EHB 354E PROJECT

NAME OF THE STUDENT: ADNAN CAN BÜYÜKŞİRİN

STUDENT NUMBER: 040190237

PROJECT DEADLINE: 24/05/2023

TITLE OF THE COURSE: OBJECT ORIENTED PROGRAMMING

CRN OF THE COURSE: 24547

Project Report

Introduction

In this project, we tried to implement a classic Mario game using the SFML library in C++. Mario will move with the Up, Down, Left, and Right keys. It can jump and stay on bricks. Turtles are Mario's enemies. They drop from the pipes every 7 seconds. The maximum amount of turtles is 5. If the turtles reach the bottom pipes, they teleport back to the top pipes. If Mario touches the turtles from above, the turtle dies. The dead turtle will fall through the screen and disappear. If Mario kills a turtle, the player gains 100 points. If the turtle intersects with Mario from above, left, or right, Mario dies. Again, Mario will fall from the screen and respawns. Each time Mario dies, the player loses 1 life. The player has a total of 3 lives. If the player loses all of their lives, the game will show the "Game Over!" text on the screen and the game returns to the main menu. On the other hand, if Mario kills all the turtles, (i.e. if the player achieves 500 points) Mario wins the game, and the game will display the "You Win" text. Then it will return to the main menu to restart the game. It should be noted that turtles increase their velocity (aggravate) by 1.1 times (aggravation ratio) every 20 seconds. For the game to be smooth, after the aggravation ratio exceeds 2.5, the game will not allow the turtles to aggravate. If the turtles meet with each other, they will change their direction. Also, Mario can kill a turtle by hitting from below the brick which the turtle stands on. The main menu and the game screen can be seen below in Figures 1, and 2.

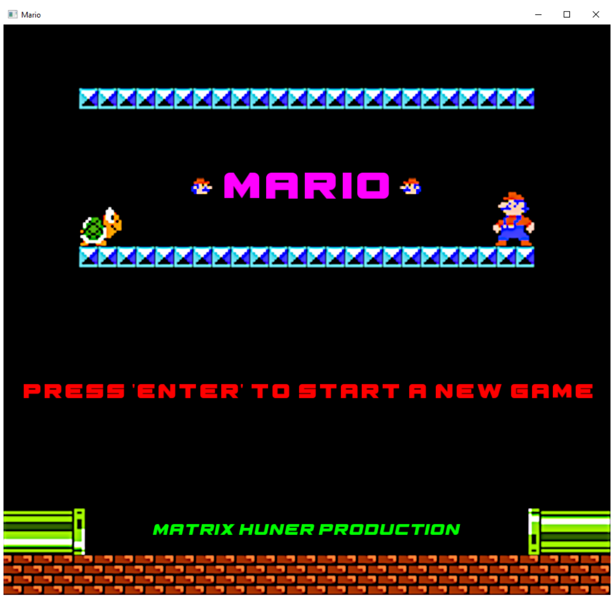

Figure 1. Snapshot of the Main Menu

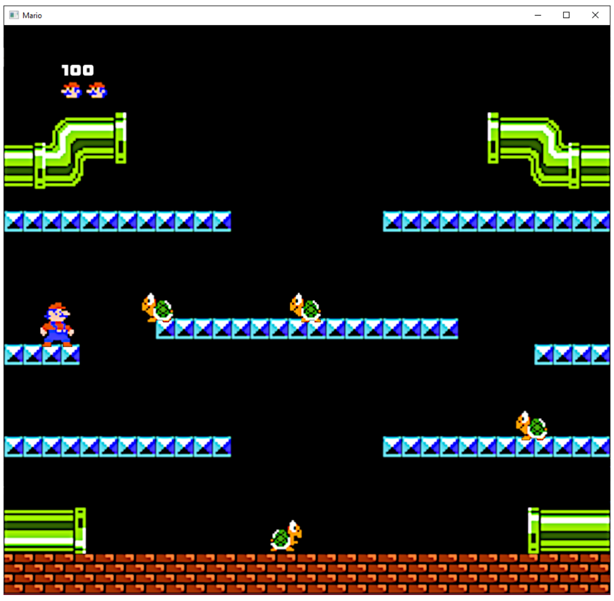

Figure 2. Snapshot of the Game

Team Info

I implemented this project with Efe Mert Çırpar. He did the Mario and all the background objects classes (bricks, pipes, ground). I did the Object, Turtle, and the Scoreboard. The menu class was first written by me, but my teammate redesigned the graphics of the menu. He made the menu more appealing. The communication between the Menu, and the Game Classes is done in the main file by me. The menu class is not on the project description file, but we thought that the menu should have its own class even though it could have been implemented inside the Game Class. For the Game class, we divide it between us. I implemented drawBackground (sf::RenderWindow& window), and onFloor(Object* object) methods, and he implemented the checkCollusion(Turtle* t, Mario* m) method. He also wrote different onFloor() methods specific to Mario since it has a different shape than the turtle. Thus, the methods need different tolerances to compare the heights, and widths of the objects. We implemented more methods in Game Class. I created methods for the turtle to check if the turtle reaches the pipes. I also implemented a Heterogeneous List to hold both Mario and the turtles. Hence all the methods for implementing the list, getting the turtle objects, and the Mario object from that list, and the methods for deleting a turtle was implemented by me. For the Heterogeneous List, I used polymorphism which will be explained in more detail later in the report. Also, the method to show the end game displays were implemented by me. Although we did those separately, we always checked each other's codes and made them better. Also, we helped each other throughout the entire process.

Implementation

In this section, the project will be described as class by class.

1) Object Class

This class is created to be an abstract class. In other words, no objects will be created from this class. So the Object Class is our base class. We derive Mario, Turtle, and the background classes from this class using inheritance. Although we create an object pointer, it holds Mario and the turtles. Since from that pointer, we reach Mario and the turtles, all of the methods that Mario and the turtles use must be specifically defined as virtual so that the base class pointer can access the derived class methods. This is called "polymorphism". It should be noted that those virtually defined methods are abstract methods that will be overridden. Thus, those methods have empty bodies. Also, to have no memory leak when deleting an object from that list, the destructor of the Object class is defined as virtual as well. So if we delete a Mario or a turtle object from that list, it will not only call the base class's pointer, it will call the derived class's pointer as well. Since we create a heterogeneous linked list, the *next pointer attribute is added inside the object class privately. In other words, the *next pointer is "encapsulated" inside the Object class. Furthermore, Game Class is defined as a "friend class" so that when we create a heterogeneous list, Game Class can reach out to the attributes (even the private ones). In that way, the Game class will be able to make use of the *next pointer of the Object Class. Moreover, some virtual methods of Object class have the same name but with different parameters and return types. This event is called "Overloading" in Object Oriented Programming.

As we have mentioned before, Mario, Turtle, and background classes are derived from the Object class. So to reach Objects attributes from those classes, we define those attributes as "protected". Visualization of this inheritance can be seen below in Figure 3 as a UML diagram.

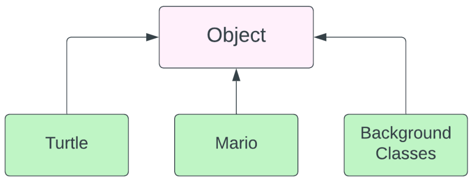

Figure 3. Inheritance Diagram

Those inherited classes all share common attributes and methods. Those needed attributes are written as "protected", and those methods are written as "public". Some of these common methods are listed as such: void setPosition(Vector2f), Vector2f getPosition(void), void draw(RenderWindow&), IntRect boundingBox(void)…

2) Turtle Class

As stated in the Object Class section, the Turtle class is derived from Object Class using inheritance. The movement is done by assigning a constant velocity value to the Vx (x-velocity) and multiplying it by the aggravation ratio as said in the Object Class section. This aggravation ratio is updated every 20 seconds. When the ratio is updated, Vx changes accordingly. If the turtle reaches the edge of the brick, it falls. This falling operation is done by updating the Vy (y-velocity) by the ratio of the specified gravity constant. This Vy update method is defined in Object Class so that Mario and the turtles can use that method. If the turtle reaches a side of the map, we basically multiply Vx by (-1) so that turtle turns back. Reaching a side on the map is done by basically some if() check based on the turtle's sprite's coordinates (x, y), width, and height. If Mario kills a turtle, the turtle needs to fall to the ground and disappear. For that, we need to mark the dead turtles such that they will not do the other operations as the alive turtles. Thus, in the Turtle class, we created an attribute called "death_marker" so that we know that the turtle is dead. How the turtles are implemented will be analyzed in detail in the Game Class section.

When a turtle reaches one of the pipes below, it teleports to the pipes above. If the turtle reaches the bottom left pipe, it will teleport to the top right pipe, and if it reaches the bottom right pipe, it will teleport to the top left pipe. This teleportation is done by comparing the turtle's sprite's coordinates, width, and height with the pipes. When the collision of the turtle with the pipe is done, we basically set the coordinates of the turtle to the specific pipe at the top. In turtle class, which direction is turtle headed is held in the Base Class (Object Class) as "int heading". That attribute is updated if the turtle reaches a side, or if it teleports.

The animation of the turtle is done by implementing a state diagram as shown below in Figure 4. Each i'th state corresponds to the i'th texture. Those textures are initialized at the constructor of the Turtle Class. By rapidly changing the states, we produce the moving animation of the turtle. To achieve a smoother animation, a clock is applied. The applied clock duration is not high enough to affect game performance. Also, from the state diagram below we can see that, the state change is based on the heading, and the Vx of the turtle.

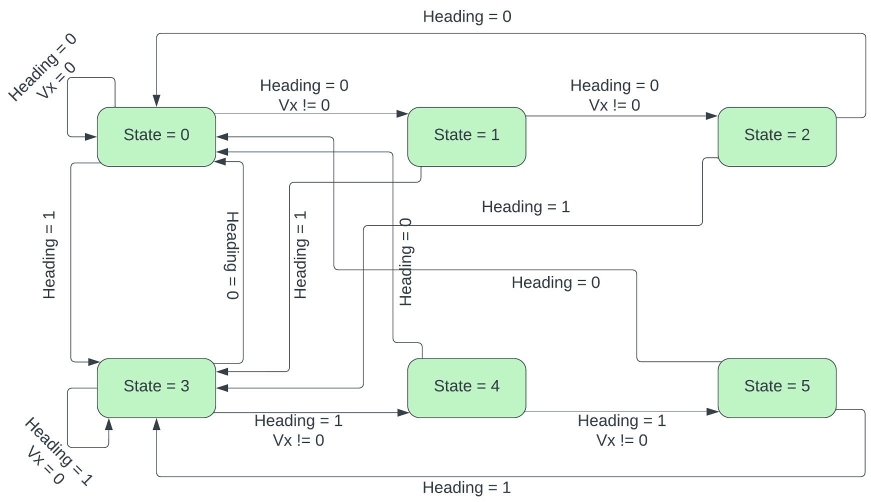

Figure 4. Turtle State Diagram

3) Game Class

As stated in the Team Info part, the implementation of the Game Class is divided between me and my teammate. The game class holds the following important game states (attributes):

- Pointers to Background objects (Pipes, bricks, floor)
- A pointer to the Scoreboard object
- An Object Class pointer to hold a heterogeneous list (A list to hold the turtles and the Mario).

All the objects are initialized in the constructor of the Game class. For the heterogeneous list, we add Mario and a turtle to the constructor to begin the game. Thus, when the game is on, a turtle and Mario exist in it. The heterogeneous list implemented in this class is actually a stack. The visualization of the stack can be seen below in Figure 5.

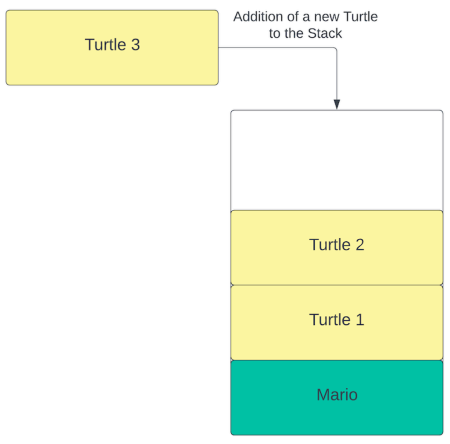

Figure 5. Visualization of the used Heterogeneous List

From Figure 5, it can be understood that if the list size is N, the turtles can be found by traversing the first N-1 elements of the list. That is the main intuition for the Turtle* getTurtle(int index) function that is implemented. We can get the i'th turtle easily since we know for sure that turtles are the first N-1 elements of the list. This function returns a pointer to i'th turtle so that we can access the methods of it using polymorphism. Since the methods of the Turtle and the Mario are defined as "virtual", we can access the methods of the Turtle and the Mario classes from the base pointer that holds the derived classes' pointer objects. As it was stated in the Object Class section, the destructors of the Object Class are defined as "virtual". So, when we create a deletion method for deleting a turtle from that list, we do not have a memory leak. If the destructor of the Object class was not declared as virtual, deleting a turtle object in that list would only call the destructor for the Object class. It would not have been called the destructor of the Turtle class. Thus, it would have created a memory leak.

The collision controls are implemented inside the Game Class. For those methods, we basically get the positions, widths, and heights of the objects of interest. By taking the object's origins into account, we have defined methods to check if the object is on a specific floor, or if the object is colliding with the other object of interest. All of these methods return true if the objects are colliding. Otherwise, the methods return false. These bool return types are checked in the "int update(void)" function which is the main function where the game runs. The reason for the return type is defined as int will be explained later in the report.

In the update() methods, first we check the lives of the Mario, and the score to see if the game ends. Depending on the satisfied condition endGameDisplay() function will be called and it will print on the screen whether Mario loses or wins. That display will be on the screen for 3 seconds. After that game screen closes, and the menu screen opens.

The Mario operations were done by my teammate. The operations were to check the Mario is on any ground, move the Mario by keyboard inputs, and check if the Mario is colliding with another object. After jumping, it was checked whether Mario fell onto a floor so that Mario could not jump again without falling down. Thus, Mario can no longer jump while in the air. Also, when a turtle collides with Mario from below, left and right, Mario is marked as dead. In this way, it would go into a different loop so that the falling animation of Mario is successfully applied. If Mario collides with a turtle from above, the same logic is done such that the turtle is marked as dead, and it will go into a different loop so that it can perform the falling (death) animation. The Mario operations would be analyzed thoroughly by my teammate's report. In this report, turtles operations in the "update()" method will be analyzed in more detail. The flowchart of the turtle operations can be seen below in Figure 6.

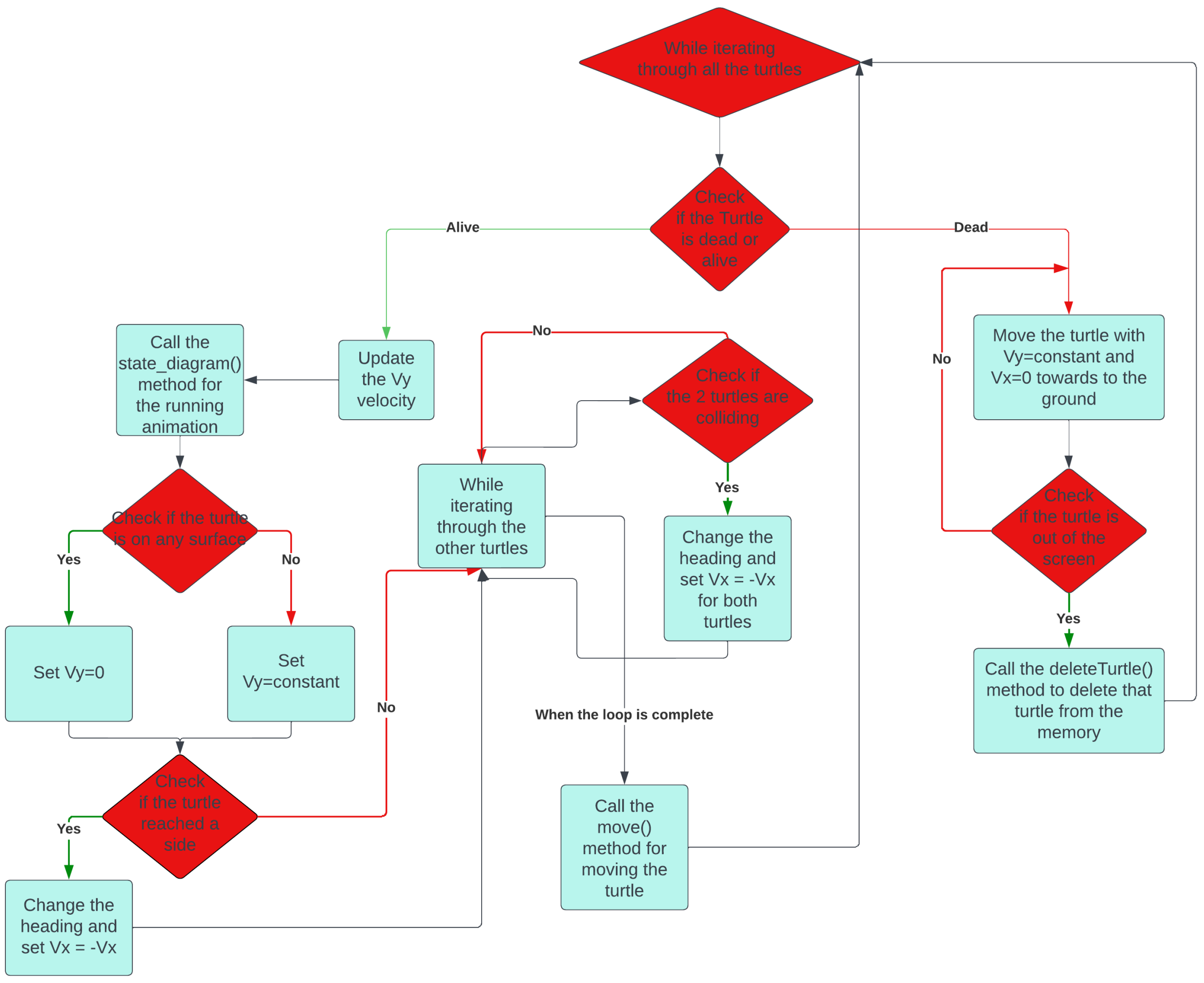
While iterating through all the turtles
Check if the Turtle is dead or alive
Alive
Dead
Update the Vy velocity
Call the state_diagram() method for the running animation
Check if the turtle is on any surface
Yes
No
Set Vy=0
Set Vy=constant
Check if the turtle reached a side
Yes
No
Change the heading and set Vx = -Vx
While iterating through the other turtles
No
Check if the 2 turtles are colliding
Yes
Change the heading and set Vx = -Vx for both turtles
When the loop is complete
Call the move() method for moving the turtle
Move the turtle with Vy=constant and Vx=0 towards to the ground
Check if the turtle is out of the screen
No
Yes
Call the deleteTurtle() method to delete that turtle from the memory

Figure 6. Flowchart of the Turtle Operations

A crucial thing to be noted is that there exists a bug about the collision of the turtles. If somehow 2 turtles collide like they are on top of each other, the collision works but they cannot get rid of each other. If another turtle comes and collides with them, it sometimes separates those turtles. This issue is temporarily solved by changing the time a new turtle drops from the pipe. But it may still happen during the game even though it is a rare condition.

4) Scoreboard Class

This class is nothing special. It holds the score, and the number of lives left. It also holds the print methods for printing out the score and the number of lives left. Moreover, it has "getters" and "setters" for the score, and the amount of lives left. The constructor of this class is used for setting the initial score and lives. Furthermore, it is used to load "mariohead.png", and the "font.ttf" files.

5) Background Classes

These classes are written by my teammate. Those classes are derived from the Object class. They do not have any methods in them. They only have their constructors for initializing their textures correspondingly. All the methods they use, are written inside the Object Class. Some of those methods can be listed as: void setPosition(Vector2f), Vector2f getPosition(void), void draw(RenderWindow&) …

6) Mario Class

This class is written by my teammate. It has similar methods to the Turtle Class. This class does not have a method for constructing a state diagram. The moving effects of Mario are done by setting a Clock and checking the elapsed time. Other methods are mostly similar to the ones in Turtle Class. This part will be explained in more detail in my teammate's report.

7) Menu Class

This class is created to design the Menu. It has a constructor to initialize the font, texts, and sprites. It has only one method which is the MainMenu() function. This function is used to draw the texts and the sprites into the screen.

The Main Function

When the program runs, it calls the main() function. In this function there exists only 2 objects:

- Game Object Pointer
- Menu Object Pointer

It should be noted that the "update()" method in the Game class returns an integer. It returns 0 if the game ends, or it returns 2 if the user clicks on the exit button on the screen. Moreover, the "MainMenu()" method inside the Menu class returns 1 if the user presses the "Enter" key, or it returns 2 if the user clicks on the exit button. This relationship inside the main function can be seen below as a flowchart In Figure 7. It is important to state that the game starts with the condition "0" in the switch case block.

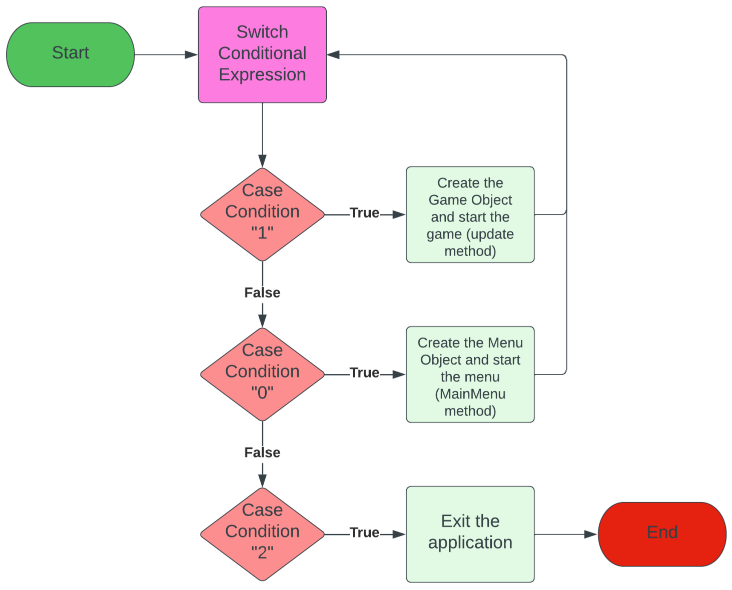
Start
Switch Conditional Expression
Case Condition "1"
True
Create the Game Object and start the game (update method)
False
Case Condition "0"
True
Create the Menu Object and start the menu (MainMenu method)
False
Case Condition "2"
True
Exit the application
End

Figure 8. Flowchart of the Main Function

Discussion

First of all, working with a teammate on a coding project was hard to get used to at first. Combining the codes was the hardest part. We always faced a lot of problems such as bugs that weren't there in our codes appearing when we combined our codes together. Other than that, implementing the onFloor() method was the hardest part for me. But, once I got used to the SFML library, the project was fun to code.

For the improvement part of the code, as I have mentioned before, there exists a bug when 2 turtles are colliding. Also, we could not able to implement the last 2 bonuses fully. The turtle dies directly when Mario hits turtle under the brick which it stands on. Also, when turtles collide with each other, they do not get surprised. Furthermore, the boundaries of the collisions might be tuned to be much more precise. Those are the things that can be improved in our game.