HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



**PROJECT REPORT**

**OBJECT-ORIENTED PROGRAMMING**

**Group:** 02

**Class code:** 141177

**Examiner:** Prof. Nguyen Thi Thu Trang

**Contributors**

|  |  |  |
| --- | --- | --- |
| Student Name | Student ID | Student Email |
| Nguyễn Hoàng Anh | 20214946 | Anh.nh214946@sis.hust.edu.vn |
| Ngô Việt Anh | 20214875 | Anh.nv214875@sis.hust.edu.vn |
| Trần Hoàng Anh | 20210023 | anh.th210023@sis.hust.edu.vn |
| Phan Công Anh | 20210078 | Anh.pc210078@sis.hust.edu.vn |

|  |  |
| --- | --- |
| Name | Task |
| Nguyễn Hoàng Anh | Programming Moves, Gem, BigGem, HelpController and Main classes; as well as the HelpScreen.fxml and Gem.fxml files. Writing the report, designing the power point and class diagram. |
| Ngô Việt Anh | Implement fxml, Style.css, MainPain, HomeDisplayController, GameDisplayController class, write report, design slide and record video demo. |
| Trần Hoàng Anh | Implement fxml, Controller |
| Phan Công Anh | Implement GameModel.Model (to display the game), Utils in UI |

Contents

[1. Problem Overview 2](#_Toc140441638)

[2. Mini Project Overview 3](#_Toc140441639)

[3. Use Case diagram 3](#_Toc140441640)

[3.1. Start game 4](#_Toc140441641)

[3.2. Change setting 4](#_Toc140441642)

[3.3. Play the game 5](#_Toc140441643)

[3.5. Exit the game 5](#_Toc140441644)

[4. Class Diagram 5](#_Toc140441645)

[5. OOP theories implementation 7](#_Toc140441646)

[5.1. Inheritance 7](#_Toc140441647)

[5.2. Association 7](#_Toc140441648)

[5.3. Overloading 7](#_Toc140441649)

[5.4. Overriding 7](#_Toc140441650)

[5.5. Polymorphism 8](#_Toc140441651)

1. Problem Overview

Ô ăn quan is a traditional boardgame which has been one of the most common traditional game for Vietnamese children. The game is simple, easy to play and very valuable for enhancing calcultating and strategical ability for children. The game has been played for thousand of years, but nowadays Vietnamese children no longer had the same passion for game like those in the past. For this circumstance, developers and researchers have been trying to take actions, to preserve and spread this game as a vital cultural heritage, and we as a team has developed a digital version of Ô ăn quan that can be easily accessed on internet, using some basic Object-oriented techniques.

# Mini Project Overview

In this project, we make an application for 2 players to play the traditional Ô ăn quan game. The specification of the game design can be described including:

* **GUI**: The GUI can be freely designed.
* **Design**: The application must have these functions:
  + On the main screen:
    - **Start**: start the game. For convenience, the difficulty is set as default
    - **Exit**: exit the program. Be sure to ask users if they want to quit the game.
    - **Help**: Show guide for playing the game.
  + In the game:
    - **Gameboard**: Gameboard has 10 squares arranged in 2 rows, and 2 half-circles are placed at each end. Initially, each square contains 5 small gems, while each half-circle contains 1 big gem. These gems have point values: each small gem is worth 1 point, and each big gem is worth 5 or 10 points, here in the game is 5 points

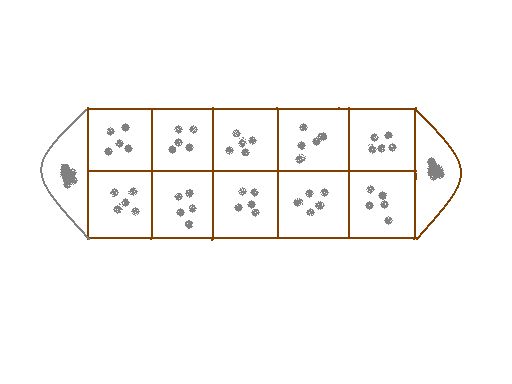


Figure 1: The default gameboard of Ô ăn quan

* + - **Gameplay:** During gameplay, the application must clearly indicate whose turn it is. A player selects a square and a direction to spread the gems, earning points when the spreading leaves one empty square followed by a square with gems (square 1). If another empty square and a square with gems (square 2) follow square 1, the player can also earn points for square 2. This is called a "streak" and can continue indefinitely as long as there are empty squares and squares with gems available. When the next square is a half-circle, or the next two squares are empty, it becomes the other player's turn. If their side of the board is empty, they must use five previously-won pieces to place one piece in each square on their side before repeating the distribution. The game ends when there are no gems left in either half-circle. The application will then notify the winner and display the score of each player.

# Use Case diagram

There are 5 main actions that players can do while playing this game: Start game, Load game, Play game, Guide and Exit the Game. Details of each action are shown below.

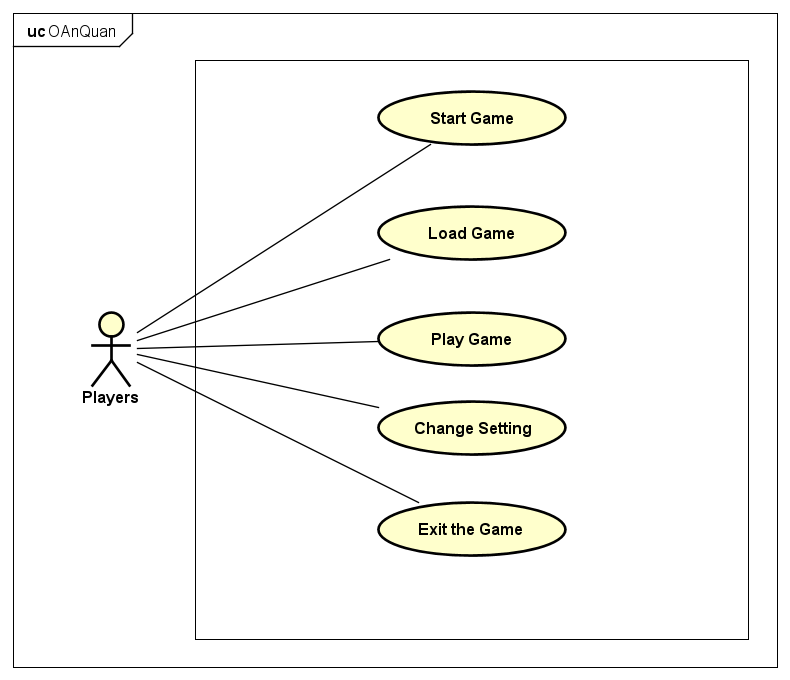


Figure 2: Use Case Diagram of the Application

## Start game

When the game is launched, the user will be presented with a home screen offering 4 options. Clicking the "Start" button will initiate gameplay, while clicking "Load" will load any previously saved games. The "Options" button provides access to customizable settings. Lastly, clicking "Exit" will exit the game, prompting the user with a confirmation message and saving the current game and settings.

## Change setting

Clicking the "Setting" button on the home screen will open a settings screen. At the middle of the screen, there is a list of buttons. The "Help" button opens corresponding screens with helpful information. Lastly, the "Home" button returns the user to the home screen and saves the current game if one is being played.

## Play the game

During the user's turn, they can select one of the five squares under their control. The user can then drag the selected square left or right to indicate the desired direction for the gems to spread. The game will automatically spread the gems and update the score, which is immediately displayed to the players. The next player's turn is also indicated. Once the game ends, a screen will display the winner and the final score of the match.

* 1. Load game

In the main menu, clicking the "Load" button will load a previously saved game, if one exists. To save the current game, the user can click the options button during gameplay and select the "Home" button.

## Exit the game

Clicking the "Exit" button or manually closing the game using the menu bar will exit the game and trigger an alert dialog to confirm the player's choice to exit. Exiting the game will save the current game and the selected settings options.

|  |  |
| --- | --- |
| Class Diagram |  |

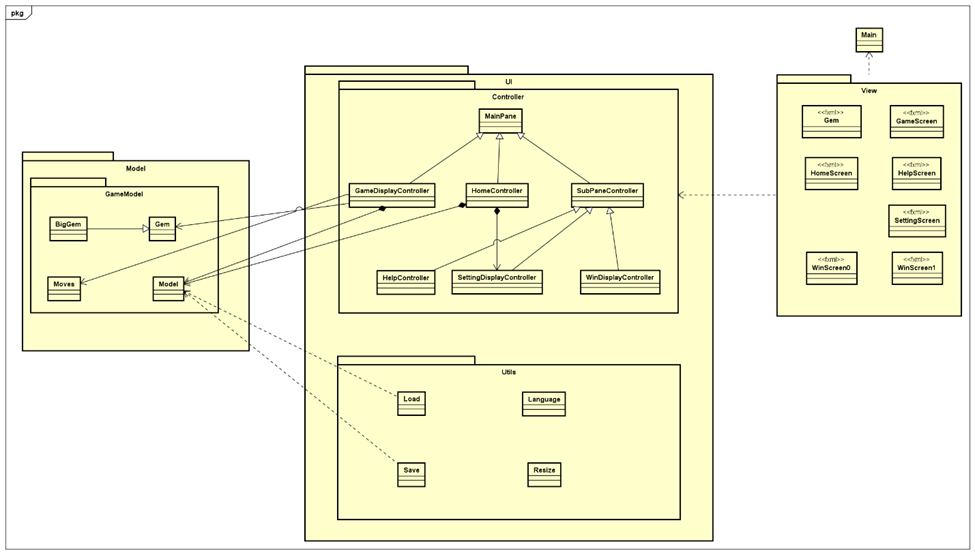


Figure 3: General Class Diagram

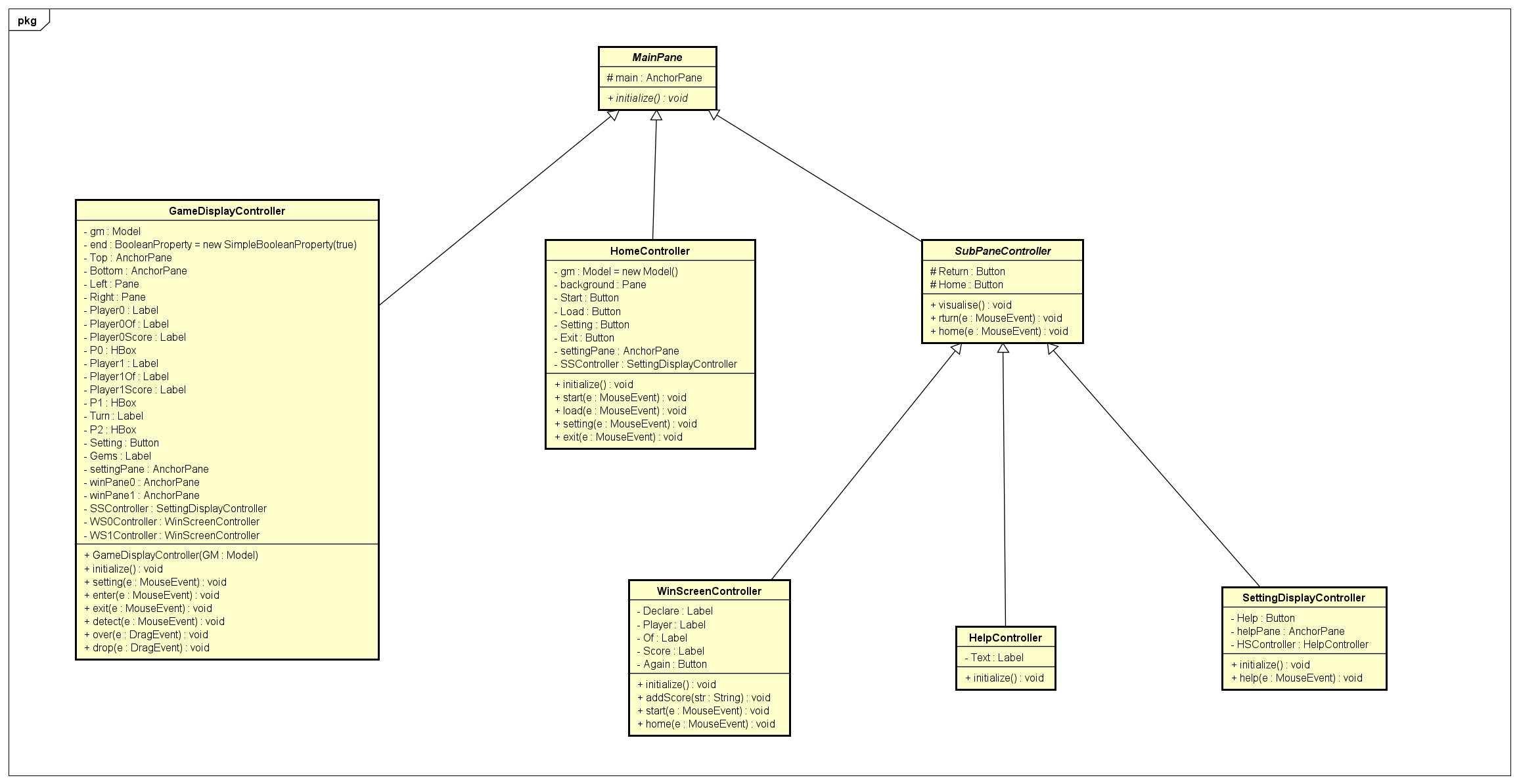


Figure 4: Controller Package Class Diagram

In general, our project uses JavaFX-framework. Main class is inherited from Application class. The start() method was overridden, and the UI is defined by a corresponding .fxml file which is named HomeScreen.fxml, then the behavior of the components is defined in the UI controller class associated with the FXML file. Moreover, the UI controller classes “GameDisplayController” uses all the classes in package “model” to control the game action.

In detail, first, the Main class sets up the JavaFX Application and loads the HomeScreen.fxml file using FXMLLoader class. Then FXMLLoader class creates a Parent node from the HomeScreen.fxml file, which defines the layout and appearance of the home screen. After that, the Main class creates a Scene object from the Parent node and sets it to the Stage object, which is shown to users using the show method. The HomeDisplayController class is the controller class for the HomeScreen.fxml file. This class contains event handlers for the buttons on the home screen: "Start", "Load", "Setting", and "Exit". When the "Start" button is clicked, the start method creates a new instance of the Model class and loads the game screen. When the "Load" button is clicked, the load method creates a new instance of the GameDisplayController class, which is the controller class for the GameScreen.fxml file. The FXMLLoader class is used to load the GameScreen.fxml file and create a Parent node from it. The GameDisplayController instance is set as the controller for the GameScreen.fxml file. The Parent node is set as the root of the current scene, and the game screen is displayed. When the "Setting" button is clicked, the setting method shows the SettingScreen.fxml file by calling the visualise method of the SettingDisplayController class. When the "Exit" button is clicked, an alert dialog is displayed to confirm the user's choice to exit the application. If the user clicks "OK" in the alert dialog, the application is terminated by calling the System.exit(0) method.

# OOP theories implementation

## Inheritance

Inheritance implementation is everywhere in our project. Most of our GUI-related classes are inherited from a class in the JavaFX package. For example, Main class inherits Application class; Gem class inherits Circle; **BigGem** class inherits **Gem** class.

Moreover, **SettingDisplay, WinDisplay, Help** controller inherit the initialize and visualize method from abstract class **SubPaneController**, which inherit from the class **MainPane**. All the Display controllers all have a Pane which is their main container. They also share the initialize and visualize method.

## Association

Associations are shown between GameDisplayController class and Model, Gem as well as Moves class.

## Overloading

In this application, when the user chooses to save the game, the game status will be saved. It can be loaded when the user clicks the load button. With the same technique, the setting options are also saved every time they close the game and will be loaded when the application runs again. The method **save** is overloaded to save the game status and the setting option in one method name. The method **load** is also overloaded to load back the saved information when needed.

## Overriding

Main.java class overrides the method “start” from class Application to customize the UI of our App.

BigGem class overrides the method “bind” from class Gem to adjust the size of the gem to be bigger.

## Polymorphism

When the Bind method is invoked in the Resize class, it creates an instance of either Button or Label, which is then upcasted to the parent class, Region.

Moreover, the Bind method in Gem and BigGem class works differently in these 2 classes despite being the same method in classes.

The class **MainPane** has an abstract **initialize** method. Inherited classes from **MainPane** all have their own **initialize** method implementations to perform their own initialization routines, such as binding the controls to the pane or binding the language localizations to the correct labels.

## Encapsulation

The most of classes in our project use **encapsulation** to protect private data. The properties of these classes are usually declared as private or protected, accessible only through public getter and setter methods. This ensures that data is only accessed and changed according to the rules defined by the class. The classes in the UI package similarly use encapsulation to manage and protect UI elements. The Board, Player, and Game classes have public methods to access and change data in their respective objects. For example, Player has getScore() and setScore() methods to access and set player's score.

***References:***

- [*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)*

*- https://docs.oracle.com/javase/8/javafx/api/toc.htm*