# ASSIGNMENT 2

Design and Development document

COM410 SOFTWARE DEVELOPMENT II
B00792485 B00751280

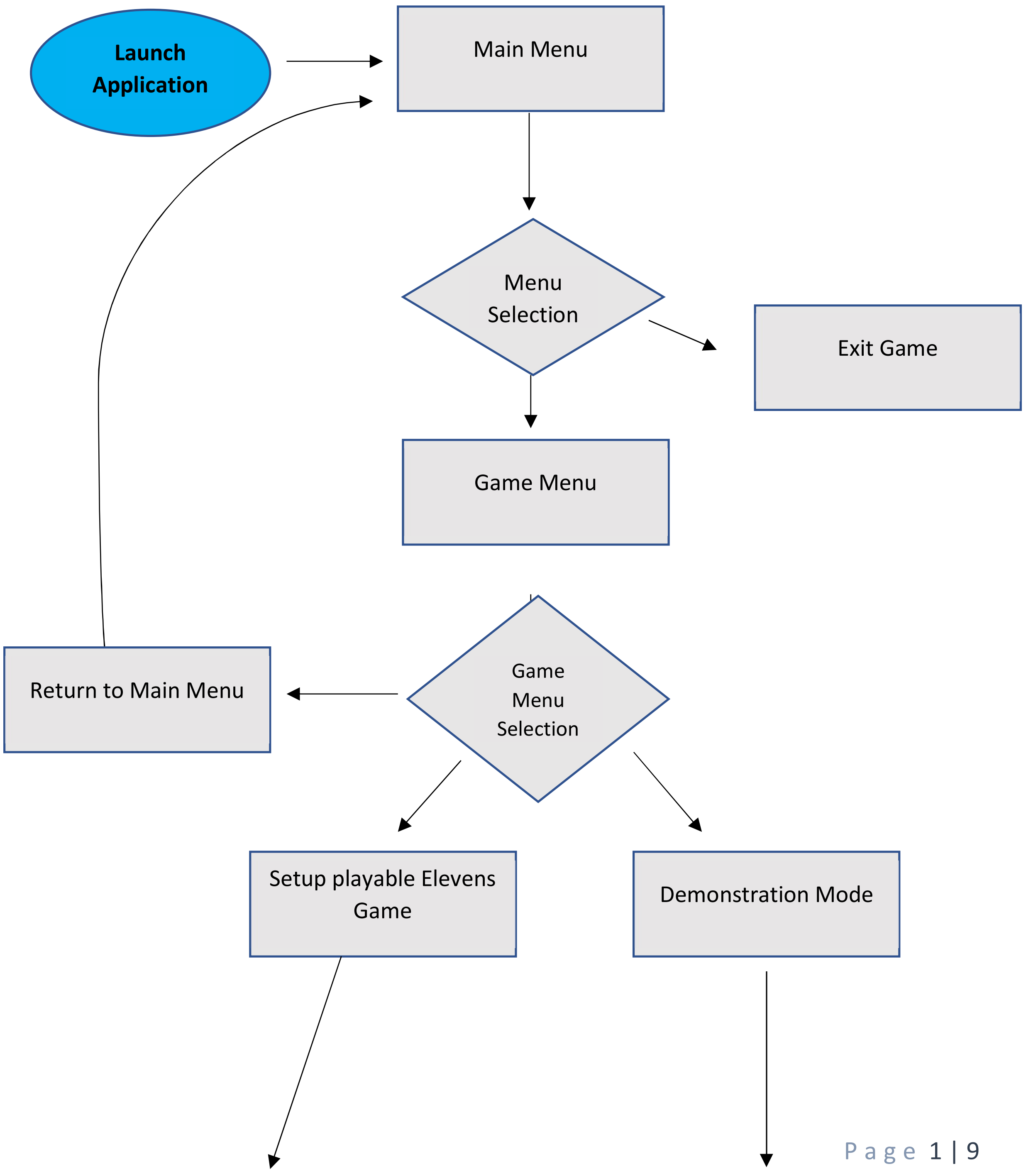
## Contents

## Intro

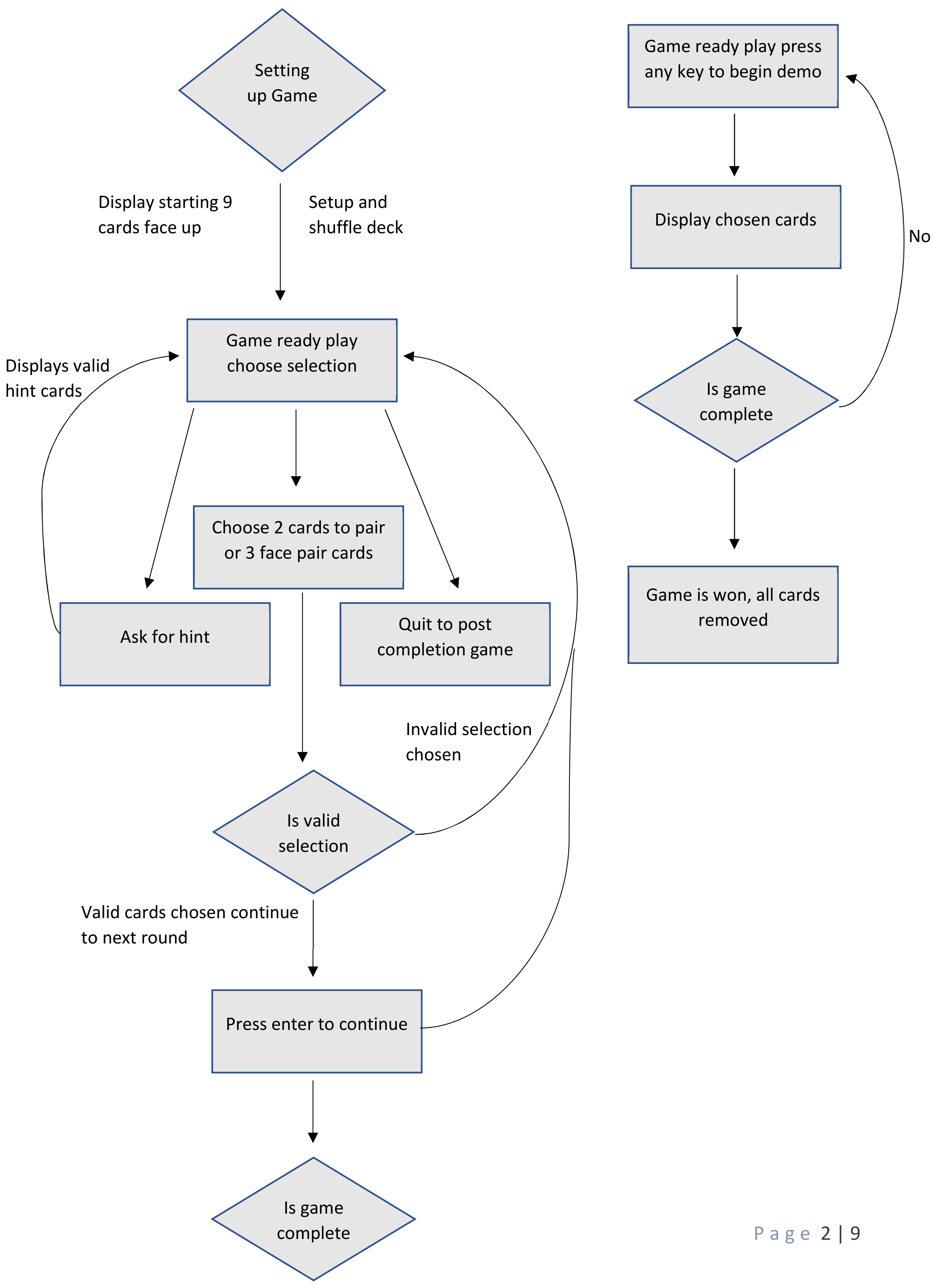
To start off the development project we met on zoom and talked around the specification of the coursework set to us and outlined the requirements needed to be developed in our java application. We did some research on the rules and how to play the elevens game in real life to gain a better understanding how the game is played in the real world.

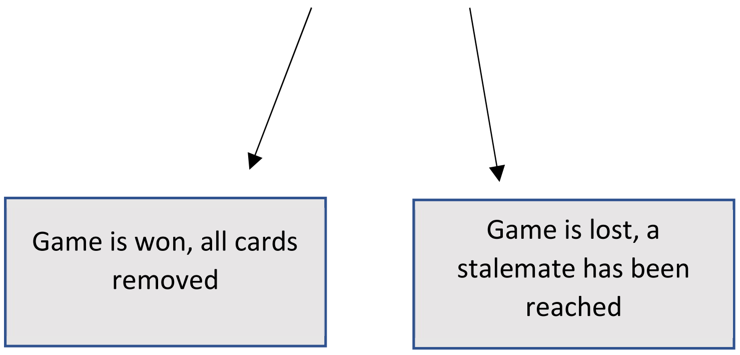
Having gone over the rules and coursework specification document we went about creating a flow chart to show how the application of the Elevens game will work and how it will flow.

## FLOW chart

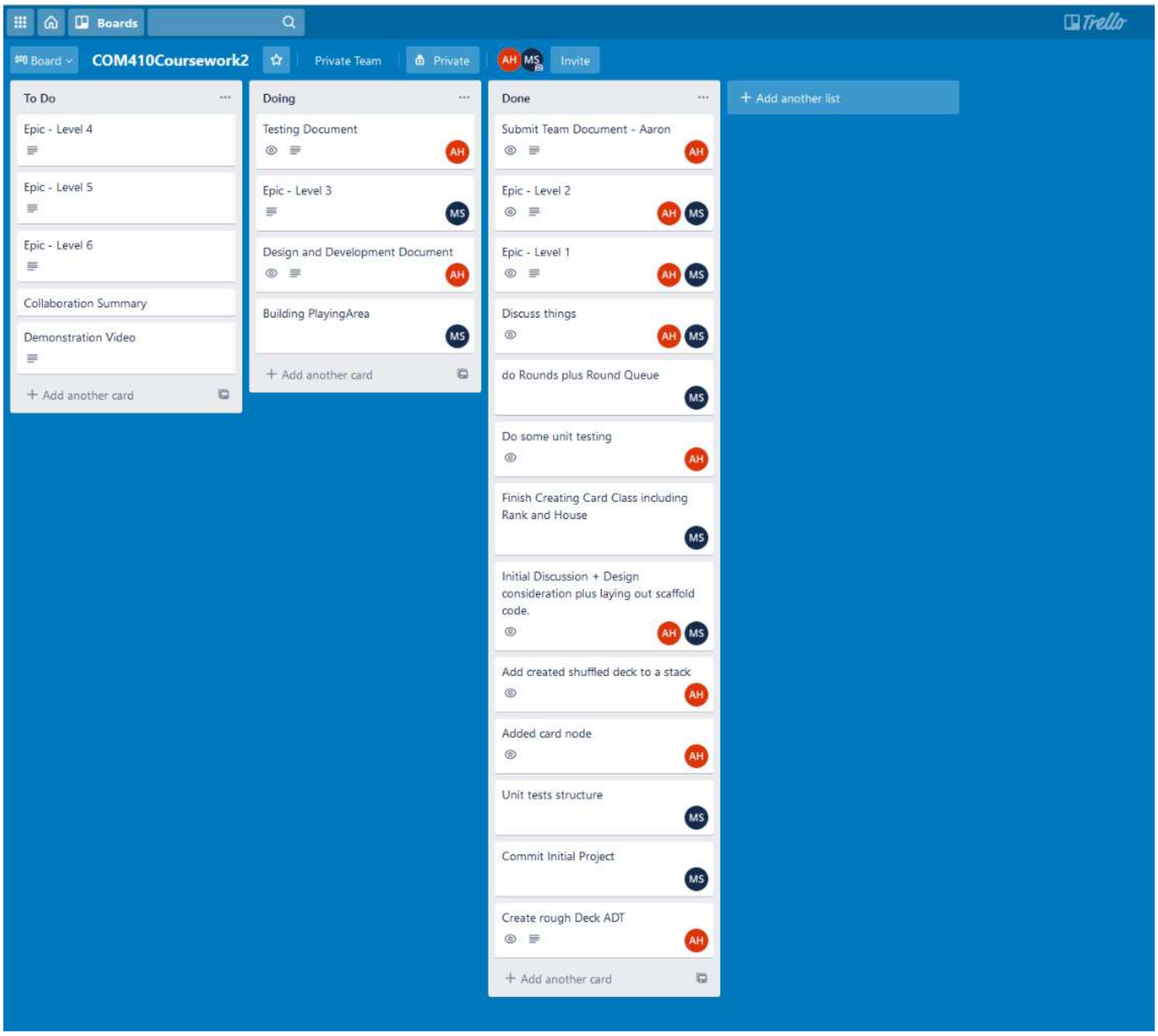


Aaron Hoy B00792485 & Michael Watters B00751280


Setting up Game
Display starting 9 cards face up
Setup and shuffle deck
Game ready play choose selection
Displays valid hint cards
Choose 2 cards to pair or 3 face pair cards
Ask for hint
Quit to post completion game
Invalid selection chosen
Is valid selection
Valid cards chosen continue to next round
Press enter to continue
Is game complete
Game ready play press any key to begin demo
Display chosen cards
No
Is game complete
Game is won, all cards removed


Game is won, all cards removed
Game is lost, a stalemate has been reached

Now having some understanding of the rules and the requirements we went about doing some project management. We created a Trello board, to keep track of tasks and ensure the workload is shared out and to ensure we can keep an eye on the deadline and have the app developed on time and maintain organisation. We also created a github repository to store the code base to be accessible and push changed to in a central location, so both members of the team can work on the same code base and keep it up to date.


COM410Coursework2

| To Do | Doing | Done |
|---|---|---|
| Epic - Level 4 | Testing Document (AH) | Submit Team Document - Aaron (AH) |
| Epic - Level 5 | Epic - Level 3 (MS) | Epic - Level 2 (AH, MS) |
| Epic - Level 6 | Design and Development Document (AH) | Epic - Level 1 (AH, MS) |
| Collaboration Summary | Building PlayingArea (MS) | Discuss things (AH, MS) |
| Demonstration Video | | do Rounds plus Round Queue (MS) |
| | | Do some unit testing (AH) |
| | | Finish Creating Card Class including Rank and House (MS) |
| | | Initial Discussion + Design consideration plus laying out scaffold code. (AH, MS) |
| | | Add created shuffled deck to a stack (AH) |
| | | Added card node (AH) |
| | | Unit tests structure (MS) |
| | | Commit Initial Project (MS) |
| | | Create rough Deck ADT (AH) |

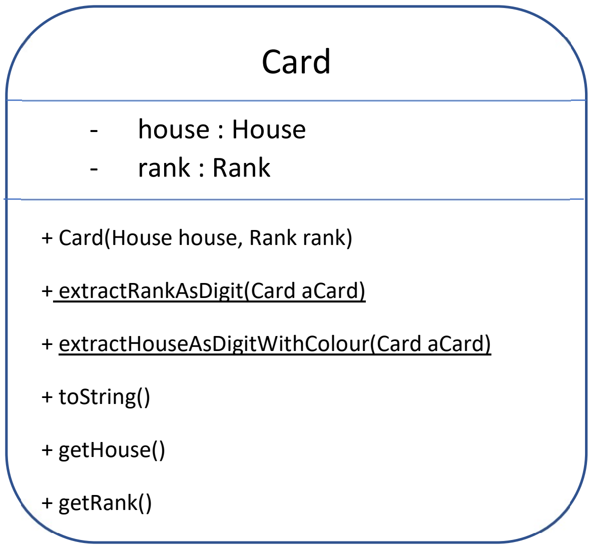
The requirements of the Elevens game were then discussed and broken down into a table below and set under a choice of titles to show their priority, required, desirable, nice to have and out of scope. This was to help prioritise which parts of the functionality should be developed first and in what order to be developed in.

## Requirements

| Required functionality by end user: | Required/ Desirable/ Nice to have/ Out of scope |
|---|---|
| Functionality for the player to have a new game present when they launch the elevens game. | Required |
| The player should have a ready to play deck being dealt 9 cards from a shuffled deck. | Required |
| The player should be able to choose pairs of cards to be removed from the game according to elevens rules, having two cards add up to 11. | Required |
| When cards have been removed, the player should automatically have new cards dealt from the deck, to their playing hand, 9 cards before the next round begins. | Required |
| The application should be able to inform the player when the game has been won, when all cards have been removed from the game. | Required |
| The application should be able to inform the player when the game has been lost, when there are no more cards pairs which add to give 11, therefore the game cannot progress. | Required |
| The game should be able to provide a hint to the player on the players request, giving a valid move or informing the player that no moves are possible. | Required |
| When the game has finished the app should be able to replay the game, showing each move made, allowing the player to prompt the app to show each replay move by a keystroke. | Desirable |
| The app should play a game in demo mode where the player only has the prompt the game to play the next available move. The game should show a commentary of each move played. | Nice to have |

After agreeing the requirements priority after going through the coursework specifications. We then though about what classes could be create in the java project that might represent different parts of the game and functionality to be implemented. So, after these thoughts on what classes there might be, we went about making a Unified modelling language class diagram for the java application.

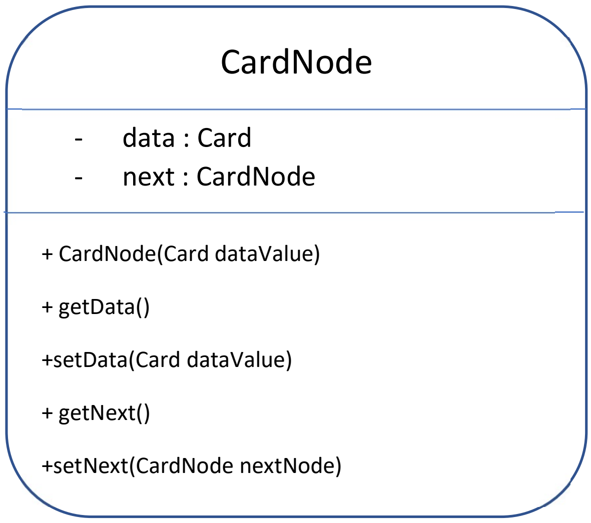
## UML class diagrams



Card

- house : House
- rank : Rank

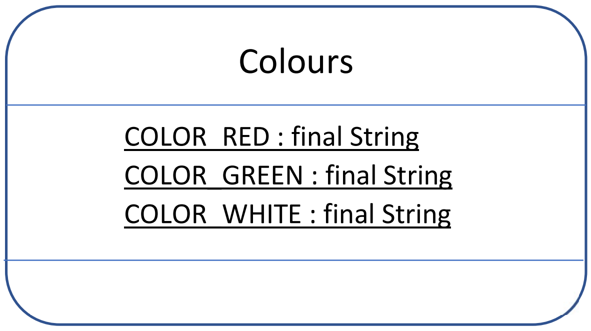
+ Card(House house, Rank rank)
+ extractRankAsDigit(Card aCard)
+ extractHouseAsDigitWithColour(Card aCard)
+ toString()
+ getHouse()
+ getRank()



CardNode

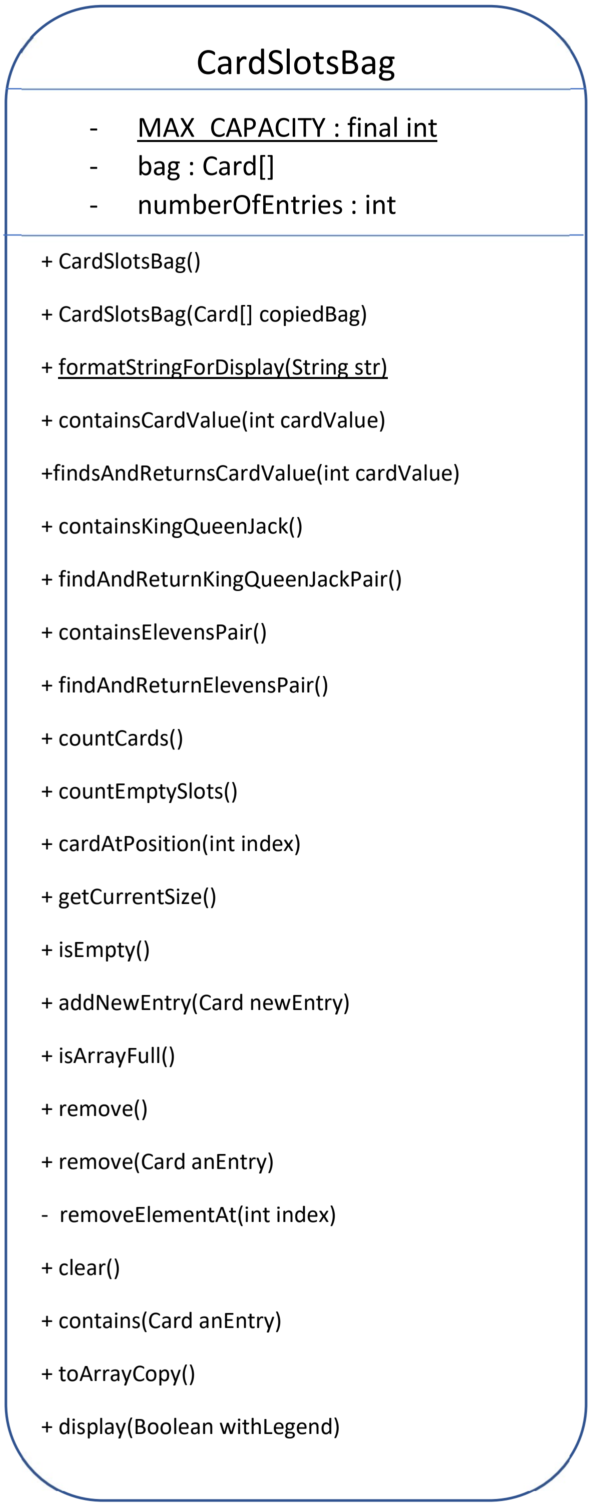
- data : Card
- next : CardNode

+ CardNode(Card dataValue)
+ getData()
+setData(Card dataValue)
+ getNext()
+setNext(CardNode nextNode)



Colours

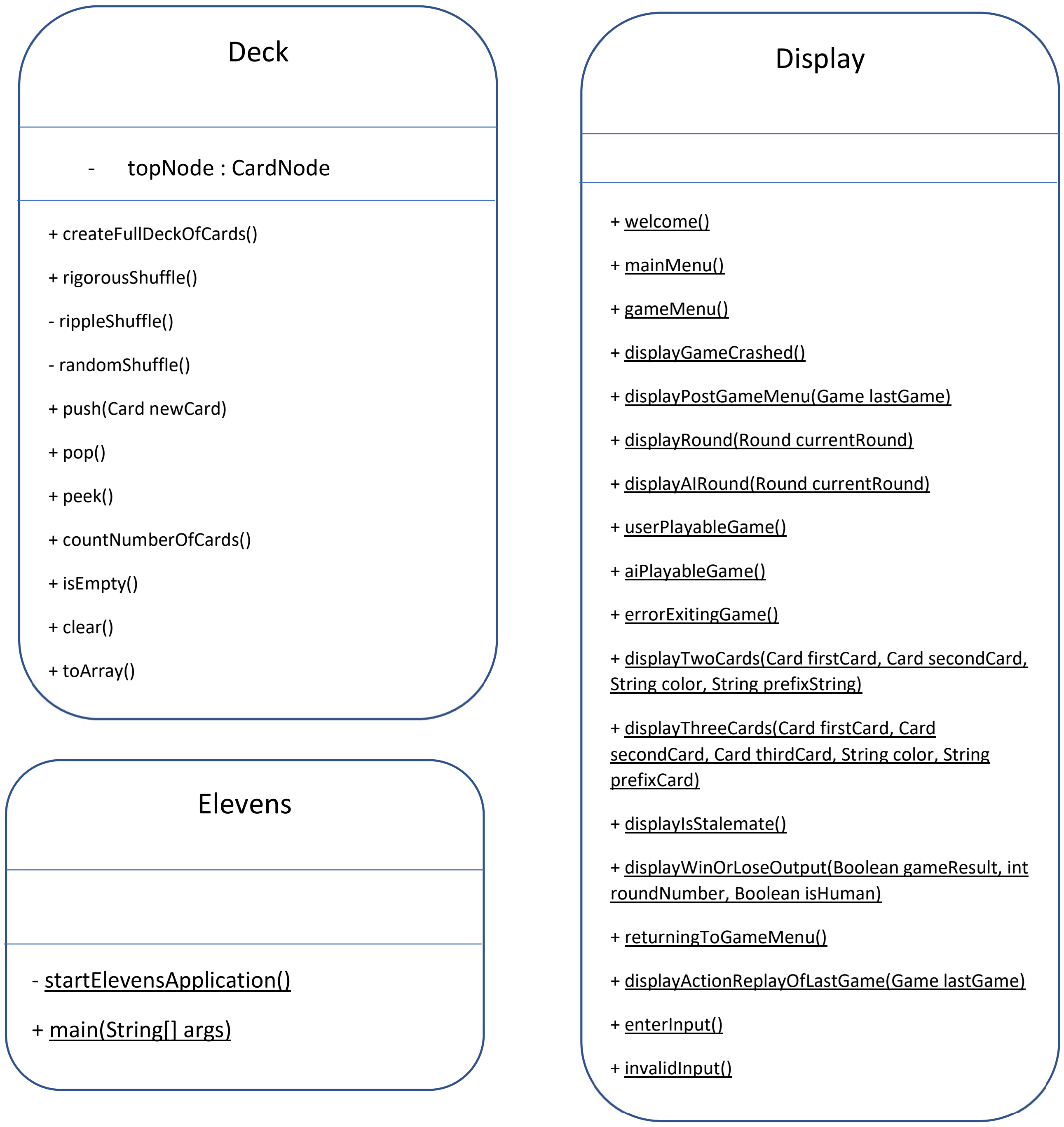
COLOR_RED : final String
COLOR_GREEN : final String
COLOR_WHITE : final String



CardSlotsBag

- MAX_CAPACITY : final int
- bag : Card[]
- numberOfEntries : int

+ CardSlotsBag()
+ CardSlotsBag(Card[] copiedBag)
+ formatStringForDisplay(String str)
+ containsCardValue(int cardValue)
+findsAndReturnsCardValue(int cardValue)
+ containsKingQueenJack()
+ findAndReturnKingQueenJackPair()
+ containsElevensPair()
+ findAndReturnElevensPair()
+ countCards()
+ countEmptySlots()
+ cardAtPosition(int index)
+ getCurrentSize()
+ isEmpty()
+ addNewEntry(Card newEntry)
+ isArrayFull()
+ remove()
+ remove(Card anEntry)
- removeElementAt(int index)
+ clear()
+ contains(Card anEntry)
+ toArrayCopy()
+ display(Boolean withLegend)



**Deck**

- topNode : CardNode

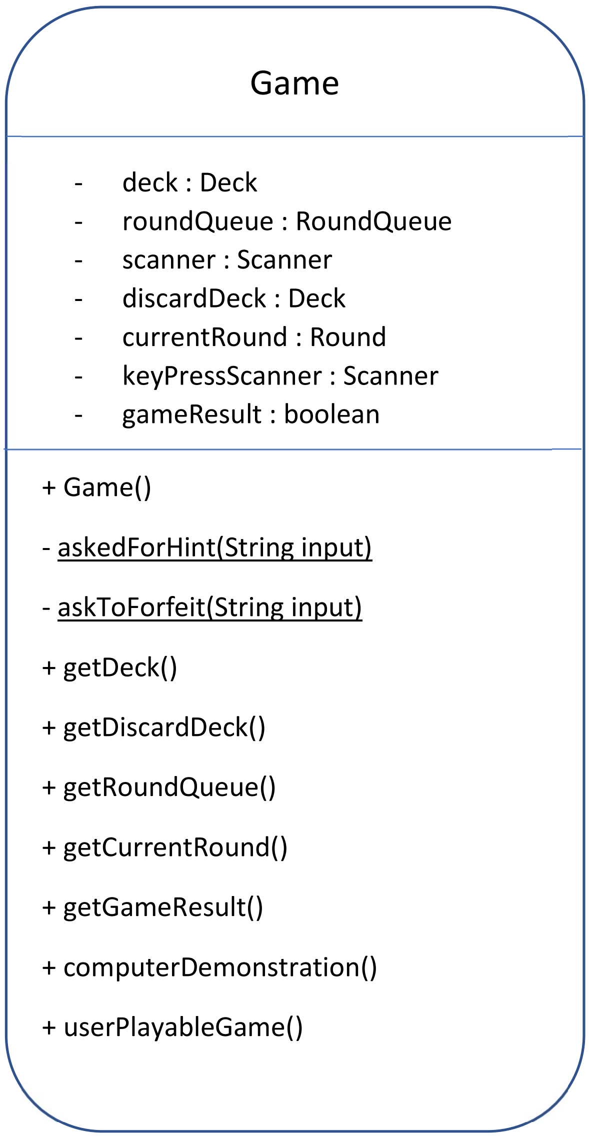
+ createFullDeckOfCards()
+ rigorousShuffle()
- rippleShuffle()
- randomShuffle()
+ push(Card newCard)
+ pop()
+ peek()
+ countNumberOfCards()
+ isEmpty()
+ clear()
+ toArray()

**Elevens**

- startElevensApplication()
+ main(String[] args)

**Display**

+ welcome()
+ mainMenu()
+ gameMenu()
+ displayGameCrashed()
+ displayPostGameMenu(Game lastGame)
+ displayRound(Round currentRound)
+ displayAIRound(Round currentRound)
+ userPlayableGame()
+ aiPlayableGame()
+ errorExitingGame()
+ displayTwoCards(Card firstCard, Card secondCard, String color, String prefixString)
+ displayThreeCards(Card firstCard, Card secondCard, Card thirdCard, String color, String prefixCard)
+ displayIsStalemate()
+ displayWinOrLoseOutput(Boolean gameResult, int roundNumber, Boolean isHuman)
+ returningToGameMenu()
+ displayActionReplayOfLastGame(Game lastGame)
+ enterInput()
+ invalidInput()


Game

- deck : Deck
- roundQueue : RoundQueue
- scanner : Scanner
- discardDeck : Deck
- currentRound : Round
- keyPressScanner : Scanner
- gameResult : boolean

+ Game()

- askedForHint(String input)

- askToForfeit(String input)

+ getDeck()

+ getDiscardDeck()

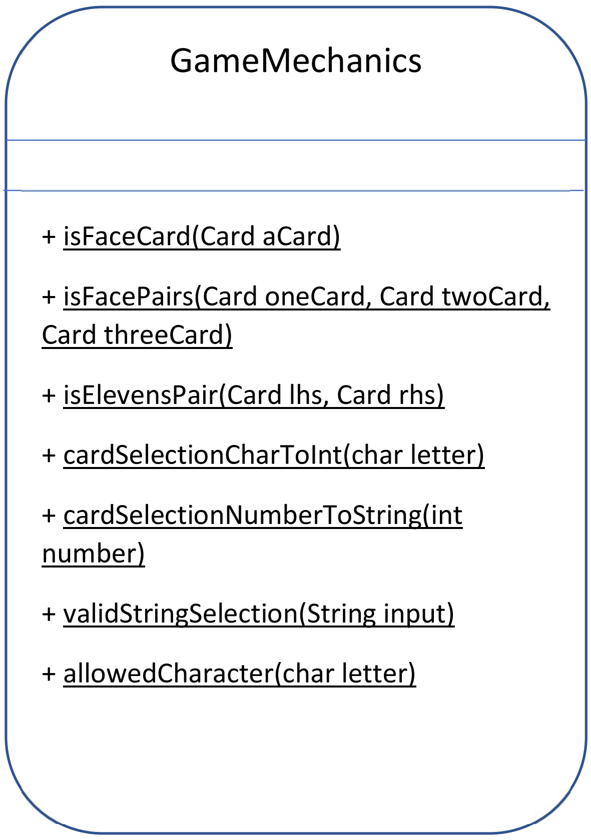
+ getRoundQueue()

+ getCurrentRound()

+ getGameResult()

+ computerDemonstration()

+ userPlayableGame()


GameMechanics

+ isFaceCard(Card aCard)

+ isFacePairs(Card oneCard, Card twoCard, Card threeCard)

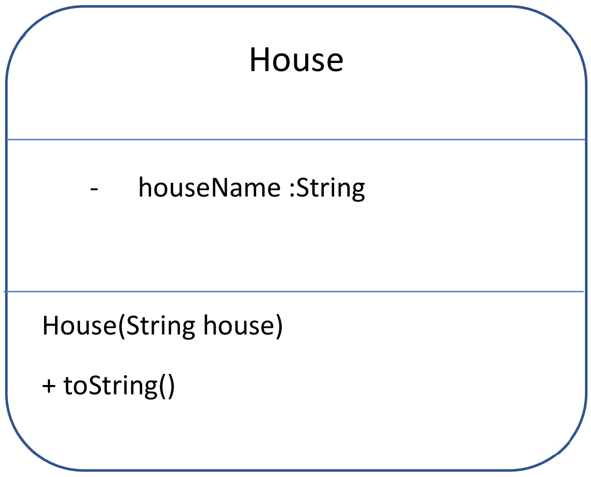
+ isElevensPair(Card lhs, Card rhs)

+ cardSelectionCharToInt(char letter)

+ cardSelectionNumberToString(int number)

+ validStringSelection(String input)

+ allowedCharacter(char letter)


House

- houseName :String

House(String house)

+ toString()

## Menu

- scanner : Scanner
- keyPressScanner : Scanner

+ MainMenu()

+ GameMenu()

+ PostGameMenu()

## Rank

- rank : String
- value : int

Rank(String rank, int value)

+ getRank()

+ getValue()

+toString()

## Round

- roundNumber : int
- nextRound : Round
- cardsInPlayBag: CardSlotsBag
- roundMemoryDrawCards : CardSlotsBag
- roundMemoryDiscardCards : CardSlotsBag

+ Round(int roundNumber, CardSlotsBag cardsInPlayBag)

+ Round(int roundNumber)

- drawFromDeck(Deck deck)

+ isStalemate()

+ replaceEmptyCardSlots(Deck deck)

+ getRoundMemoryDrawCards()

+ getRoundMemoryDiscardCards()

+ updateDiscardCardMemory(Card card)

+ getRoundNumber()

+ setRoundNumber(int roundNumber)

+ getCardsInPlayBag()

+ setCardsInPlayBag(CardSlotsBag cardsInPlayBag)

+ getNextRound()

+ setNextRound(Round nextRound)

## RoundQueue

- front : Round
- rear : Round

+ RoundQueue()

+ enqueue(Round newRound)

+ dequeue()

+ getFront()

+ getRear()

+ isEmpty()

+ clear()

We then went to creating the project, scaffolding out some classes for the basic parts of functionality and thought of certain tasks that we need to get working on to fulfil the first couple of requirements. We went through each class and thought about which abstract data types we should use. We were going to use arrays for some of the classes, such as the deck class to hold the deck of cards. But we decided that I made most sense to use a stack ADT for the deck, since a deck of cards is lifted from top of the stack to the bottom in the Elevens card game in real life. It was decided that we would use a queue to add the round into that works along with the round class. We also used a bag class for the Card class to choose the cards. The deck class also works with the cardnode class based on the Node class from the lecture example, to get the position and data of the items in the stack. We then added those tasks to the Trello board to keep track it. And got started on our initial tasks to get the app underway. We kept in contact through WhatsApp to what work is needed and what we were working on and if we encountered any issues.