|  |
| --- |
| HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY  SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY  DEPARTMENT OF SOFTWARE ENGINEERING  -----\*-----  hutlogobig.jpg |
| PROJECT REPORT |
| Object Oriented Programing  Lecture:  PhD. Nguyen Thi Thu Trang |
| Students: |
| **Do Quoc An – 20194414 – DSAI-K64**  **Nguyen Hoang Dang – 20194423 – DSAI-K64**  **Ha Vu Thanh Dat – 20194424 – DSAI-K64** |
|  |

|  |
| --- |
| Hanoi, June 2021 |

Table of Contents

[1. Assignment of Members 3](#_Toc73008479)

[1.1. Detail for classed/method 3](#_Toc73008480)

[1.2. Idea/Souce Code 5](#_Toc73008481)

[2. Mini-Project Description 5](#_Toc73008482)

[2.1. Details project requirement 5](#_Toc73008483)

[2.2. Use case diagram and Explanation 6](#_Toc73008484)

[3. Design 7](#_Toc73008485)

[3.1. General Class Diagram 7](#_Toc73008486)

[3.2. Serveral Class Diagram 8](#_Toc73008487)

[3.3. Explanation of Design 9](#_Toc73008488)

# 1. Assignment of Members

## 1.1. Detail for classed/method

* + 1. Package Constant: Save constant attribute of game.
* frameWidth, frameHeight: Size of frame.
* houseSize: Size of house.
* START\_X, START\_Y: Start position of X, Y.
* GAME\_FPS: Frames per second.
* STARTING, VISUALIZING, INTRODUCE, GAME\_CONTENT\_LOADING, MAIN\_MENU, CHOOSE\_MODE, OPTIONS, RULES, PLAYING, GAMEOVER, DESTROYED, PAUSE, SUMMARY: Possible states of the game.
  + 1. Package Calculate:
* Class BossSave: to save state of bossHouse.
* Class PlayerSave: to save score of players.
  + 1. Package GameComponent:
* Class BossHouse: Draw the Boss House and some properties of BossHouse.
* Class ChessHouse: Draw the Chess House and some properties of ChessHouse.
* Both of it inherit the Abstract Class House: HouseID, NumberStone, Population, Shape,…
  + 1. Package Static: Include images and sound of the game.
* Class ImageGame: Load all images of the game.
* Class Sound: Load sound for the game.
  + 1. Class Animation: for creating animation
* Int x, y: coordinates
* Boolean active: State of animation to check if it finished.
* Method changeCoordinates(): Change the coordinates of the animation.
* Method Update(): Check when to show next frame of animation or if animation is finished.
* Method Draw(): Draw current frame of the animation.
  + 1. Class Board:
* Int x, y: Position of board.
* chessHouse: 10 chessHouse from 1->5 and 7->11.
* bossHouse: 2 bossHouse(0,6)
* Method initBoard(): Create and set initial coordinate for Board.
* Method draw(): Draw the house of chess and boss.
  + 1. Class Controller:
* Game game, Player player, Board board: Game component.
* stoneInHand, x\_hand, y\_hand: Image hand.
* Int action\_eat: Eat stone.
* Int action\_pick: Get stone from board. If action\_pick = 1=> pick\_stone, if = 2 => put hand down.
* Boolean action\_drop: Drop stone.
* Method handleGame(): Execute step.
* Method handleCalculate(): Execute step and eat.
* Method setBoardByGameState(): Set Board by game state.
* Method setPlayerByGameState(): Set Player by game state.
* Method addMoreStone(): Add more Stone in Calculate function.
* Method addStone(): Add one stone to each house of next turn player when drop.
* Method getStone(): Get all stone of this house and set number of stone to 0.
* Method moveNextHouse(): Move the next house in player turn.
* Method getNextHouse(): Get the next house id.
* Method drop\_oneStone(): Drop stine in hand. If number of stone in hand is 0 => change the state to CHECK.
* Method dropStone(): get Stone and drop to the board until number of stone in hand is 0.
* Method checkFinalHouse(): Check if cannot pick the stone.
* Method checkEmpty(): Check if house is empty or not.
* Method eatStone(Boss): Eat Stone(Boss).
* Method calculateScore(): Calculate the Score when game finish.
* Method isPlayer1(): Check if the turn is player 1 or not.
* Method nextTurnIsPlayer1(): Next turn player 1.
* Method setTurnToken(): Set token for next turn.
* Method checkBoardPlayer(): Check if all house is empty to add stone.
* Method setCoordinateGetStone(): Set coordinate of hand to get stone.
* Method increaseCoordinate(): Increasing the coordinate.
  + 1. Class Frame:
* Method Draw(): Drawing the screen.
* Method keyboardKeyState(): Check if the keyboard button is pressed. Return True if pressed, else false.
* Override some KeyListener method.
* Method mouseButtonState(): Check if mouse button 1 and 2 are pressed or not.
* Override sone MouseListener method.
  + 1. Class Game:
* BufferedImage (NameAttribute): Image for game.
* Method Initialize(): set Variables and objects for the game.
* Method RestartGame(): Restart game – reset some variables.
* Method UpdateGame(): update game logic.
* Method Draw(): Draw the game to screen.
* Method animation\_1(), animation\_2(): create animation for the game.
* Method LoadContent(): Load all the game’s component (images, sounds,…).
  + 1. Class GameHistory: This class is use for the “Return” option when each player want play again their turn.
    2. Class GameState: to save some important attribute.
* Method GameState(): Save step, boss, house, turn, player, stone,…
  + 1. Class GUI: File to Run the Game.
    2. Class MainGame:
* player1ScoreCount, player2ScoreCount, player1Score, player2Score, choose\_bot: Score of each Player.
* gameOverCount: State of GameOver.
* BufferedImage (NameAttribute): Image of each game’s component.
* gameTime, lastTime: Time of the game.
* gameStateMenu, preState: Current State of Game
* Method Initiallize(): Set the variables and objects for this class, variables and objects for actual game can be set in Game.java.
* Method LoadContent(): Load file for this class, file for game will be set in Game.java.
* Method GameLoop(): Update the Game in specific intervals of time.
* Method gameMenu(): Display Game Menu.
* Method gameRule(): Rules of game.
* Method newGame(): Start new game.
* Method restartGame(): Restart game and reset some variables.
* Method Draw(): Draw the game to Screen.
* Method mousePosition(): return position of the mouse pointer in game frame/window.
* Method keyReleasedGameWindow: Call when keyboard key is released. Eg: Esc: exit,…
* Method mouseClicked: Call when mouse button is clicked.
  + 1. Class Player:
* Method initPlayer(): set up Variables.
* Method setSize(): Set player side.
* Method turn(): Make all sequence of action when player choose house.
* Method auto(): Auto play if overtime.
* Method giveTurnToken(): Give turn for the Controller.
* Method giveTokenRollBack(): Set Token for history.
* Method getStep(): Return Step.

## Idea/Source Code

[*https://github.com/thanglongnamnay/O-an-quan*](https://github.com/thanglongnamnay/O-an-quan)

[*https://github.com/MinhTrung0406/gameOAQ*](https://github.com/MinhTrung0406/gameOAQ)

[*https://github.com/18130158/GameOAnQuan/tree/main/src*](https://github.com/18130158/GameOAnQuan/tree/main/src)

# 2. Mini-Project Description

## 2.1. Details project requirement

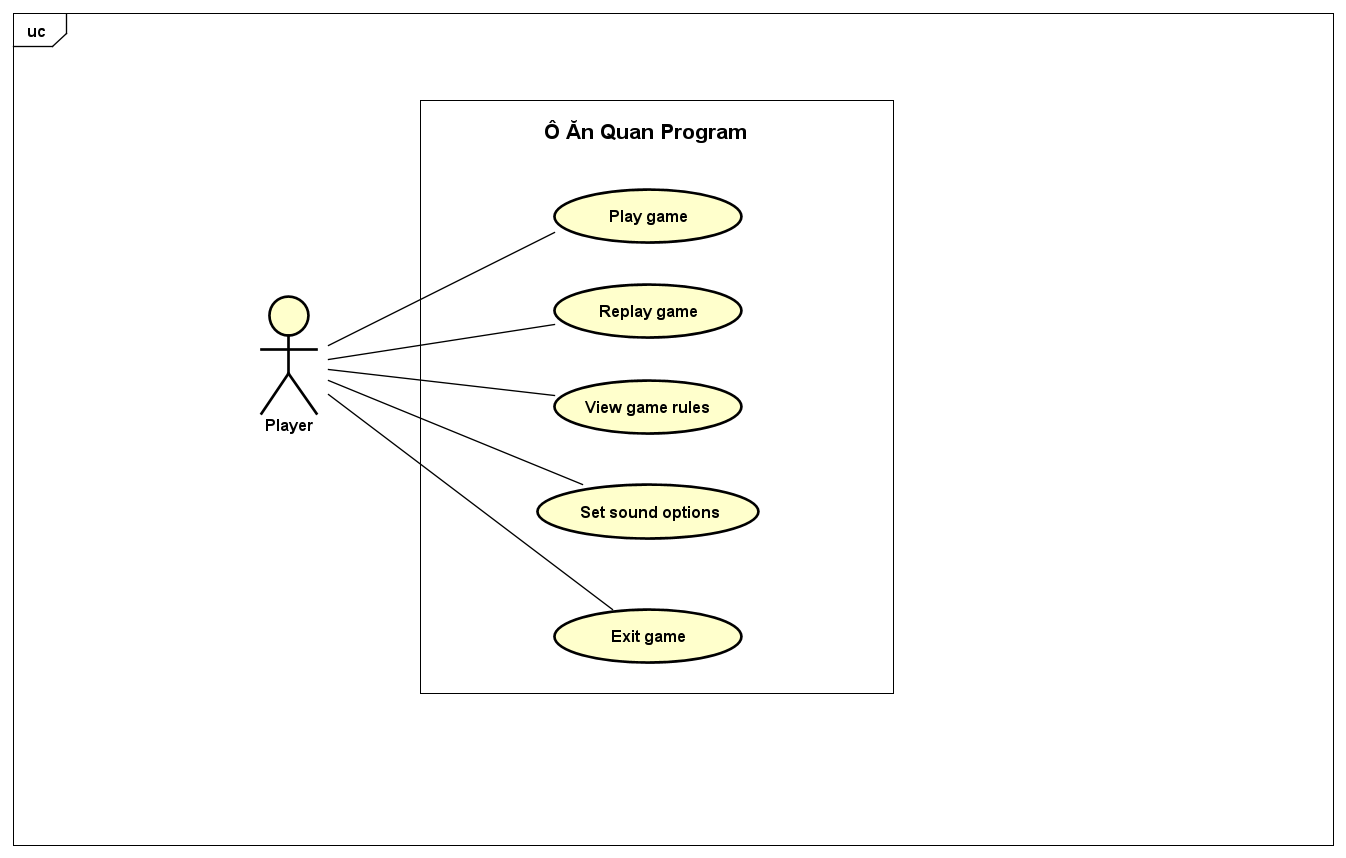
+ On the main screen:

* Start: start the game. For convenient, you do not have to create different difficulties
* Exit: exit the program. Be sure to ask users if theyreally want to quit the game
* Help: Show guide for playing the game

+ In the game:

* Gameboard: The gameboard consists of 10 squares, divided into 2 rows, and 2 half-circle on the 2 ends of the board. Initially, each square has 5 small gems, and each half-circle has 1 big gem. Each small gem equals 1 point, and each big gem equals 5 points.
* For each turn, the application must show clearly whose turn it is. A player will select asquare and a direction to spread the gems. He got points when after finishing spreading, there is one empty square followed by a square with gems. The score the gotfor thatturn is equal to the number of gems in that followed square (see the gameplay for moredetails about streaks)
* The game ends when there is no gem in both half-circles. The application must notify who is the winner and the score of each player.

## 2.2. Use case diagram and Explanation



* Users can play game
* They can replay game after a game ended.
* They can view game rules while they are playing game by clicking on ***HELP*** button on the pause panel or main screen.
* Additionally, users can turn on or tun off game sound by clicking on ***OPTIONS*** button on the main screen.
* Finally, they can exit game by clicking on ***EXIT*** button on the main screen or ***X*** button on the right corner of the game window.

# 3. Design

## 3.1. General Class Diagram

Diagram

Description automatically generated

## 3.2. Several Class Diagram

Diagram, schematic

Description automatically generated

## 3.3. Explanation of Design

The GUI.java class extends JFrame and add the MainGame to ContentPane. The MainGame that controls the game, created it, update it and draw it on screen each iteration. The Game is controlled by Controller, include the Board, the Player. It updates the attributes of Board and Player each iteration and save the previous to GameHistory. The Board contains BossHouse and ChessHouse. The BossHouse contains stone, the boss or empty, the ChessHouse contains stone or empty.

\* Create a JPanel on which we draw and listen for keyboard and mouse events.

Class Frame.java

\* MainGame that controls the game (Game.java) that created it, update it.  
\* Draw it on the screen.

Class MainGame.java:

* Extends the Frame.java abstract class. The Frame.java extends the JPanel.
* Create a Thread and run the GameLoop() method. Each loop, we repaint the screen.
* GameLoop() use enum GameSateMenu:
  + Init state is VISUALIZING: get attributes of the frame and change the state to STARTING.
  + State = STARTING: set initial variables and load files, sounds, images… and change the state to MAIN\_MENU.
  + State = MAIN\_MENU: show the menu of game.
  + State = PLAY: create a new Game and change the state to GAME\_CONTENT\_LOADING.
  + State = GAME\_CONTENT\_LOADING: load files, sounds, images of game and change the state to PLAYING.
  + State = PAUSE: show the pause window.
  + State = OPTIONS: show the options window.
  + State = RULES: show the rules of game.
  + State = SUMMARY: calculate score and show the game result. Change the state to GAMEOVER.
  + State = GAMEOVER: game over. Can play again or turn back to game menu

\* Actual game of the program. Controlled by Controller.

\* Contain Board, GameHistory, and Player.  
\* Update the game logic.

Class Game.java:

Update the game by UpdateGame() function and use the turnToken.  
turnToken use to describe the turn:

* 0 => finish game and summary.
* 1 => turn of player 1.
* 2 => turn of player 2.
* 3 => turn of controller handle player 1.
* 4 => turn of controller handle player 2.
* 5 => turn of animation, prepare for turn of player 2.
* 6 => turn of animation, prepare for turn of player 1.
* 7 => turn of history for RollBack.

\* Controller of the game (Game.java) and player (Player.java).

\* Update the board, player and gameState.  
\* Handle the game with 5 state.

Class Controller (the Hand):

* First state: WAIT
* Handle
  + state = WAIT:
    - Get the step (choose\_Id and direction) of current player and handle then update it and assign to this.selected\_Id and this.direction variable.
    - Set coordinate of the hand to get stone by setCoordinateGetStone() function. Get the number stone is gotten by getStone() function.
    - Assign to stoneInHand variable.
    - Change the state to GO.
  + state = GO:
    - Drop the stone and update the coordinate of Hand.
    - If stoneInHand = 0 => change the state to CHECK.
  + state = CHECK:
    - Use checkFinalHouse() function to check the current\_Id. The function will return an integer number.
      * If result = 0: finish this turn. Change the state to STOP.
      * If result = 1: Get stone and drop continue. setCoordinateGetStone(), getStone(). Change the state to GO.
      * If result = 2. Change the state to EAT.
  + state = STOP:
    - Change the state to WAIT.
    - Set the turnToken to prepare the turn of player1 or player2.
    - Check if game continue or not.
      * If continue:
        + Check if the board empty or not:

If empty => add more stone to the board from score.

* + - * + Add the current GameState to GameHistory.
      * Else:
        + Change the turnToken = 0 => gameOver.
        + Calculate the score of each player.
  + state = EAT:
    - Eat stone and change the state to CHECK to check if stop go, or eat continue.
* The flow of controller: WAIT => GO => CHECK => STOP or GO or EAT =>…STOP => Finish the game.

\* Describe the step of current turn

Step.java

* Include choose\_Id, direction, end number of times you eat stone of current turn (count\_eat\_number).
* Set the winner of game.

\* Save gameState after each turn

GameHistory.java

\* The state of game. Save the information of player and board

GameState.java

* Save information of Player by PlayerSave.java.
* Save information of BossHouse by BossSave.java.
* Save information of ChessHouse by numberStoneSave.

\* Create to save the constant attributes of the game.

Constant.java

\* Load files, sounds, images, fonts…

Sound.java, CustomFont.java, ImageGame.java, AudioAction.java