Gliwice, 24.01.2019

Semester: 3
Group: 3

Computer Programming Laboratory

Arkanoid

Author: Marcin Markefka
Email: marcmar879@student.polsl.pl
Tutor:Anna Gorawska

1.Task topic:

The topic of the task was to create and implement code for game “Arkanoid” using at least 4 methods presented on the lectures and containing at least 5 classes.

2.Project analysis:

The project uses the SFML library which provides the possibility of the use of graphics, the control of keyboard, mouse and even audio files. The library enables to create 2D games like the “Arkanoid”. SFML is not included to Visual Studio so before implementation of the program I had to download, install it and enclose to project.
The program uses such possibilities of SFML library: the control of keyboard and mouse, printing graphics, creating of simple geometrical shapes, animation of move of different objects, registering collision between 2 objects, printing the text on the screen.
The program creates separate window which can be closed and resized by the user. The window consists of predetermined number of units of height and width which create an x-y coordinate system. A lot of SFML functions use vectors to determine the positions of objects on the x-y coordinates system. The origin of the system is in the upper-left corner of the window.

The program uses basic data structures like arrays to order some data and not complex algorithms for example to create net of blocks on the playing field.
From 7 methods presented on lectures I used 4 of them: polymorphism, inheritance, templates and I/O streams.

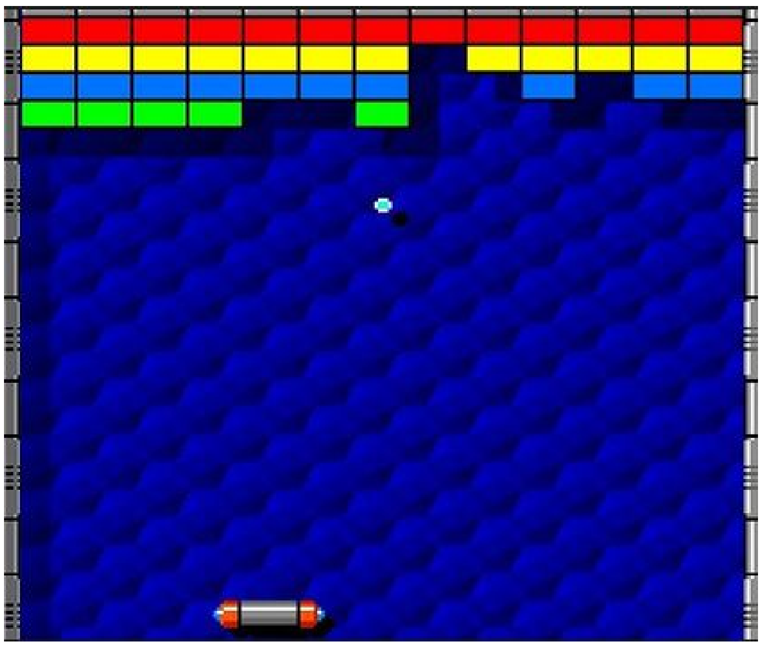
I/O streams are used to loads short text from a file and display it on the screen after choosing one from the options at the very beginning of the program.
Polymorphism is used in functions of 2 different classes responsible for move of the objects.
Inheritance is used to enable the access to one of the function from base class the another function from derived class.
Templates were used in 2 different classes to set the maximal number of elements displayed on the screen.

3.External specification:

The rules and goals of Arkanoid:

- the player controls a small rectangle called "paddle" which prevents a ball from falling from the playing field.
- the field is open only from one side so after exceeding permissible field there is game over.
- the ball can bounce against 3 others sides of field and against blocks.
- after collision with a block, the block disappears.
- the goal of the game is to smash all the blocks on the playing field.

Example of the playing field:



Source: http://www.edziecko.pl/edziecko/1,166810,9775670,Klasyczny_Arkanoid.html

After launching the program, it should be displayed the application’s window and console window. In each stage of the program, there’s should be a possibility to close the application’s window by pressing the button “X” in the right-upper corner of window and to resize the window.
At the very beginning there should be displayed the main menu of the game. There are 3 possible choices: starting the game (field “PLAY”), looking up the instruction (“INSTRUCTION”) and exit the program(“EXIT”).

After pressing the button “PLAY”, the game starts. To control the paddle, press the key A or D on keyboard. If you press A, the paddle will move in the left direction. If you press D, the paddle will move in the right direction.

If the ball exceeds the permissible playing field, there will be displayed on the screen message “YOU LOST” and the ball will stop moving. To retry playing the game, it is needed to start over whole program.
After the first collision of the ball with a block, there will be screened the counter, which shows the current number of remaining blocks.

The second choice in main menu is “INSTRUCTION”. After pressing the button, there will be displayed the text with manual for Arkanoid. To come back to main menu, the player should press the key Z on the keyboard.

To exit the program in main menu, press the button “EXIT”.

4.Internal specification:

Functions and commands from SFML library(classes):

sf::RenderWindow:

- sf::RenderWindow window(sf::VideoMode(800,600), "Arkanoid"): the most important command for SFML library, constructs a new window, sets the size and name of the window;
- window.setFramerateLimit(60): limits the framerate to a maximum fixed frequency, sets the frequency (rate) at which consecutive images (called frames) appear on a display;
- while (window.isOpen()){}: tells whether or not the window is open;
- while (window.pollEvent(event)){}:pops the event on top of the event queue, if any, and returns it;
- window.clear():clears the previous content of the target, clears the screen with given frequency by function setFrameRate, contributes to move of the paddle and the ball, removing their previous move’s states;
- window.draw(sprite/shape/texture/text): draws different types of objects, has to be set between window.clear() and window.display();
- window.display():enables to display on the screen new content of the target, displays objects on the screen with given frequency by function setFrameRate, contributes to move of the paddle and the ball. screening their new move’s states ;
- window.close():closes the application’s window;

sf::Texture/sf::Sprite/sf::RectangleShape:

- texture.loadFromFile(""): loads the texture from a file;
- sprite.setTexture(texture): gets the source texture of the sprite;

- sprite.setScale(sf::Vector2f(1,0.9)):sets the size of the object;
- if(sprite1.getGlobalBounds().intersects(sprite2.getGlobalBounds())): checks whether the bounds of sprite1 intersects with bounds of sprite 2, detects the collision between 2 objects;
- shape.setSize(sf::Vector2f(x, y)): sets the size of the objects, vertical size-y, horizontal size - x;
- sprite.move(x,y):moves the sprite by given parameters x and y;
- sprite.setPosition(sf::Vector2f(x,y)): sets position of object in a such way that the upper-left corner of object has position x,y;
- sprite.getPosition().x/y:gets the current position of the object;
- shape.setFillColor(sf::Color::color): fills the shape with given colour
- sprite.getSize().x/y : gets the current size of the object

sf::View:

- sf::View view(sf::FloatRect(0, 0,x,y)): defines the zone for background to display

sf::Text/sf::Font

- text.setString(string): sets the text's string;
- text.setFont(font): sets the font of the text, fonts are loaded from computer by the function loadFromFile();
- text.setCharacterSize(number): sets the size of the text;
- text.setPosition(sf::Vector2f(x,y)): sets the position of the text;
- text.setColor(sf::Color::color): sets the colour of the text;

sf::Event

- case sf::Event::Closed: checks the case whether the event of closing the window is made;
- case sf::Event::Resized: checks the case whether the event of resizing the window is made;

sf::Keyboard//sf::Mouse

- if (sf::Keyboard::isKeyPressed(sf::Keyboard::key)): checks whether is pressed the specific key on keyboard
- event.mouseButton.x: returns current position of the cursor

Designed function:

class Ball:public Movement, public Menu

- void Ball::creation(sf::Texture & texture): loads the texture “ball.png” ;
- void Ball::parameter_of_ball(sf::Sprite & sprite): sets the parameters(scale and initial position) of the sprite-ball;
- int Ball::movement_objects(sf::Sprite & ball, sf::RectangleShape & rectangle): the function is used to implement the polymorphism, sets the velocity of the ball, considers cases when ball bounces

from the bounds of playing fields and decides in these cases how ball should behave, if ball exceeds the permissible playing field , the function inherits another function last_menu(string) from base class Menu and sets the string-message for the player:”YOU LOST”, in that case returns 1;

class Blocks(uses templates)

- void Blocks<N>::creating_of_blocks(sf::Texture &texture2): uses the templates for setting maximal number of blocks which can be displayed, sets the size, position and texture of blocks, sets how they should be displayed on the screen(spaces between blocks, maximal y position);

class Counter

- void Counter::counting(int i): loads the specific font from file .ttf, sets the string, position, size, colour and font of the counting number of remaining blocks on the screen;

class Lines(uses templates)

- Lines<N>::Lines(): the constructor sets the lines on the blocks which are used during detection of collisions between ball and blocks, created lines are invisible for player, lines have the same parameters and are created in the same way like blocks;

class Menu(uses I/O streams)

- void Menu::field_creation(): creates fields in main menu on which are set text, sets the size, position and colour of this fields;
- void Menu::string_creation(): creates text(“PLAY”, “INSTRUCTION”, “EXIT”) which are screened in main menu, sets the strings, positions, fonts, colours and sizes of captions;
- int Menu::location(sf::Event & event): detects whether the left mouse button is clicked, if yes checks what is the position of the cursor, if is over the first field(“PLAY”) returns 1, over the second field (“INSTRUCTION”) returns 2, over the third field(“EXIT) returns 3, if isn’t over none of the fields, returns 0;
- void Menu::instruction(sf::RenderWindow & window, sf::Event & event): loads text from file to instruction (I/O streams), prepare loaded text to be displayed on the screen, sets its size, colour, font, position;
- void Menu::last_menu(string s): prepares the text to be displayed in 2 cases: when the ball exceeds permissible playing field or when all blocks will be smashed, sets position, size, colour, font and string of the text;

class Movement(abstract class):

- virtual int movement_objects(sf::Sprite &sprite, sf::RectangleShape &rectangle)=0 : pure virtual function, implemented to use polymorphism;

class Paddle:public Movement

- void Paddle::createpaddle(sf::RectangleShape &paddle): sets the parameters of shape-paddle(size, position and colour)
- int Paddle::movement_objects(sf::Sprite &sprite, sf::RectangleShape &paddle): the function uses polymorphism, checks if specific keys(A,D) on keyboard are pressed, if yes the position of paddle is changed;

beyond any class:

- double random(double upper, double lower): generates random number from interval <lower,upper>, generated number is a multiplier of parameters X and Y responsible for move of the ball, the goal is to change a little trajectory after collision with blocks;
- void closing_resizing(sf::RenderWindow &window, sf::Event &event): closes the window after pressing the icon “X” and enables to resize the window;
- void Uni(Movement *x): calls the function movement_objects(sprite, paddle) depending on which class points out the argument x, used in polymorphism;

Other functions:

- exit(0) - terminates the work of program;
- file.open(“file.txt”, ios::in) - opens the file ;
- if(file.isopen()) - checks if the file is open;
- file.eof() - checks if there is end of file;
- getline(file, string) - loads lines of text from file;

Definitions #define:

#define MAX_SCREEN_X - defines size of screen for horizontal axis;
#define MAX_SCREEN_Y - defines size of screen for vertical axis;
#define X_VALUE_RECT - defines size of single block in horizontal;
#define Y_VALUE_RECT - defines size of single block in vertical;
#define FIELD_X - defines size of single field from main menu in horizontal;
#define FIELD_Y - defines size of single field from main menu in vertical;
#define POSITION_MENUS_BLOCKS_X -defines the horizontal position of first field in main menu ;
#define POSITION_MENUS_BLOCKS_Y -defines the horizontal position of first field in main menu ;

Global variables:

sf::Sprite sprite2 - stores the information about ball;

sf::RectangleShape paddle- stores the information about paddle;

Description of some solutions:

1) **creating the net of lines**

```
int i = 0;

  int a = 1;
  int b = 1;
  while (b < (MAX_SCREEN_Y - 360))
  {
        while (a < (MAX_SCREEN_X - 50)) {
                line_top[i].setSize(sf::Vector2f(X_VALUE_RECT, 1));
                line_bottom[i].setSize(sf::Vector2f(X_VALUE_RECT, 1));
                line_right[i].setSize(sf::Vector2f(1, Y_VALUE_RECT));
                line_left[i].setSize(sf::Vector2f(1, Y_VALUE_RECT));

                line_top[i].setPosition(sf::Vector2f(15 + a, 100 + b));
                line_bottom[i].setPosition(sf::Vector2f(15 + a, 100 + b + Y_VALUE_RECT));
                line_left[i].setPosition(sf::Vector2f(15 + a, 100 + b));
                line_right[i].setPosition(sf::Vector2f(15 + a + X_VALUE_RECT, 100 + b));


                a += X_VALUE_RECT + 2; // 2 determines horizontal space between blocks

                i++;
        }
        a = 1;
        b += Y_VALUE_RECT + 2; // 2 determines vertical space between blocks
  }
}
```

The goal of creating lines with the same positions like blocks invisible for the player was to simplify the moment of collision of the ball with blocks. Each line represents top, bottom, left and right bound of the block. Without that, the program doesn't know which component of the vector should change the sign. The lines are created in such way: at the beginning b is smaller than MAX_SCREEN_Y(=600)-360 (creates space between paddle and blocks) so compiler goes into the first while loop. A is also smaller than MAX-SCREEN_X(800)-50 so compiler again goes into second while loop. Then there are setted sizes and positions of lines of the first block, the value a is increased by the horizontal length of a block plus determined space between blocks and value i is increased by 1 to go into another quartet of lines.The situation repeats until a is smaller than before-mentioned value. If a is bigger, then compiler gets out from the second loop, sets 'a' as 1(to begin the second row from the beginning) and increases value of b by the height of blocks plus space between 2 blocks. The situation repeats until the value b will be greater than before-mentioned value.

2) **collision of the ball with blocks**

```
if (sprite2.getGlobalBounds().intersects(lines.line_top[i].getGlobalBounds()) ||
sprite2.getGlobalBounds().intersects(lines.line_bottom[i].getGlobalBounds()))
   {
  ball.Y = -ball.Y*random(1.01,0.99);
  blocks.block[i].setSize(sf::Vector2f(0, 0));
  lines.line_bottom[i].setSize(sf::Vector2f(0, 0));
   lines.line_top[i].setSize(sf::Vector2f(0, 0));
  lines.line_right[i].setSize(sf::Vector2f(0, 0));
   lines.line_left[i].setSize(sf::Vector2f(0, 0));
   blocks.i--;
  counter.counting(blocks.i);
}
```

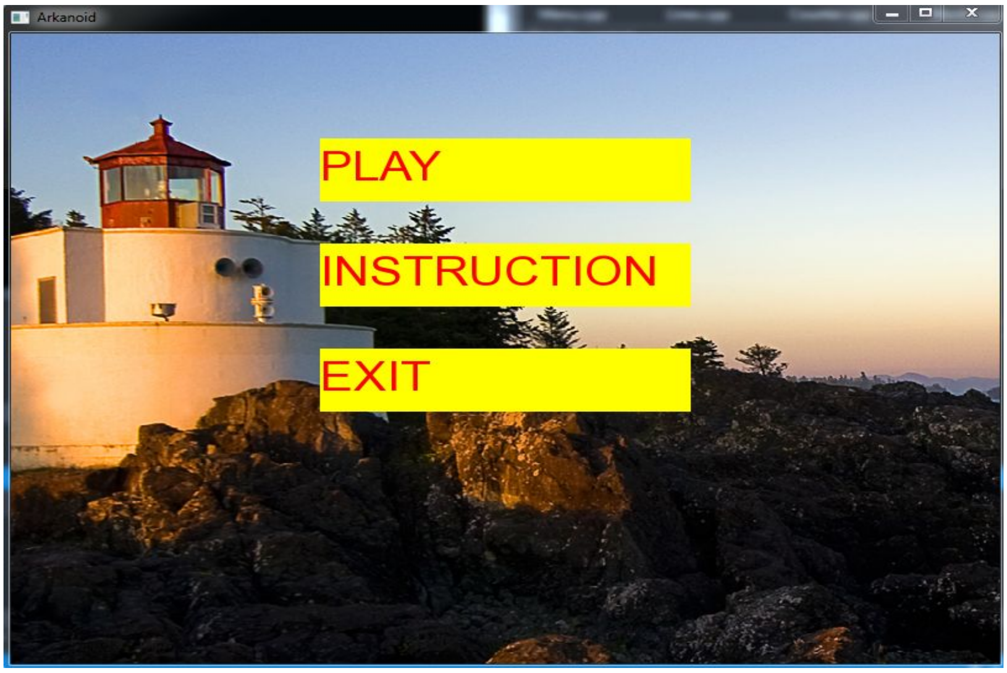
During the collision the program checks with which line collides the ball. If the ball collides with top or bottom bound of a block (like in the exemplary part of the code), the vector [X,Y] changes into [X,-Y]. If the ball collides with right or left bound, the vector [X,Y] changes into [-x,y]. After the collision ,the size of the all lines and block are set as [0,0] - block disappears.

5. Source code

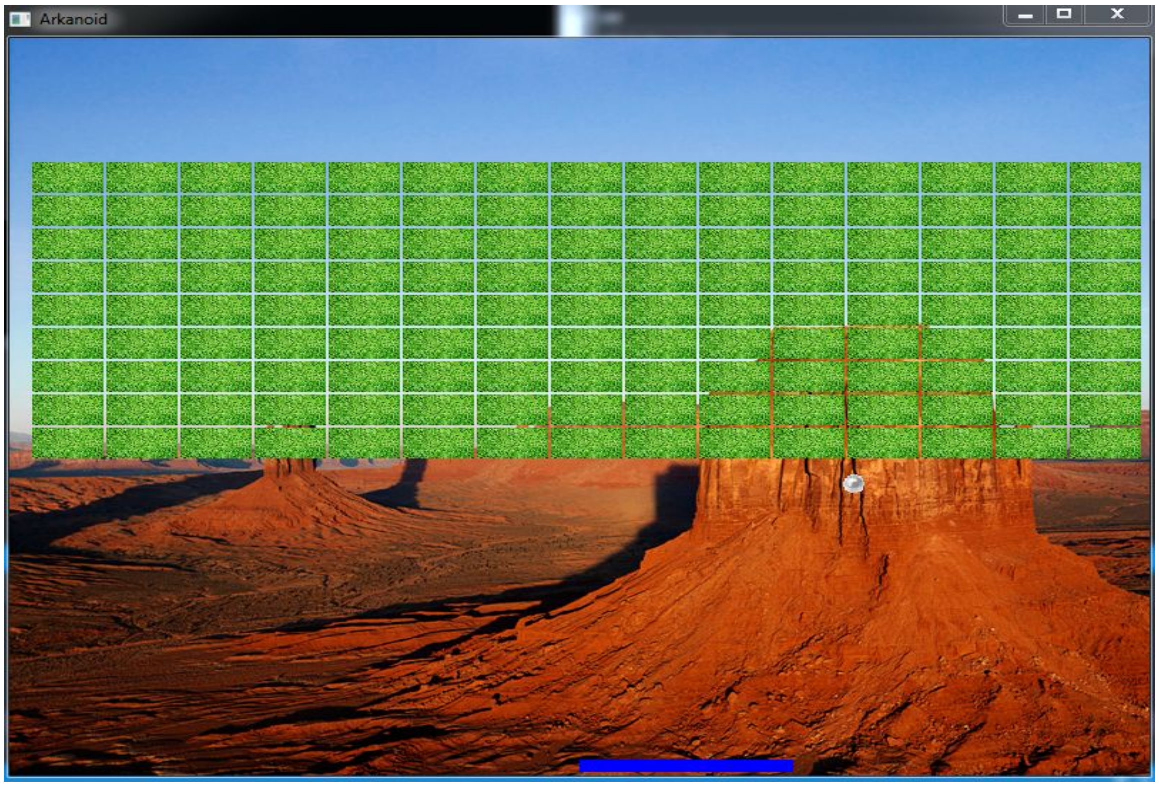
https://github.com/marcin2019

6. Tests

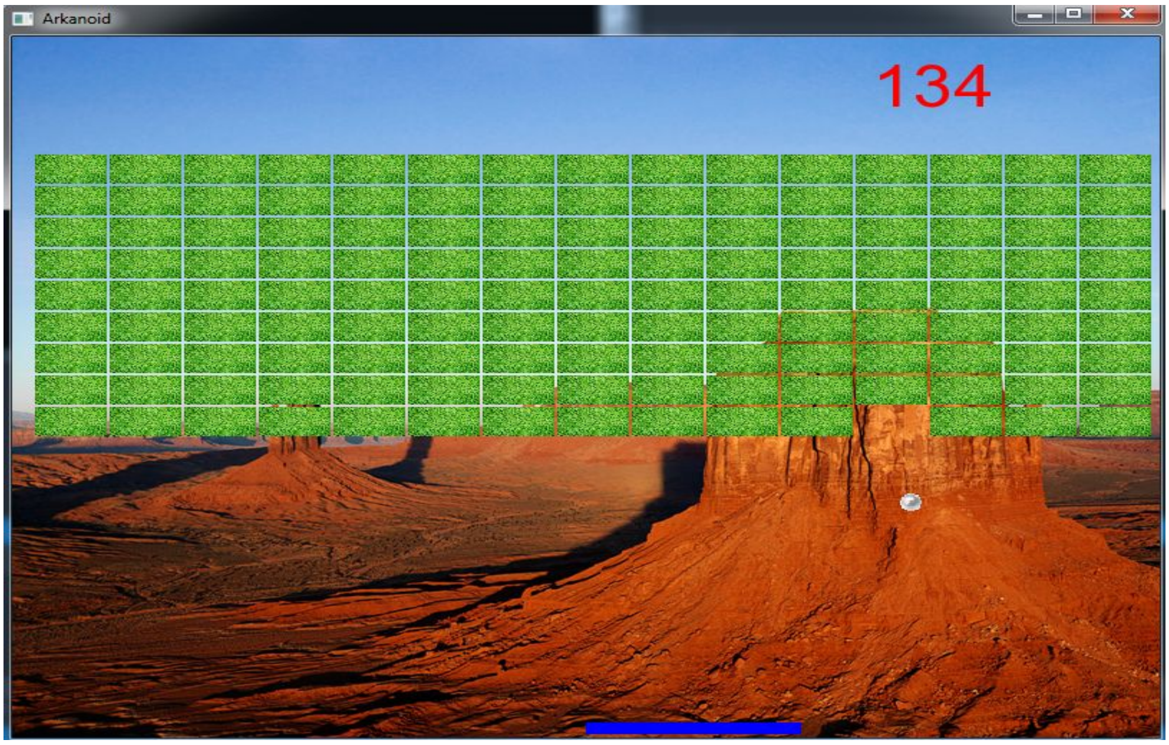
At the very beginning there is screened the main menu of the game:

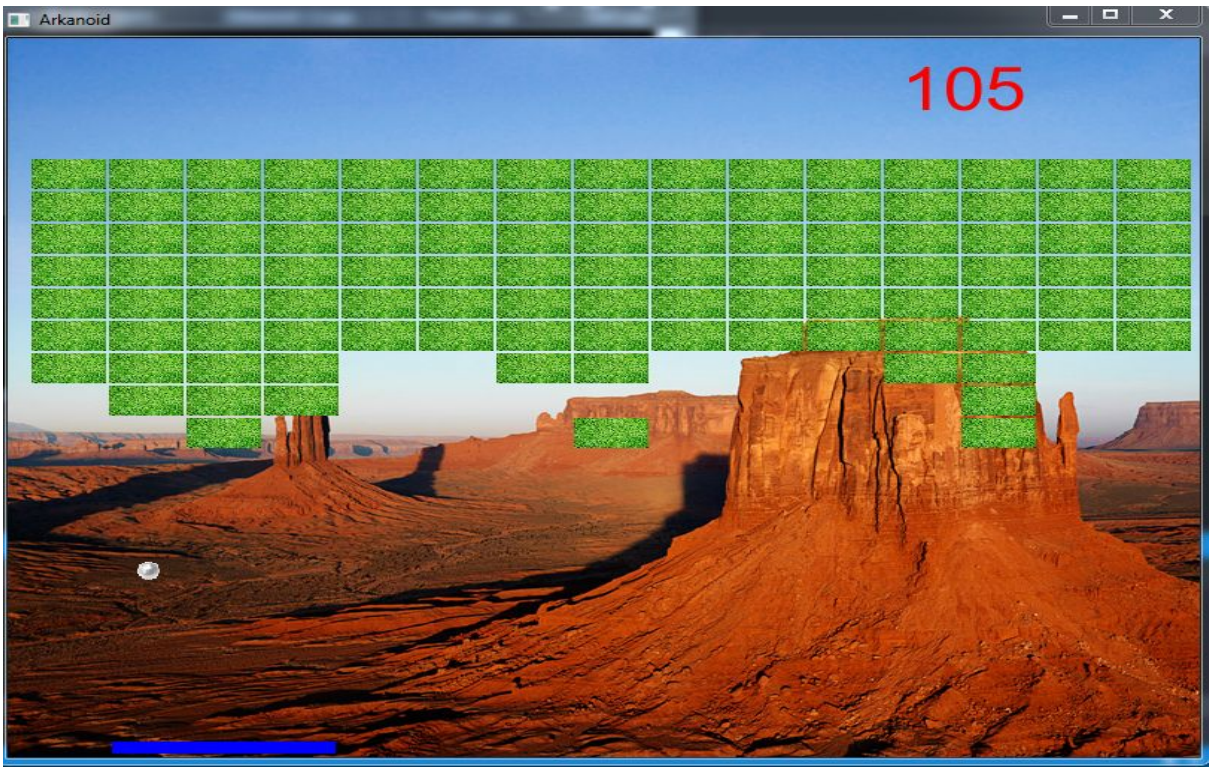


Now we have 3 possibilities. If we press on the first button - “PLAY”, the game will start.

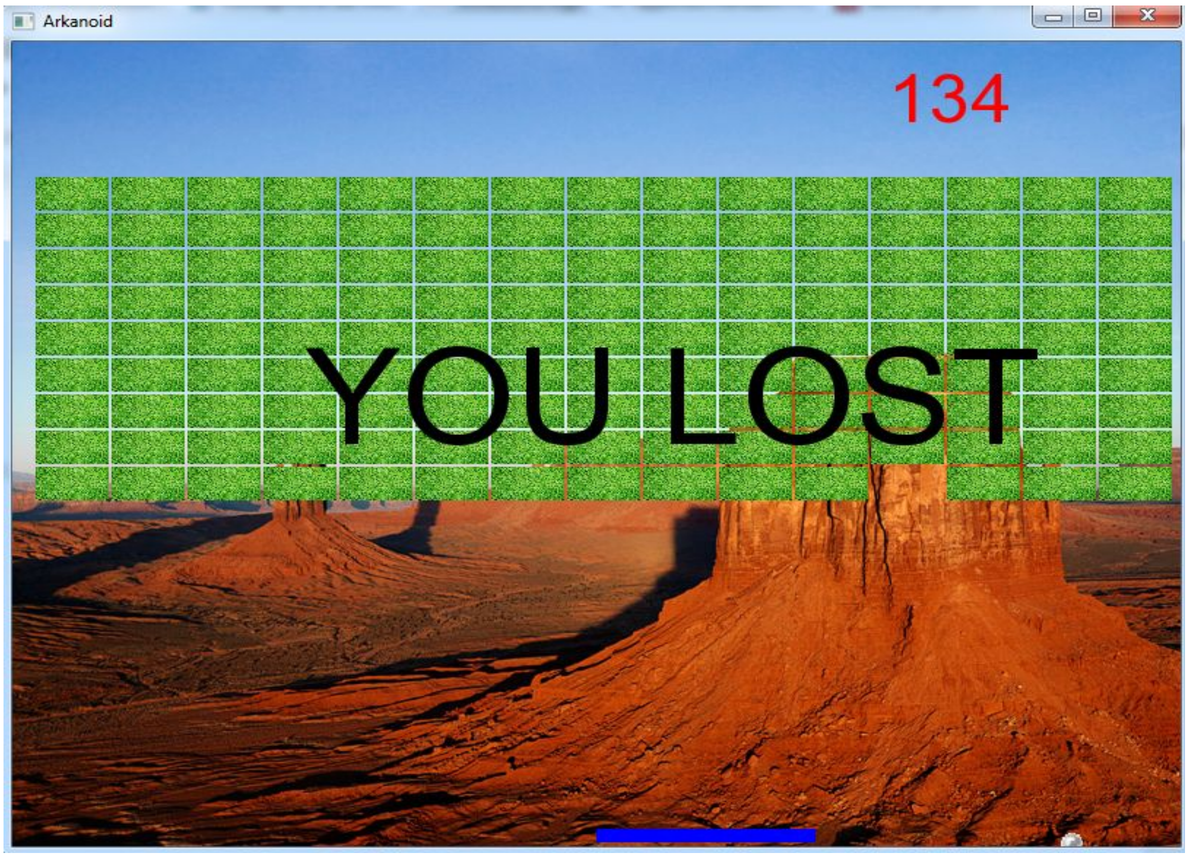


After the collision with first block, the counter appears in the right upper corner of window

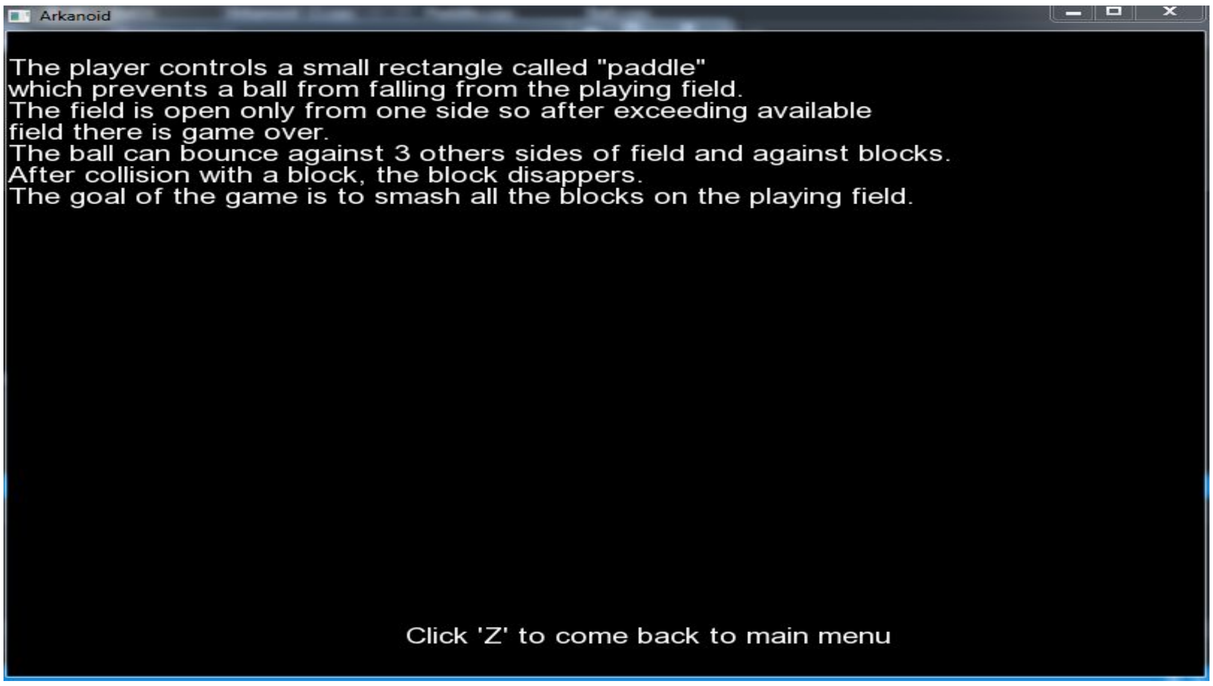




If the ball exceeds the permissible playing field, there is displayed the message: ”YOU LOST” and the ball stops moving



The second possibility in the main menu is “INSTRUCTION”. After pressing on the proper button, there is screened the instruction of the game and the information how to come back to main menu.



```
The player controls a small rectangle called "paddle"
which prevents a ball from falling from the playing field.
The field is open only from one side so after exceeding available
field there is game over.
The ball can bounce against 3 others sides of field and against blocks.
After collision with a block, the block disappers.
The goal of the game is to smash all the blocks on the playing field.

Click 'Z' to come back to main menu
```

The third possibility is button “EXIT”. After pressing on the button, the program closed. It is also possible to exit from the game by pressing the icon “X” in the right upper corner of window.

7. Conclusions

- All implemented parts of the program like pressing the buttons in menu, control of paddle, move of ball work properly.
- Loading all pictures, text from file “Arkanoid.txt” and fonts work
- Some drawbacks:
    - the lack of complex data structure;
    - some of the parameters are fixed and adjusted for example to size of the window, amount of lines in text file;
    - insufficient clearness of the code;
    - some solutions are memory-consuming for example lines overlay on the blocks;
    - there are some blunders connected with collisions: after collision in the same time with 2 different blocks, the vector of the ball don’t change his direction;
    - there is no possible to pause the game, come back directly to main menu from playing field, save the player’s attempts;

Sources:

- wikipedia
- https://sonarlearning.co.uk/coursepage.php?topic=game&course=sfml
- https://www.sfml-dev.org/documentation/2.5.1