Hanoi University of Science and Technology

School of Information & Communication Technology

Object-Oriented-Programming
Mini-Project Report

Students: Pham Duc Thanh 20210795

Nguyen Ba Thiem 20214931

Doan Trong Tan 20194831

Le Thanh Thang 20194451

Team 19

Instructor: Nguyễn Thị Thu Trang

ACKNOWLEDGEMENTS

We would like to express our deepest appreciation to Prof. Nguyen Thi Thu Trang who gave us a golden opportunity to work on this project. This work would not have been possible without your tremendous support and guidance.

1. Assignment of members:

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

1. Nguyen Ba Thiem: 40%
2. Pham Duc Thanh: 40%
3. Doan Trong Tan: 10%
4. Le Thanh Thang:10%

Member	Model	View	Controller	Others related works
Pham Duc Thanh	- Board(40%) :board, cell. - Test for board package and player package	- Create Help.fxml - Fix Home.fxml and Play.fxml to suitable with Help.fxml	- HelpScreen-Controller - Fix HomeContr-oller and PlayController	- Use case and several class diagram - Writing report
Nguyen Ba Thiem	- Board(60%) : Half-circle, pickable, square - Player(100 %)	- Create Home.fxml and Play.fxml	- Home-Controller PlayController	- Writing slide - Create project backgroun d and gameplay images
Doan Trong Tan	- Gem package, application			
Le Thanh Thang		- Commit images		- Exception

During the development of the object-oriented program, our team worked collaboratively, providing support to one another and leveraging our collective expertise. We began by brainstorming ideas related to the program's topic, discussing the class diagrams, and outlining the workflow required to execute the project successfully. Once we had a clear plan in place, we divided the project into main tasks, ensuring that each team member had a specific area to focus on. However, despite the task allocation, we continued to assist and collaborate with one another to achieve a high-quality outcome.

2. Description

2.1. Mini-project description: [link video demo O An Quan]

Describe in detail mini-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 they really want to quit the game
  - 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.
  - For simplicity, you do not have to build a bot to play with human

2.2. Use case diagram

Play Game: Users can initiate the game by pressing the Start button on the Home Screen. Once the game begins, they must adhere to the specified rules in order to manipulate or acquire gems. Throughout the game, players have the option to access instructions for assistance. Alternatively, they can exit the game or replay at any time according to their preference.

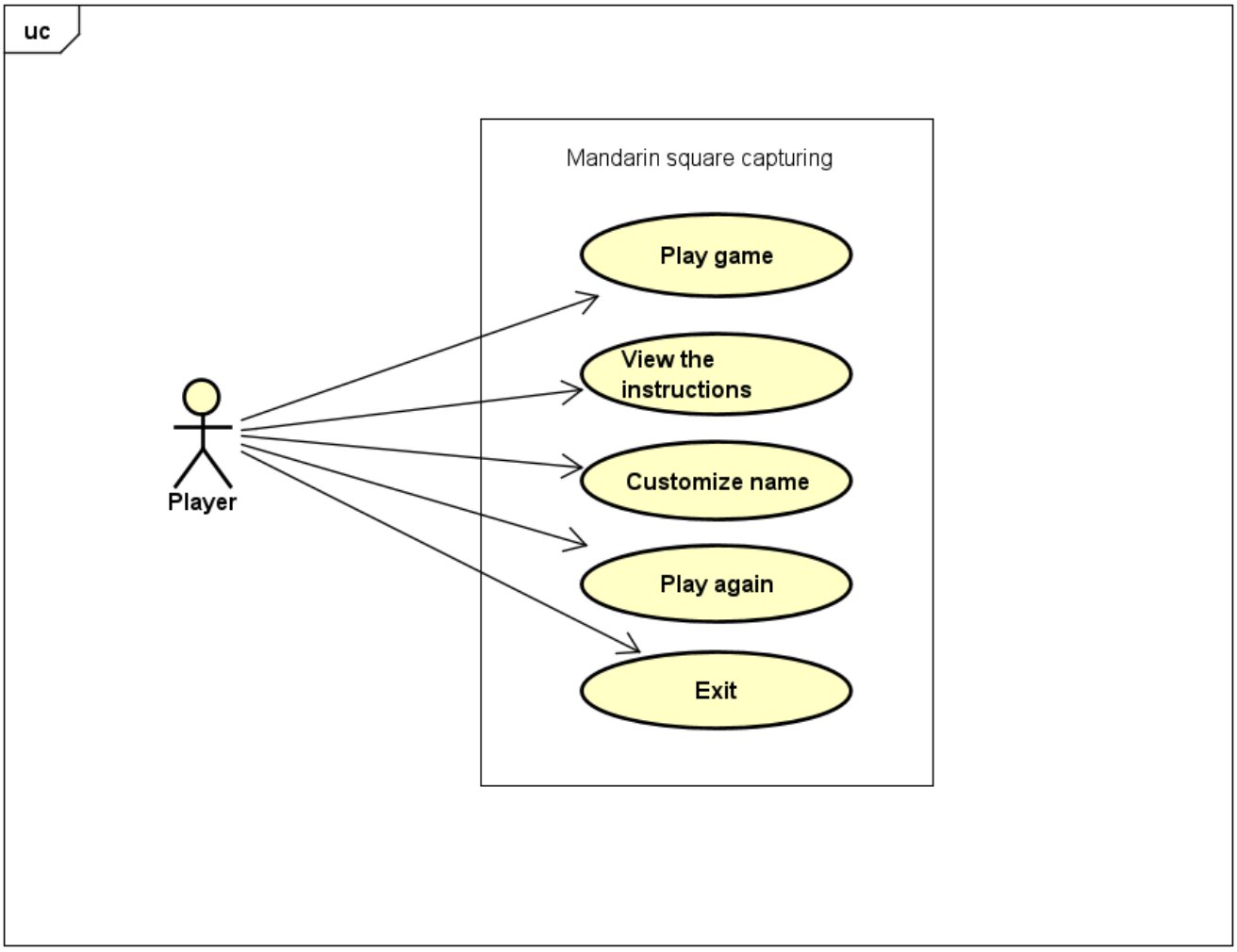
View Intructions: Prior to commencing the game, players have the option to read the instructions by selecting the Tutorial button at Home Screen. This allows them to gain a better understanding of the game mechanics and improve their gameplay. In the event that players forget the rules while playing, they can easily revisit the instructions by clicking the Help button.

Customize Name: Players can personalize their gaming experience by customizing their name after clicking the Start button on the Home screen. This allows them to leave their unique signature while playing the game.

Play Again: While engaged in the game, users can choose to start a new game if they wish to play again or after completing a match. This feature enables users to prolong their enjoyment and continue playing until they achieve victory, for example. They can initiate a replay by

clicking the Replay button during gameplay, or the Play Again button once the winner screen (end game screen) appears.

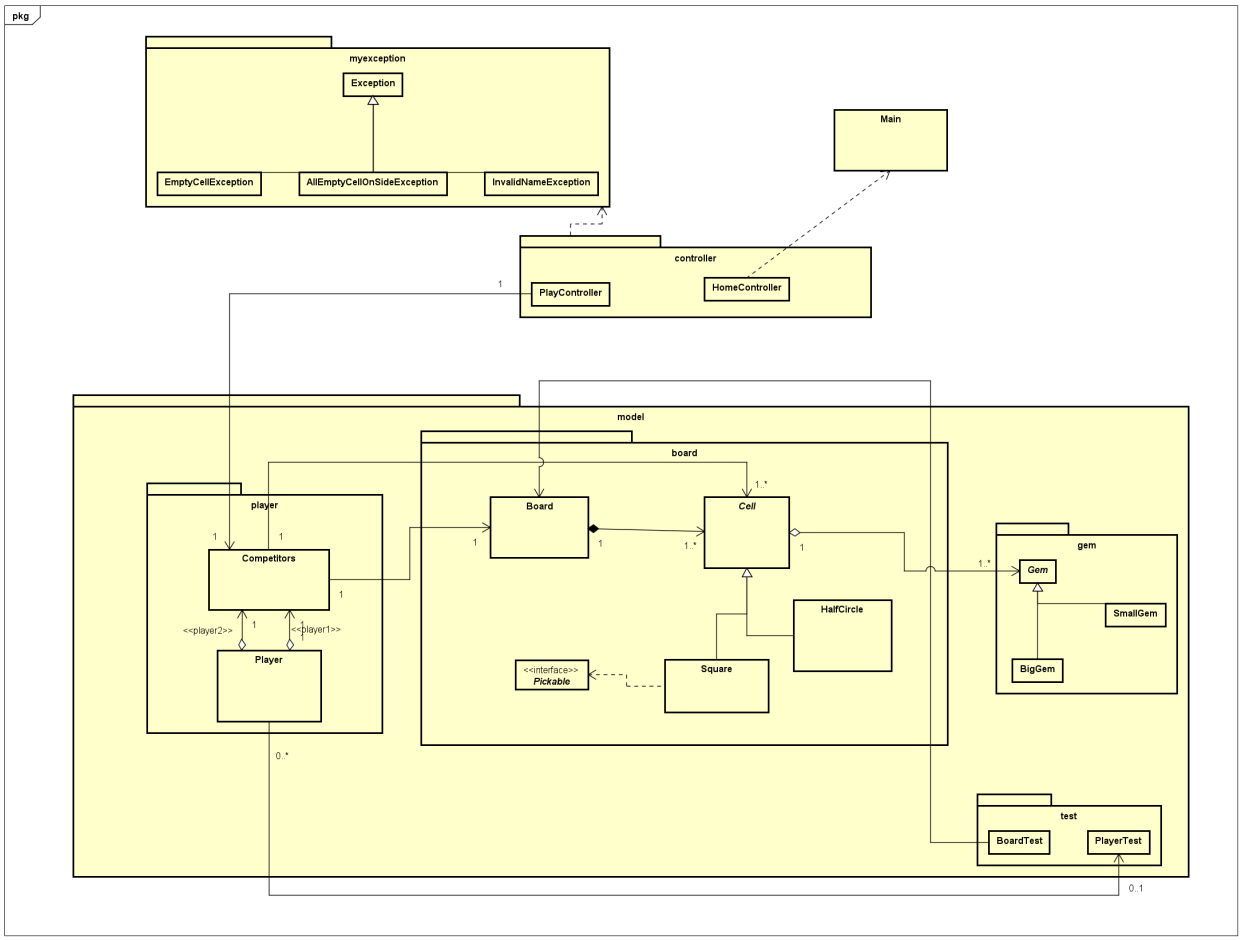
Exit Game: This functionality allows users to log out of the current game when they no longer wish to continue or simply want to refresh the window. Whether on the intro screen or during gameplay, players can click the Exit button to quit the program. A confirmation dialog should be displayed, giving the player the option to either exit the program by clicking OK or remain within the program by selecting Cancel.



Use case diagram

3. Design:

A general class diagram: Class diagram may be with packages, including all classes without attributes/operations:



General Class Diagram

In my object-oriented program, the class diagram provides a visual representation of the program's structure. It includes various packages, such as Model, Controller, and Myexception. The Model package consists of classes related to the game's entities, including gems, the board with cells (both square and circle), and players who interact with the gems and the board. The Controller package contains the HomeController and PlayController, which are responsible for controlling the game flow.

The class diagram demonstrates different relationships between classes:

1. Inheritance:

- The classes SmallGem and BigGem inherit from the abstract class Gem.
- The classes Halfcircle and Square inherit from the classes Cell and Square, respectively.

2. Association:

- The class Competitors is associated with the Board in a one-to-one relationship.
- Competitors are also associated with cells.
- The HomeController and PlayController classes are associated with the Competitors.

3. Aggregation:

- Cells aggregate gems.
- Competitors class is aggregation of 2 player.

4. Composition:

- Cells are composed to form the Board.

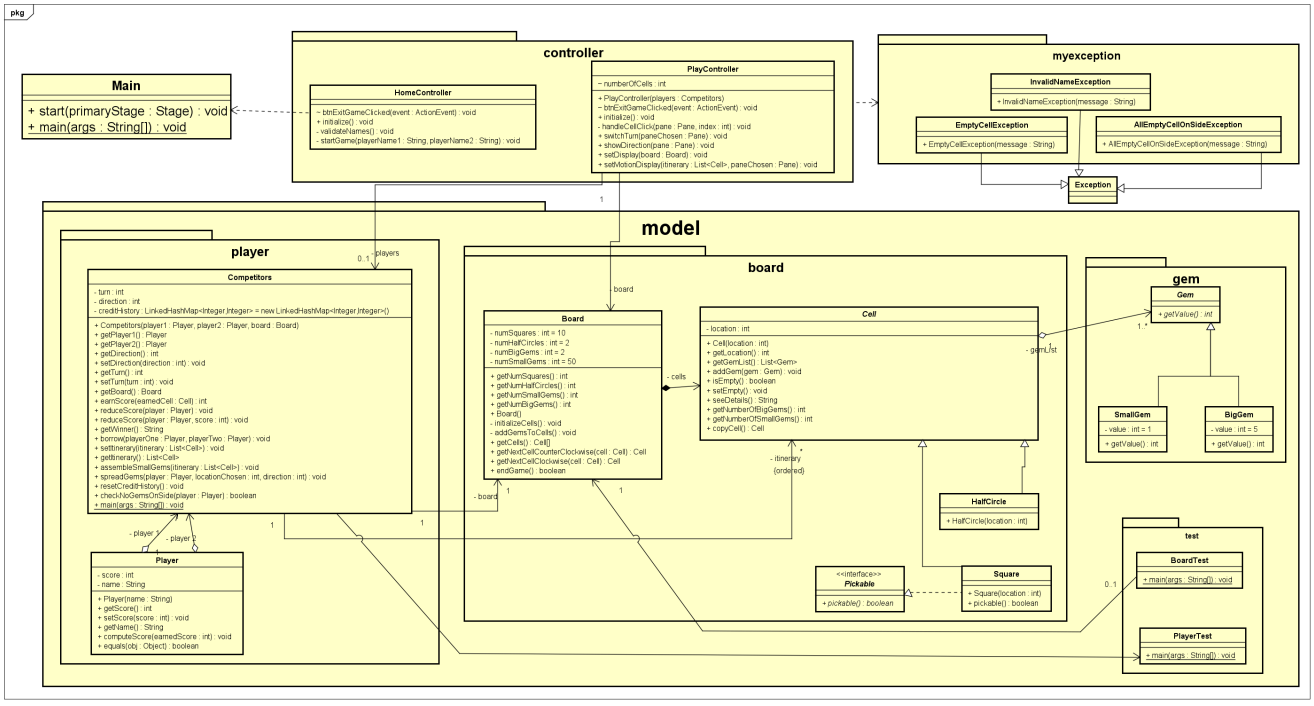
5. Realization:

- All square cells implement the Pickable interface.

6. Dependency:

- The Main class depends on the HomeController.
- The PlayController depends on Myexception, which includes exceptions such as InvalidNameException, EmptyCellException, and AllEmptyOnSideException.

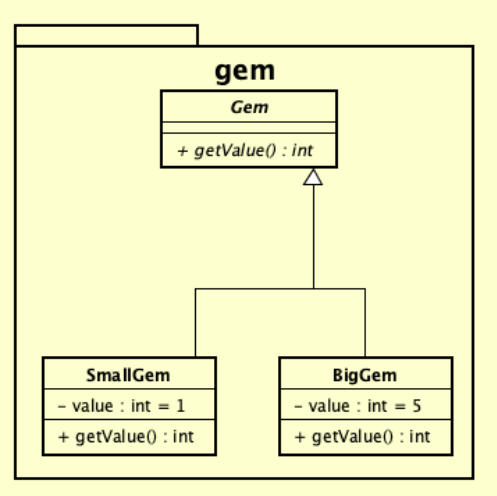
To provide a more detailed understanding of the classes, refer to the attached images illustrating the relationships between the classes in the program.



Detail class diagrams

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

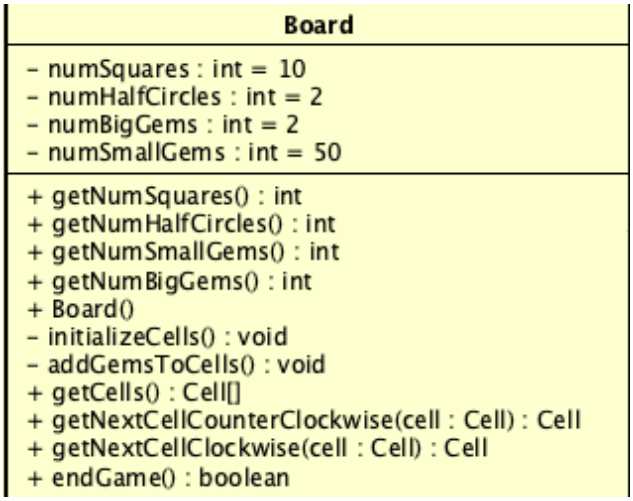
Class Diagram: Gem



Gem class

The Gem class serves as the foundation for the gems used by players in our program to interact with other key classes like Cell and Board. Each Gem object has a single attribute called position, representing the cell that contains the gem. This abstract class has two derived classes: BigGem and SmallGem. Both derived classes include an additional attribute called VALUE, which determines the gem's value. In our program, big gems are assigned a value of 5 points, while small gems have a value of 1 point. These points are later converted into scores for the players. **Polymorphism** is applicable in this context, allowing both BigGem and SmallGem objects to be treated as Gem objects while having different values for the VALUE attribute. Player scores are calculated based on the gem's value, which varies depending on the gem type.

Class Diagram: Board



Board class

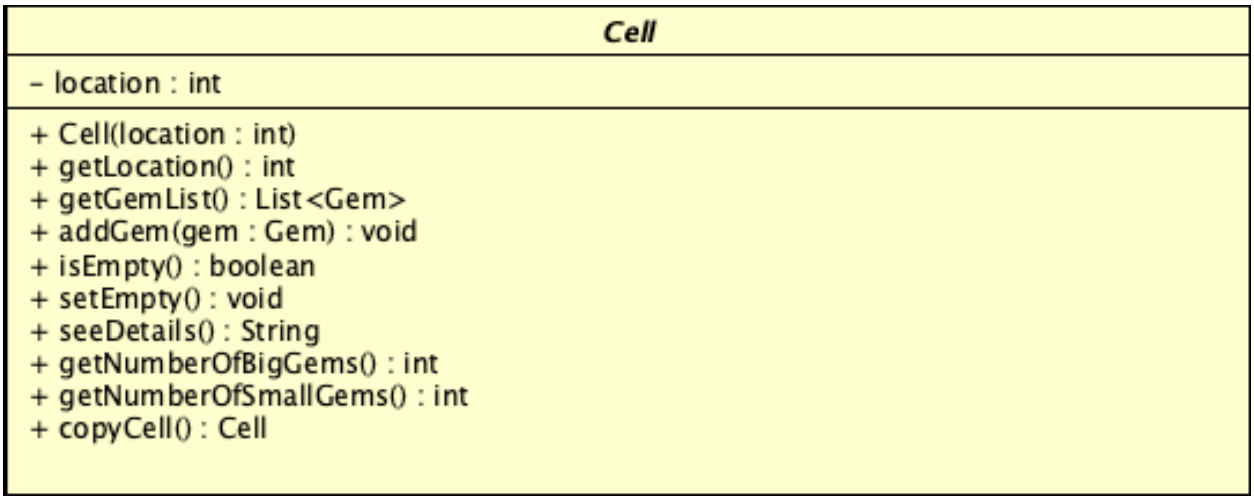
The Board class is a fundamental component of our program, representing the game board where players interact. It consists of Cell objects, which can be either half-circles or squares. The Board class includes attributes related to the initial state of the game, such as the number of squares, number of half-circles, number of small gems, and number of big gems. It also maintains a list of cells (the board) for easy 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

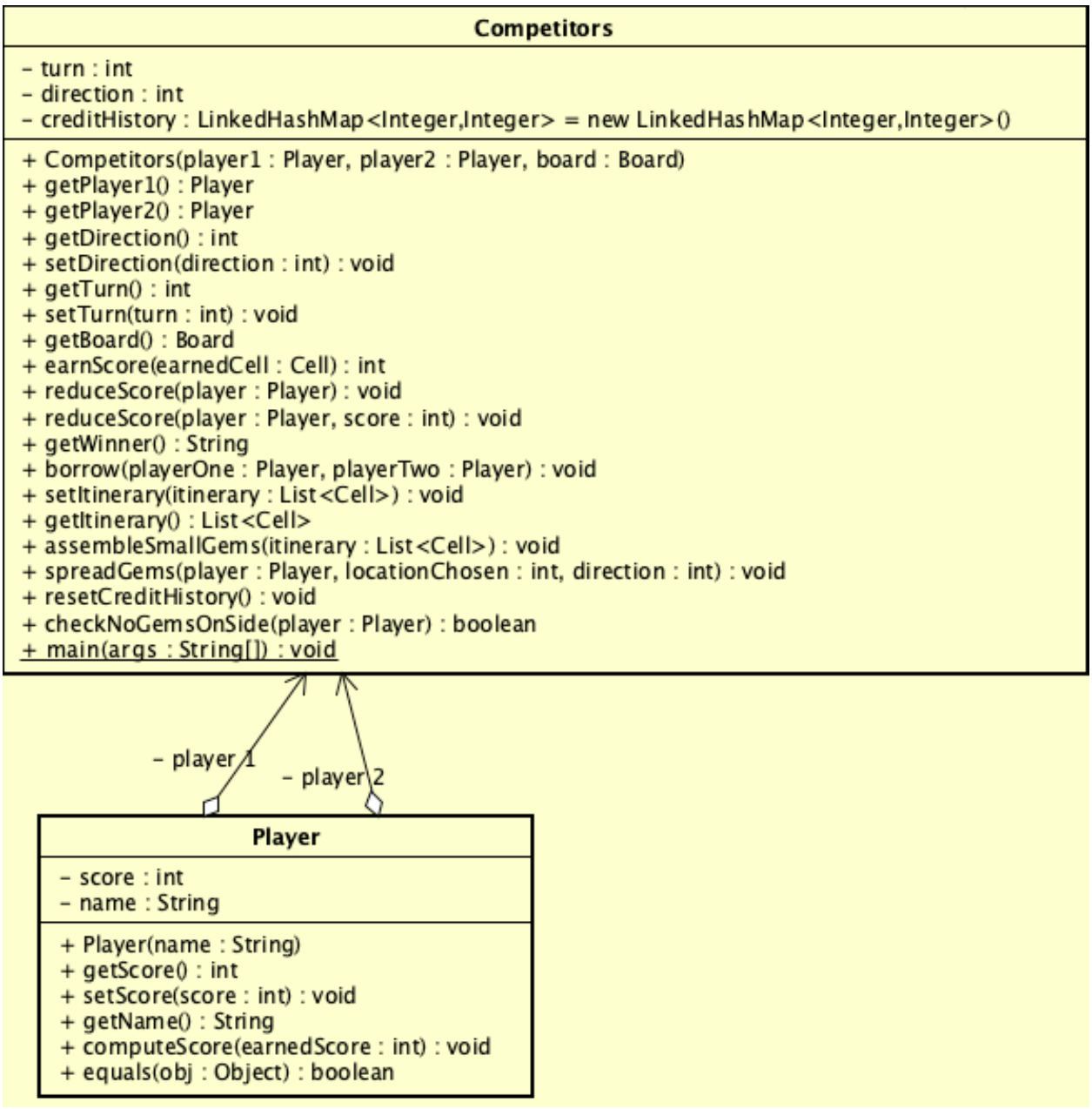


Cell class

Cell objects are integral components of a Board instance, making the Board class composite the Cell class. Cells are constructed based on their location on the board. The Cell class utilizes the addGem() method to add big gems or small gems to the cell, demonstrating **polymorphism.** The setEmpty() method is employed to clear a cell after transferring all the gems to other cells. To preserve the itinerary, the **copyCell()** method is implemented to ensure that the cells in the list do not reference the current cell. Instead, they are copies of cells with the same location and number of gems.

Both the HalfCircle and Square classes inherit from the Cell class and share three attributes: position, numOfGems, and gemList. Half circles and squares have distinct roles and position characteristics. Half circles are located at the ends of the board and are the only instances capable of containing big gems. On the other hand, squares can only accommodate small gems. Only squares are considered pickable cells, requiring them to implement the **pickable interface**.

Class Diagram: Competitors and Player

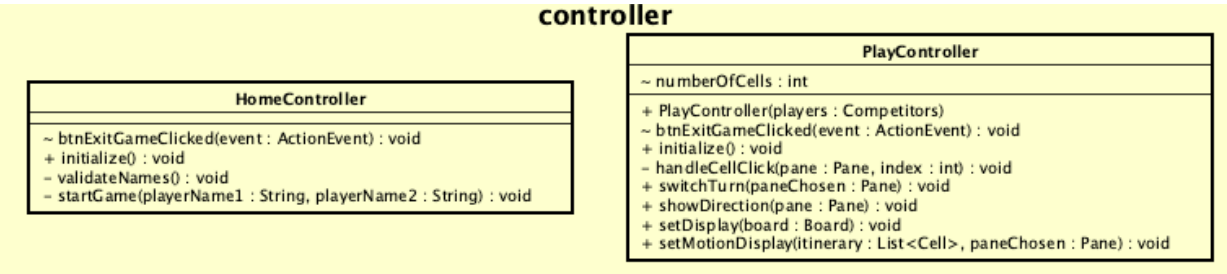


Competitors and Player class

The Competitors class plays a central role in the program as it processes data from the user, implements game logic, and provides information to the controller. Competitors is an aggregation of players, such as Player 1 and Player 2. The Player class, on the other hand, contains information specific to each player, such as their name and score. Additionally, Competitors keeps track of the turn of each player, spreads gems according to the chosen direction, sets the movement direction, and saves the itinerary to display the gem's path.

Important methods:
- The "spreadGems()" method in the Player class controls the movement of gems within the game. It takes inputs such as the spreading direction (0 for counter-clockwise, 1 for clockwise), the player performing the spread, and the chosen location.
  - This method retrieves the gems from the chosen cell and spreads them in the selected direction.
  - After completing the gem spread, it checks the next cell after the stop cell to determine whether the spread should continue or if any gems are earned.
- The "getItinerary()" method retrieves the itinerary of the gem spread for further display purposes.
- The Player class also includes methods to store and compute the player's score. Additionally, the "equals()" method is **overridden** to compare players based on their names.
- The "reduceScore()" method is used by **overloading** in the Player class to handle scenarios where all cells on the player's side are empty. The default score reduction value is 5. The method is defined as "reduceScore(player, score)" and can be called as "reduceScore(player)".

Class Diagram: Controller



HomeController and PlayController class

The Controller class implements methods to control data flow between the model and the display. It handles the display of the number of gems, scores, and player turns. It also sets actions for elements such as buttons, directions, and cells.

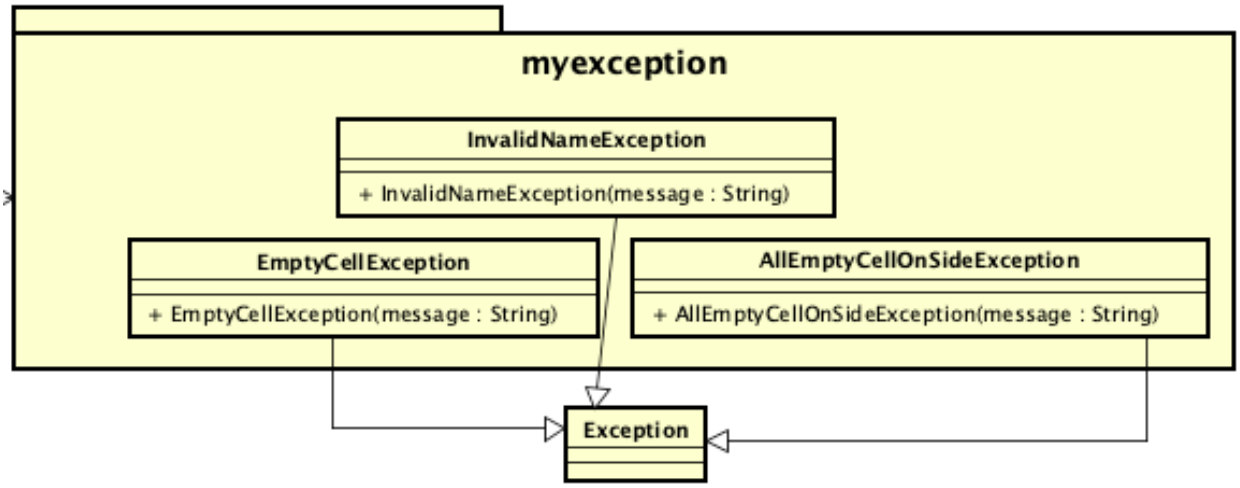
- The "switchTurn()" method in the Player class is responsible for checking the game's status through the return value of the "getTurn()" method and displaying the current player's turn.

- **Downcasting** is used to convert instances of the Node class (a child of Pane) to Text or ImageView objects. This enables the setting of gem displays, the number of gems, and the movement direction.

- The "setMotionDisplay()" method shows the animation of gems while they are spreading. This is achieved using the **Timeline** package, where each frame corresponds to the changes in the gems stored in the itinerary.

- When a user inputs their name, **binding** techniques are employed to update a label below the text field. Whenever the name changes, the label dynamically displays the user's name alongside "Player 1" or "Player 2" to indicate the active player.

Class Diagram: Exception



Exception class

The Exception class handles errors that may occur during the program execution. It raises exceptions when the user inputs invalid names, clicks on an empty cell, or attempts to spread gems when all cells are empty. The Exception class provides mechanisms to handle these specific exceptions and provide appropriate error messages or actions.

4. Reference:

[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]

Demo link: [link video demo O An Quan]