# Hong Kong Diploma Of Secondary Education
# Information and Communication Technology
# School-based Assessment (SBA)

St. Paul's College

Year: 2017

# Title: Big2

---

Name: Wong Tsz Nok Joshua  Class: 6D  Class No.: 19

# Content

# A. Objectives

## 1. Description of Big2

Big2 is a iOS game coded with Swift[1] originated from a poker card game called Big Two[2], which is very popular in Asia especially in Hong Kong. A deck without pokers is distributed to 4 players randomly and evenly (every player get 13 cards). The player got Diamond 3 will start first and take turns in clockwise direction. The objective is to get rid of all of the player's cards. Each turn can only play cards with valid combinations which show in the following table.

**Table 1: Big2 Game Rule**

| Type | Combinations | Detail | Priority (1 highest) |
|---|---|---|---|
| Five-card Hands | Straight | Any cards in sequence and determine value by biggest card in straight sequence. e.g. smallest straight is 1-2-3-4-5 and largest is 10-J-Q-K-A | 5 |
| | Flush | Any 5 cards of same suit | 4 |
| | Full House | Composite of a triple and a pair | 3 |
| | Four-of-a-kind | Composite of a quad and a card | 2 |
| | Straight Flush | Composite of straight and flush | 1 |
| Triples | Triple | 3 cards with same rank | N/A |
| Pair | Pair | 2 cards with same rank | N/A |
| Single | Single | 1 card | N/A |

In each turn, a player can play a hand, which is combinations of cards, or decide to pass. The player should play the hand with the same type as the previous hand and in greater value, except the first player. Once there are no other players deal a hand greater than the hand on the top, that player can play any type with corresponding combinations. Each hand has a comparing card which is used to compare the size of a hand and determine can next player place the hand. Whenever next player remains 1 card, the player is required to "prevent" which is playing the greatest value in his hand, so as to make clear no conflicts of interest.

[1] Apple Developer Swift: https://developer.apple.com/swift/

[2] Big Two (Wikipedia): https://en.wikipedia.org/wiki/Big_two

## 2. Intended Users' Requirement

In this project, I have observed different kinds of Big Two card game in iOS Platform, which are 博雅鋤大地[3] and Big 2 Online[4]. Through observation, I can get the first-hand experience of users and analysis the game system.


**Screenshot 1: Other Big Two Card Game in iOS Platform**

The basic functions I found in both games including menu, setting, sorting of cards. Some features like searching for five-hand combination, animation effects, different kinds of rules and timing makes the game more interesting and faster pace. There is a scoring system which calculates the points get by each player. For a better experience for timing, the computer will automatically decide which card(s) should be played for players. Both games also support online matching to allow different users can play the game together which keeps the game refreshing.

[3] 博雅•鋤大地 (App Store): https://itunes.apple.com/cn/app/bo-ya-chu-da-de/id549724881?mt=8

[4] Big 2 Online (App Store): https://itunes.apple.com/hk/app/big2-online/id375750079?mt=8

In order to get user's requirement, I have interviewed some students for some information of the game. By doing the interview, I can collect some expectation from users. The result follows.

**Table 2: Interview From Students**

| Questions | Student 1 | Student 2 | Student 3 | Student 4 | Student 5 |
|---|---|---|---|---|---|
| **What basic functions other than the game should be included?** | Setting, Sound Effects | Scoring System | Menu, Timing | Leaderboard | Auto Save & Load Game |
| **What features can be incorporated into the game?** | Offline & Online gaming | In-app Purchase, Level System | Game Centre | Auto Sorting | Multiplayer, Search Five-card Hands |
| **What can make the game more interesting?** | Emotes | Chatting | Multiplayer, More Rules | Sound Effects | Better Graphics |
| **Will game with graphics interface better than command line interface?** | Yes | Yes | Yes | Yes | Yes |
| **Which input device is expected?** | Touch Screen | Mouse | Touch Screen | Touch Screen | Mouse |

Summarising the interview result, multiple players and using pointing devices as input can make the game having a better user experience. Some of the basic functions and features are mentioned in the games I observed. All of the interviewees agree that graphics is important in gaming.

From the observation and interview, basics function of Big2 should include a menu, setting, timing as well as scoring system. To make the game much interesting, features like multiplayer, new rules for Big Two and a level system can be incorporated into the game.

## 3. Objectives

There are lists of objectives during this project in basics and features of the game so as to makes the progress smooth. These objectives are referenced to the observation and interview I had conducted.

**Table 3: Objectives in Basic Functions**

| | Basic Functions |
|---|---|
| 1 | Visualising Elements in the Game. |
| 2 | Distributing and Collecting Cards from Players. |
| 3 | Playing Cards with Validation of Combinations. |
| 4 | Sorting Cards of Players. |
| 5 | Determine Winner. |
| 6 | Menu. |
| 7 | Settings. |
| 8 | Save & Load Game. |
| 9 | Tutorial Mode. |

**Table 4: Objectives in Feature Functions**

| | Feature Functions |
|---|---|
| 1 | Multiple Levels & Rules (Modes). |
| 2 | Scoring System. |
| 3 | Timing System. |
| 4 | Search Five-card Hands. |
| 5 | Leaderboard. |
| 6 | Smooth Animation. |
| 7 | Sound Effects. |
| 8 | Pause & Restart Game. |
| 9 | Profiles of players. |

## 4. Scope of Application

Big2 is an application aimed at allowing users/players to playing the game of Big Two. During the game, player's decision of hand is manually inputted via the touch screen. Computers' decisions are automatically processed and output their decision accordingly.

The inputs to the system include pointing coordinates, the user record, previous game record and the decision of player. These inputs are needed in order to maintain the game system. Status data is kept by the system to make the system smooth. For example, cards are facing up or down, are the cards ready to deal to ensure the gaming experience of players which do not show others card or cover player's card. The number of cards in player's hand is to be examined every turn when the player play a hand. Cards played by players and computers are outputted so as to alert user what cards have been played by others. A winning message will be sent to the screen when the player had played all of his cards.

Users of the gaming system of Big2 include mobile game players. It allows players to play Big Two with various functions like sorting of cards. The interface between hardware and software program is the touch screen of an iOS device as a communication boundary of users and gaming system. Big2 allows users to entertain through the game and make them relaxed.

This project consists of creating an application of game based on Big Two. It will be completed by January 2017. Modules of the game will include a game starter, game runner and game terminator. It aims at motivating players to practice logical thinking and memory which contributed to learning as well as entertaining.

# B. Analysis

## 1. Programming Languages

Before implementation, I need to decide which programming language will be used. This table summarised the feature of programming languages I have considered.

**Table 5: Comparison of Programming Languages**

| | C | C++ | Swift | Scala |
|---|---|---|---|---|
| **Paradigm** | Imperative, Structured | Multi-paradigm: Procedural, Functional, Object-oriented, Generic | Multi-paradigm: Protocol-oriented, Object-oriented, Functional, Imperative, Block Structured | Multi-paradigm: Functional, Object-oriented, Imperative, Concurrent |
| **Stable Release** | C11 (12/2011) | C++14 (18/8/2014) | 3.0 (13/9/2016) | 2.11.8 (3/8/2016) |
| **First Appeared** | 1972 | 1983 | 2/6/2014 | 20/1/2004 |
| **Platfrom** | Cross-platform | Cross-platform | Darwin, Linux, FreeBSD | JVM, JavaScript |
| **Time Learnt** | 4 years | 2 years | 1 year | 0.5 year |
| **Readability** | High | High | Very High | High |
| **Extend of Graphics Programming** | Low | High | Very High | High |
| **Supported Library** | Moderate | High | Very High | Very High |

From the above comparison[5] [6] [7] [8], Swift can manage more different kind of paradigm than the others and feature of protocol-oriented allow developers create more organised source code. The stable release of Swift has been the most updated, which have much more up-to-date features and bug fixes on language. Swift is also the latest appear language which has a much modern style of programming language and influenced by other languages. Readability, extend of graphics programming and supported library are very high which is suitable for

[5] C (Programming Language) (Wikipedia): https://en.wikipedia.org/wiki/C_(programming_language)

[6] C++ (Wikipedia): https://en.wikipedia.org/wiki/C%2B%2B

[7] Swift (Programming Language) (Wikipedia): https://en.wikipedia.org/wiki/Swift_(programming_language)

[8] Scala (Programming Language) (Wikipedia): https://en.wikipedia.org/wiki/Scala_(programming_language)

developing this project. Despite I have only learnt 1 year, Swift is easy to learn and mainly influenced by C family. Thus, the writing style is similar and able to learn quickly.

As Integrated Development Environment IDE of the language is also a vital factor in deciding using which language, I am going to compare them.

**Table 6: Comparison of Programming Language in IDE**

| | C | C++ | Swift | Scala |
|---|---|---|---|---|
| **IDE** | Code::Blocks, Dev-C++ | Code::Blocks, Dev-C++ | Xcode | Eclipse |
| **Debugging Tools** | Moderate | Moderate | Very High | High |
| **Pre-compile** | No | No | Yes | Yes |
| **Simulators** | No | No | Yes | Yes |
| **Ease to develop** | Moderate | Moderate | Very High | High |
| **Coding Conversion** | No | No | Yes | No |

Xcode[9] has a lot debugging tools and simulators for developing an application much easier. Swift code can be pre-compile and provide debug area so as to correct syntax error as soon as possible. Xcode provides a better experience to develop the game more easily with shorter development time. Coding conversion to latest version allows developers to do long term large scale development more convenient as some syntax may change when time passes.

To sum up the comparison[10] [11] [12] [13], Swift with its IDE Xcode seems to be the most suitable language to develop Big2 which requires a graphic interface. Using Xcode can make the project develop less difficult while producing similar elements and gaming system.

[9] Apple Developer Xcode: https://developer.apple.com/xcode/

[10] Code::Blocks (Wikipedia): https://en.wikipedia.org/wiki/Code::Blocks

[11] Dev-C++ (Wikipedia): https://en.wikipedia.org/wiki/Dev-C%2B%2B

[12] Xcode (Wikipedia): https://en.wikipedia.org/wiki/Xcode

[13] Eclipse (software) (Wikipedia): https://en.wikipedia.org/wiki/Eclipse_(software)

## 2. Hardware & Operating System Comparison

There are two major operating systems I can develop on, which are iOS[14] and macOS[15]. The Swift program can be run in both OS and perform well. I am going to compare which system should be developed in first priority.

**Table 7: Comparison of Operating System**

| | macOS | iOS |
|---|---|---|
| **Developer** | Apple Inc. | Apple Inc. |
| **Latest Release** | 10.12 Sierra (20/9/2016) | 10.0.2 (23/9/2016) |
| **Target** | Personal Computer,<br>Desktop,<br>Laptop | Smartphone,<br>Tablet |
| **Input Device** | Keyboard,<br>Mouse,<br>Trackpad | Touchscreen |
| **Platforms** | x86-64,<br>IA-32,<br>PowerPC | iPhone,<br>iPod,<br>iPad |
| **Suitable for Gaming** | Moderate | High |

By comparing macOS[16] and iOS[17], there are myriads of similarities as they are both designed by Apple Inc.. iOS has a benefit of targeting at smartphones and tablets as the popularity increases and outweigh desktop computers and laptops. Furthermore, the input device of iOS has a touchscreen which embedded keyboard and pointing device which can be used in multitouch. Platforms of iOS have an averagely high platform usage and computational power which beneficial to developing a game.

[14] Apple iOS: http://www.apple.com/hk/ios/ios-10/

[15] Apple macOS: http://www.apple.com/hk/macos

[16] macOS (Wikipedia): https://en.wikipedia.org/wiki/MacOS

[17] iOS (Wikipedia): https://en.wikipedia.org/wiki/IOS

# C. Design

## 1. Structure Overview

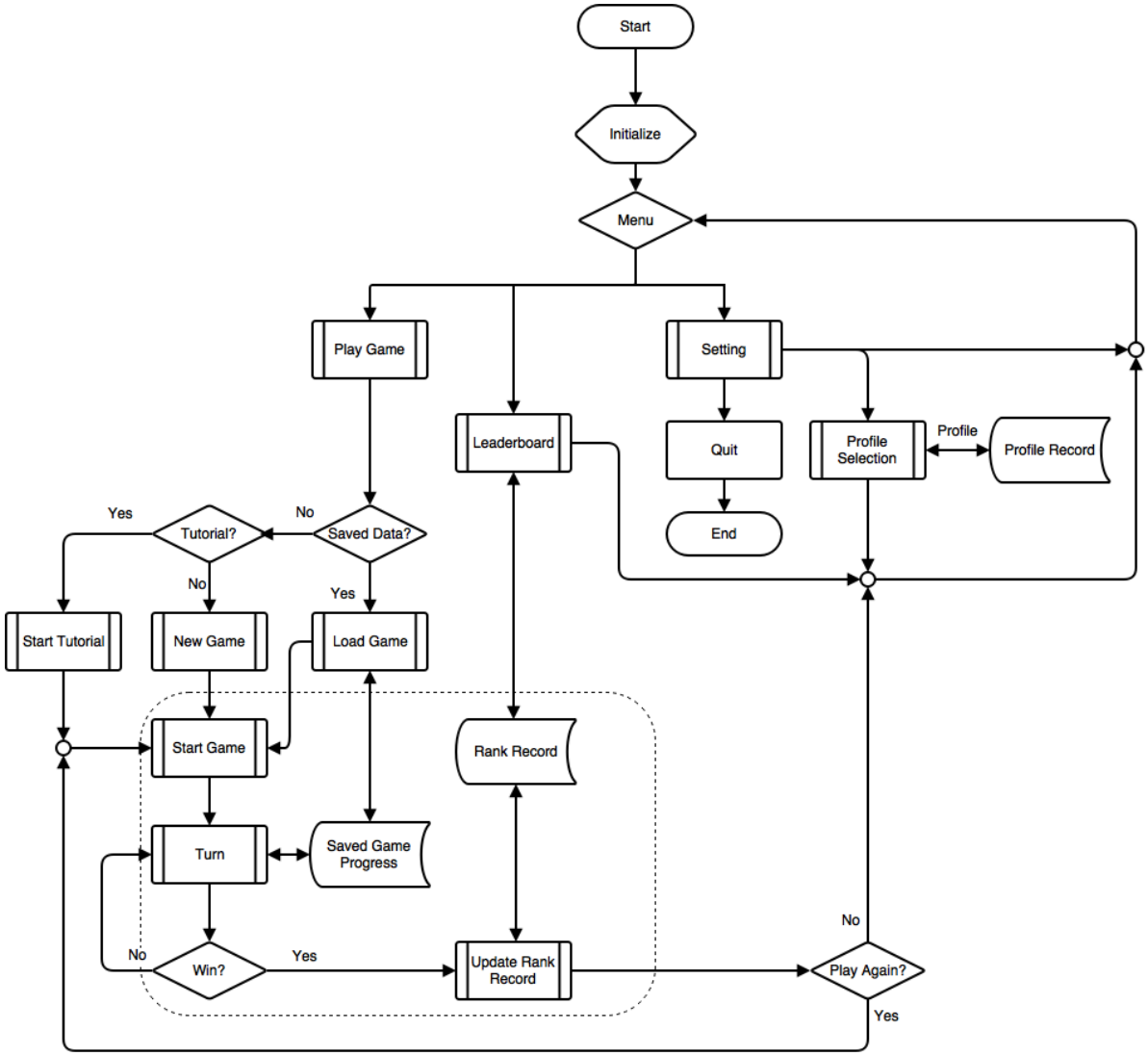



**Diagram 1: System Structure**

The system structural flowchart of Big2 above is a simplified version of the whole application with the main focus on overall process and data flow. The system includes a menu and game system. There are 3 options available in the main menu, which are play game, leaderboard and setting. The region highlighted with the dotted line is the main game system of Big2 and will be discussed later.

## 2. Data Structure

There are 3 main data structures, classes, in the Game System, namely, `Card`, `Hand` and `Computer`.

```
class Card : SpriteNode {
    var rank: String
    var suit: String
    var deal: Boolean
    var faceup: Boolean
    var cardFace: Texture
    var cardBack: Texture
}
```

`Card` class is the object of a card in a physical world. `rank` and `suit` are `String` which determines what the card is. `deal` and `faceup` are `Boolean` for flagging is the card for dealing and show to player respectively. `cardFace` and `cardBack` are the `Texture` which represents the rank and suit in a graphical manner. This structure defines what `card` is and makes development more handy with this framework.

```
class Hand {
    var Cards: [Card]
    var powerValue: String?
    var comparingCard: Card?
}
```

`Hand` class is the object of a set of cards for playing combination. `Cards` is an array of `Card` class to store an array of physical cards. `powerValue` is the power of the five-card hand combination which is an optional type for those hand with 1, 2 or 3 cards does not have a power value. `comparingCard` is the card for comparing the size of hand. With this structure, during deal cards will have a fixed structure for programming use.

```
class Computer {
    var Cards: [Card]
    var playerName: String
    var mode: Mode [easy|normal|hard]
    var position: Point
    var direction: Direction [horizontal|vertical]
    var scoreLabel: LabelNode
    var score: Int
}
```

`Computer` class is the object of a physical player, where `Player` class inherits from `Computer` class. `Cards` is an array of `Card` class on player's hand. `playerName` is a String for the name of the player. `mode` is a variable of enum `Mode` with easy, normal and hard. `position` is the central point of the cards of players. `direction` is a variable of enum `Direction` with horizontal and vertical for the order of cards. `score` and `scoreLabel` are used to store the score of that player.

With these 3 main classes, I can develop the game system more easily by inheriting this 3 objects.

## 3. Entity Relationship

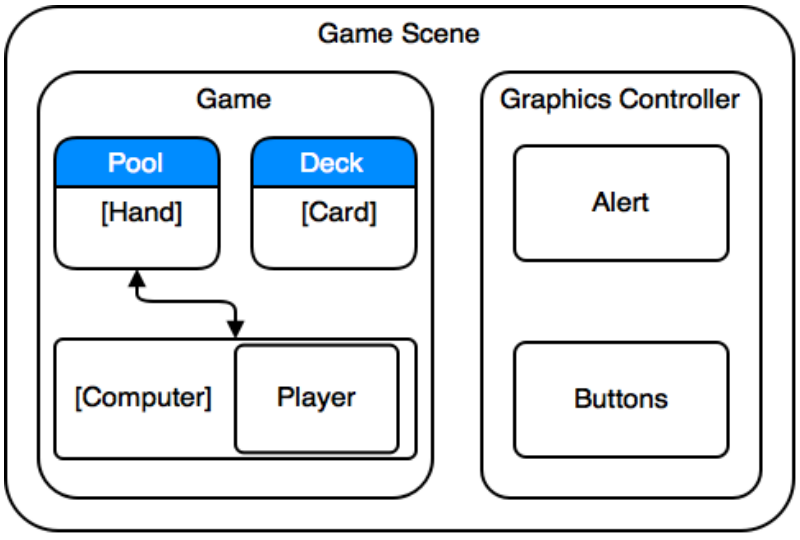



**Diagram 2: Entity Relationship**

This is the major entity relationship among the game system. Various component of the application can be visualised more easily. Players are interacting with the pool during the game.

## 4. Data Flow

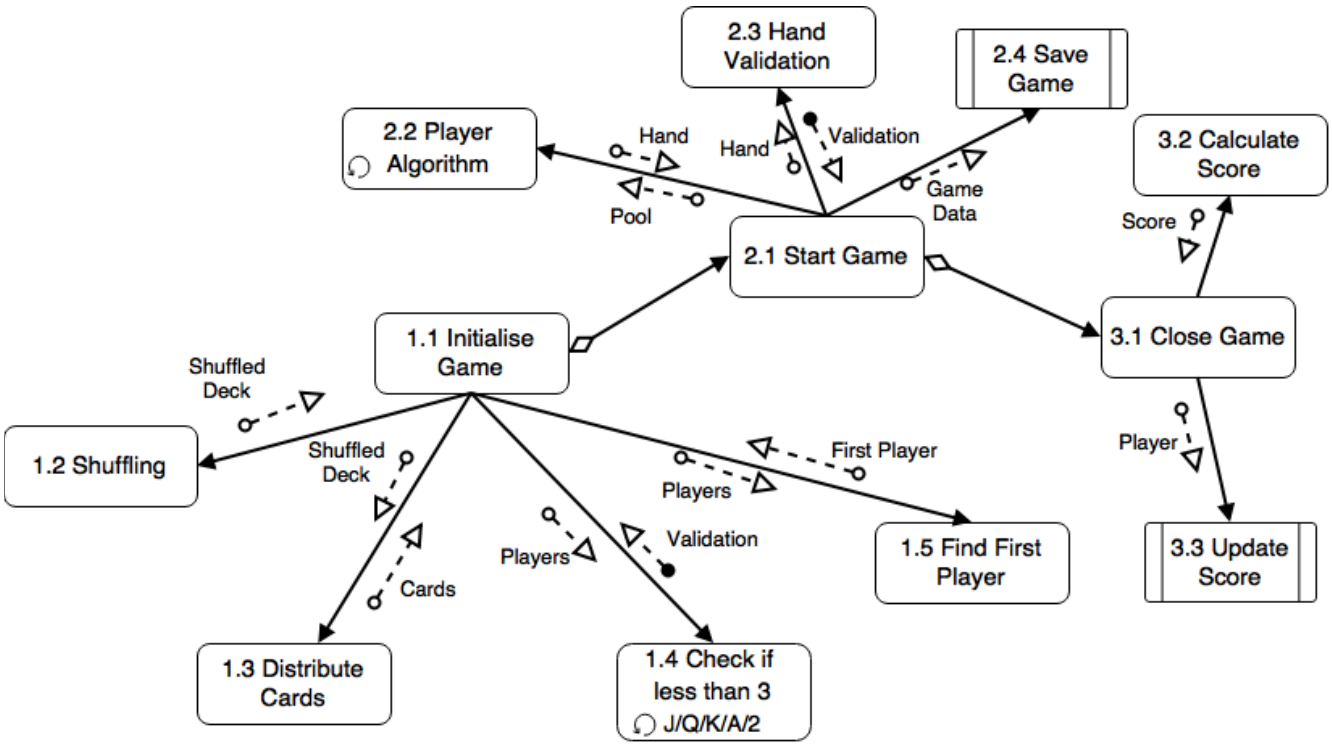



**Diagram 3: Data Flow Structure**

This is the main data flow structure in the game. There are 3 major stages include initialisation, game and after the game. Each stage consists of several modules having data and control flow among entities and modules.

First, shuffled deck is needed for distributing cards for players. Validating players having more than 3 J/Q/K/A/2 cards to prevent an unfair round of the game. If the above conditions satisfied, the program will find the first player of the game.

Then, each turn takes player deal card algorithm inputting pool and return a decision of hand. Validation of hand and save game will conducted by sending hand and game data.

At the end, the scores of the players are calculated by taking score data. Players are submitted to score library for updating record.

## 5. Gaming System

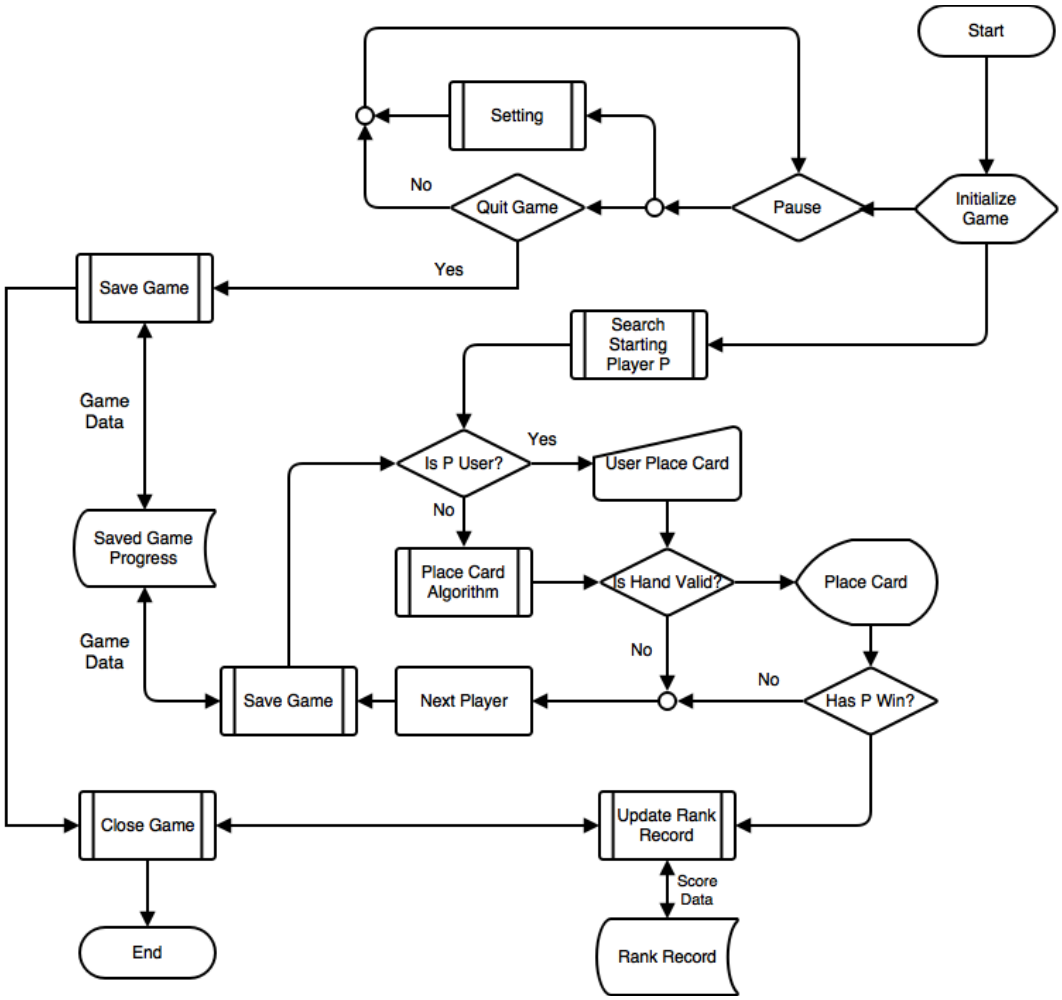



**Diagram 4: Game Flow Structure**

The game flow contains parallel operations in pause and game system. The main game flow consists of turns for playing and each round having validation of hand before placing the card. The game end when player P has no more cards on his hand and win.

There are 2 major data retrieve and update in the gaming system, which is saving the game progress and updating the rank record. This data is stored in the internal storage which will be used later and allow memory stored properly before terminating the application.

The game closing includes returning all cards and calculate scores of the players. This is crucial as next round of game will have issues when the game is not closed properly.

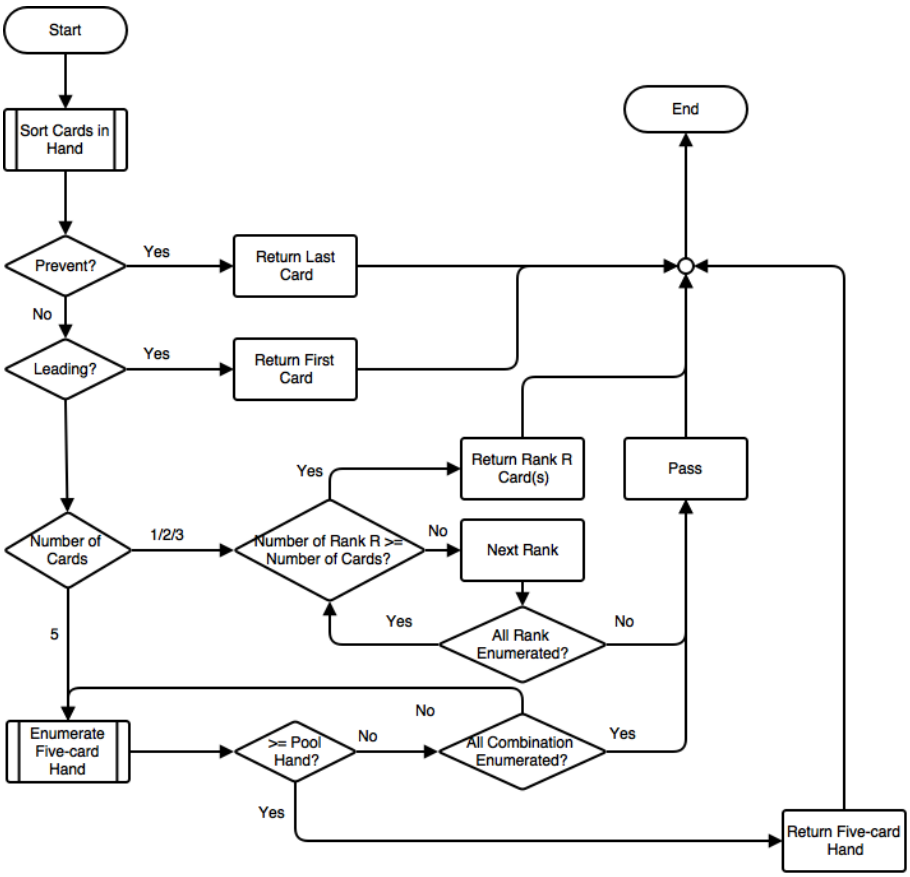



**Diagram 5: Place Card Algorithm**

There are 3 major decisions in choosing hand: Is there any prevention for the player having 1 card left? Is player leading which can place any card? Is the proposed hand greater than the hand in the pool? This is the outline of the algorithm and it varies when more difficulty of computers.

The algorithm showed above is the simplest method to enumerate what hands can be dealt. The final decision includes playing which hand or pass for the round. This algorithm does not require internal storage for calculation purpose.

In a more advanced algorithm, some techniques may be applied and modified. For instance, if the computer needs to prevent another player having 1 card, it can place more than one card; the computer can pre-estimate player's move by putting different hands to the pool. These techniques may require extra storage for calculation purpose such as marking which cards have been placed so as to react and decide a better move.

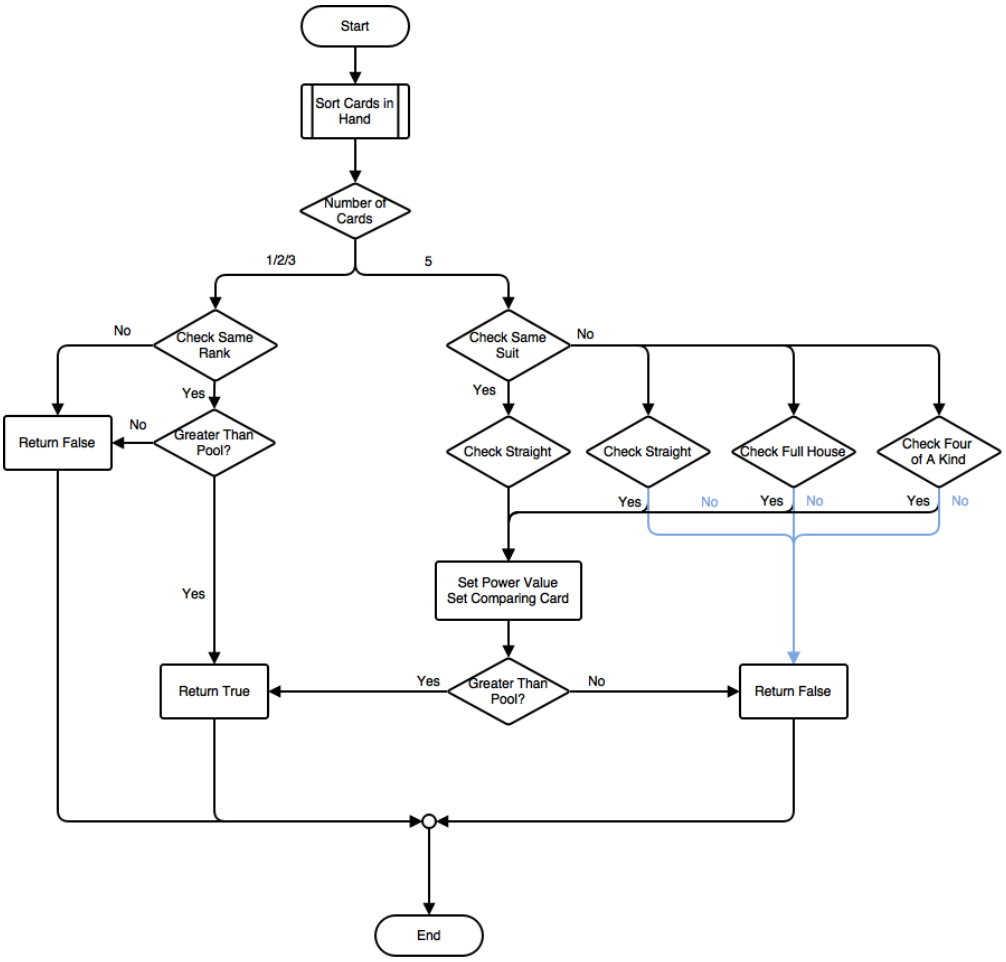



**Diagram 6: Hand Validation**

The validation of hand is divided into 2 section, 1 to 3 cards and 5 cards. By series of conditions, the type of combination of hand can be determined and return whether the value of the hand is greater than the hand in the pool.

In 1 to 3-cards hand, I will adopt the same algorithm to check as they are all of the same ranks. By using this characteristic of 1 to 3 cards, checking whether their ranks are same can validate quickly. As the hand in sorted, the program only needs to compare the last card of the hand to ensure the value of the hand is greater than that in the pool.

In 5-card-hand, checking the consistency of suit can determine flush and straight flush followed by checking straight. Otherwise, the hand can be checked one by one in straight, full house and quad. For every five-card-hand, the power value of the type and comparing card are needed for determining the value of hand for comparison.

## 6. Text File Structure

There are 3 major data required storage, game progress, profile and rank record. As profile and rank record can be directly stored in internal storage, it will not be stored in a text file. I will focus on the text structure of game progress. The Progress.txt stores the game progress and its drafted structure is below.

**Table 8: Text File Structure**

| | | |
|---|---|---|
| **Pool** | 3 | |
| | 2 | Card1, Card2 |
| | 5 | Card1, Card2, …, Card5 |
| | 1 | Card1 |
| **Player1** | 3 | Card1, Card2, Card3 |
| **Player2** | 2 | Card1, Card2 |
| **Player3** | 6 | Card1, Card2, …, Card6 |
| **Player4** | 8 | Card1, Card2, …, Card8 |
| **Game Information** | 1 | 1 |

The first row contains an integer n indicating the number of hands in the pool. In the following n lines, an integer x indicates the number of cards in the hand and followed by x cards represented by the rank and suit. The next 4 lines each has an integer y representing the number of cards on the corresponding players' hand and followed by y cards represented by the rank and suit. The file ends with 2 integers p and q representing the current player and leading player.

By following this structure, data access and modify can be done more systematically to reduce error during file read and write. The structure includes all main data for the game which able to construct the game even the application has been terminated.

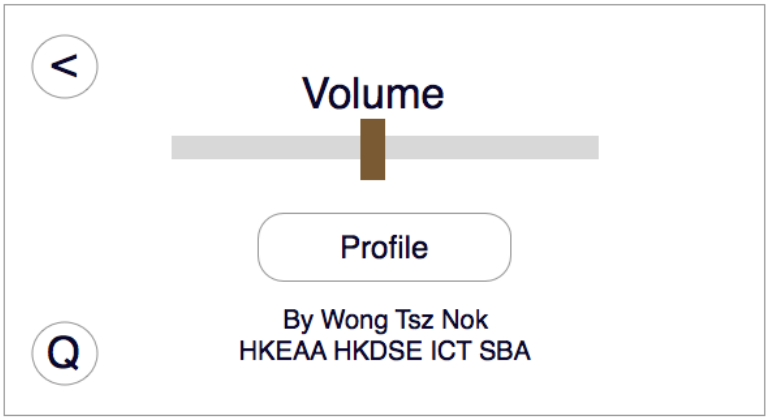
The data in the file will be loaded into the game when the start button is pushed. In each turn, the game progress will be updated. However, this may increase the load of hardware. Therefore, if possible, the game progress will be updated whenever the game terminates or quit.

## 7. Graphical User Interface

In order to create a good gaming experience for users, I will use GUI in stead of command line driven interface decided in the previous chapter. The input and output of Big2 will be a touch screen of iPhone or iPad which is a pointing device. The buttons and objects should put in an orderly manner to give better user experience.

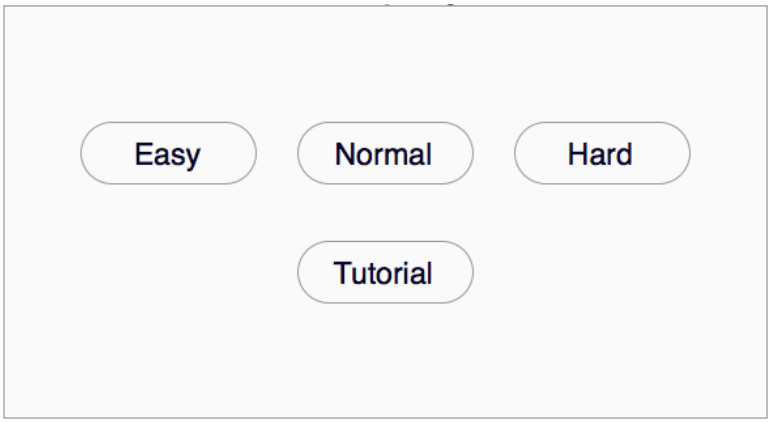

**Draft 1: Main Menu Design**

The main menu includes a setting button at the top-left corner. The start button is located at the centre of the display screen and a leaderboard button below it.



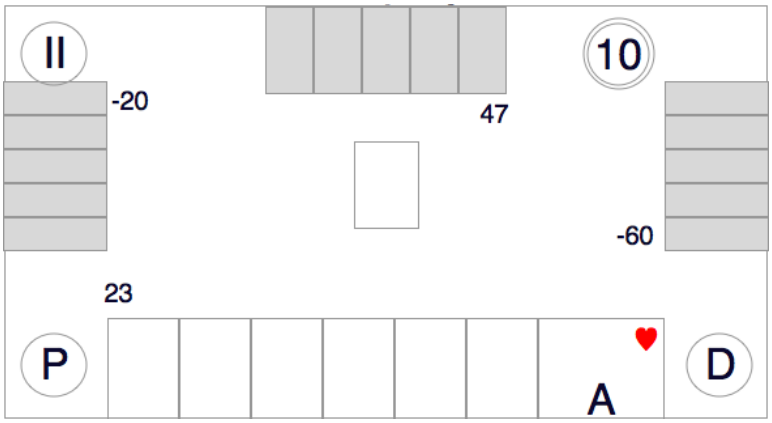

**Draft 2: Setting Design**

Volume control is at the centre of setting page and profile selection below it. Back button is at the top-left corner while quit button is at the bottom-right corner. Information of Big2 is included at the bottom.

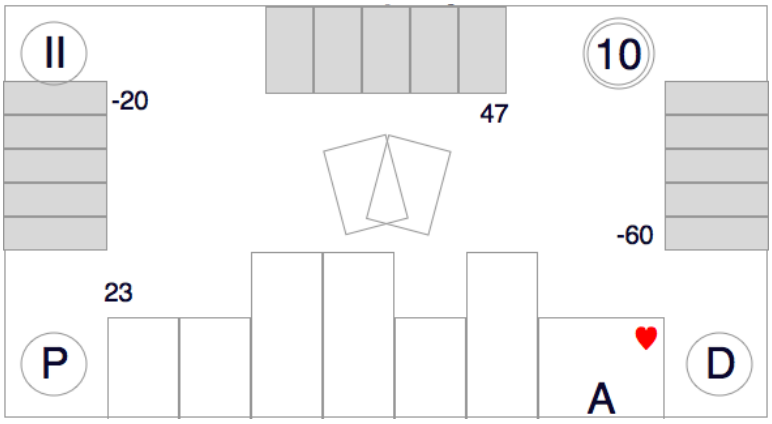



**Draft 3: Level Design**

There are 4 modes of the game which can be chosen when the start button is pushed and return to the main menu by tapping grey region.




**Draft 4: Game Design 1**

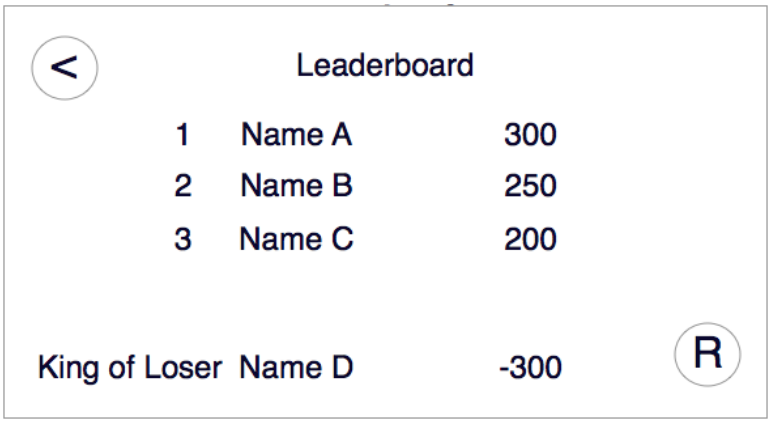



**Draft 5: Game Design 2**

The pause button is at the top-left corner. Pass and deal button are at bottom-left and bottom-right corner respectively. The timer will be at top-right corner. The pool is at the centre of the screen while players' card is displayed around it.

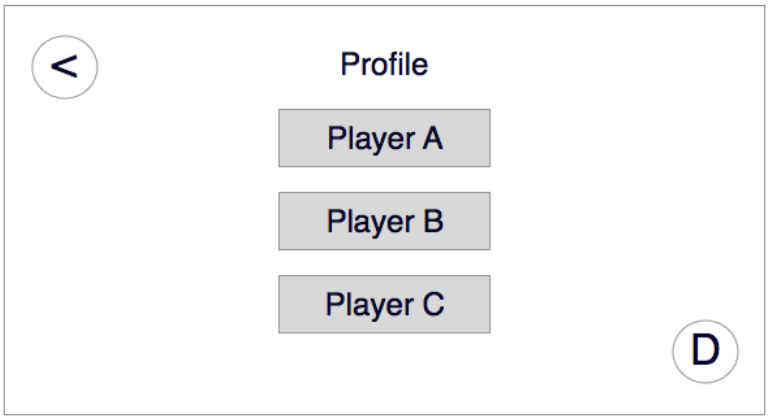

**Draft 6: Scoreboard Design**

This is the scoreboard design when the game ends. Winning message will be sent out at the centre with score calculation below it. The calculation is performed under the rule: the negative number of cards remains for losers and winner gain all those scores. Next button for next game is at the bottom-right corner and top-left corner is back button which goes back to the main menu.




**Draft 7: Leaderboard Design**

A Leaderboard is a place for users to check the highest scores. This leaderboard will be in the middle which shows the top 3 scores in descending order and the lowest score. Reset data of leaderboard can be done by pushing restart button at the bottom-right corner.




**Draft 8: Profile Selection Design**

Profile selection is a place for users to use different player's name for playing, which is useful when the game is given to different people to play. There are 3 buttons for accessing the player profile at centre. The delete button is at the bottom-right corner is used to toggle load and delete profiles. Back button is at the top-left corner to return the previous page. The profile of the player includes the current score and the highest score reached.

Users can directly touch the object on a screen and interact with them which makes them much easier to use by simply tapping simple meaning objects. This user interface can allow users to play with fewer skills. Using GUI not only giving better user interface but also a better user experience. This provides user to play the game much more efficiently and attract them by using colourful graphics.

## 8. Non-Graphical User Interface

Other than the graphical user interface, Big2 can provide alternative pathways for users to interact with iPhones and iPads. There are 2 other interfaces mainly, they are audio and motion.

In the part of audio, users can listen audio via speakers of the devices which makes them easily catch up the game situation quickly. Big2 will play sound effects when certain events are triggered. For instance, card flipping sound effect will be added when distributing cards or victory sound effect will be added when the player wins the game, which makes users more enthusiastic in playing the game. Sound effects can simulate the reality of playing big two. Creating illusion and controlling users mood, sound effects is crucial to provide a delightful user experience and making the game much more vividly.

Furthermore, background music can be played continuously. Music will allow users to relax and more enjoyable during gaming. With only graphical user interface, players can get easily bored as the interacting elements are not changing. However, music provides continuous movement in the game and keep players attracted. This contributes to providing a comfort user experience during gameplay.

Motion interface let users feel the situation via the vibrator of the devices which allows them being alerted when certain events happen when there is no audio output. For example, vibration will be taken when the game takes to the turn of player. This makes users focus on the game and enjoy the game moment. The vibration will become imperative when the game exhibits with the timer or online function.

## 9. Process Planning

**Table 9: Gantt Chart**

| | 2015 | | 2016 | | | | | | | | | | | | 2017 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Nov | Dec | Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec | Jan |
| Analysis | | | | | | | | | | | | | | | |
| Design | | | | | | | | | | | | | | | |
| Implementation | | | | | | | | | | | | | | | |
| Testing & Debug | | | | | | | | | | | | | | | |
| Evaluation | | | | | | | | | | | | | | | |
| Documentation | | | | | | | | | | | | | | | |

The project is started from November 2015 to January 2017. The steps involved are analysis, design, implementation, testing & debug, evaluation and documentation.

**Task Distribution**

| | Task | Earliest Start | Length | Preceded by |
|---|---|---|---|---|
| 1 | Interview users | Nov 15 | 1 | |
| 2 | Observation of game | Nov 15 | 1 | |
| 3 | Analysis of game | Nov 15 | 1 | |
| 4 | Setup Objectives | Nov 15 | 1 | 1, 2, 3 |
| 5 | Selection of Programming Language | Dec 15 | 1 | 4 |
| 6 | Selection of OS/Hardware | Dec 15 | 1 | 4 |
| 7 | Design of Data Structure | Dec 15 | 1 | 4 |
| 8 | Design of Game Flow | Dec 15 | 2 | 4 |
| 9 | Design of User Interface | Dec 15 | 1 | 4 |
| 10 | Programming (Basic System) | Feb 16 | 1 | 5, 6, 7, 8 |
| 11 | Programming (User Interface) | Feb 16 | 5 | 9, 10 |
| 12 | Programming (System Development) | Feb 16 | 6 | 10 |
| 13 | Debugging of System | Mar 16 | 6 | 10 |
| 14 | Testing of System | Mar 16 | 2 | 10, 11, 12 |
| 15 | Evaluation | Nov 16 | 1 | 14 |
| 16 | Documentation | Nov 15 | 10 | |

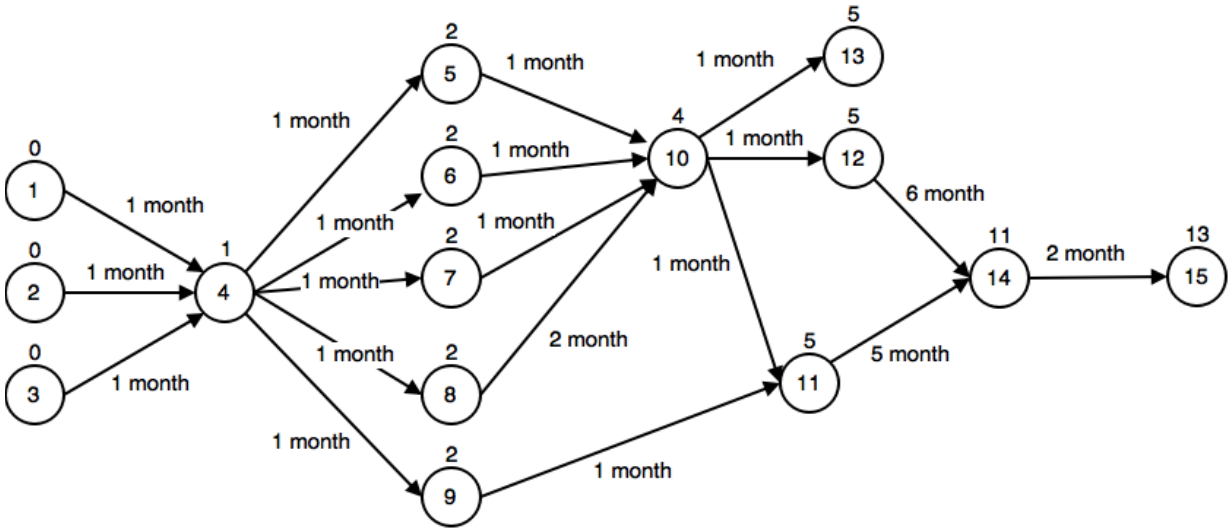


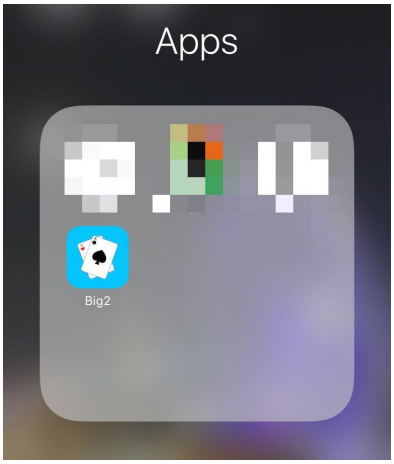

**Diagram 7: Critical Path Diagram**

The above diagram shows the critical path of this project development and the earliest start time of last task 15 is 13 months (December 2016) and the project was expected to end in January 2017. The time schedule is closely packed which dealt may lead to insufficient of time for complete some part of Big2.

By following the schedule of the project, it helps me to monitor the progress during the project period. Few tasks are critical, like task 4 setup objectives, task 8 design of game flow, task 10 programming (basic system), task 12 programming (system development) and task 14 testing of the system. These tasks are critical which can seriously affect the overall project schedule. In order to finish the project on time completely, I will more aware of the time used in these tasks.

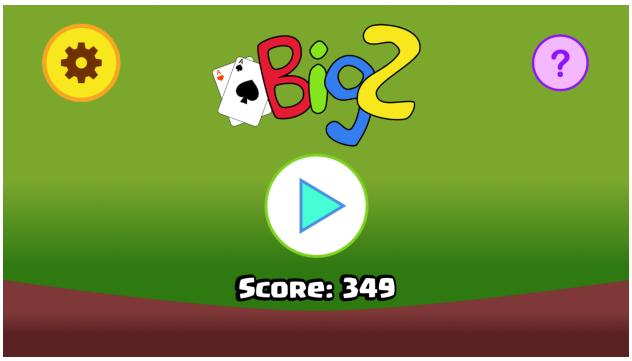
# D. Implementation

## 1. Flow of Big2

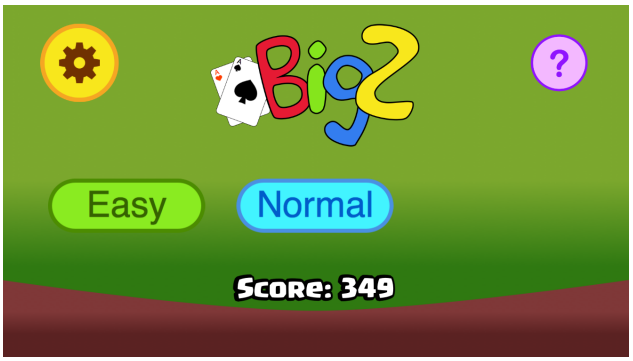



**Screenshot 2: App Icon**

Big2 has an app icon which allows users easily distinguish and remember it. Once users tap on the icon of Big2, the application launches. Big2 can be removed by long pressing app icon, pressing the cross and confirming removal.

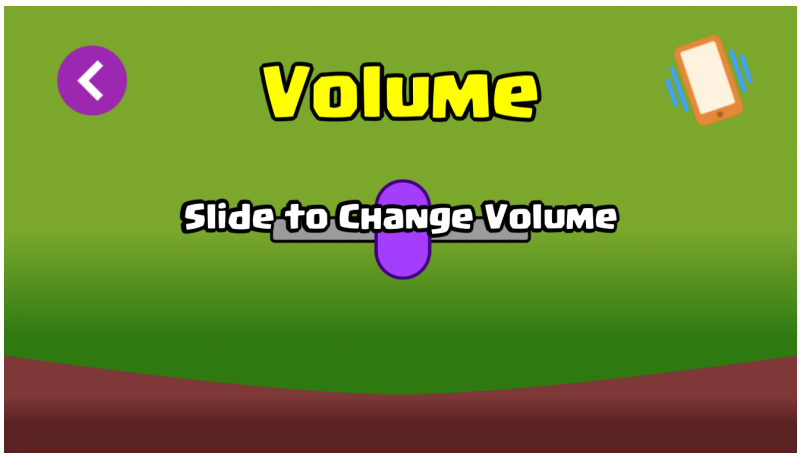



**Screenshot 3: Main Menu Page**



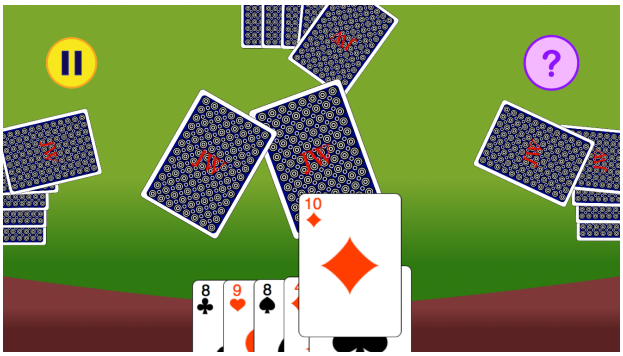

**Screenshot 4: Selection of Difficulty**

This is the Main Menu of Big2 with a background design similar to gambling desk. Logo of the game is right at the upper centre of the screen which allows users knowing the game directly. Current score of the player is shown at the bottom. Setting button is on the top-left corner and allow users to configure the setting by pushing the setting button. Players can tap the start button to start the game and choose the difficulty of computer artificial intelligence. Tapping difficulty buttons, like “Easy”, will set the difficulty and start the game. Players can also read instructions by tapping the button with a question mark on the top-right corner, details will be discussed later.




**Screenshot 5: Setting Page**

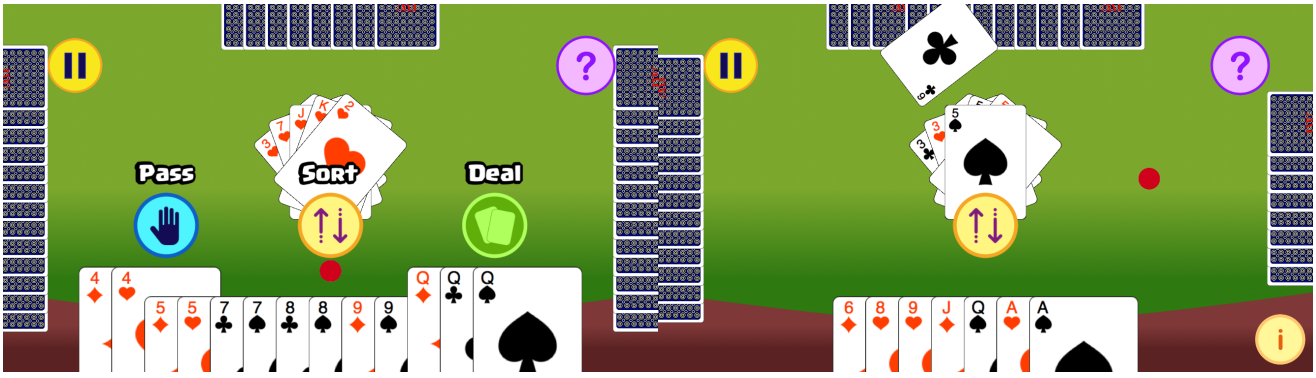
After pressing the setting button in the main menu, setting scene will be transited. The volume slider is at the centre and users can change the volume of audio by sliding left and right, where left is the smallest volume and right is the largest volume. The vibration button at the top-right corner can be toggled and enable or disable the vibration made by tapping the button directly. The back operation button is at the top-left corner which ables users to return main menu page by tapping the back button. There will be a reminder message to instruct players slide to adjust the volume.



**Screenshot 6: Distributing Cards**


**Screenshot 7: Reshuffling Cards**

When users start the game, the scene will transit to game scene. The game starts with the animation of distributing cards from the deck and every player gets 13 cards. Whenever any player has less than 3 cards with rank greater than or equal to J, Q, K, A and 2, the game will automatically collect all the cards back from players, reshuffle the deck and redistribute cards until the condition above is satisfied.




**Screenshot 8: Game Progress**

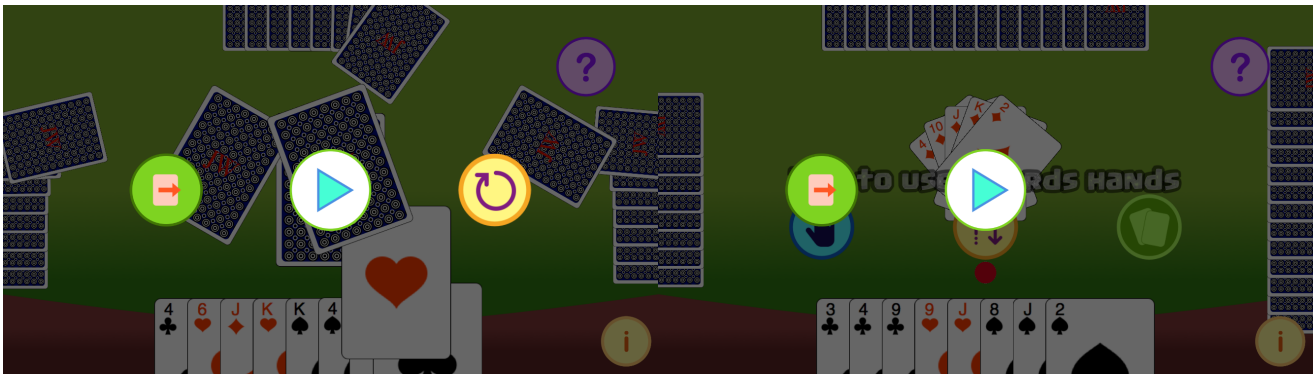
Once the distribution of cards is done, the game will start from the person who gets the diamond 3, which is the smallest rank and suit in the deck. Players are taking turns in the clockwise direction with the indication of the red dot. The pass and the deal button will show up when it is the turn of player. Users can tap on the cards to toggle a card to be played or not, where the cards moved upward is to be played. Once player made his decision, he can push pass or deal button to submit the placement of card. The sort button is designed for toggling the sorting of the cards in either by rank or by suit. The details button, on the bottom-right corner, is to show the button information for the game. The pause button, on the top-left corner, is to pause the game. Players can read instructions by tapping the button with a question mark on the top-right corner.


**Screenshot 9: Alert User**

Each player needs to deal a combination of cards greater than the top of the pool unless the player is leading. When the hand is not in a correct combination or smaller than the hand of the top of the pool, an alert will be displayed to remind the user to play the hand correctly.


**Screenshot 10: Tips**

For every 30 seconds, a random tip will be pop up at the centre of the screen to remind players that there is different strategies or underlying rules of the game. These tips can help player resolve confusion in the game.



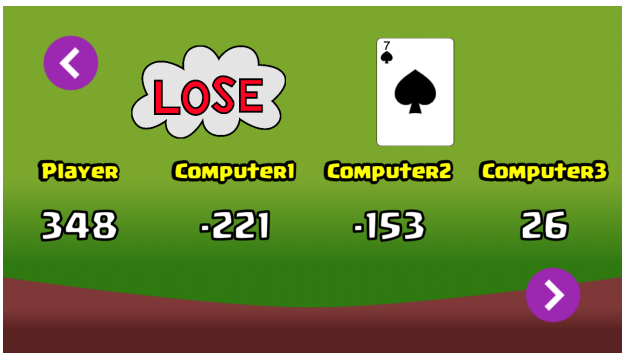
**Screenshot 11: Pause Game**

When the pause button is pushed, all animations will be paused and show up pause interface. The play button at the centre resumes the game. Players can restart the game by tapping restart button on the right side. However, once every player have played a hand, it will not allow players to restart the game. Players can also quit game and back to the main menu by pushing the button on the left side. Whenever players deactivate the game, for example, leaving application by tapping the home button, calling Siri or interrupted by incoming calls, the game will automatically pause. After that, when players activate the game, pause interface will appear.


**Screenshot 12: How To Play**

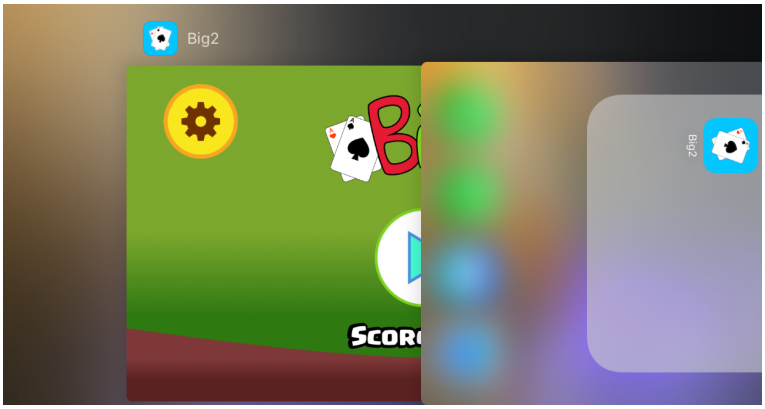
Instruction page will be showed whenever player tapped the how to play button, both in menu and game scene. There is a total of 16 detail instruction pages arranged in decreasing importance. Players who know how to play may read a few page or skip it. These pages are designated for players who do not know or have never played Big Two. There are back and next button which allows players to switch from instruction pages. Players can close the page by tapping close button on top-right corner. When the page is shown up, the game will also pause.


**Screenshot 13: Result**




**Screenshot 14: Score Change**

After any one of the players remains no cards, the game will end and showing do the player wins. Afterwards, there is a calculation in the change of scores among players and showing the latest result. Users can choose to continue the game by tapping next button at the bottom-right corner or back to the main menu by pushing back button at the top-left corner.




**Screenshot 15: Terminate Application**

When users want to exit the game, they can press the home button to leave or terminate the application by double tapping at the home button to reach the multitasking scene. Users can slide the application away at this time to terminate application completely.

## 2. Features

Big2 is an iOS game application and having different features. These features can greatly increase the user experience and completeness of the game.

User-Friendly Interaction

All scenes are inputted from the touch screen, the pointing device. Users can choose their desired options tapping directly on the object, which is more enjoyable than using a keyboard to control. This also reduces the skill needed to learn for playing Big2.

User-Friendly Interface

All scenes are carefully designed to fully utilise the screen and provide fancy graphics for users. The interface is also designed to show details of the game. Volume slider in setting can show the exact ratio of the volume output, vibration button in setting can show the status of the vibrator. Cards in players hand are automatically adjusted to the corresponding position whenever there are changes. Cards in the pool will rotate to show the whole hand when there is more than 1 card in a hand. Toggling of cards displays what cards will be played by a simple way. The red dot in taking turn point the current player for users to follow. This allows users to read a lot of information with a simple way and near the way how player observe in the game of big two in reality.

Score Calculation

When a game round ends, the current score will be calculated for the user and accumulate the result. This help player understands how well they play Big2 continuously. More, score in the game is the indication of achievement which can be used to compare with friends. Players who remain card in his hand will deduct scores and winner will get all scores from other players. The score is calculated and weighted by following rules.

**Table 11: Score Calculation Weighting**

| Number of Cards Remains | Each Card Weights |
|---:|---|
| less than 10 | 1 |
| 10 to 12 | 2 |
| 13 | 3 |

Automatic Sorting of Cards

The cards in player's hand are automatically sorted according to the value of the card, determined by its rank and suit, and show in ascending order from left to right. The game also allows sorting by suit. This organises the cards well and allows users to find cards and manage their decision in each round easily.

Simulation of Real Shuffling

In each round, cards are randomly shuffled by overhand or riffle. The shuffling function in the deck is simulating how people shuffle cards in real life. This contributes to the randomness of card distribution based on real life gameplay.

Automatic Reshuffling of Cards

Whenever any player gets disadvantage of having less than 3 J, Q, K, A and 2, automatic reshuffling of cards will be performed to avoid unfair gaming in Big2. As a small number of high-value cards can cause a huge disadvantage in the game, the decision of reshuffling is needed to ensure fairness of the game.

Automatic Pause when Inactive

Players will not stay in the application for a long time and sometimes requires switching between application. When the game is active again, it will not directly start the game but pause to provide more time ready for playing.

Audio and Motion Interaction

There are some sound effects and background music as the audio interaction with the user. Background music will be played when the start of the application. Sound effects are played during movement of cards which allow players to enjoy the moment of playing Big2 by simulating the reality. This also contributes to bringing the player to enthusiastic in the game.

Vibration is the motion interaction with the user in order to bring dynamic to the game. When the player is taken to next turn, a vibration of the device will be performed and alert user. With both of these interactions, the game will be more vivid. The setting of the game allows users to change the volume of the audio and toggle the availability of vibration.

## 3. Interesting Algorithm

There are few interesting algorithms in Big2 and I will select some of these to introduce here.

```
Game.startGame()
```

```
func startGame() throws {

    deck.shuffling()
    /// Check number of players
    /// Generate Tips

    func distributeNext(_ playerIndex: Int) {


        if deck.Cards.count == 0 {
                /// Check disadvantagedness
            for player in players {
                /// Count number of cards in J/Q/K/A/2
                if count < 3 {
                    /// Reshuffling and return
                }
            }
            for player in players {
                /// Search first player
                /// Sort cards and final prepare game start
                do {
                    try computerMove()
                } catch {}
                return
            }

        }

        let distribute = SKAction.run({ () in
            /// Move card from deck to player
            /// Distribute cards to next player
        })

    }

    distributeNext(0)

}
```

This part is the `startGame()` function in `Game` class. As there is a need to delay for the animation of distributing cards, distribution to players have to be used in a closure. The usual way of implementation will not be able to facilitate the delay function due to thread problem. As a result, I used recursion and convert closure to `SKAction` so as to put the distribution function in a closure and repeatedly distribute to next player. Moreover, once distribution is done, it will automatically check the need of reshuffling and final initialise before the game starts.

Pool.checkHand()

```
internal func checkHand(_ hand: inout Hand, leading: Bool) throws {

    func checkSame(_ cards: [Card], value: Value) throws -> Bool {
        /// Check number of cards & value
        /// Get first comparing value
        /// Compare with comparing value
        /// Return are the card values all same
    }
    func checkContinuous(_ cards: [Card]) throws -> Bool {
        /// Check number of cards
        /// Check is the card rank is greater than the previous by 1
        /// Return are the cards in continuous
    }
    func checkStraight() throws -> Bool {
        /// Check number of cards
        if valid && cards.first!.rank == "J" {
            /// General check continuous except JQKA2
        } else if !valid {
            if valid && cards.first?.rank == "3" && cards.last?.rank == "2" {
                /// Special check continuous for A2345 23456
            }
        }
        /// Set comparing card
        /// Return are the cards in Straight
    }
    func checkFullHouse() throws -> Bool {
        /// Check number of cards
        /// Check Full House 3/2 & 2/3 combination
        /// Return are the cards in Full House
    }
    func checkFourOfAKind() throws -> Bool {
        /// Check number of cards
        /// Check Quad 4/1 & 1/4 combination
        /// Return are the cards in Four Of A Kind
    }
    func checkDuplicate() -> Bool {
            /// Initialize used table
            for card in hand.Cards {
                /// Mark used
            }
            /// Return any card duplicated
        }
    /// Check type of hand can be played
    switch hand.Cards.count {
        case 1 ... 3:  /// Check are the cards rank same
        case 5:
            /// Check Straight Flush/ Flush
            /// Check Straight/ Full House/ Four Of A Kind
            /// Check validity of hand card combination
            /// Check power value beats hand in pool
        default: /// Invalid hand card combination
    }
}
```

Another interesting algorithm is `checkHand()` function in `Pool` class. Checking validity of hands with comparing to the pool is a complicated and crucial process, as this part builds up the main rule of Big2. In this part, I adopted the divide and conquer to split up a number of sub-problems, for instance, checking same value and checking continuous rank. All checking of the different five-card-hands process will involve using the two base functions several times. After these smaller modules are solved, the largest module of checking hand can be done.

Deck.shuffling()

```
internal func shuffling() {

    /// Random number of times shuffle
    for _ in 1 ... to {
        /// Random mode for shuffling
        switch mode {
            case 0:       ///Overhand
                for _ in 1 ... 3 {
                    /// Cut cards and rejoin
                }
            case 1:       ///Riffle
                /// Cut two decks
                while firstDeck.isEmpty == false || secondDeck.isEmpty == false {
                    /// Random merge cards
                }
            default: break
        }
    }
}
```

`shuffling()` function in `Deck` class simulates the real shuffling in reality and contributes to the randomness of cards in nature. In riffle shuffling, I used the technique similar to merge sort to merge two decks, but with randomisation.

Deck.returnCards()

```
func returnCards(_ pool: Pool?, players: [Computer]) throws {

    /// Check number of players
    /// Collect cards from pool

    for player in players {
        if player.Cards.count > 0 {
            /// Record change of score for losers with weightings
            /// Add up change of score for winner
        }
    }

    /// Record change of score for winner
    /// Revert situation of reshuffling close game
    /// Collect cards from players
    /// Reverse order of card
    /// Check number of cards

}
```

It is interesting in the algorithm of `returnCards()` function in `Deck` class. During calculation of score, I record the change of score instead of directly changing the score. Change of score is performed in `GameoverScene` so as to animate the change of score. This also helps players to understand how well they play in this round in a succinct way.

Computer.dealCards()

```
internal func dealCards(_ moves: [Hand?], prevention: Bool) -> Hand? {
        func getSameRankCards(greaterThan fix: Card = Card(rank: "3", suit: "Diamond"),
                    numberOfCard max: Int? = nil, allCards: Bool = false) -> [Card]? {
            for card in Cards {
                /// Count same rank
                /// Return [Card] if requirement satisfied
                /// Reset counter

            }
            /// Return no [Card] found
        }
        func searchFiveCardHand(greaterThan fix: Hand? = nil) -> Hand? {
            func checkFullHouse() -> Hand? {
                /// Compare 5-cards hand value
                /// Search full house
            }
            func checkFourOfAKind() -> Hand? {
                /// Compare 5-cards hand value
                /// Search four of a kind
            }
            func checkStraightFlush() -> Hand? {
                /// Compare 5-cards hand value
                /// Search straight flush
            }
            func checkStraight() -> Hand? {
                /// Compare 5-cards hand value
                /// Search straight
            }
            func checkFlush() -> Hand? {
                /// Compare 5-cards hand value
                /// Search flush
            }

            /// Manage data required
            /// Check 5-cards hand availability & return if found
            /// Return no hand found
        }
        func easyAI() -> Hand? {

            /// Prevention required action
            /// Leading action
            /// General action
            /// Return no hand found
        }
        /// Return no card can played
        /// Get pool top hand
        switch mode {
        case .normal :              /// Normal AI
            /// Search 5-cards hand
            /// Easy AI algorithm
        case .easy :                /// Easy AI
        default: break
        }
        /// Return no card can played
    }
```

`dealCards()` function in `Computer` class is the crux of building artificial intelligence in dealing cards. During returning hand, certain conditions are needed, like, is prevention needed or the computer leading. After defining the situation, AI will choose the correct algorithm towards the game and return the corresponding move.

## 4. Details of Big2 Implementation

In order to allow other developers easily understand the system of Big2, this section is particular for them to read. In the implementation of Big2, I made use of the modularity of Swift to save time from re-writing algorithm and organised different modules, which helps for easier testing and debugging. Furthermore, some classes are inherited from other so as to simplify the system control and share useful methods and attributes at the same time but with characteristics of polymorphism.

MenuScene.swift

```swift
class MenuScene: SKScene {

    override func didMove(to view: SKView) {
        /// Called when the scene is transited to MenuScene
        /// Initialize menu view
    }

    override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
        /// Called when a touch ends
        for touch in touches {
            /// Confirm button pushed
            /// How to play actions
            /// Start game actions
        }
    }

}
```

`MenuScene` is the class inherited from `SKScene` and process the main menu view. This scene mainly manages the transition of scenes and touches control on the touch screen and convert to proper input of the system.

GameScene.swift

```
class GameScene: SKScene {

    var sortBy: Value = .rank               /// Sorting value

    func gameView() {
        /// Initialize the game view
    }
    override func didMove(to view: SKView) {
        /// Called when the scene is transited to GameScene
        /// Call game view
    }
    override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
        /// Called when a touch ends
        for touch in touches {
            /// Confirm button pushed
            /// Toggle cards to be played
            if isPaused {
                /// Actions when paused
            }
            /// How to play actions
            /// Pause game
            /// Show details
            /// Sort cards
            /// Confirm decision
        }
    }

}
```

GameScene is the class inherited from SKScene and process the game view. This scene mainly manages the Game class operations with touches input and convert to desired input of the system. This class is also imperative to interact with other classes and intersection point in the game view.

GameOverScene.swift

```
class GameoverScene: SKScene {

    func saveProgress() {
        /// Save players' scores
    }
    override func didMove(to view: SKView) {
        /// Called when the scene is transited to GameOverScene
        /// Initialize game over view
    }
    override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
        /// Called when a touch ends
        for touch in touches {
            /// Confirm button pushed
        }
    }

}
```

GameOverScene is the class inherited from SKScene and process the end game view. This scene mainly manages the transition of scenes to GameScene or MenuScene.

SettingScene.swift

```
class SettingScene: SKScene {

    override func didMove(to view: SKView) {
        /// Called when the scene is transited to SettingScene
        /// Initialize setting view
    }
    override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
        /// Called when a touch ends
        for touch in touches {
            /// Confirm button pushed
            /// Save setting changed
        }
    }
    override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        /// Called when a touch moves
        for touch in touches {
            /// Confirm slider movement
            /// Change volume
        }
    }

}
```

SettingScene is the class inherited from SKScene and process the setting view. This scene mainly manages volume and vibrate control via touches and facilitate frame of the setting. The setting will be saved to internal storage for further use even the game has terminated.

Game.swift

```
class Game {

    var deck = Deck()                       /// Deck of the game
    var pool = Pool()                       /// Pool of the game
    var players = [Computer]()              /// Players of the game
    var currentPlayer: Int? = nil           /// Current player pointer
    var leadingPlayer: Int? = nil           /// Leading player pointer
    var mode = Mode()                       /// Game mode
    var endGame: Bool = false               /// End game flag
    var move: Int = 0                       /// Count moves
    var moves = [Hand?]()                   /// Last 3 moves
    var lastHand: Hand?                     /// Last hand of game

    init() {
        /// Initialize players
        /// Initialize moves
    }
    func initGame(_ scene: GameScene) {
        /// Initialize a game
    }
```

```
    func startGame() throws {
        /// Throw: invalid number of players
        /// Generate Tips
        /// Card shuffling
        func distributeNext(_ playerIndex: Int) {
            /// Check disadvantagedness
            /// Search first player
            /// Sort players' cards
            /// Visualize pointer
            /// Initiate first play
            /// Distribute next card
        }
        /// Card distribution
    }
    func placeCard(_ player: Computer) throws {
        /// Throw: leading player do not play
        ///        no hand is dealt
        func checkOnHand(_ hand: Hand, player: Computer) throws {
            /// Throw: invalid card
            /// Check hand placed is on player's hand
        }
        func removeOnHand(_ hand: Hand, player: Computer) {
            /// Remove cards in player's hand
        }
        defer {
            /// Visualize cards
        }
        /// Get player's cards for deal
        /// Check is hand valid
        /// Accept valid hand
        /// Check is player pass
    }
    func endTurn() {
        /// End turn
        /// Check end game
        /// Initiate next turn
    }
    func computerMove() throws {
        /// Throw: invalid current player
        ///        invalid leading player
        /// Toggle player's buttons
        /// Delay & place card
    }
    func checkEndGame() -> Bool {
        /// Return: is game ended
        /// Check end game
    }
    func closeGame() {
        /// Mark last hand
        /// Close game
    }

}
```

Game is the class for controlling the whole gaming system, including procedures, take turns, play card and end game. This class interact with all elements related to the system of the game and manage the procedure of game by providing a platform for other classes.

Card.swift

```
class Card : SKSpriteNode {

    var rank = String()                     /// Rank of card
    var suit = String()                     /// Suit of card
    var deal = Bool()                       /// Is card deal
    var faceup = Bool()                     /// Is card face up
    var cardFace = SKTexture()              /// Texture of card face
    var cardBack = SKTexture()              /// Texture of card back

    convenience init (rank: String, suit: String) {
        /// Initialize card
    }
    internal func initCardSprite() {
        /// Initialize card for start of game
    }
    internal func printCard() {
        /// Print card
    }
    internal func flip() {
        /// Flip face of card
    }
    internal func toggle() {
        /// Toggle placement of card
    }

}
```

`Card` is the class inherited from `SKSpriteNode` which is the representation of a physical card. This is one of the major component of game system Big2 as this is the game all about cards. This transforms to a physical card by inheriting node and attach extra attributes to characterise a card.

Hand.swift

```
class Hand {

    var Cards = [Card]()                    /// Cards of a hand
    var powerValue: String?                 /// 5-cards power value
    var comparingCard: Card?                /// Card of comparison
    var playedBy: String = ""               /// Name of player dealt the hand

    internal func printHand() {
        /// Print hand
    }
    internal func handVisualize(_ pos: CGPoint, zPosition: CGFloat, duration:
TimeInterval) {
        /// Visualize hand at a position
    }
}
```

`Hand` is the class represent a hand of cards. The major component is an array of `Cards` and who played the hand. This is also a unit in the pool which is used for playing. In each hand, there is a comparing card which is used for comparing with other hands in pool.

Deck.swift

```
class Deck {

    var Cards = [Card]()                    /// Cards in deck

    init() {
        /// Initialize a deck
    }
    internal func printDeck() {
        /// Print deck
    }
    internal func shuffling() {
        /// Random shuffling by overhand and riffle
    }
    func returnCards(_ pool: Pool?, players: [Computer]) throws {
        /// Throw: invalid number of players
        ///        invalid number of cards
        /// Collect cards from pool and players
        /// Calculate score change of players
        /// Reverse order of cards
    }

}
```

`Deck` is the class for controlling a full set of poker cards. This represents the dealer of the game and manages the cards before and after playing Big2. It is formed by an array of `Cards` with special methods to characterise as a deck.

Pool.swift

```
class Pool {

    var Hands = [Hand]()                    /// Hands of pool

    internal func checkHand(_ hand: inout Hand, leading: Bool) throws {
        /// Throw: number of cards not equal to pool
        ///        invalid card
        ///        invalid hand card combination
        ///        hand value smaller than in pool
        func handValueSmallerThanInPool() -> Error {
            /// Return: hand value smaller than in pool
            /// Print hand and hand in top of pool
        }
        func checkSame(_ cards: [Card], value: Value) throws -> Bool {
            /// Throw: invalid number of cards
            ///        invalid value
            /// Return: is all cards in same value
        }
        func checkContinuous(_ cards: [Card]) throws -> Bool {
            /// Throw: invalid number of cards
            /// Return: is the cards in continuous order
        }
        func checkStraight() throws -> Bool {
            /// Throw: invalid number of cards
            /// Return: is the hand a straight
        }
        func checkFullHouse() throws -> Bool {
            /// Throw: invalid number of cards
            /// Return: is the hand a full house
        }
```

```
        func checkFourOfAKind() throws -> Bool {
            /// Throw: invalid number of cards
            /// Return: is the hand a four of a kind
        }
        func checkDuplicate() -> Bool {
            /// Return: is card duplicated
        }
        /// Check validity of the hand corresponding to number of cards
}
```

`Pool` is the class managing all hand dealt. This class mainly act as the storage of hands dealt and checking the hands inputted during the game. Any invalidity or error will result when hand tried to input is incorrect. This class also build up the rule of big two and is the crux of the game system.

Player.swift

```
class Computer: Equatable {

    var Cards = [Card]()                        /// Cards of player
    var playerName = String()                   /// Name of player
    var mode = Mode()                           /// Mode of AI
    var position = CGPoint()                    /// Position of player
    var direction = Direction()                 /// Direction of cards
    var scoreLabel = SKOutlinedLabelNode()      /// Score label of player
    var score = Int()                           /// Score of player
    var changeScore: Int = 0                    /// Change of score of player
    var pass: Bool = false                      /// Is player pass

    init(name: String) {
        /// Initialize computer
    }
    internal func initPlayer(_ position: CGPoint) {
        /// Initialize computer for start of game
    }
    internal func printCards() {
        /// Print cards in computer's hand
    }
    internal func dealCards(_ moves: [Hand?], prevention: Bool) -> Hand? {
        /// Return: hand to be played

        func getSameRankCards(greaterThan fix: Card = Card(rank: "3", suit: "Diamond"),
                    numberOfCard max: Int? = nil, allCards: Bool = false) -> [Card]? {
            /// Return: array of cards having same rank
        }
        func searchFiveCardHand(greaterThan fix: Hand? = nil) -> Hand? {
            /// Return: 5-cards hand available
            func checkFullHouse() -> Hand? {
                /// Return: full house available
            }
            func checkFourOfAKind() -> Hand? {
                /// Return: four of a kind available
            }
            func checkStraightFlush() -> Hand? {
                /// Return: straight flush available
            }
            func checkStraight() -> Hand? {
                /// Return: straight available
            }
            func checkFlush() -> Hand? {
                /// Return: flush available
            }
            /// Manage data required
            /// Check 5-cards hand availability
        }
```

```
        func easyAI() -> Hand? {
            /// Return: hand using easy AI

            /// Prevention required action
            /// Leading action
            /// General action
        }
        /// Get pool top hand
        switch mode {
        case .normal :              /// Normal AI
        case .easy :                /// Easy AI
        }
    }
    internal func cardVisualize() {
        /// Visualize cards
    }

}

class Player: Computer {

    override func initPlayer(_ position: CGPoint) {
        /// Initialize player for start of game
    }
    override func dealCards(_ moves: [Hand?], prevention: Bool) -> Hand? {
        /// Return: hand to be played
        /// Get cards to be played

    }
    override func cardVisualize() {
        /// Visualize cards
    }

}
```

`Computer` is the class for artificial intelligence of the game. `Player` is the class inherited from `Computer` to customise for the player. Both classes have its own characteristics and similarities as the polymorphism goes. These classes help the configuration of card placement and score changes.

Function.swift

```
func randomInt(min: Int, max: Int) -> Int {
    /// Return: random Integer within a range
}
func degreeToRadian(degree: Double) -> Double {
    /// Return: radian from degree
}
func == (lhs: Computer, rhs: Computer) -> Bool {
    /// Return: computer equality
}
func == (lhs: Card, rhs: Card) -> Bool {
    /// Return: card equality
}
func < (lhs: Card, rhs: Card) -> Bool {
    /// Return: comparing of card (less than)
}
func > (lhs: Card, rhs: Card) -> Bool {
    /// Return: comparing of card (greater than)
}
```

```
func >= (lhs: Card, rhs: Card) -> Bool {
    /// Return: comparing of card (greater than or equal to)
}
func <= (lhs: Card, rhs: Card) -> Bool {
    /// Return: comparing of card (less than or equal to)
}
func delay(_ delay:Double, closure:@escaping ()->()) {
    /// Delay closure
}
```

`Function.swift` gathers all functions that will use in almost all classes and the functions which are general.

```
Extension.swift
extension Bool {

    mutating func toggle() {
        /// Toggle the value of Boolean expression and return its value.
    }

}

extension Array where Element: Card {

    fileprivate func isOrderByRank(_ card1: Card, card2: Card) -> Bool {
        /// Return: comparing of card by rank
    }
    fileprivate func isOrderBySuit(_ card1: Card, card2: Card) -> Bool {
        /// Return: comparing of card by suit
    }
    internal mutating func sortCards(by value: Value) throws {
        /// Throw: invalid value
        /// Sort cards according value
    }
    internal mutating func removeCard(_ element: Array.Iterator.Element) throws {
        /// Throws: InvalidCard
        /// Remove card
    }
    internal func printCards() {
        /// Print array of card
    }

}

extension SKScene {

    func clearScene() {
        /// Clear scene
    }
    func transitToGameoverScene() {
        /// Transit the current scene to GameoverScene
    }
    func transitToMenuScene() {
        /// Transit the current scene to MenuScene
    }
    func transitToGameScene() {
        /// Transit the current scene to GameScene
    }
    func transitToSettingScene() {
        /// Transit the current scene to SettingScene
    }
```

```
    override open func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        /// Called when a touch begins
        for touch in touches {
            /// Button animation when tapped
            /// Pause from inactive
        }
    }
}

extension SKNode {

    func addChildren(_ Nodes: [SKNode]) {
        /// Add array of SKNode as child
        for node in Nodes {
            addChild(node)
        }
    }
}
```

Extension.swift gathers all extension to the predefined classes so as to customise the classes for the use in Big2.

globalConstant.swift

```
let Rank = ["3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A", "2"]
let Suit = ["Diamond", "Club", "Heart", "Spade"]
let fiveCardHand = ["Straight", "Flush", "FullHouse", "FourOfAKind", "StraightFlush"]
let Tips = ["Lose as less as possible", "Diamond 3 play first", "Try to use 5-cards
hands"]
let totalInstructionPage = 16
let graphicsController = GraphicsController()
let audioController = AudioController()
let game = Game()

var gameScene = GameScene()
var ViewWidth = CGFloat()
var ViewHeight = CGFloat()
var vibrate = true
var isGameScene = false

enum Value {
    case rank, suit
}

enum Direction {
    case horizontal, vertical
    init() {
        /// horizontal
    }
}

enum Mode {
    case easy, normal, hard
    init() {
        /// easy
    }
}

enum error: Error {
    /// Errors
}
```

globalConstant.swift gathers all constants, variables and enumerations which all should belong to the majority of classes or the application.

GraphicsController.swift

```swift
let defaultFontName = "Supercell-Magic"                 /// Font name
let defaultFontSize = CGFloat(22)                       /// Font size

class GraphicsController {

    /// Basic sprites
    /// Menu sprites
    /// How to play sprites
    /// Gaming sprites
    /// Pause sprite
    /// Setting Sprite

    init() {
        /// Initialize
    }

    internal func initGameView(_ scene: SKScene) {
        /// Initialize game view
    }

    internal func gameStart() {
        /// Show hidden sprite
    }

    internal func updateRedDot() {
        /// Update red dot pointer position
    }

    internal func hidePlayerButtons(state: Bool) {
        /// Set player's buttons hidden state
    }

    internal func showDetails(_ scene: SKScene) {
        /// Show label of buttons
    }

    internal func removeDetails(_ scene: SKScene) {
        /// Remove label of buttons
    }

    internal func showInstructions(_ scene: SKScene) {
        /// Show instruction interface
    }

    internal func updateInstructions(_ scene: SKScene) {
        /// Update instruction page
    }

    internal func removeInstructions(_ scene: SKScene) {
        /// Remove instruction page
    }

    internal func pause() {
        /// Pause game
    }

    internal func resume() {
        /// Resume game
    }

    internal func initMenuView(_ scene: SKScene) {
        /// Initialize menu view
    }

    internal func showGameModes(state: Bool) {
        /// Set game modes hidden state
    }
```

```
    internal func initGameoverView(_ scene: SKScene) {
        /// Initialize gameover view
        /// Animate gameover scene
        /// Set score result table
        func addValue() {
            /// Animate and update score results
            /// Initiate adding next value
        }
        /// Add value
    }

    internal func reshuffling() {
        /// Add reshuffling message
    }

    internal func randomTips() {
        /// Add random tip message
    }

    internal func alertPlayCardCorrectly() {
        /// Add alert to player (play card correctly)
    }

    internal func initSettingView(_ scene: SKScene) {
        /// Initialize setting view
    }
}

class Button: SKSpriteNode {

    internal override func tap() {
        /// Perform tap animation
    }
}

class SKOutlinedLabelNode: SKLabelNode {
    /// Modified from MKOutlinedLabelNode by Mario Klavar
    /// More details in disclaimer
}

class Message: SKOutlinedLabelNode {

    internal func setDefaultLabel() {
        /// Set property of alert message
    }

    internal func fadeInFadeOut(_ scene: SKScene) {
        /// Set animation of alert message
    }
}
```

`GraphicsController` is a class for controlling a large amount of graphics and animation process so as to separate from other regular processes of the Big2 framework. It also helps initialise all scenes in graphics user interface. `Button`, `SKOutlinedLabelNode` and `Message` are classes which help to facilitate different kinds of graphics and animations. This provides a simple implementation in future when the controller builds all the things needed for graphics.

AudioController.swift

```
class AudioController {

    var volume: Float = 0.5                             /// Volume of audio
    var BGMPlayer = AVAudioPlayer()                     /// Player of background music
    var Player = SKNode()                               /// Player node

    init() {
        /// Get data from internal storage
        /// Set background music
    }
    internal func changeVolume(to: Float) {
        /// Change volume
    }
    internal func intro() {
        /// Add introduction sound effect
    }
    internal func dealCard() {
        /// Add deal card sound effect
    }
    private func addSoundEffect(fileNamed: String, duration: TimeInterval) {
        /// Change add sound effect
    }
    internal func vibrator() {
        /// Add vibration
    }
    internal func saveSetting() {
        /// Save setting
    }
}
```

`AudioController` is a class for controlling audio and motion (vibration) process. This concentrates all the function facilitate the audio output. This provides a simple implementation in future when controller builds all things needed for audio.

AppDelegate.swift

```
class AppDelegate: UIResponder, UIApplicationDelegate {

    func applicationWillResignActive(_ application: UIApplication) {
        /// Pause background music
    }

    func applicationDidBecomeActive(_ application: UIApplication) {
        /// Stop gameScene
        /// Resume background music
    }

    func applicationWillTerminate(_ application: UIApplication) {
        /// Save players' scores
        /// Save setting
    }

}
```

`AppDelegate` is a responder where methods will be called by iOS system when various events happen. This is used for saving data to internal storage and pause the current game.

# E. Testing

## 1. Test Plan

There will be two parts of testing, which are game and system. Both of them contains a number of unit tests and system tests will be evaluated is the individual units compiles together. The game is the part that from the start of a game to end, from distributing cards to determine a win or lose. The system is the remaining part that game do not include in Big2.

**Table 12: Test Plan for Game**

| 1. Game | |
|---|---|
| 1 | Game Visualisation |
| 2 | Card Distribution |
| 3 | Players & Turn Taking |
| 4 | Artificial Intelligence |
| 5 | Actions by Player |
| 6 | Hand Checking |
| 7 | Game Result (Win & Lose) |

**Table 13: Test Plan for System**

| 2. System | |
|---|---|
| 1 | Object Interaction |
| 2 | How To Play Interface |
| 3 | Pause |
| 4 | Data Storage |

# 1. Game

## 1.1 Game Visualisation

Test visualisation of game objects in every scenes and screen whether the objects show up properly and perform a particular action when tapped. This can ensure that the graphics are the basic I/O of the game system. The application will interact correctly when an input touch is received.

## 1.2 Card Distribution

Test distribution of cards whether each player has 13 cards and both pool and deck is empty. Cards should be redistributed when disadvantages exist. This ensures that the game starts correctly and no unfairness due to the cards in hand.

## 1.3 Players & Turn Taking

Test whether the game initiated correctly by finding the player who gets a diamond 3 and players will take turn accordingly. This makes sure the foundation system of the game is working in order.

## 1.4 Artificial Intelligence

Test whether computers play a hand correctly according to the game mode and required prevention as well as leading hand. This ensures the flow of the game and artificial intelligence is up to the desired standard.

## 1.5 Actions of Player

Test all possible actions of a player, including pass and deal valid and invalid hands. This ensures all actions of a player in the scope can be performed in order.

## 1.6 Hand Checking

Test hand dealt by using both valid and invalid test case. This makes sure valid hand can be dealt while invalid will be rejected.

## 1.7 Game Result (Win & Lose)

Test each player win in different situations, like the last pair. Score calculation will also be tested to ensure each player has a correct score change in a different situation. This ensures the winning or losing message display correctly and the score was calculated properly.

## 2. System

### 2.1 Object Interaction

Test whether all objects can be selected appropriately, including buttons and cards. This ensures graphics are the basic I/O of the whole system which ables users to interact with object properly. This also check whether the objects can correctly trigger certain actions

### 2.2 How To Play Interface

Test how to play interface whether show and close properly. Moving pages function will also be tested, like prevent going page beyond the pages available by using boundary cases. This ensures the instruction menu can deliver message and function properly.

### 2.3 Pause

Test all possible time for pausing. This ensures the pause function work differently when the game goes to a certain extent, like preventing a restart of the game once the first round is passed. Test some interruptions from the system, like incoming call and deactivating the application. This makes sure the next active of the application is paused accordingly.

### 2.4 Data Storage

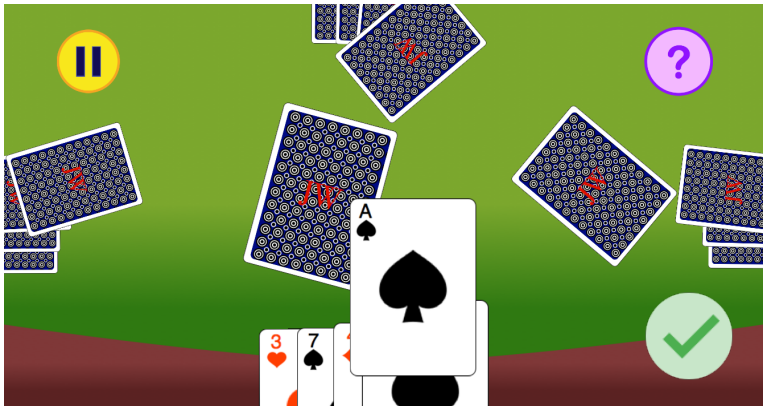
Test terminating the application in different events and observe the save of current scores and setting value. This ensures the scores can be saved and load without errors and data loss.

2. Test Result

---

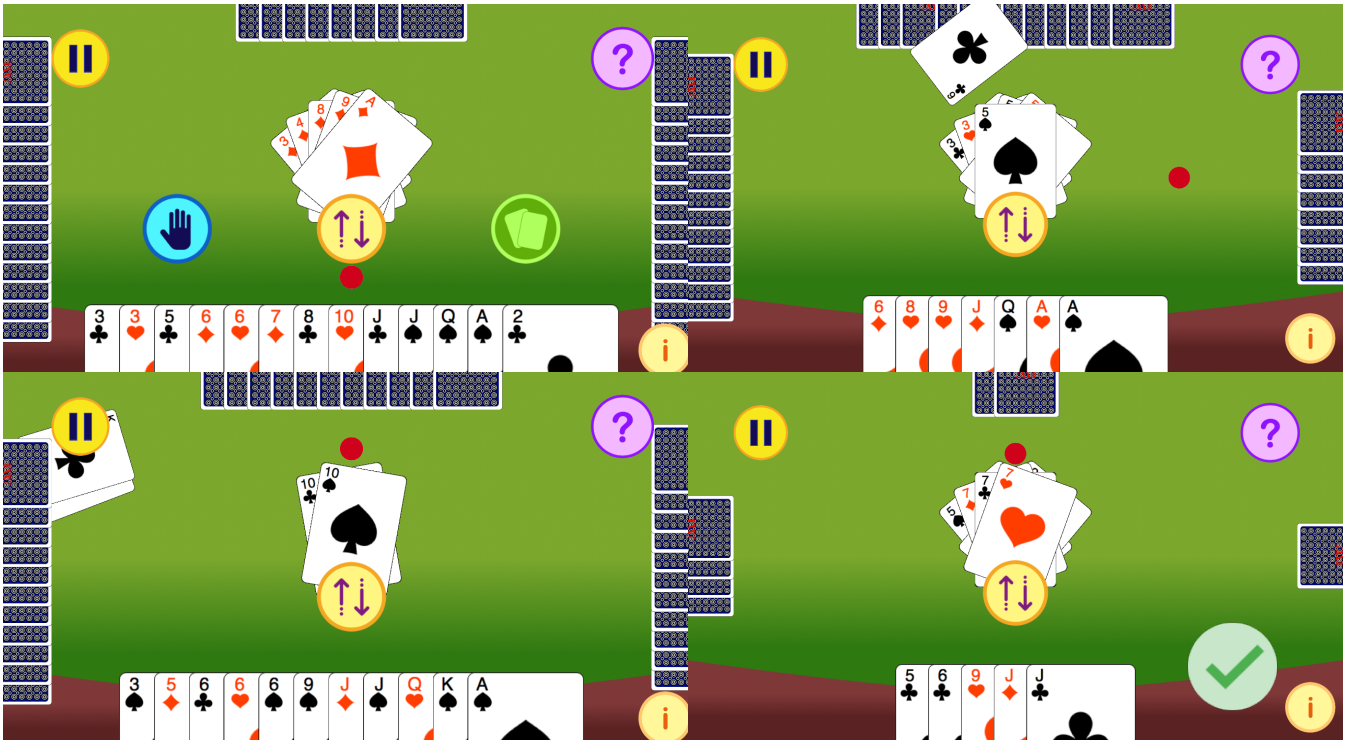
1. Game

## 1.1 Game Visualisation



**Screenshot 16: Visualisation of Distributing Cards**

The expected result is the cards belong to a player will group together in four edges. Only the hands of the player will show the face while others are faced down. Actual test results are same as expected.



**Screenshot 17: Visualisation of Game View**

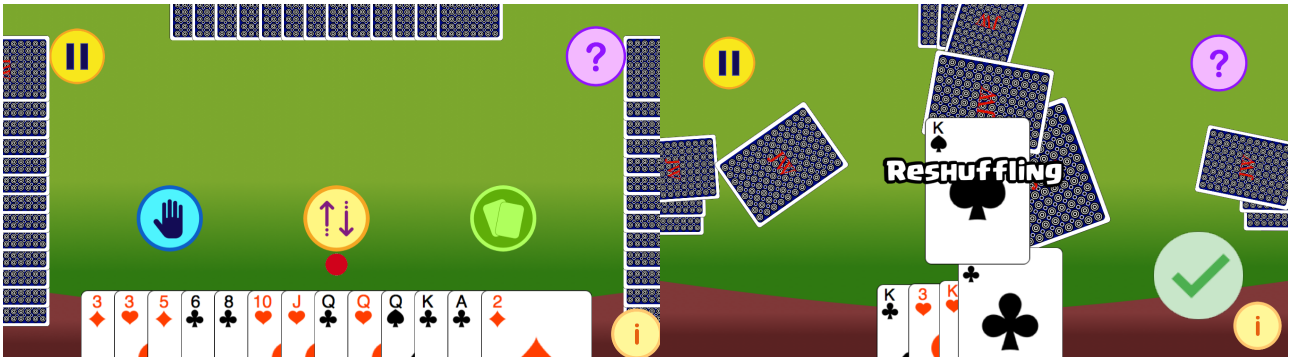
The expected result is the pass and deal button will show up only when the player is in the turn. Hand will be placed at the centre of the screen. Whenever there is more than one card the cards will rotate which allow players to see the cards clearly. In the test, single, pair, triple and 5-cards hand will be dealt to test whether rotations are correct. The results have no abnormalities.


**Screenshot 18: Visualisation of Hand in Other Position**

The result is expected to visualise any hand in all scenes and position. Single, pair, triple and 5-cards hand can all be positioned in both game scene and game over scene. Moreover, it should be able to position in any coordinates on the screen. The actual results follow the expectation.

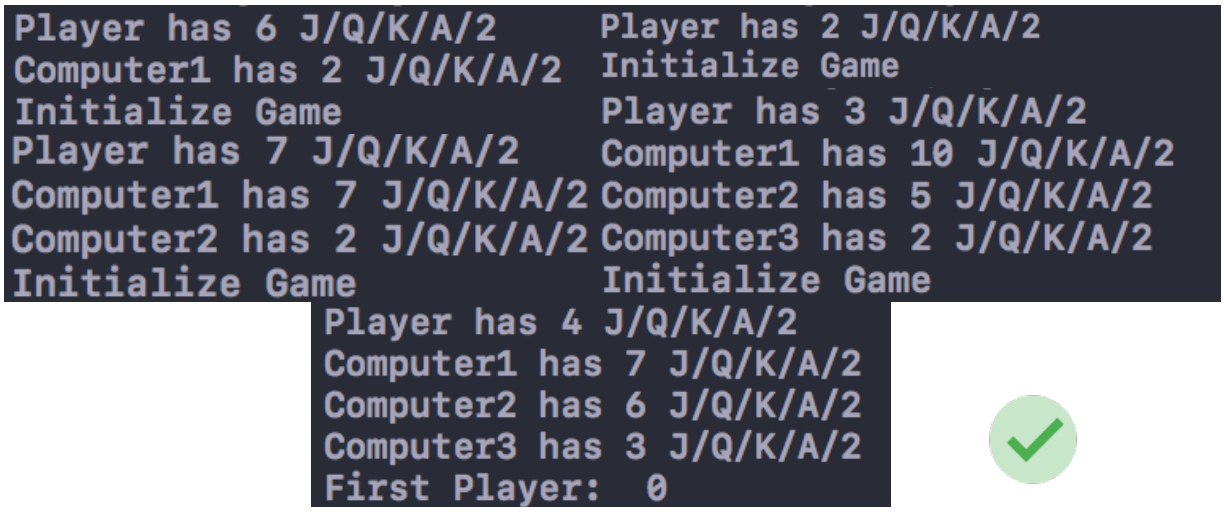
To conclude, the visualisation of the game has no abnormalities, showing the game elements in the correct position. It can be a reliable output standard of the system.

## 1.2 Card Distribution




**Screenshot 19: Card Distribution & Reshuffling**

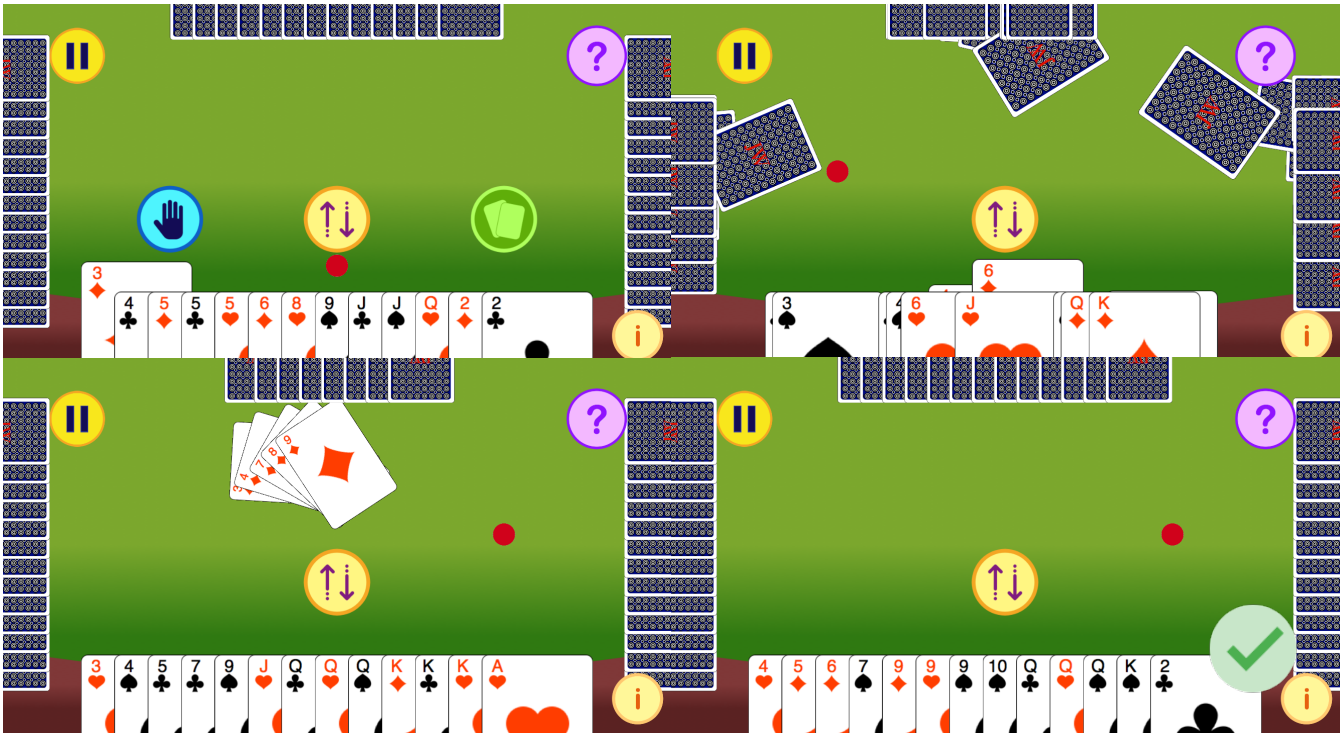
The expected result is the every player has 13 cards and once any of the players has less than 3 J/Q/K/A/2 will be reshuffled. Whenever there is 3 or more J/Q/K/A/2, cards will not be reshuffled. The actual test results are same as expected.




**Screenshot 20: Card Distribution & Reshuffling Console**

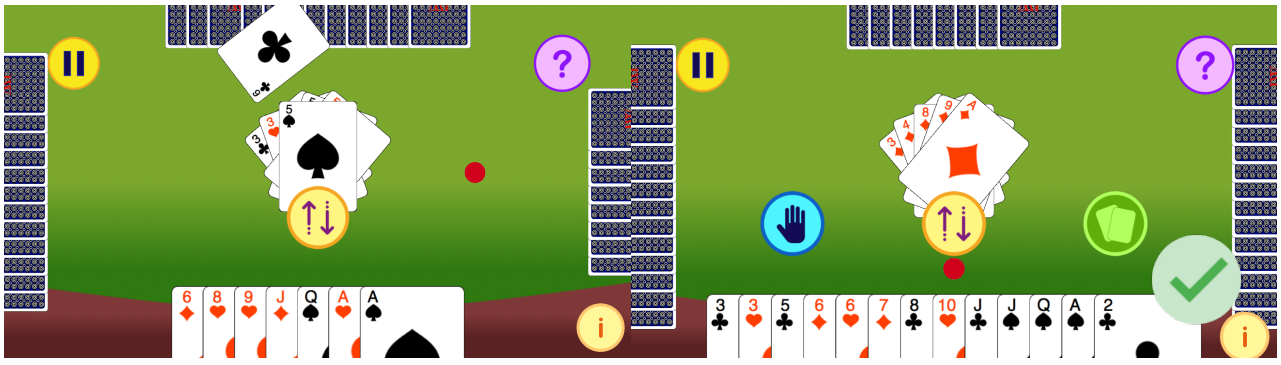
Briefly speaking, card distribution is functioning well, each player has exactly 13 cards. Disadvantagedness will not be in any players, as it is the validation to start a game.

## 1.3 Players & Turn Taking



**Screenshot 21: Starting Player**

The expected test result of a starting player is having a diamond 3 in his hand. The red dot will be placed in the first player, but the player can choose to place a diamond 3 or not. It has been tested both player and computers being the first player and the first player does have a diamond 3 in his hand.



**Screenshot 22: Turn Taking**

The expected test result of turn taking is red dot will rotate in a clockwise direction and players can play a hand in his turn. The red dot will also stop when it is player's turn and wait for player's action. The actual results follow the expectation made previously.

Generally speaking, the operation of the game system is at normal condition and found no abnormalities, players take turns in a correct order from a true first player. The game system is available for playing in a basic manner.

## 1.4 Artificial Intelligence

### 1.4.1 Easy Mode


**Screenshot 23: First Player & Play Single Hand**

The test result of a leading player should be able to play any single cards. In the easy mode, computers will play the hand having the lowest value that greater than the hand in the pool. The actual results conclude that the computers are able to identify themselves as leading player and play the lowest card accordingly.


**Screenshot 24: Play Pair Hand**

The expected test result of pair hand is computer will play a pair when the previous player has played a pair and the computer also has a pair. The computer who is leading will also play a pair when the lowest rank cards have 2 cards. In the test, computers will play a pair according to the above rules.


**Screenshot 25: Play Triple Hand**

The computer is expected to play a triple when the previous player has played a triple and the computer also has a triple. The computer who is leading will also play a triple when the lowest rank cards have 3 cards. The actual test result shows the computer follows the above rule.


**Screenshot 26: 5-Hand Cards**

In easy mode, the computers are expected not to play any 5-cards hand. It was tested by playing a low-value 5-cards hand and observe any placement. The results follow the expectation.

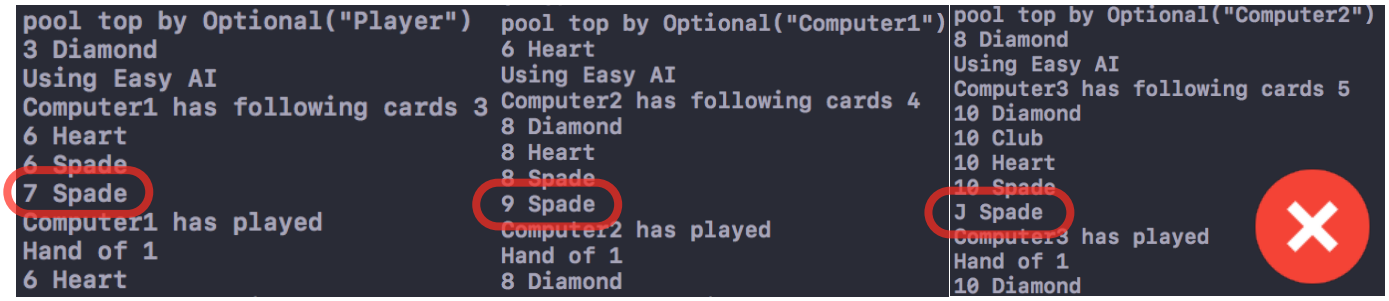
In a brief summary, easy mode artificial intelligence behave normally in general situations and play cards in a leading position ordinarily, including playing the lowest value hand in its hand, playing singles, pairs and triples but not the 5-cards hand.

In the following tests, the artificial intelligence of easy mode is tested in specific conditions. These tests are alpha tests as the data is made up. To simplify the process of testing, no graphics will be involved and check of disadvantagedness will be temporarily disabled.

For test case 1 to 3, it checks whether the computer will reserve pairs, triples and quads. The expected results are computers will play next hand and reserve for pairs, triples and quads. The following table summarises test case 1 to 3.

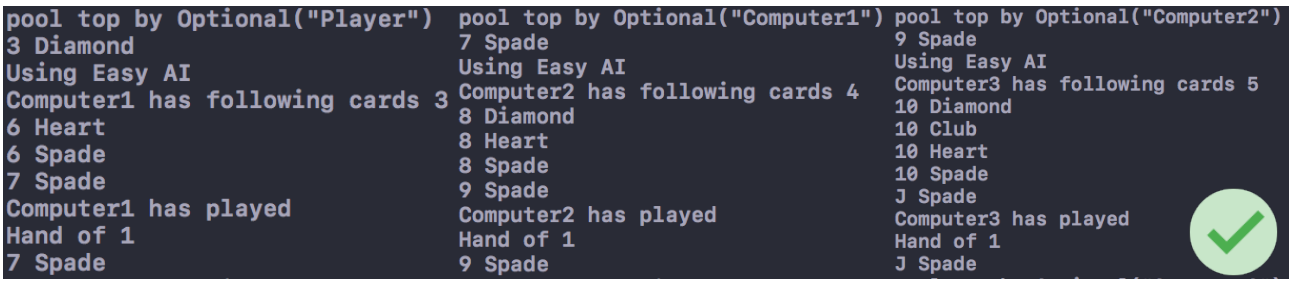
**Table 14: Test Case 1 to 3**

| Test Case | Type of Cards Playing Previously | Remarks | Computer 1 | Computer 2 | Computer 3 |
|---|---|---|---|---|---|
| 1 | Single (3) | Smaller | Pair (6) | Triple (8) | Quad (10) |
| | | Larger | Single (7) | Single (9) | Single (J) |
| 2 | Pair (4) | Smaller | N/A | Triple (8) | Quad (10) |
| | | Larger | N/A | Pair (9) | Pair (J) |
| 3 | Triple (5) | Smaller | N/A | N/A | Quad (10) |
| | | Larger | N/A | N/A | Triple (J) |



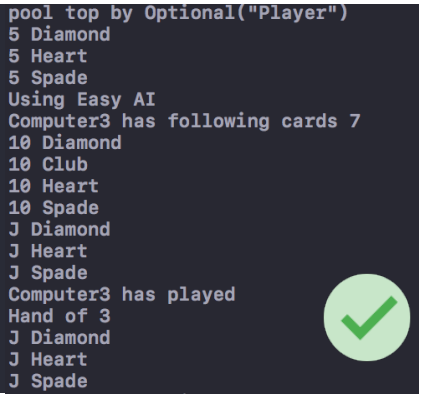

**Screenshot 27: Test Case 1**

The expected results of test case 1 to 3 is the computer will able to reserve pairs, triples and quads when there is another single hand can be dealt. The actual results do not match with the expectation. It was found that in the general action of AI does not reserve due to missing parameter in `getSameRankCards()` function. After modify, the test results follow back the expectations.




**Screenshot 28: Test Case 1 (2nd Trial)**


**Screenshot 29: Test Case 2**




**Screenshot 30: Test Case 3**

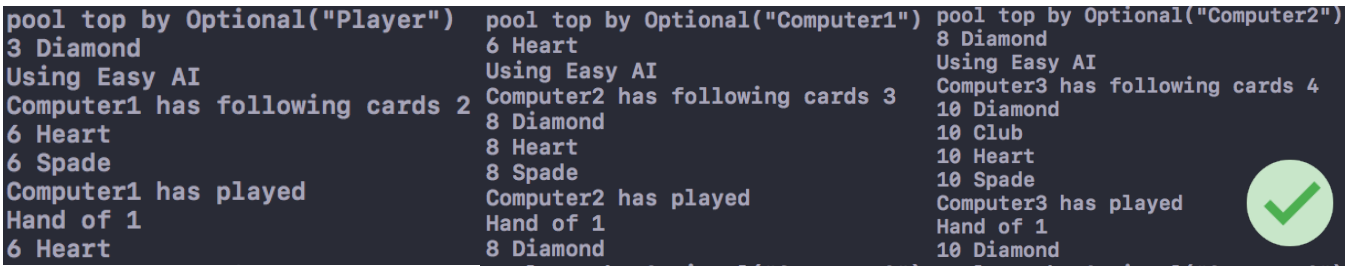
Test case 2 and 3 was expected that the computer will able to reserve cards when there is other pairs and triples can be dealt respectively. The actual results also follow the expectation.

In a small conclusion, the artificial intelligence of easy mode is able to reserve pairs, triples and quads as part of the algorithm. This makes the game more untraceable and fun.

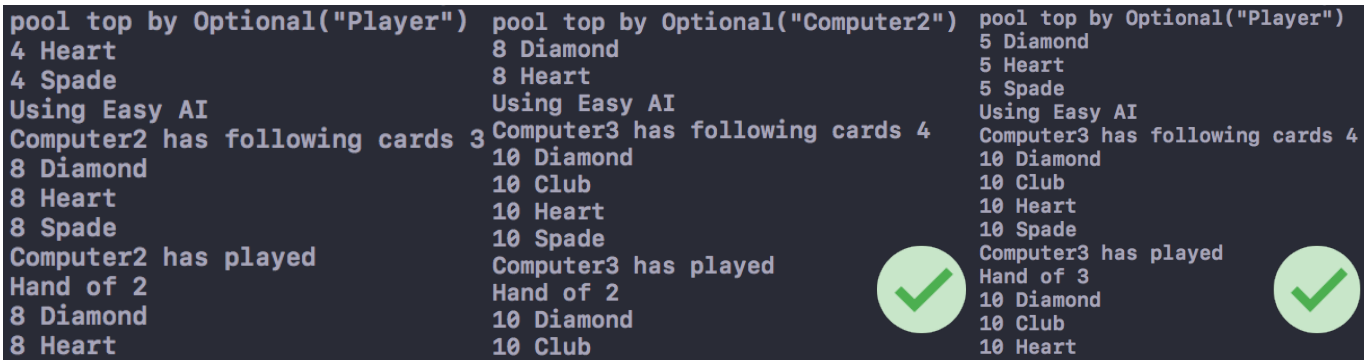
For test case 4 to 6, it checks whether the computer will break pairs, triples and quads when no more previous playing type available. The expected results are computers will break pairs, triples and quads and play the cards respectively. The following table summarises test case 4 to 6.

**Table 15: Test Case 4 to 6**

| Test Case | Type of Cards Playing Previously | Computer 1 | Computer 2 | Computer 3 |
|---|---|---|---|---|
| 4 | Single (3) | Pair (6) | Triple (8) | Quad (10) |
| 5 | Pair (4) | N/A | Triple (8) | Quad (10) |
| 6 | Triple (5) | N/A | N/A | Quad (10) |




**Screenshot 31: Test Case 4**




**Screenshot 32: Test Case 5**

**Screenshot 33: Test Case 6**

The expected results of test case 4 to 6 are that computers are able to break down pairs, triples and quads only when they do not have other hands can be dealt. The actual tests show positive results and computers are able to identify no more other hands can be dealt and break pairs, triples and quads.

To summarise briefly, in terms of general actions of the artificial intelligence, it was found that a bug but solved. Computers will deal hand correctly, in general, break pairs, triples and quads in order to place cards.

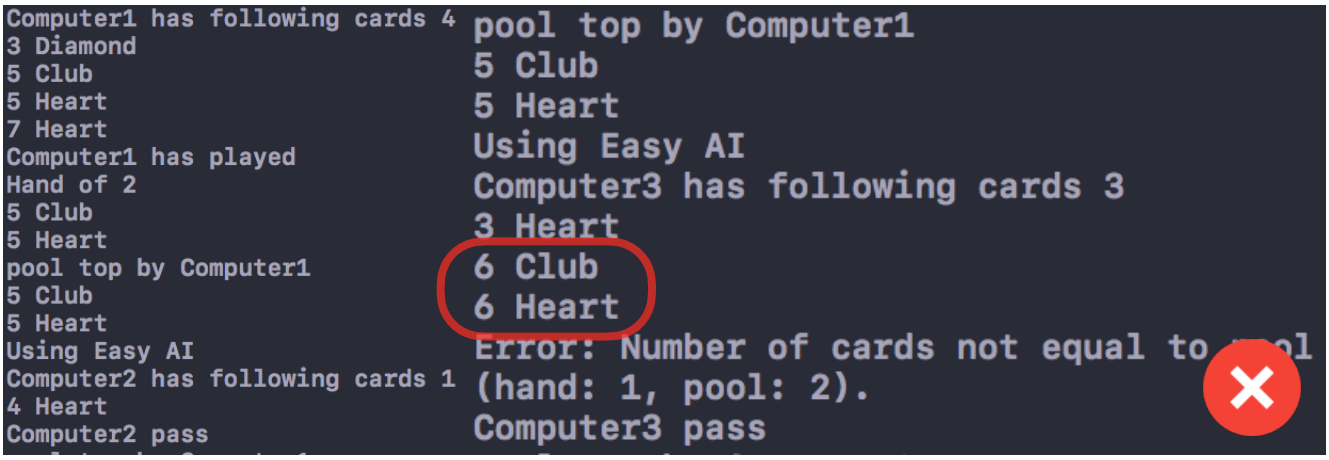
For test case 7 to 10, it checks whether the computer will prevent when the next player remains 1 card both in leading and non-leading situation. It is important to prevent as it is part of the rule of big two and makes the game more fun. The following table summarises test case 7 to 10.

**Table 16: Test Case 7 to 10**

| Test Case | Remarks | Computer 1 | Computer 2 | Computer 3 | Player |
|---|---|---|---|---|---|
| 7 | Computer 1 leading | Single (5) | 1 Card | Single (8) | 1 Card |
| 8 | Computer 1 leading | Pair (5) | 1 Card | Pair (6) | 1 Card |
| 9 | Computer 1 leading | Triple (5) | 1 Card | Triple (6) | 1 Card |
| 10 | Computer 1 leading | Quad (5) | 1 Card | Quad (6) | 1 Card |

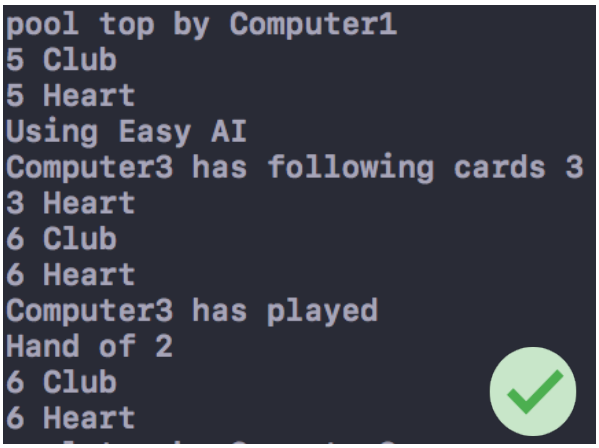

**Screenshot 34: Test Case 7**

The expected result of test case 7 is Computer 1 will deal its greatest value single card when leading. Computer 3 will also deal its greatest value card when not leading. The actual test result follows the expectation.




**Screenshot 35: Test Case 8**

Test case 8 was expected Computer 1 will deal its pair instead of a single card. Computer 3 will continue to deal pair. The first part of the test has been successfully confirmed the computer will deal pairs first if it is leading. However, it was found that Computer 3 does not continue the deal. There was a logic error which leads to forcing the AI to deal greatest single card instead when not leading and required prevention. After modified, Computer 3 will continue dealing pair when next player remains 1 card.



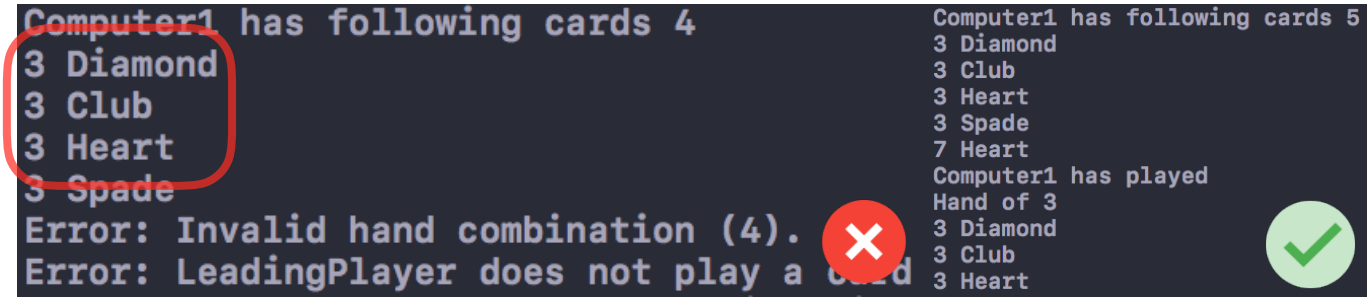

**Screenshot 36: Test Case 8 (2nd Trial)**


**Screenshot 37: Test Case 9**

Test case 9 was expected Computer 1 will deal its triple instead of a single card. Computer 3 will continue to deal triple. The actual test results show a positive result in comparing with expectation.


**Screenshot 38: Test Case 10a**

Test case 10a was expected Computer 1 will deal its triple instead of a single card. Computer 3 will continue to deal triple. The actual test results show a positive result in comparing with expectation.




**Screenshot 39: Test Case 10b** **Screenshot 40: Test Case 10b (2nd Trial)**

The expected results of test case 10b are that Computer 1 deals a triple when next player remains 1 card. However, it found out there is a bug that when leading player has a quad as its lowest value rank cards. With the little amendment, the bug resolved and continue the testing. The actual results follow the expectation.

To conclude, the artificial intelligence of easy mode is able to deal hand when leading, prevention and normal, showing no abnormalities. This ensures the base of artificial intelligence which makes testing in the normal mode will be easier.

### 1.4.2 Normal Mode

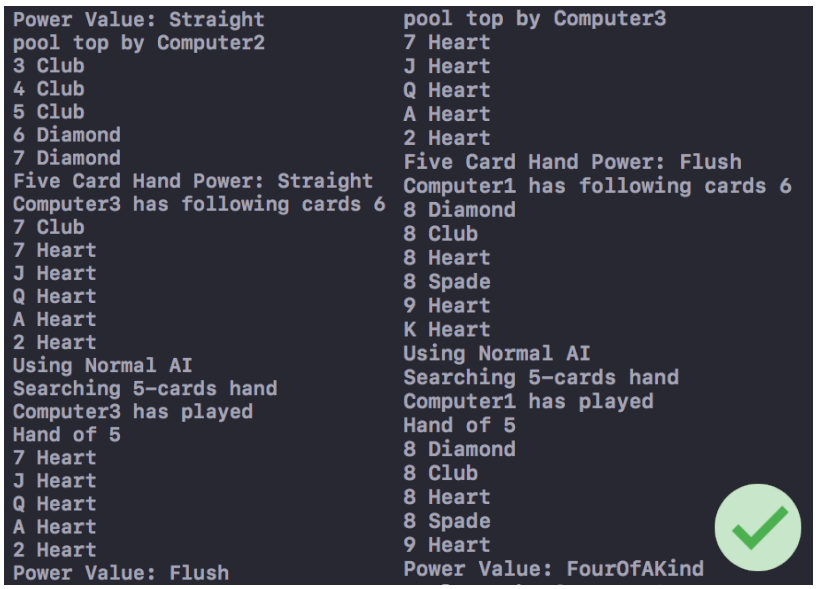
As part of the normal mode includes easy mode artificial intelligence, in the following tests, the AI is tested in specific conditions for 5-cards hand only which is an alpha testing.

For test case 1 to 7, it checks whether the computer will deal 5-cards hand both in leading and non-leading. The expected results are computers will be able to play 5-cards hand in these 2 situations. The following table summarises test case 1 to 7.

**Table 17: Test Case 1 to 7**

| Test Case | Remark | Player | Computer 1 | Computer 2 | Computer 3 |
|---|---|---|---|---|---|
| 1 | Computer 1 leading | N/A | Straight/ Four-of-a-kind | 2 Straights | Flush |
| 2 | Computer 1 leading | N/A | Flush/ Straight Flush | 2 Flushs | Full House |
| 3 | Computer 1 leading | N/A | Full House | 2 Full Houses | Four-of-a-kind |
| 4 | Computer 1 leading | N/A | Four-of-a-kind | 2 Four-of-a-kinds | Straight Flush |
| 5 | Computer 1 leading | N/A | Straight Flush | 2 Straight Flushes | N/A |
| 6 | Player leading | Straight | Straight Flush/ Straight | Four-of-a-kind/ Straight | Full House |
| 7 | Player leading | Flush | Straight Flush/ Straight | J/Q/K/A/2/3/4 | N/A |


**Screenshot 41: Test Case 1**

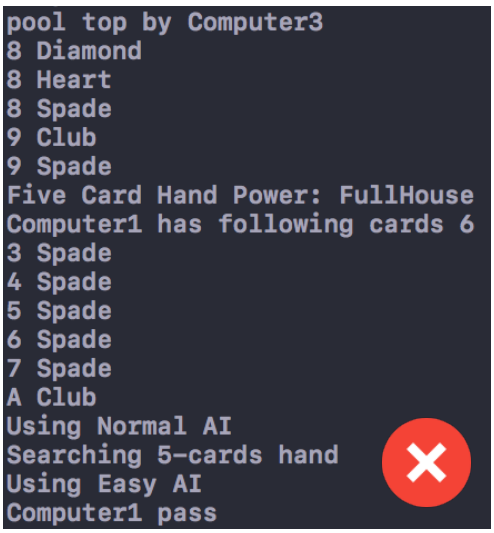



**Screenshot 42: Test Case 1**

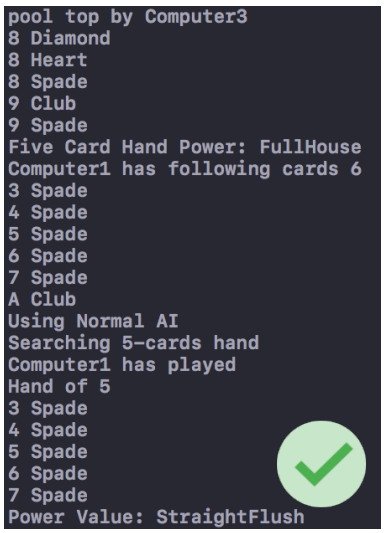
In test case 1, it is expected that Computer 1 will deal a straight in leading, while Computer 2 will deal a greater value straight instead of smaller one. Computer 3 will deal a flush and Computer 1 will deal a four-of-a-kind. The actual results follow the expectation.


**Screenshot 43: Test Case 2**

It is expected that Computer 1 will deal a flush in leading and Computer 2 will deal a greater value flush instead of smaller one in test case 2. Computer 3 will then deal a full house and Computer 1 will deal a straight flush. However, in the second round, Computer 1 passes instead of playing a straight flush. It was found that straight flush will compare the comparing card even power value is already greater. After modification, it follows the expectation.




**Screenshot 44: Test Case 2**




**Screenshot 45: Test Case 2 (2nd Trial)**


**Screenshot 46: Test Case 3**

In test case 3, it is expected that Computer 1 will deal a full house, Computer 2 will deal a greater value full house instead of smaller one and Computer 3 will deal a four-of-a-kind. The actual test results are same as expectation.


**Screenshot 47: Test Case 4**

The expected results of test case 4 is that Computer 1 deal a four-of-a-kind, Computer 2 deal a greater value four-of-a-kind instead of smaller one and Computer 3 deal a straight flush. The actual test results show no difference with the expectation.


**Screenshot 48: Test Case 5**

Test case 5 is expected that Computer 1 will deal a straight flush and Computer 2 will deal a greater value straight flush instead of smaller one. The test results follow the expectation.


**Screenshot 49: Test Case 6**

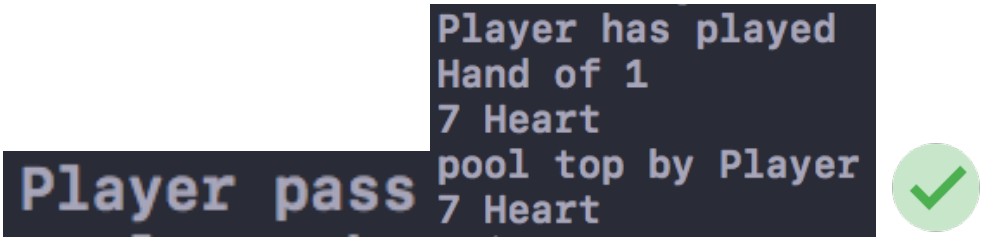
In test case 6, it is expected that a straight flush, a four-of-a-kind and a full house can be dealt after a straight. The test result show computer can identify and place the 5-cards hand accordingly.


**Screenshot 50: Test Case 7**

The last test case is expected that straight flush can be dealt after a flush and J/Q/K/A/2/3/4 cannot form any straight. The test successfully passes the expectation.

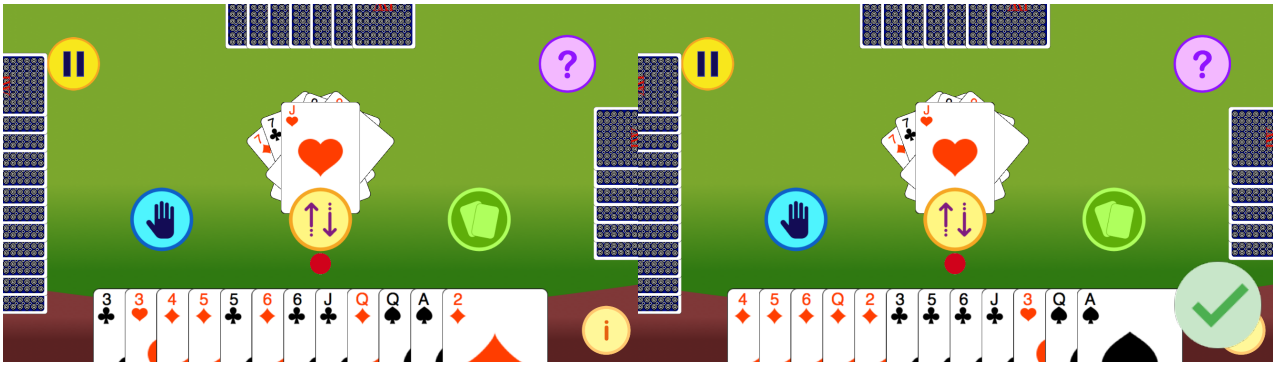
To conclude, there are no abnormalities in the artificial intelligence of both easy and normal mode. The AI will react correctly when its turn by playing the corresponding hand, like singles, pairs, triples and 5-cards hand.

## 1.5 Actions of Player




**Screenshot 51: Pass and Deal Hand**

This part is testing the actions that the player can do. It is expected that the game system can receive the pass and deal signal from the player. The toggling of cards can be the input of deal card of the system. The actual results are same as expectation.



**Screenshot 52: Sorting of Cards**

The sorting of cards is expected that when the button is pushed, the cards will toggle the sort between rank and suit. The actual test results show the expectation is true.

Briefly speaking, the actions from the player are valid and able to interact with the system. Players can play cards as they want to the system, pass when no hand can be dealt.

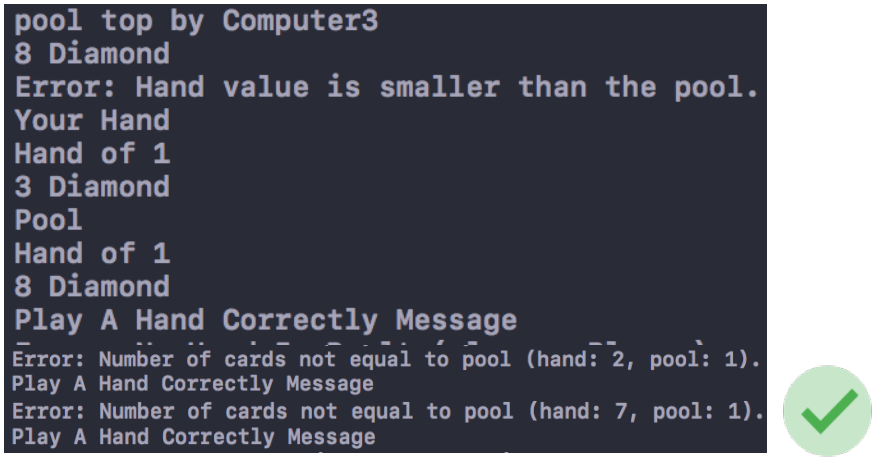
## 1.6 Hand Checking

The following tests will be tested by made up datum. This part makes sure that the rule of the game system is valid and allow users play game smoothly.

As checking of valid hand has been tested in the previous testing for a lot of times, this part will focus on all invalid hands and 5-cards hands. The following table summarises the test case from 1 to 5.

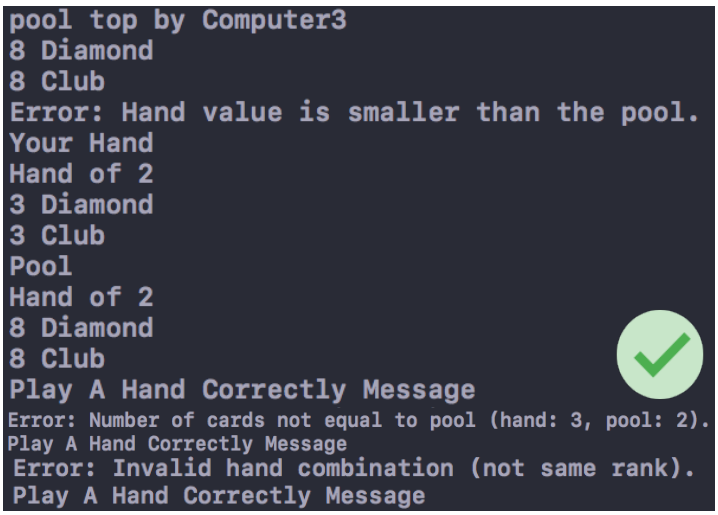
**Table 18: Test Case 1 to 5**

| Test Case | Type of Cards Playing | Invalid Cases |
|---|---|---|
| 1 | Singles | Smaller Value/ Non-singles |
| 2 | Pairs | Smaller Value/ Inconsistent Rank/ Non-pairs |
| 3 | Triples | Smaller Value/ Inconsistent Rank/ Non-triples |
| 4 | 5-Cards Hands | Smaller Value/ Straight Invalids/ Non-5-cards hands |
| 5 | Leading | Non-singles/ Non-pairs/ Non-triples/ Non 5-cards hands |



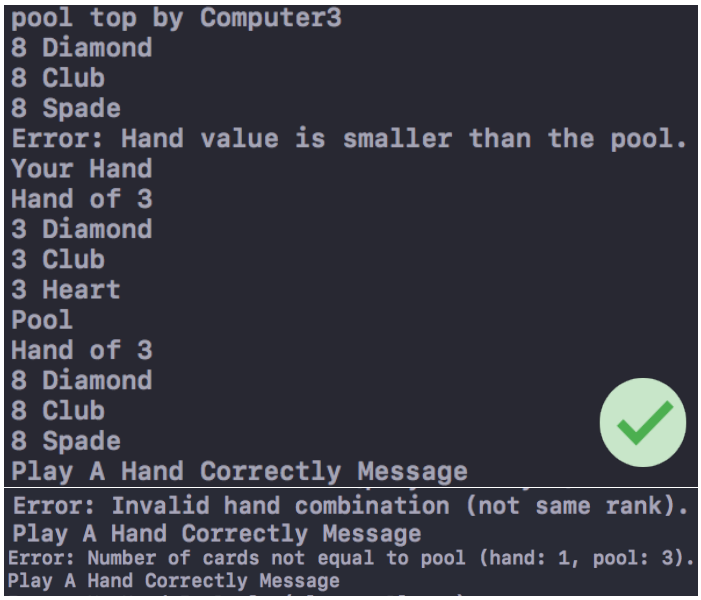

**Screenshot 53: Test Case 1**

In test case 1, it is expected that smaller value single and non-singles hand will be rejected by the game system. The actual test results show no difference with the expectation.



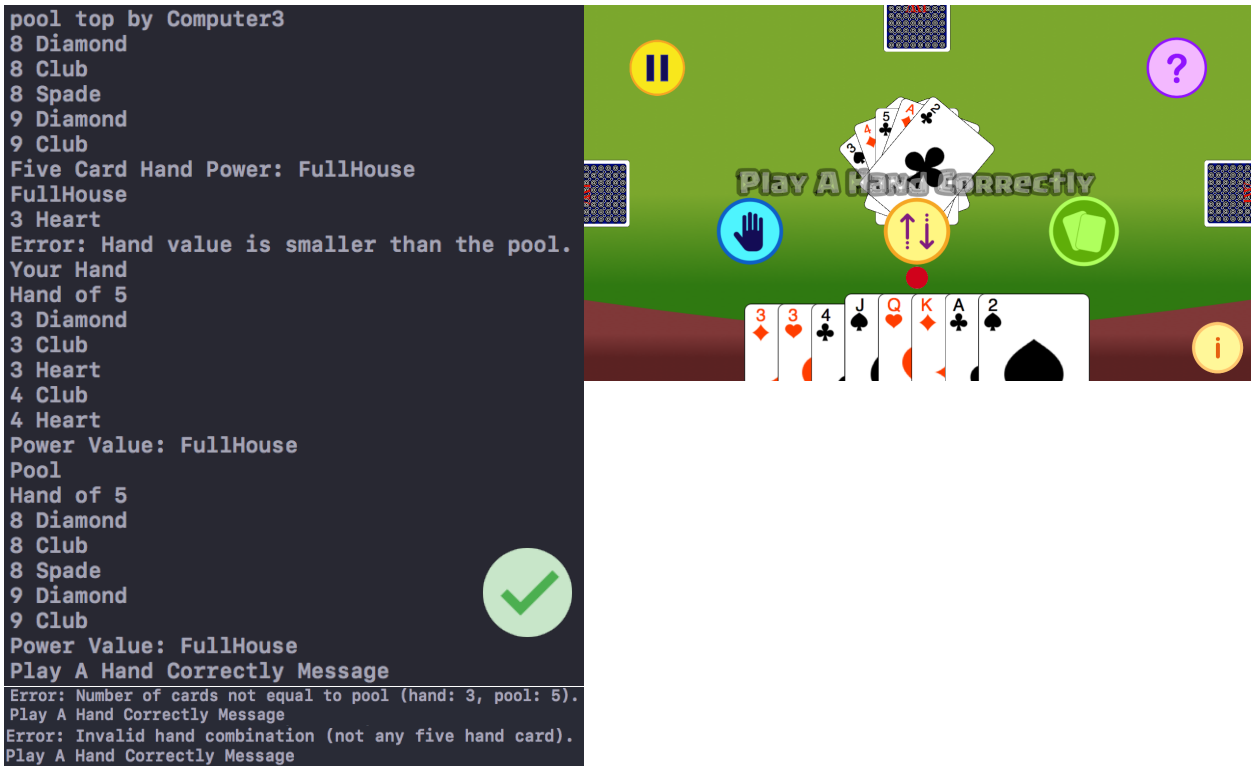

**Screenshot 54: Test Case 2**

Test case 2 is expected that smaller value pair, non-pairs hand and inconsistent rank will be rejected by the game system. The actual results follow the expectation.



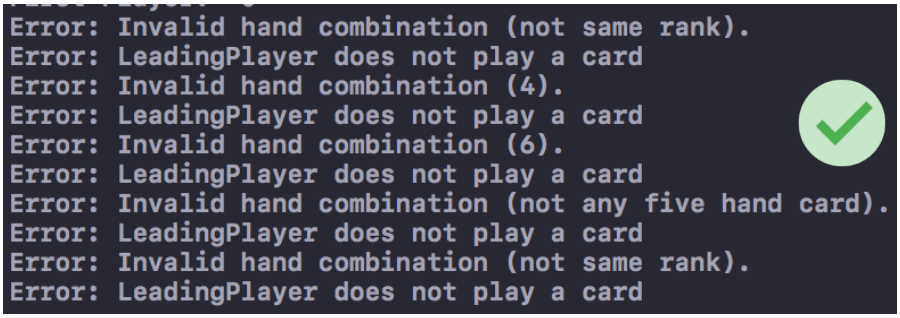

**Screenshot 55: Test Case 3**

Smaller value triple, non-triples hand and inconsistent rank are expected to be rejected by the game system. The actual test results are same as the expectation.




**Screenshot 56: Test Case 4**

In this test, results are expected that smaller value 5-cards hands, non-5-cards hand and invalid straight are rejected by the game system. The results follow the expectation.

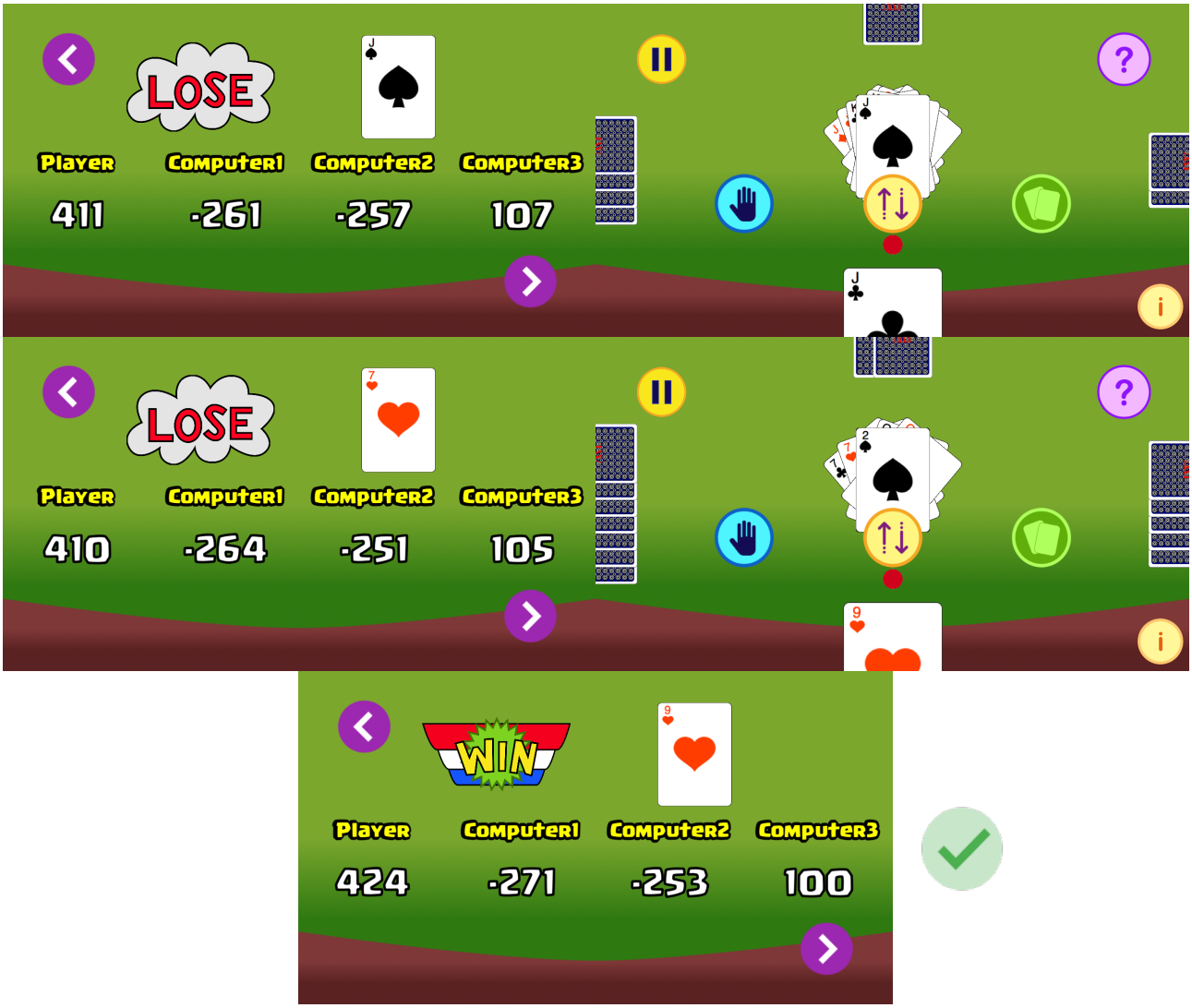



**Screenshot 57: Test Case 5**

Test case 5 are expected to test whether can check invalid hand combination even if the player is leading. The actual test results prove leading player cannot place any combination.

Generally speaking, hand checking function of the system is able to identify invalid hand combinations and reject them, which has no abnormalities and build a solid platform of the game Big2.

## 1.7 Game Result (Win & Lose)




**Screenshot 58: Win/ Lose Result**

The expected test result are the game system can correctly identify who is the winner and showing the corresponding win or lose result to players. The actual test results show that the system display result properly and successful animate the game over scene.


**Screenshot 58: Score Calculation**

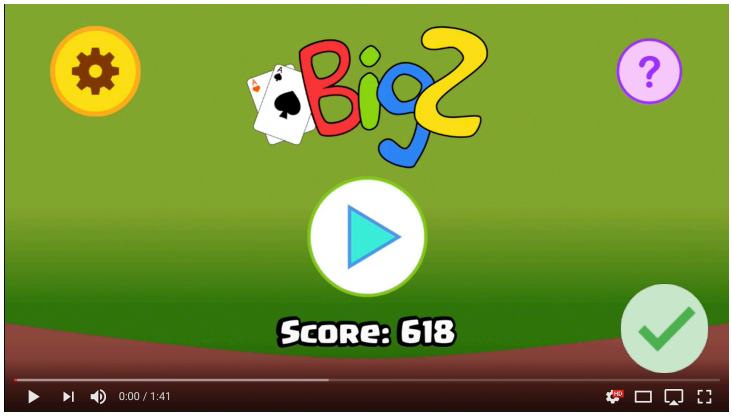
The expectation of score calculation is system can correctly identify the range of cards remain and calculate the weighted score change. At last the score of each player change accordingly. The actual results show no difference with the expectation.

To conclude, both showing win or lose result and score calculation work properly and show no abnormalities. In the end of testing of the game, it can be concluded that all functions are working normally and users can play the game smooth and enjoyable.

# 2. System

## 2.1 Object Interaction

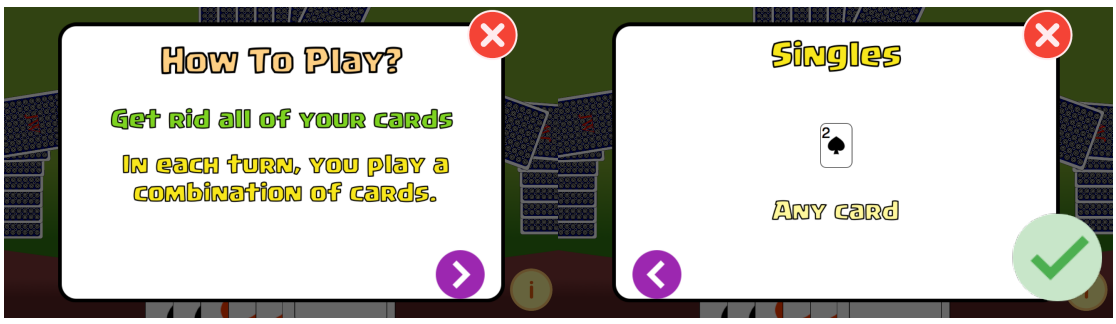
The expected results of object interaction are different sprites on the screen will react to touch accordingly. The test process is upload to YouTube(https://youtu.be/bfQN1TqTvwU) as the testing process is difficult to screenshot down. The actual results show that the object on screen will trigger operations of the application and the interaction is successful. Therefore, graphics are one of the basic I/O of the system.




**Screenshot 59: Testing Object Interaction**

## 2.2 How To Play Interface

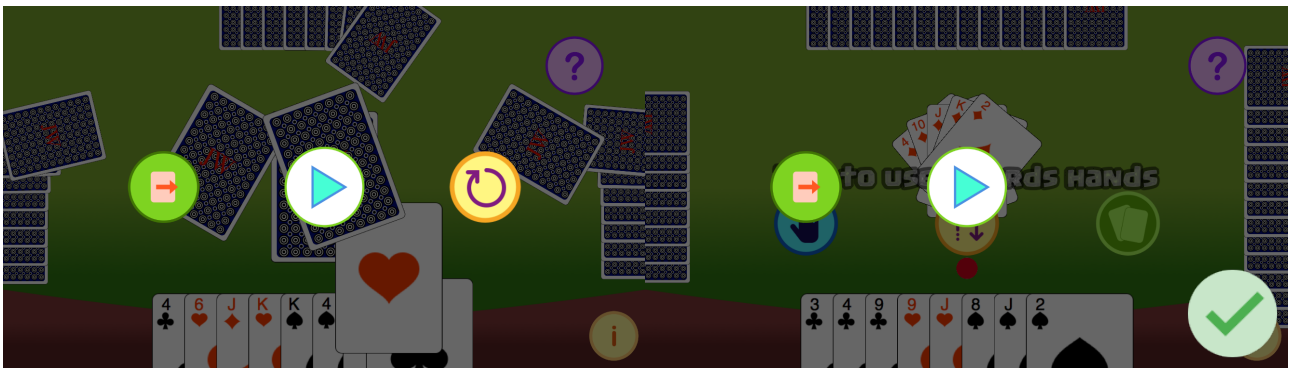
The how to play interface is expected that can only be closed by tapping the close button. The pages will interchange when tapping the back and next buttons. The first and last page will hide the back and next button respectively in order to prevent the page going beyond the availability using boundary cases. If players are in the game scene, the game will also pause at the same time. The actual test results show no difference with the expectation.




**Screenshot 60: Front & Back Page of How To Play Interface**

## 2.3 Pause

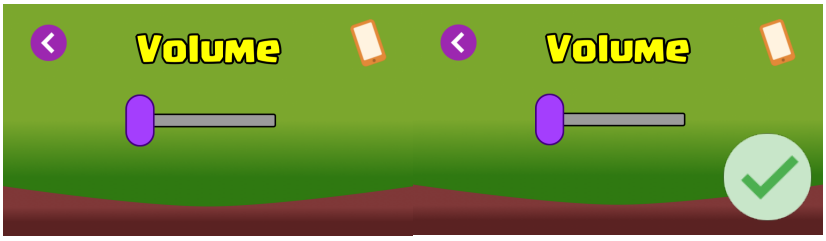
Expected results of pause are the game animation and program will also be paused. When the first round is over, the game will not be able to restart in the pause menu. Moreover, a pause will be triggered when any interruption of the application arise. The actual test results are same as expectation which pauses the game appropriately.



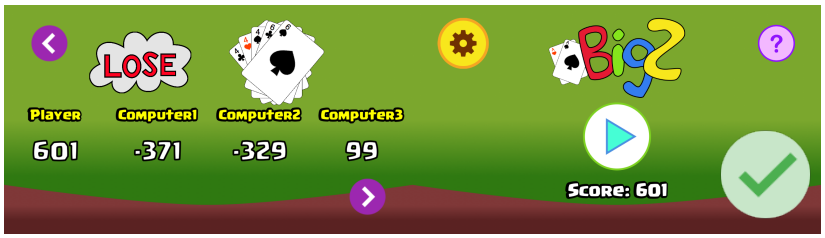
**Screenshot 61: Pause Game**

## 2.4 Data Storage

It is expected that the game will save the setting (volume & vibration) status and the scores of player and computers. The test results show the datum are saved properly whenever the game terminates.




**Screenshot 62: Save Volume**




**Screenshot 63: Save Score**

## 3. Summary

**Table 19: Test Summary for Game**

| | 1. Game | Result |
|---|---|---|
| 1 | Game Visualisation | ✓ |
| 2 | Card Distribution | ✓ |
| 3 | Players & Turn Taking | ✓ |
| 4 | Artificial Intelligence | ✓ |
| 5 | Actions by Player | ✓ |
| 6 | Hand Checking | ✓ |
| 7 | Game Result (Win & Lose) | ✓ |

**Table 20: Test Summary for System**

| | 2. System | Result |
|---|---|---|
| 1 | Object Interaction | ✓ |
| 2 | How To Play Interface | ✓ |
| 3 | Pause | ✓ |
| 4 | Data Storage | ✓ |

The application passes total 11 unit tests and both 2 system tests have passed. The units in the system compile well together. Although some bugs were found in the game, but they were identified and removed quickly. Those bugs do not bring significant impact to the gameplay as most of them appear only in very specified conditions. In general, the testing can proceed to the beta and user acceptance test.

In future, if the game continues to develop, these tests provide a solid foundation for the development, though there will be more test to be carried out. I believe these tests gave a great opportunity for me to discover some hidden loopholes of the system. As these remove earlier, less cost will be needed to fix in future.

# F. Evaluation

## 1. Evaluations of Requirements

At the start of the project, I have set a number of objectives, which hoped to accomplished. The following tables are my evaluation on to what extent I have accomplished.

**Table 21: Objectives in Basic Functions**

| | Basic Functions | Percentage |
|---|---|---|
| 1 | Visualising Elements in the Game. | 100% |
| 2 | Distributing and Collecting Cards from Players. | 100% |
| 3 | Playing Cards with Validation of Combinations. | 100% |
| 4 | Sorting Cards of Players. | 100% |
| 5 | Determine Winner. | 100% |
| 6 | Menu. | 100% |
| 7 | Settings. | 80% |
| 8 | Save & Load Game. | N/A |
| 9 | Tutorial Mode. | 50% |

**Table 22: Objectives in Feature Functions**

| | Feature Functions | Percentage |
|---|---|---|
| 1 | Multiple Levels & Rules (Modes). | 60% |
| 2 | Scoring System. | 90% |
| 3 | Timing System. | N/A |
| 4 | Search Five-card Hands. | 70% |
| 5 | Leaderboard. | N/A |
| 6 | Smooth Animation. | 90% |
| 7 | Sound Effects. | 70% |
| 8 | Pause & Restart Game. | 100% |
| 9 | Profiles of players. | 20% |

Summarising the objectives of basic functions, most of the functions has successfully achieved. In particularly discussing those are not accomplished, setting function is not completely same as expected in quality and quantity. More settings should be available, like toggling the automatic reminder and changing players' profiles.

The save & load game function is not yet developed due to insufficient time and unfamiliar with the storage method of swift. Because of the limitation, I can only save limited data, such as player scores and previous settings, to the internal storage. In the design stage, I planned using the test file as a storage method, however, there is another kind of storage method, which can store persist data much efficiently. If more time is allowed, the game can be saved easily by this method.

Big2 does not have a real tutorial mode, but having a complete tutorial guide in the how to play menu and random automatic reminder. These are used to replace tutorial mode as it requires much more time cost to complete the project. How to play menu provides a detail game rule for players, while automatic reminder provides a succinct message to remind players. Although these two types are different from the tutorial mode, the function of teaching players is still achievable but with a relatively poor user-friendly interface.

Some objectives focus on the feature functions and most of them I have been implemented but not good enough. Multiple levels & rules/modes are expected to complete, making the game with more variations and fun. Yet, I have only complete easy and normal game mode as game modes having a higher level require deep investigation in the AI system to give a well-developed level. With limited of time, the level content cannot be as rich as expected.

The timing system is not implemented also due to insufficient time and timing system is not a foundation compare with other functions. But undoubtedly, the timing function can bring more excitement to the game. Although this is not the first priority of the development, it can be developed in a relatively short time compare with other feature functions.

Although searching 5-cards hand has been successfully implemented, there are no strategies to reserve and plan how those hands will be played. This function requires more time to develop, which makes the AI much untraceable. Searching 5-cards hand is not a difficult task, but instead arranging how they will be played is much difficult. It requires more deep learning in the field of Artificial Intelligence and the game, big two, itself.

As the profiles of different players are not implemented, developing a leaderboard is meaningless. The leaderboard is originally designed for a place that shows the overall record of the players playing the game in different profiles. But with no profiles, the only reason to continue developing the leaderboard is an online leaderboard, gathers profiles of each device to form a leaderboard. However, this is not one of the aims of the project.

Sound effects are implemented but not in a much greater extent like announcing what hand has been played, especially when 5-cards hand is dealt. However, the overall audio system is well developed which is able to add more sound effects easily.

As mentioned before, profiles of players is not well developed. Big2 can only save 1 player's profile. But this is due to another observation for development in mobile devices. Most of the mobile devices are owned by a person only. This contributes to changing profiles much just a fancy function but not useful at all. Profiles of players should store in online database and login profile in different devices. The outdated multiple profiles in a device might need to change in order to cater the current trend.

Regarding other basics and features functions, I believe these functions were done very well and close to expectation. From the visualisation of elements to the whole game system, all of them are carefully designed for user-friendly and easy to use bases. Big2 works as expected with a precise and clear interface. Overall, The game is well-developed in terms of developed by a single person. I am satisfied by the game made for a long time and with a lot effort. Moreover, the application is implemented with documentation and clear definition of functions. This can help to develop other function in the game easily.

## 2. Improvements & Further Development

There are a few improvements during the project. Most of the features cannot be accomplished is due to insufficient time although it is completed on time. So, I should manage time much better next time. A good time management can help to complete modules step by step. This makes sure each module can be developed within a certain time. The planning of module implementation is also crucial which requires determining self-ability. With a better time planning, I believe the project can be completed more successfully.

During the project, there are some amendments in the design of the application, due to different factors, like problems arise when implementation or the design is not suitable at all. Larger amendment in the design not only does not achieve the original goal but also easily distracted by other not included aims. Although some modifications might need, but this should be minimised. For the next project, the design should be thought deeply, which includes whether the function or aim caters the users' need. With a more well-designed plan, the project can be implemented smoothly with a clear goal.

In terms of the game, there are a lot of rooms for improvement, including the graphics, animation and sound effects. However, this not only needs a good artistic design but also considering is the interface user-friendly with a good user experience. This is much important as a better user experience gives a better impression to users. This might be only fancy, but it really attracts players' attention and create a good atmosphere to them.

To give a much detail further development, I would first talk about the artificial intelligence of the game. To be honest, the AI in the current game system is extremely basic. In order to develop a professional level or new game mode, AI is needed to improve. By using an "ability" of cards, which means how can the card be played, this can help maximising the win rate of AI. The AI should be much more untraceable to provide a varying game condition instead of boring gameplay. In the future development, this part is undeniably important and in a first priority.

The implementation of the save and load the game is also quite important. This allows players to continue their game when they have already quitted. Players can continue the previous game as a good hand card might not be got easily. This function is particularly useful when the device left no batteries or accidentally terminate the game. Therefore, I believe the further development cannot be done without the automatic save game function.

The tutorial mode will be one of the future development aims. The current how to play interface although is informative, but it is quite boring. Player need to read a lot of information to play a game if they have never played. The desire tutorial mode is a special level that only teaches the player how to play the game in depth step by step, like pausing the game to instruct what player should do or can do, as a computer-based training. This method to teach players is much interactive and interesting. The tutorial mode will also be forced to activate when the first time enter the game so as to make sure all players know how to play big two.

Timing function has not been implemented currently, but it should be done in future. A timer for urging players to deal card can train up their reaction time and thinking, which pushes them thinking continuously. This function can be quickly implemented into the system in future. The timer also keeps constant pressure on players and keep them play. For another reason, if online matching will be implemented, this is needed to make sure no players will over time to provide a good user experience for all of the players.

Online matching can be one of the developing direction in future. This not only reduces the demand on developing the artificial intelligence but also brings much more variations to the game. Players can interact through online matching and play with different players. Firebase is one of the possible ways to make online matching possible. However, this will certainly be a long way if only a person going to implement it. As online matching requires programming skill in the network, like preventing over players in a room, only client side programming might not good enough for future development. Server side programming may need to be learnt in order to make a good and functional online matching system.

I believe, if I am going to continue the project, I will improve the project management skills and uses these further development direction as the next objective of Big2 2.0. The system can surely be enhanced in future.

## G. Discussion

There are a lot of things learnt through the project. Project management skills are crucial to the quality of the application going to develop. A project holder should have a good time management skill and analytical skill at the start of the project. These help the project being planned in an organised way and increase my working efficiency. The flow of producing a product have been experienced. I have tried how to analyse, design, implement and test the whole project. This is surely beneficial to my future projects.

In the part of design, I have learnt to design an user-friendly interface. This is quite important that the popularity trend is to use an interface that is user-friendly. The user experience also cannot be neglected as some interface looks awesome but not easy to use at all. Moreover, I realised that the flow of analysing and design cannot be reversed. There are a lot of results from analysing the situation and problem that will directly affect the design. Therefore, to make a good design, the analysis should also be done to get users' requirement.

During implementation, this is a tough time for me to practise and learn a new programming language as I have learnt Swift only about a year. Not only figure out how to solve problems with skills and concepts but also the actual implementation. An object-oriented programming language is good for organising a lot of different elements and objects in the system. It also requires high modularity as this simplifies the code and reuse efficiently. Furthermore, this increases the readability of the source code which helps programmers continue developing Big2 by understanding how to build.

On the other hand, the implementation also trained up my programming skills. Some of the algorithms may are inspired from lessons, like recursive distributing cards and merging cards. To solve a big problem, we can split the problem into smaller sub-problems by top-down approach. Divide and conquer can also be used in the implementation. We can see that outlining tasks and solve them separately can be used to develop a solution and system.

Testing is also an important step which helps me to discover own limitation. Before that, I might always think that the program does not have serious bug or programs do not require such detailed planned test. However, I found that these plans are a great place to test the program in an organised way. Some bugs can be found even though they exist very rarely. I learnt that every program should be tested in a harsh environment which can lead to beneficial to both developer and users. The later the discovery of the bug, it requires the larger cost to fix it. These tests help to discover bugs quickly and fixed with a lower cost. To the user, the application has fewer bugs bring better user experience and more enjoy the game.

The system development life cycle of the project is mainly using the waterfall model. I have identified the users' requirements in the analysis stage. Each phase from analysis to testing is proceeding in strict order. Some errors in the design and analysis stage were found, but they do not have a significant impact on the overall project. It is undeniable that the project should have some extent of flexibility. Prototyping has been used which continuously building the system from simplified version and refine the prototype to a complete system. Some interface design errors can be fixed quickly. But sometimes, because of this, I out focus the original aim and focus on minor features. This might be a hard time to choose whether develop a new function. Therefore, rapid application development methodology is used instead. For a good project holder using rapid application development, he should have a clear idea and plan about the scope of the program and this is a best opportunity to learn from this.

This project is also an opportunity to revise and practice what I have learnt throughout the New Senior Secondary Curriculum of ICT. The system development might not have a lot of chances to put learning into practical. This helps me develop the application step by step and accomplish the tasks. During the implementation, I found the current system is difficult to develop further. Then, I decide to directly cut over the old system and convert to a new system. Though it is a high-risk strategy, but a backup of source code can recover the risk quickly.

In the documentation, I not only revise what should be included in the document but also how to use a word processor to tidy up the document. For example, using the table of content, photo, footnotes and tables can create a better view of the document. This help me to organise and present the document in a better way.

The revision of Input-Process-Output defines the operation of each function. On top of that, the input and output devices have to be considered seriously in the design stage as this build up the foundation and basic I/O of the system. As this project is not as usual use a keyboard or mouse to input, but a touch screen. This simplifies the skill need to be used and allow me to design a tailor-made interface for users.

As mentioned before, I have found that the profile system in the design stage is not appropriate due to the revision in a different computer system. Big2 runs on iOS devices including smartphones and tablets. These devices become much more personal than before. Thus, the expected users and frequent users should be the owner of the device. This eventually helps me to determine an outdated function and stop develop a low-cost efficiency function.

This project has significantly revised intellectual property as Big2 includes different multimedia elements with different authors. It is very important to acknowledge the author and check the licence agreement how the object can be used. Otherwise, this might lead to copyright infringement. Permission should be granted when the multimedia elements and other intellectual properties are used. In this Information Age, people should be information literate and respect the intellectual property right owner.

Most importantly, programming is much a self-learning process which requires a lot of motivation by the programmer itself. Although creating a complex application is a great challenge, I overcame it and learnt a lot from it. In the future, I hope I can sustain a self-learning process in programming and continue to develop applications.

# H. Conclusion

Big2 has finally come to an end. It is an iOS application developed in Swift language. Big2 is a card game using poker to play big two, which is very popular in Hong Kong. The game aims at getting rid all of the player's cards. This project has been completed within the schedule.

Players can toggle the cards to be played and tap decision buttons to pass, sort and deal correspondingly. It has a scoring system which calculated by cards remain on hand. Therefore, all players should get rid all of his cards when possible. There are a total 2 modes, easy and normal. The artificial intelligence in the game will behave like a person to play with players. Big2 provides different functions to the game, like setting, pause, how to play menu and automatic restart. These allow players more enjoy the game with these features.

The application developed in Swift, object-oriented programming language, has a well-organised structure with the classes and methods. Future developers can reuse, inherit and modify the source code easily and enhance the application. It has high extensibility which can add new features in a simplified way.

The further development of Big2 can range from new artificial intelligence, new game mode to online matching. There are a lot of possibilities when further developed. As the original developer of Big2, I am glad to finish the first stage of the application. However, there is a long way to achieving success by a single person. I hope Big2 can continue to be developed and available for download in App Store one day.

All in all, this project has come to an end. The process is tough but I can learn a lot from it which is beneficial to my future. It is sad to stop developing Big2, but it will be another start of my life. I hope Big2 will be developed in future, but I also hope more applications and games can be developed in future, to bring convenience and happiness to every single user.

# I. Reference

## 1. Disclaimer

Big2 Disclaimer


St. Paul's College 6D 19

Big2 is a swift application for submitting Hong Kong Diploma of Secondary Education(HKDSE) Examination, Information and Communication Technology(ICT) School-based Assessment(SBA) in the year 2017. Further details of this application are included in the report. In the following paragraphs, “the creator” represents “Wong Tsz Nok Joshua”.

For "Supercell-magic-webfont.ttf" font file, it is downloaded from cocland.com. This font is used for system labels of this application. Download link: http://cocland.com/miscellaneous/download-clash-of-clans-font

Class "SKOutlinedLabelNode" is modified from MKOutlinedLabelNode by Mario Klavar. This class is under The MIT License(MIT). it is used for adding a border to the
labels to display text clearly. Source Link: https://github.com/marioklaver/MKOutlinedLabelNode/blob/master/MKOutlinedLabelNode.swift

MIT: Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT

HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

All images in "Assets.xcassets" is drawn, merged and used by the creator. Some of the sub-images are from VectorIconBoxFree and Icons8 and used under the licenses freely. Source Link: https://icons8.com

For "DealCard*.caf" audio file, it is downloaded and modified from YouTube Audio Library "Card Dealing" under the terms and conditions of this library (Music from this library is intended solely for use by you in videos and other content that you create.) Source Link: https://www.youtube.com/audiolibrary/soundeffects

For "Lights.mp3" music file, it is downloaded from NoCopyrightSounds and created by Jim Yosef without limitation in the use of music. Source Link: http://nocopyrightsounds.co.uk/video/jim-yosef-lights-ncs-release/

All other audio resources are recorded, modified and used by the creator.

## 2. System Requirement

| | |
|---|---|
| Operating System: | iOS 9.0 or above |
| Devices: | All iOS devices[18] installed iOS version 9.0 or above |
| Storage Size: | 46 MB |

[18] Big2 is developed and tested only in iPhone 6s installed with iOS 10.2. Other devices with the requirement are only expected to operate same as tested.

## 3. Installation

Files: Big2
- -AppDelegate.swift
- -Assets.xcassets
- -Audio Resources
  - --DealCard*.caf
  - --IntroEffect.wav
  - --Lights.mp3
- -AudioController.swift
- -Base.lproj
- -Card.swift
- -Deck.swift
- -Disclaimer.txt
- -Extension.swift
- -Function.swift
- -Game.swift
- -GameoverScene.swift
- -GameScene.sks
- -GameScene.swift
- -GameViewController.swift
- -globalConstant.swift
- -GraphicsController.swift
- -Hand.swift
- -Info.plist
- -MenuScene.swift
- -Player.swift
- -Pool.swift
- -SettingScene.swift
- -Supercell-magic-webfont.ttf

Big2.xcodeproj

Open Big2.xcodeproj with Xcode (please refer to https://itunes.apple.com/hk/app/xcode/id497799835?l=zh&mt=12 for more information)
Select device/simulator to install at top left corner.
Run the scheme by selecting from the menu > Product > Run (or pressing ⌘R).
Open the application once installed.

## 4. Websites


1. Big2 Demonstration: https://youtu.be/bfQN1TqTvwU
2. Apple Developer Swift: https://developer.apple.com/swift/
3. Big Two (Wikipedia): https://en.wikipedia.org/wiki/Big_two
4. 博雅•鋤大地 (App Store): https://itunes.apple.com/cn/app/bo-ya-chu-da-de/id549724881?mt=8
5. Big 2 Online (App Store): https://itunes.apple.com/hk/app/big2-online/id375750079?mt=8
6. C (Programming Language) (Wikipedia): https://en.wikipedia.org/wiki/C_(programming_language)
7. C++ (Wikipedia): https://en.wikipedia.org/wiki/C%2B%2B
8. Swift (Programming Language) (Wikipedia): https://en.wikipedia.org/wiki/Swift_(programming_language)
9. Scala (Programming Language) (Wikipedia): https://en.wikipedia.org/wiki/Scala_(programming_language)
10. Apple Developer Xcode: https://developer.apple.com/xcode/
11. Code::Blocks (Wikipedia): https://en.wikipedia.org/wiki/Code::Blocks
12. Dev-C++ (Wikipedia): https://en.wikipedia.org/wiki/Dev-C%2B%2B
13. Xcode (Wikipedia): https://en.wikipedia.org/wiki/Xcode
14. Eclipse (software) (Wikipedia): https://en.wikipedia.org/wiki/Eclipse_(software)
15. Apple iOS: http://www.apple.com/hk/ios/ios-10/
16. Apple macOS: http://www.apple.com/hk/macos
17. macOS (Wikipedia): https://en.wikipedia.org/wiki/MacOS
18. iOS (Wikipedia): https://en.wikipedia.org/wiki/IOS
19. Game font: http://cocland.com/miscellaneous/download-clash-of-clans-font
20. MKOutlinedLabelNode: https://github.com/marioklaver/MKOutlinedLabelNode/blob/master/MKOutlinedLabelNode.swift
21. Icon8: https://icons8.com
22. YouTube Sound Effects: https://www.youtube.com/audiolibrary/soundeffects
23. Background music (Light): http://nocopyrightsounds.co.uk/video/jim-yosef-lights-ncs-release/
24. Swift Documentation: https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/index.html#//apple_ref/doc/uid/TP40014097-CH3-ID0