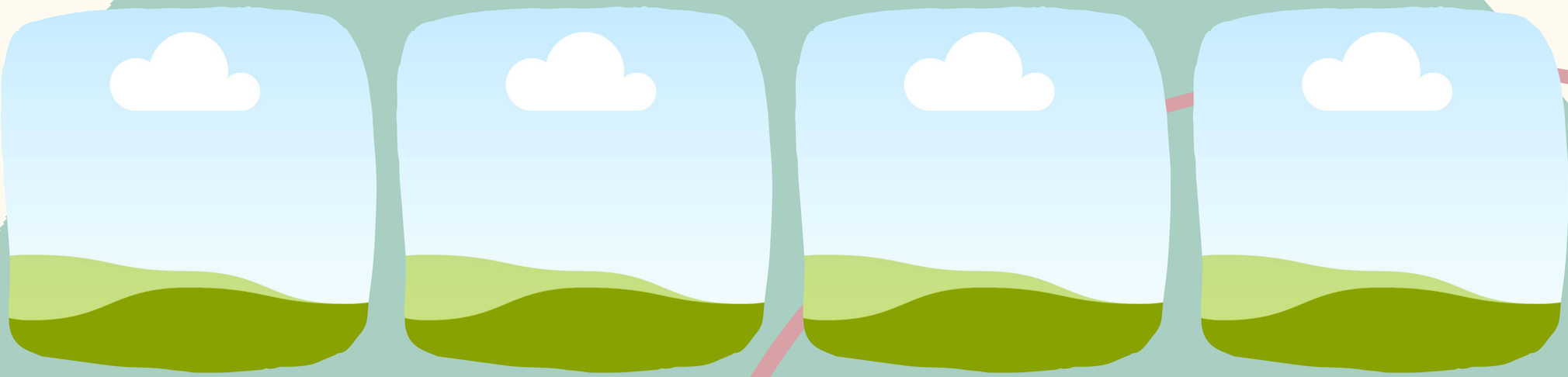
Trường Công Nghệ Thông Tin và Truyền Thông

Mandarin Square Capturing

Presented by Group 08

Group Members



Nguyen Thi Huong Giang
20204909

Nguyen Huu Tuan Duy
20204907

Nguyen Trung Dung
20204906

Duong Tung Giang
20204908

Outline

1. Detailed requirement

2. Use case and Interaction

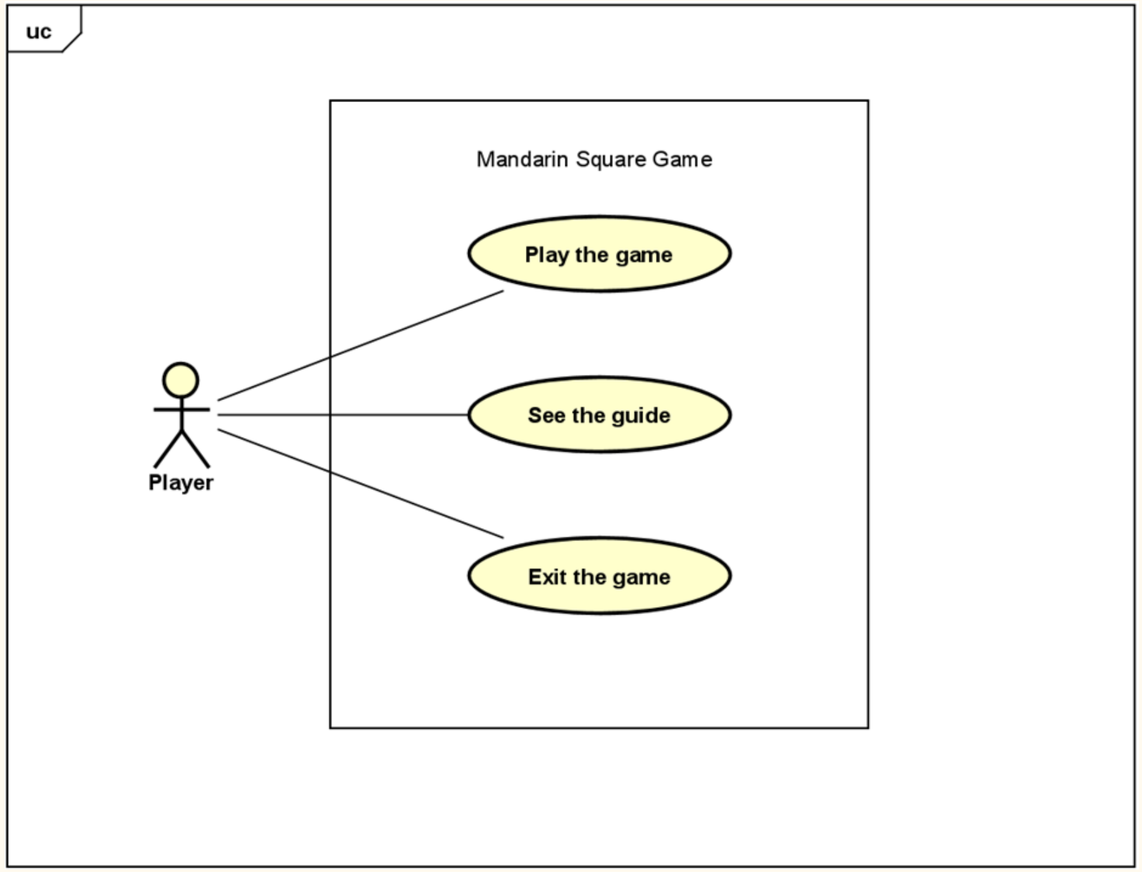
3. Design

1. Detailed 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:
  • Gameboard: 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 (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

2. Use case and Interaction

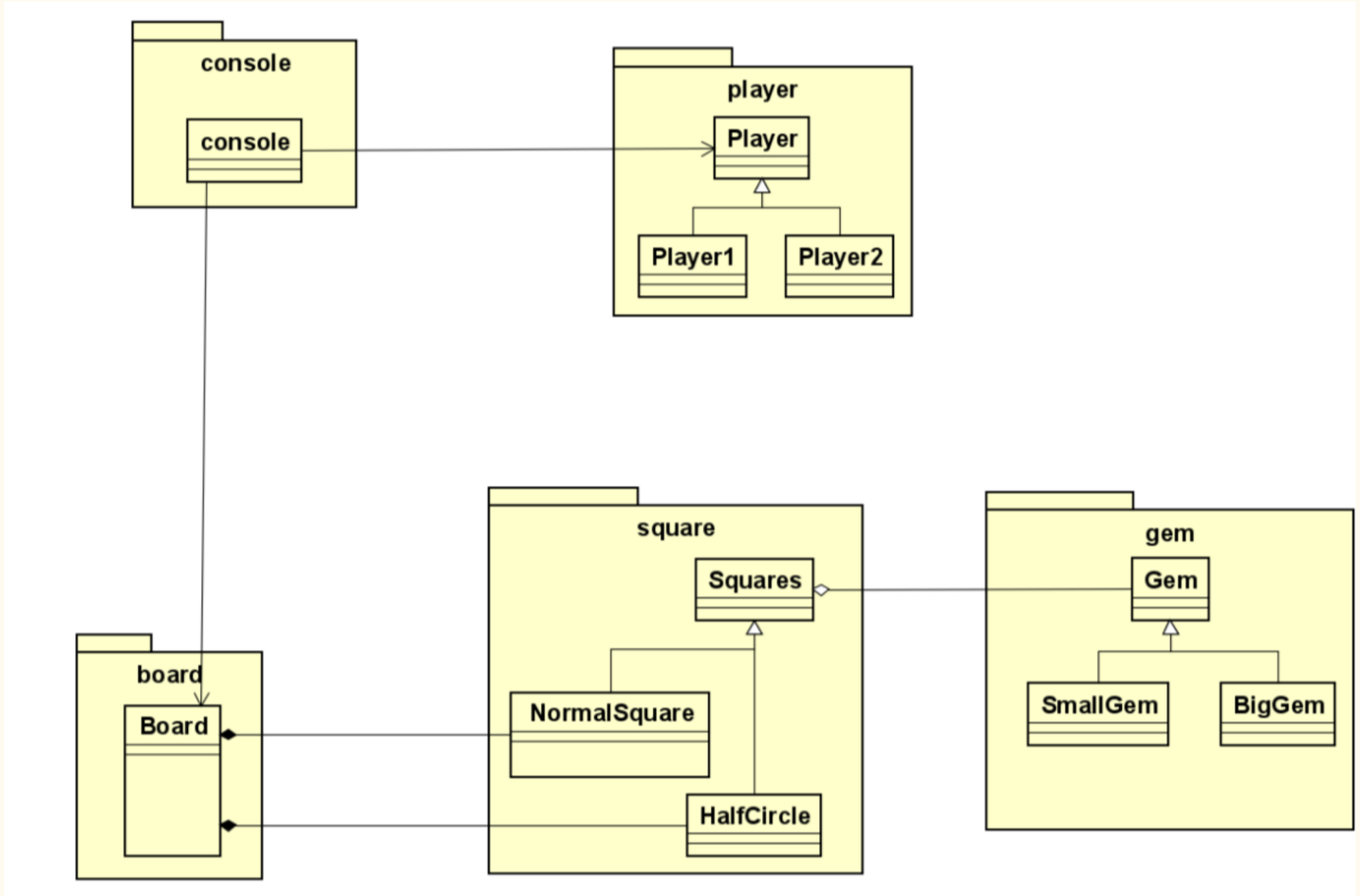


Interaction:

- User can start the game by clicking on Play button
- User can see the guide by clicking on Help button
- User can exit the game by clicking on Exit button

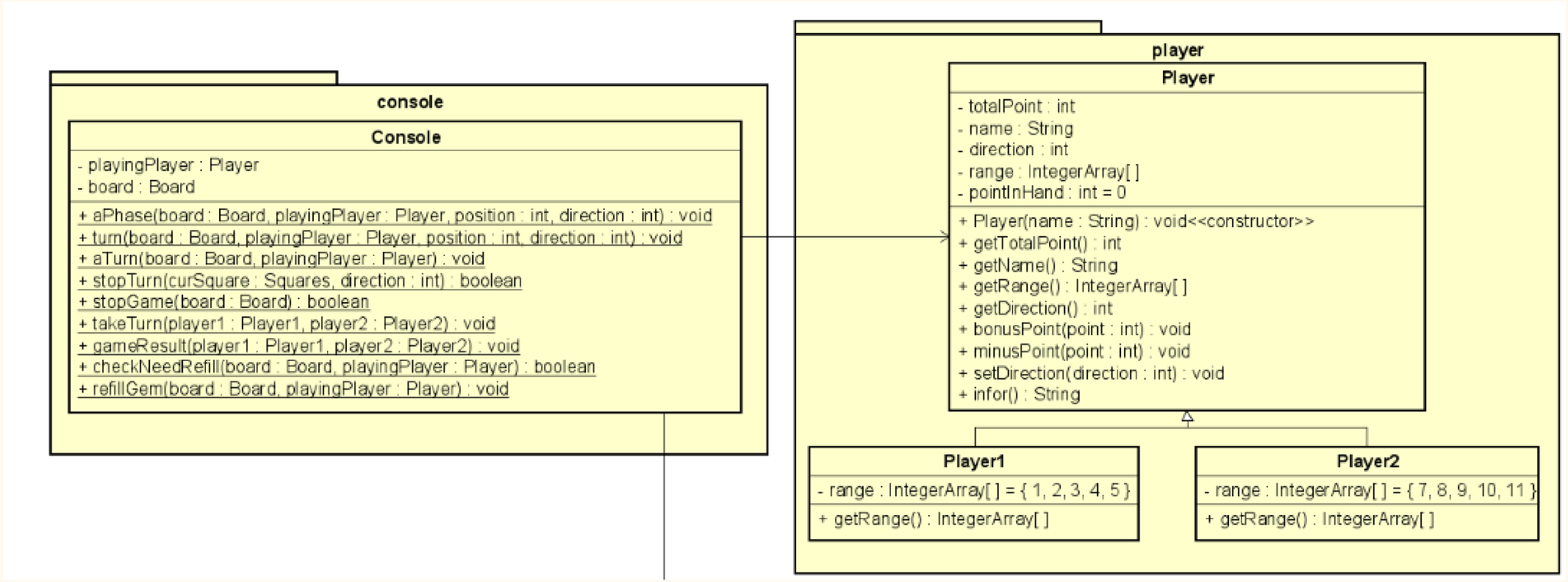
3. Design

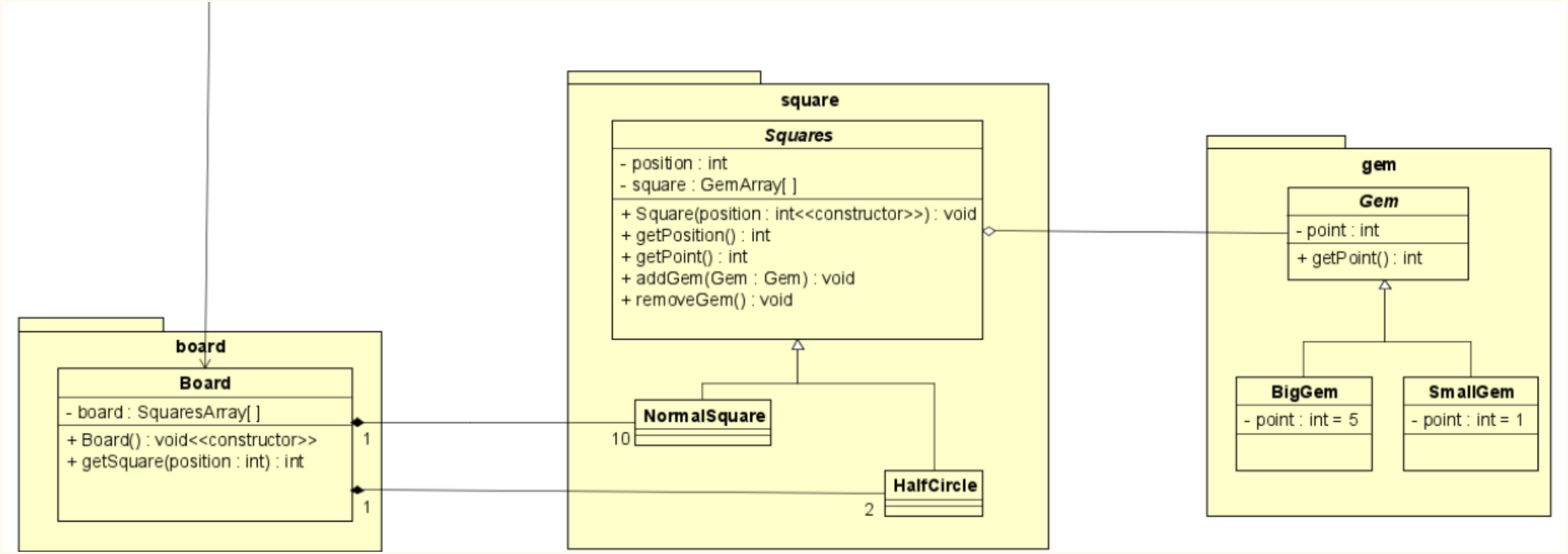
3.1. General Class Diagram



3. Design

3.2. Several Class Diagram




board
Board
- board : SquaresArray[ ]
+ Board() : void<<constructor>>
+ getSquare(position : int) : int
1
1
square
Squares
- position : int
- square : GemArray[ ]
+ Square(position : int<<constructor>>) : void
+ getPosition() : int
+ getPoint() : int
+ addGem(Gem : Gem) : void
+ removeGem() : void
NormalSquare
10
HalfCircle
2
gem
Gem
- point : int
+ getPoint() : int
BigGem
- point : int = 5
SmallGem
- point : int = 1

3. Design

3.3. Detailed for class/ method

Package	Class	Method
square	+ Squares: attribute: *position*: each square have a unique number, from 0 to 11 + NormalSquare: child class of Squares (represent 10 normal squares, position from 1 to 5 and 7 to 11) + HalfCircle: child class of Squares (represent 2 half circles, position 0 and 6)	+ getPosition(): return the position of the square + getPoint(): return the number of point in the square + addGem(): add 1 gem in the square + removeGem(): remove 1 gem in the square (if there is no gem do nothing)

player	+ Player: 5 attributes: *totalPoint* (the total point of player), *name*, *pointInHand* (the point in hand, or more technically, the number of remaining step), *direction* (the direction to move, represent by 1 and -1(clockwise and counterclockwise respectively)), *range* (an integer array include numbers as position player could choose in a legal move) + Player1: child class of Player. Represent player number 1. *range* of Player1 is {1,2,3,4,5} + Player2: child class of Player. Represent player number 2. *range* of Player2 is {7,8,9,10,11}	+ getTotalPoint(): return the point of the player + getName(): return the name of the player + getRange(): return the range of position which the player can make a legal move + getDirection(): return 1 or -1 based on current direction + bonusPoint(int n): add n point to the player + minusPoint(int n): remove n point to the player + setDirection(int direction): set the direction to new value
gem	+ Gem: attribute: *point* + SmallGem: child class of Gem, attribute *point* set to 1 + BigGem: child class of Gem, attribute *point* set to 5	+ getPoint(): return point of the gem
board	+ Board: attribute *board*: an array to store 10 NormalSquare and 2 HalfCircle	+ getSquare(int position): return the position of the square + print(): print the board (position of each square and point of this square)

console	+ Console: attribute: *board*, *playingPlayer* (an instance of class Player, playingPlayer represents the player who is playing the turn)	+ aPhase(): represent the scattering phase, if pointInHand !=0 then move to the next position by using nextposition = curposition + direction, then addGem() to that new Square and pointInHand -=1 + stopTurn(): check the condition to end the turn + stopGame(): check the condition to end the game + takeTurn(): switch to the other player when one's turn end + gameResult(): return the result of the game + checkNeedRefill(): check if the player field has any gem (by the rule, if player did not have any legal move, they have to use captured gem to fill the square to have a legal move, if player did not have enough captured gem to fill, they have to borrow from opponent) + refillGems(): use the captured gem to fill the player's square + turn(): initiate one turn + aTurn(): print all the necessary guideline, then initiate one turn

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explo...

- AimsProject [Lab01 mai
- GUIProject [Lab01 main]
- sourcecode [OOP.DSAI.Ma
  - src
    - board
    - console
    - controller
    - gem
    - GUI
      - EndScreen.java
      - Game.java
      - MenuScreen.java
    - player
    - square
    - view
  - JRE System Library [jdk
  - JavaFX SDK
  - JavaFX
  - Image

Game.fxml | Square.fxml | GameControll... | SquareContro... | Console.java | Game.java | Board.java | ViewStoreCo... | Squares.java | EndScreen.fxml | MenuScreen.java

```java
package GUI;

import java.io.IOException;

public class MenuScreen extends Application {
    private static Player1 player1;
    private static Player2 player2;
    private static Board board;
    private static Player playingPlayer;

    @Override
    public void start(Stage primaryStage) {
        try {
            final String FILE_PATH = "/view/MenuScreen.fxml";
            FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource(FILE_PATH));
            MenuScreenController menuController = new MenuScreenController(board, player1, player2, playingPlayer);
            fxmlLoader.setController(menuController);
            Parent root;
            root = fxmlLoader.load();
            primaryStage.setTitle(null);
            primaryStage.setScene(new Scene(root));
            primaryStage.show();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        player1 = new Player1("player1");
        player2 = new Player2("player2");
        board = new Board();
        playingPlayer = player1;

        launch(args);
    }
}
```

Writable | Smart Insert | 29 : 54 : 914

Thanks for listening

Any questions ?