Nathan LEMALE 7002827

Bowling simulator

1. Ideas

   The purpose of this class was to create a simulation tool. Then, in the end, use the whole environment to create a game or concept.

   In the end we only had squares, rectangles, and spheres available to build a game. Because I wanted to use both rectangle / square and sphere shape, my idea was to create a bowling simulation. This way, it could show both ball physics, kneels as rectangles physics, and physics interactions between every element themselves and the ground.

2. Features

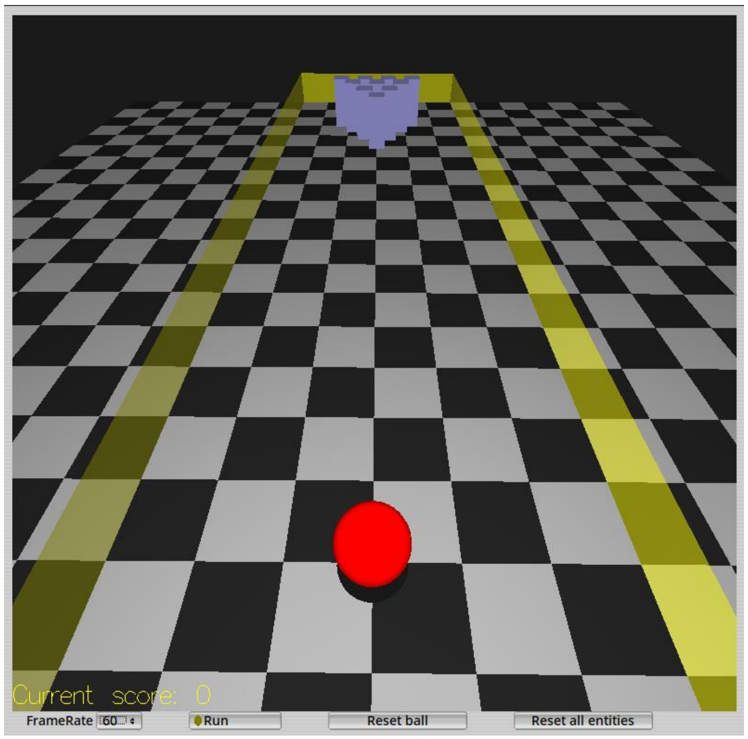
   As said previously, the four main elements of a bowling game are:

   - A bowling ball
   - A set of 10 kneels
   - A terrain
   - A score

<u>The terrain:</u>

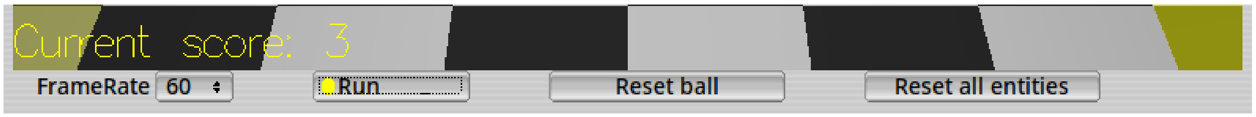
As we can see in the picture here the ball is on the bottom close to the player. Kneels are far away in the back and yellow planes are used to limit the size of the playable area.

The player can only grab the ball on the screen and throw it within the playground.



<u>User interface:</u>

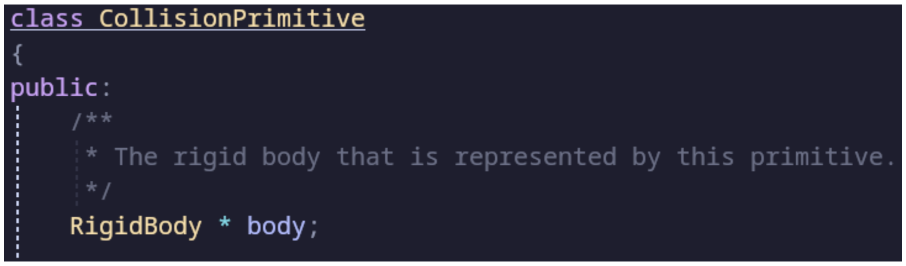
The player has multiple information and actions available on the window.



To start any gameplay, the player must press the "***RUN***" buttons. A score is displayed on the bottom left corner of the screen.
The score system is straight forward. The more keels are down, the more points the player has. This goes up to a maximum of 10 points. To throw the ball multiple times the player can press the buttons on the bottom of the window.
The 2 actions available are resetting the ball position and resetting both keels and ball position. Those actions are split due to bowling rules where the player can throw a ball multiple times with the same set of keels. Until the player reset every position the score will not be reset.

3. Implementation
The implementation was a hard challenge for 2 reasons. First, the library used to simulate physics is old and not used at all nowadays. Therefore, finding reliable documentation was not always a simple thing. The second point is that I could only work on Linux during the class, so I needed to recreate a big part of the library by myself because it does not support Linux core system. This is why I must set the framerate to 1920 fps because with the basic 60 fps given, I only had 1 image per second. This also implies physics calculation between elements.

For this project, the first step I took was creating entity management through a main physics calculation. On the first frame of the program, every kneel and the ball are created within a vector for each. This lets me manage them with efficiency.
Both kneels and ball are inherited from a class called "CollisionPrimitive" which creates a rigid body used for physics update such as collision, gravity, position, and all the entity physics element related.



```
class CollisionPrimitive
{
public:
    /**
     * The rigid body that is represented by this primitive.
     */
    RigidBody * body;
```

Then on every frame of the project this body and its characteristics must be updated, and collisions must be checked to display the correct behavior on screen. This is done in two steps, first we are generating the current contacts between entities and the ground if some exist and calculate the correct behavior to execute regarding the current element's position, velocity, acceleration… Then the object has every information required about where its characteristics for the next frame, so we apply them.

And the final general physics element I implemented is the sided planes. Those are used to determine the playable range. Any element interacting with those is stopped and will bounce off. This is implemented the same way as the ground location and size is created.

4. What I learn

During this class, the biggest thing I learnt was about OpenGL. Since I never used it before it was a discovery for me. Even if I cannot use this in a current project because as I said the tool has not been updated anymore for years now. It has given me the basics of what physics creation could look like. I also discovered Unity for the first time this year, therefore it has helped me to have a better understanding of how the most common properties like rigid body and collision detection can work together.
Since I did a lot of C++ before this class, I do not think this is something I improved during this class but other concepts like quaternions management and usage were very new to me. Also, some physical properties between objects in a more mathematical perspective were not familiar to me since I did not do advanced math classes.