OOP MINI-PROJECT REPORT

*Group 08: Mandarin Square Capturing*

Nguyễn Trung Dũng – 20204906

Nguyễn Hữu Tuấn Duy – 20204907

Dương Tùng Giang – 20204908

Nguyễn Thị Hương Giang – 20200185

1. **Assignment of members**

|  |  |
| --- | --- |
| Name | Tasks |
| Nguyễn Trung Dũng | Use-case diagram, class Gem, GUI MenuScreen |
| Nguyễn Hữu Tuấn Duy | Package Console (half), slide, rules |
| Dương Tùng Giang | Report, use-case diagram, class diagram |
| Nguyễn Thị Hương Giang | Package Player, Board, Square, Console (half), GUI GameScreen&EndScreen |

1. **Mini-project description**

Traditional game: Mandarin Square Capturing (Ô ăn quan)

1. Detailed requirement

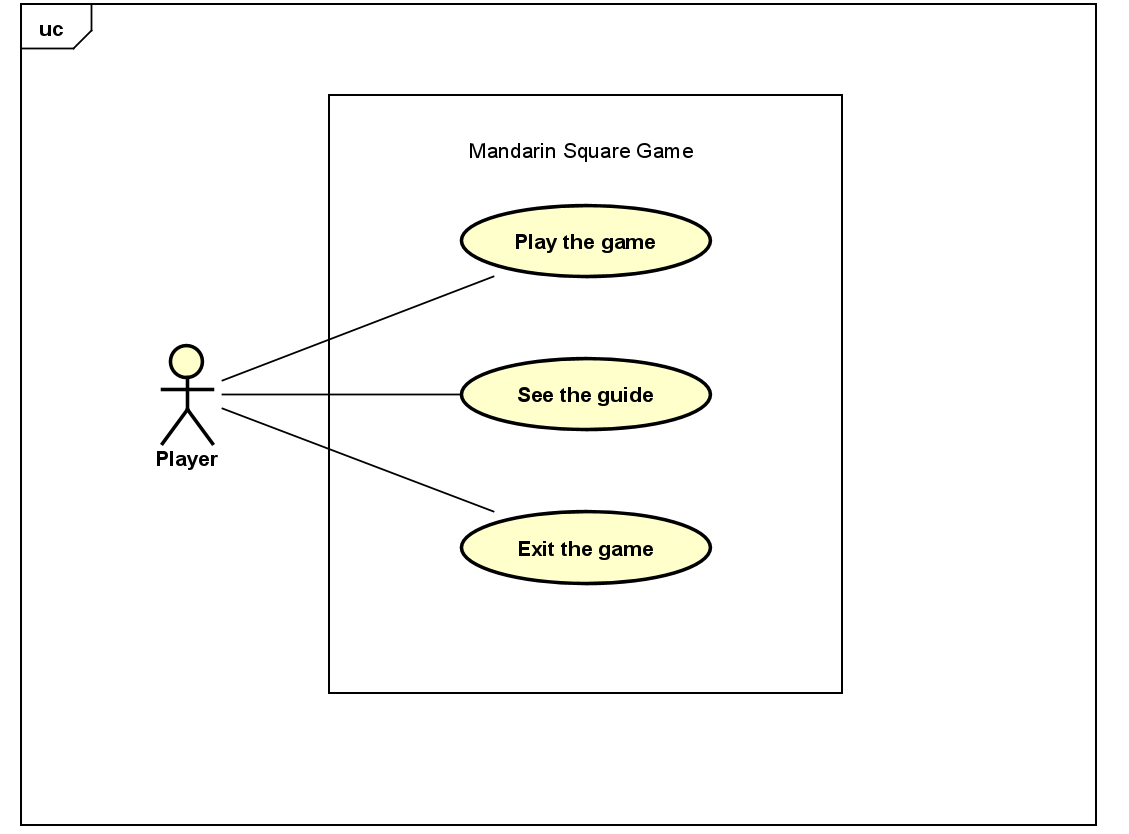
+ On the main screen:

* 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 guideline 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

1. Use case and interaction



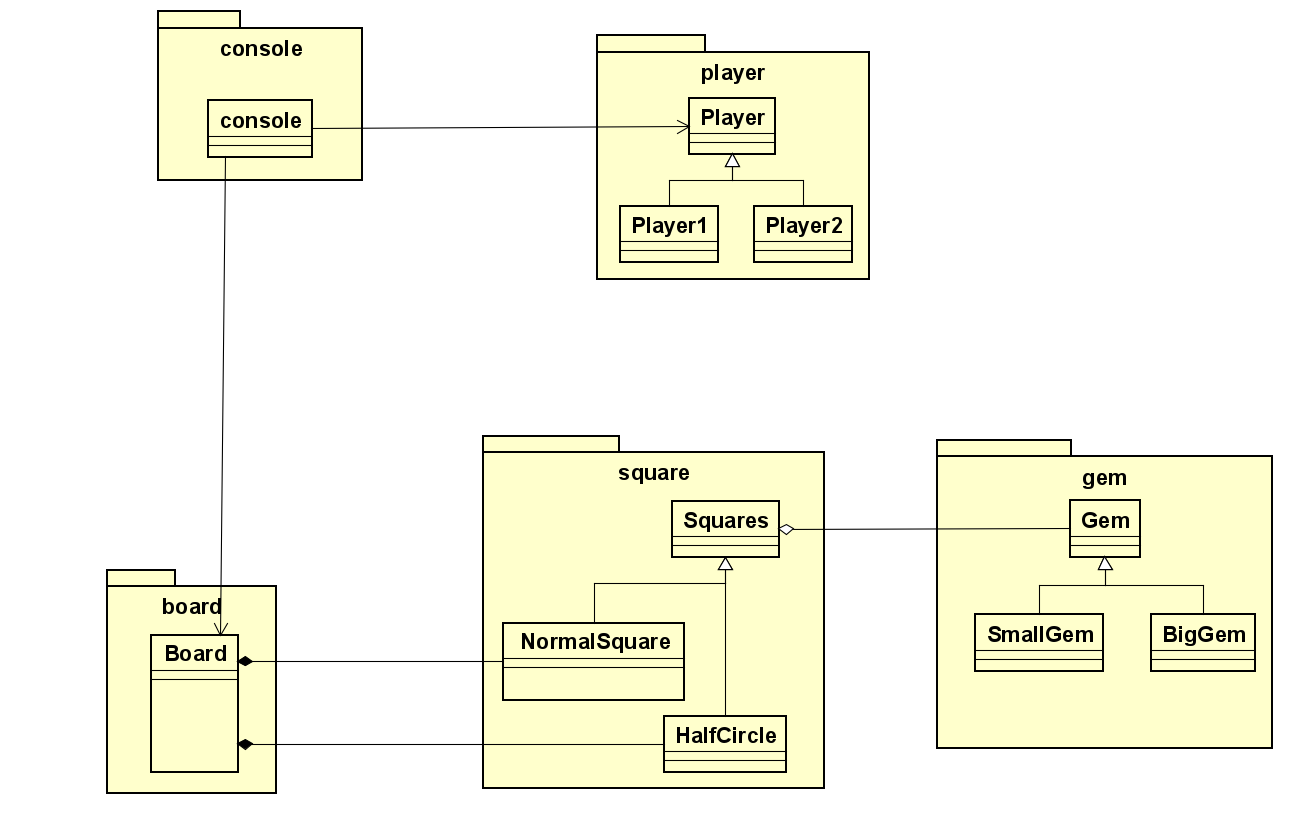
Interaction: User can play the game

User can see the guideline

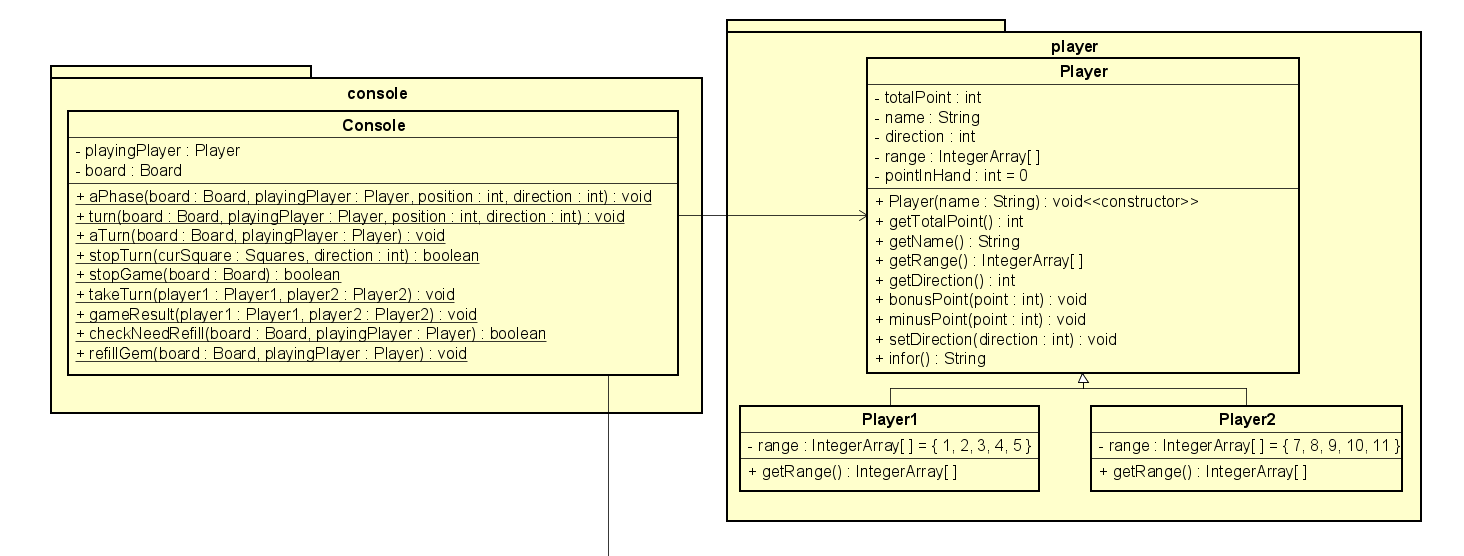
User can exit the game

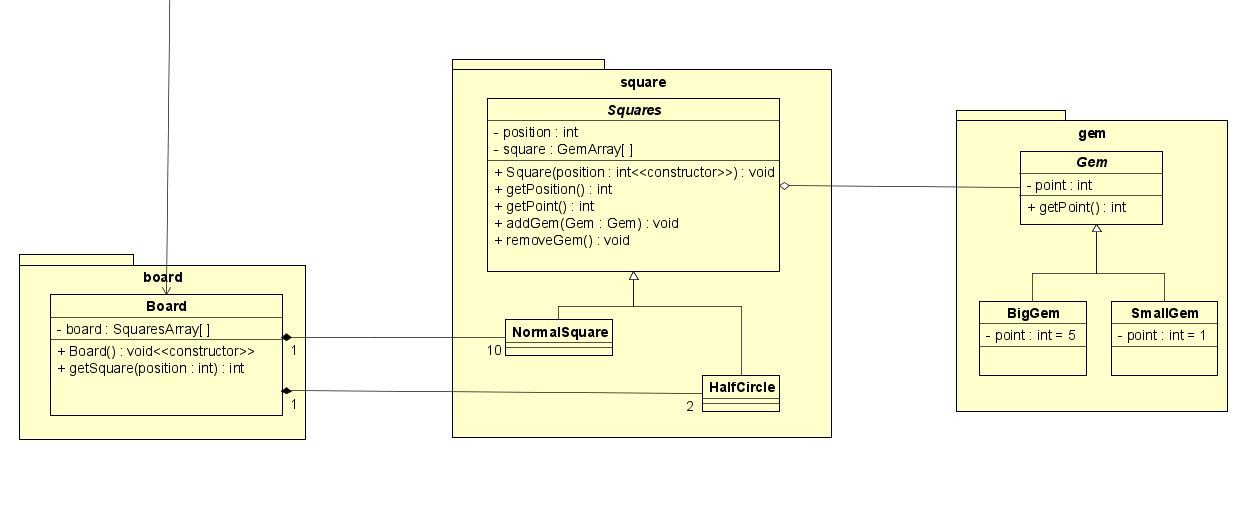
1. Design

3.1. General class diagram



3.2. Several class diagram





3.3. Design explanation

class BigGem, SmallGem – class Gem: Both are child class of abstract class Gem

class NormalSquare, HalfCircle – class Squares: Both are child class of abstract class Squares

class Player1, Player2 – class Player: Both are child class of class Player

class Gem – class Squares: Gem is a part of Squares, but Gem can exist without Board

class NormalSquare – class Board: NormalSquare is a part of Board, Board can not exist without NormalSquare, and one Board has ten NormalSquare

class HalfCircle – class Board: HalfCircle is a part of Board, Board can not exist without HalfCircle, and one Board has two HalfCircle

class Console – class Player: Console controls Player

class Console – class Board: Console controls Board

Class Console initiates the game, creates two players by class Player and creates the board by class Board.

Board has ten squares as instance of class NormalSquare and two half circles as instance of class HalfCircle stored in attribute *board*

Each square has five small gems as instance of class SmallGem, each half circle has one big gem as instance of class BigGem

For each turn, the game requires input to process, by using unique number for each square, from 1 to 5 for player 1 and 7 to 11 for player 2, to determine the square, then by using 1 and -1 to determine the direction. After receiving the input, the console will run a complete turn.

All point earned after a turn will be stored (point of player 1 will be stored in instance of class Player1, and point of player 2 will be stored in instance of class Player2)

After that, the game switch to the other player, and repeat the process.

Before each turn, the game will check the condition of the game (is there any gem on the square to make a legal move?, is the two BigGem in HalfCircle already captured?) to decide (refill the gem and continue the turn or end the game)

When the game ends, result will be shown

3.4. Detail 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 |