# Chapter I - Introduction

For the last project of Object-Oriented Programming class, we have produced a project which is using basic and kind of advantaged of OOP knowledge and design pattern like Singleton, Publish-Subscribe.

Java is an only programming language we have used in this project, a representative language in OOP. Besides that, we have used Swing, Socket for our display and networking.

In this report, we will give a detail description on our project. The structure of our report is outlined as follows:

Chapter I: Introduction of our project.

Chapter II: Rules and gameplay.

Chapter III: Display UML class diagram of the project.

Chapter IV: The possible improvements in the future.

# Chapter II - Rules and gameplay

* 1. Gameplay

This is a type-based 1v1 street fighting game in which players will have to type a combination of keys in order to use skills. Each skill has its own graphics, element type, and a set of combination of keys.



*Figure 1. Starting of game*

*Figure 2 - Several scenes of gameframe*

**

On each 5ms, the game frame will be reset and received, rendered the key which is pressed by player or taking next position of the existing skills.

****The game will end when one of the players being taken down (out of blood).

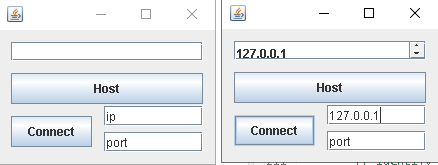
*Figure 3. Ending of the game*

* 1. Networking

Our game just only taken one game per time. If you want play again, just reopen it and play it again.

Beside the main game, we product the networking system that connect two devices in one Lan network, one is host, and one is client.

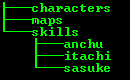
So that, two players can play through the Internet.

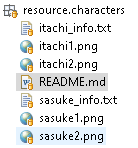


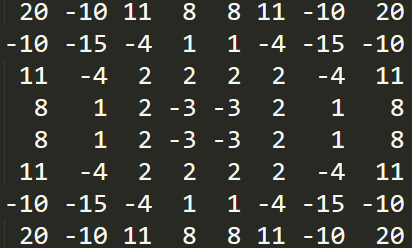
*Figure 4. Game connecting*

# Chapter III – The detail of Game technique

* 1. Main game tree
     1. Introduction about function of used class:
* Controllers:
* class Main():
  + Implement ActionListener class:
  + By implementing ActionListener, we can use many offered functions like:
    - actionPerformed() (doing when the timer call the class per 5ms).
    - KeyListener() (get the keyboard action in order to take the combination skill).
  + There are two variables which Main() received from ServerUI in order to determine the information of player that is: id (id of player) and character (character chosen by player)
* class RenderManager():
  + Render background, player and existing skill. Then, adding to jframe in order to display it.
* class PlayerManager():
  + Saving the information of two player, HP, mana, shield, and status of player.
* class SkillManager():
  + Saving the existing skill of both player.
  + Detecting when skill hit the opponent player or hit skill of the opponent player.
* class NetworkManager():
  + aaa
* Player:
* Player():
  + Implement Serializable(): serialize the data, in order to transport the data.
  + Received name of character and its id.
  + Get information of that character.
  + Call SkillRender() in order to generate own skills.
* PlayerRender():
  + Extends JPanel(): render the image, hp, mana, shield.
  + Calling only by RenderManager() each frame in order to update display information.
* Skill:
* SkillRender():
  + Check the legalization skill.
  + Save information of each unit of skill.
  + Update the next position of each skill.
    1. Flow of main game
* First, there is a singleton class Main() in which cover all of the window, frame, action… of the game.
* Second, the Main() call init() in order to:
  + Define the jframe.
  + Generate background.
  + Generate player.
* Next
  + Each frame will listening for the KeyListener() in order to get key from keyboard, saving it.
  + If ‘Space’ is pressed, Player will check the number of combination keys. Then, call the SkillRender() class to checking and generating that skill.
  + If the skill is generated, the skill will be append into SkillManager(). Therefore, we can re-calling and re-painting each frame.
  1. Main server tree
  2. Resources
* The resource of game will be held in ./src/resource folder.

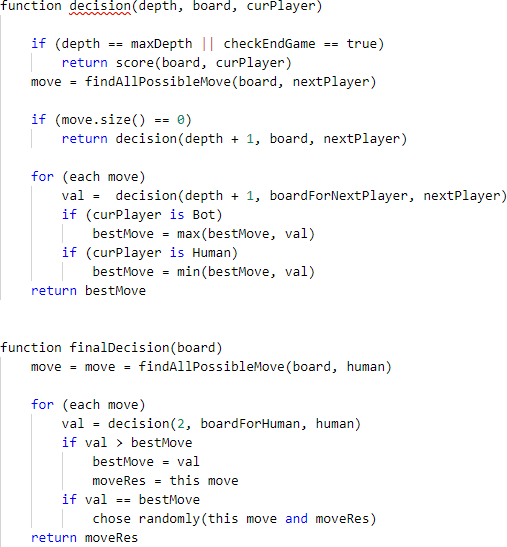


* Include:
* Characters:
  + <character\_info.txt>: save information of character, structure in README.md file.
  + Two images of character, one for the left side, and one for the right side.
* Maps:
  + Image of background.
  + Image of ending scene.
* Skills

The main idea of Minimax is recursion. At every depth of recursion, this algorithm will list all moves and continue to recur them. If the current depth is a BOT move, this algorithm will choose the move that has the highest score. If more than one of the highest scores, the algorithm will randomly choose any of them. If the current depth is player move, the algorithm will do the opposite. At the maximum depth, this algorithm will compute the region score. The region score is defined as the sum of all cells in which that chess pieces is located. Each cell has either negative or positive weight, depending on its importance.

*Figure 5. Weight in each cell*

This is Pseudocode for Minimax Reversi algorithms



# Chapter IV - UML Class Diagram

|  |  |  |
| --- | --- | --- |
| Options | |  |
| -yellowJButton: JButton  -greenJButton: JButton  -grayJButton: JButton  -brownJButton: JButton  -JPanel1: Jpanel  -jLabel5: JLabel  -jLabel4: JLabel  -jLabel3: JLabel  -jLabel2: JLabel  -jLabel1: JLabel  +Options()  -initComponents(): void  -brownJButtonActionPerformed(evt: ActionEvent): void  -greenJButtonActionPerformed(evt: ActionEvent): void  -yellowJButtonActionPerformed(evt: ActionEvent): void  -grayJButtonActionPerformed(evt: ActionEvent): void | |
|  |  | |

|  |
| --- |
| MenuView |
| -titleJLabel: Jlabel  -playervsplayerJButton: JButton  -playervspcJButton:JButton  -optionsJButton:JButton  -jPanel1: Jpanel  -jLabel9: JLabel  -jLabel8: JLabel  -jLabel7: JLabel  -jLabel6: JLabel  -jLabel5: JLabel  -jLabel4: JLabel  -jLabel3: JLabel  -jLabel2: JLabel  -jLabel1: JLabel  -jEditorPanel: JEditorPanel  -guideJButton: JButton  +playerNamesClose: int  +instance: MenuView |
| -MenuView()  -initComponents(): void  -playervsplayerJButtonActionPerformed(evt: ActionEvent): void  +actionGui(): void  -optionsJButtonActionPerformed(evt: ActionEvent): void  -playervspcJButtonActionPerformed(evt: ActionEvent): void  -guideJButtonActionPerformed(evt: ActionEvent): void  +main(args: String[]): void  +getInstance(): void |

|  |  |
| --- | --- |
|  | PlayerNames |
| -titleJLabel: JLabel |
| -saveJButton: JButton |
| -player2JLabel: JLabel |
| -player1JLabel: JLabel |
| -name2JTextField: JTextField |
| -name1JTextField: JTextField |
| -jRadioButton1: JRadioButton |
| -jPanel1: JPanel |
|  |
| -jLabel5: JLabel |
| -jLabel4: JLabel |
| -jLabel2: JLabel |
| + pn2: String |
| + pn1: String |
| + checkClose: Boolean |
| + PlayerNames() |
| -initComponents(): void |
| -saveJButtonActionPerformed(evt: ActionEvent): void |

*Figure 6. Class Diagram of the Reversi Project*

+drawRules(int : width, int: heigth)

+paintComponent(Graphics: g)

-image: BufferedImage

-height: int

-width: int

DrawRules

+Rules()

-initComponents(): void

-jButto1nActionPerformed(evt: ActionEvent): void

-jPanel1: JPanel

-jButton1: JButton

Rules

In MenuComponent package, we focus on handling the first menu when starting a game. Main class in this package is MenuView. It will control every class in this package. Options class will show chessboard image options for user choose. PlayerNames is a class for users who input their name. Almost code in this package we use NetBeans to generate.

extend

-copyBoard(int[][] board, GamePlay gamePlay, int x, int y, int player): int[][]

+decision(int depth, int[][] board, int curPlayer): int

+decision(int x, int y, int[][] board): cond

-int score(int[][] arr, int turn): int

+getOptimalMove(int[][] board, int x, int y): cond

-row : int = 8

-column: int = 8

-max Depth: int = 6

-regionScore: int[][] = Regionsc ore.RegionScore()

Minimax

-botThink: void

+press(): void #playervsBot

+getInstance: PlayervsBot

-instance: PlayervsBot

PlayervsBot

+getRegionScore: int[][]

-row : int = 8

-column: int = 8

-regionScore: int[][] =

{20, -10, 11, 8, 8, 11, -10, 20},

{ -3, -15, -4 , 1, 1, -4, -15, -3},

{11, -4, 2, 2, 2, 2, -4, 11},

{8, 1, 2, -3, -3, 2, 1, 8},

{8, 1, 2, -3, -3, 2, 1, 8},

{11, -4, 2, 2, 2, 2, -4, 11},

{-10, -15, -4, 1, 1, -4, -15, -10},

{20, -3, 11, 8, 8, 11, -10, 20}

RegionScore

+checkPossibleMove(int[][] check, int player) : int

+countPlayerScore(int[][] board): Coordinate

+flipChess(int[][] check, int playrt, int x, int y) : void

+checkEndGame(int[][] board) : boolean

-row : int = 8

-column: int = 8

+arrPossibleMove: ArrayList<cond>

y

GamePla

+resetArray: void

#checkCanMove(int x, int y): Boolean #computeBoard(): void #getRowColumn(int x, int y): int #winner(): void

+getXY(): void

+press(int x, int y)

+actionGame(): void

+newGame():

+running(): void #PlayervsPlayer()

+getInstance: PlayervsPlayer

+Coordinate(int x, int y, int type)

+setXY(int x, int y): void

+seetXYType(int x, int y, int type)

+ x, y, type: int

-stage: ArrayList<Coordinate>

-step : int

-x, y: int

-p1Score, p2Score: int

-boolean: endGame

+boardStage: int[][]

-possibleMove: int[][]

-board: int[][]

-score: Coordiante

-gameplay: Gameplay

-instance: PlayervsPlayer

Coordiante

PlayervsPlayer

+leftDiagonal, rigthDiagonal:int

+horizontal, vertical: int

+x, y: int

Cond

*Figure 7. Class diagram of Core Package*

In the Core package, we focus on handling the back end of this game. PlayervsPlayer and PlayervsBot use singleton pattern because we want to have only one class at a time. The Minimax class is a class that handles the Minimax algorithm. GamePlay is a class to handle all rules game.

|  |
| --- |
| Gui |
| +playervsPlayer: PlayerVsPlayer  +playervsBot: PlayerVsBot  -music: PlayeringAudioFileTest  -menuBar: JMenuBar  -gameOptionsMenu: JMenu  -musicMenu: JMenu  -mainMenu: JMenu  -exitJmenu: JMenuItem |
| +Gui()  -startPlayerVsPlayer(): void  -startPlayerVsBot(): void  -setNewGameJMenu():void  -setBacktoMenu(): void  -setStopMusic(): void  -setPlayMusic(): void  -setExitJMenu(): void  -setMenu(): void  +newGamePlayerVsPlayer():void  +newGamePlayerVsBot(): void |

|  |  |  |
| --- | --- | --- |
| Render |  | |
| -board: ArrayList<Coordinate>  +renderBoard: RenderBoard  +renderChess: RenderChess |
| +Render()  +Render(ArrList<Coordinate> board)  +setBoard(ArrList<Coordinate> board, int p1Score, int p2Score, int  step): void  +winner(int id): void  +noMoves(int step): void  #paintComponent(Graphics g): void |  | RenderBoard |
| -instance: RenderBoard |
|  |
| -RenderBoard()  +paintBoard(Graphics g): void  +getInstance(): RenderBoard |
|  | |

*Figure 8. Class Diagram of GUI Package*

-RenderChess()

+paintScore(Graphics g, int p1Score, int p2Score): void

+paintCurrentMove(Graphics g, int step):void

+paintChess(Graphics g, ArrayList<Coordinate> board): void

+getInstance(): RenderChess

-instance: RenderChess

RenderChess

+xStart: int = 100

+yStart: int = 50

+row = column: int = 8

+size: int = 560

+stepSize: int = 70

+width: int = 70

+height: int = 70

+path: String

+board:String

+grayBoard: String

+greenBoard: String

+defaultBoard:String

+brownBoard:String

+whiteChess:String

+blackChess:String

+winner:String

+suggest: String

+logo: String

+cannotMove: String

+background: String

+music: String

+player1: String

+player2: String

Parameter

The focus of the GUI package is on handling the front end of this game. 2 classes support RenderChess and RenderBoard. Render is an extended class from JPanel. To draw all necessary components, it will pass a graphics g to RenderChess and RenderBoard. The parameter class only saves all constant variables, such as the path of the image, the name of the player or the size of the image.

# Chapter V - Evaluation

From this project, our team have more experience to work with GUI. Our team has experience dealing with multi-+ in Java. The multithread that our team has faced is the Java swing component. This project is also the first project in which I am incorporating competitive knowledge programming experience. That makes the project code quicker and cleaner for me. Another lesson is from this project I see that the way that Java show bug is not clear.