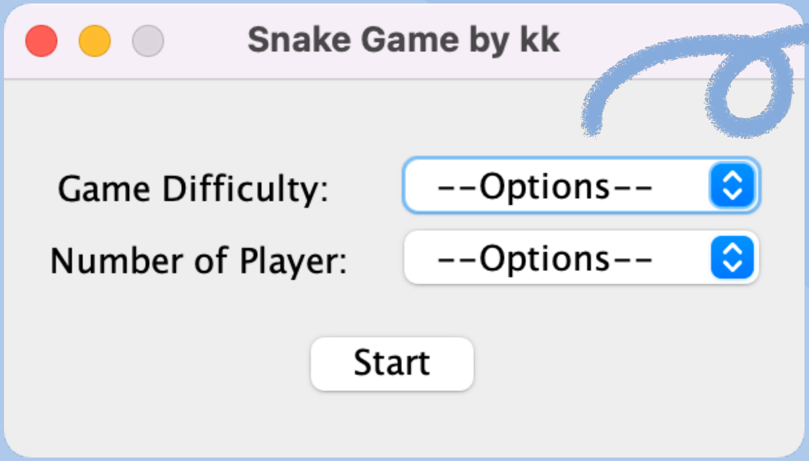
SP22 CS 5004 Final Project

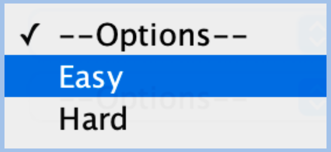
DRAGON GAME

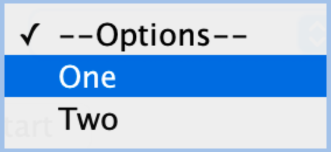
Ke Chen
Instructor: Mark L. Miller, Ph.D.
TA: Meilin Liu
5/5/2022

Contents

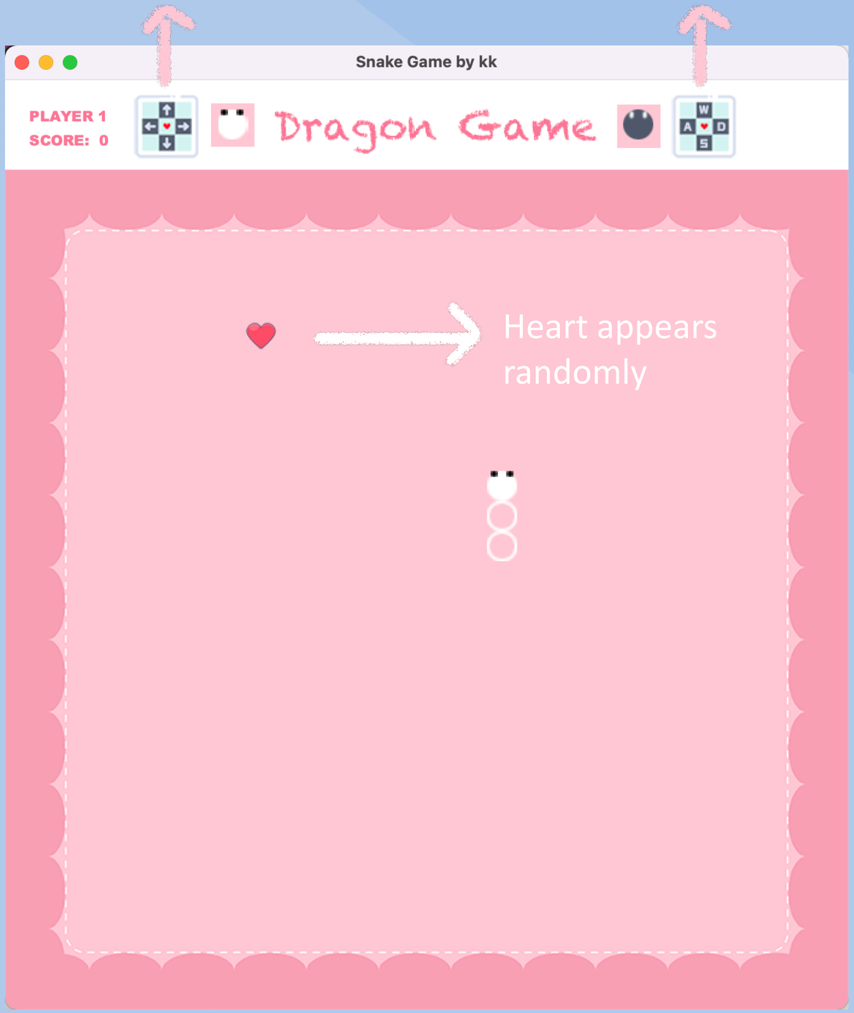
1 PROJECT DEMO ► Interface Design ► Option Window



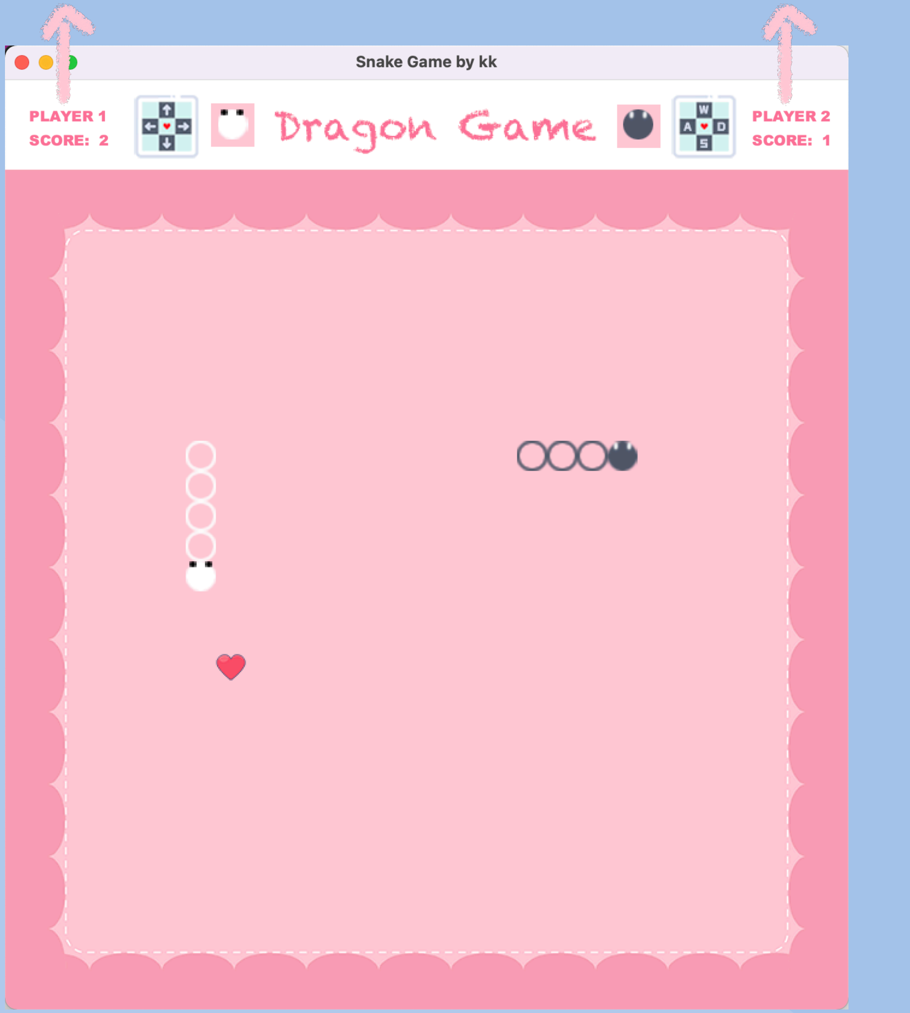




1 PROJECT DEMO ► Interface Design ► Game Panel

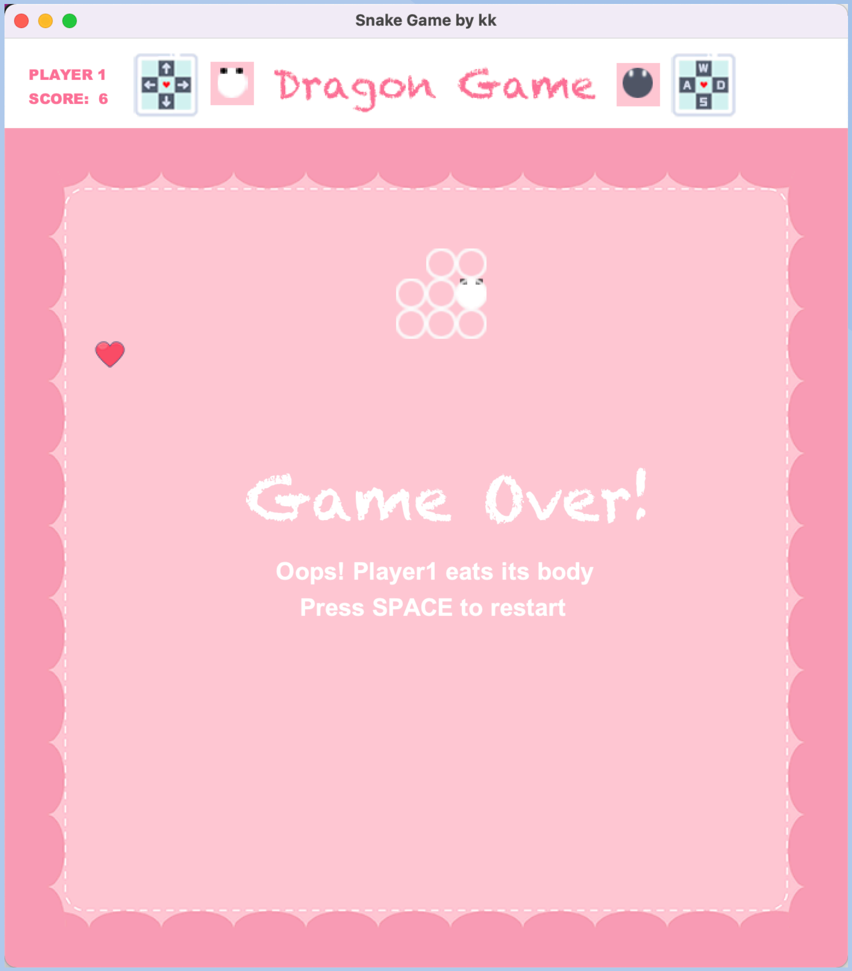


One Player Mode



Two Players Mode

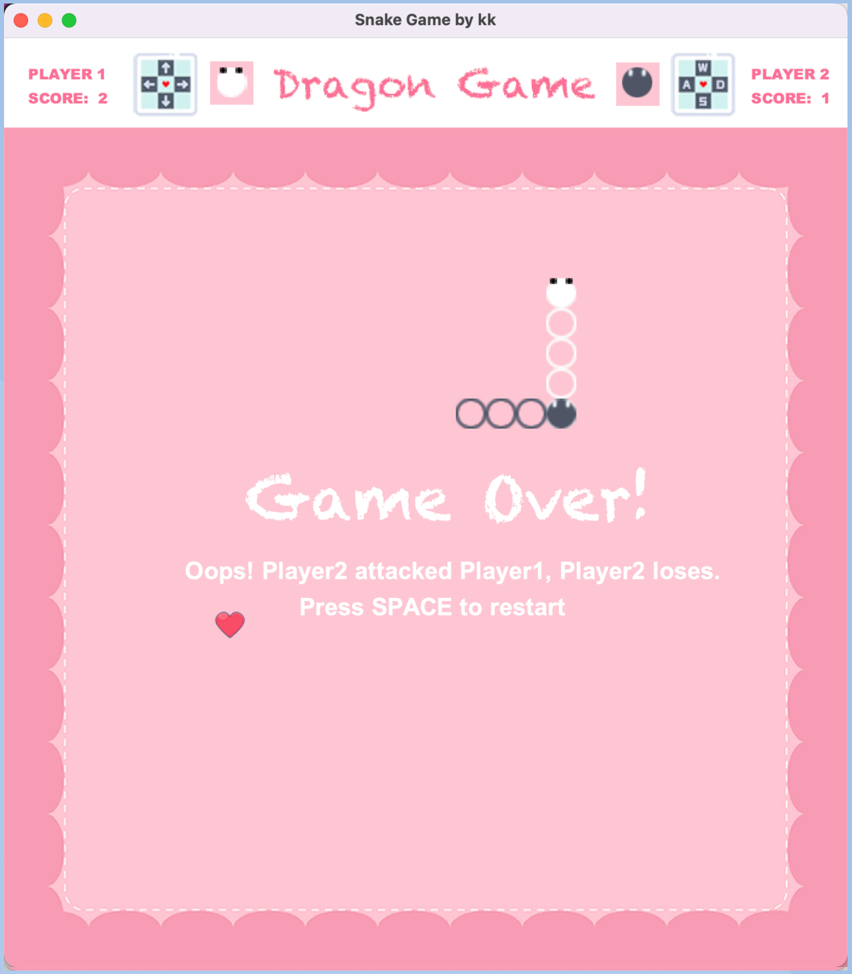
1 PROJECT DEMO ► Interface Design ► Game Panel



Game Over!

Oops! Player1 eats its body
Press SPACE to restart

One Player Mode



Game Over!

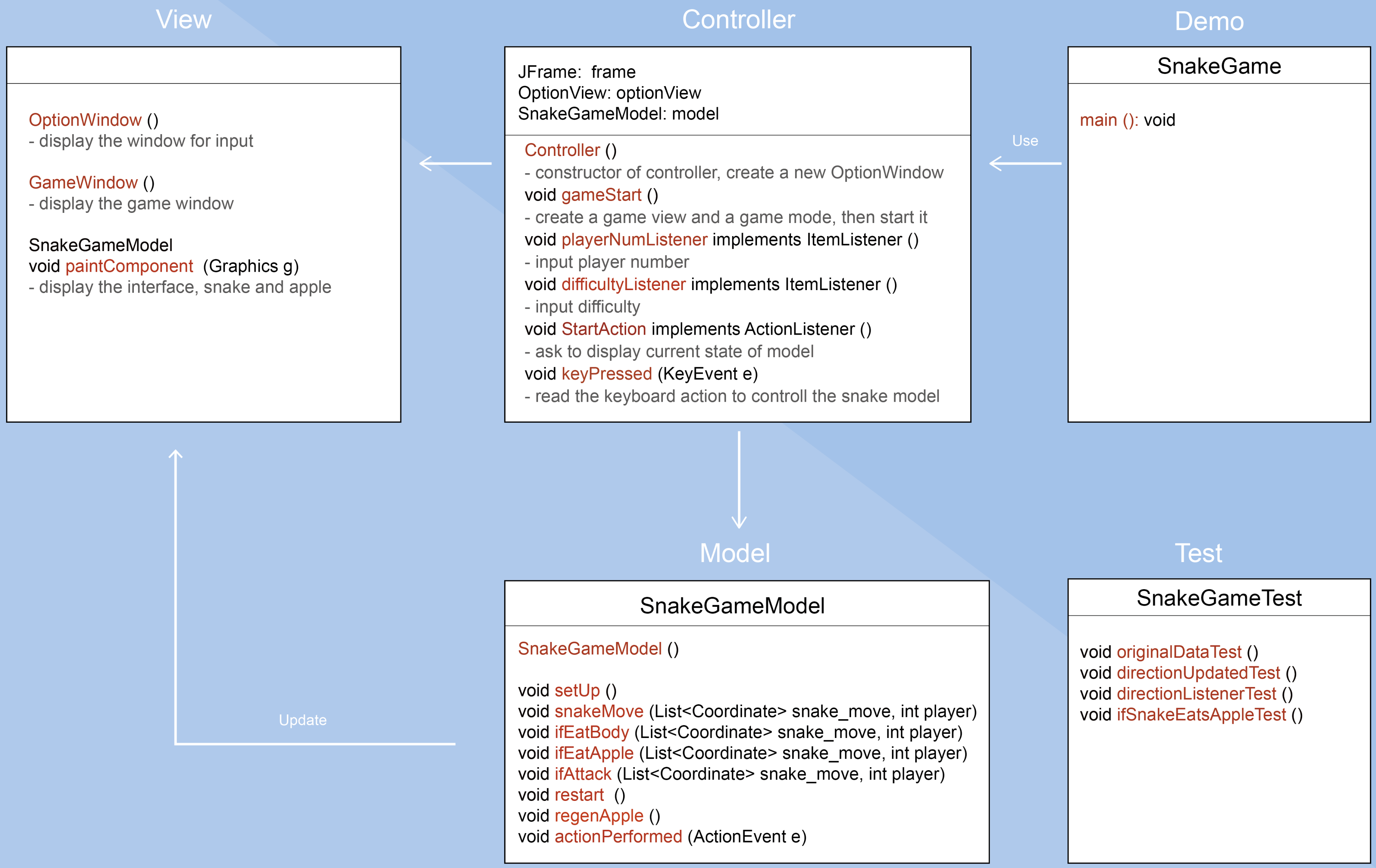
Oops! Player2 attacked Player1, Player2 loses.
Press SPACE to restart

Two Players Mode

1 PROJECT DEMO

Let's play the game !

2 MVC MODE DESIGN



View

OptionWindow ()
- display the window for input

GameWindow ()
- display the game window

SnakeGameModel
void paintComponent (Graphics g)
- display the interface, snake and apple

Controller

JFrame: frame
OptionView: optionView
SnakeGameModel: model

Controller ()
- constructor of controller, create a new OptionWindow
void gameStart ()
- create a game view and a game mode, then start it
void playerNumListener implements ItemListener ()
- input player number
void difficultyListener implements ItemListener ()
- input difficulty
void StartAction implements ActionListener ()
- ask to display current state of model
void keyPressed (KeyEvent e)
- read the keyboard action to controll the snake model

Demo

SnakeGame

main (): void

Model

SnakeGameModel

SnakeGameModel ()

void setUp ()
void snakeMove (List<Coordinate> snake_move, int player)
void ifEatBody (List<Coordinate> snake_move, int player)
void ifEatApple (List<Coordinate> snake_move, int player)
void ifAttack (List<Coordinate> snake_move, int player)
void restart ()
void regenApple ()
void actionPerformed (ActionEvent e)

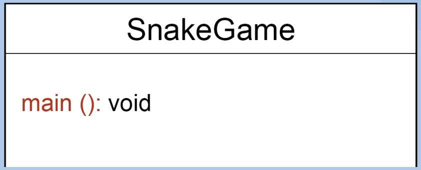
Test

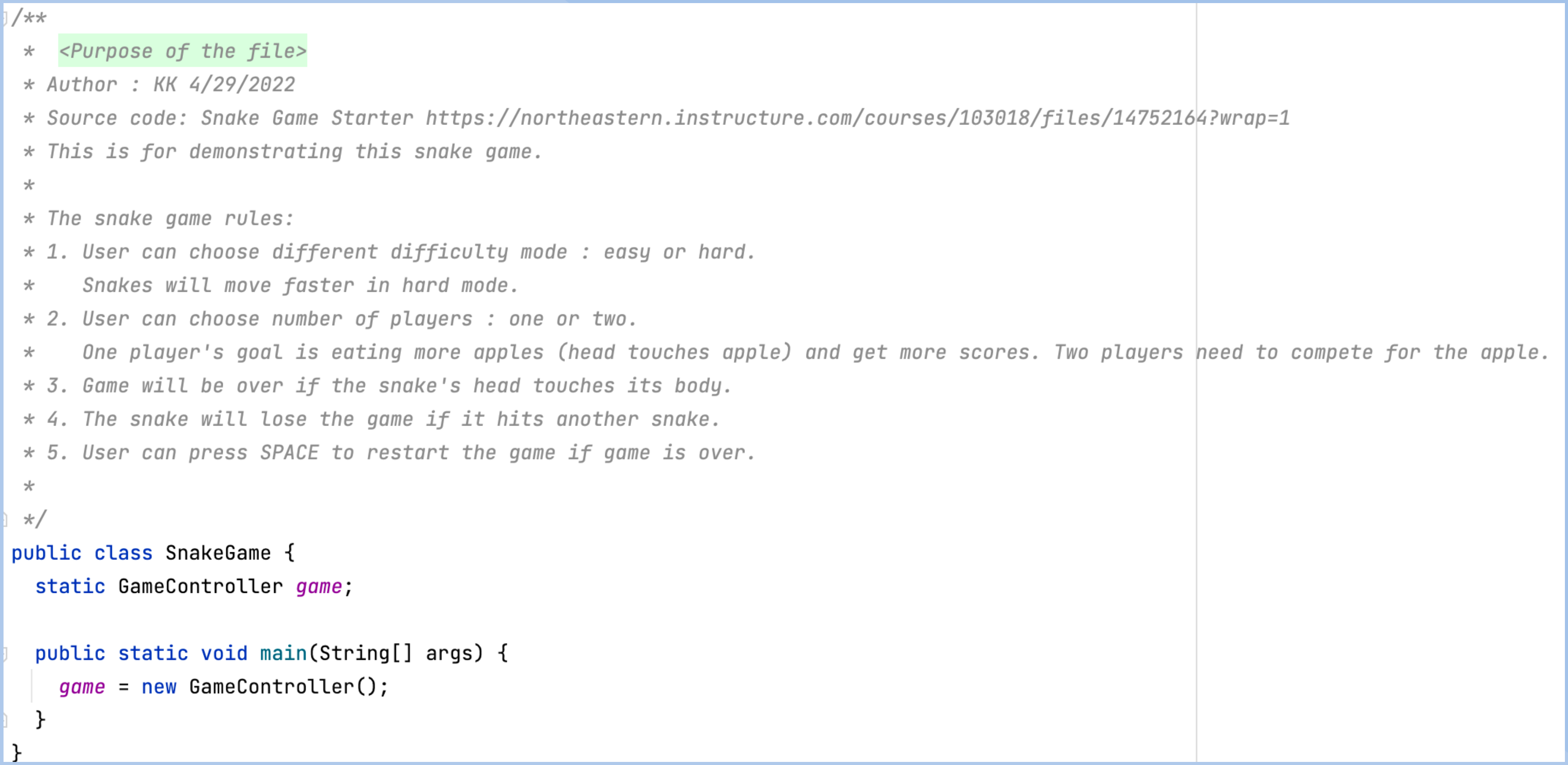
SnakeGameTest

void originalDataTest ()
void directionUpdatedTest ()
void directionListenerTest ()
void ifSnakeEatsAppleTest ()

03 CODE DESIGN

Demo

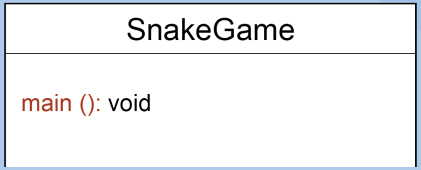


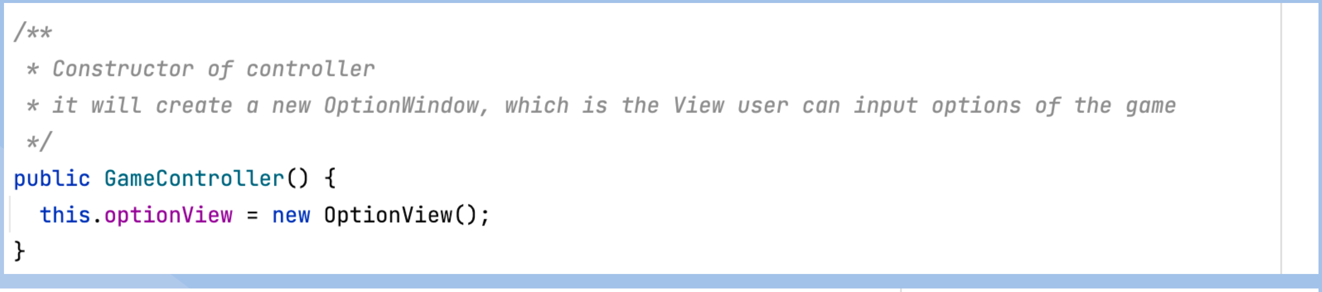


```java
/**
 *  <Purpose of the file>
 * Author : KK 4/29/2022
 * Source code: Snake Game Starter https://northeastern.instructure.com/courses/103018/files/14752164?wrap=1
 * This is for demonstrating this snake game.
 *
 * The snake game rules:
 * 1. User can choose different difficulty mode : easy or hard.
 *    Snakes will move faster in hard mode.
 * 2. User can choose number of players : one or two.
 *    One player's goal is eating more apples (head touches apple) and get more scores. Two players need to compete for the apple.
 * 3. Game will be over if the snake's head touches its body.
 * 4. The snake will lose the game if it hits another snake.
 * 5. User can press SPACE to restart the game if game is over.
 *
 */
public class SnakeGame {
  static GameController game;

  public static void main(String[] args) {
    game = new GameController();
  }
}
```

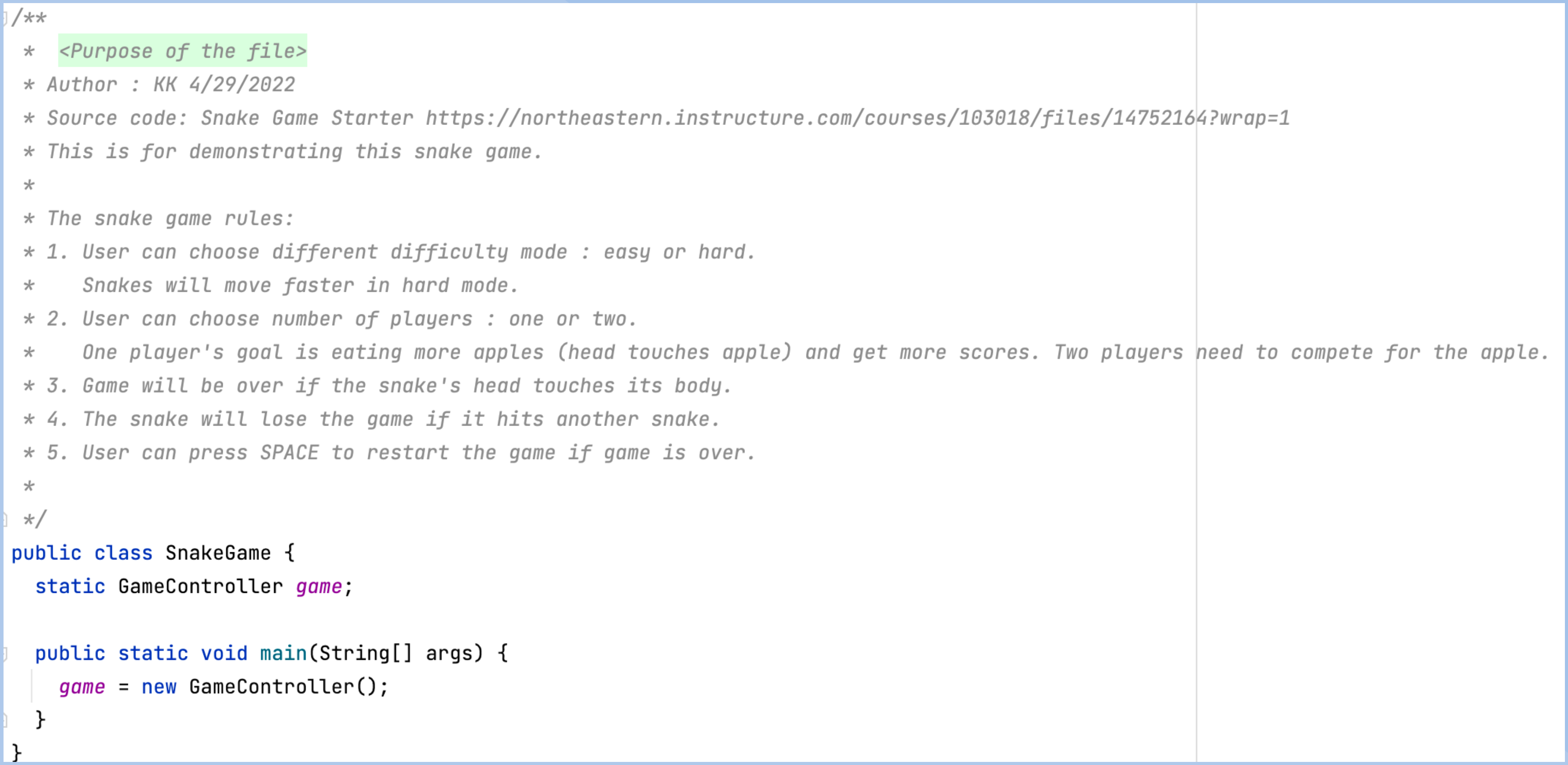
3 CODE DESIGN

Demo





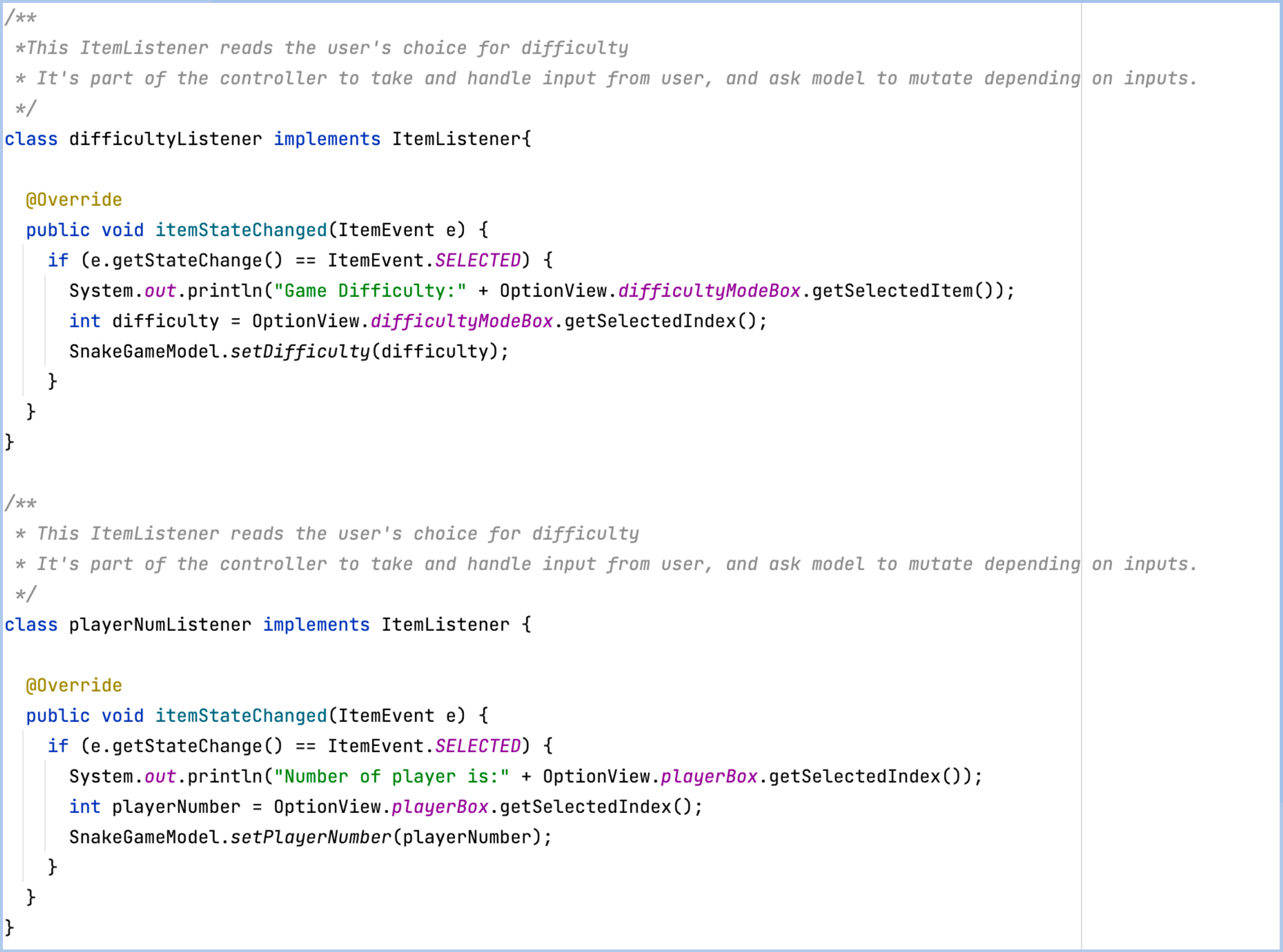
```
/**
 * Constructor of controller
 * it will create a new OptionWindow, which is the View user can input options of the game
 */
public GameController() {
  this.optionView = new OptionView();
}
```



```
/**
 *  <Purpose of the file>
 * Author : KK 4/29/2022
 * Source code: Snake Game Starter https://northeastern.instructure.com/courses/103018/files/14752164?wrap=1
 * This is for demonstrating this snake game.
 *
 * The snake game rules:
 * 1. User can choose different difficulty mode : easy or hard.
 *    Snakes will move faster in hard mode.
 * 2. User can choose number of players : one or two.
 *    One player's goal is eating more apples (head touches apple) and get more scores. Two players need to compete for the apple.
 * 3. Game will be over if the snake's head touches its body.
 * 4. The snake will lose the game if it hits another snake.
 * 5. User can press SPACE to restart the game if game is over.
 *
 */
public class SnakeGame {
  static GameController game;

  public static void main(String[] args) {
    game = new GameController();
  }
}
```

3 CODE DESIGN

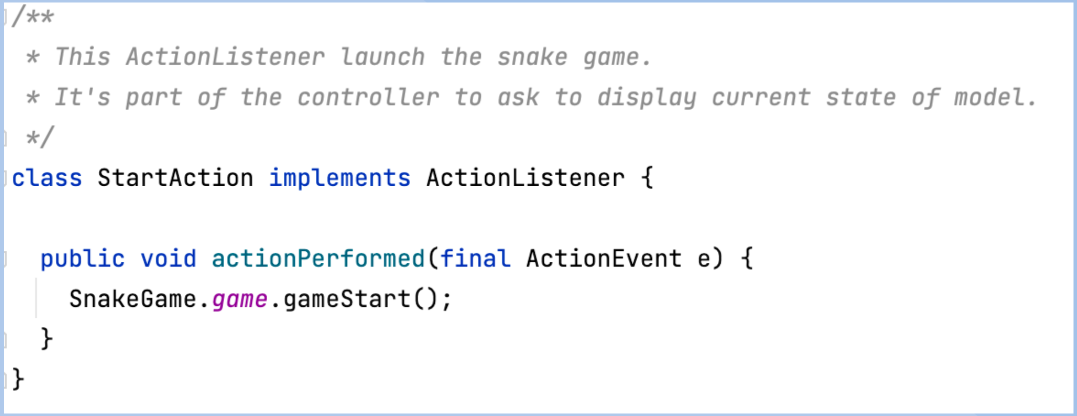
Controller - ItemListener : read options – pass to the model



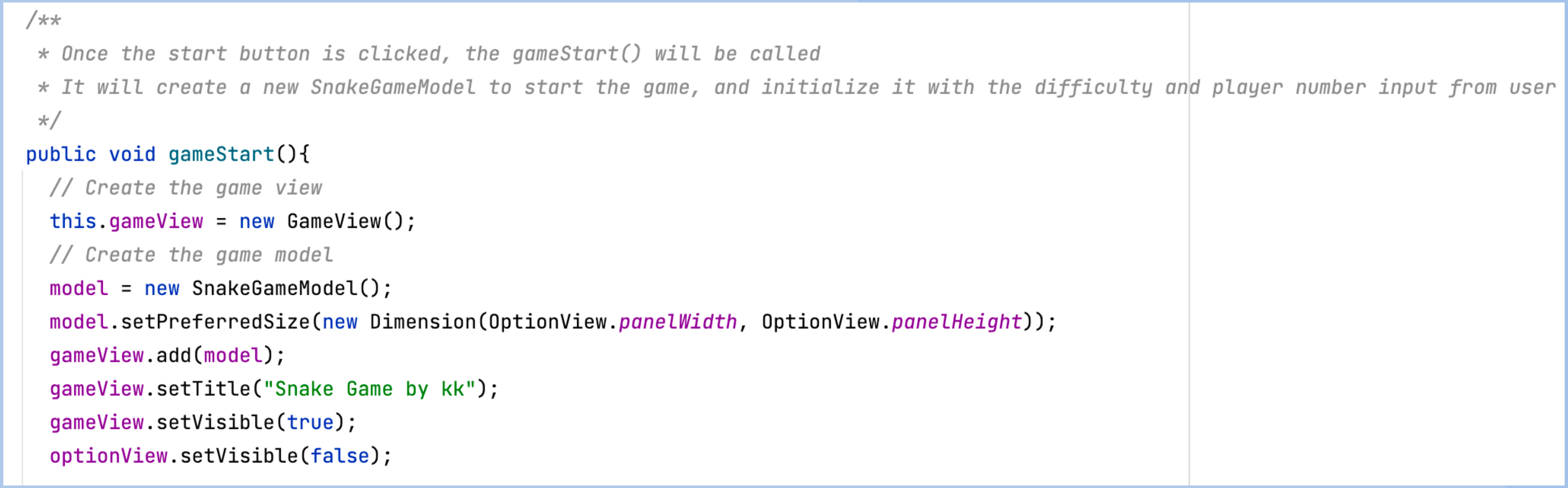
```
/**
 *This ItemListener reads the user's choice for difficulty
 * It's part of the controller to take and handle input from user, and ask model to mutate depending on inputs.
 */
class difficultyListener implements ItemListener{

  @Override
  public void itemStateChanged(ItemEvent e) {
    if (e.getStateChange() == ItemEvent.SELECTED) {
      System.out.println("Game Difficulty:" + OptionView.difficultyModeBox.getSelectedItem());
      int difficulty = OptionView.difficultyModeBox.getSelectedIndex();
      SnakeGameModel.setDifficulty(difficulty);
    }
  }
}

/**
 * This ItemListener reads the user's choice for difficulty
 * It's part of the controller to take and handle input from user, and ask model to mutate depending on inputs.
 */
class playerNumListener implements ItemListener {

  @Override
  public void itemStateChanged(ItemEvent e) {
    if (e.getStateChange() == ItemEvent.SELECTED) {
      System.out.println("Number of player is:" + OptionView.playerBox.getSelectedIndex());
      int playerNumber = OptionView.playerBox.getSelectedIndex();
      SnakeGameModel.setPlayerNumber(playerNumber);
    }
  }
}
```

3 CODE DESIGN

Controller - Click Start Button – ActionListner : Call Controller.gameStart

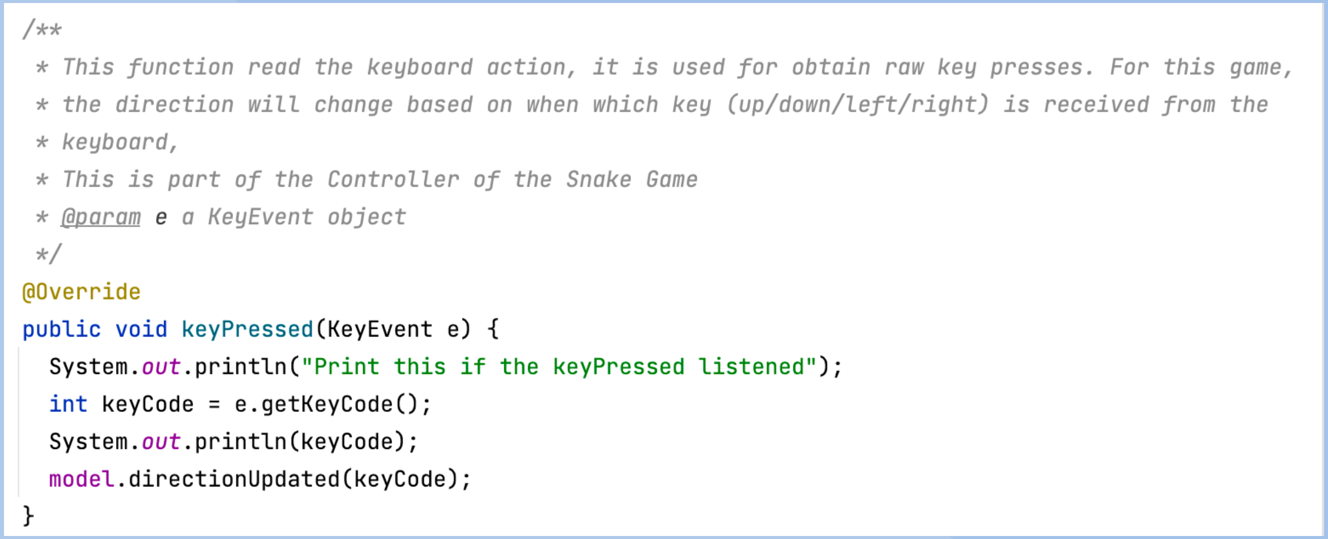


```java
/**
 * This ActionListener launch the snake game.
 * It's part of the controller to ask to display current state of model.
 */
class StartAction implements ActionListener {

  public void actionPerformed(final ActionEvent e) {
    SnakeGame.game.gameStart();
  }
}
```

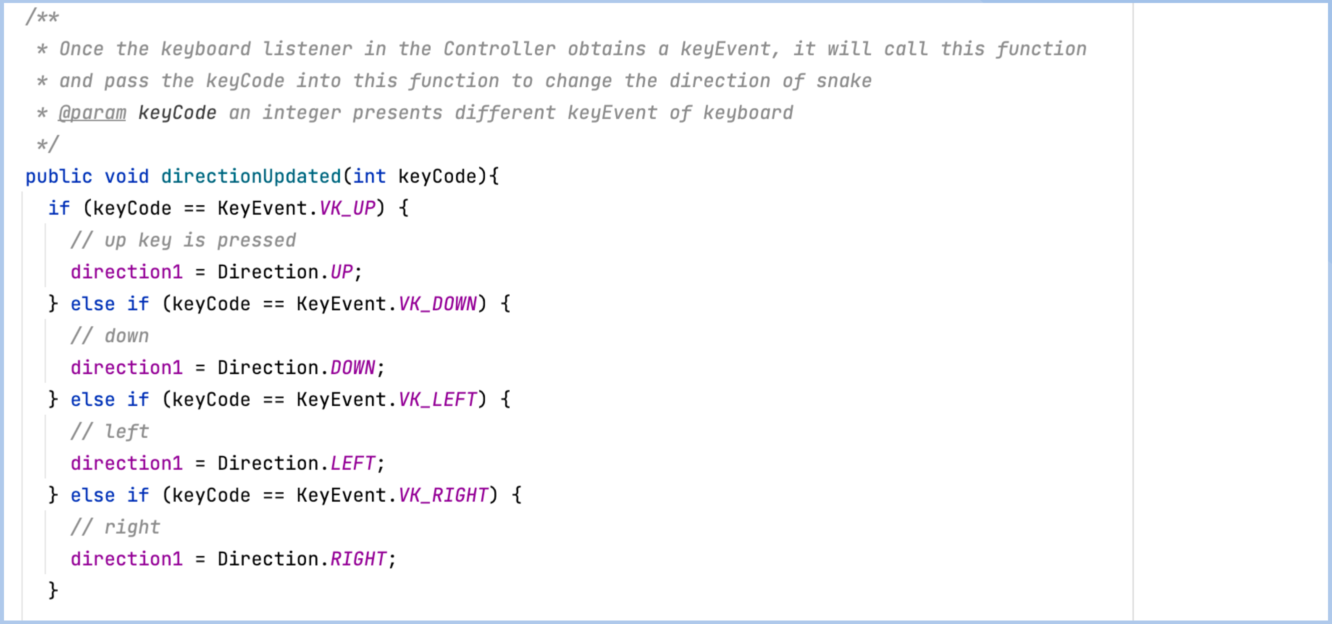


```java
/**
 * Once the start button is clicked, the gameStart() will be called
 * It will create a new SnakeGameModel to start the game, and initialize it with the difficulty and player number input from user
 */
public void gameStart(){
  // Create the game view
  this.gameView = new GameView();
  // Create the game model
  model = new SnakeGameModel();
  model.setPreferredSize(new Dimension(OptionView.panelWidth, OptionView.panelHeight));
  gameView.add(model);
  gameView.setTitle("Snake Game by kk");
  gameView.setVisible(true);
  optionView.setVisible(false);
```

3 CODE DESIGN



```java
/**
 * This function read the keyboard action, it is used for obtain raw key presses. For this game,
 * the direction will change based on when which key (up/down/left/right) is received from the
 * keyboard,
 * This is part of the Controller of the Snake Game
 * @param e a KeyEvent object
 */
@Override
public void keyPressed(KeyEvent e) {
  System.out.println("Print this if the keyPressed listened");
  int keyCode = e.getKeyCode();
  System.out.println(keyCode);
  model.directionUpdated(keyCode);
}
```



```java
/**
 * Once the keyboard listener in the Controller obtains a keyEvent, it will call this function
 * and pass the keyCode into this function to change the direction of snake
 * @param keyCode an integer presents different keyEvent of keyboard
 */
public void directionUpdated(int keyCode){
  if (keyCode == KeyEvent.VK_UP) {
    // up key is pressed
    direction1 = Direction.UP;
  } else if (keyCode == KeyEvent.VK_DOWN) {
    // down
    direction1 = Direction.DOWN;
  } else if (keyCode == KeyEvent.VK_LEFT) {
    // left
    direction1 = Direction.LEFT;
  } else if (keyCode == KeyEvent.VK_RIGHT) {
    // right
    direction1 = Direction.RIGHT;
  }
```

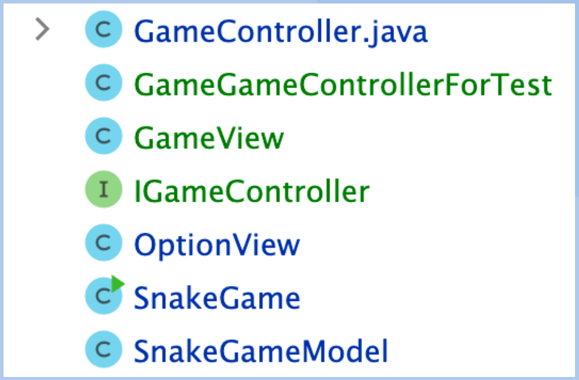
Controller - KeyListener

- Read Keyboard
- Pass it to the model
- Change the direction

REFLECTION

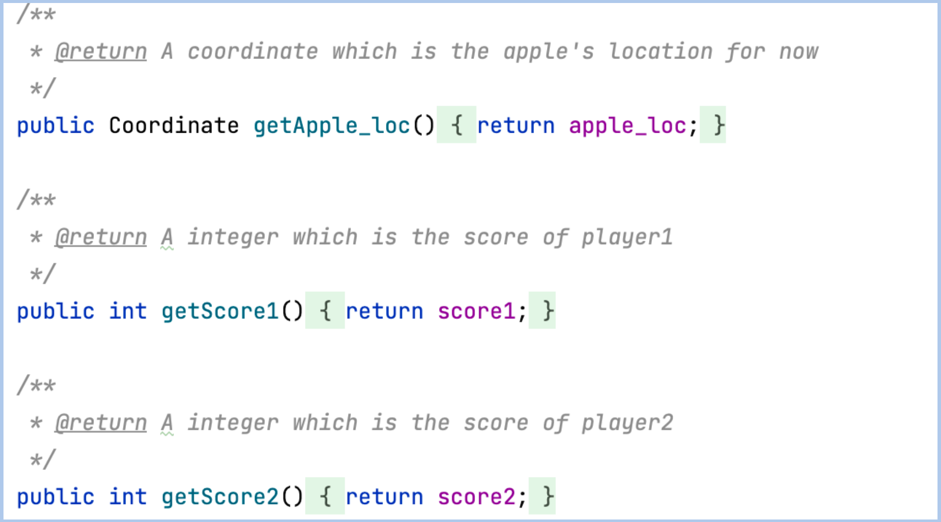
Ideas about OOD:

- MVC



GameController.java
GameGameControllerForTest
GameView
IGameController
OptionView
SnakeGame
SnakeGameModel

- Encapsulation

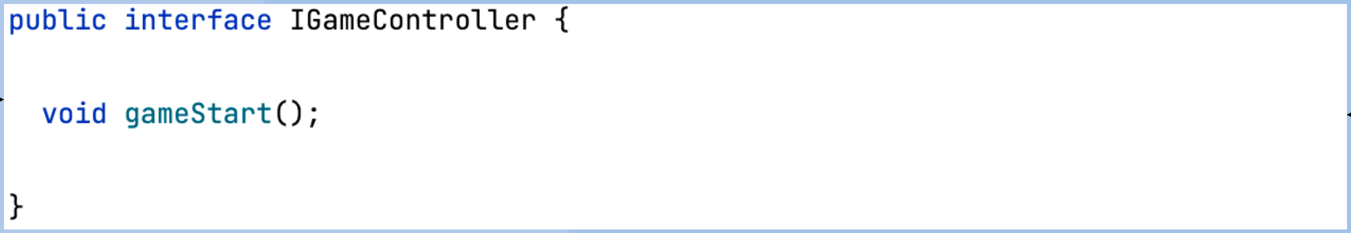


```
/**
 * @return A coordinate which is the apple's location for now
 */
public Coordinate getApple_loc() { return apple_loc; }

/**
 * @return A integer which is the score of player1
 */
public int getScore1() { return score1; }

/**
 * @return A integer which is the score of player2
 */
public int getScore2() { return score2; }
```

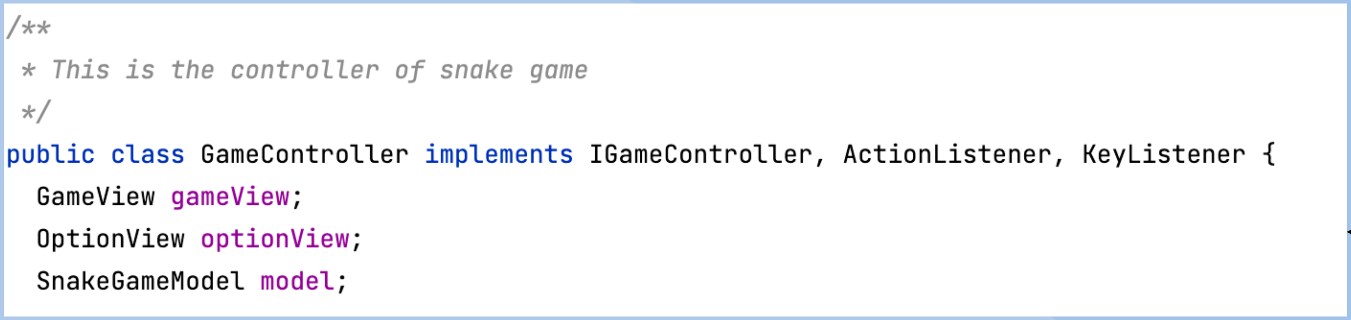
REFLECTION

Ideas about OOD:

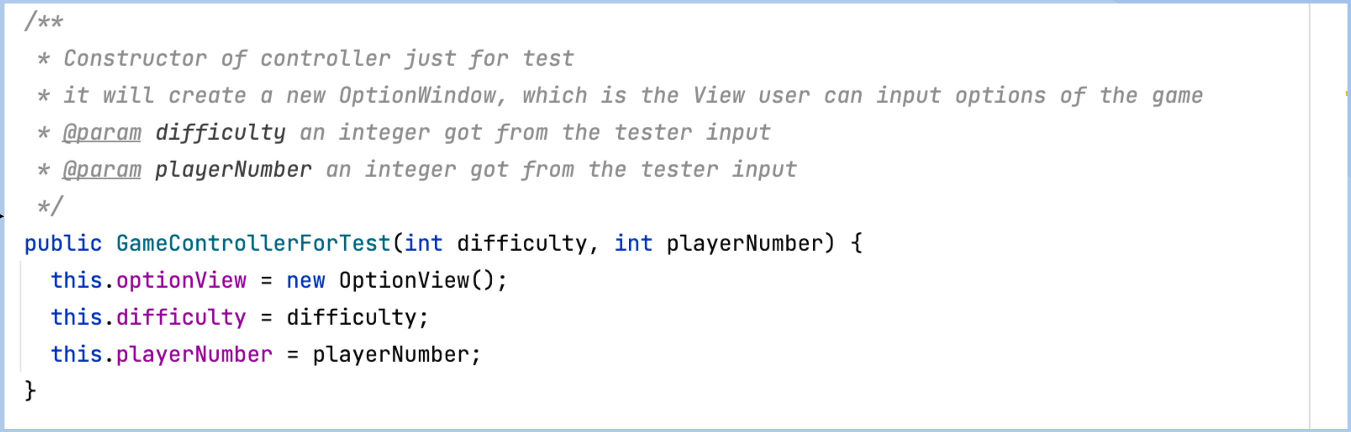
- Polymorphism



```java
public interface IGameController {

  void gameStart();

}
```

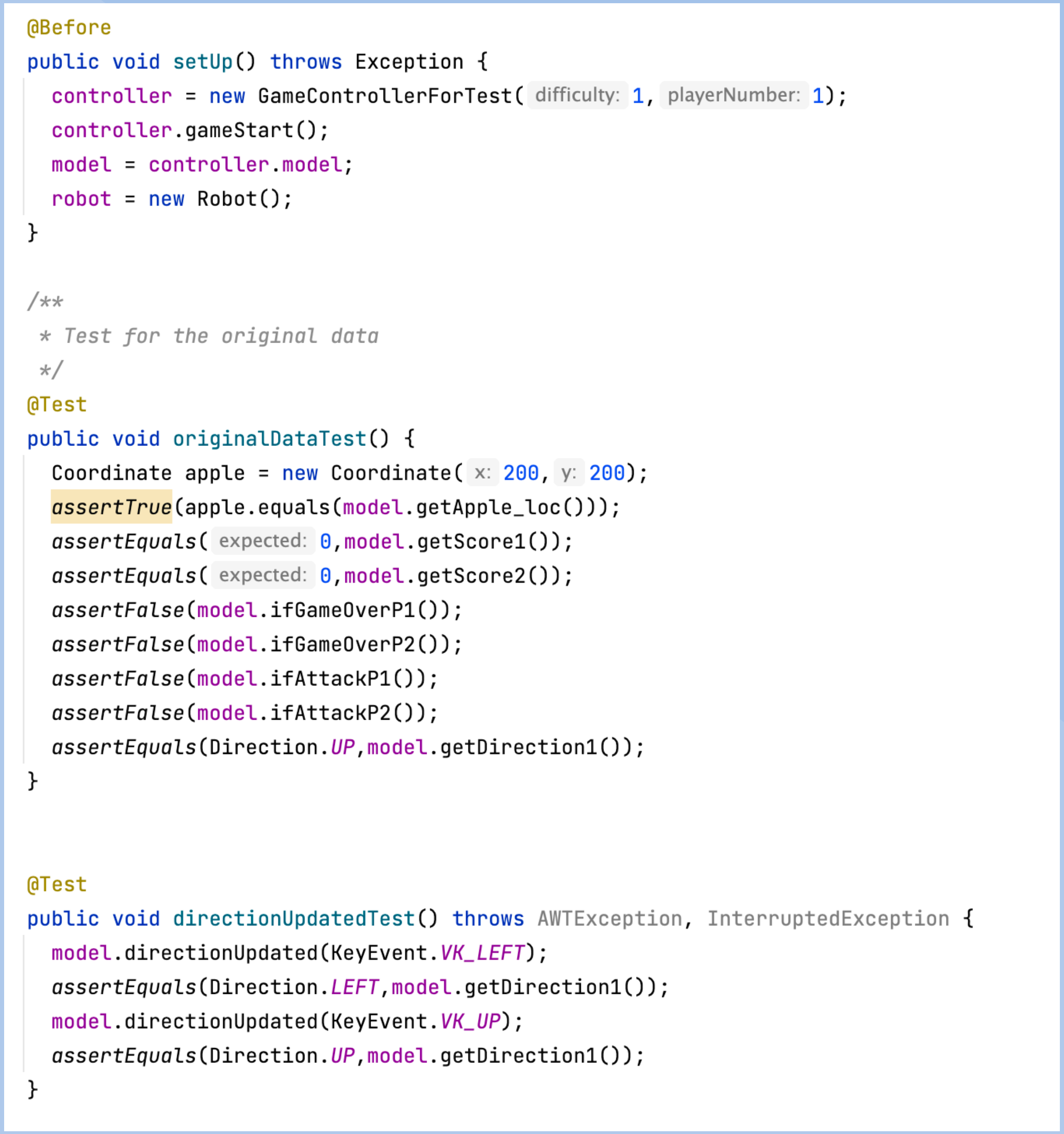


```java
/**
 * This is the controller of snake game
 */
public class GameController implements IGameController, ActionListener, KeyListener {
  GameView gameView;
  OptionView optionView;
  SnakeGameModel model;
```



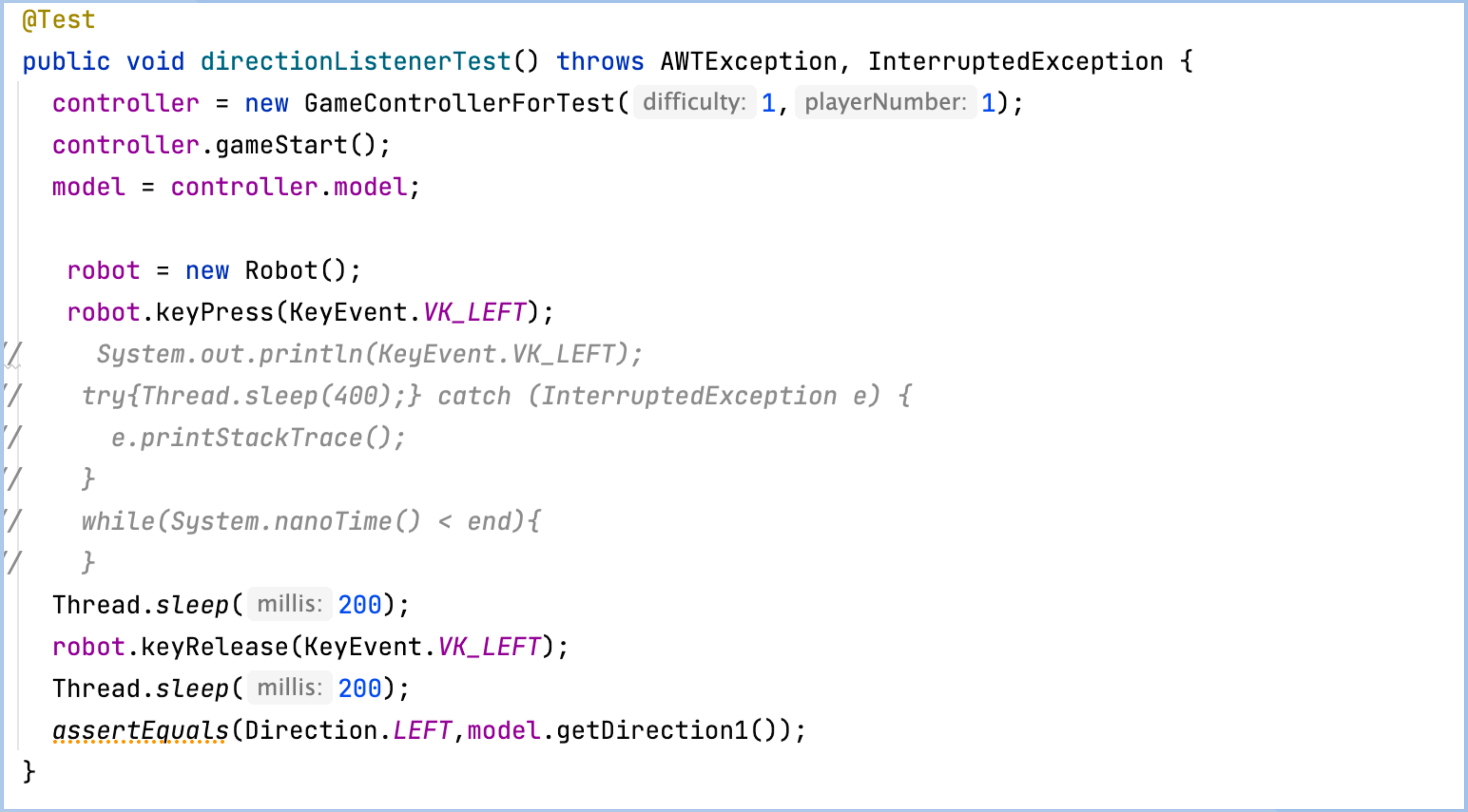
```java
/**
 * Constructor of controller just for test
 * it will create a new OptionWindow, which is the View user can input options of the game
 * @param difficulty an integer got from the tester input
 * @param playerNumber an integer got from the tester input
 */
public GameControllerForTest(int difficulty, int playerNumber) {
  this.optionView = new OptionView();
  this.difficulty = difficulty;
  this.playerNumber = playerNumber;
}
```

REFLECTION Junit Test



```
@Before
public void setUp() throws Exception {
  controller = new GameControllerForTest( difficulty: 1, playerNumber: 1);
  controller.gameStart();
  model = controller.model;
  robot = new Robot();
}

/**
 * Test for the original data
 */
@Test
public void originalDataTest() {
  Coordinate apple = new Coordinate( x: 200, y: 200);
  assertTrue(apple.equals(model.getApple_loc()));
  assertEquals( expected: 0,model.getScore1());
  assertEquals( expected: 0,model.getScore2());
  assertFalse(model.ifGameOverP1());
  assertFalse(model.ifGameOverP2());
  assertFalse(model.ifAttackP1());
  assertFalse(model.ifAttackP2());
  assertEquals(Direction.UP,model.getDirection1());
}


@Test
public void directionUpdatedTest() throws AWTException, InterruptedException {
  model.directionUpdated(KeyEvent.VK_LEFT);
  assertEquals(Direction.LEFT,model.getDirection1());
  model.directionUpdated(KeyEvent.VK_UP);
  assertEquals(Direction.UP,model.getDirection1());
}
```

REFLECTION Junit Test



```
@Test
public void directionListenerTest() throws AWTException, InterruptedException {
  controller = new GameControllerForTest( difficulty: 1, playerNumber: 1);
  controller.gameStart();
  model = controller.model;

   robot = new Robot();
   robot.keyPress(KeyEvent.VK_LEFT);
/      System.out.println(KeyEvent.VK_LEFT);
/     try{Thread.sleep(400);} catch (InterruptedException e) {
/       e.printStackTrace();
/     }
/     while(System.nanoTime() < end){
/     }
  Thread.sleep( millis: 200);
  robot.keyRelease(KeyEvent.VK_LEFT);
  Thread.sleep( millis: 200);
  assertEquals(Direction.LEFT,model.getDirection1());
}
```

REFLECTION

Tools and Techniques:

- GUI Swing
- KeyListener — Control the snake
- ActionListener — Click start button
- ItemListener — Read ComboBox option

Challenges:

- Separate the Model, View and Controller
- Write the Junit Test

Future Extensions:

- Game rule: put obstacles
- Code: polish organization and Junit Test

- Thank you !