Ping - pong game „Heart-pong!”

Oliwia Mlonek

Computer Programming
Semester Project
Teacher: dr inż. Krzysztof Pasterak

Wydział Automatyki, Elektroniki i Informatyki
Politechnika Śląska
Polska
25.01.2020

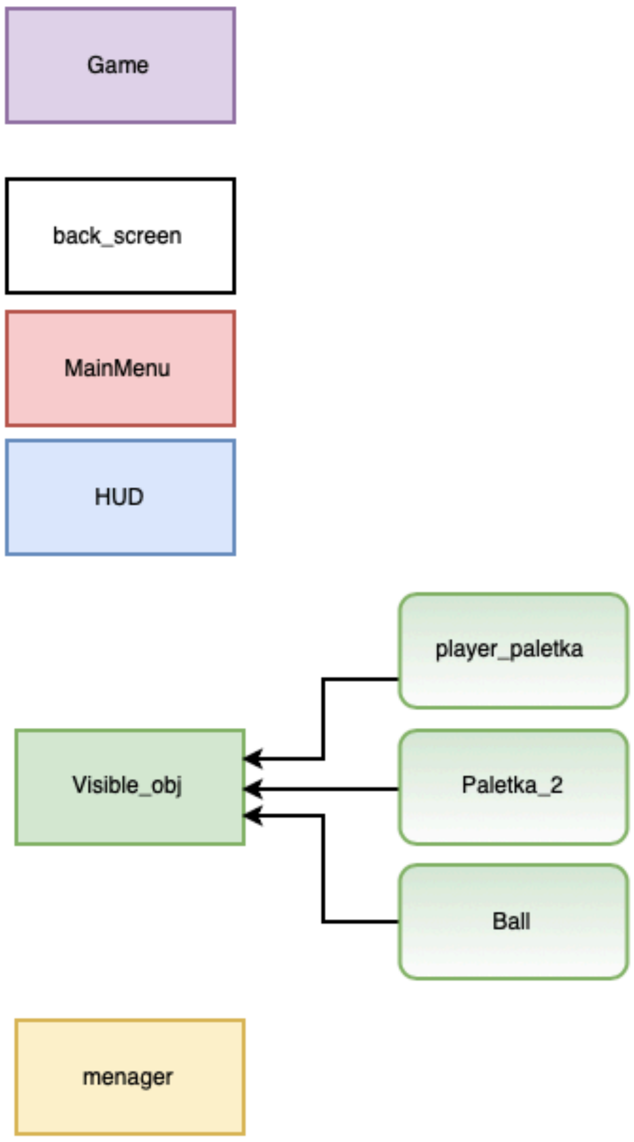
1 Topic

The main goal of the project was to get familiar with using object oriented features and mechanisms of the C++ programming language. My choice was to create a game with fairly simple logic, but allowing the use of various object-oriented programming tools. I decided to build ping-pong game, with ‘false’ artificial intelligence that controls the second paddle. One of the player's goals is to prevent the ball from falling down the screen, in other words the ball cannot bypass the paddle that the player controls. If user fails, AI scores. The second objective is to make the ball speed so fast that the AI-controlled paddle will not be able to bounce it back, thus the ball will fly to the top edge of the screen and the player will score a point.

2 Analysis and Development

The first decision in the design process was to choose a library that would provide interfaces for displaying a window and rendering objects to that window. I decided the program will use SFML library, since it had very clear and detailed documentation, which was easy to use.

After that a preliminary picture of classes and their connections was created, which, in the process of creation, ended in such form:



2.1 Data structures

The data structures used in the project are as follows:

- **std::list<MenuItem> _menuItems** holds the various MenuItems that compose MainMenu.
- **std::map<std::string, visible_obj*> _gameObjects** is composed of a collection of std::pair<> objects.

2.2 Algorithms

2.2.1 Physics

One of the main challenges in creating the game was to write an algorithm imitating the laws of physics to obtain a reliable flight path and bounce the ball in the game.

Firstly I figured out the amount I want the ball to move. For example ,elapsed time is 1/5th a second (0.2) and given that our velocity is 230 pixels per second, multiplying velocity by 0.2 will give up 46 pixels, meaning that to move 230 pixels in a second, in this update I want to move 46 pixels. After that, it is important to determine where the ball should move. By multiplying the result of linear velocity of X axis and Y axis by our current velocity, I get how much I want to move in the X and Y directory this frame.

```
  float moveAmount = _velocity  * elapsedTime;
  float moveByX = LinearVelocityX(_angle) * moveAmount;
  float moveByY = LinearVelocityY(_angle) * moveAmount;
```

However, before the ball is actually moved, I check to see if the movement is going to cause a collision of some kind. If the sprite position was going to go off the side of the screen on either the left or right, the ricochet off the side by inverting the angle. For example, if the sprite was going to hit the right side of the screen at 90 degrees, it will ricochet out at 270 degrees. If the ball hits either side at +- 10 degrees to straight on, 20 degrees is added to the resulting angle to prevent all slowly ricochet's left and right over and over. Also, the sign on the X move amount is flipped, so if it was going left (–) it is now going right (+) and vice versa.

```
  if(GetPosition().x + moveByX <= 0 + GetWidth()/2 || GetPosition().x + GetHeight()/2 +
moveByX >= Game::SCREEN_WIDTH)
   {

       _angle = 360.0f - _angle;
       if(_angle > 260.0f && _angle < 280.0f) _angle += 20.0f;
       if(_angle > 80.0f && _angle < 100.0f) _angle += 20.0f;
       moveByX = -moveByX;
   }
```

After that I want to check if the GameBall is about to collide with the player's paddle. I do this by getting the bounding box (a sfml rect) of the player_paletka and testing if it intersects the ball's bounding rectangle. Once I determine that a collision occurred, I need to decide how the ball will ricochet. I calculate the angle the ball will bounce out as we invert the Y direction.

I also add certain amount of "spin" on collision, based on how the player is moving at the time of the collision. If the player is moving to the left (negative velocity) 20 degrees is subtracted from the ricochet angle, while if the player is moving right (positive velocity) 20 degrees is added to the ricochet angle. It entails a little variety to the game and gives the player a bit more control over how the ball rebounds.

```
player_paletka* player1 = dynamic_cast<player_paletka*>(Game::GetGameObjectManager().Get("Paddle1"));
  if(player1 != NULL)
  {
    sf::Rect<float> p1BB = player1->GetBoundingRect();

    if(p1BB.intersects(GetBoundingRect()))
    {
      _angle =  360.0f - (_angle - 180.0f);
      if(_angle > 360.0f) _angle -= 360.0f;



      moveByY = -moveByY;


      if(GetBoundingRect().width > player1->GetBoundingRect().top)
      {
        set_position(GetPosition().x,player1->GetBoundingRect().top - GetWidth()/2 -1 );
      }


      float playerVelocity = player1->GetVelocity();

      if(playerVelocity < 0)
      {

        _angle -= 20.0f;
        if(_angle < 0 ) _angle = 360.0f - _angle;
      }
      else if(playerVelocity > 0)
      {
        _angle += 20.0f;
        if(_angle > 360.0f) _angle = _angle - 360.0f;
      }


      _velocity += 5.0f;
    }

    if(GetPosition().y - GetHeight()/2 <= 0)
    {
      _angle =  180 - _angle;
      moveByY = -moveByY;
    }
```

In addition, another important part is to notice a collision with the top and bottom edges of the screen. It is necessary to calculate the result of the game, i.e. which player did not manage to return the ball and lost the game. If the ball gets to the top or bottom of the screen, it simply bounce it back by inverting the angle and direction of the ball. In the case however the ball get's past the any player paddle, the ball moves back to the middle of the screen, restart the 3 second count down then send the ball back out at a random angle slowed back down to 220 pixels per second.

For real player (player1):

```
  if(GetPosition().y + GetHeight()/2 + moveByY >= Game::SCREEN_HEIGHT)
      {
        GetSprite().setPosition(Game::SCREEN_WIDTH/2, Game::SCREEN_HEIGHT/2);
        _angle = (rand()%360)+1;
        _velocity = 230.0f;
        _elapsedTimeSinceStart = 0.0f;
         player1->score += 1;
      }
```

and for AI player (player2):

```
   if(( GetPosition().y - GetHeight() / 2 - moveByY <= 0))
      {

          GetSprite().setPosition(Game::SCREEN_WIDTH/2, Game::SCREEN_HEIGHT/2);
          _angle = (rand()%360)+1;
          _velocity = 230.0f;
          _elapsedTimeSinceStart = 0.0f;
          player2->score += 1;
      }
```

It may seem that the way I add points to individual players is mistaken (the point goes to the loser) but this is a planned operation, caused by the construction of the class and access to pointers. The result display is correct, since the scores in text that shows the results is also swapped:

```
   text.setString("UNICORN: "+std::to_string(player->score)+ "\n\t" + " YOU: "+
std::to_string(player2->score));
```

2.2.2 Fake AI

Another challenge was creating the impression that the other player, in the form of artificial intelligence, controls the second pad at the top of the game screen. The logic behind it is very simple. The position of the paddle always follows the position of the ball.

```
   const Ball* gameBall = static_cast<Ball*>
   (Game::GetGameObjectManager().Get("Ball"));
   sf::Vector2f ballPosition = gameBall->GetPosition();

if(GetPosition().x  < ballPosition.x)
      _velocity += 100.0f;

   else if(GetPosition().x > ballPosition.x)
      _velocity -= 100.0f;

   if(gameBall->_velocity==0.0f){
      GetSprite().setPosition((Game::SCREEN_WIDTH/2),40);
      _velocity = 0.0f;
   }
```

```
if(_velocity > _maxVelocity)
    _velocity = _maxVelocity;

if(_velocity < -_maxVelocity)
    _velocity = -_maxVelocity;
```

3 External specification

User’s manual.

3.1 Controls

For controlling the paddle, the player can use the left and right arrow keys to move and spacebar to stop the movement

3.2 Gameplay

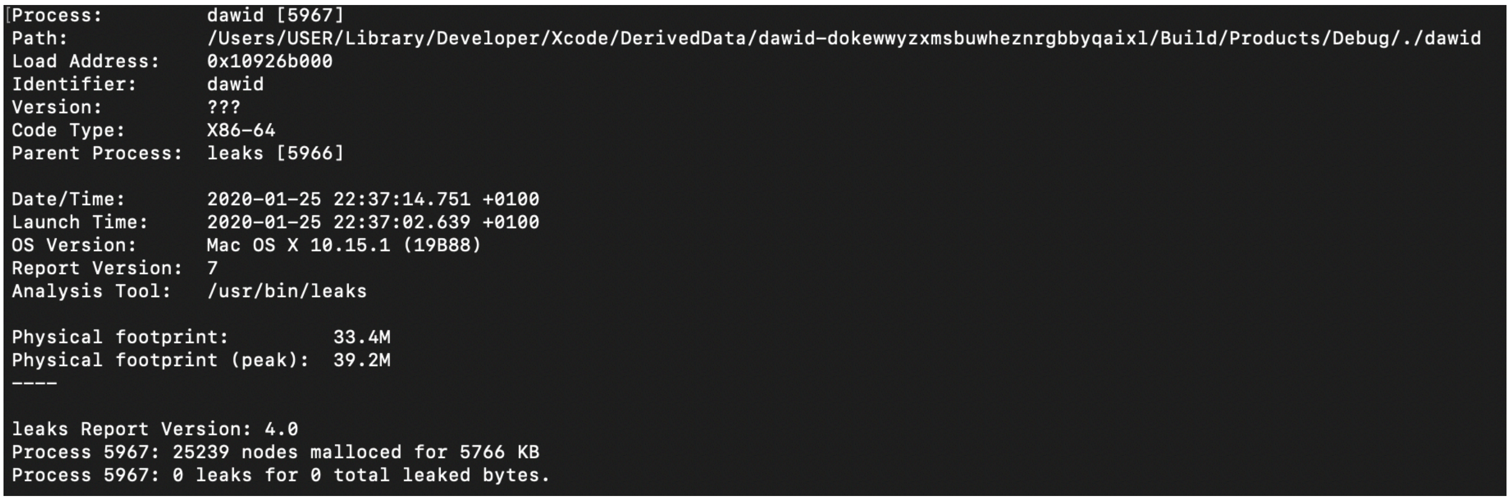
The main objective of the game is: once, bouncing the ball with the paddle as long as possible and not letting it fall down the screen, two, bounce it in such a way and add it to such speed that the pad controlled by AI does not hit it. Then the player wins the game.

4 Internal specification

The program is implemented according to the object oriented paradigm. Technical documentation in the doxygen style is attached to this report in a separate file (refman.pdf).

5 Memory leaks

The program was tested for memory leaks and no leaks were found. The built-in system leaks utility was used for the test.



```
[Process:         dawid [5967]
Path:            /Users/USER/Library/Developer/Xcode/DerivedData/dawid-dokewwyzxmsbuwheznrgbbyqaixl/Build/Products/Debug/./dawid
Load Address:    0x10926b000
Identifier:      dawid
Version:         ???
Code Type:       X86-64
Parent Process:  leaks [5966]

Date/Time:       2020-01-25 22:37:14.751 +0100
Launch Time:     2020-01-25 22:37:02.639 +0100
OS Version:      Mac OS X 10.15.1 (19B88)
Report Version:  7
Analysis Tool:   /usr/bin/leaks

Physical footprint:         33.4M
Physical footprint (peak):  39.2M
----

leaks Report Version: 4.0
Process 5967: 25239 nodes malloced for 5766 KB
Process 5967: 0 leaks for 0 total leaked bytes.
```

It was built and run on the macOS Catalina (10.15.1) operating system.

6 Conclusions

Writing a program using the object oriented paradigm for the first time is a really difficult task, but it really improves logical and abstract thinking about our code. Moreover, thanks to such project we are able to get familiar with graphic libraries, which are a very useful tool when creating any games, but not only. Games of this type are a challenge on the mathematical and physical side, which is not always easy, but when you achieve the intended effect, it is very satisfying.