Introduction

This project is a simple and one map mario game. The main goal is to kill all 5 turtles and win. the turtles will be killed by jumping on them.

Team Info

I made the project all by myself.

Implementation

There are 5 classes in this project: Game, object, mario, turtle and scoreboard. The classes will be discussed separately but the main thing is that the game class makes the game by using its own and other classes' attributes and methods. At first I didn't notice the “minimum requirements”, so my methods are a little different but I tried to fix them as much as I can. But the thing is I finished the project already when I noticed my mistake.

I couldn't move the lives and the checkCollision to the classes that you want. The CheckCollision method is in object class and the lives are in the object class. But these do not affect the game

Object Class

This class has 9 attributes and 11 methods.

Attributes are pos, state, life, dead, changeddirection,verticalSpeed, horizontalSpeed, sprite and 8 texture. the sprite will change its texture while the objects are walking, jumping or dying. the verticalSpeed will change with some conditions and the turtles horizontal speed will increase after we kill turtle. The life is remaining life of the object, marios life will decrease by one after he get killed by turtle. dead is a state whether the object is dead or not. for some methods of objects and its childs methods this state is very important. state is used to change between different textures while walking to create a walking animation. the changeddirection is used to determine if the object change direction of its or not, the move method of the objects get the symmetry of the sprite according to this value.

```
7       Vector2f pos;
8       int state;
9   protected:
10      int life;
11      bool dead;
12      bool changedirection;
13      int verticalSpeed;
14      int horizontalSpeed;
15      Sprite sprite;
16      Texture textures[8];
17
```

attributes of Object class

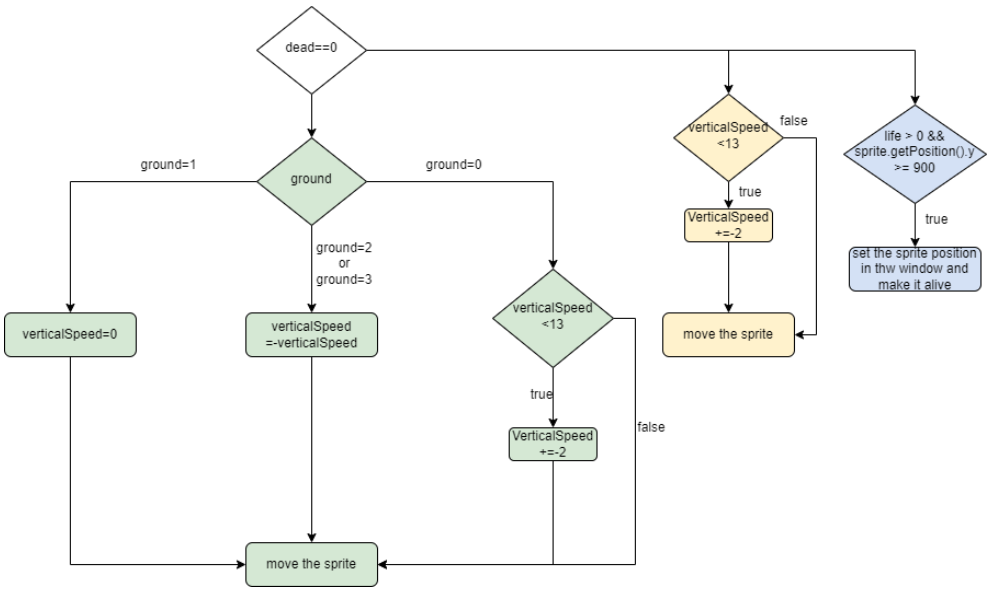
Only 2 of 11 methods are not simple like the others. Simple methods are getposition(), boundingBox(), getLife(), makeAlive(), makeDead(), getDead(), setPosition, draw(Render window*) and jump(). The get methods return the value of that thing, setPosition, makedead, makeAlive methods change the attributes of the object. The draw method will draw the sprite. boundingBox method will return the boundary of the sprite and jump method will assign some value to the verticalspeed.

```
Vector2f Object::getPosition()
{
    return sprite.getPosition();
}

void Object::setPosition(Vector2f pos)
{
    this->pos = pos;
    sprite.setPosition(pos);
}

void Object::draw(RenderWindow* window)
{
    window->draw(sprite);
}

void Object::makeDead()
{
    dead = 1;
    life = life - 1;
}

void Object::makeAlive()
{
    dead = 0;
    sprite.setPosition(5, 100);

}

void Object::jump()
{
    verticalSpeed = -30;
    sprite.move(0, verticalSpeed);
}

bool Object::getDead()
{
    return dead;
}

int Object::getLife()
{
    return life;
}

FloatRect Object::boundingBox()
{
    FloatRect spriteBound = sprite.getGlobalBounds();
    return spriteBound;
}
```

simple methods of Object class

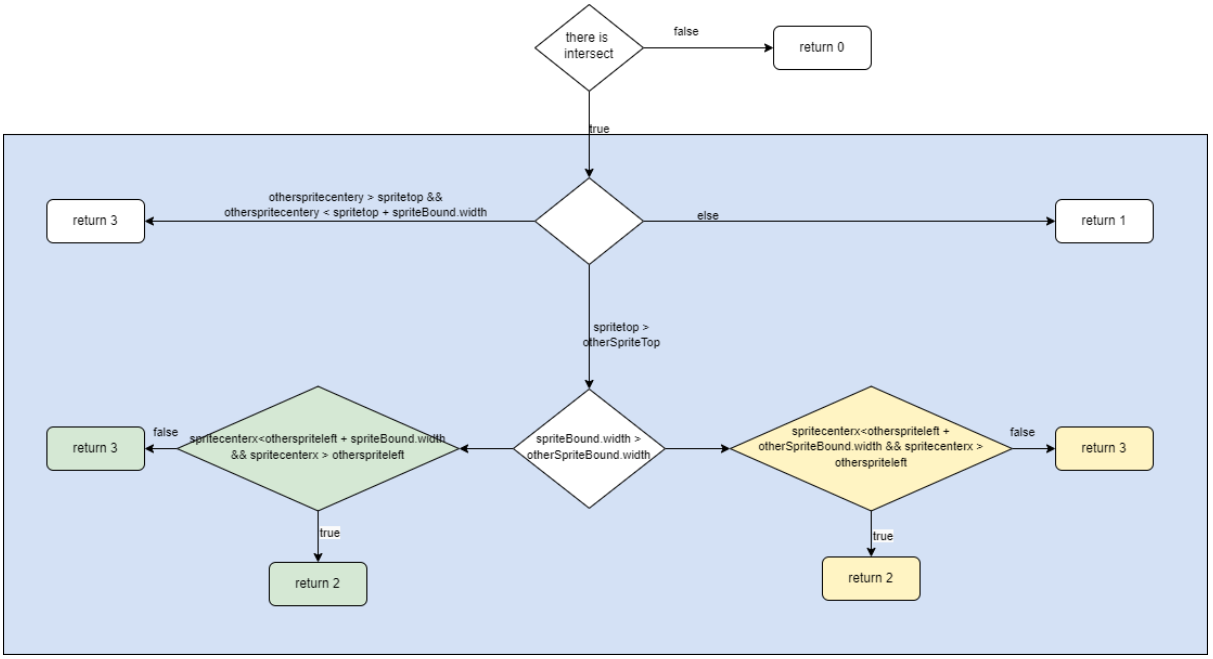
The other two methods, fall and checkCollision, are a little bigger. checkCollision looks for a collision between any bounding box with the object. so the parameter of this method will come from the boundingBox method. The fall method will arrange the vertical move of the objects. It has one parameter. This parameter will come from the onFloor method in game class which returns 1, 2, 3 and 0. These numbers represent where the sprite is. 1 means it's on the floor or bricks, 2, 3 means it's colliding with the bricks from under or near. and 0 means it's not colliding with any floor or brick. And i add a little if statement to set the objects on the floor with a specific y coordinate value to facilitate the teleportation of the turtles. to make these two function more clear I add some flowcharts.

```
58  void Object::fall(int ground)
59  {
60      if(dead==0)
61      {
62          if (ground == 1) verticalSpeed = 0;
63          else if (ground == 2 || ground == 3)
64          {
65              if (verticalSpeed < 0)
66              {
67                  verticalSpeed = -verticalSpeed;
68              }
69          }
70          else if (ground == 0)
71          {
72              if (verticalSpeed < 13)
73                  verticalSpeed += 2;
74          }
75          sprite.move(0, verticalSpeed);
76          if (sprite.getPosition().y + sprite.getGlobalBounds().height > WINDOW_HEIGHT - 62)
77          {
78             sprite.setPosition(sprite.getPosition().x, WINDOW_HEIGHT - 62 - sprite.getGlobalBounds().height + 5);
79          }
80      }
81      else
82      {
83          if (verticalSpeed < 13)
84              verticalSpeed += 2;
85          sprite.move(0, verticalSpeed);
86          if (life > 0 && sprite.getPosition().y >= 900)
87          {
88              sprite.setPosition(700, 100);
89              dead = 0;
90          }
91      }
92  }
```



the Fall method and its flowchart

```cpp
int Object::checkCollision(FloatRect otherSpriteBound)
{
    FloatRect spriteBound = sprite.getGlobalBounds();


    if (spriteBound.intersects(otherSpriteBound))
    {
        float spritetop = spriteBound.top;
        float spritecenterx = spriteBound.left + spriteBound.width / 2;
        float otherSpriteTop = otherSpriteBound.top;
        float otherspriteleft = otherSpriteBound.left;
        float otherspritecentery = otherSpriteBound.top + otherSpriteBound.height / 2;
        if (otherspritecentery > spritetop && otherspritecentery < spritetop + spriteBound.width) return 3;

        else if (spritetop > otherSpriteTop)
        {
            if (spriteBound.width > otherSpriteBound.width)
            {
                if (spritecenterx<otherspriteleft + spriteBound.width && spritecenterx > otherspriteleft) return 2;
                else return 3;
            }
            else
            {
                if (spritecenterx<otherspriteleft + otherSpriteBound.width && spritecenterx > otherspriteleft) return 2;
                else return 3;
            }
        }
        else return 1;
    }
    else return 0; //bu son olsun amk

}
```



the CheckCollision method and its flowchart

Mario Class

This class is a child class of Object class. It has 2 attributes and 2 methods,newGame and move, and a constructor different from the Object class.

The attributes are state and last heading which is used for the walking animation that performs by move method, other attribute is last heading which is used for the taking symmetry of the sprite.

constructor of this class initializes the attributes and sets the textures for the game. newGame() method is used for the second game the player will play. When the game ends these methods will call before the player goes to the menu, it basically does nearly the same things with the constructor, assigning appropriate values for the attributes. The move method is used for the horizontal movement of the mario. In this method the marios state changes

between 0-8 and every change the sprite text will change accordingly and the sprite will move if the player hits left or right keys and if the mario is alive. if it's not alive there won't be any horizontal move and the method will only set the appropriate texture for the dead mario. i add a state diagram for this method for better understanding

```
25  void Mario::newGame(void)
26  {
27      life = 3;
28      verticalSpeed = 10;
29      changedirection = 0;
30      lastheading = 1;
31      dead = 0;
32      state = 0;
33      sprite.setTexture(textures[state]);
34      sprite.setPosition(700, WINDOW_HEIGHT - 100);
35      sprite.setOrigin(33, 0);
36
37  }
38
```

newGame method

```
41  void Mario::move(WalkDirection dir)
42  {
43      if (!dead)
44      {
45          switch (state) {
46          case 0:
47              if (dir == WalkDirection::Left)
48              {
49                  state = 1;
50                  sprite.move(Vector2f(-horizontalSpeed, 0));
51
52                  if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
53                  else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
54                  if (!lastheading) sprite.scale(-1.f, 1.f);
55              }
56              else if (dir == WalkDirection::Right)
57              {
58                  sprite.move(Vector2f(horizontalSpeed, 0));
59
60                  if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
61                  else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
62                  state = 5;
63                  if(lastheading) sprite.scale(-1.f, 1.f);
64              }
65              break;
66          case 1:
67          {
68              if (dir == WalkDirection::Left)
69              {
70
71                  state = 2;
72                  sprite.move(Vector2f(-horizontalSpeed, 0));
73
74                  if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
75                  else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
76                  changedirection = 0;
77
78              }
79              else if (dir == WalkDirection::Right)
80              {
81                  sprite.move(Vector2f(horizontalSpeed, 0));
82
83                  if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
84                  else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
85                  state = 5;
86                  changedirection = 0;
87              }
88              else if (dir == WalkDirection::NONE)
89              {
90                  state = 0;
91                  lastheading = 1;
92              }
93              break;
```

```
        case 2:
        {
            if (dir == WalkDirection::Left)
            {
                sprite.move(Vector2f(-horizontalSpeed, 0));

                if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
                else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
                state = 3;
                changedirection = 0;
            }
            else if (dir == WalkDirection::Right)
            {
                sprite.move(Vector2f(horizontalSpeed, 0));

                if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
                else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
                state = 5;
                changedirection = 1;
            }
            else if (dir == WalkDirection::NONE)
            {
                state = 0;
                lastheading = 1;
            }
            break;

        }
        case 3:
        {
            if (dir == WalkDirection::Left)
            {
                sprite.move(Vector2f(-horizontalSpeed, 0));

                if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
                else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
                state = 4;
                changedirection = 0;
            }
            else if (dir == WalkDirection::Right)
            {
                sprite.move(Vector2f(horizontalSpeed, 0));

                if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
                else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
                state = 5;
                changedirection = 1;
            }
            else if (dir == WalkDirection::NONE)
            {
                state = 0;
                lastheading = 1;
            }
            break;
        }
```

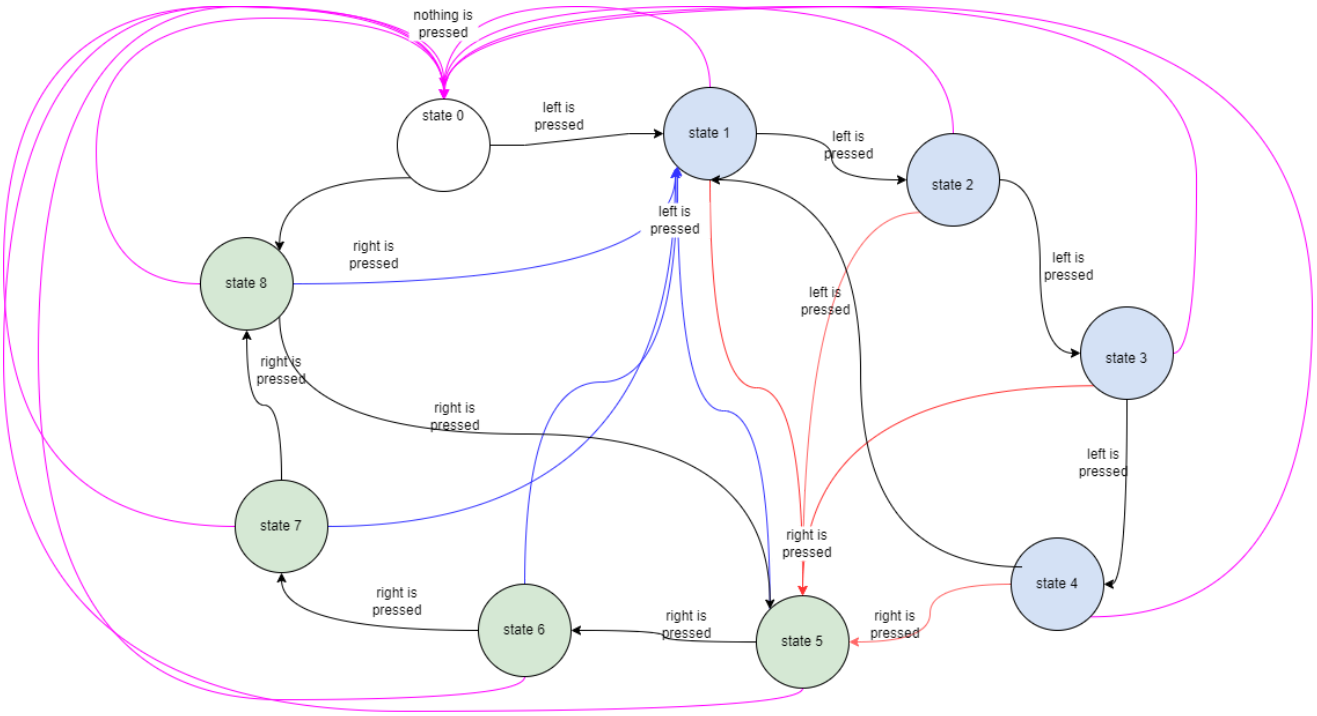
```
        case 4:
        {
            if (dir == WalkDirection::Left)
            {
                sprite.move(Vector2f(-horizontalSpeed, 0));

                if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
                else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
                state = 1;
                changedirection = 0;
            }
            else if (dir == WalkDirection::Right)
            {
                sprite.move(Vector2f(horizontalSpeed, 0));

                if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
                else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
                state = 5;
                changedirection = 1;
            }
            else if (dir == WalkDirection::NONE)
            {
                state = 0;
                lastheading = 1;
            }
            break;
        }
        case 5:
        {
            if (dir == WalkDirection::Left)
            {


                state = 1;
                sprite.move(Vector2f(-horizontalSpeed, 0));

                if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
                else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
                changedirection = 1;


            }
            else if (dir == WalkDirection::Right)
            {
                sprite.move(Vector2f(horizontalSpeed, 0));

                if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
                else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
                state = 6;
                changedirection = 0;
            }
            else if (dir == WalkDirection::NONE)
            {
                state = 0;
                lastheading = 0;
            }
            break;
        }
```

```
263         case 8:
264         {
265             if (dir == WalkDirection::Left)
266             {
267                 sprite.move(Vector2f(-horizontalSpeed, 0));
268 
269                 if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
270                 else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
271                 state = 1;
272                 changedirection = 1;
273             }
274             else if (dir == WalkDirection::Right)
275             {
276                 sprite.move(Vector2f(horizontalSpeed, 0));
277                 if (sprite.getPosition().x > WINDOW_WIDTH) sprite.setPosition(WINDOW_WIDTH, sprite.getPosition().y);
278                 else if (sprite.getPosition().x < 0) sprite.setPosition(5, sprite.getPosition().y);
279                 state = 5;
280                 changedirection = 0;
281             }
282             else if (dir == WalkDirection::NONE)
283             {
284                 state = 0;
285                 lastheading = 0;
286             }
287             break;
288         }
289         }
290 
291         if (state == 0) sprite.setTexture(textures[0]);
292         else if (state > 0 && state < 5) sprite.setTexture(textures[state]);
293         else if (state == 4) sprite.setTexture(textures[2]);
294         else if (state == 8) sprite.setTexture(textures[2]);
295         else if (state > 4 && state < 8) sprite.setTexture(textures[state - 4]);
296 
297 
298         if (changedirection)
299         {
300             sprite.scale(-1.f, 1.f);
301             changedirection = 0;
302         }
303     }
304     else
305     {
306     sprite.setTexture(textures[6]);
307     }
308 }
```

the move method.

in the code there is a case statement until line 290. After the case statement the sprites texture will be set accordingly and sprites symmetry will be taken according to changeddirection. other than these at line 304 the mario will be dead and dead texture will be set.

i add an if else statement to prevent that mario goes out of the screen. After the sprite had moved if its position is out of screen the if else statement set its position inside the screen.



state diagram for move method.

the blue states indicates that mario is going to the left and the green states indicates that mario is going right and whether the mario has changed his direction or not can be understood from red and blue lines. at blue and red line transitions changeddirection becomes one and the sprites symmetry will be taken. other than that magenta lines indicates that the player don't press any left or right key so at magenta transitions we save the lastheading of the mario. the textures will be change accordingly to the states

Turtle Class

The turtle class is a child class of object class to.it is similar to mario class. it has one attribute, 1 constructor and 3 methods other than object class attributes and methods.

The state attribute is used to obtain a walking animation during the game.

```
4    class Turtle :public Object {
5        int state;
```

attribute of the turtle class

the constructor initializes necessary attributes. The move and newGame methods are very similar to marios methods. The newgame method assigns reasonable values to necessary attributes and move method is used for horizontal move and and change the sprite texture accordingly. The fastenturtle method is used to fasten the turtle to make the game harder after Mario kills the turtle. again there is a state diagram for move method for better understanding.

```
21  void Turtle::newGame(void)
22  {
23      changedirection = 0;
24      life = 1;
25      horizontalSpeed = 5;
26      dead = 0;
27      state = 5;
28      sprite.setTexture(textures[state]);
29      sprite.setScale(-1, 1);
30      sprite.setPosition(5, 900);
31      sprite.setOrigin(33, 0);
32  }
33
34  void Turtle::fastenturtle()
35  {
36      horizontalSpeed = horizontalSpeed + 4;
37  }
38
```

newGame and fastenTurtle methods

```
39  void Turtle::move()
40  {
41      if (!dead)
42      {
43          switch (state) {
44          case 1:
45          {
46              if (sprite.getPosition().x < 1400)
47              {
48
49                  state = 2;
50                  sprite.move(Vector2f(horizontalSpeed, 0));
51                  changedirection = 0;
52
53              }
54              else
55              {
56                  if (sprite.getPosition().y == WINDOW_HEIGHT - 62 - sprite.getGlobalBounds().height + 5)
57                  {
58                      sprite.setPosition(5, WINDOW_HEIGHT - 62 - 74 - 5 - 660);
59                      sprite.move(Vector2f(horizontalSpeed, 0));
60                      changedirection = 0;
61                      state = 2;
62                  }
63                  else
64                  {
65                      sprite.move(Vector2f(-horizontalSpeed, 0));
66                      sprite.setPosition(1400, sprite.getPosition().y);
67                      state = 5;
68                      changedirection = 1;
69                  }
70              }
71              break;
72
73          }
74          case 2:
75          {
76              if (sprite.getPosition().x < 1400)
77              {
78
79                  state = 3;
80                  sprite.move(Vector2f(horizontalSpeed, 0));
81                  changedirection = 0;
82
83              }
84              else
85              {
86                  if (sprite.getPosition().y == WINDOW_HEIGHT - 62 - sprite.getGlobalBounds().height + 5)
87                  {
88                      sprite.setPosition(5, WINDOW_HEIGHT - 62 - 74 - 5 - 660);
89                      sprite.move(Vector2f(horizontalSpeed, 0));
90                      changedirection = 0;
91                      state = 3;
92                  }
93                  else
94                  {
95                      sprite.move(Vector2f(-horizontalSpeed, 0));
96                      sprite.setPosition(1400, sprite.getPosition().y);
97                      state = 5;
98                      changedirection = 1;
99                  }
100             }
101             break;
102         }
```

(field) int Object::horizontalSpeed

Search Online

```
        case 3:
        {
            if (sprite.getPosition().x < 1400)
            {

                state = 4;
                sprite.move(Vector2f(horizontalSpeed, 0));
                changedirection = 0;

            }
            else
            {
                if (sprite.getPosition().y == WINDOW_HEIGHT - 62 - sprite.getGlobalBounds().height + 5)
                {
                    sprite.setPosition(5, WINDOW_HEIGHT - 62 - 74 - 5 - 660);
                    sprite.move(Vector2f(horizontalSpeed, 0));
                    changedirection = 0;
                    state = 4;
                }
                else
                {
                    sprite.move(Vector2f(-horizontalSpeed, 0));
                    sprite.setPosition(1400, sprite.getPosition().y);
                    state = 5;
                    changedirection = 1;
                }
            }
            break;

        }
        case 4:
        {
            if (sprite.getPosition().x < 1400)
            {

                state = 1;
                sprite.move(Vector2f(horizontalSpeed, 0));
                changedirection = 0;

            }
            else
            {
                if (sprite.getPosition().y == WINDOW_HEIGHT - 62 - sprite.getGlobalBounds().height + 5)
                {
                    sprite.setPosition(5, WINDOW_HEIGHT - 62 - 74 - 5 - 660);
                    sprite.move(Vector2f(horizontalSpeed, 0));
                    changedirection = 0;
                    state = 2;
                }
                else
                {
                    sprite.move(Vector2f(-horizontalSpeed, 0));
                    sprite.setPosition(1400, sprite.getPosition().y);
                    state = 5;
                    changedirection = 1;
                }
            }
            break;
        }
```

```
        case 5:
        {
            if (sprite.getPosition().x > 0)
            {

                state = 6;
                sprite.move(Vector2f(-horizontalSpeed, 0));
                changedirection = 0;

            }
            else
            {
                if (sprite.getPosition().y == WINDOW_HEIGHT - 62 - sprite.getGlobalBounds().height + 5)
                {
                    sprite.setPosition(1395, WINDOW_HEIGHT - 62 - 74 - 5 - 660);
                    sprite.move(Vector2f(-horizontalSpeed, 0));
                    changedirection = 0;
                    state = 6;
                }
                else
                {
                    sprite.move(Vector2f(horizontalSpeed, 0));
                    sprite.setPosition(5, sprite.getPosition().y);
                    state = 1;
                    changedirection = 1;
                }
            }
            break;

        }
        case 6:
        {
            if (sprite.getPosition().x > 0)
            {

                state = 7;
                sprite.move(Vector2f(-horizontalSpeed, 0));
                changedirection = 0;

            }
            else
            {
                if (sprite.getPosition().y == WINDOW_HEIGHT - 62 - sprite.getGlobalBounds().height + 5)
                {
                    sprite.setPosition(1395, WINDOW_HEIGHT - 62 - 74 - 5 - 660);
                    sprite.move(Vector2f(-horizontalSpeed, 0));
                    changedirection = 0;
                    state = 7;
                }
                else
                {
                    sprite.move(Vector2f(horizontalSpeed, 0));
                    sprite.setPosition(5, sprite.getPosition().y);
                    state = 1;
                    changedirection = 1;
                }
            }
            break;

        }
```

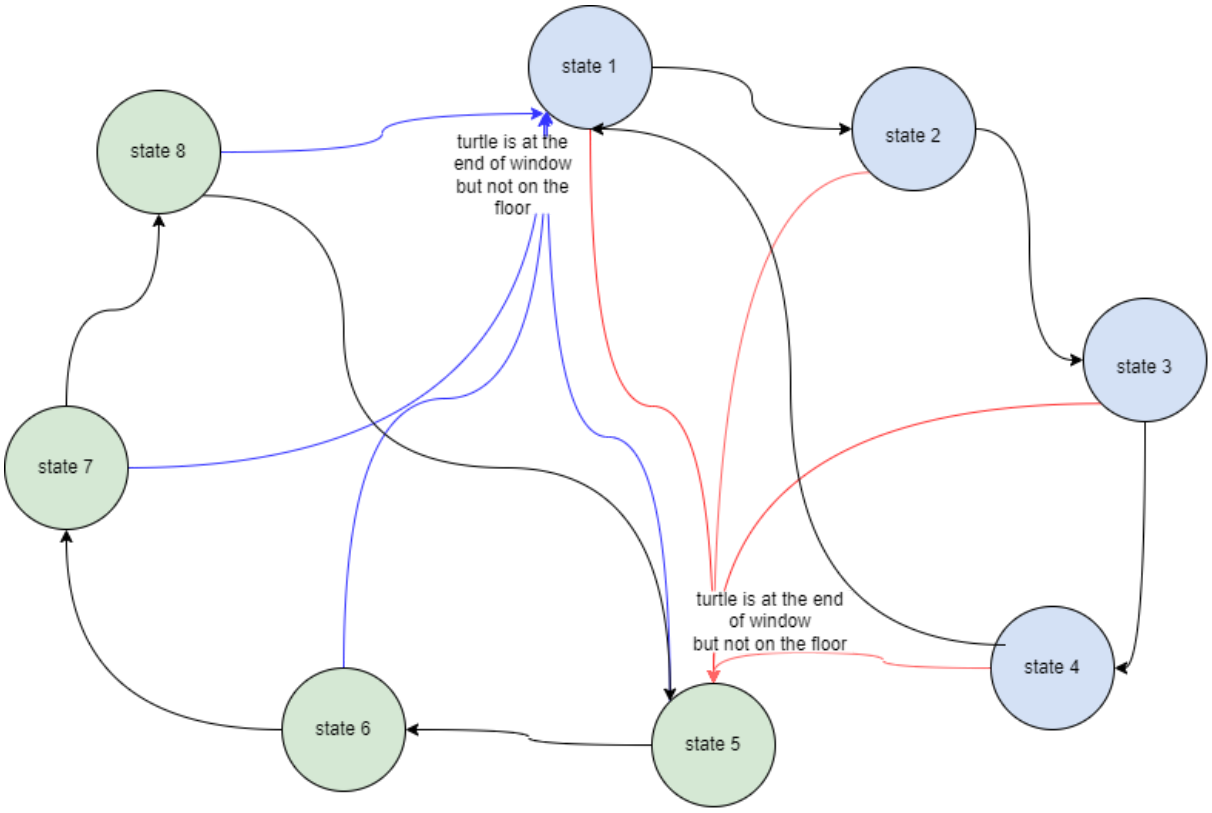
```
        case 7:
        {
            if (sprite.getPosition().x > 0)
            {

                state = 8;
                sprite.move(Vector2f(-horizontalSpeed, 0));
                changedirection = 0;

            }
            else
            {
                if (sprite.getPosition().y == WINDOW_HEIGHT - 62 - sprite.getGlobalBounds().height + 5)
                {
                    sprite.setPosition(1395, WINDOW_HEIGHT - 62 - 74 - 5 - 660);
                    sprite.move(Vector2f(-horizontalSpeed, 0));
                    changedirection = 0;
                    state = 8;
                }
                else
                {
                    sprite.move(Vector2f(horizontalSpeed, 0));
                    sprite.setPosition(5, sprite.getPosition().y);
                    state = 1;
                    changedirection = 1;
                }
            }
            break;

        }
        case 8:
        {
            if (sprite.getPosition().x > 0)
            {

                state = 5;
                sprite.move(Vector2f(-horizontalSpeed, 0));
                changedirection = 0;

            }
            else
            {
                if (sprite.getPosition().y == WINDOW_HEIGHT - 62 - sprite.getGlobalBounds().height + 5)
                {
                    sprite.setPosition(1395, WINDOW_HEIGHT - 62 - 74 - 5 - 660);
                    sprite.move(Vector2f(-horizontalSpeed, 0));
                    changedirection = 0;
                    state = 5;
                }
                else
                {
                    sprite.move(Vector2f(horizontalSpeed, 0));
                    sprite.setPosition(5, sprite.getPosition().y);
                    state = 1;
                    changedirection = 1;
                }
            }
            break;
        }
        }

        sprite.setOrigin(34, 0);
        if (state > 0 && state < 4) sprite.setTexture(textures[state]);
        else if (state == 4) sprite.setTexture(textures[2]);
        else if (state == 8) sprite.setTexture(textures[2]);
        else if (state > 4 && state < 8) sprite.setTexture(textures[state - 4]);


        if (changedirection)
        {
            sprite.scale(-1.f, 1.f);
        }
    }
    else
    {
    sprite.setTexture(textures[5]);
    }
}
```

move method of turtle class

The turtle move is just a little different from marios move. in marios move method the Mario changes its direction according to key presses but in turtles move the turtle changes its direction if it comes to the end of screen and on the bricks. if it's on the floor and comes to the end of the screen the method sets its position to the upper side of the screen with this we obtain a teleportation between pipes.

After the case statement the turtle's sprite texture will set accordingly and symmetry of the sprite will be taken if it changes its direction.

after line 295 the sprite for the dead turtle will set. so if turtle is not dead it moves horizontally and its sprite changes periodically, if its dead it doesn't move horizontally and only dead texture will be set



state diagram for turtles move method

The green states indicate the turtle is going right and the blue states indicate that the turtle is going left. The red and blue lines represent the turtle changing its direction due to the end of the window, so at these transitions we take the symmetry of the sprite. other than that the states will make a loop.

Game Class

This class is the main class for the game. It has sprites and textures for the background, a window, a scoreboard, five turtles, one mario and other attributes to use in the game update method.

other than its constructor this class has 4 methods. The game will start with the menu method which is the method that sets the menu background and waits for enter key press to start the game. The setBackground method is used for setting the texture of background objects sprites and placing them. The onFloor method is used to determine if the object is on the floor or not. This method uses the object class' checkColision method with every turtle and mario between every brick and floor. The update method is a little long so I prepared a line diagram for this method.

```
90
91  void Game::menu(void)
92  {
93      generalText.setString("Press Enter to Start");
94      generalText.setCharacterSize(24);
95      generalText.setPosition(0, 0);
96
97      while (window->isOpen())
98      {
99
100         Event event;
101         while (window->pollEvent(event))
102         {
103
104
105             if (event.type == Event::Closed)
106                 window->close();
107             else if (event.type == Event::KeyPressed)
108             {
109                 if (Keyboard::isKeyPressed(Keyboard::Enter))
110                     update();
111             }
112         }
113         window->draw(startSprite);
114         window->draw(generalText);
115         window->display();
116     }
117 }
118
```

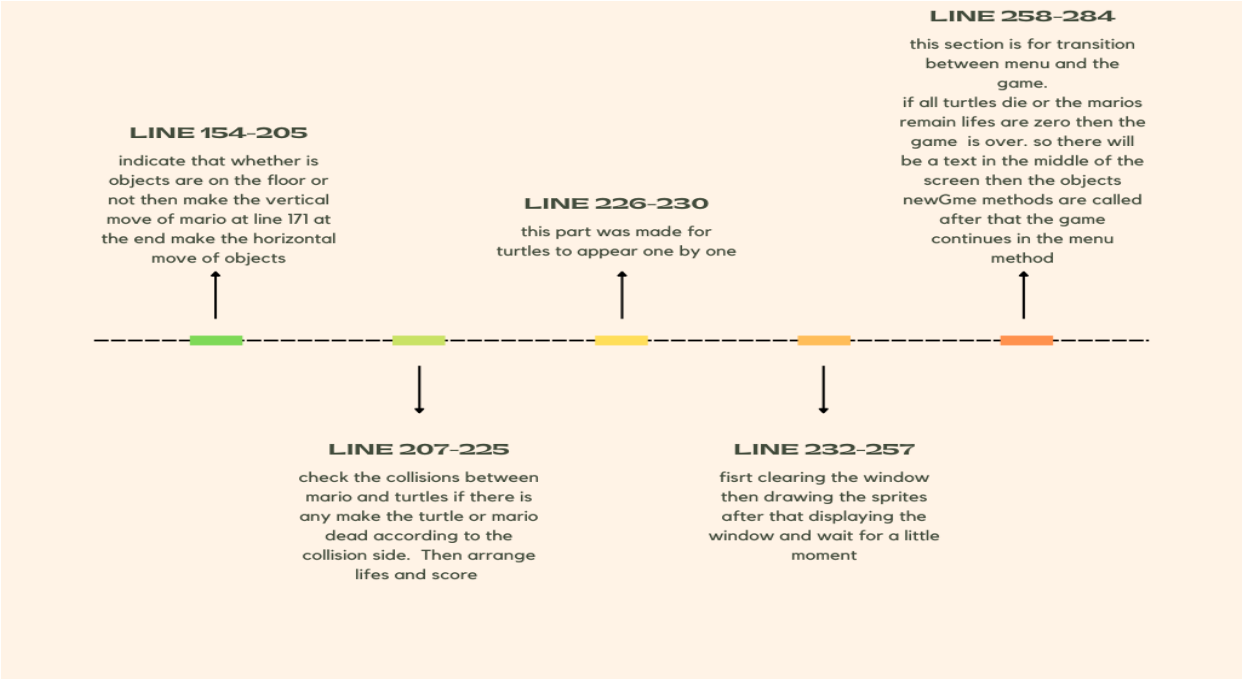
menu method

```
119 int Game::onFloor(Object* object)
120 {
121     int objectbricktouch;
122     for (int i = 1; i < 87; i++)
123     {
124         int objectbricktouch1 = object->checkCollision(brickSprite[i - 1].getGlobalBounds());
125         int objectbricktouch2 = object->checkCollision(brickSprite[i].getGlobalBounds());
126         int objectbricktouch3 = object->checkCollision(brickSprite[i + 1].getGlobalBounds());
127
128         if (objectbricktouch1 == 1 || objectbricktouch2 == 1 || objectbricktouch3 == 1)
129         {
130             objectbricktouch = 1;
131             break;
132         }
133
134         else if (objectbricktouch1 == 2 || objectbricktouch2 == 2 || objectbricktouch3 == 2)
135         {
136             objectbricktouch = 2;
137             break;
138         }
139
140         else if (objectbricktouch1 == 3 || objectbricktouch2 == 3 || objectbricktouch3 == 3)
141         {
142             objectbricktouch = 3;
143             break;
144         }
145         else objectbricktouch = 0;
146     }
147
148     int objectfloortouch = object->checkCollision(floorSprite.getGlobalBounds());
149     if (objectfloortouch == 0) return objectbricktouch;
150     else return 1;
151 }
```

onFloor method

```cpp
void Game::setBackground(RenderWindow*window)
{
    floorTexture.loadFromFile("../assets/floor.png");
    floorSprite.setTexture(floorTexture);
    floorSprite.scale(1.5f, 1);
    floorSprite.setPosition(0, WINDOW_HEIGHT - 62);

    pipeTexture.loadFromFile("../assets/pipe.png");
    pipeSprite.setTexture(pipeTexture);
    pipeSprite.setPosition(WINDOW_WIDTH - 130, WINDOW_HEIGHT - 62 - 74 - 5);

    pipe1Sprite.setTexture(pipeTexture);
    pipe1Sprite.scale(-1.f, 1.f);//simetrik alma
    pipe1Sprite.setPosition(130, WINDOW_HEIGHT - 62 - 74 - 5);


    brickTexture.loadFromFile("../assets/brick.png");
    for (int i = 0; i < 15; i++)
    {
        brickSprite[i].setTexture(brickTexture);
        brickSprite[i].setPosition(0 + 30 * i, WINDOW_HEIGHT - 62 - 74 - 5 - 100);
    }

    for (int i = 15; i < 30; i++)
    {
        brickSprite[i].setTexture(brickTexture);
        brickSprite[i].setPosition(1400 - 30 * (i - 15), WINDOW_HEIGHT - 62 - 74 - 5 - 100);
    }

    for (int i = 30; i < 50; i++)
    {
        brickSprite[i].setTexture(brickTexture);
        brickSprite[i].setPosition(400 + 30 * (i - 30), WINDOW_HEIGHT - 62 - 74 - 5 - 100 - 225);
    }

    for (int i = 50; i < 65; i++)
    {
        brickSprite[i].setTexture(brickTexture);
        brickSprite[i].setPosition(0 + 30 * (i - 50), WINDOW_HEIGHT - 62 - 74 - 5 - 550);
    }

    for (int i = 65; i < 80; i++)
    {
        brickSprite[i].setTexture(brickTexture);
        brickSprite[i].setPosition(1400 - 30 * (i - 65), WINDOW_HEIGHT - 62 - 74 - 5 - 550);
    }

    for (int i = 80; i < 84; i++)
    {
        brickSprite[i].setTexture(brickTexture);
        brickSprite[i].setPosition(1400 - 30 * (i - 80), WINDOW_HEIGHT - 62 - 74 - 5 - 240);
    }

    for (int i = 84; i < 88; i++)
    {
        brickSprite[i].setTexture(brickTexture);
        brickSprite[i].setPosition(0 + 30 * (i - 84), WINDOW_HEIGHT - 62 - 74 - 5 - 240);
    }

    spipeTexture.loadFromFile("../assets/pipeS.png");
    spipeSprite.setTexture(spipeTexture);
    spipeSprite.setPosition(WINDOW_WIDTH - 130, WINDOW_HEIGHT - 62 - 74 - 5 - 660);

    spipe1Sprite.setTexture(spipeTexture);
    spipe1Sprite.scale(-1.f, 1.f);//simetrik alma
    spipe1Sprite.setPosition(130, WINDOW_HEIGHT - 62 - 74 - 5 - 660);
}

```

setBackground method



line diagram for update method

```
152  void Game::update(void)
153  {
154      while (window->isOpen())
155      {
156
157          for (int j = 0; j < 5; j++)
158          {
159              turtle[j].fall(onFloor(&turtle[j]));
160          }
161          mario.fall(onFloor(&mario));
162
163          Event event;
164          while (window->pollEvent(event))
165          {
166              if (event.type == Event::Closed)
167                  window->close();
168              else if (event.type == Event::KeyPressed)
169              {
170                  if (Keyboard::isKeyPressed(Keyboard::Up) && onFloor(&mario)==1)
171                      mario.jump();
172                  else if (event.key.code == Keyboard::Right)
173                  {
174
175                      rightisPressed = 1;
176                      leftisPressed = 0;
177                  }
178
179                  else if (event.key.code == Keyboard::Left)
180                  {
181                      leftisPressed = 1;
182                      rightisPressed = 0;
183                  }
184                  if (event.key.code == Keyboard::Escape)
185                      window->close();
186              }
187              else
188              {
189                  leftisPressed = 0;
190                  rightisPressed = 0;
191              }
192          }
193
194
195          if (leftisPressed == 1)
196              mario.move(Mario::WalkDirection::Left);
197          else if (rightisPressed == 1)
198              mario.move(Mario::WalkDirection::Right);
199          else mario.move(Mario::WalkDirection::NONE);
200
201          for (int i = 0; i < 5; i++)
202          {
203              turtle[i].move();
204          }
```

1st part of update method, line 154-205

```
206
207         for (int i = 0; i < 5; i++)
208         {
209             if (mario.checkCollision(turtle[i].boundingBox()) == 1 && !mario.getDead() && !turtle[i].getDead())
210             {
211                 turtle[i].jump();
212                 turtle[i].makeDead();
213                 for (int j = 0; j < 5; j++)
214                 {
215                     turtle[j].fastenturtle();
216                 }
217                 score += 100;
218             }
219             else if (mario.checkCollision(turtle[i].boundingBox()) == 3 && !turtle[i].getDead() && !mario.getDead())
220             {
221                 mario.jump();
222                 mario.makeDead();
223             }
224         }
225
```

2nd part of update method, line 207-225

```
226         if (forturtles == 100 || forturtles == 200 || forturtles == 300 || forturtles == 400 || forturtles == 0)
227         {
228             turtle[forturtles/100].makeAlive();
229         }
```

3rd part of update method, line 226-230

```
231
232         window->clear();
233
234         for (int i = 0; i < 88; i++)
235         {
236             window->draw(brickSprite[i]);
237         }
238
239         window->draw(floorSprite);
240         window->draw(pipeSprite);
241         window->draw(pipe1Sprite);
242         window->draw(spipeSprite);
243         window->draw(spipe1Sprite);
244         scoreBoard.setScore(score, window);
245         scoreBoard.setLives(mario.getLife(),window);
246         mario.draw(window);
247         for (int i = 0; i < 5; i++)
248         {
249             turtle[i].draw(window);
250         }
251
252         window->display();
253
254         if (forturtles<1000) forturtles++;
255         sf::sleep(sf::milliseconds(1000 /40));
256
```

4th part of update method, line 232-257

```
257
258     if (mario.getLife() == 0 || score ==500)
259     {
260         if (mario.getLife() == 0)
261         {
262
263             generalText.setString("You Lost :(");
264             generalText.setPosition(WINDOW_WIDTH / 2-100, WINDOW_HEIGHT / 2);
265             window->draw(generalText);
266             window->display();
267         }
268         else
269         {
270             generalText.setString("You Won <3");
271             generalText.setPosition(WINDOW_WIDTH / 2-100, WINDOW_HEIGHT / 2);
272             window->draw(generalText);
273             window->display();
274         }
275         forturtles = 0;
276         mario.newGame();
277         score = 0;
278         for (int i = 0; i < 5; i++)
279         {
280             turtle[i].newGame();
281             turtle[i].makeDead();
282         }
283         sf::sleep(sf::milliseconds(5000));
284         menu();
285     }
```

5th part of update method line 258-284

ScoreBoard class

This class is implemented to display the score and remaining lives of mario it has 5 attributes: headTexture and headSprite to display remaining lives of mario; font, texture, score.

```
Texture headTexture;
Sprite headSprite;
Font font;
Text text;
string  score;
```

Attributes of ScoreBoard Class

it has 2 methods and one constructor. The constructor loads the head texture and sets it to headSprite, other than that it loads and sets font. The SetScore method converts the integer score to a text score then positions it after that displays it. setLives method positions and draws up the headSprites to remaining lives with a simple for loop.

```
#include "ScoreBoard.h"

ScoreBoard::ScoreBoard()
{
    score = "000000";
    font.loadFromFile("../assets/font.ttf");
    headTexture.loadFromFile("../assets/mario7.png");
    headSprite.setTexture(headTexture);
    this->text.setFont(font);
}

void ScoreBoard::setScore(int score, RenderWindow* window)
{
    this->score = to_string(score);
    this->score = std::string(6 - this->score.length(), '0') + this->score;
    text.setString(this->score);
    text.setCharacterSize(42);
    text.setPosition(0, 0);
window->draw(text);
}

void ScoreBoard::setLives(int lives, RenderWindow* window)
{

    for (int i = 0; i < lives; i++)
    {
        headSprite.setPosition(0 + i * headSprite.getGlobalBounds().width, 50);
        window->draw(headSprite);
    }
}

```

methods of ScoreBoard Class

Discussion

The biggest problem that I faced was that ı did the project we-rong and i didn't have time to fix all of it because of other projects and exams. But i learn my lesson that i should read the documents more carefully when i do something all by myself

Another problem I faced was writing the checkCollision method because sometimes the first sprite is bigger and sometimes the other i spent much more time on this method. I couldn't delay this method because I am using this method to determine whether the sprites are on the floor or not. I noticed my mistake when I was sketching the situations in this method. So I decided İ should do the sketching first for visual things like this.

In the end I recognize that I improved in object oriented programming more than I expected. At first I didn't even think i could do this project with a group but i did it all by myself even though it's not perfect and has some mistakes :)

https://www.youtube.com/watch?v=hmCiKvzNkkw