# OBJECT-ORIENTED PROGRAMMING AND THEORY - 20232

Mini-Project Report - Team 16
Topic 6 - Ô ăn quan

## I. Member contributions

| Member | List of contributions | Contribution |
|---|---|---|
| Đỗ Khánh Nam<br>20225988 | - Implementing and testing all of *game* pkg. (30%)<br>- Writing report. (10%)<br>- Creating use-case, class diagram. (5%) | 45% |
| Ngô Anh Tú<br>20226005 | - Implementing and testing all of *gui* pkg. (30%)<br>- Recording demo video. (5%) | 35% |
| Vũ Bình Minh<br>20226058 | - Creating background image for *gui* pkg. (10%)<br>- Creating class diagram. (5%) | 15% |
| Nguyễn Đức Anh<br>20226009 | - Making presentation slide. (5%) | 5% |

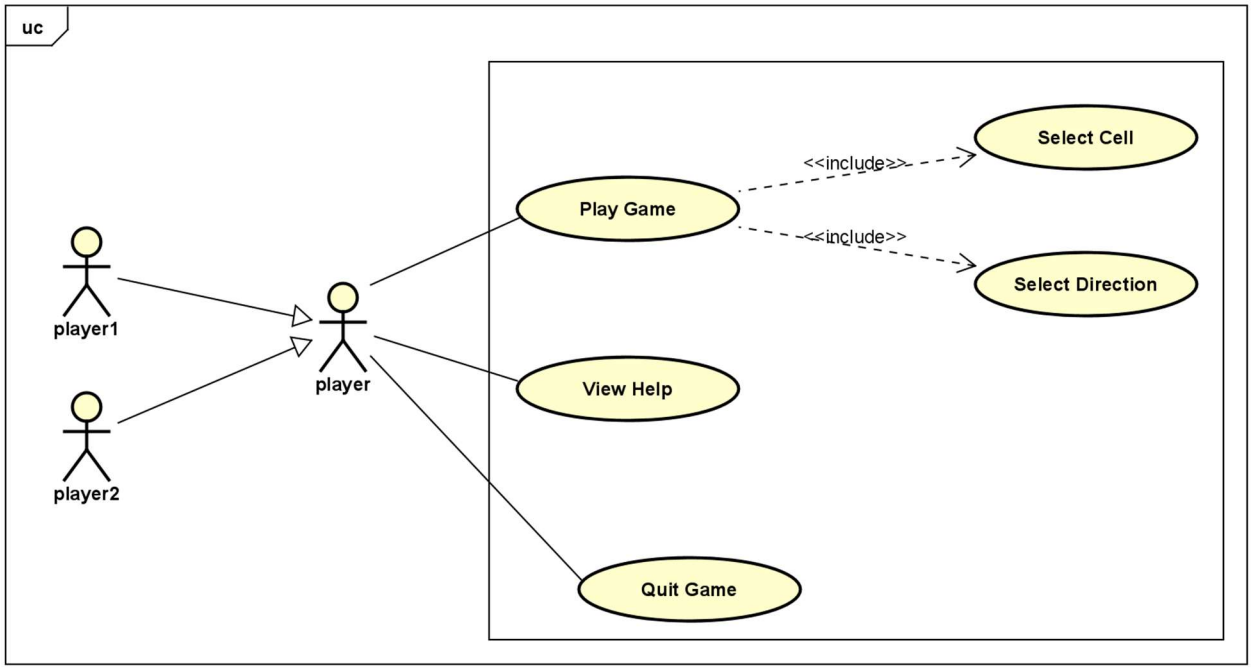
## II. Project Description

In this project, we have made an application for 2 users to play the traditional game Ô ăn quan.

### 1. Game rules

- Gameboard: 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 stones, and each half circle has 1 big stone. Each small stone equals 1 point, and each big stone equals 5 points.
- For each turn, a player will select a square and a direction to spread the stones. He got points when after finishing spreading, there is one empty square followed by a square with gems. The score they got for that turn is equal to the value of stones in that followed square.
- If a player is in turn, but all the squares that player possesses are empty, the player must take five stones that he has taken and place one in each of five squares.
- The game ends when there is no gem in both half-circles.

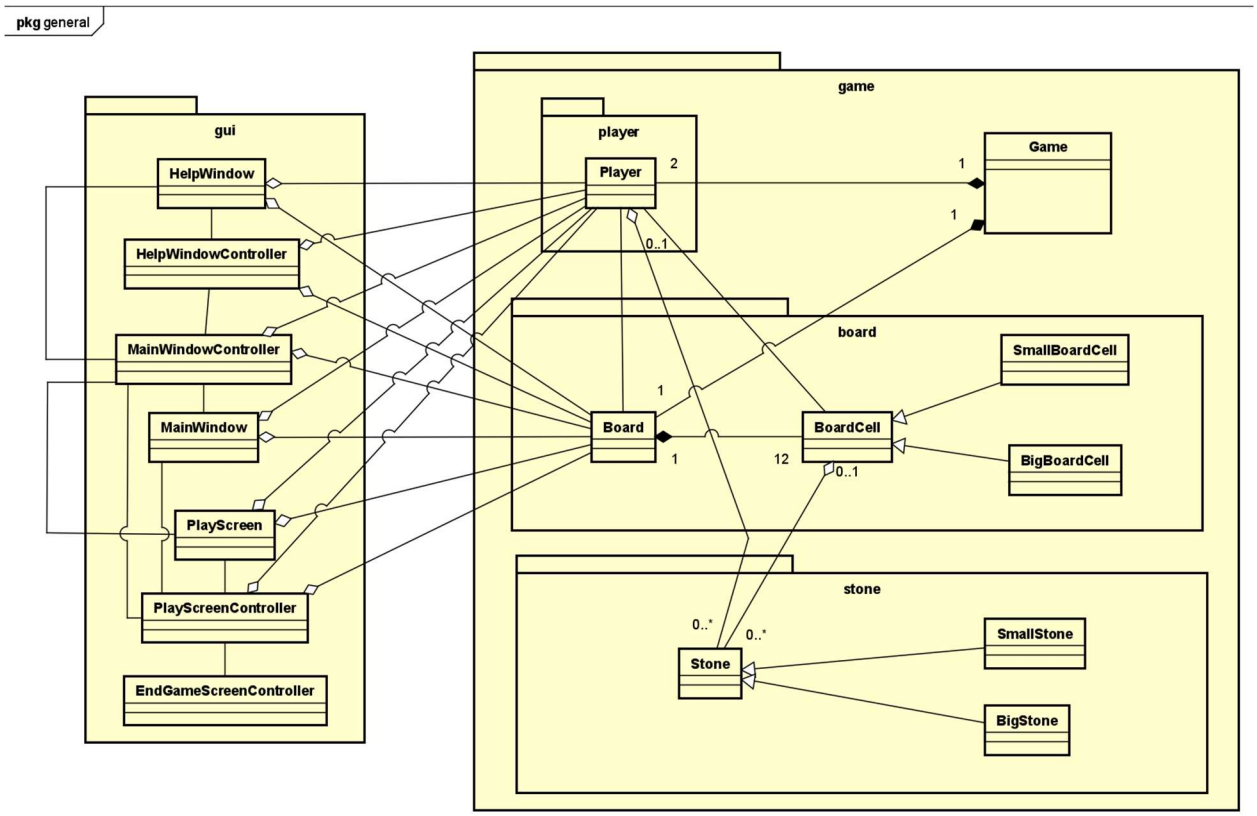
### 2. Requirements




- Play game: The user starts a new game instance. Then, the application displays the starting game board. The players play the game by choosing the square and direction of each move in accordance with the rules stated until the game ends.
- View help: show instructions to play the game.
- Exit: close the application.

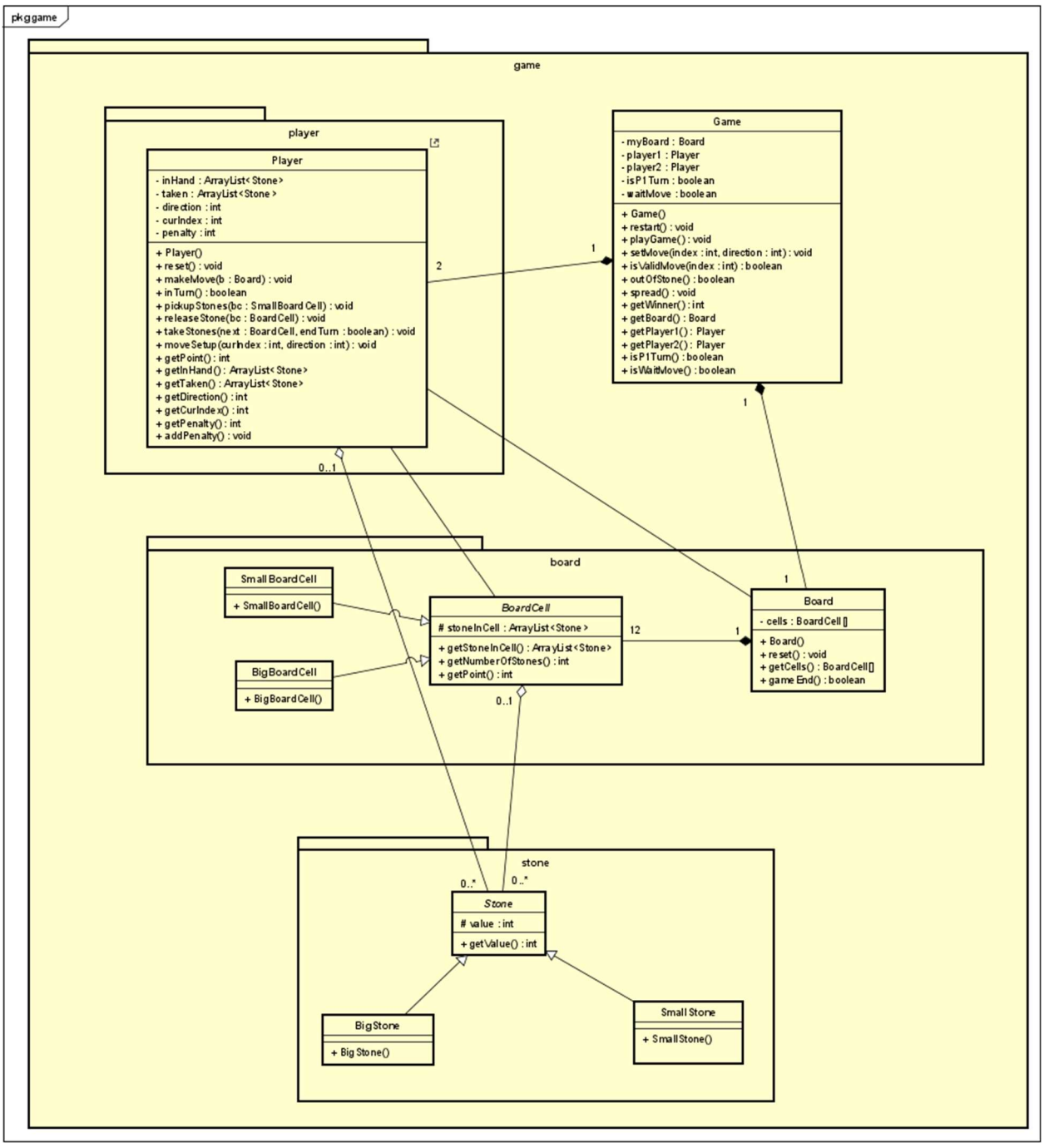
## III. Design

### 1. General class diagram




- There are 2 main components in this application:
  - *Game* pkg includes all objects that control the game: stone, gameboard, player, game.
  - *gui* pkg includes gui components.
- **<u>Relationships</u>:**
  - Associations: Direct connections between GUI components and their controllers, Player and Board and BoardCell.
  - Compositions: Game is composed of Player, Board. Board is composed of BoardCell.
  - Aggregations: Stone has aggregation relationship with Player and BoardCell, Player and Board have aggregation relationship with GUI components.
  - Generalizations and Specializations: SmallBoardCell and BigBoardCell are specializations of BoardCell, and SmallStone and BigStone are specializations of Stone.

## 2. Game package




**Class Stone:**

- This is the most basic class in our program, representing the stones that players use to interact with other base classes *BoardCell* and *Board*. This abstract class has one single attribute:

    - int value: represent the value of that stone.
- The *Stone* class has two other inherited classes: *BigStone* and *SmallStone*. Value of *BigStone* is 5 while the value of *SmallStone* is 1.

**Class BoardCell:**

- This abstract class represents the cells inside the board, with which users interact while playing. This abstract class has one attribute:
    - ArrayList<Stone>stonesInCell: store all the stones in that cell.
- There are two classes that inherit this class: SmallBoardCell, *BigBoardCell*. The class *SmallBoardCell* is initialized with 5 small stones, While class *BigBoardCell* is initialized with 1 big stone.

**Class Board:**

- This class represents the game board, which includes 12 *BoardCell*(10 *SmallBoardCell* and 2 *BigBoardCell*).This class has one attribute:
    - BoardCell[] cells: an array of size 12 to store all cells in the board.
- The constructor method of this class initializes a board with 2 *BigBoardCell* at index 5 and 11 and 10 *SmallBoardCell* at other indexes.
- This class also has following methods:
    - void reset(): reset the board to initial state.
    - boolean gameEnd(): If both of the big cells are empty, return true else return false.

**Class player:**

- The *Player* class encapsulates the behavior and state of a player in the O An Quan game. It manages the player's actions, such as making moves, picking up, releasing, and capturing stones. Additionally, it tracks the player's score and penalties, ensuring the game rules are adhered to during play. By coordinating these actions and maintaining the player's state, the Player class is essential for the dynamic flow and strategic elements of the game. The class has the following attributes:
    - ArrayList<Stone> inHand: represents the stones that the player is holding in hand.
    - ArrayList<Stone> taken: represents the stones that the player has taken.
    - int direction: represents the direction of spreading.
    - int curIndex: represents the index of the cell that the player is playing.
    - int penalty: represents the number of penalties that player has received.
- The constructor of this class will initialize inHand and taken array and set direction to 0, curIndex to -1, penalty to 0. The followings are methods of this class:
    - void reset(): reset the player’s attributes to initialization.
    - boolean inTurn(): check if the player is playing his turn(curIndex >= 0).

- void pickupStones(SmallBoardCell bc): move all stones in a cell to inHand. Used to pick up stones at the start of a turn.
- void releaseStone(BoardCell bc): release one stone in a selected cell, used in the procedure of spreading where users need to drop the stones into the cells on their path.
- void takeStones(BoardCell next, boolean endTurn): move all stones in a selected cell to taken. Used when the player finishes spreading. End turn if there is no more cell to take stones from, continue if there are.
- void moveSetup(int curIndex, int direction): set up the index of the starting cell and the direction to spread in a turn.
- int getPoint(): return the point a player has by calculating the sum of the value of all the captured stones and subtract 5 points for each penalty.
- void makeMove(Board b): This method executes the player's move based on the current game state.
    - Check if in turn: The move proceeds only if the player is currently in a turn (inTurn() returns true).
    - Calculate indexes: Determines the next and after indexes based on the current index and direction.
    - Release stone: If the player has stones in hand, one stone is released to the current cell.
    - Pickup stones: If no stones are in hand and the current cell has stones, the player picks up all stones from the current cell.
    - Capture conditions: If the player lands on an empty cell with the next cell containing stones, various capture scenarios are handled.

**Class game:**

- This class is the central controller for a game of O An Quan, managing the game board, player actions, and game state transitions. It is responsible for initializing the game components, handling player turns, validating and executing moves, checking game conditions, and determining the winner. This class has the following attribute:
    - Board myBoard: represents the game board on which the game is played.
    - Player player1: represents player 1 in the game.
    - Player player2: represents player 2 in the game.
    - boolean isP1Turn: indicates if it is currently player 1's turn.
    - boolean waitMove: indicates if the program is currently waiting for a player to make a move.
- The constructor method initializes the game components and sets the initial state of the game. Creates a new *Board* instance. Creates new instances for player1 and player2. Sets isP1Turn to true indicating player 1 starts first.Sets waitMove to true indicating the program is waiting for a move to be made. The followings are methods of this class:

- void restart(): resets the game to its initial state.
- void playGame(): executes the game's main loop, managing the flow of turns and moves.
    - Checks if the game has ended; if so, sets waitMove to true.
    - If not waiting for a move, it checks whose turn it is and makes a move accordingly.
    - Updates the waitMove and isP1Turn flags based on whether the player's turn has ended.
- public void setMove(int index, int direction): sets up a move for the current player. If the move is valid, set the move for the current player (player1 or player2) using moveSetup. Sets waitMove to false, indicating a move has been set.
- boolean isValidMove(int index): checks if the proposed move is valid. Validates the move based on the current player's turn, the index range, and whether the cell has stones. Ensures the game is waiting for a move.
- boolean outOfStone(): checks if the current player has run out of stones in their cells. Sums the number of stones in the current player's cells. If the sum is zero, calls spread to redistribute stones. Returns true if the player is out of stones, otherwise false.
- void spread(): redistributes stones to the current player's cells when they are out of stones. Adds one small stone to each of the current player's cells. Adds a penalty to the current player.
- int getWinner(): Determines the winner of the game. Returns 1 if player1 wins, 2 if player2 wins, and 3 if it's a tie.