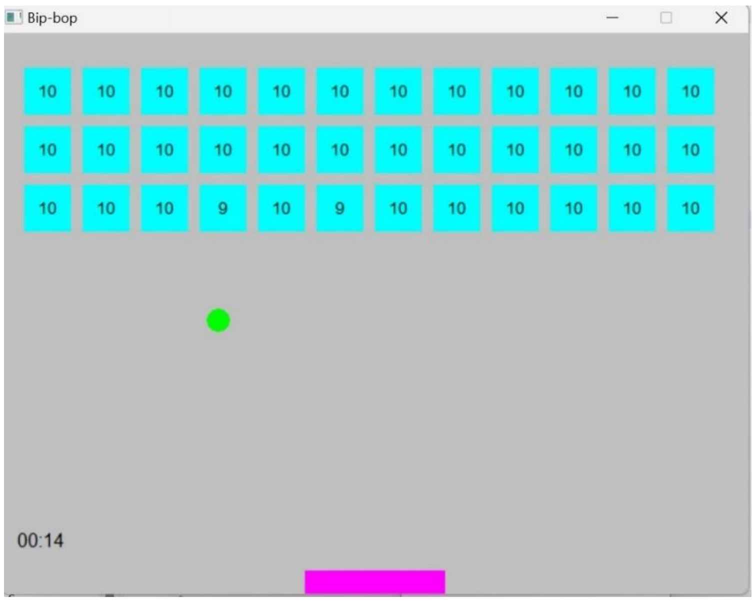
Programming with C/C++

Academic year: 2022-2023, Fall semester

Programming assignment

The BipBop

An Atari game



Student name: Serenay Doganca Cetin

Student number: 2049681

1. Overview of project results

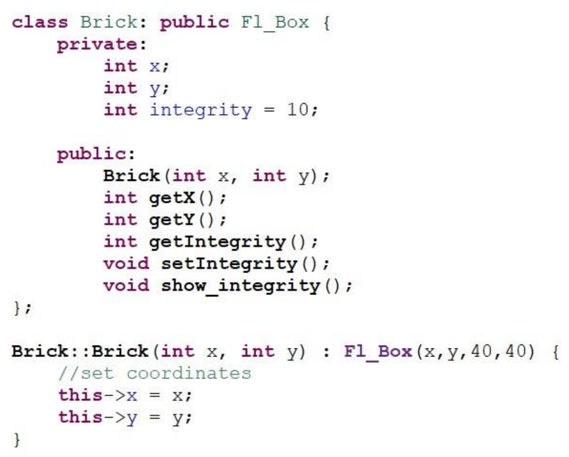
The project consists of five files; one main file with the main function and one for the ball, the bricks, the paddle, and the timer. I have implemented respectively creating the bricks (1 week), creating the paddle and movement of the paddle according to keyboards pressed (two weeks), creating the ball and the movement of the ball, its behavior (bouncing back with the same angle) when it hits the ball, paddle or the window boundaries (three weeks), and finally creating the timer for when it reaches 0, the game is over (two days). When the game window is open, the ball is situated in the middle of the window and its direction (top-right, top-left, bottom-right, bottom-left) is randomly assigned. When the ball hits the paddle, it bounces back at the same angle. Each of the bricks has an integrity number which is initially 10 and is visible on each brick. If the ball hits one of the bricks, its integrity number decreases by one and when the integrity number reaches zero, the brick breaks and disappears. The user wins when all of the bricks are broken before the time limit. However, if the time limit expires, or the paddle fails to catch the ball and the ball touches the lower boundary three times, the user loses. The most challenging part was implementing the ball's movement since it required a delay to observe the movement.

2. Tasks and objectives

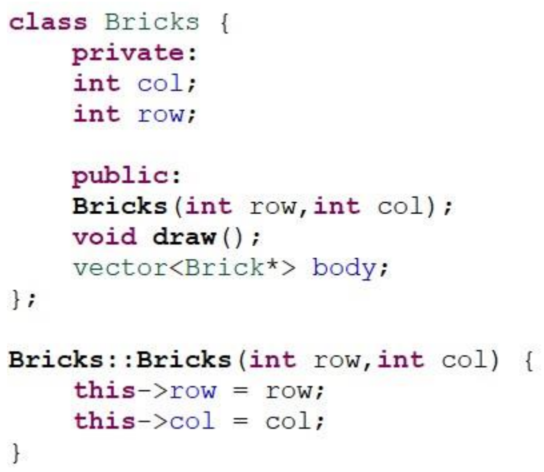
a. Implementation of window, board, ball, and bricks

Ball, Paddle, and Bricks are all different classes inherited from Fl_Box class.

- Bricks



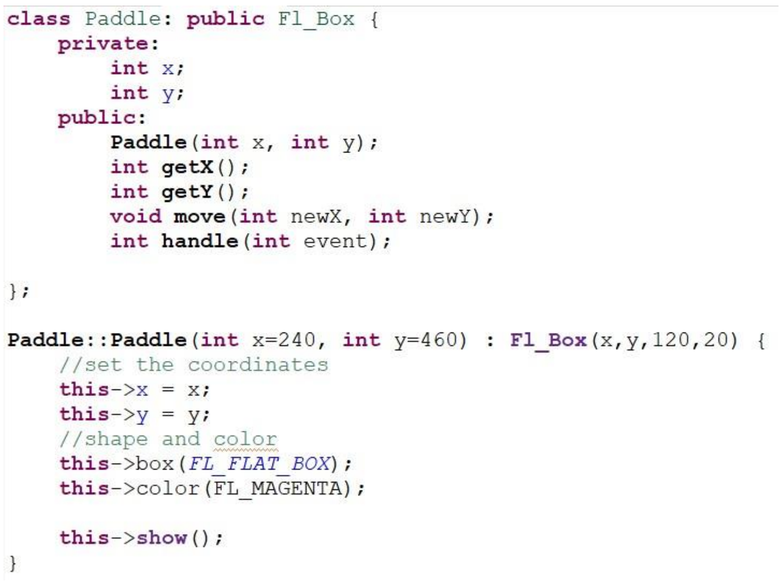
```
class Brick: public Fl_Box {
    private:
        int x;
        int y;
        int integrity = 10;

    public:
        Brick(int x, int y);
        int getX();
        int getY();
        int getIntegrity();
        void setIntegrity();
        void show_integrity();
};

Brick::Brick(int x, int y) : Fl_Box(x,y,40,40) {
    //set coordinates
    this->x = x;
    this->y = y;
}
```



```
class Bricks {
    private:
    int col;
    int row;

    public:
    Bricks(int row,int col);
    void draw();
    vector<Brick*> body;
};

Bricks::Bricks(int row,int col) {
    this->row = row;
    this->col = col;
}
```

First, a Brick class is created for the implementation of the bricks, which inherits from Fl_Box. A brick object has a height and width equal to 40 and the coordinate of the top left corner should be specified when creating a new object. In order to store all the bricks together, the Bricks class is created. It takes the number of rows and columns as arguments and creates Brick objects, draws them, shows the integrity numbers, and stores them in a vector.

- Paddle



```cpp
class Paddle: public Fl_Box {
    private:
        int x;
        int y;
    public:
        Paddle(int x, int y);
        int getX();
        int getY();
        void move(int newX, int newY);
        int handle(int event);

};

Paddle::Paddle(int x=240, int y=460) : Fl_Box(x,y,120,20) {
    //set the coordinates
    this->x = x;
    this->y = y;
    //shape and color
    this->box(FL_FLAT_BOX);
    this->color(FL_MAGENTA);

    this->show();
}
```

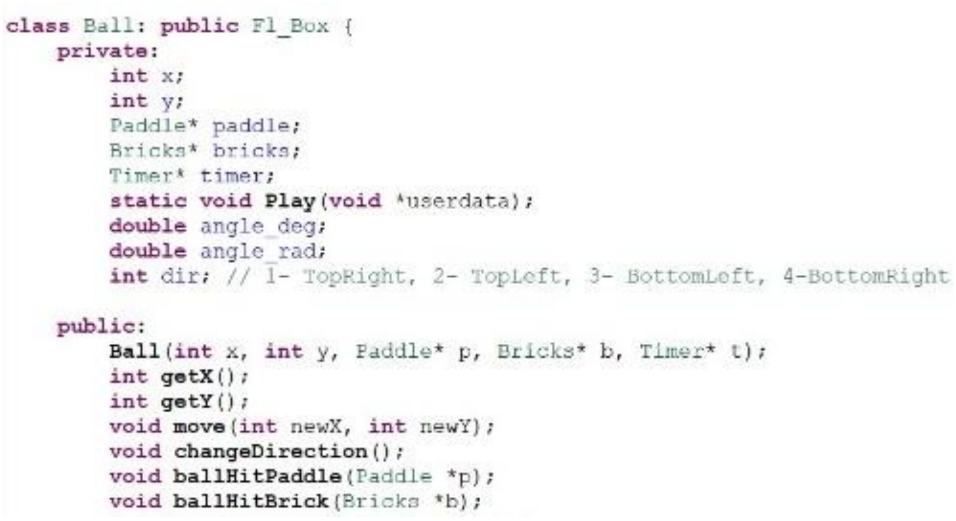
A Paddle object also requires a coordinate (x,y) to be created and its dimensions are 120 x 20 units. For the control mechanism of the paddle, the handle() method is used where if the left key is pressed, it moves to the left and if the right key is pressed, it moves to the right. It is not allowed to move beyond the left and right window boundaries. In every keypress, it moves 20 units which seems like an optimum speed. Every time move() method is called, the paddle hides, relocates to the new coordinates, and reappears.

- Window

The window is created inside the main() function using Fl_Window and its dimensions are 640 x 480 units.

- Ball

The ball class is the most elaborate class with many methods because it is where the whole gameplay is also implemented. Since I was not able to implement the movement or control of the ball at a separate file or class, and attempts of communicating the ball with other objects that are created in the main file also failed, the Ball class had to be the gameplay class as well.



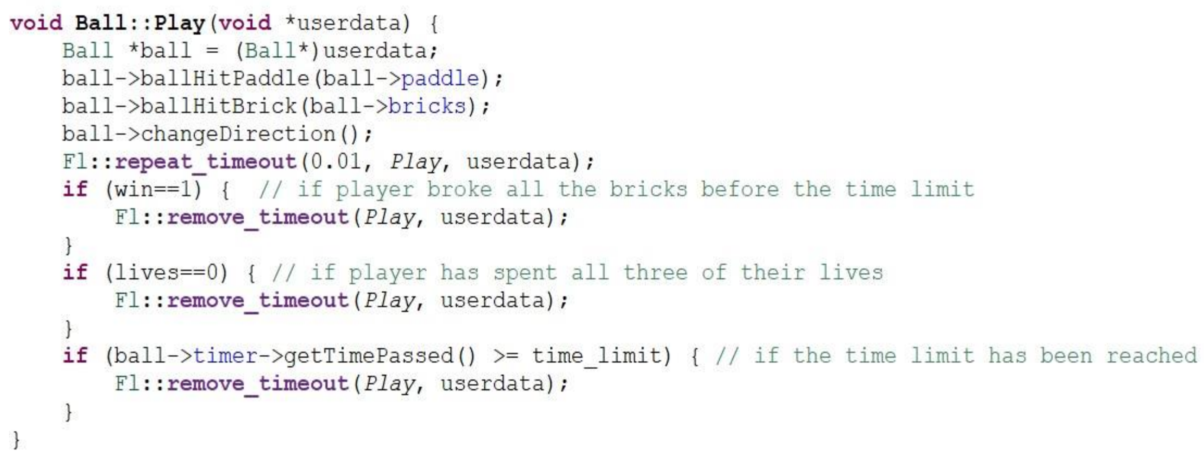
```
class Ball: public Fl_Box {
    private:
        int x;
        int y;
        Paddle* paddle;
        Bricks* bricks;
        Timer* timer;
        static void Play(void *userdata);
        double angle_deg;
        double angle_rad;
        int dir; // 1- TopRight, 2- TopLeft, 3- BottomLeft, 4-BottomRight

    public:
        Ball(int x, int y, Paddle* p, Bricks* b, Timer* t);
        int getX();
        int getY();
        void move(int newX, int newY);
        void changeDirection();
        void ballHitPaddle(Paddle *p);
        void ballHitBrick(Bricks *b);
```

The movement of the ball is implemented using the Fl_timeout method. This part was very challenging, so most of the logic of the game had to be implemented in the Play() method of the Ball class.

A ball object has coordinates, direction, and degree attributes. The direction has four options as can be seen in the above code. The ball moves two units in the y-axis, and 2/(tan(degree)) units in the x-axis every time the move() function, which works in the same principles as the move() function of the Paddle class, is called. However, the degree of the ball never changes throughout the game. At first, assigning the ball a random degree at the beginning of the game was planned, but for different degrees, the speed of the ball also changed because it was dependent on the angle and it caused inconsistencies in the game, which is why it was removed.

As mentioned before, since the communication between the ball and the other objects that were created in other files or classes was not able to be established, the paddle, bricks, and timer objects that are created inside the main() function are provided in the Ball object as arguments when creating a new instance.

The ball has 20 units of radius and moves in its direction until it hits the paddle, a brick, or a window boundary. Then according to its direction and position, a new direction is assigned to the ball and it continues its movement until the game is over.



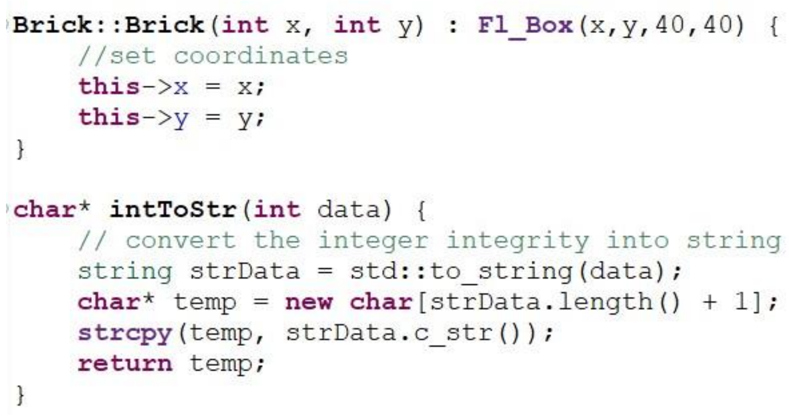
```
void Ball::Play(void *userdata) {
    Ball *ball = (Ball*)userdata;
    ball->ballHitPaddle(ball->paddle);
    ball->ballHitBrick(ball->bricks);
    ball->changeDirection();
    Fl::repeat_timeout(0.01, Play, userdata);
    if (win==1) {  // if player broke all the bricks before the time limit
        Fl::remove_timeout(Play, userdata);
    }
    if (lives==0) { // if player has spent all three of their lives
        Fl::remove_timeout(Play, userdata);
    }
    if (ball->timer->getTimePassed() >= time_limit) { // if the time limit has been reached
        Fl::remove_timeout(Play, userdata);
    }
}
```

3. Challenges

3.1 Challenges solved

- The handle method for the paddle was not able to capture or recognize keyboard events in my guess, because it didn't perform the tasks that were inside the conditional statements. I have no idea how I solved it, it might have solved itself because magically it started working.

- Converting the integrity number into a string was challenging. Since I used the label() method and it takes const char* as an argument, I tried to convert from int to const char*. After many failing attempts, I realized I need to turn it into some kind of an array, then I successfully use the converted string as an argument in the label() method.



```
Brick::Brick(int x, int y) : Fl_Box(x,y,40,40) {
    //set coordinates
    this->x = x;
    this->y = y;
}

char* intToStr(int data) {
    // convert the integer integrity into string
    string strData = std::to_string(data);
    char* temp = new char[strData.length() + 1];
    strcpy(temp, strData.c_str());
    return temp;
}
```

3.2 Challenges addressed but not solved

- In order to display the movement of the ball, the add_timeout method was used. It requires a Fl_Timeout_Handler as an argument which is of type void*. From what I understand, the (void* userdata) term is used to capture an object of the certain class to where this method belongs, created somewhere else. So, in order to capture the Ball object created in the main file, I had to create this timeout handler inside the Ball class. Otherwise, (* userdata) does not know what object to capture. Also. Since the movement of the ball happens inside this method, I had to find a way to include paddle, brick, and timer objects inside it. That's why, when creating a ball, all the other objects are given as an argument, and that way we can access their positions and other attributes.

3.3 Challenges not solved

- An image or a text indicating if the user has won or lost the game was considered to be shown at the end of the game. Because of the reasons mentioned above, the command for it also had to be included in the timeout handler of the ball. However, when the code to show an image or text was included inside this method, it did not work because it did not have access to the window. I could solve this by creating the image or text objects in the main file and adding them as arguments to the Ball class, but it would make the already ugly code uglier, which is why it is not implemented.

- Implementation of showing how many lives are left was considered, however it was not accomplished. I tried to use fl_draw() method for it but even though I didn't receive any errors, it was not visible in the game window.

4. Discussion and future work

As mentioned in the previous sections, if the gameplay could be created as a separate class, the code would be more efficient because this way, all the actions happen inside of a class that actually belongs to an object. Also, an image or text could be added at the end of the game indicating if the player has won or lost. An AI agent that plays the game is already a part of this project, however, due to time limitations, it could not be implemented. So, an AI agent that plays the game could be a new addition to the project.