Programming with C/C++
Academic year: 2022-2023, Fall semester,
Programming assignment

The AI BipBop

[Report for project]

Katarzyna Klęczek,
u118042
Anr 2059098

Overview

Overall I think I managed to succeed in most of the tasks and what is important for me a lot of the solutions of mine not exactly as the solutions from the lectures and they work. I think that the project took a lot more of my time than I expected it to take. I started right away but since the fltk was only briefly introduced and also I couldn't use the files provided by the book I was struggling at first even with the simplests tasks. Later on I found some useful websites in the internet and lectures that helped with moving forward. First I started with a code that is not based on OOP but slowly once I have more object incorporated I moved into OOP for the clarity and better structure of the code and project. At the end I had a main code that is running a window from fltk and almost everything else being attached or happening in the window file. Things that are attached to the window also has different header files and most of the functions that are about them are inside those files. The rest is in the window file.

Tasks

1) Displaying window, board, ball and bricks

Window

The window is the most basic, biggest, and first element of the game that is created in the game. It is the separate part of the screen that is going to be used to represent the whole world, or environment, of the game. In this project, the window dimensions were specified to be 480x640 (height x width). The background color was set to black in this implementation and it was based mostly on personal preference as well as because in some cases perceived attractiveness of a product displayed on a black background is significantly higher than on other colored backgrounds (Howell & Schifferstein, 2019), additionally in the gamer industry black background may resemble a dark mode which is known to be healthier than its light equivalent and not cause fatigue, eye soreness or screen flickering (Benedetto et al., 2014 , Kim et al., 2019) and have similar properties in terms of energy usage of the screen (Löffler et al., 2017). The window is also used to display other elements that are later attached to it.

The window is a part of the FLTK library and is a fl_window, and it is also put in a separate header file that also has functions such as:

- init_window() - to display the window and objects that are attached to it, such as the lives, ball, and plank
- plank_bounce() - function to make the ball bounce from the plank depending on different positions on where the ball falls in the plank or outside the plank
- check_lifes() - a function that checks if the ball has fallen below the plank level and subtracts the life from the player (number of lives is declared as a private property of the class window and is set to 3 at the beginning of the game) and thus displays fewer lives at the top of the screen.

- isRunning() which is moving the ball as well as calling the functions plank_bounce and check_lifes when the window is running and is called inside the timeout function.
- Timer and animate are both callback functions. The first calls the timeout function and displays the lives, and checks the boundaries, and the second increments the timer.
- bounce_bricks() which handles the brick-ball collision, part of it is in the Appendixes
- smart_movement() function that is preventing the ball from falling on the ground. An attempt to make the game smart.

The result is that the window almost becomes a new object, which is a complex program all on its own, from a simple fl_window.

Board

The second element of the game is the plank, also called the board or paddle. The dimensions of this element were specified upfront to be 20x120 (height x width). The choice of color, just like with the window, was left to the student and because of the black background, I decided to make the plank white so that the contrast would help the player find it and indicate its importance, thus grabbing their attention (Morrone et al., 2002). The plank is based on the existing FLTK component (Fl_Box), 100 for width and 20 for height. It is incorporated into the game in a separate header file and later made a part of the object window. The board is a private part of the window and is initialized at the position (maxymalnex-100)/2,maxymalney-20), where maxymalnex and maxymalney are defined global variables for the width and height of the window. That makes the board appear on the bottom of it and in the center in terms of width.

Ball

The next element is a ball. As with all previous elements, this one also has a predefined size which here is r=10, where *r* refers to the radius of the ball. Red was proven to have a very high impact on people's attention and is also associated with danger, and since the ball falling on the ground is the only way to lose the game, it seems reasonable to use that color for the ball (Kuniecki et al., 2015). The ball is based on the existing object from the FLTK library (Fl_Box)Similar to previous elements, this one is created in a header where movement functions are defined (similar to plank) and later attached to the window as its own property in a spot above the board.

Bricks

The last element that had any specifications predefined was "the bricks". A single brick’s dimensions are 40x40, and it has to have 10 levels of integrity. My implementation of it will have an impact on the color of a brick so this will be explained in the later part. The

bricks are allowed to appear only in the upper part of the screen and have padding around them that is set to 5 units. The main goal of the game is to destroy those bricks. In terms of programming, the bricks are objects that are built on a class Block that is based on the class Fl_Box which is a part of the FLTK library. There are some newly implemented or overwritten functions such as draw_all(), which create a set of bricks in a vector and display them only in the upper part of the screen. The function can be seen in appendix 3.

2) Movement of the ball

Moving

Movement of the ball is implemented in the window's function animate and ball function move which makes the ball move every 0.05s based on the Speedx of the ball and Speedy that are adding or subtracting values from the current coordinates in the move function in order to create new coordinates and redraw the ball in the new position.

Bouncing from screen boundaries

The ball bounces from the screen boarders as shown in the code in the appendix 2.

3) Movement of the board

Plank movement is limited to the horizontal axis and is set to 15 units per key press, and it also cannot move outside the size of the display window. The movement is enabled through the event handler function and a move() function that is hiding a widget, changes its coordinates, and redraws it in the adjusted position. The function had to be added as it is neither a basic function of fl_widget nor fl_box. Once either the left arrow or "A" is pressed the function will redraw the plank 15 units to the left. If the right arrow or "D" is pressed, the same will happen but in the opposite direction.

4) Gameplay

Implement a timer

Another element of the game is a timer, the timer just shows the time that has left from the start of the game (in some cases that may also be an indicator of how skilled someone is if the goal is to finish the game the fastest). And the score is defined by how many times the ball hit a brick. It increases by one per hit. The timer goes up every second thanks to a callback function, and the score goes up every time the brick is hit.

Function to reduce the integrity of a brick when it is hit by a ball

All the bricks have integrity 10, as specified in the project description. In my implementation, the integrity is called hardness and is visible in the color of the brick. Once the brick is hit, the hardness is lowered and the color of the bricks changes. In appendix 5 there is an example graphic of how bricks with different hardness in different rows look like. Once the brick's hardness drops to 0, the function hit would check the

hardness and move the brick outside the window so that it is not interfering with the ball movement. The function “hit” is shown in the appendix 6.

Lost a life when ball is below the board level
The window.h function check_lifes() subtract the lives of the player once the ball is on the height of the board but the board is not there. “Life” is an element of the game that indicates how many lives the player has - how many times the ball can fall on the ground before the game is over. It is also implemented as a separate header file and subsequently attached to the window. It is based on the Fl_Box to which an image is attached. Once the life is lost, which is checked in the window function that checks the ball when it is about to hit the ground, one of the boxes is deleted. It is in appendix 8.

Ball bounces from bricks
The ball bounces from the bricks due to the function bricks_bounce which when the ball is in the upper part of the screen, checks the position of the ball and every brick in the brick vector and if the ball is either touching a brick or is inside it. The ball bounces in the opposite direction that it was moving and the brick gets hit. That activates the hit function from the block.h file, changing the integrity of the brick and thus, also its color. The last part of the code can be seen in appendix 9.

Ball bounces from plank
The ball is not controlled by the player (nor the algorithm), but rather has predetermined movements and rules for movement. The rules apply to which part of the board ball falls into, depending on that it bounces back under different angles, the speed on the y-axis is constant and was semi-randomly determined by me to be 5 units per second. The rules of bouncing are calculated by the code in appendix 1 and incorporated into the window function plank_bounce().

5) Make the game smart
My idea for making the game smart was to make an algorithm that would be able to move left or right depending on which yields a higher score. Depending on the position of the ball and the position of the bricks in the vector brick it should try to maximize the score which increases every time you hit the brick. For now, I managed to make the plank follow the ball and thus not let it fall on the ground so the game is smart enough to not lose, but not smart enough to win. It takes into account only the ball and plank position, not looking at the bricks or score at the moment. It is in appendix 7.

Challenges

1. Creating the ball was the first challenge for me as I didn't really know how to create a circular object. It turned out to be quite easy as the type of the Fl_box can be set to one that resembles a circle.
2. The movement of the ball was a second huge challenge. In contrast to the plank movement, the ball has to move on its own. And it was really hard for me to come up with a solution or a function that would be called repetitively over time. And where to put that function was another challenge.
3. Another challenge was calculating angles. Even after finishing the project, I am still not sure if the angles are calculated correctly. In my opinion, it was a convoluted task. It required some mathematical knowledge and theorems to apply in order to compute the angles correctly. And on top of that my laptop crashed every time I tried to use tan().
4. Displaying images was difficult because I didn't know how to resize the images. After changing the size of an image in the paint application it was fine, but I still don't know how to resize the pictures in fltk.
5. The timer and score display were also problematic. I tried with the timeout function as well as with validators but firstly nothing worked. The timer and the score itself were functional, just displaying them didn't, but after a while, I figured out how to convert integers into a label. So it is all working for now.
6. Writing a function that would check for collision between the ball and bricks was an exhausting process that is not working properly all the time in this version yet. I believe the problem is that I treat the ball as a square, not a circle.
7. Making the game smart was another challenge. I don't know exactly how to make it smart, but I've managed to come up with a semi-intelligent function to keep it from losing.

Discussion

I think that my code is rather well structured and efficient but it still lacks the AI and sometimes has some bugs, so that could be improved in the future. In my opinion, the project was quite complicated taking into consideration my skills and previous knowledge. Without lectures and help the project would be almost impossible to make if one wanted to do it in c++ in fltk. Additionally, once a solution was shown during the lecture and we knew how to handle a problem, we still had to come up with a solution on our own to not commit plagiarism. It was really hard. Also if I came up with something before it was discussed in class it may look like I didn't have an original solution to a problem.

References

Benedetto, S., Carbone, A., Drai-Zerbib, V., Pedrotti, M., & Baccino, T. (2014). Effects of luminance and illuminance on visual fatigue and arousal during digital reading.

Computers in Human Behavior, *41*, 112–119. https://doi.org/10.1016/j.chb.2014.09.023

Howell, B. F., & Schifferstein, H. N. (2019). How neutral coloured backgrounds affect the attractiveness and expensiveness of fresh produce. *Food Quality and Preference*, *78*, 103718. https://doi.org/10.1016/j.foodqual.2019.05.018

Kim, K., Erickson, A., Lambert, A., Bruder, G., & Welch, G. (2019). Effects of Dark Mode on Visual Fatigue and Acuity in Optical See-Through Head-Mounted Displays. *Symposium on Spatial User Interaction*. https://doi.org/10.1145/3357251.3357584

Kuniecki, M., Pilarczyk, J., & Wichary, S. (2015). The color red attracts attention in an emotional context. An ERP study. *Frontiers in Human Neuroscience*, *9*. https://doi.org/10.3389/fnhum.2015.00212

Löffler, D., Giron, L., & Hurtienne, J. (2017). Night Mode, Dark Thoughts: Background Color Influences the Perceived Sentiment of Chat Messages. *Human-Computer Interaction - INTERACT 2017*, 184–201. https://doi.org/10.1007/978-3-319-67684-5_12

Morrone, M. C., Denti, V., & Spinelli, D. (2002). Color and Luminance Contrasts Attract Independent Attention. *Current Biology*, *12*(13), 1134–1137. https://doi.org/10.1016/s0960-9822(02)00921-1

Appendix 1 - plank bounce

```
void TheWindow::plank_bounce(){
            // of the ball is 40 units above ground (or for some reason slightly below)   -> checking if the paddle is there
            if (this->ball->getY()>=maxymalney-40){
                //if the ball x coordinate is larger than the plank x coordinate (-10) because of the radius of the ball
                // and the ball x coordinate is smaller than the plank x coordinate + it's length minus the radius of the ball
                // that means the ball is hitting the plank and thus the speedY should change to UP
                if(this->ball->getX()>=(this->plank->getX()-10) && (this->ball->getX()<=(this->plank->getX()+110))){
                    this->ball->speedY=-this->ball->speedY;
                    if ((this->ball->getX()<= this->plank->getX()+14)){
                        // going from left to right
                        if (this->ball->speedX >= 0){
                            this->ball->speedX-=2;   // should calculate 22.5 degrees but the computer crashes when I try to calculate
                        }
                        //going from right to left or at 90 degrees
                        else{
                            this->ball->speedX+=2;
                        }
                    }
                    else if ((this->ball->getX()>= this->plank->getX()+86)){
                        //going from left to right
                        if (this->ball->speedX >= 0){
                            this->ball->speedX+=2;
                        }
                        // going from right to left or at 90 degrees
                        else{
                            this->ball->speedX-=2;
                        }
                    }
                }
                else{
                    //that means the ball fall on the ground and thus one life is subtracted
                    this->life-=1;
                    // and reposition ball to the starting position and wait for user input to start the ball movement again
                    this->ball->speedY=0;
                    this->ball->speedX=0;
                    this->ball->move(maxymalnex/2-10, maxymalney-45);
```

Appendix 2

```
void borders_bounce(){
    // statements that prevents the ball from going
    //over the right or the left of the screen
    if (this->getX() >=620){
        this->speedX=-abs(this->speedX);
    }
    if (this->getX() <=0){
        this->speedX=abs(this->speedX);
    }
    // statement that prevents the ball from going
    //over the top or the bottom of the screen
    if (this->getY() <=60 || this->getY() >=460 ){
                this->speedY=-this->speedY;
            }
}
```

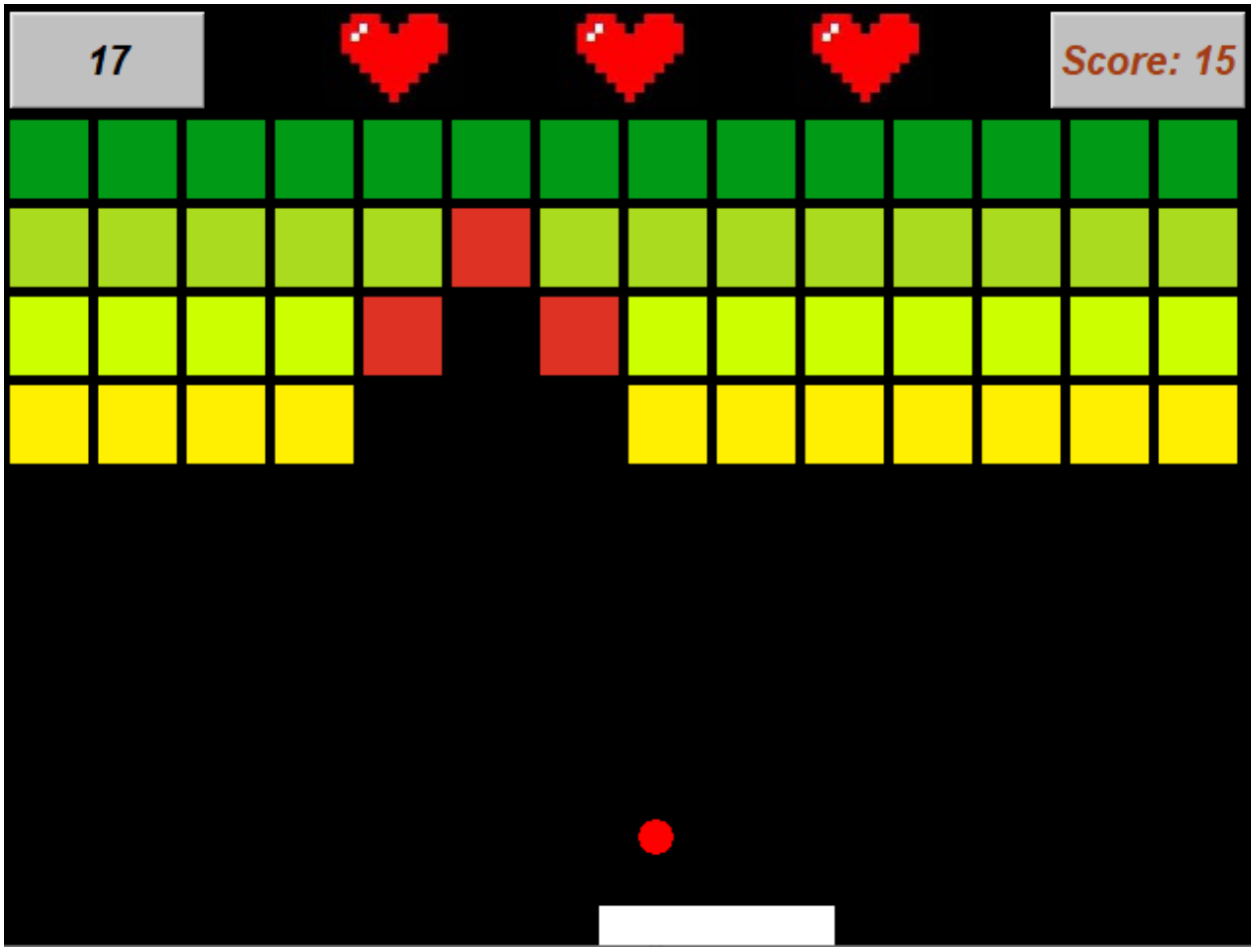
Appendix 3

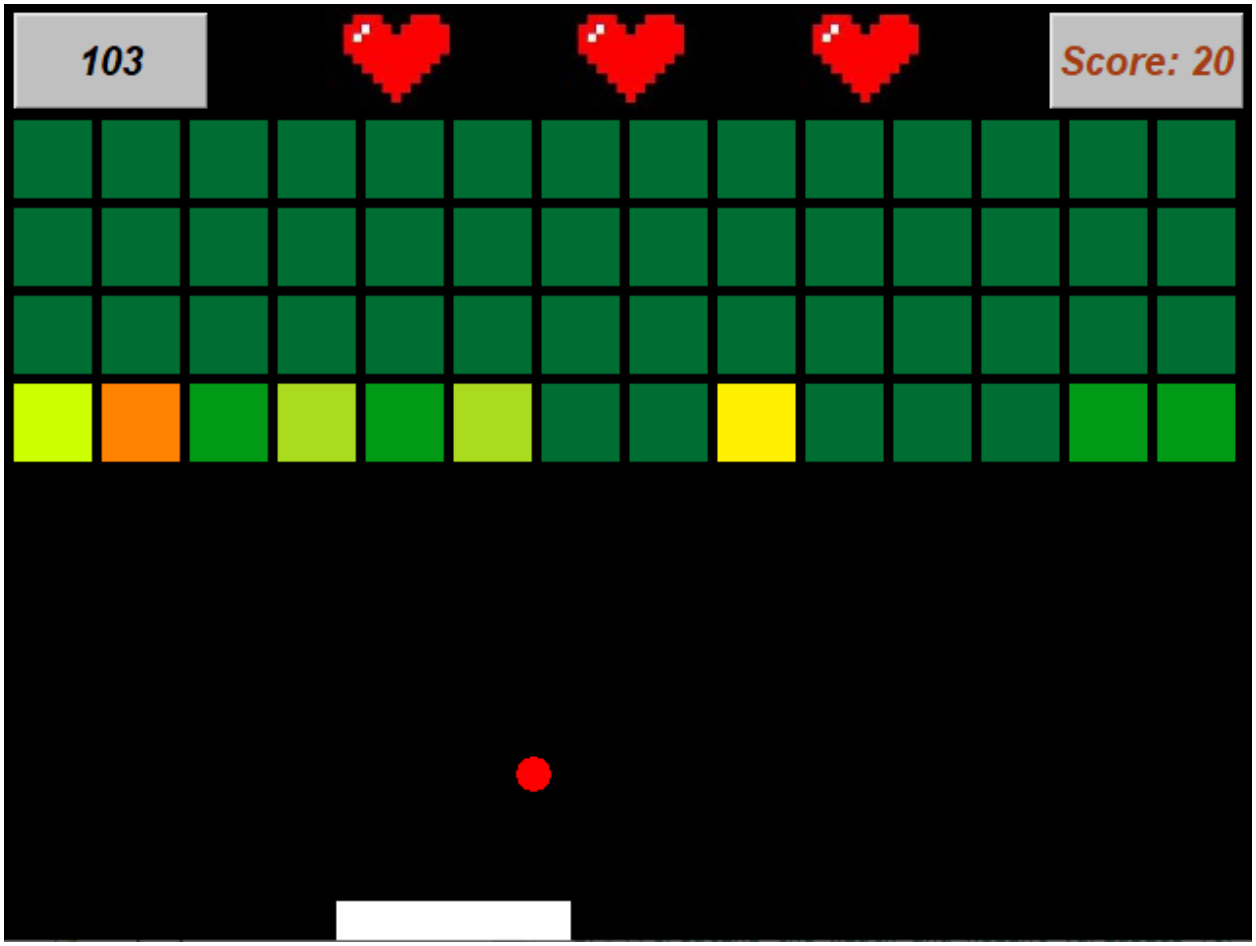
```
  int maxx_brick = 640;
  //they are allowed to show up only in the upper part of the screen
  int maxy_brick = 480/2;
  // since the dimensions are not accessible from this scope I redefine
  int brickx = 40;
  int bricky = 40;
  // getting the coordinates of the first brick
  int row = this->getX();
  int col = this->getY();
 creating a nested loop that will create bricks that are going to be 5
  // once a row is filled with bricks it will move one row lower and cc
        for (int i=col; i<=(maxy_brick-bricky); i+=bricky+5){
            for (int j= row; j<=(maxx_brick-brickx); j+=brickx+5){
                // Building brick
                //Brick*brick = new Brick(j,i);
                Block*brick = new Block(j,i);
                //giving it the right color
                brick -> color(colors[hardness]);
                //adding a brick to brick's vector
                this->body.push_back(brick);
            }
        }
        //displaying bricks on the screen
   this-> show();
```

Appendix 4 - starting movement of the ball

```
int Ball::handle(int event){

    //getting current position of an object
    int currentx=this->getX();
    int currenty=this->getY();

    //an algorithm that will execute different things depending on which key will be pressed
    switch(event){
    case FL_KEYUP:// keyboard key released
                {

        if(Fl::event_key() == FL_Up){
            this->speedY=-5;
            this->move((currentx+speedX),currenty+this->speedY);
            redraw();
            return 1;
            break;
        }

     //no break
     default: return Fl_Widget::handle(event);
    }
    }
}
```

Appendix 5 - integrity of the bricks


17
Score: 15



Appendix 6

```
void Block::hit(){
        // if the brick is hit, set its hardness to a lower one
        //and change its color so that it correspond the hardness, then show the brick
            this->hardness-=1;
            this -> color(colors[hardness]);
            this-> show();
            //if the brick is destroyed move to right lower corner so that it does not
            //interfere with ball movements
            if(hardness<=0){
                this->move(640,480);
        }
}
```

Appendix 7

```
void TheWindow::smart_movement(){
    // how to start a game
    if (this->ball->speedY == 0 && life>0){
        this->ball->speedY=-5;
        this->ball->speedX=2.1;
    }
    // calculate if the ball is outside the plank, to its right
    int differnece_right = (this->plank->getX()+100) - (this -> ball->getX());
    //if yes, move the plank to the right
    if( differnece_right < 0){
        int oldX =  this->plank->getX();
        int oldY = this->plank->getY();
        this->plank->move(oldX+15, oldY);
    }
    // check if the ball is outside the plank but on the left
    int differnece_left = (this->plank->getX()) - (this -> ball->getX());
    //if yes, move plank to the left
    if( differnece_left > 0){
        int oldX =  this->plank->getX();
        int oldY = this->plank->getY();
        this->plank->move(oldX-15, oldY);
    }

}
```

Appendix 8 - the function to check lifes and handle loosing the game

```
void TheWindow::check_lifes(){
    // if life is lost for the first time hide the most left heart
    if( this->life == 2){
        heart1->hide();
    }
    //if life is lost for the second time, hide the middle heart
    else if(this->life == 1){
        heart2 -> hide();
    }

    //if the third life is lost, hide the last heart
    else if(this->life <=0){
        heart3 -> hide();
        //stop the game;
        this->ball->hide();
        this->ball->speedY=0;
        this->ball->speedX=0;
        this->plank->hide();
        // display the game over box
        this->game_over-> show();

    }
}
```

Appendix 9 - last part of te function that deals with the brick-ball collision as well as with winning the game

```
}
// if the booleans for change are true change the speed how specified
if (changeX || changeY){
    if(changeX){
        this->ball->speedX=-this->ball->speedX;

    }
    if(changeY){
        this->ball->speedY=-this->ball->speedY;
    }
//change the score and hit an adequate brick
brick->hit();
this->score+=1;
this->scorecount= "Score: " + to_string(score);

//if all the bricks are destroyed, the player wins
if (score>=560){
    //stop the game
    this->ball->hide();
    this->ball->speedY=0;
    this->ball->speedX=0;
    this->plank->hide();
    // display the winning message
    this->winning-> show();

}
break;
}
```