**HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY**



**SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**OBJECT-ORIENTED PROGRAMMING**

**MINI PROJECT REPORT**

**TRADITIONAL GAME: Ô ĂN QUAN**

***Lecturer***: **Nguyen Thi Thu Trang, PhD**

***Group Number:* 16**

***Group Member:* Dinh Ngoc Lap Thanh 20226000**

**CONTENTS**

[I. Team member and contribution 3](#_Toc168863092)

[II. Disclaimer 3](#_Toc168863093)

[III. Mini Project Specification 4](#_Toc168863094)

[IV. Project Description 4](#_Toc168863095)

[1. Mini Project Overview 4](#_Toc168863096)

[2. Mini Project Requirements 4](#_Toc168863097)

[2.1. Main Window 4](#_Toc168863098)

[2.2. Game Window 5](#_Toc168863099)

[2.3. End Game Window 6](#_Toc168863100)

[3. Use Case Diagram and explanation 7](#_Toc168863101)

[V. Design explanations 8](#_Toc168863102)

[1. General Class Diagram 8](#_Toc168863103)

[2. Package Class Diagram 8](#_Toc168863104)

[2.1. Frame Package 8](#_Toc168863105)

[**2.2.** Game Pieces Package 11](#_Toc168863106)

[2.3. Panel Package 12](#_Toc168863107)

[2.4. Player Package 13](#_Toc168863108)

Team member and contribution

Table 1 shows the list of contribution of each team member to the mini project categorized by their percentage of involvement in each task.

|  |  |  |
| --- | --- | --- |
| **Task** | **Name of team member** | **Percentage of contribution** |
| Use Case Diagram | Dinh Ngoc Lap Thanh | 100% |
| General Class Diagram |
| Package Diagram |
| Project’s source code |
| Report |
| Presentation |

Table 1. Classification of mini project tasks by member’s contribution

Disclaimer

**Claim:** This project does not directly copy and/or copy with modify any part of the source code from any similar project. Some parts of the program use the idea provided on Stack Overflow’s forum.

Mini Project Specification

From this part of the report onward, the writing format will be specified as the example below:

* Class:
  + abc: **abc**
* Attributes/fields:
  + abc: *abc*
* Methods:
  + abc: *abc()*

Project Description

1. Mini Project Overview

In this project, I am designing an application to play the traditional Vietnamese game, Ô Ăn Quan along with an interactive Graphic User Interface (GUI) that allows two players to take turn spreading gems.

Through making the project, I utilize Version Control System, mainly GitHub, to track change and manage different versions of code. Astah UML was also used to design Use Case Diagram, General Class Diagram and Detail Class Diagrams. For the graphic interface, I decided to use Java Swing for its robust set of features and malleability.

2. Mini Project Requirements

2.1. Main Window

* Start Button: for starting the game, this will dispose the main screen and create a game screen
* Help Button: this will create a JOptionPane window that provides users with rules and scoring system of the game and end game conditions.
* Exit Button: this will exit the program after getting a second confirmation from the user.

Figure 1 demonstrates the composition of the main screen.

A screenshot of a computer

Description automatically generated

Figure 1: GUI demonstration of main window

2.2. Game Window

After pressing the Play Button, the user will be asked for the name of the two players playing the game, after which the game screen will be displayed to the user thereby starting the game. The game screen requires the following components and interactions:

* Two players’ information containers that display the name and point of two players, as well as an indicator of whose turn it is.
* A timer label counts down each turn. After 60 seconds, if the player in turn doesn’t perform an action, they are considered to forfeit their turn and the timer will start counting for the other player’s turn.
* A gem label indicates the number of gems left to be spread, initial value is set to be 0.
* A gameboard consists of 10 squares, divided into 2 rows and 2 half circles on 2 ends of the board. Each square contains 5 ‘dan’ and each half circle contains 1 ‘quan’, each quan is equal to 5 dans. The player in turn can pick a square in their respective row and choose to spread it left or right if the square contains more than 0 dan.
* After each turn, the player will get awarded the number of points equal to the number of dan and/or quan that they got after spreading in their turn. If the number of gems in both half circles after collecting is 0 then the game is considered end, and the game screen is terminated.

Figure 2 illustrates the game screen user interface.

A screenshot of a computer game

Description automatically generated

Figure 2: GUI demonstration of game window

2.3. End Game Window

In the end game, a window is created to signify the winner of the game as well as the score of each player.

Figure 3 showcases the end game screen with the following conditions:

* The winner: player 2
* Player 1 score: 17
* Player 2 score: 43

A screenshot of a game

Description automatically generated

Figure 3: GUI demonstration of end game window

3. Use Case Diagram and explanation

Figure 4 shows the Use Case Diagram of the overall program. Regarding the action the users can perform on the program, I have the following:

* In the main window, the user can choose to start the game by clicking on the Start button, see the guideline to playing by clicking on the Help button or choose to exit the program by clicking the Exit button (this will trigger a dialog window that ask for a second confirmation from the user).
* In the game window, the players trade turn spreading gems in their rows by dragging the mouse on a desired square and clicking on a direction to spread gems, either left or right. At any point, the user can also choose to exit the match by closing the game window.

A diagram of a game

Description automatically generated

Figure 4: Use Case Diagram

Design explanations

1. General Class Diagram

Figure 5 presents the general class diagram of the mini project.

A diagram of a computer

Description automatically generated

Figure 5: General Class Diagram

2. Package Class Diagram

2.1. Frame Package

Figure 6 shows the Class Diagram for Frame package

A screenshot of a computer

Description automatically generated

Figure 6: Detail Class Diagram for Frame package

Package Frame is responsible for create the GUI for Ô Ăn Quan Application, with all of its component’s class inherited from class JFrame. The relationship between them is described as follows:

MainFrame: creates the main screen of the application. Class MainFrame implements MouseListener to listen for user interaction (click) to its attribute: startButton, helpButton and exitButton.

*startButton*: dispose the main window and create the game window.

helpButton: method showGuideline will read from guideline.txt file to display a dialog window to the user describing the rules of the game as well as scoring system and end game conditions.

exitButton: create a confirmation dialog window require the user to confirm their choice of exit the program. If the user chooses no, return to the main window.

GameFrame: creates the game screen of the application. The constructor will first initialize its components through packed methods, including:

createTimer(): initialize the timer label for count down each round.

createGemIndicator(): initialize the gemInHand label for indicating the gem left to spread, to be update after the player choose to spread gem.

createGameBoard(): initialize gameboard field as a instance of **GameBoard** class that holds a reference to the current game frame as its attribute.

createGameBoardContainer(): initialize the interactive component of the game window, boardGameContainer, with BorderLayout and holding 12 **MyPanel** instances as tiles. Each tiles[i] will act as a graphic container of the **Tile** object. I will passed in it the respective **Tile** instance in the gameboard object in the form of gameboard.tiles[i], which will hold a hard copy of the tile information. To perform action on the tile GUI, each tiles[i] in GameFrame will be wired to the mouse listener in GameFrame instance. And since we expect different behaviors and layout from square tiles and half-circles tile, the program will down casting them to their respective class.

timerCountDown(): this method will be called firstly when all GUI components are done being initialize and after each turn. It will update the player point and turn indicator according to the point they gain in the previous round and the gameboard.getTurn(). It will then check for end game conditions, being if two half-circles don’t have any gem left. If the game isn’t finished, it will begin counting down from 60 seconds, updating the timerLabel each second. The initial delay will be set to 10ms to prevent the timerLabel to take the passing arguments when initializing its constructor of 1000ms as initial delay.

checkEndGame(): if two half-circles tiles no longer have any gems, the rest of the gems in the upper and lower row will be counted as points for player 2 and player 1 respectively. The game window will be disposed of to create the end game window which accepts two **Player** instances as passing arguments.

afterTurnAction(): after each turn, gameBoard will change turn, checking if upper or lower rows have any gems left; if not, the responsible player will be deducted 5 points to pass in each tile a gem and call timerCountDown() again, starting a new round.

mouseClicked(e: MouseEvent): since the users are not allowed to move half-circles tiles, we will check if the clicked tile is a instance of **SquarePanel** and has a positive number of dan. Player 1 can only move tiles in the lower row and player 2 can only move tiles in the upper row. The program will calculate if the player is wanting to move left or right, disable the gameFrame from perceiving any more user actions, stop the timer and begin spreading gems from that tile by calling gameBoard.spreadGems(i: int, direction: String).

EndFrame: accept two Player object as constructor’s parameters and create end game window. The winner is displayed on the top of the frame and two players’ information will be stored in *player1Info* and *player2Info*, including their name, point and Image Icon.

**2.2.** Game Pieces Package

Figure 7 shows the Class Diagram for GamePieces package.

A screenshot of a computer program

Description automatically generated

Figure 7: Detail Class Diagram for GamePieces package

Tile: hold the tile’s information, including its index in the gameboard as well as the number of dan and quan. Two GUI related fields, *isPointed* and *isCollected*, keep track of whether the tile is being used to spread gems or to collect gems. All attributes can have a getter and setter.

GameBoard: compact all game logic and necessary Ô Ăn Quan mechanics. The constructor takes a GameFrame instance named *parent* as a parameter. This is to keep a reference to the current user interface so GameBoard object can utilize its timer attributes can toggle the GameFrame *setEnabled()* method before and after each spreading and collecting gems action.

* *spreadGems(i: int, direction: String):* for tile that’s not half-circles, we take the number of gems in *tiles[index]* with *index = i,* determine the path of spreading from the given index and direction (*step=*-1 for spreading clockwise and *step* = 1 for counterclockwise), and begin spreading. *parent.timer* is use to synchronize backend passing gem action and tile’s gem passing indication. After finish distributing gems, the function stop the timer and calls *recursiveSpreadGems()* on *startIndex* and *direction.*
* *recursiveSpreadGems(startIndex: int, direction: String):* with index marking the tile one after the last tile to receive gem, we check whether the next tile succeed in not being an outer tiles and have the number of gems larger than 0. If it does, we continue to spread gems. If not, we begin counting total points to add then finish the round by enabling user interaction on *parent.*
* *addPoint():* field *index* hold the index of the tile to check for lack of gem to collect the gems in the tile after that. Each iteration, we move *index* forward and set the last tile isCollected condition to false. If *tiles[index].getDan()* > 0 then we collect gems in that tile. If the number of gems equal 0 or the number of gems in next tile larger than 0, we stop timer and mark the end of that turn by calling *parent.afterTurnAction().*

2.3. Panel Package

Figure 8 shows the Class Diagram for Panel package.

A screenshot of a computer program

Description automatically generated

Figure 8: Detail Class Diagram for Panel package

**MyPanel:** an abstract class that acts as a model for creating square and half-circles panel for the game frame. Each instance holds a **Tile** object and a *gemsIndicator* label.

* *paintComponent(g: Graphics):* Override from **JPanel** class, we set background color to the panel on its conditions (is pointed, is collected) and draw tile’s shape through *drawTiles(g2D)* and draw tile’s gems layout through *drawGems(g2D)*
* *drawGems(g2D: Graphics2D):* to be override by its child classes
* *drawTiles(g2D:* *Graphics2D):* to be override by its child classes

**SquarePanel:** inherited from **MyPanel** class, **SquarePanel** can be further classified in its upper or lower row root by *UoL* attributes. When the *mouseEnter()* method in game frame is called, *showArrow* will be set to true, drawing the arrow shape triangular to signify the user is hovering over this tile.

**HalfCircleSquare:** inherited from **MyPanel** class, **HalfCircleSquare** can be classified by its orientation, left or right.

2.4. Player Package

Figure 9 illustrates the Class Diagram for Player package.

A screenshot of a computer

Description automatically generated

Figure 9: Detail Class Diagram for Player package

**Player:** hold player information: name, point and avatar.

**PlayerContainer:** a visualize panel of **Player** class that displays player information.