**OBJECT-ORIENTED PROGRAMMING AND THEORY - 20222**

Mini-Project Report - Team 04 *(Ô ăn quan - Mandarin Square Capturing)*

*Instructor: Nguyen Thi Thu Trang*

|  |  |
| --- | --- |
| **Member name** | **Student ID** |
| DAU VAN CAN | 20214879 |
| TRAN DUONG CHINH | 20210122 |
| DANG MANH CUONG | 20214949 |
| DOAN NGOC CUONG | 20210141 |

**I. Member assignments**

|  |  |  |
| --- | --- | --- |
| Member | List of contributions | Contribution Percentage (%) |
| Dau Van Can | - Implements the `Player` and `Game` classes for `controls` package  - Make the presentation slides | 20 |
| Tran Duong Chinh | - Make the general class diagram  - Implements the `BoardDrawer`, `PlayerDrawer`, `Drawer`, `GameCanvas` classes for `gui` package | 26 |
| Dang Manh Cuong | - Implements the `board` subpackage in `controls` package  - Make the detailed class diagram for `controls` package  - Test the code for bugs & clean the diagrams  - Write the report & film the demo video | 27 |
| Doan Ngoc Cuong | - Make the use case diagram & detailed class diagram for `gui` package  - Implements the `MainMenu`, `MainWindow`, `HelpMenu` classes for `gui` package  - Write the report  - Make the presentation slides | 27 |

**II. Mini-project description**

*2.1. About the mini-project*

This project consists of an application for 2 users to play Mandarin Square Capturing - a traditional Vietnamese game. A playing bot is not implemented in this program - 2 real humans are required to play this game. The application has a help panel to guide the user how to play the game.

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 a square 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 got for that turn is equal to the number of gems in that followed square.

The game ends when there is no gem in both half-circles. The winner and the score of each player is announced.

*2.2. About the game*

This two-player game consists of one board with 10 squares, divided into 2 rows, and 2 half-circles on the two ends of the board. Initially, each square has 5 small gems, and each half-circle has one big gem. Each small gem equals 1 point, and each big gem equals 5 points. The first player to start the game is chosen randomly.

Each player possesses 5 squares on their side and can start their turn from any of those squares, pick up all gems from this square and spread in either direction: clockwise or counter-clockwise. The player must drop one gem into every cell on their path and continue until there is no gem left in their hand. Starting turns from any of the half circles is prohibited.

After spreading all gems in hand, the square next to the final square on the player’s path is called “terminal”. If “terminal” has gems, then the player can continue using the gems in this square to spread in the same direction. A spread is finished if the “terminal” is empty.

After finishing spreading, if the “terminal” (empty) is followed by a square with gems, the player can earn all gems inside this square. If there is another empty square after the earned square, followed by a square with gems, then the player can continue the streak in the same fashion.

During one turn, the players cannot change their direction at any time.

The score is evaluated by the number of gems earned by each player (a small gem equals to 1 point, and one big gem equals to 5 points).

The game ends if one of the following cases happens:

1. Both of the half circles are empty.

2. If a player is in turn, but all the squares that player possesses are empty.

*2.3. Use cases*

A diagram of a game

Description automatically generated

In this use case, the actor is the player. As the game requires two players to play, two players are represented as two children inheriting the parent (player). The player has 4 use cases: Play the game, view Help, check game status and quit the game.

- Play the game: The user can start a new game instance from the main menu screen. Then, the application displays the game board; the players play the game in accordance with the rules stated until the game ends.

- Check game status: The players can easily check how the game progresses with real-time score displaying for each player.

- View Help: The user can choose to view the help guide for the game by choosing the Help button from the main menu screen. The guide is available in both English and Vietnamese.

- Quit the game: The player has two ways to exit the game: in the main menu screen (the Exit button) or during the game (the Quit button on the game board). In either case, a confirmation dialog will be displayed to confirm the user's intention. The user can click OK to confirm their intention, or click Cancel should the user changes their minds.

It's worth noting that choosing Quit from the game board will only take the user back to the main menu. To completely exit the program, the user should use the Exit button on the main menu screen.

**III. Design**

**1 . General class diagram and classes relationship**

**A diagram of a computer program

Description automatically generated**

There is a clear distinction between two packages of the program: the *controls* package - which represents the inner working parts of the program; and the *gui* package - which contains the Graphical User Interface for the program. There is also a MainProgram class which is our entry to the program.

The *controls* package contains two subpackages: *player, board* and the *Game* class which is the central point of the game.

**Relationships**: For the *controls* package, The abstract *Stone* class has two child classes inherited from it: *BigGem* and *SmallGem*; in turn, this *Stone* class are aggregated by the abstract *BoardCell* class (with two child classes: *SmallBoardCell* and *BigBoardCell*). The *Board* class are then aggregated by the *BoardCell* class, which will become the game board for players to interact with. The *Player* class is associated with *Board* and *BoardCell* class, and has a Composition relationship with *Stone class*. The *Game* class has a Composition relationship with both *Player* and *Board.*

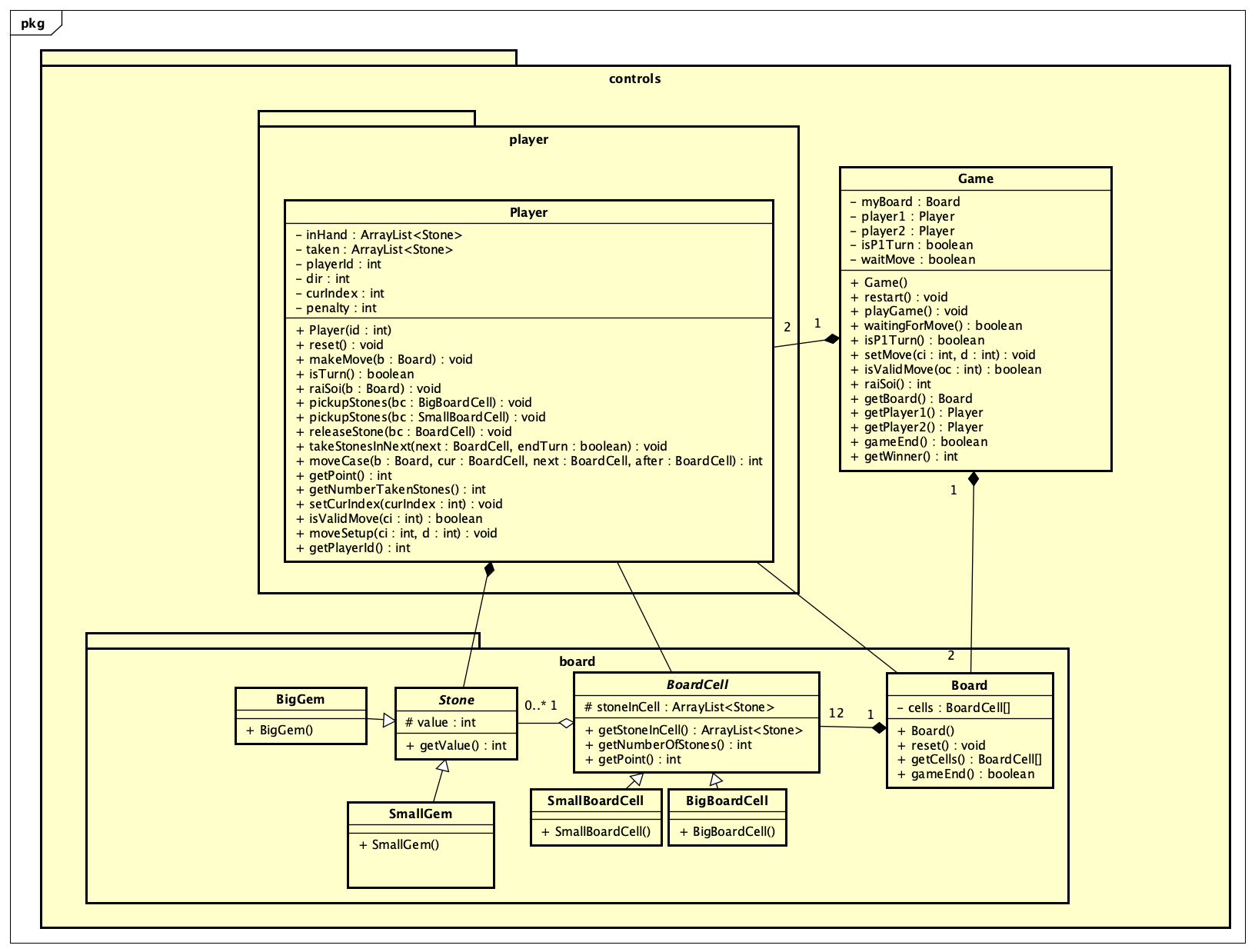
For the *gui* package, *MainWindow* class has a Composition relationship with *MainMenu, HelpPanel* and *GamePanel*. The *GamePanel* class has a Composition relationship with *PlayerDrawer* and *BoardDrawer*, both of which are inherited from *Drawer*.

The *MainProgram* class is aggregated by *Game* and is composited by *MainWindow.*

Further details on the classes will be discussed in the following sections.

**2. Detailed class diagram and classes relationship:**

*\* controls Package*

**

*a) board Subpackage*

*- Stone class: the basic class to create stones in the game. This class has a property `value` to store the value of the rock and a method `getValue()` to return the value of the rock.*

*- SmallGem class: class inheriting from Stone class to create small stones. This class initializes the value of stone to 1.*

*- BigGem class: class inheriting from Stone class to create large stones. This class initializes the value of stone to 5.*

*- BoardCell class: class to create tiles on the chessboard. This class has a property `stonesInCell` to store a list of stones in the cell and a method `getStonesInCell()` to return a list of stones in the cell, a method `getNumberOfStones()` to return the number of stones in the cell. , and the `getPoint()` method to calculate the total score of the stones in the cell.*

*- Board class: class to create chessboard. This class has a property `cells` to store a list of tiles on the chessboard and a `getCell()` method to return the cell at a specific position, a `move()` method to move the stones from one cell to another, and the `getQuanPlayer()` method returns the current player.*

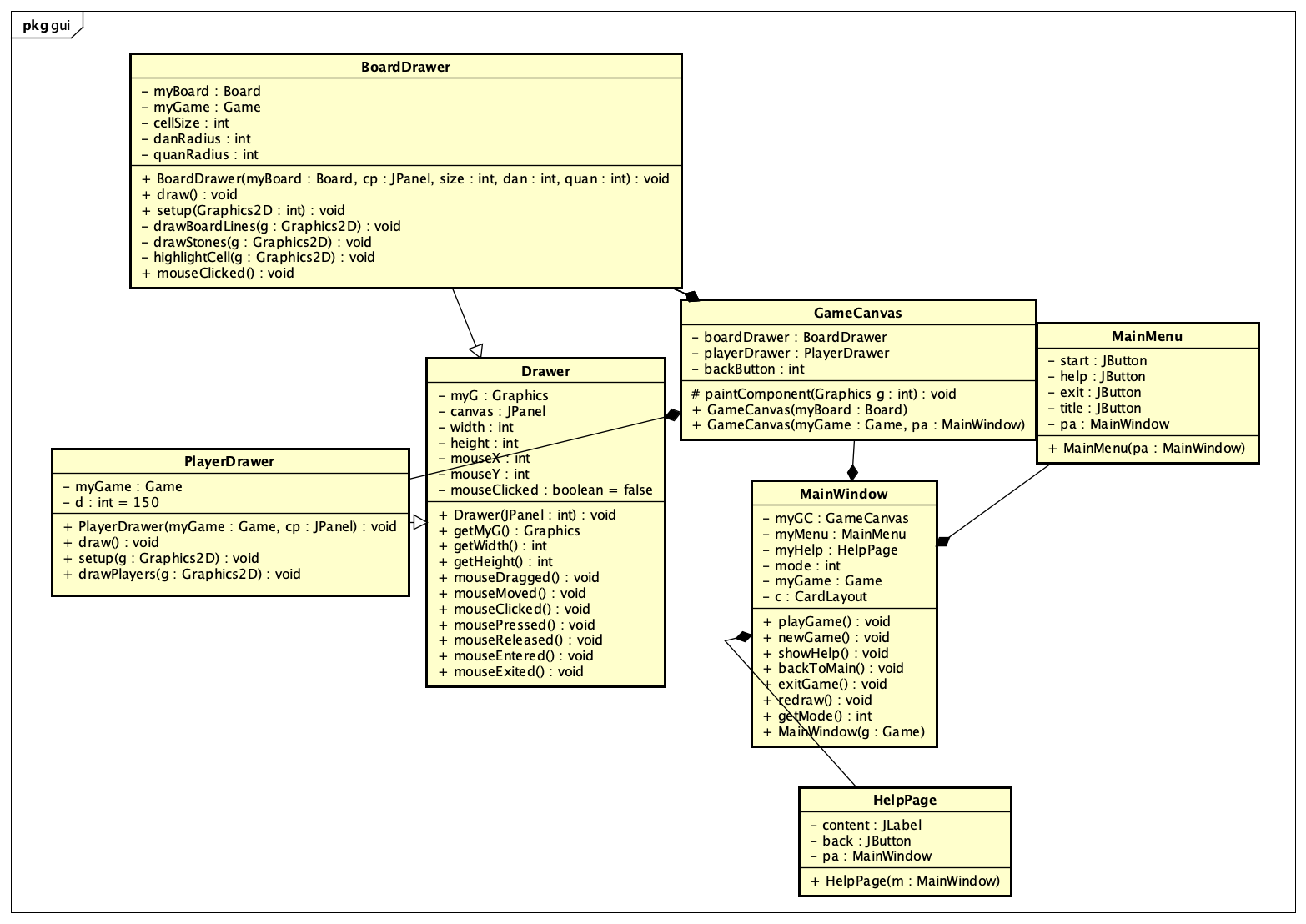
*b) player Subpackage*

*- Player class: class to create players. This class has a property `name` to store the player's name and a method `getName()` to return the player's name. ### Package guide - BoardDrawer class: class to draw chessboard. This class has a `draw()` method to draw a chessboard and related objects.*

*c) MainProgram class*

*- MainProgram class: the main class of the game, containing the `main()` method to initialize the game and interface. In the `main()` method, you create a `Game` object and a `MainWindow` object, and then use an infinite loop to render the interface and play the game. In the loop, you use the `redraw()` method to redraw the interface and the `playGame()` method to play the game.*

*\* gui package*

**

*- Drawer class: class to draw. This is an abstract class and has no concrete methods.*

*- GameCanvas class: class for drawing chessboards and related objects. This class has a `paintComponent()` method to draw objects.*

*- PlayerDrawer class: class to draw players. This class has a `draw()` method to draw player information.*

*- HelpPage class: class to display the help page. This class has a `display()` method to display the help page.*

*- MainMenu class: class to display the main menu. This class has a `display()` method to display the menu and a `getMode()` method to return the selected game mode.*

*- MainWindow class: class to display the main interface of the game. This class has a property `game` to save the state of the game, a method `redraw()` to redraw the interface, a method `playGame()` to play the game, and other methods to handle events .*

***References***

- Game rules *(*[*https://hocvienboardgame.vn/huong-dan-tro-choi-o-an-quan/*](https://hocvienboardgame.vn/huong-dan-tro-choi-o-an-quan/)*)*

- Mini-Project Guidelines *(Object-Oriented Language and Theory, 2023, Nguyen Thi Thu Trang):*