2D Physics OpenGL

Explanation of how it works:

For my assignment, I have coded several classes for the behaviour of my 2D physics. I have also coded classes that will help me debug and organize my code.

The first class that I have is Clocker. It is a static singleton class for keeping track of the deltaTime and having a consistent deltaTime during the game's main loop. I also have a static class that will help me to convert degrees to Radians.

The following five classes I will mention are the core of my whole system. First of all, I have the SDWINDOW class; this is the class of the game that will handle the updates, render the events, and create all the instances of my objects.

The other class is the Transform. All the objects will inherit from the transform class. The Transform keep track of the position, the vertices, the up and right vector, the rotation, and a physics pointer. Hence, the SDWINDOW will store a list of shared pointers of Transform type so that I can handle all the object updates and renders in this game class.

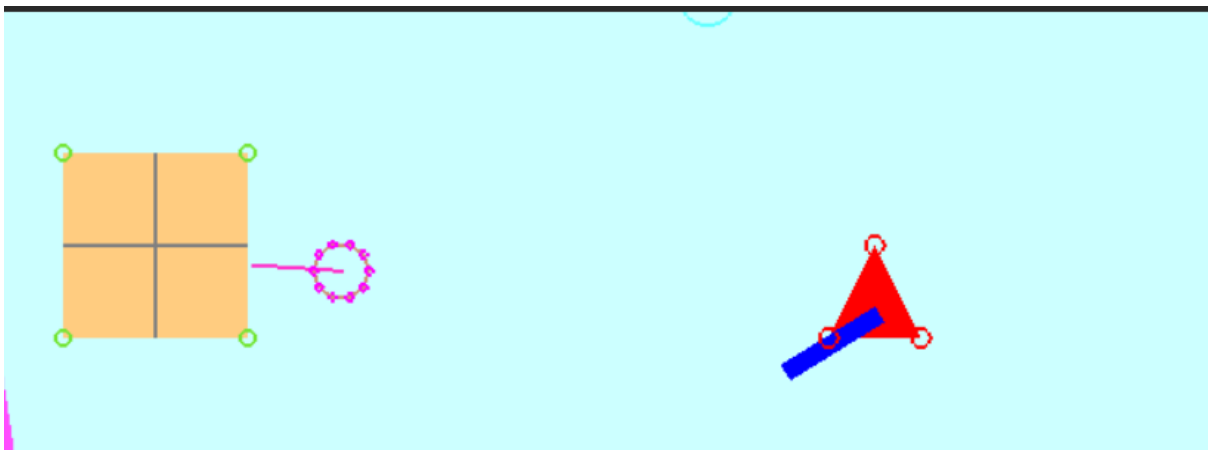
Physics2D class is one of the most essential classes since it is the one that will detect the collisions and trigger the collision response on each object individually. On the SDWINDOW inside the update function, I am calling to the physics class that will loop through all the objects and check for collisions.

The next class is the Vector2D, which handles all types of Vectors formulas. However, this class has a problem since it can modify vector objects that we don't want to change; this happens due to the memory management of the pointers. However, in my system, I fix all of the errors that I had with the Vector2D, but if you want to use it, try to use the static function rather than the standard functions or the operator overloads since it can modify undesired vector references that we don't want to change.

The final class, and one of the most important, is the gizmos class. Similar to Unity. This class helped me to debug my system visually. It can draw points, rays and lines. Hence, I used for visualizing my vertices, motion vectors and normal collision points. It also helps me to visualize the up and right vector (basis vector) of the objects, which is active on the player object. Without this class, it would have been complicated to develop my assignment. So, I coded a gizmos class similar to the one of Unity, where I can specify the start position and the direction or end position. The gizmo is a class that is not attached to any other object. So, it has its own render functions.

I will now explain the algorithm that I use for detecting collisions. I use the algorithm for detecting collision with the deltaTime, the phit and thit. Instead of detecting collision with the object itself I raycast a line. This will be the current position for checking collisions. You can find this algorithm in my physics class. However, it had unexpected behaviours, so I coded another algorithm which helped me to detect the collision of multiple objects. And don't depend on a raycast line.

The following picture is from my game, where I used the algorithm of phit and thit.

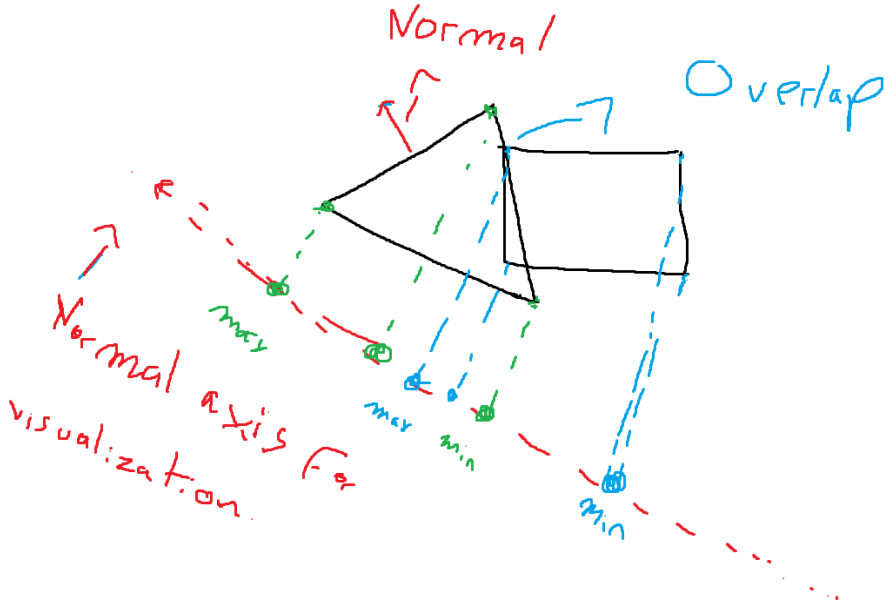


Separate Axis theorem

This algorithm will select an object and compare it with another object. On the first object, it will get the normal axis of each edge. Then, it will loop through all the vertices of itself and the other entity and project the vertex with the dot product on the normal axis; it will repeat this on all the normal of the object. After this, it will get all the normal axes of the other object and repeat the same.

It will always keep track of the maximum and minimum projected vertices on the normal. If it finds a separation, it means that the projected maximum and minimum vertices didn't overlap. This would mean that there was no collision. But if it didn't detect separation, there was an overlap, thus, we detected collision.

Picture of the algorithm.



In the picture above is a demonstration of the algorithm. The normal axis is drawn below just for visualization purposes. We get each vertex's projected max and min and check for overlaps.

The last step is for collision response, in the case of the normal objects. I get the centre of the object, and find the direction of the object. Hence, with the dot product, we can get the correct direction of the normal and use this direction to move the object away.

In the case of the bullet, I move it away and add a reflection so the bullet reflects and doesn't overlap with the object. I use the vector reflection formula to reflect the bullet. And I added to the normal' s direction to don't overlap with the object.

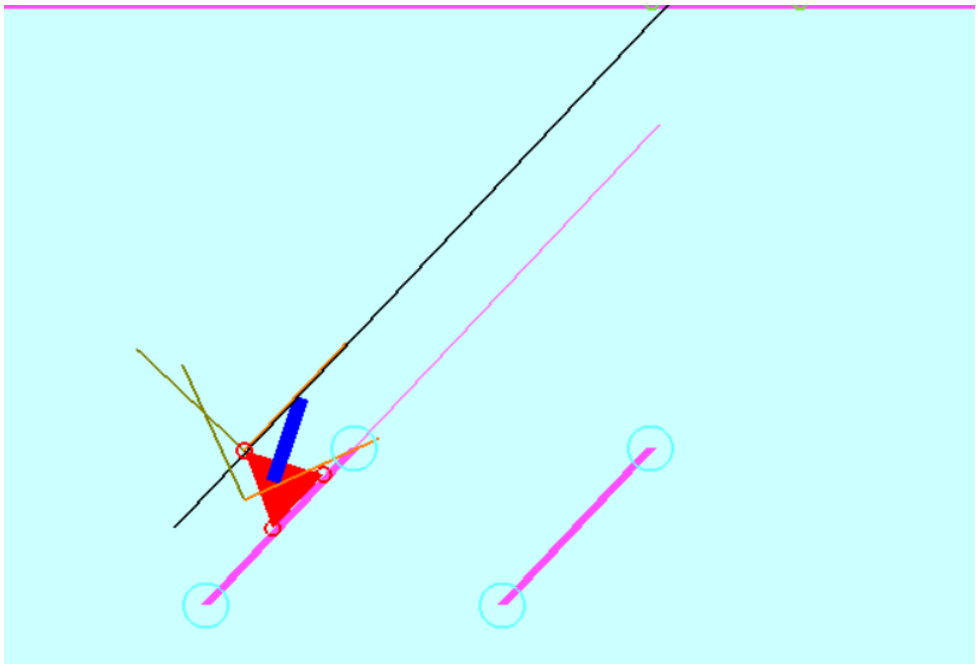
All of the code is on the physics2D class.

Another important feature.

With the normal, the perpendicular and the angle, I am checking if the player collides with a diagonal platform. If it is hitting with a diagonal and not 90 degrees, I use the vector projection formula to project the player's direction. I only project the direction, not the motion of the player. I can also rotate depending on the projected vector. This code is commented, but you can uncomment the following lines of code, and the triangle will rotate. With a space bar, you can turn the player to its initial position.

```
    position.x += newProjectedMotion.x * deltaTime;
    position.y += newProjectedMotion.y * deltaTime;

    //uncomment for doing rotation
    handleRotations(angle);


    //comment the next 4 lines of code if u want to do rotations
    leftVertex.x += newProjectedMotion.x * deltaTime ;
    leftVertex.y += newProjectedMotion.y * deltaTime;
    rightVertex.x += newProjectedMotion.x * deltaTime;
    rightVertex.y += newProjectedMotion.y * deltaTime;



}
```

Picture of the projected vector and rotation.



The thin purple line is the orthogonal vector of the normal. The black line is the projected vector. As you can see, the player is rotating towards the projected vector.

Also, I use the ortho2D, of OpenGL to follow the player.

Things that didn't work out.

I had a problem with my memory management and my main loop (update loop). I will develop for the next time a simple entity component system for handling my memory management and the objects on the window much better.

However, for this project, due to the update loop. I was not able to apply gravity, it would mean making drastic changes to the update loop and my physics.

The main problem with my memory management is the enemy. I tried to make a simple AI that shoot bullets to the enemy however, due to the memory management problem, when the AI shoot more than 3 bullets the game will crash, since is trying to access a null pointer. I tried to fix this and changed my whole system from raw pointers to smart pointers, however I was not able to find the problem and fix it, therefore this feature is not working properly you can test it if you press g. But is something that would need to be fixed. Hence for the next time I will use an entity component system.

Rules of Game

With 1 and 2, you can change the maps.

With r you can active the player death and with t you can unactive the player death. The player dies if it collides with the ball.

With g you can shoot bullets from the enemy.

This is just a simple physics simulation you can shoot and push and move boxes. Is just a physics simulation of collision a vector reflection. There is a lot of code, and it can be mess up.