Geometry Dash

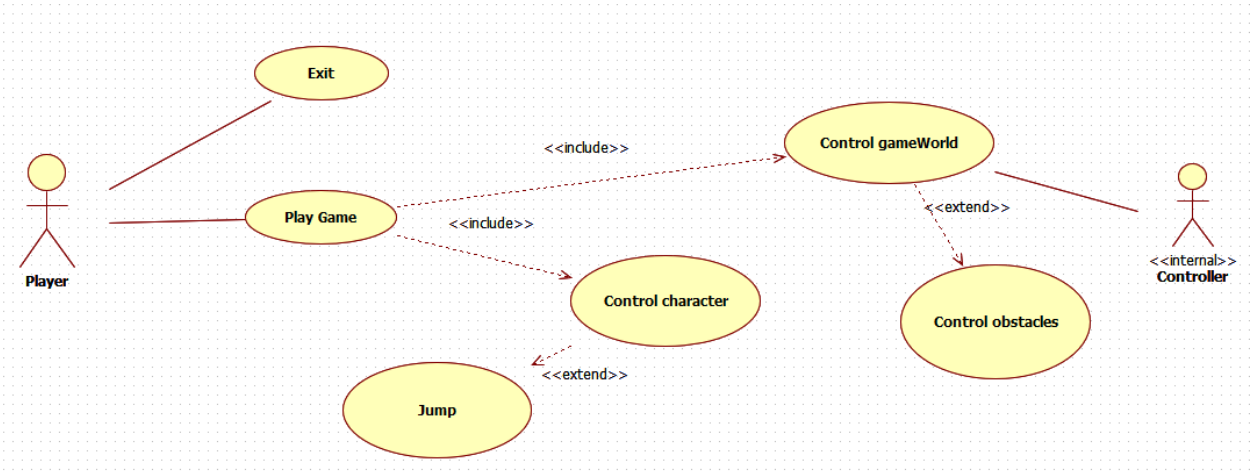
Programming 3 Project

1 Description of the task

This program is to implement the popular game “Geometry dash” using the “Swing” graphical user interface . It is a horizontal runner-style game. The player takes the form of a cube. Using one-touch gameplay, the player must try to navigate through a series of interactive obstacles such as cubes or triangles to reach the end of level without crashing.

2 Use cases

2.1 Use case diagram



3 Structural description

3.1 Description of the classes

1.1.1 `public class Panel extends JPanel implements Runnable`

Responsibilities

Extends the JPanel class, implements Runnable interface . Performs main functions of the gameplay, the game screen states.

Attributes

`public static enum GameState`	Has 3 main states of the game: MENU, GAME, END
`public static GameState state`	The state of the game that's why static, initially MENU
`private Thread thread`	The thread to update changes of the game objects during the game process.
`private Player player`	The Geometry dash main character
`private Ground ground`	The ground on which the main character moves, jumps
`private Ground background`	The background of the game
`private Obstacles obstacles`	The array of the spikes which player may face during the game
`private Menu menu`	The menu of the game
`private PlayerDatabase playerDatabase`	The serializable attribute

Methods

`public void start()`	Starts the thread of the game, calls the run() function of the Panel
`public void run()`	Updates the the gameobjects and repaint the panel every thread time
`public void update()`	Updates every game object sequentially if the gamestate is GAME
`public void paint(Graphics g)`	Usage of Graphics class to draw the game objects. Called every time when the repaint function implemented.

1.1.2 `public class Window extends JFrame`

Responsibilities

Cretes the Window for the game

Attributes

`private Panel panel`	The panel of the to show the content to the screen

`private KL keyListener`	The Key Listener to detect pressing on the buttons
`private ML mouseListener`	Mouse Listener to detect mouse clicking
`private static Window window = null`	The window

Methods

`public void start()`	Calls the panel's start

1.1.3 `public class ML implements MouseListener`

Responsibilities

Mouse Listener to detect mouse clicking for buttons

Attributes

`private Panel panel`	The panel of the Window Frame

Methods

`public void mousePressed(MouseEvent e)`	Change the state of the game depending on which button is pressed. Calls the JOptionPane class to record the name of the player.

1.1.4 `public class KL extends KeyAdapter implements KeyListener`

Responsibilities

The Key Listener to detect pressing on the buttons, so the state is changed or the player jumps

Attributes

`private Panel panel`	<description>

Methods

`public void keyReleased(KeyEvent e)`	State changes: MENU -> GAME, END->MENU. GAME -> Player jumps.

1.1.5 `public class Menu`

Responsibilities

Draws the menu interface to the Panel.

Attributes

`private Rectangle playButton`	Implements the Play button after which the actual game starts, state changes to GAME

`private Rectangle quitButton`	Implements the Exit button after which the the program terminates, state changes to END

Methods

`public void draw(Graphics g)`	Rectangle, Image using

1.1.6 `public class Player implements Comparable<gameObjects.Player>, Serializable`

Responsibilities

The main character of the game. The Player controls the character.

Attributes

`private float x = Constants.PLAYER_POSITION_X;`	Player position in X-axis
`private float y = Constants.PLAYER_POSITION_Y;`	Player position in Y-axis
`private float y_velocity = Constants.PLAYER_JUMP_VELOCITY;`	Player's jump velocity in Y-axis
`private Rectangle rectangle;`	Rectangle making bounds of the main character to detect collision
`private boolean isAlive = true;`	To check if the character alive
`private int score = 0;`	the score of the player, initially 0
`private String playerName = "";`	The name of the player which he/she entered at the beginning of the game

Methods

`public int compareTo(gameObjects.Player other)`	Compares the score of the players, used to order the list of the players in database
`public void update()`	Updates the position of the player
`public void draw(Graphics g)`	Draws the main character according to the current position
`public void jump()`	Changes th position in Y-axis using the jumping velocity
`public void incrementScore()`	Increments the score by 1

1.1.7 `public class Spike`

Responsibilities

Implements the triangles on the ground. If the player collides with one of them the game is over.

Attributes

`private BufferedImage image;`	Assigns the image to the Spike
`private int x, y`	The position of the spike
`private Rectangle rectangle`	The Rectangle of the position

Methods

`public void update()`	Updates the position of the spike
`public void draw(Graphics g)`	Draws the spike
`public boolean isOvercome()`	Returns true if the spike out of left side screen

1.1.8 `public class Obstacles`

Responsibilities

Creates some random spikes on the screen

Attributes

`private List<Spike> spikes`	The list of the Spikes on the screen
`private Player player`	The main character to increment the score

Methods

`public Spike getRandomSpike()`	returns the random spike (random picture of the spike)
`public void update()`	Updates every spike in the list, increments player score if the spike overcome. Detects the collision
`public void draw(Graphics g)`	draws the spikes of the list

1.1.9 `public class Ground`

Responsibilities

Represents the ground and also the background of the game

Attributes

`private ArrayList<PictureGround> listPictures`	Contains the pictures of the ground which are on the screen
`private float speed;`	The speed the ground goes back

Methods

`public void draw(Graphics g)`	Draws the ground
`public void update()`	Updates the position of the ground according the speed
`public class PictureGround`	Nested class to present picture of the ground

1.1.10 `public class PlayerDatabase implements Serializable`

Responsibilities

To store the previous players and make a dashboard. Load the previous players using deserialization, add the current player, and rewrite the file with updated List of players using serialization.

Attributes

`private String filename`	The file name where to store the players list.
`private List<Player> playerList;`	List containing all players

Methods

`public void append(Player player)`	Adds the player to the list of all players -> serialize.
`public void serialize()`	Writes the List to the file
`public ArrayList<Player> deserialize`	Reads the List from the file

1.1.11 Picture

Responsibilities

To get the image shortly

Methods

`public static BufferedImage getPicture(String path)`	Returns the image according to the path

1.1.12 Constants

Responsibilities

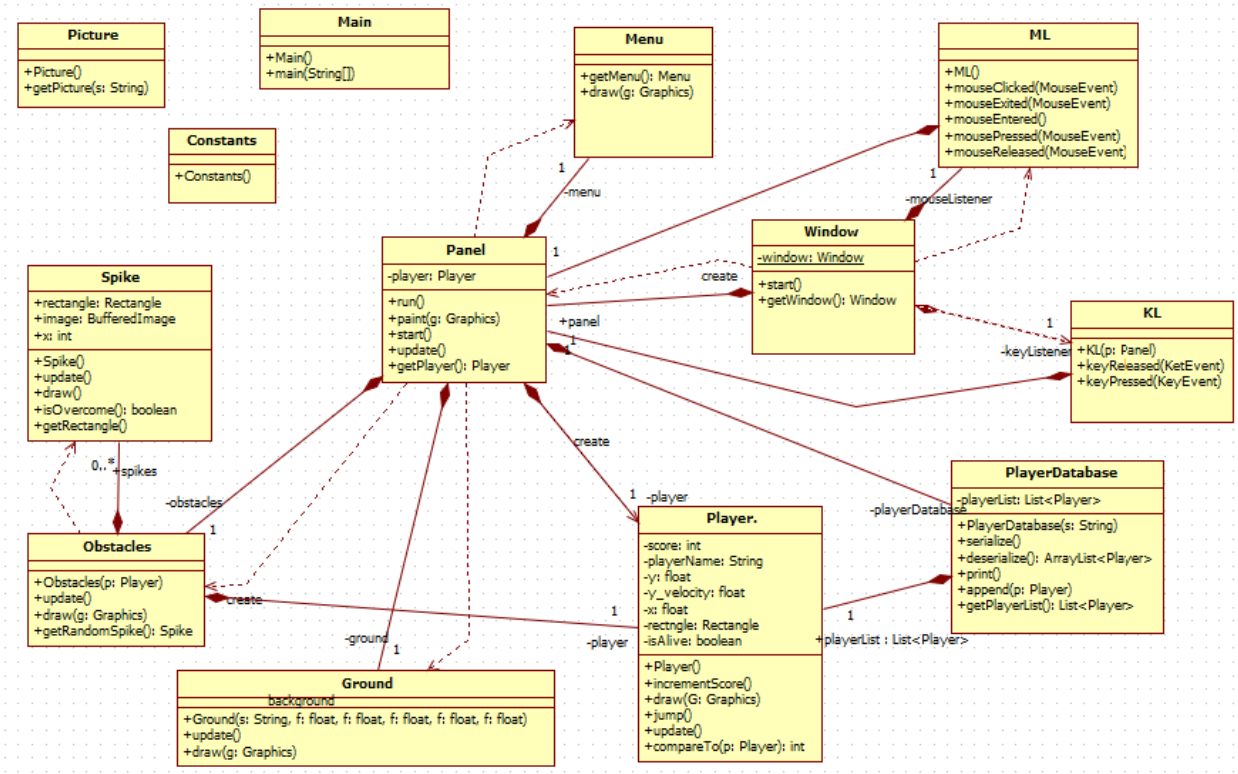
Contains some useful constants.

1.1.13 Main

Responsibilities

Runs the program, creates the instance of the Window and calls the thread.

1.2 Class Diagram



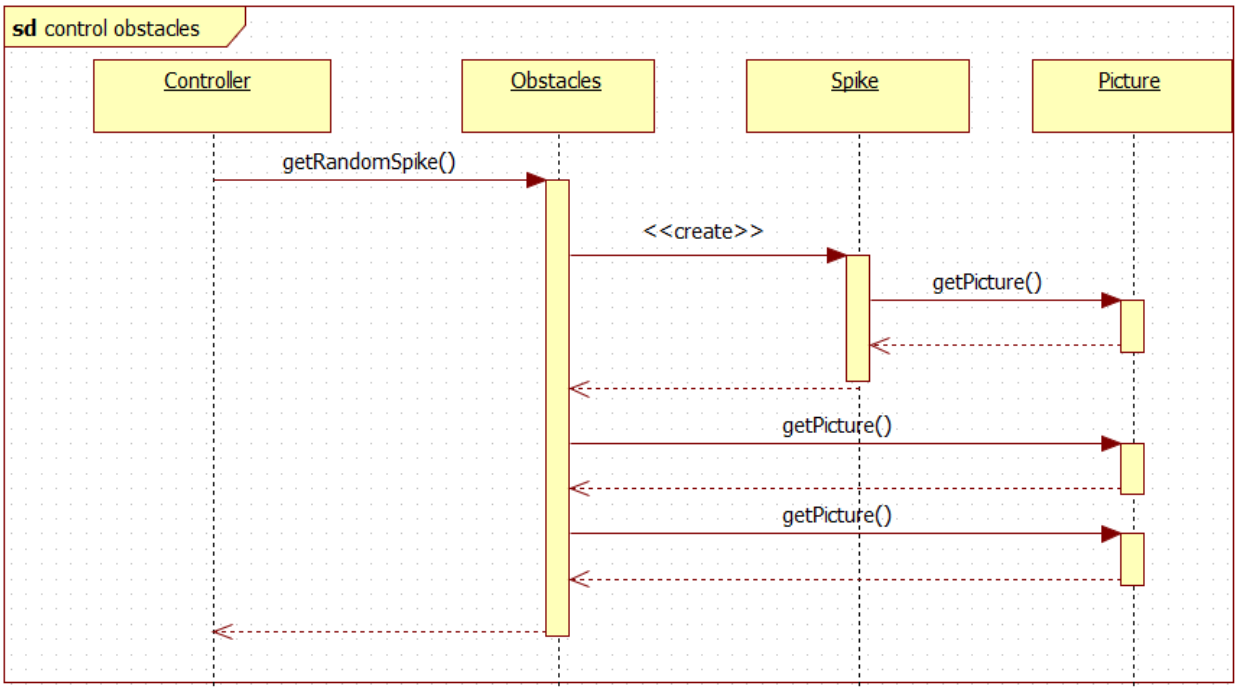
Note: some attributes, getters and setters was removed for simplicity

4 Behavioral description

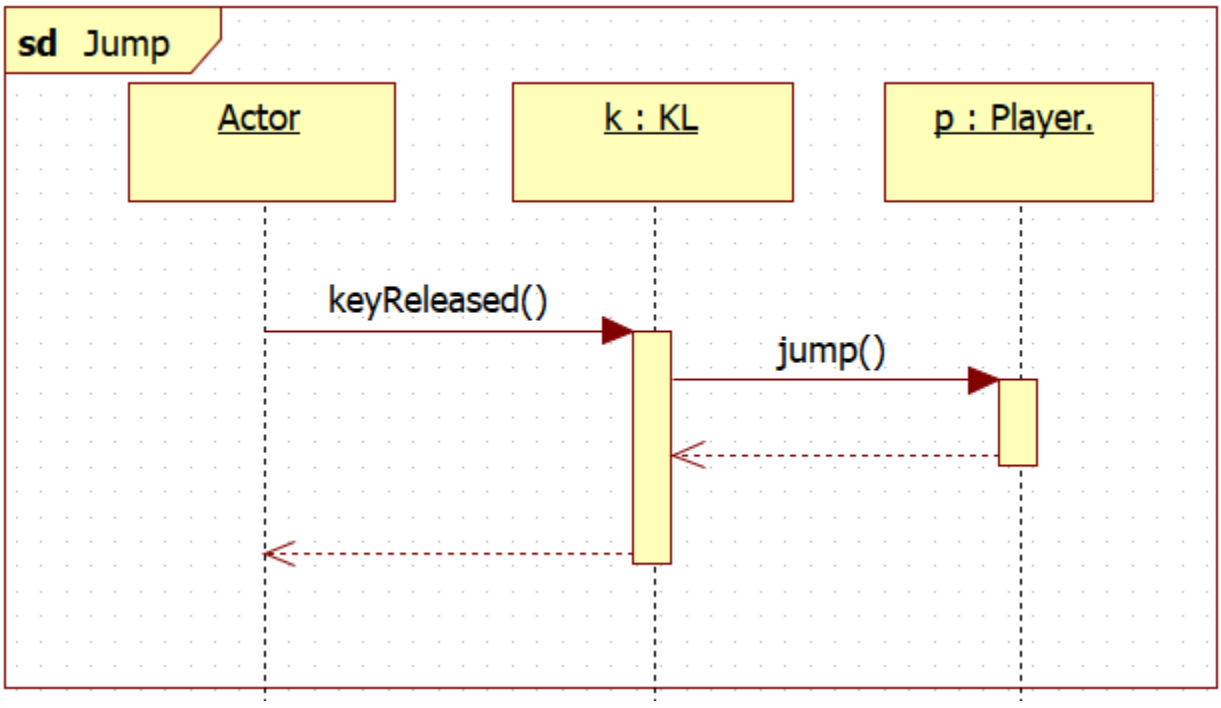
4.1 Sequence Diagrams

<for each sequence the following section:>

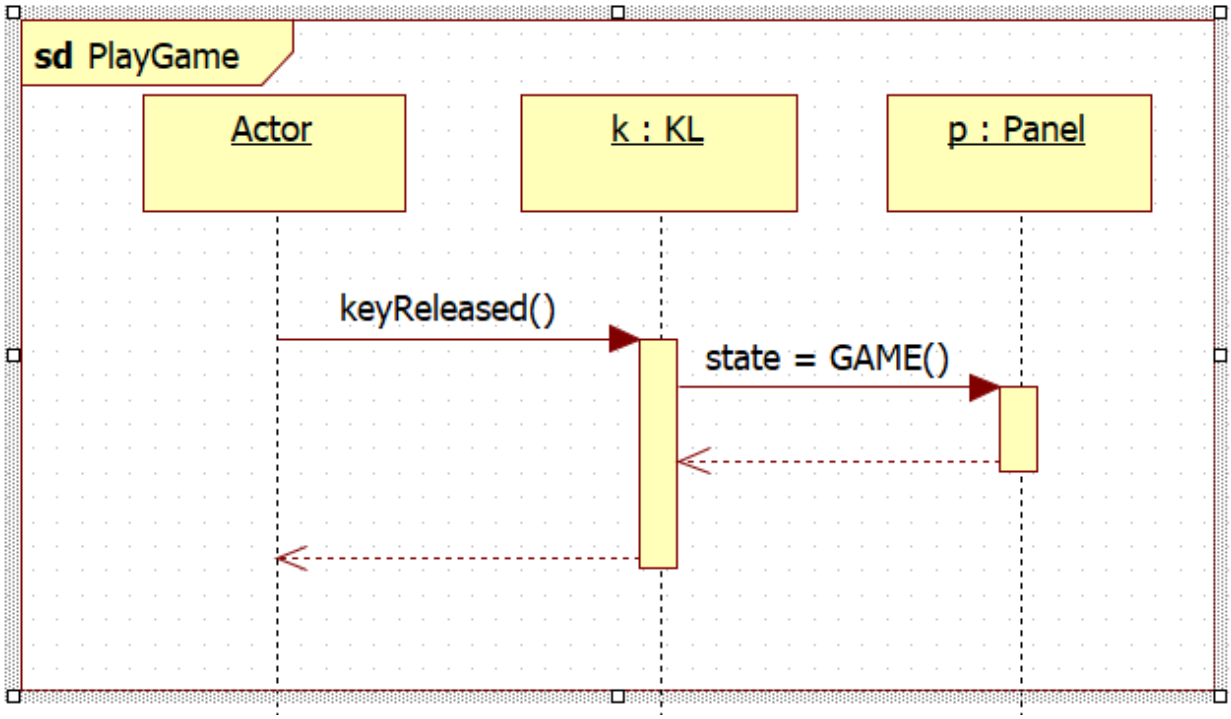
1.2.1 Controller controls the obstacles



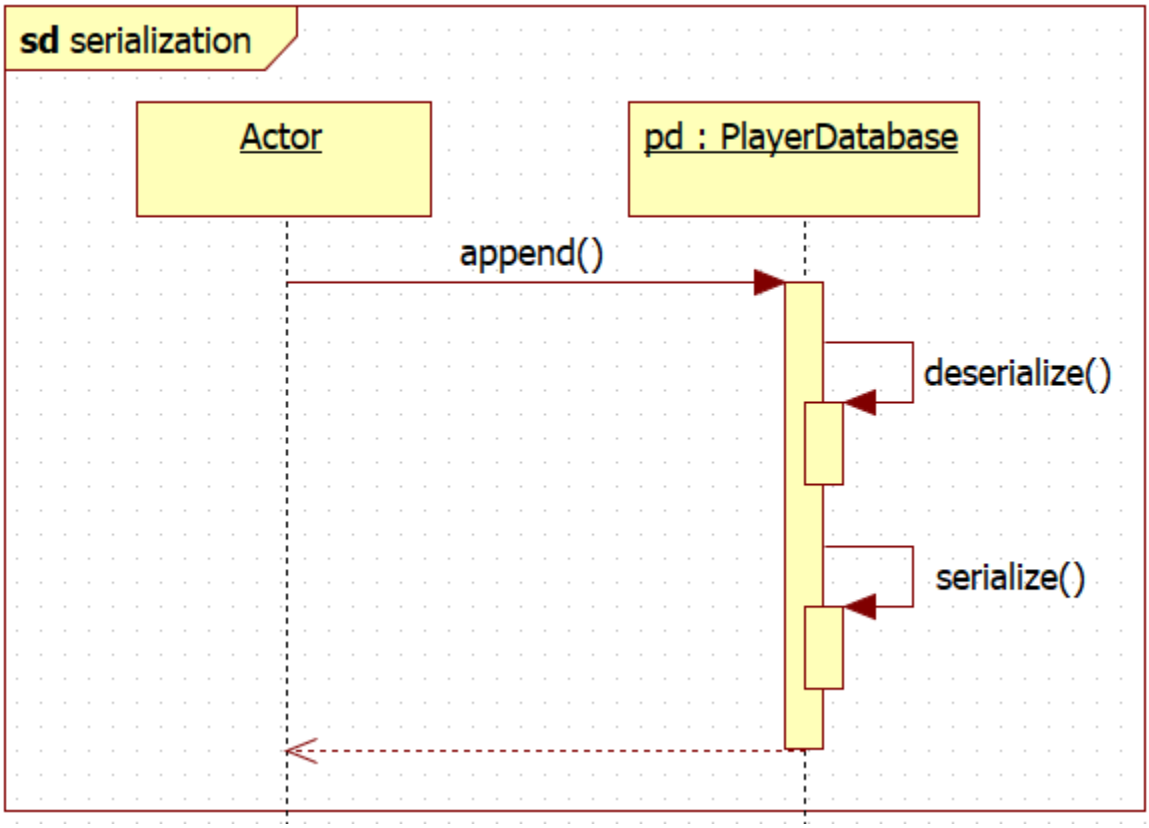
1.2.2 Jump



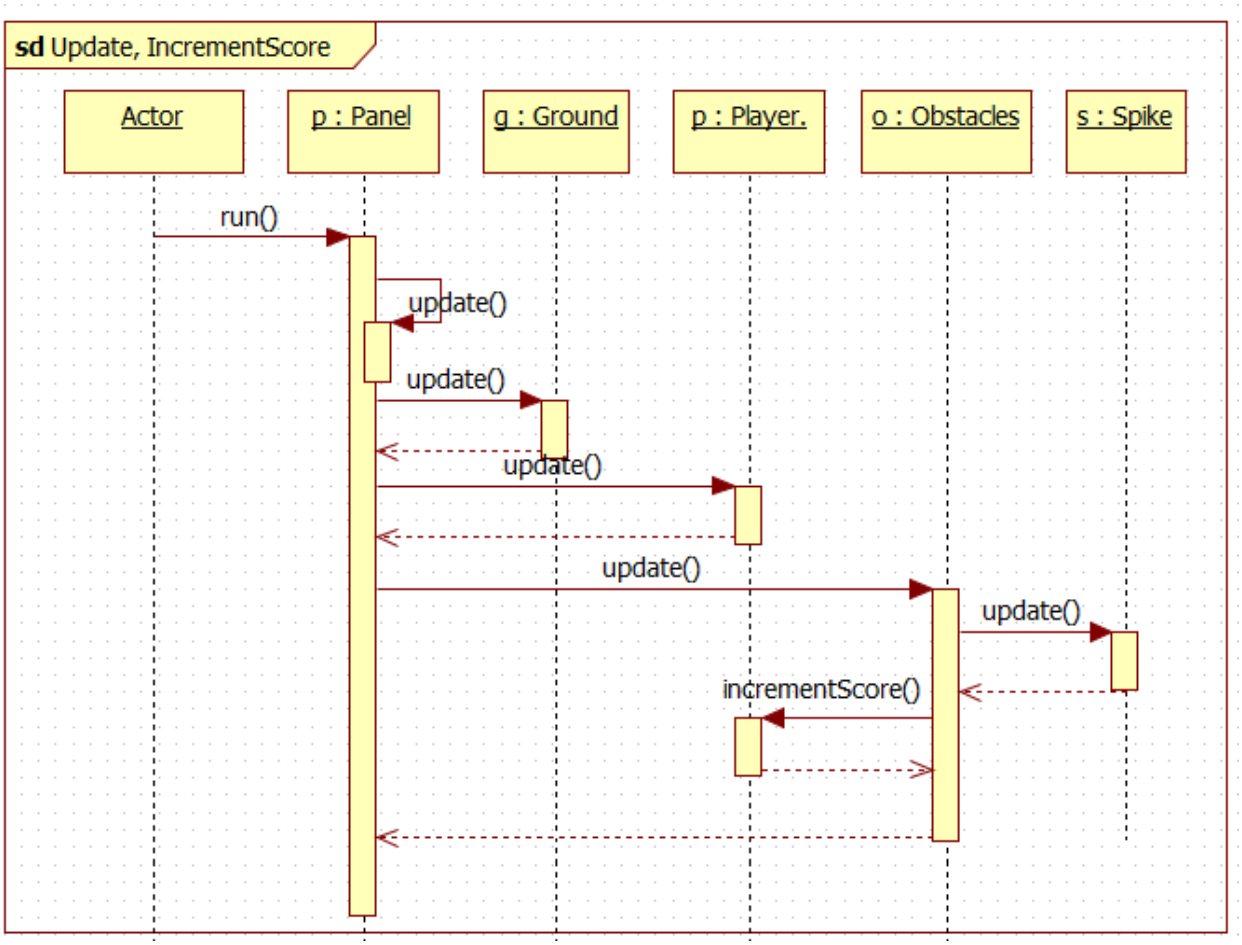
1.2.3 Player can Play Game



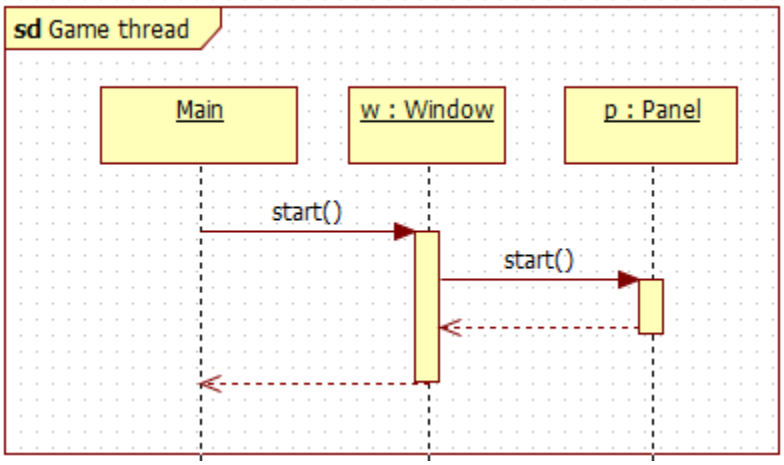
1.2.4 Serialization



1.2.5 Update and Increment score of the player



1.2.6 Game Thread



4.2 Tests Description

1.2.7 PlayerTest and Obstacles test

There are the Unit tests of the Player class, the instance of the class given as p1.

`void compareTo()`	Compare the scores of two players. Sets the score of players in such a way that the p1.score is greater than p2.score. The test is passed when the function returns a positive value.
`void compareTo1()`	Compare the scores of two players. Sets the score of players in such a way that the p1.score is less than p3.score. The test is passed when the function returns a negative value.
`void compareTo2()`	Compare the scores of two players. Sets the score of players in such a way that the p1.score is equal to p4.score. The test is passed when the function returns 0.
`void updateFirstCondition()`	Check the first condition of the update function. Sets the y position of the player. Passed when the velocity is changed to 0 after update function
`void updateSecondCondition()`	Check the second condition of the update function. Sets the y position of the player. Passed when the y is changed to previous position y plus current velocity after update function
`void update()`	Check the update function. Sets the position of the player. Passed if the Rectangle has the same coordinates as the player after update function.
`void getRectangle()`	Passed if the returned rectangle equals to the setted rectangle
`void setRectangle()`	Passed if the setted rectangle equals to the returned rectangle
`void jump()`	Passed if the position of the player after jump corresponds to the several changes in previous position
`void incrementScore()`	Passed if the score greater by 1 unit than the previous score
`void isAlive()`	Sets the Alive to true. Passed if the returned result is also true
`void setAlive()`	Sets the Alive to false. Passed if the returned result is also false
`void getScore()`	Sets the expected value. Passed if the returned value equal to that value
`void setScore()`	Sets the expected value. Passed if the returned value equal to that value
`void getPlayerName()`	Sets the expected name. Passed if the returned value equal to that name.
`void setPlayerName()`	Assign the value to String. Passed if after the set function the get returns that value.
`void updateCollision()`	Sets the player and spikes coordinates(their rectangles) in such a way that there is a collision. After the update function the player sets isAlive as false.

`void updateScoreIncrement()`	Sets the coordinates of the spike so that it is fully out of screen. The score must be incremented by 1 after update function
`void getRandomSpike()`	This function is already called in the constructor. Sets the spikes position X to 1000 where it should be created randomly. Traverse the list of obstacles, checks the position of spikes in that list.