HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION COMMUNICATION TECHNOLOGY

MINI PROJECT REPORT:

Mandarin Square Capturing

Supervised by:

Mr. Hung Tran The

Group members:

Quach Tuan Anh - 20225469

Pham Tran Tuan Khang - 20225503

Dinh Van Kien – 20225505

Nguyen Tran Nghia - 20225452

Hanoi - Vietnam

2024

ACKNOWLEDGEMENT

We extend our heartfelt appreciation to Mr. Hung Tran The for his invaluable guidance and steadfast support throughout the duration of our project of topic Mandarin Square Capturing. Mr. Hung's extensive expertise and knowledge were pivotal in navigating the intricacies of this research endeavor. His encouragement and insightful feedback consistently inspired us to pursue excellence and surmount the obstacles we faced. We are profoundly grateful for his dedication and contributions, which were integral to the triumphant culmination of this project.

ASSIGNMENT OF MEMBERS

The following table is an overview of each team member's contribution and their primary responsibilities:

Members	ID	Roles
Pham Tran Tuan Khang	20225503	Package board, package player, package gem
Dinh Van Kien	20225505	Create and fix fxml, fix models
Nguyen Tran Nghia	20225452	Create and fix fxml, write report
Quach Tuan Anh	20225469	Package exception, design graphic art, write report

Throughout the creation of the object-oriented program, our team engaged in a collaborative effort, offering assistance and utilizing our combined skills and knowledge. Our process started with a brainstorming session to generate ideas related to the program's theme, followed by detailed discussions on class diagrams and mapping out the necessary steps to ensure the project's success. With a solid plan in place, we divided the workload into individual tasks to ensure each team member had a clear focus area. Despite the allocation of tasks, we maintained a supportive and cooperative atmosphere, working together to produce a top-notch final product.

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION.....1

1.1 Mini-Project Description.....1

1.2 Use Case Diagram 2

CHAPTER 2. DESIGN 4

REFERENCES 13

CHAPTER 1. INTRODUCTION

1.1 Mini-Project Description

Rooted deeply in cultural heritage, Mandarin Square Capturing has endured through generations as a test of strategic prowess and intellectual acuity. Our project endeavors to encapsulate the essence of this timeless game within an interactive digital environment, tailored for two players to engage in spirited competition.

Here is an in-depth explanation of the game

- On the main screen:
    - Start: start the game.
    - Exit: exit the program.
    - Help: Show guide for playing the game.

- In the game:
    - Game board: The game board consists of 10 squares, divided into 2 rows, and 2 half- circles 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 (see the gameplay for more details 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.

1.2 Use Case Diagram

The use case diagram for our mini-project is displayed below:

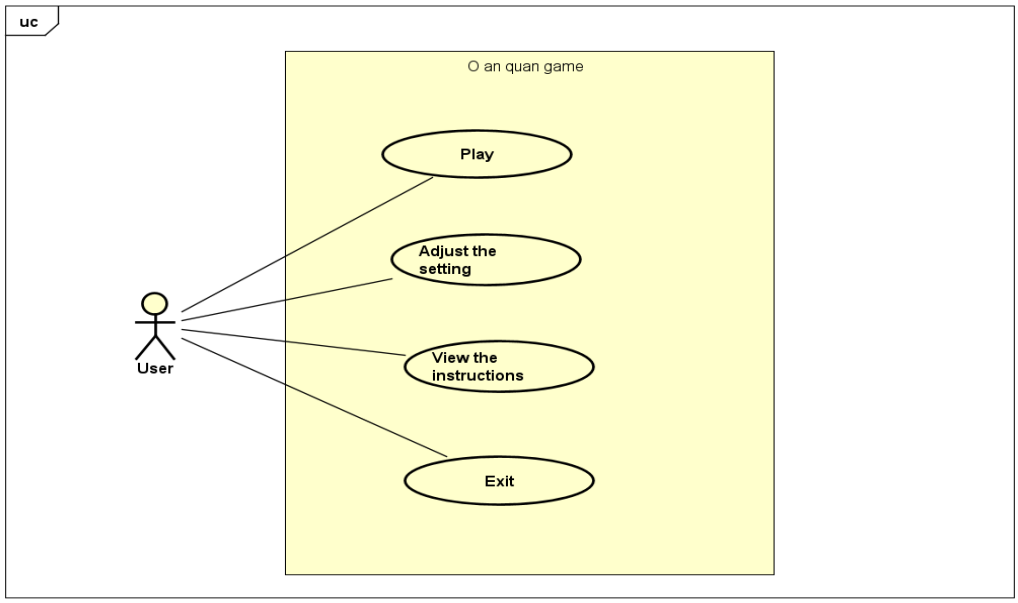

Figure 1. Use case diagram

Play Game: Users can start the game by pressing the Start button on their Home Screen. Once the game starts, they must follow the rules in order to manipulate or acquire gems. Throughout the game, players can access instructions for assistance. Alternatively, they can exit or replay the game whenever they want.

View Intructions: Prior to starting the game, players can read the instructions by selecting the Tutorial button on the Home Screen. This allows them to better understand the game mechanics and improve their performance. If players forget the rules while playing, they can easily review the instructions by clicking the Help button.

Customize Name: After clicking the Start button on the Home screen, players can customize their gaming experience by entering their name. This allows them to leave their own signature while playing the game.

Play Again: While playing the game, users can start a new game if they want to play again or after finishing a match. This feature allows users to extend their enjoyment

and keep playing until they win. They can start a replay by clicking the Replay button while playing, or the Play Again button when the winner screen (end game screen) appears.

Exit Game: This feature allows players to log out of the current game if they do not wish to continue or simply want to refresh the window. Players can stop the program by clicking the Exit button, whether on the initial screen or during gameplay. A confirmation box should appear, providing the player the choice of exiting the program by clicking OK or remaining in the program by selecting Cancel.

CHAPTER 2. DESIGN

Below is our general Class diagram for the mini project with all packages/classes without attributes/operations:

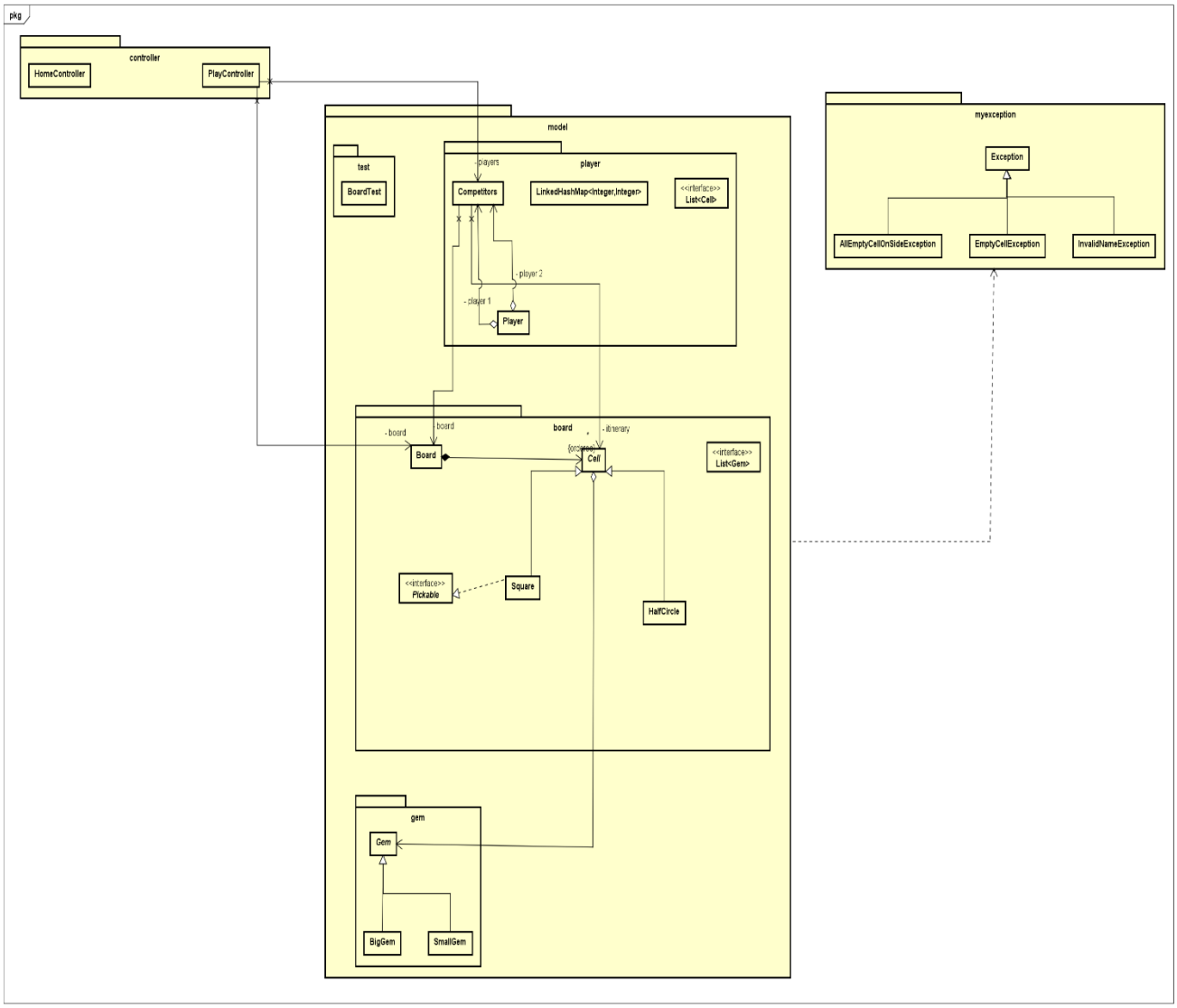

Figure 2. General Class diagram

In our object-oriented program, the class diagram depicts the program's structure. It contains several packages, including '*board'*, '*player'*, and '*gem'* which are all corresponded to game entities.

Different types of relationship can be observed from the class diagram:

Inheritance:

- The classes SmallGem and BigGem inherit from the abstract class Gem.
- The classes HalfCircle and Square inherit from the class Cell.

Encapsulation:

- Classes in package models (like player, board, and gem) use **encapsulation** to protect private data.
- Each class has its fields (attributes) and methods (functions) bundled together.

Abstraction:

- Abstract class.

Association:

- The class Competitors is associated with the Board in a one-to-one relationship.
- Competitors are also associated with Cell.
- The PlayController class is associated with Competitors.

Aggregation:

- Cell aggregates Gem.
- Competitors class is aggregation of player 1 and player 2.

Composition:

- The board is composed of Cell objects

Realization:

- All Square cells implement the Pickable interface.

To provide a more detailed understanding of the classes, we refer to the attached images illustrating the relationships between the classes in the program.

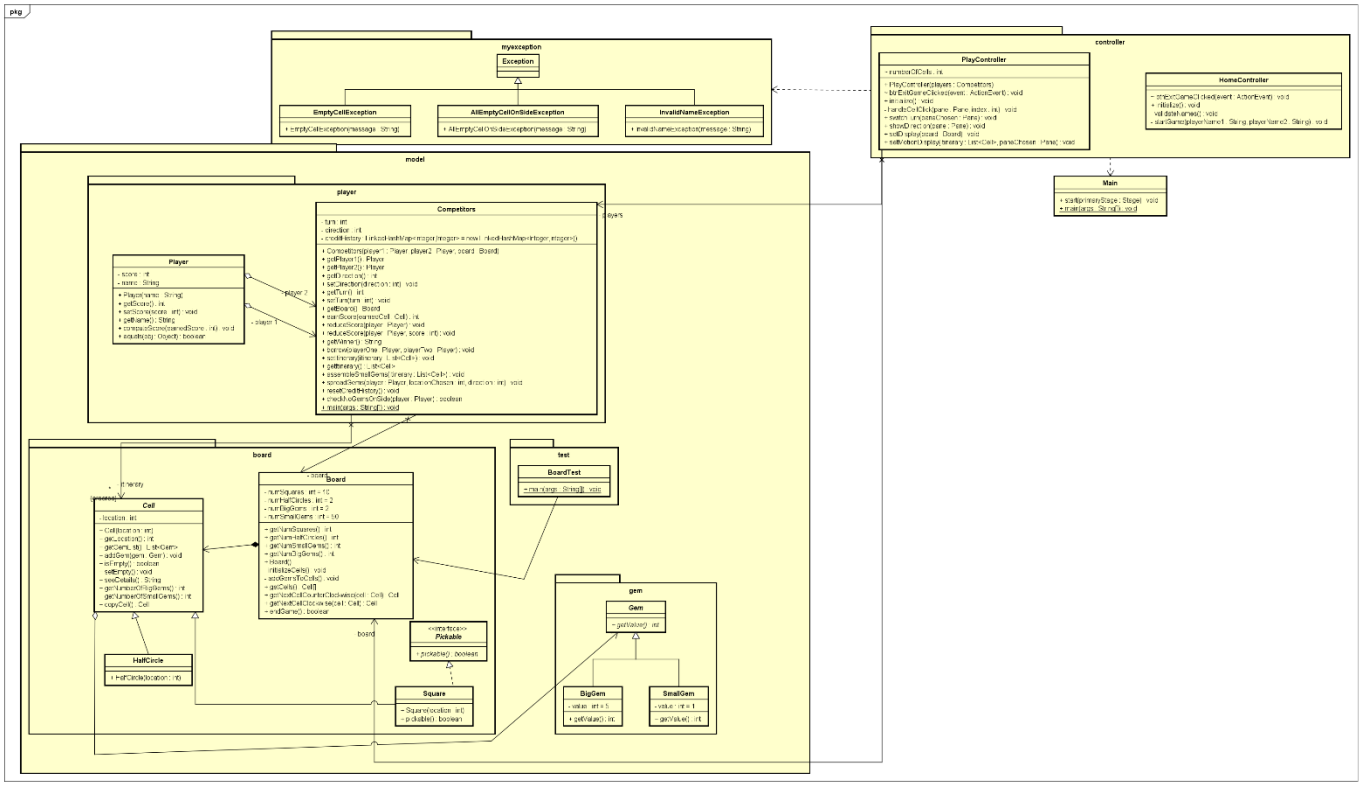

Figure 3. Detailed Class diagram

Detail class diagrams for each package or several packages, with detail attributes/operations for each class and the implements of some important methods.

Class Diagram: Gem

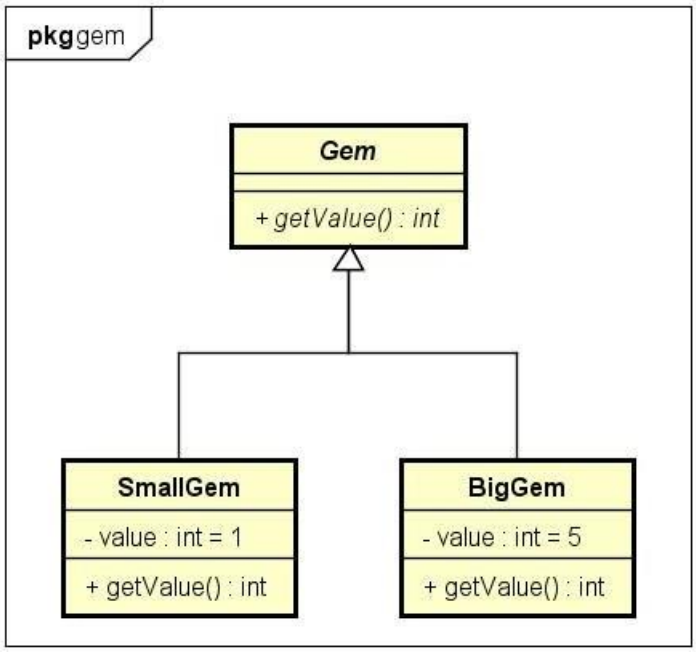

Figure 4. 'Gem' Class

The Gem class is the foundation for the gems that our program's players utilize to interact with other critical classes such as Cell and Board. Each Gem object has a single attribute, location, which represents the cell that holds the gem. This abstract class has two derived classes, BigGem and SmallGem. Both derived classes have an additional attribute named VALUE that determines the gem's value. In our program, big jewels are worth 5 points, while small gems are for 1 point. These points are then transformed into individual player scores. **Polymorphism** is applicable in this context, allowing both BigGem and SmallGem objects to be treated as Gem objects despite having different VALUE attributes. Player scores are calculated using the gem's value, which varies according to the gem type.

Class Diagram: Board

.

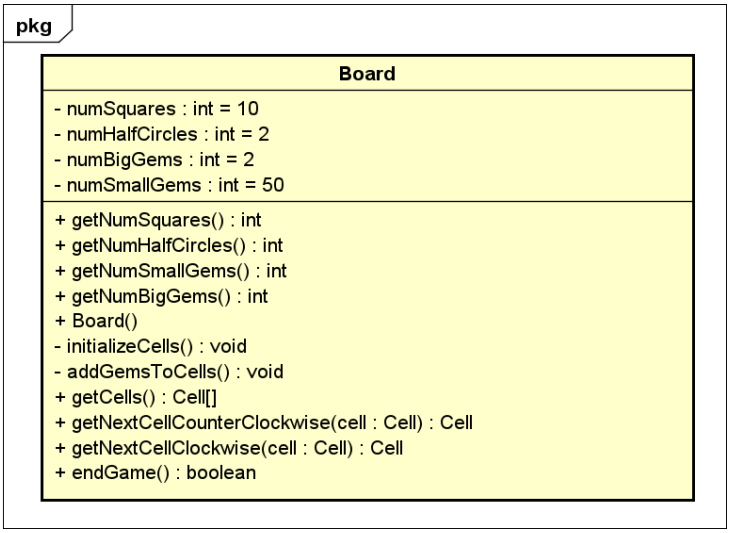

Figure 5. 'Board' Class

The Board class is an essential component of our program, representing the game board on which players interact. It is made up of Cell objects, which can be either half circles or squares. The Board class contains attributes relating to the game's initial state, such as the amount of squares, half-circles, small gems, and big gems. It also keeps a list of cells (the board) for quick manipulation of Board objects.

During the construction of a Board instance, the following methods are used:

- *initializeCells()*: This method creates the required number of square and half-circle cells to build the board.
- *addGemToCells()*: It places gems into the initialized cells. Big gems are initially placed in the half-circles, located at positions 0 and 6 on the board, while the squares will have five gems each.

The *getNextCellCounterClockwise()* and *getNextCellClockwise()* methods are crucial for spreading gems among users.

Class Diagram: Cell

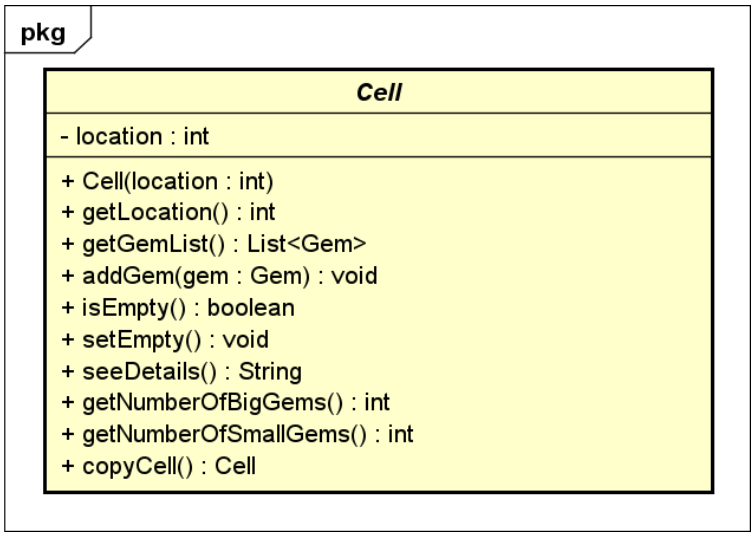

Figure 6. 'Cell' Class

Cell objects are essential components of a Board instance, hence the Board class include the Cell class. Cells are formed based on their position on the board. The Cell class uses the *addGem()* function to add big and small gems to the cell, illustrating **polymorphism**. The *setEmpty()* method is used to empty a cell after transferring all of the diamonds to other cells. To preserve the itinerary, the *copyCell()* method is used to guarantee that no cells in the list reference the current cell. Instead, they are duplicates of cells with the same position and amount of gems.

Both the HalfCircle and Square classes inherit from the Cell class and share three properties: location, numberOfGems, and gemList. Half circles and squares have different responsibilities and positional properties. Half circles appear at the ends of the board and are the only occurrences capable of containing big gems. On the other side, squares can only hold small gems. Only squares are considered pickable cells, hence they must implement the pickable interface.

Class Diagram: Competitor and Player

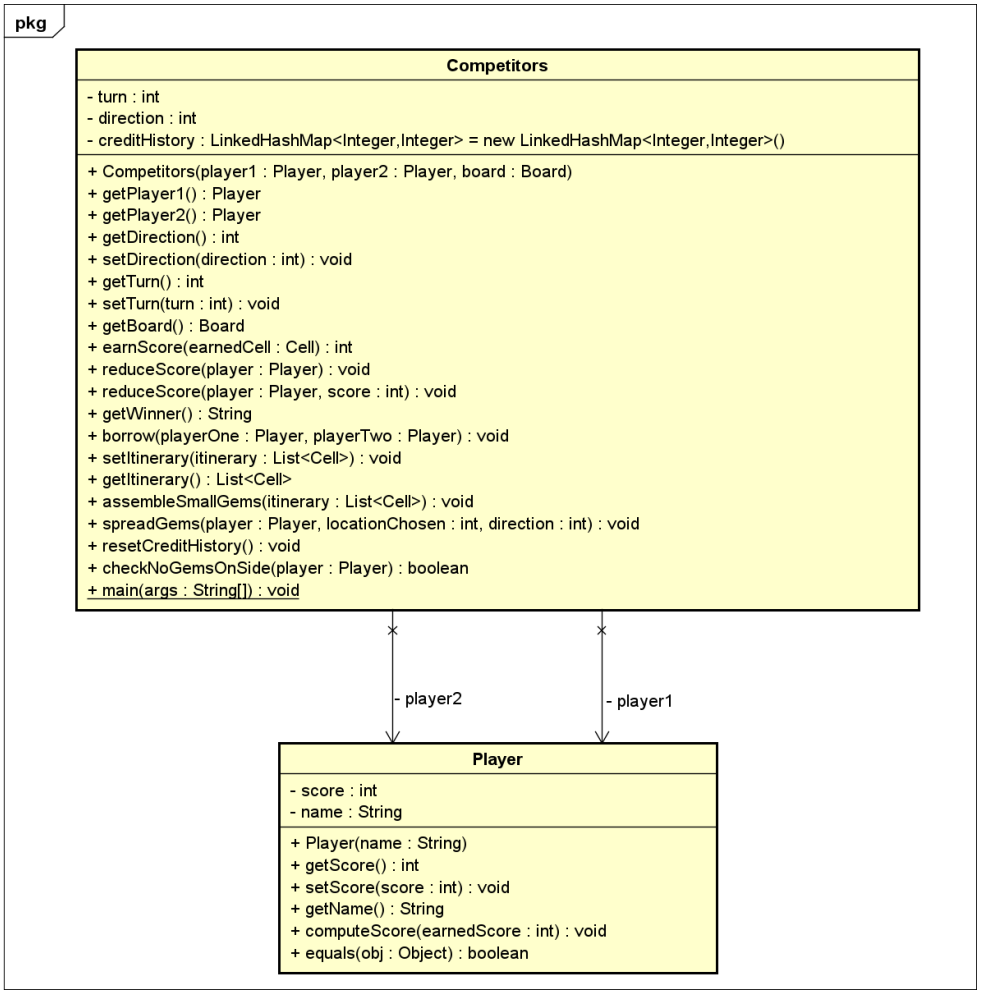

Figure 7. 'Competitors' Class and 'Player' Class

The Competitors class is crucial to the program because it analyzes user data, performs game logic, and communicates with the controller. Competitors are a group of players, such as Player 1 and Player 2. The Player class, on the other hand, stores information unique to each player, such as their name and score. Competitors also maintains track of each player's turn, distributes gems according to the chosen direction, determines the movement direction, and records the itinerary to display the gem's course.

Important methods:

- The *spreadGems()* method of the Player class manages the movement of gems across the game. It accepts inputs such as the spreading direction (0 for counter-clockwise, 1 for clockwise), the player doing the spreading, and the

desired position. This method retrieves the gems from the chosen cell and distributes them in the selected direction. After finishing the gem spread, it checks the following cell after the stop cell to see if the spread should continue or if any gems are earned..

- The *getItinerary()* method retrieves the itinerary of the gem spread for subsequent presentation.
- The Player class additionally offers methods for storing and calculating the player's scores. Additionally, the *equals()* method is modified to compare players based on their names.
- The *reduceScore()* method is overridden in the Player class to accommodate cases in which all cells on the player's side are empty. The default score reduction value is 5. The technique is defined as "reduceScore(player, score)" and can be referred to as "reduceScore(player)".

Class Diagram: PlayController

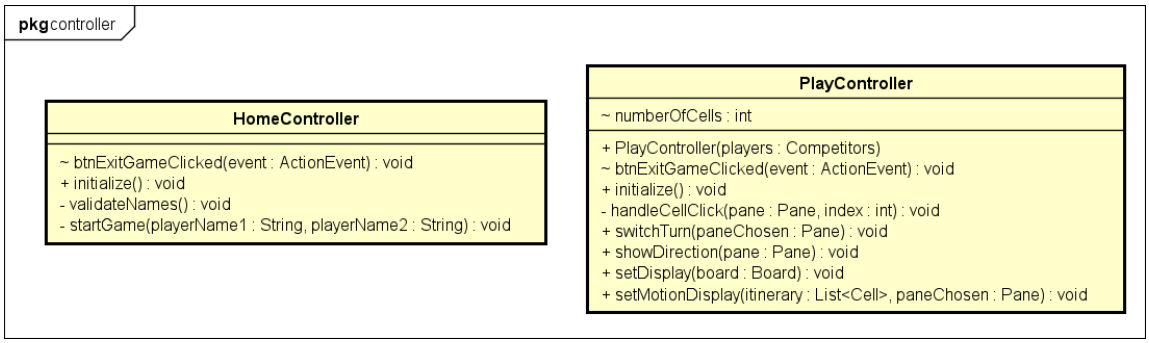

Figure 8. 'PlayController' and 'HomeController' Classes

The PlayController class has methods for controlling data flow between the model and the display. It displays the quantity of gems, scores, and player turns. It also defines the actions for objects like buttons, directions, and cells.

- The *switchTurn()* method in the PlayController class is in charge of checking the game's status using the return result from the *getTurn()* method and showing the current player's turn.

- Downcasting is used to turn Node class instances (which are children of Pane) into Text or ImageView objects. This allows you to customize the gem displays, quantity of gems, and movement direction.

- The *setMotionDisplay()* method animates gems as they spread. This is accomplished using the Timeline package, with each frame representing the changes in the gems recorded in the itinerary.

- Binding techniques are used to update a label below the text field when a user submits their name. The user's name is dynamically shown beside "Player 1" or "Player 2" to signify the active player whenever the name changes.

Class Diagram: Exception

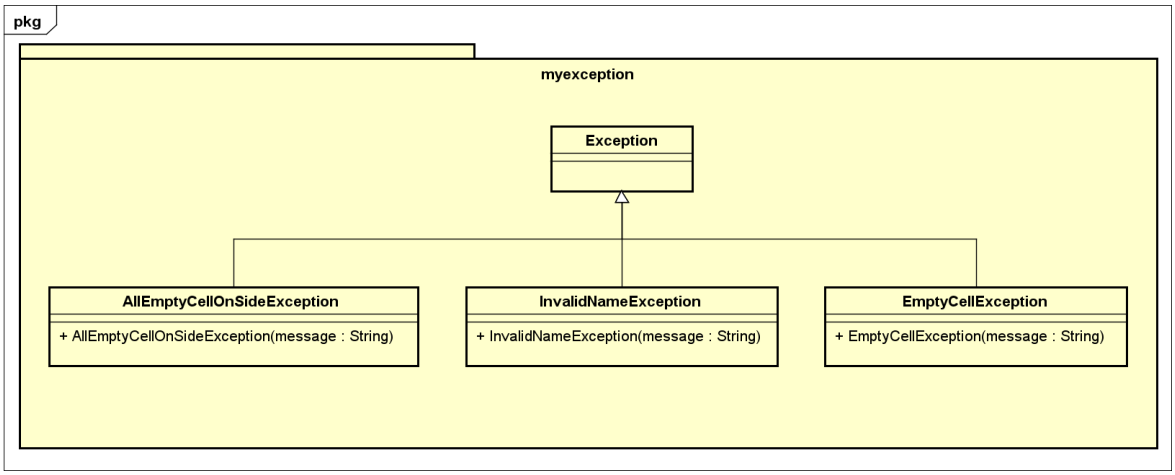

Figure 9. 'myexception' Class

Errors that may arise while the program is being executed are handled by the Exception class. When a user clicks on an empty cell, enters an incorrect name, or tries to distribute gems while every cell is empty, it throws exceptions. Mechanisms for handling these particular problems and delivering the proper error messages or actions are provided by the Exception class. The PlayController class depends on this class which includes exceptions such as InvalidNameException, EmptyCellException, and AllEmptyOnSideException.

REFERENCES

[1] Ideas for design of user interfaces

[Game Ô ăn quan - Dân gian - Game Vui]

[2] Game rules

[https://hocvienboardgame.vn/huong-dan-tro-choi-o-an-quan/]

[3] The video tutorial about using timeline for animation

[(5377) JavaFX and Scene Builder - IntelliJ: Alarm clock with Timeline - YouTube]

[4] Idea about set up gui for game

[https://github.com/Querz/chess.git]

[5] Oracle API for JavaFX with numerous numbers of examples

[https://docs.oracle.com/javase/8/javafx/api/toc.htm]