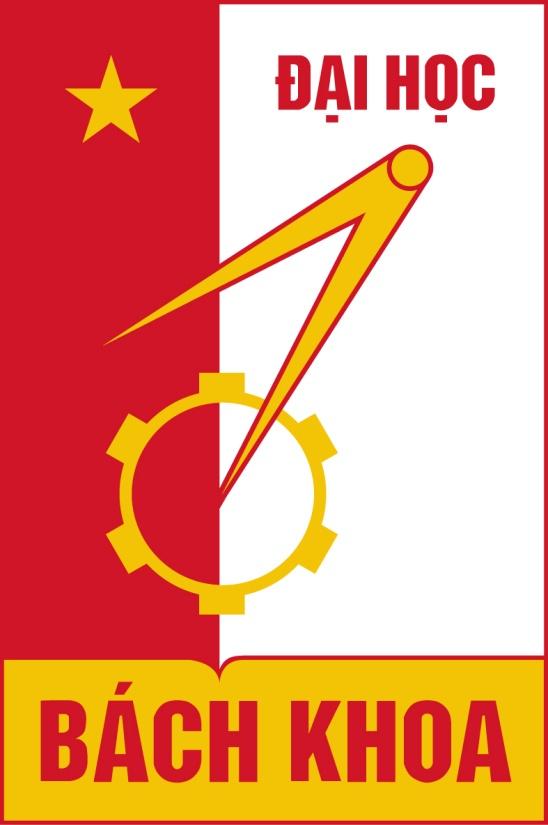
**Hanoi University of Science and Technology**

**School of Information & Communication Technology**



**Object-Oriented Language and Theory**

**Mini-Project Report**

Students: Nguyễn Thị Thu Giang 20194750

Tống Thị Thu Anh 20194728

Class ID: 122154/122156

Instructor: Nguyễn Thị Thu Trang

Hanoi, 06/2021

## Assignment of members:

* Work assignments: All members working all tasks together via video call on Teams, including design GUI, design diagrams, ideas of how to implement each class and method and coding work.
* Idea: From an open source code at  <https://github.com/thanglongnamnay/O-an-quan>

## Mini-project description:

* 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:

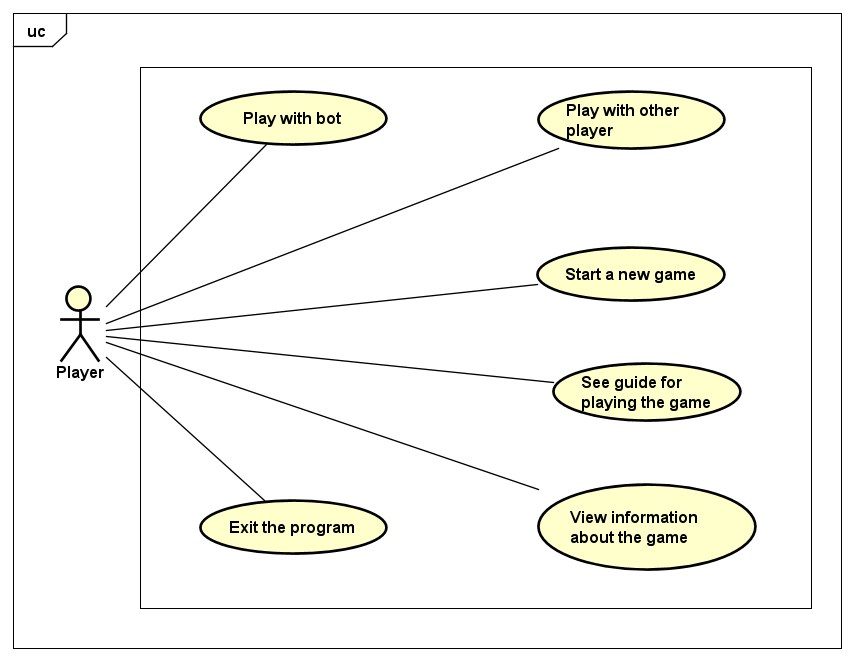
+ Game board: The game board consists of 10 squares, divided into 2 rows, and 2 half- circles on the 2 ends of the board. Initially, each square has 5 small gems, and each half- circle has 1 big gem. Each small gem equals 1 point, and each big gem equals 5 points.

+ 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

* Use case diagram and explanation: How the users interact to the software with use cases



* Users can start a new game, choose play mode or exit the game.

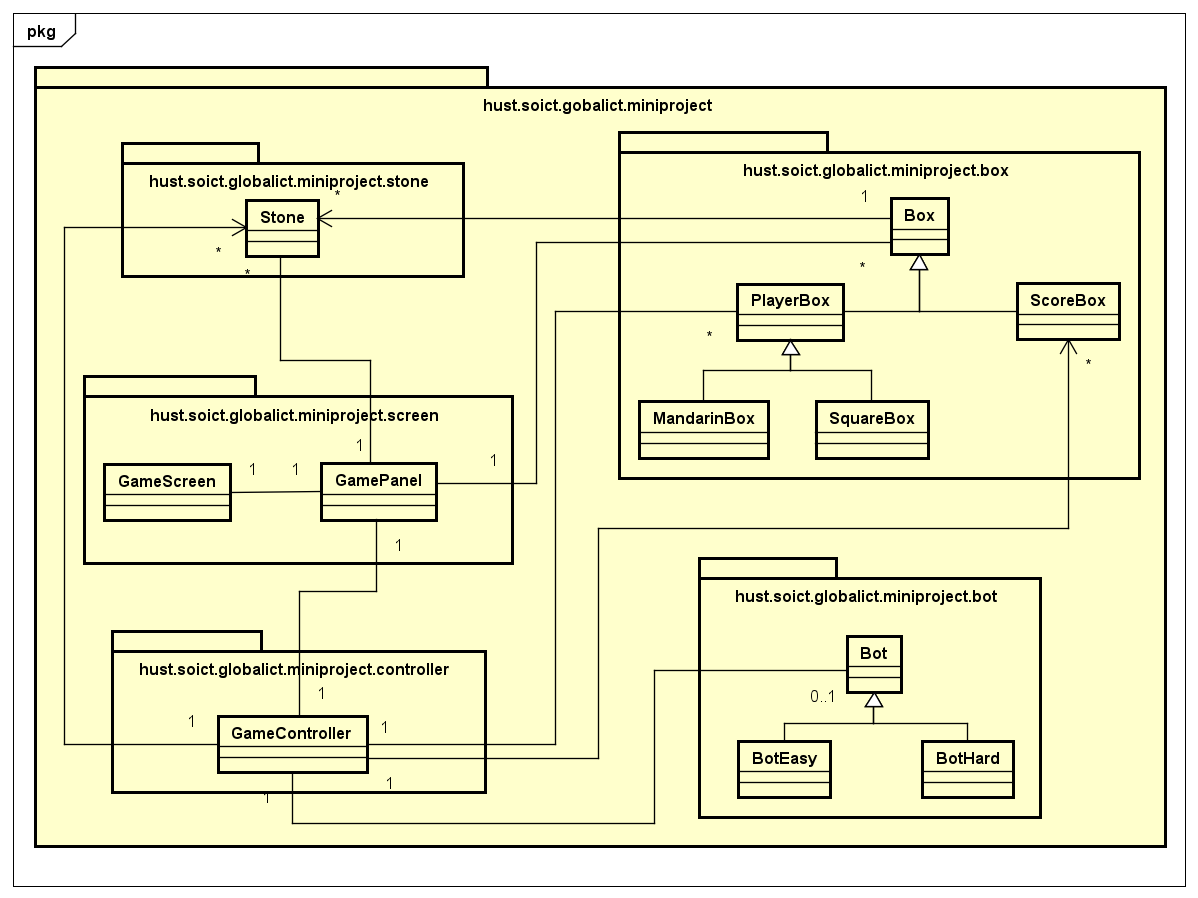
+ In play mode: Users can choose mode 1 Player (play with bot) or 2 Player (play with other player).

+ In the 1 Player mode, users can choose a level of difficulty: Easy or Hard.

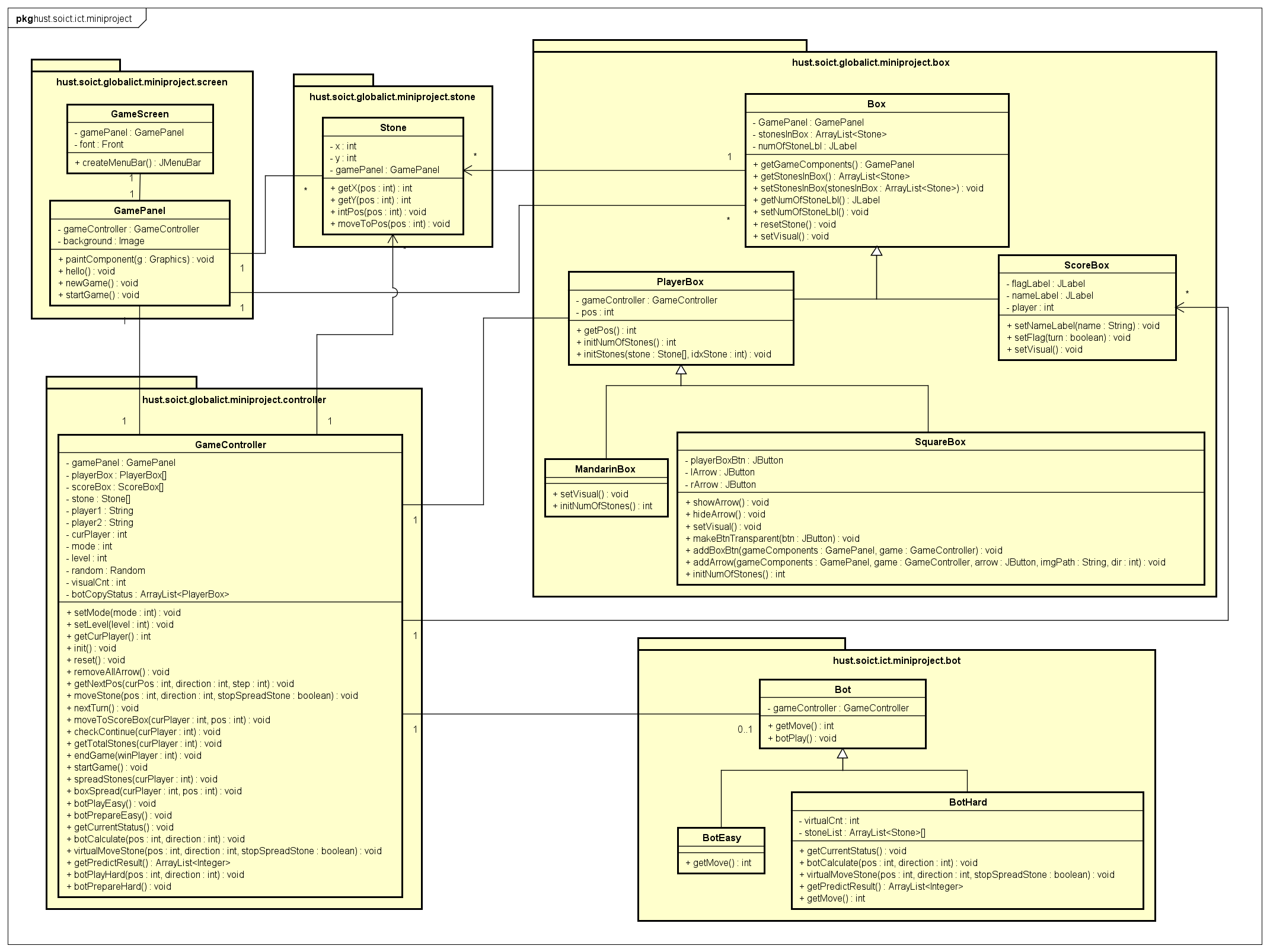
* User can also see the guide of the game and view information about the game

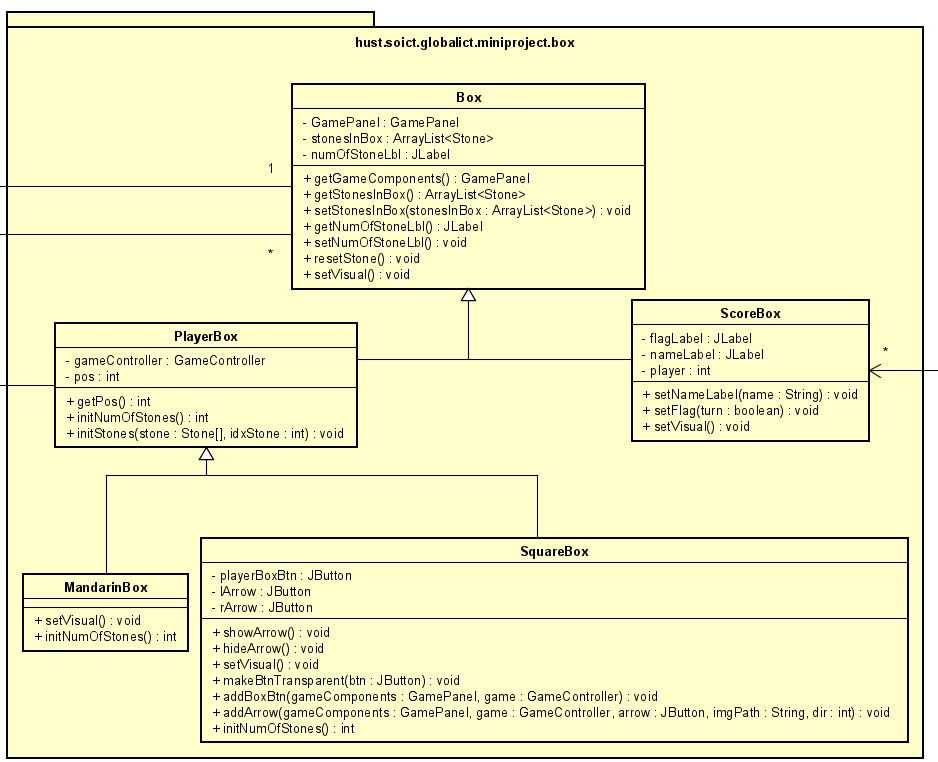
## Design:

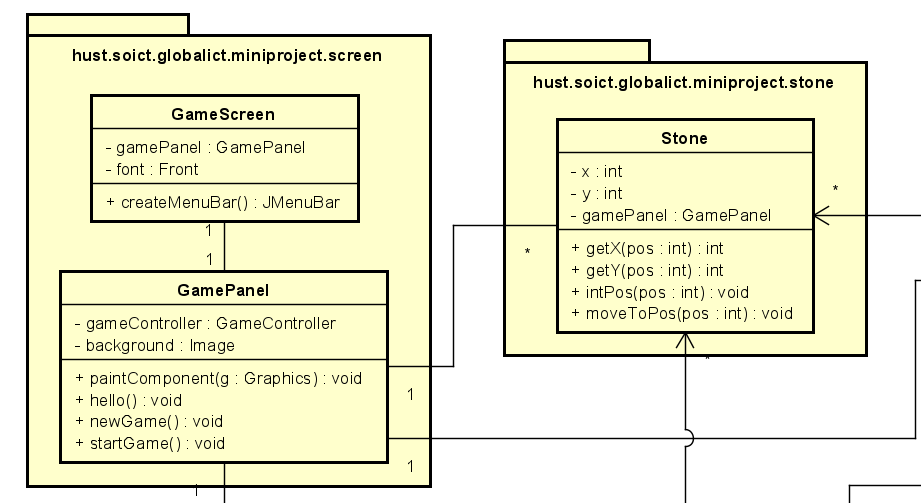
* A general class diagram: Class diagram may be with packages, including all classes without attributes/operations

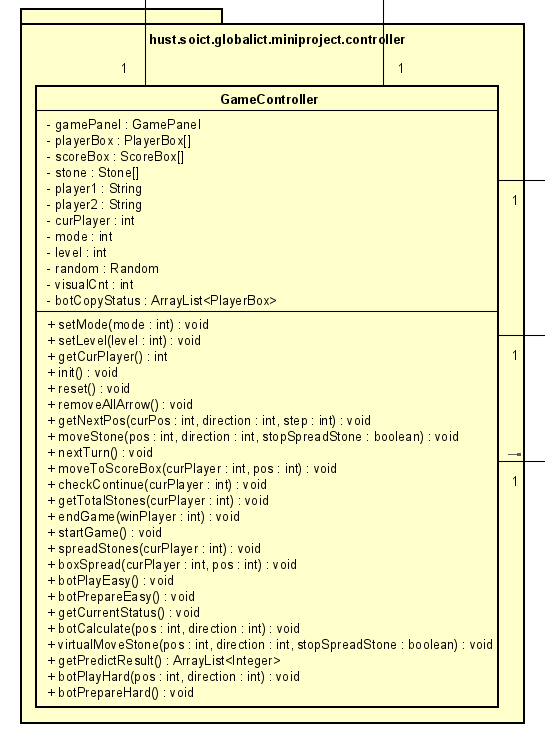


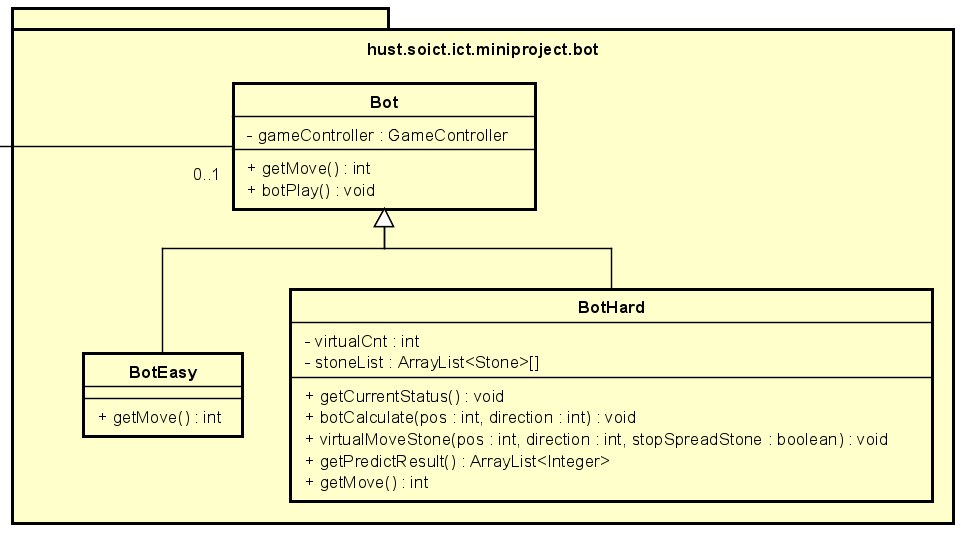
* Detail class diagrams for each package or several packages, with detail attributes/operations for each class











* Explanation of the design: Describe the relationships between classes, the implementations of some important methods
* Class relationship:

+ Inheritance

* Class ScoreBox and abstract class PlayerBox inherit Box
* Multi-level inheritance: Class MandarinBox and SquareBox inherit PlayerBox and implement ActionListener by using anonymous class
* BotEasy and BotHard inherit Bot

+ Association:

* Classes Box and Stone have one-to-many association relationship
* GamePanel (JPanel) is a component of GameScreen (JFrame) (one-to-one association)
* Classes Box, Stone have some components (JButton, JLabel, Image) of GamePanel(many-to-one association)
* Classes GameController associate with PlayerBox, ScoreBox, Stone (one-to-many) and Bot(one-to- zero or one)
* Important methods:

+ Method moveStone() in class GameController: to control stone’s movements in the game

* Firstly, check if current box has stones to move
* If current box is empty, then check if we can earn stones in the next box
* If current box is not empty, check if we can continue to move:
* If current box is mandarin box, end turn
* Else if current box is square box, continue to move

+ Method checkContinue() in class Game: to check if there is a end game situation or out of stones in square boxes of the current player

* If two mandarin boxes have both been taken, end game situation occurs, return 0
* If all five squares on current player’s side are emptied, this player must place one stone he/she has aside back in each of the five squares:
* If the player has enough stones (at least 5), the game can resume, return 1
* Else this player lose, end game, return -1
* Method nextTurn() in class Game:

+ Firstly, check status of the game via the return value of checkStatus() method

+ If status = 0, i.e two mandarin boxes are both empty, check for the winner. Winner is the player has more stones in his/her side (in both square boxes and score box)

+ If status = 1, i.e out of stones in current player's side and he/she have enough stones to spread out, spread stones and the game resume

+ If status = -1, i.e out of stones in current player's side but he/she does not have enough stones to spread out, this player lose, end game

* Methods related to bot in Bot package:

+ Easy level: bot plays randomly by get a random possible box to move in random direction

+ Hard level: bot calculate all possible ways to move and take the best option, while best option here is considered to earn as many as possible stones

* Copy current status to calculate in virtually (method getCurrentStatus())
* Bot calculation has not to affect current status of the game, so it must move stone in virtually (method virtualMoveStone(int, int, boolean))
* Calculate result (stones earned) if move from a specific position in a specific direction (method botCalculate(int,int))
* Get all possible results (number of earned stones) by method getPredictResult()