// Executes the create table query

THIS.conn.execute(tableQuery)

// Inserts a new user into the database

FUNCTION addWinner(winnerName, moves, time):

RETURNS NOTHING // Edits the database

score = createScore (moves,time)

// Runs an insert query, creating the score

insertNewWinner = INSERT INTO SCORES VALUES

(winnerName, moves, time, score)

THIS.conn.execute(insertNewWinner)

END FUNCTION

// Gets the data, which is displayed on the home screen

FUNCTION getData ()

RETURNS NOTHING

// Simply selects all the data, which is sorted by score

showData = SELECT \* FROM SCORES

// Sorting data with insertion sort

data = []

data = THIS.conn.execute(showData)

// Uses an insertion sort to sort the data by score

insertionSort(data)

END FUNCTION

// Deletes the table

FUNCTION deleteTable ()

RETURNS NOTHING

deleteTable = DROP TABLE SCORES

THIS.conn.execute(deleteTable)

END FUNCTION

// Generates the score

FUNCTION createScore (moves, time)

RETURNS INTEGER // The calculated score

RETURN moves \* 4 + time

END FUNCTION

END CLASS

Integration Design

Abstraction

Comparison of Design to Requirements

# Implementation

## Programming Language

The first problem I faced when thinking about designing and implementing my code was which language to write in, I decided on Python as I was already comfortable with the language and had been using it for some years prior to this project. I had also had some experience making games and doing projects of a similar nature in Python previously, however this was the first time I was using actual graphics to display an interactive User Interface, previously I had made projects that displayed with simple text-based outputs.

This came with an issue however, as Python doesn’t have an inbuilt graphics library and displaying Solitaire as a text-based game wasn’t an option. I decided to use an extension of tkinter, a popular Python graphics library, that I downloaded from the internet. This is a single file module called graphics.py which provides basic handling for images, text and basic shapes on a canvas.

## User Interface

### Graphics Library

I used the graphics.py library to display a window containing cards that could be interacted with, and to check when the user clicks, drops, or drags - getting the coordinates of these inputs. The user’s input is then interpreted by the GameController and changes are then made to the GameState.

One advantage of a library like graphics.py is that each image, text or shape is an object and can be moved around on the screen by simply changing its coordinates. This is in contrast to a bitmapped library which merges information with the background when drawing of shapes and images and doesn’t permit them to be moved without erasing and redrawing.

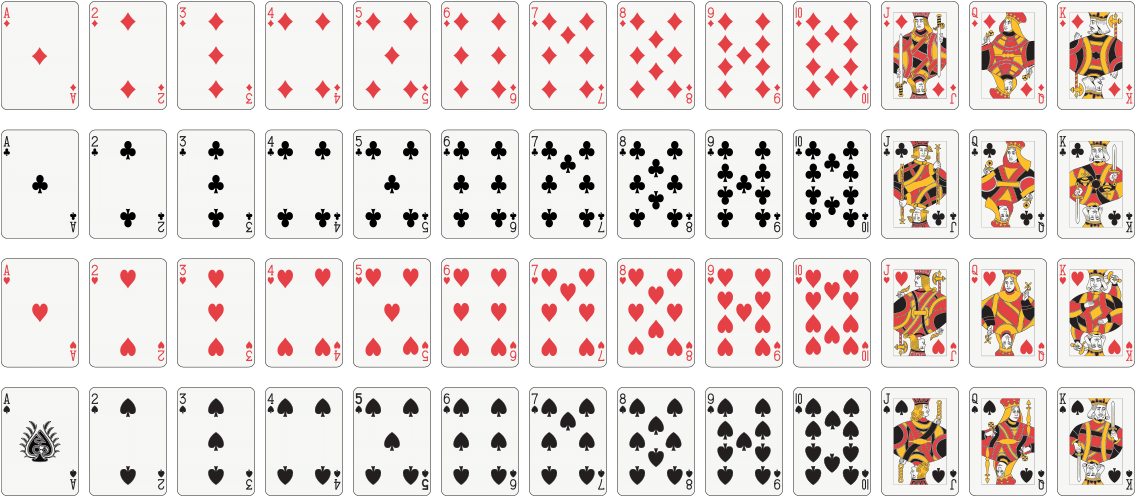
This would have been a problem for my program, for example in the card stacks, when the top card of the card stack is moved, by the user dragging the card to a different place on the screen, the card directly below it would be visually cut in half. This would be a big issue as it could lead to the User Interface being misinterpreted.

Using graphics.py avoids this problem as the images of the cards were treated like individual entities and the canvas has a log of everything that is drawn and undrawn on it, meaning that everything on the window is stored as an individual image object which can be manipulated and moved.

For a game like Solitaire this makes a big difference as dragging a card around is smooth and doesn’t involve redrawing the whole screen on every move.

### Playing Card Images

One of the things required for my game is a set of images of playing cards. I searched on the internet and found a single image which shows all of the cards in a deck laid out in rows and columns by suit. I decided to use this to generate my playing card images.



The program below was written to perform the operation of cutting the large card image into smaller pieces, each representing a single playing card.

### Text Description automatically generatedCard Size Problem

Unfortunately, when I was extracting the individual cards from the image, I found that the cards were not equidistant. Some cards had a border of 2 pixels rather than 1 due to the image having being resized. I manually created an array of distances between cards when cutting them out, to ensure that all cards were the exact same size.

First, I set the height and width of all of the cards as cardWidth and cardHeight. Initially I was using y\_between for the distance between the cards in the initial image however this didn’t work as the vertical distances between the cards wasn’t the same for all cards. So, I manually created the cardTop and cardBot arrays so that the cards that were cut out didn’t include any whitespace at the top or the bottom of the cards.

Luckily the horizontal distance and the orientation of the cards were the same, so I could use x\_between for the horizontal distance between cards.

I used a nested loop to loop through all the cards on the screen, calculating the location of the left-hand side of the current card using the variables from the loops multiplied by the distance between the cards. I calculated the location of the right of the card using the location of the left + the width of the card. The top and bottom of the card I simply pulled from the cardTop and cardBot lists.

I then cropped the initial image with these parameters and set it to the variable im1, which then gets saved and automatically generated a filename using the suits and cardNum lists. These filenames are used later to access the individual cards and display them on the screen.

The format of the filenames I used for each card is:

*<suit-name><card-rank>.png*

So, for instance, the Queen of Clubs card image has the name CQ.png

## Integration

### Structure

I ran into a few errors with the structuring of my code early on, it wasn’t feasible to have all the code in the same file simply mixed together, so I made new classes called GameState and PlayingArea in new files to keep the code readable and maintainable.

I started with all of my code in my main file, called Solitaire Game.py, but I decided to move the creation of my stacks, into a new GameState class, which handles all the data about the current state of the game, i.e. what the current game that the user is playing looks like. This contains information such as which cards are in the stacks, suit stacks, deck and deck discard (see terminology section).

I also created the new class called PlayingArea, this class handles everything about the UI, drawing the cards on the screen and it contains the functions that get called when the user inputs a drop, drag or click event.

#### Text Description automatically generatedText Description automatically generatedCode before:

#### Text Description automatically generatedText Description automatically generatedCode after:

## Log of Ongoing Testing

### Blank Boxes

The graphics library I was using, Tkinter, caused blank boxes to pop up every time it displayed an image. The graphics library had a bug where it used the wrong root object when displaying images with the filename provided. I changed the image display to be the same as for a blank image, which didn’t cause a box to pop up. This fixed the bug.

#### Chart, histogram Description automatically generatedScreenshot:

#### Text Description automatically generatedCode Before:

#### Text Description automatically generatedCode After:

### CardStack class

With the current version of my code I was struggling with the card stack (see terminology section) knowing how many of its cards were showing the back and how many were showing the front, so I decided to make a card stack class rather than a list so that it could record the number of cards that have not been turned over in a backNum, which it could then use when drawing the card stack on the screen.

The game state now contained a list of card stack objects. Each of those card stack object contained a list of cards that the card stack had in it, as well as several functions to perform various operations on the individual card stack, such as adding to the cards list and removing all cards below a certain point.

#### Text Description automatically generatedCode Before:

Text

Description automatically generated

#### Text Description automatically generatedText Description automatically generatedCode After:

### Middle of card

The point used when displaying a card wasn’t the top left of the card as I originally thought, but was instead the middle of the card, I had a function that checked whether a card was clicked on but the box that registered the click wasn’t on the card, I amended my function to check for the middle of the card instead.

#### Code before:

#### Code after:

### Displaying deck discard

Text

Description automatically generatedI was struggling to find a way to display 3 cards if the deck discard has more than 3 cards but only display 0, 1 or 2 if the deck discard has less than 3 cards. I ended up using min to the find the number of cards to display in the discard pile, which would be a maximum of 3 when the length of list card deck >= 3 and the length of the deck discard if it is < 3.

### Finding the clicked card

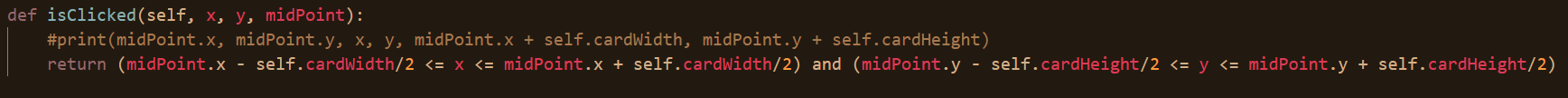
There are multiple different cards that can be clicked on, or nothing at all so I had to search the rectangle of each card to see if the clicked point was inside it. Once I found the clicked on card (or found that the click was not on a card) I had to decide whether the card was able to be moved or not. Any face-up card that can be clicked on can be moved, with the exception of the cards lower in the deck discard, however I decided that if the user clicked on any card in the deck discard, the top card of the deck discard would be moved.

My first version was simply searching through the places it was valid to click on and returning true or false depending on if the click was valid or not, I ended up changing the structure of my code and moved the validation of the click into an validator function in the game controller, which controls the rules of the game.

My new code combined the click and drop validators into one function called findEventLocation, which would find the location of both a click and a drop and would return variables to check the validation of both, which would be handled in the game controller.

#### Code before:

Text

Description automatically generated

#### Code After:

Text

Description automatically generatedText

Description automatically generated

### Dragging in a drag

#### A picture containing square Description automatically generatedBefore move:

#### Square Description automatically generated with low confidenceAfter move:

#### Error messages:

Text

Description automatically generated

After some investigation, I worked out that the error occurs because the graphics library is maintaining a list of all items that are drawn on the screen at any time. When undrawing, the library uses this list the delete the item from the screen, however when the program undraws an item not in this list, an exception occurs.

It turns out that a drag event can occur before another drag event has finished, hence the undrawing of the first drag event results in the item being removed from the list and then the second drag event tries to undraw an item not in the list and throws an exception.

My solution to this is to ensure that the drag function cannot be re-entered by using a flag called inDrag, which is checked as soon as the drag function is entered and if set, the drag exits immediately.

### Drawing error

Occasionally, when moving a card, the display would fail to redraw everything.

I wasn’t sure why this was happening at first and thought it had something to do with the suit stacks as that was where the error occurred. However as I tested more, I realised that the problem wasn’t solely with dropping a card onto a suit stack, as the problem persisted when I tried to drop on a stack as well. This indicated that there was something more going on. That it was the act of dragging and dropping a card that caused the visual error.

#### Before move:

A picture containing square

Description automatically generated

#### After move:

A picture containing square

Description automatically generated

I found that it correctly undraws everything before the moved card but has an error with undrawing the moved card itself. I realised that this occurs because while moving, the program undraws and redraws the card at the cursor location. If the card was dropped before it redrew then the undraw function would attempt to undraw a card that had never been drawn, resulting in an error.

### Using a threading lock

I decided to add a lock to attempt to ensure that a drag couldn’t be called while a drop was being executed. I thought that would make sure that the sequence of undrawing and redrawing was always executed. So first while dragging, the lock was false, i.e. the program was in the middle of a drop, then the lock would release, allowing the drop to be performed and the drag would terminate. While dropping, the drop would acquire the lock so that a drag couldn’t be performed, then it would execute the drop and finally would release the lock so that the user could drag again.

#### Text Description automatically generatedCode before:

Text

Description automatically generated

#### Text Description automatically generatedCode After:

Text

Description automatically generated

#### Before move:

A picture containing square

Description automatically generated

#### After move:

A picture containing square

Description automatically generated

However this only made the problem worse as it simply crashed instead of giving a blank screen. After some investigation, inserting print statements into both the drag and the drop, I realised that the drag function is suspended part way through when a drop event occurs and does not continue until the drop event has completed. The problem with the lock was that the drag would obtain the lock and, before the lock was released, the drop event would occur. This meant that the drop function was unable to acquire the lock and waited indefinitely for it. Because the drag function could not continue until the drop event had completed, this caused a deadlock and the program stopped responding.

### Delayed drop

I eventually found a solution where, if a drag is in progress when a drop occurs, then I store the required information for the drop in a new variable called dropToBe. At the end of the drag, I check if dropToBe has been set and if so, execute the drop, which has been delayed. This ensures that the drag is always finished before the drop is carried out. This solution worked and fixed the issue.

Text

Description automatically generated

Text

Description automatically generated

### Dropping at the bottom of a stack

When adding the validation to move cards onto a stack, I didn’t check where in the stack I was adding the new card, meaning a card could be placed on a card higher in the stack, resulting in two cards of the same colour and number appearing in the same stack right after one another, as I was able to drop a card onto another card that is part-way up the stack, in this example I am able to drop a red six on the black seven, and then I can drop a second red six onto the same black seven.

I fixed this by passing a new parameter into the isValid function, which will check to see if the card we are dropping on is the last one in the stack.

#### A picture containing qr code Description automatically generatedBefore:

A picture containing timeline

Description automatically generatedText

Description automatically generated

#### Text Description automatically generatedA picture containing text Description automatically generatedAfter:

Text

Description automatically generated

### Row number not correct

When inserting new test entries into the database, the row number wasn’t correct when the selected elements were sorted, as shown in the screenshot below:

#### Screenshot before:

Calendar

Description automatically generated

#### Code before:

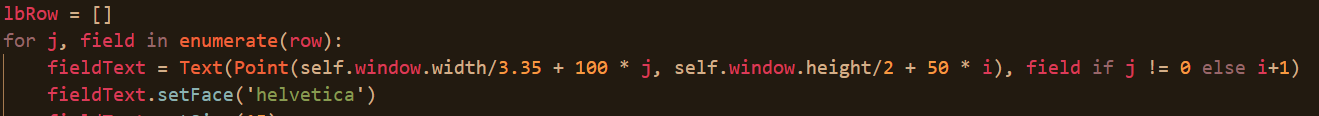
Instead of using the row number from the database, I artificially created the position on the leaderboard instead, so that the position would always increment by one going down rather than be the location of the row in the database.

#### Screenshot after:

A picture containing text, electronics

Description automatically generated

#### Code after:



### Displayed too many entries

When the home screen displayed the top 5 scores, it would instead display 6 scores, this is because of a logic error in my code that would break after 6 rows and not 5 as loops start from 0 in Python. I simply changed the > 5 to >= 5, which fixed the logic error.

#### Graphical user interface, calendar Description automatically generatedScreenshot before:

#### Graphical user interface, application, website Description automatically generatedCode before:

#### Calendar Description automatically generatedScreenshot after:

#### Code after:

Text

Description automatically generated

# Testing

In order to thoroughly test the game, I decided to implement several different forms of testing.

* Component testing
* Integrative testing
* Usability testing based on prototypes
* Final testing
* End-user testing

## Component testing

For my component testing, I decided to create unit tests for main functions in my program, to ensure that they were all working correctly, I set up test cases and ensured that the output that was given by these functions was correct using the given input.

I used the unittest module of Python, which provides a framework for creating and running unit tests at a function or object level. This framework uses assertions to test whether the function acts as expected. If it does then the unit test passes and if not then it fails.

### Insertion sort testing

Graphical user interface

Description automatically generatedThe first thing that I unit tested was the insertion sort. I created a number of different cases where I passed in different sets of data, as my insertion sort function directly edited the list that was passed in, I created a copy which I sorted with a sorted version of the list using Python’s inbuilt sort function and also with manually sorted data. I then asserted that these two lists were identical and if they were then the unit test would pass.

On the right is the results of my unit tests for the insertion sort. Meaning that all my tests passed as expected.

#### testSort:

Text

Description automatically generatedThis was the first test and it was just a simple list made of positive integers which I used to test the basic functionality of my insertion sort.

#### testSortBackwards:

Text

Description automatically generatedI wanted to test if my insertion sort could handle doing the most amount of computation possible, so I gave it a list that was as unsorted as possible, where every element will have to be moved multiple spaces.

#### testSortBoth:

Text

Description automatically generatedI was unsure if the inbuilt Python sort was sorting by ascending or descending order, and as I wanted my insertion sort to sort in ascending order, I was concerned that both were sorting in descending order, which would cause a problem later on. So instead of using an automatic sort, I decided to manually sort the list in ascending order to ensure that my sort was sorting in ascending order.

#### testSortSorted:

Text

Description automatically generatedI wanted to make sure that my insertion sort wouldn’t ruin an already perfectly sorted list, and it would just give back the same list that it was given, i.e. it wouldn’t make any changes at all.

#### testSortDecimal:

Text

Description automatically generatedI tested to make that my insertion sort could handle decimals, and wouldn’t give an error or an incorrect sort.

#### testSortNegative:

Text

Description automatically generatedI wanted to ensure that my insertion sort would also work with purely negative integers.

#### testSortNegativePositive:

Text

Description automatically generatedI wanted to make sure that the insertion sort would be correct for a list of both negative and positive integers.

#### testSortNegativePositiveBoth:

Text

Description automatically generatedI tested using another manually sorted list to test if my insertion sort was working as I wanted it to work for negative numbers and wasn’t giving an incorrect answer that happened to be the same as the Python inbuilt sorting function.

#### testSortAllSame:

Text

Description automatically generatedI tested to ensure that my function would work when all the values in the list were the same, mainly to make sure that it wouldn’t give an error when comparing identical values.

#### testSortEmpty:

Text

Description automatically generatedI wanted to make sure that my insertion sort function wouldn’t give an error when presented with an empty list.

### Card Deck testing

To test the card deck I employed a similar strategy as with the insertion sort, creating specific scenarios and having the test pass/fail if the assertion was met or not.

Testing the card deck class was slightly different however the insertion sort is a function whereas the card deck is a class. The first thing I did was to create an instance of the card deck in all my tests that I would manipulate and compare with my expectation of what should happen. Since the card deck defines the layout of the cards on the screen and logic for the game, it is important to test whether each of the functions in the class are working as intended.

Graphical user interface, application

Description automatically generated

#### testDeckLen:

The first thing I decided to test was whether the length of the cards list was the actual length of a standard deck of cards, as missing any card could easily make the game unplayable and unwinnable.

Text

Description automatically generated

#### testDeckTopCard:

Text

Description automatically generatedI wanted to make sure that the removeTop function was actually removing the correct card, so I made a copy and checked whether that card was still in the cards list. If the card wasn’t in the list then the test would fail.

#### testDeckLenRemove:

Text

Description automatically generatedI tested to make sure that removeTop was only removing a single card from the cards list.

#### testDeckRandomness:

Graphical user interface, text, chat or text message

Description automatically generatedI wanted to see whether the card deck was shuffling cards sufficiently. So I decided to create a test to check whether the list of cards was randomised. I started with turning all cards into a single value from 0 to 51, so I could quantify the difference from a fully sorted list and see if my list was suitably random.

I then made a list of the absolute differences between these values (ignoring any negatives), this would show how far apart similar cards are, for example, a two of clubs next to a nine of diamonds would give a high score.

I then took the mean of these differences and asserted that it was greater than 13. The higher the mean of differences the better as it means that similar cards are generally not very close to each other.

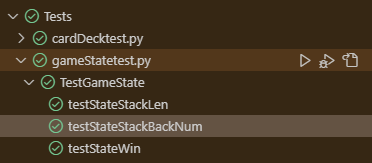
This means that on average, a card is a quarter of the deck away from the next card. This is a very basic indication of randomness, as a sorted deck, or a partially randomised deck would have a low mean. If any sets of cards were unsorted, for example, the seven, eight and nine of clubs all next to each other, the mean would be significantly reduced.

Text

Description automatically generatedI also calculated the standard deviation of the list of differences and asserted that this was greater than 10, again, we need a number that is as high as possible. This is because the higher the number, the higher the spread of differences. This is good because it means that sometimes very different cards are close and sometimes very similar cards are close. Which is a pretty accurate representation of true randomness.

### Game State testing

Testing the game state proved to be very difficult, I decided to test whether certain elements of the game state were as expected. First I created an instance of the game state that I called testState. I then tested it to see if it created elements correctly, based on what the user would expect to see on the screen.

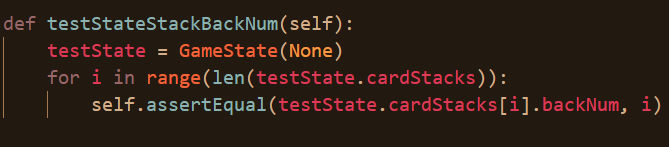


#### testStateStackLen:

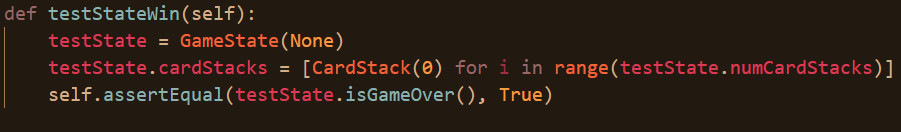
Text

Description automatically generatedThe first thing I tested was the stacks, I wanted to see if the initial length of the stacks was correct on the initialisation of the game state.

#### testStateStackBackNum:

I wanted to make sure that the backNums of all of the stacks was initially correct on initialisation.

#### testStateWin:

I wanted to test if my win condition would succeed on a very basic level if I manually set all the backNums of all the stacks to 0, then test the win condition. As my win condition checks the backNums of all stacks, the isGameOver function, which checks if the player has won, should return True.

## Integrative testing

My strategy for integrative testing is to set up a number of test cases which automatically generate game states and then present the tester with a scenario where they can make moves which test specific edge cases in the game logic.

To run a test case, a command line argument was added to the program. Starting the program with the following command will execute a test case:

python.exe “Solitaire game.py” -t test\_case

Where test\_case is one of the test case names illustrated in the table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case name | Scenario | Setup | Expected outcome | Actual outcome |
| mid\_drop | Dropping a card in the centre of two face up cards in the stack | Two sevens of same colour and an eight of opposite colour at the top of the stacks | First seven is able to be dropped on the eight but the second seven is not | As  expected |
| king\_blank | Dropping a king onto an empty stack | Ace on the first stack and a king | King can be dropped on an empty stack | As  expected |
| win | All cards turned over in the stacks | One last card to be turned over | Game ends when card is turned over | As  expected |
| queen\_king | Dropping a queen onto a king of the same colour | A king and queen of the same colour | The queen cannot be dropped on the king | As  expected |

### Mid\_drop testing

The deck is set up so that a pair of cards can easily be placed on a suit-stack and a further card which has the same value as the lower card in the suit-stack is also available. For example, two red sevens and a black eight. I tested this as when I was implementing my code, I came across this bug and wrote some code that would detect if the card was dropped at the bottom of the stack, this involves returning a new variable from the findEventLocation function that detects the position that the card is dropped on in the stack. This variable is then checked when validating the drop and if the card is not dropped at the bottom of the stack, the drop is deemed to be invalid and the card is moved back to its original position.

### King\_blank testing

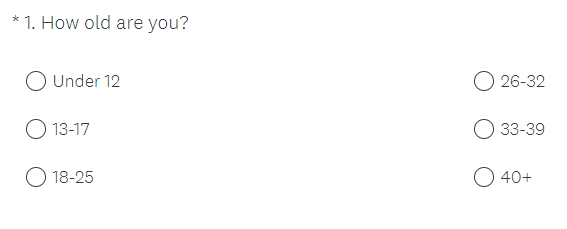
The deck is set up so that a stack can be cleared easily to allow the king to be dropped on an empty stack, for this I put an ace on the first stack, which could be easily put on any of the four blank suit stacks, this would empty the first stack and allow the king that I put on the bottom of the second stack to be put on the empty first stack.

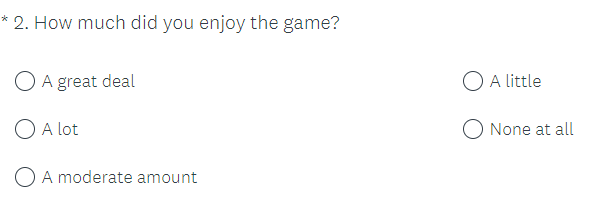
### Win testing

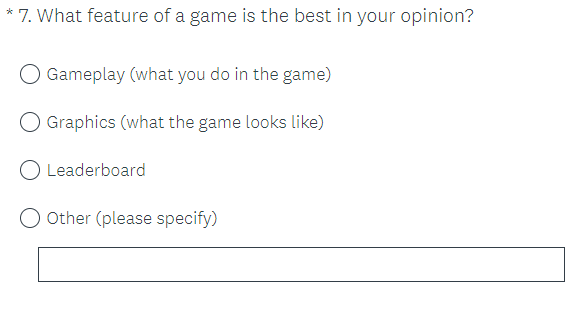
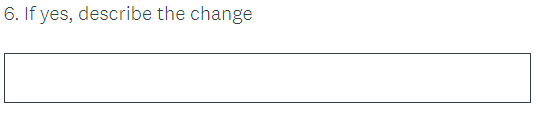
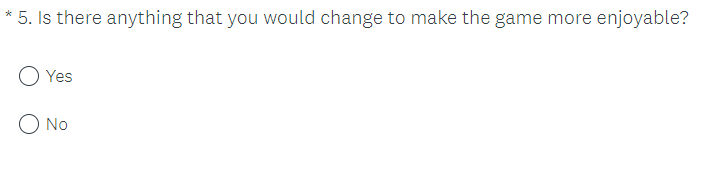
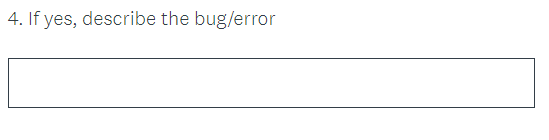
Unlike the other test scenarios, for this test I needed to do more than simply modify the stacks to create a test case. Instead I modified the entire game state to give a situation where only one card was left to be turned over. As my win condition is triggered when all the cards in the stacks have been turned over

## End-user testing

For my end-user testing, I decided to have potential users in my target audience play the game and provide feedback in the form of a survey, I created the survey to answer a few questions that I had about how well my game was designed and how it felt to play, i.e. was it clunky, slow or did it crash. I wanted to see if users would play my game instead of other, similar games that are already publicly available.







# Evaluation