PROI THE GAME

Piotr Satała

Overview

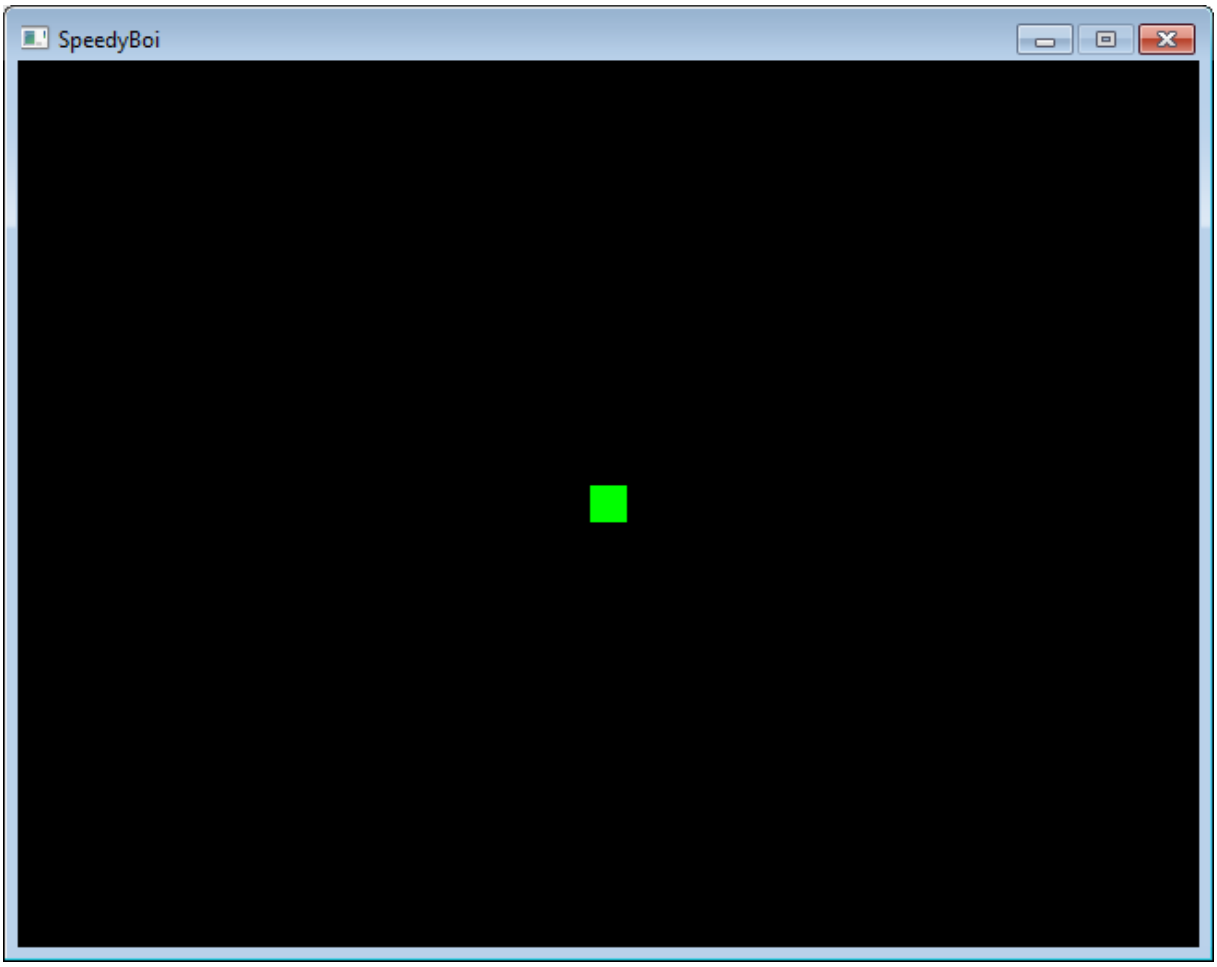
The aim of this project was to create a 2D platform game using SDL2 library. The objective of the game is to complete levels. In order to do that, user should navigate his player (green rectangle) to the finish line (blue rectangle). The game is based on a few simple mechanics:

- Horizontal movement – this allows user to control the on-screen player by moving him left or right through a constant button press
- Jump – this allows user to move the player upwards if the bottom of player's rectangle has contact with another object. The initial velocity given to the player is countered by gravity.
- Teleport – this allows user to instantly move the player by a given distance in one of four directions: up, down, left or right. To use this mechanic again, the user must wait a given amount of time.

Game objects

Player

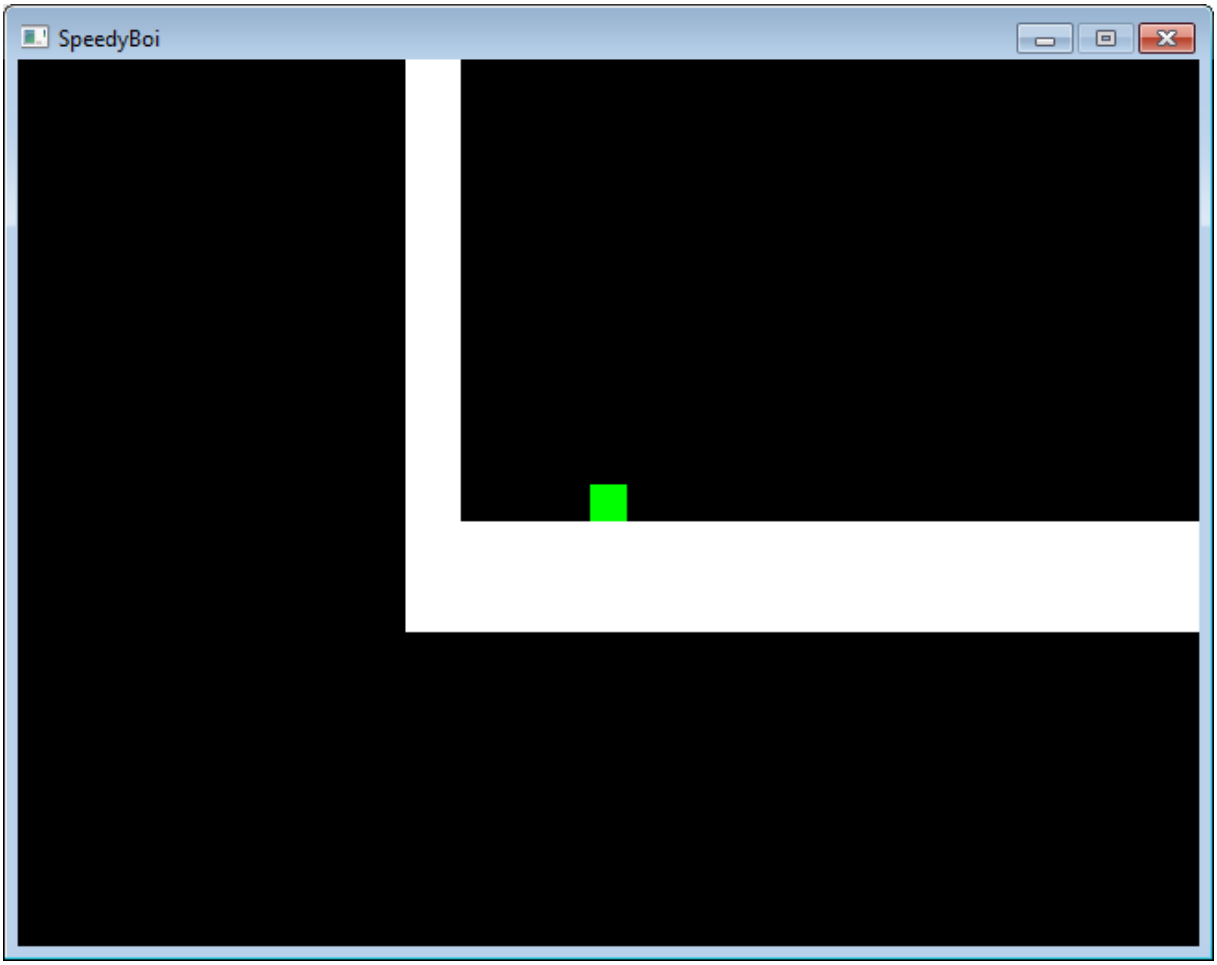
The player controlled by the user is represented by a green rectangle positioned always in the center of the screen. There may only be one player at a time.



Obstacles

Those objects function as platforms. There can be multiple instances of this object. Their color indicates the type of the obstacle.

- White – standard obstacle

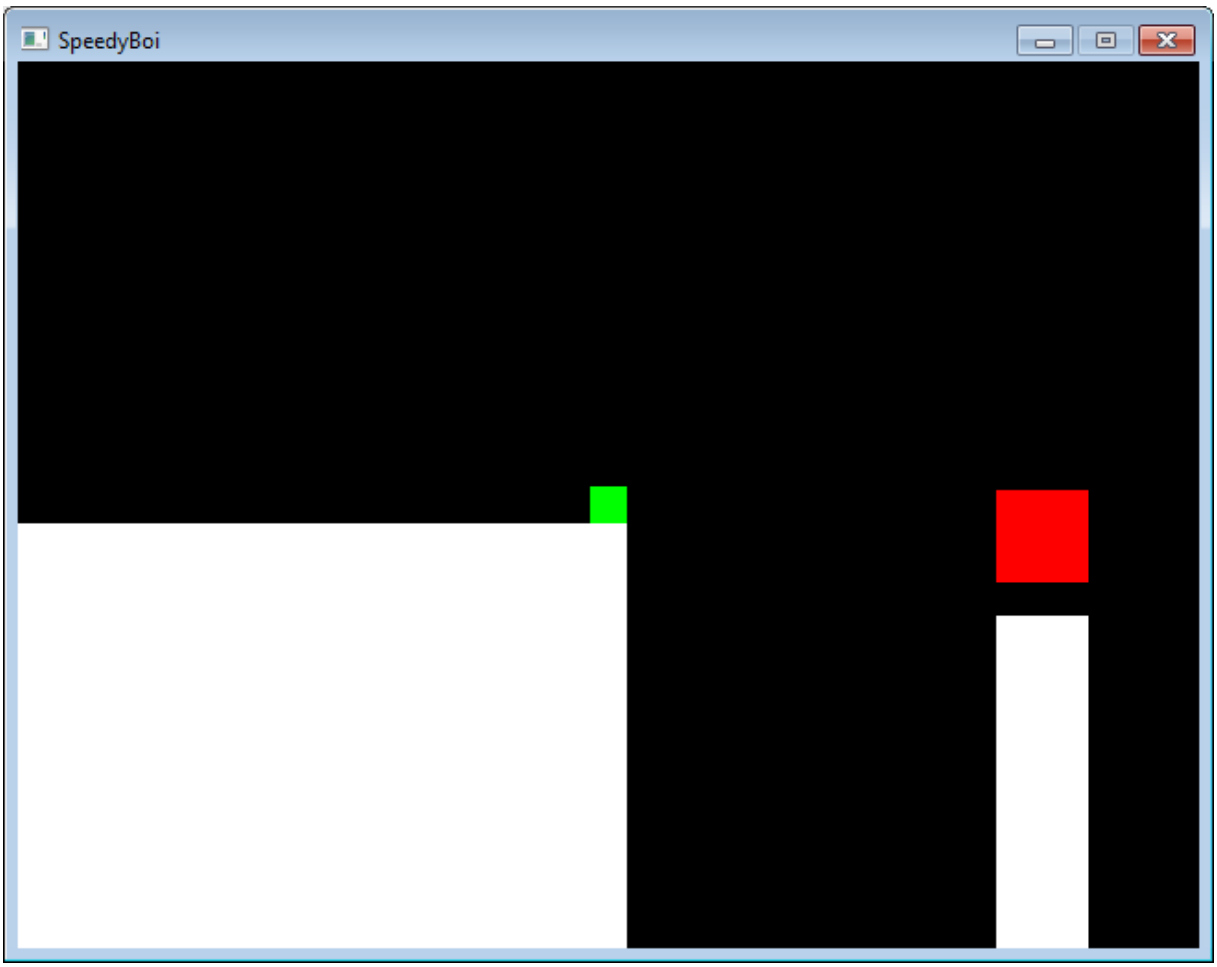


- Orange – this obstacle can cause death if the player makes any contact with it

- Blue – this obstacle is the finish line

Enemies

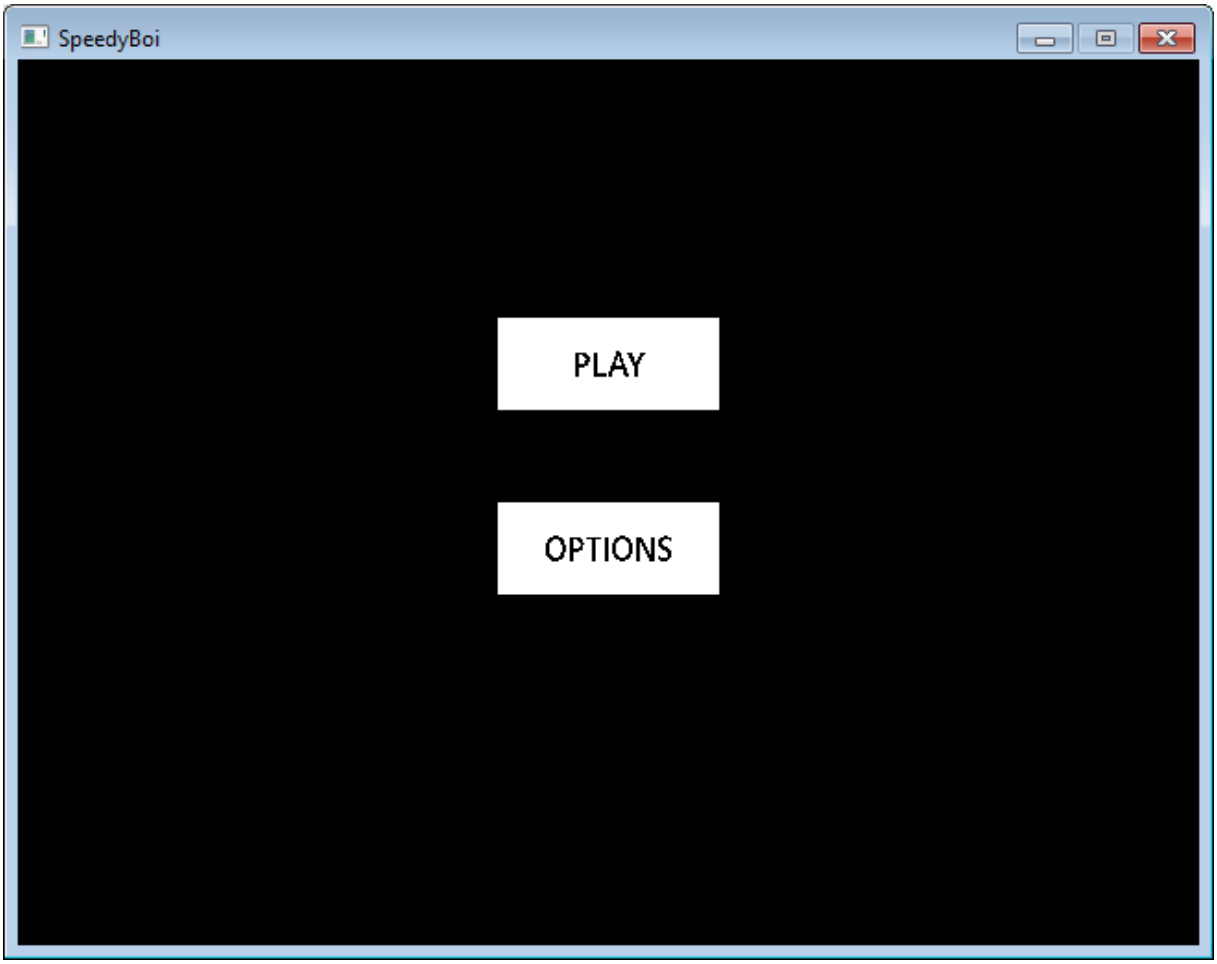
Other players that are not user-controlled. They have a specific behavior function, which determines their movement. Those objects are always represented by red rectangles. If the player's bottom makes contact with the top of the enemy, the enemy dies. Any other contact with the enemy kills the player. There can be multiple instances of this object.



Menu

The game also has a main menu. It can be navigated by using a mouse.

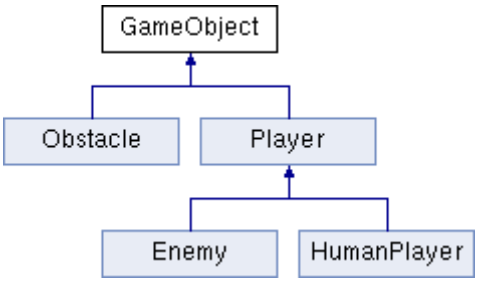
- To select a level to play go to **PLAY**, then choose the appropriate group of levels
- To view controls of the game, go to **OPTIONS->CONTROLS**



Project realization

Class hierarchy

All objects presented on previous pages are part of the main class hierarchy:



Class GameObject is an abstract class. It defines a rectangular object, which does not change its position and cannot be printed, unlike the rest in this hierarchy. Additionally, a set of virtual functions responsible for calculating next position, applying input and setting color for print functions has been implemented.

Class Obstacle, which inherits from GameObject, supports collision detection. It also specifies if it is a “dangerous” obstacle, which can cause death to the player, and whether is it a finish line.

Class Player, which also inherits from GameObject, defines a moving object. Methods responsible for moving horizontally, jumping and teleporting have been implemented. Classes derived from Player focus on converting different inputs into actions defined in Player.

Class Enemy, which is derived from Player, defines a set of behavior functions which specify a set of actions an instance of Enemy will take.

Class HumanPlayer, which base class is Player, defines a method converting user keyboard input into actions implemented in Player.

This hierarchy allows to store all objects in a single vector of GameObjects whilst still being able to call every necessary function (calculating next position, applying input etc.), because those methods have been declared as virtual.

Other classes

Class Camera is responsible for setting the position of Camera (user's player should always be in the middle) and printing other objects in relation to the Camera.

Class GameArea declares an area in which the game should take place. If the player leaves that area, an exception is thrown.

Class Game is responsible for handling the entire application, including simulating menu, loading and playing levels etc.

Class Test contains unit tests, which are called in a separate project, called UnitTestProject.

Class MenuObject defines a part of menu structure, which is a rectangle with a text to print on it. Moreover, each instance has a return value, which is an index of a function that will be called once the user clicks this object.

Templates

Momentum<T1, T2, T3> specifies momentum of a moving object (g-force, velocity in x, velocity in y). For class Player, this template is defined for T1=T2=T3=double.

Tree<T> simulates a tree container consisting of TreeElement<T> elements. For purpose of this project, the menu structure has been defined with T=MenuObject.

TreeElement<T> simulates a single element of the Tree<T> container. Its content is: pointer to a father element, vector of child elements and a pointer to a stored element.

Level creation

Syntax

Levels are stored in .txt files, which contain the following syntax:

- Single letter (P, G, O, E) – P declares a HumanPlayer, G - GameArea, O – Obstacle, E – Enemy. There must be exactly one instance of HumanPlayer and GameArea in the file
- X coordinate, Y coordinate, height and width of the object
- If O was chosen: 2 Booleans specifying whether the obstacle “can kill” and whether is it a finish line
- If P was chosen: constant for velocity in x, constant for velocity in y, g-force for the object, distance covered by a single teleport, time between teleports in milliseconds
- If E was chosen: same as for P plus a string indicating enemy behavior (BOUNCE for object that moves left and right, JUMP for enemy that constantly jumps, TELEPORT_direction where direction = {UP, DOWN, LEFT, RIGHT} for enemy constantly teleporting in the given direction, anything else for a stationary objects)

Easy levels

Those levels serve as an introduction to the game. The goal is to familiarize the user with game mechanics. Of course, knowledge of the controls is recommended.

Dev levels

Those levels were used for testing during the development of the project. They have remained in this release to provide more information about the development process and to show how the program reacts to exceptional situations.