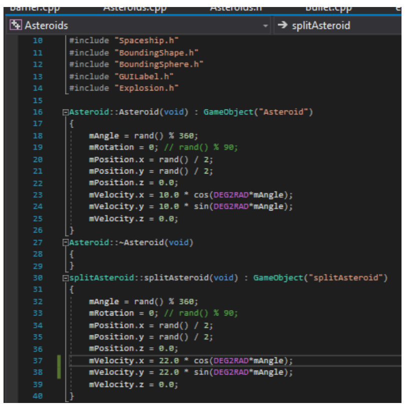
The feature I went with in part 1 was to split asteroids into smaller asteroids when shot by the player spaceship. Before creating this function, I decided to test things for example, spawning another asteroid of the same size when an asteroid object is hit. Once I figured that out it was a matter of creating the same asteroid but decreasing the scale of the sprite and possibly making the asteroid break into multiple smaller asteroids as it adds variety and fun to the game.

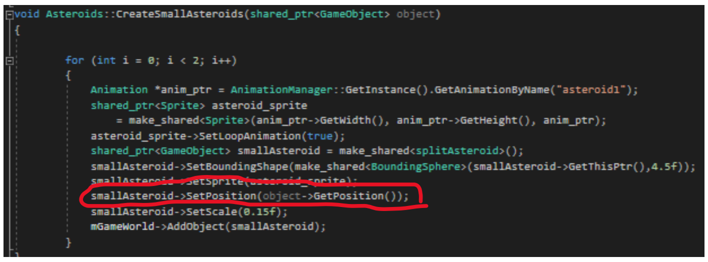
First of all, I created another class in the Asteroid class that also inherits GameObject and named it splitAsteroid. I then copied and pasted the code inside the Asteroid constructor and pasted it in the splitAsteroid Constructor so that it moves just like the normal sized asteroid. In order to make it slightly harder and give more variety to the game I decided to change the speed of the smaller asteroids so that it is much faster than the normal sized asteroids. I did this by increasing the velocity both x and y within the splitAsteroid Constructor.



```
#include "Spaceship.h"
#include "BoundingShape.h"
#include "BoundingSphere.h"
#include "GUILabel.h"
#include "Explosion.h"

Asteroid::Asteroid(void) : GameObject("Asteroid")
{
    mAngle = rand() % 360;
    mRotation = 0; // rand() % 90;
    mPosition.x = rand() / 2;
    mPosition.y = rand() / 2;
    mPosition.z = 0.0;
    mVelocity.x = 10.0 * cos(DEG2RAD*mAngle);
    mVelocity.y = 10.0 * sin(DEG2RAD*mAngle);
    mVelocity.z = 0.0;
}
Asteroid::~Asteroid(void)
{
}
splitAsteroid::splitAsteroid(void) : GameObject("splitAsteroid")
{
    mAngle = rand() % 360;
    mRotation = 0; // rand() % 90;
    mPosition.x = rand() / 2;
    mPosition.y = rand() / 2;
    mPosition.z = 0.0;
    mVelocity.x = 22.0 * cos(DEG2RAD*mAngle);
    mVelocity.y = 22.0 * sin(DEG2RAD*mAngle);
    mVelocity.z = 0.0;
}
```

Here is splitAsteroid and Asteroid constructor within the Asteroid.cpp file.

I then implemented a function within Asteroid**s**.cpp called CreateSmallAsteroids which took a GameObject as parameter. This allows us to set the position of the smallAsteroids to the object we pass in which is the normal sized asteroid from Asteroid.cpp.

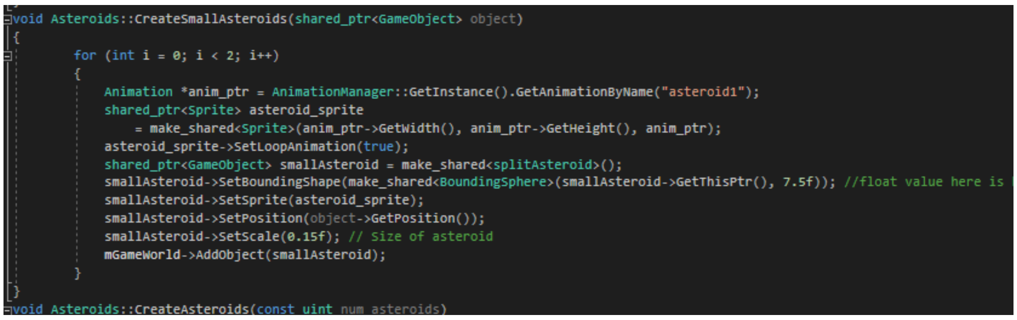


```
void Asteroids::CreateSmallAsteroids(shared_ptr<GameObject> object)
{


        for (int i = 0; i < 2; i++)
        {
            Animation *anim_ptr = AnimationManager::GetInstance().GetAnimationByName("asteroid1");
            shared_ptr<Sprite> asteroid_sprite
                = make_shared<Sprite>(anim_ptr->GetWidth(), anim_ptr->GetHeight(), anim_ptr);
            asteroid_sprite->SetLoopAnimation(true);
            shared_ptr<GameObject> smallAsteroid = make_shared<splitAsteroid>();
            smallAsteroid->SetBoundingShape(make_shared<BoundingSphere>(smallAsteroid->GetThisPtr(),4.5f));
            smallAsteroid->SetSprite(asteroid_sprite);
            smallAsteroid->SetPosition(object->GetPosition());
            smallAsteroid->SetScale(0.15f);
            mGameWorld->AddObject(smallAsteroid);
        }
```

In the screenshot we can see smallAsteroid of type GameObject use the function SetPosition to set the position of the smallAsteroid. Object which is the normal sized asteroid we called the GetPosition to specify where we want to spawn the small Asteroid.

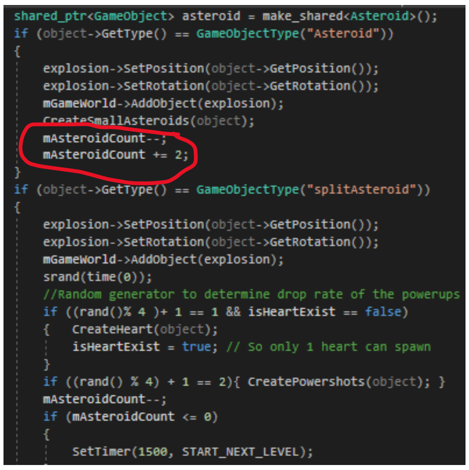
The animations and sprites were the same as the big asteroids, just the size of the sprites were scaled down and the hitbox was scaled alongside the sprite scale. I went with the size of 0.15 and a hitbox size of 4.5f which seemed to feel right and blew up as soon as the spaceship bullet touched the sprite.

I then felt like there should be more asteroids instead of just 1 being spawned when destroyed. So to add 2 smallAsteroids at the position of the normal asteroid I created a for loop within that CreateSmallAsteroids function allowing the game to spawn 2 or how ever many numbers we insert in the for loop. 2 seemed like a reasonable number.



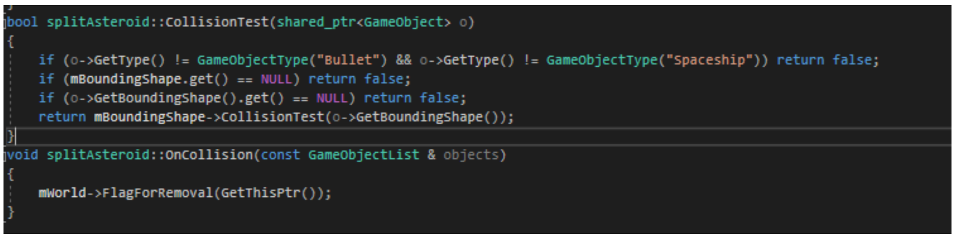
```cpp
void Asteroids::CreateSmallAsteroids(shared_ptr<GameObject> object)
{
        for (int i = 0; i < 2; i++)
        {
            Animation *anim_ptr = AnimationManager::GetInstance().GetAnimationByName("asteroid1");
            shared_ptr<Sprite> asteroid_sprite
                = make_shared<Sprite>(anim_ptr->GetWidth(), anim_ptr->GetHeight(), anim_ptr);
            asteroid_sprite->SetLoopAnimation(true);
            shared_ptr<GameObject> smallAsteroid = make_shared<splitAsteroid>();
            smallAsteroid->SetBoundingShape(make_shared<BoundingSphere>(smallAsteroid->GetThisPtr(), 7.5f)); //float value here is
            smallAsteroid->SetSprite(asteroid_sprite);
            smallAsteroid->SetPosition(object->GetPosition());
            smallAsteroid->SetScale(0.15f); // Size of asteroid
            mGameWorld->AddObject(smallAsteroid);
        }
}
void Asteroids::CreateAsteroids(const uint num_asteroids)
```

Doing this created problems such as the number of asteroids present would need to increase because more asteroids have spawned. For example, 3 normal sized asteroids have spawned and you move to the next level when the variable called mAsteroidCount set to 3 initially, is equal to 0. However, with the split asteroids there should technically be 9 asteroids in total, 3 normal sized which then get split into 2 smaller asteroids each meaning that we need to decrease mAsteroidCount by 1 but then increase by 2. So to do this I decreased the variables in the OnObjectRemoved function within Asteroids.cpp. When the “Asteroid” object is removed then decrease by 1 but add 2. The 2 being the 2 smaller asteroids that have spawned. Then when the “splitAsteroid” object is removed decrease 1 and if the mAsteroidCount is less than 0 then move onto the next level.



```cpp
shared_ptr<GameObject> asteroid = make_shared<Asteroid>();
if (object->GetType() == GameObjectType("Asteroid"))
{
    explosion->SetPosition(object->GetPosition());
    explosion->SetRotation(object->GetRotation());
    mGameWorld->AddObject(explosion);
    CreateSmallAsteroids(object);
    mAsteroidCount--;
    mAsteroidCount += 2;
}
if (object->GetType() == GameObjectType("splitAsteroid"))
{
    explosion->SetPosition(object->GetPosition());
    explosion->SetRotation(object->GetRotation());
    mGameWorld->AddObject(explosion);
    srand(time(0));
    //Random generator to determine drop rate of the powerups
    if ((rand()% 4 )+ 1 == 1 && isHeartExist == false)
    {   CreateHeart(object);
        isHeartExist = true; // So only 1 heart can spawn
    }
    if ((rand() % 4) + 1 == 2){ CreatePowershots(object); }
    mAsteroidCount--;
    if (mAsteroidCount <= 0)
    {
        SetTimer(1500, START_NEXT_LEVEL);
```

Another problem I faced during this part was that the small asteroids would not collide with the spaceship or bullet. The spaceship and bullets would just go through the small asteroid sprite. To solve this issue, I duplicated the CollisionTest but edited it so that is was within the splitAsteroid class and specified in the if statement that only return the bounding shape if the thing it collided with was a bullet or a spaceship. Which it then gets flagged for removal.



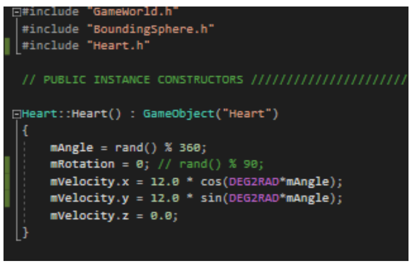
```cpp
bool splitAsteroid::CollisionTest(shared_ptr<GameObject> o)
{
    if (o->GetType() != GameObjectType("Bullet") && o->GetType() != GameObjectType("Spaceship")) return false;
    if (mBoundingShape.get() == NULL) return false;
    if (o->GetBoundingShape().get() == NULL) return false;
    return mBoundingShape->CollisionTest(o->GetBoundingShape());
}
void splitAsteroid::OnCollision(const GameObjectList & objects)
{
    mWorld->FlagForRemoval(GetThisPtr());
}
```

Also I feel that I could have added a random generator that ranges from 1.5 to 0.5. This random number within that range would be used for the small asteroid scale so that when 1 asteroid is destroyed it would spawn 2 smaller asteroids but are different sizes. The reason why I did not include this feature is because the hitbox would require the random number multiply by an Int and even then the hitbox would not fully match the size of the asteroid making it incosistent and is the main reason why I chose not to implement it. There are probably other means of doing this by figuring out a formula so that the hitbox multiplied by a number is proportional to the size of the asteroid.

Part 2

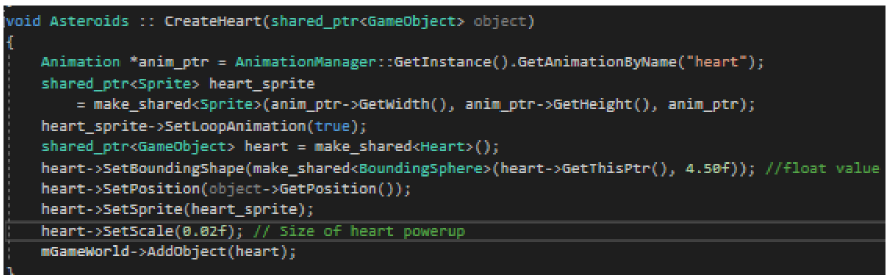
With part 2, I decided to go with the Power ups as I felt like a high score system would suit a game that has a fully fleshed out menu system. With the power ups I thought about how to manipulate the player ship in different types of ways. 1 of them is a heart up power up which gives te player an extra life. The 2nd power up is a bullet power up which allows the player to shoot 5 bullets in a wide spread. And the 3rd is a shield power up which gives the player ship a barrier that blocks any debris, asteroids or bullets from hitting the player spaceship. The barrier has its only health which will get destroyed as health reaches 0.

So to implement these features, I decided to start with creating the health power up first. I created a class for the health power up and copied and pasted the asteroid class but edited it for the Health class removing unecessary functions and variables and changing the names of each function according to the name of the class. The reason why I did this is because I want the power up to float around the map which the code in the asteroid class does. The code that moves the heart in a direction is the code in the constructor (mVelocity.x/y).

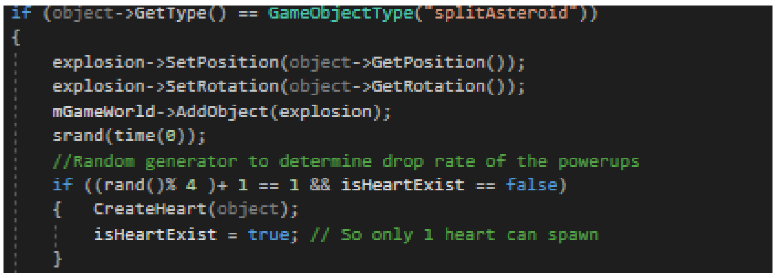


```
#include "GameWorld.h"
#include "BoundingSphere.h"
#include "Heart.h"

// PUBLIC INSTANCE CONSTRUCTORS ///////////////////////

Heart::Heart() : GameObject("Heart")
{
    mAngle = rand() % 360;
    mRotation = 0; // rand() % 90;
    mVelocity.x = 12.0 * cos(DEG2RAD*mAngle);
    mVelocity.y = 12.0 * sin(DEG2RAD*mAngle);
    mVelocity.z = 0.0;
}
```

I removed the position variable in the constructor which has the value of random number divided by 2. This is because I set the position in Asteroids.cpp when I spawn it using the Set Position function that all objects of type GameObject has.



```
void Asteroids :: CreateHeart(shared_ptr<GameObject> object)
{
    Animation *anim_ptr = AnimationManager::GetInstance().GetAnimationByName("heart");
    shared_ptr<Sprite> heart_sprite
        = make_shared<Sprite>(anim_ptr->GetWidth(), anim_ptr->GetHeight(), anim_ptr);
    heart_sprite->SetLoopAnimation(true);
    shared_ptr<GameObject> heart = make_shared<Heart>();
    heart->SetBoundingShape(make_shared<BoundingSphere>(heart->GetThisPtr(), 4.50f)); //float value
    heart->SetPosition(object->GetPosition());
    heart->SetSprite(heart_sprite);
    heart->SetScale(0.02f); // Size of heart powerup
    mGameWorld->AddObject(heart);
```

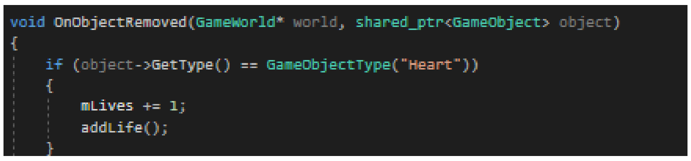


```
if (object->GetType() == GameObjectType("splitAsteroid"))
{
    explosion->SetPosition(object->GetPosition());
    explosion->SetRotation(object->GetRotation());
    mGameWorld->AddObject(explosion);
    srand(time(0));
    //Random generator to determine drop rate of the powerups
    if ((rand()% 4 )+ 1 == 1 && isHeartExist == false)
    {   CreateHeart(object);
        isHeartExist = true; // So only 1 heart can spawn
    }
```

The “object” passed in as a parameter is the “splitAsteroid” object. We call the CreateHeart function when the object splitAsteroid is destroyed. Then we get the object type that was destroyed and pass it into CreateHeart as a parameter. This then allows us to get the object that got destroyed their position allowing me to spawn a heart at the exact location of the small asteroid.

I then used a random number generator in order to make the drop rate of the hearts a certain percentage. There is a 25% chance that the heart will spawn when a small asteroid is destroyed. Also I made it so that if there is a heart power up in the world that has not been retrieved by the player, it won't spawn another heart so that it is quite difficulty to survive and players will not be able to take advantage of the heart power up meaning they will have to play more carefully.

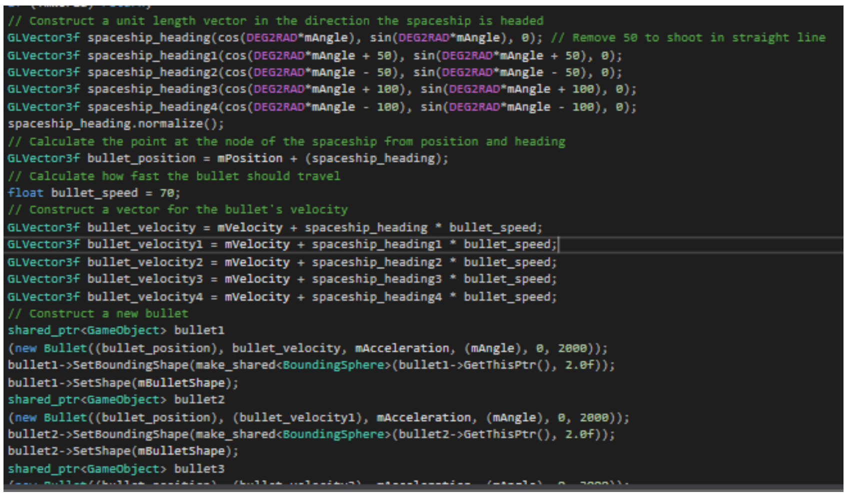
To increase the heart when player touches the heart power up I added a new if statement in the OnObjectRemoved function in Player.h. When a certain object is removed then execute this code. In our case the certain object will be of type Heart. When that object is removed then increase the lives variable by 1. Then addLife() updates the Label using listeners.



```
void OnObjectRemoved(GameWorld* world, shared_ptr<GameObject> object)
{
    if (object->GetType() == GameObjectType("Heart"))
    {
        mLives += 1;
        addLife();
    }
```

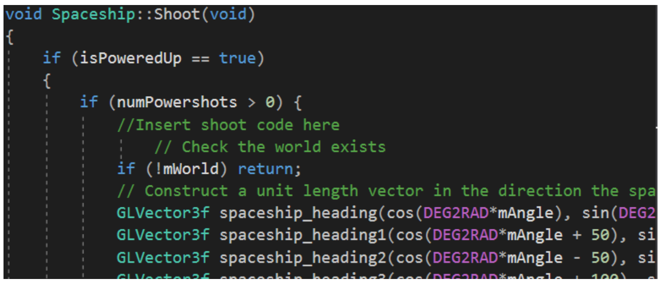
After that was done I then started to create the bullet power up. Just like the heart power up I created the constructor and initialized its velocity so that it can maneuver through the map. I then created the create bullet power up function in the Asteroids.cpp similar to the create heart. I then set the position of the power just like the hearts. I pass through the object that gets destroyed which in my case is smallAsteroid and then call the create power up function which spawns the power up at the smallAsteroid position.

To test things before I went further I decided to mess around with the shoot function in Spaceship.cpp to see how to change the bullet direction. First of all I created multiple bullet objects of type Bullet. Then I changed values of each variable to see if it changed the bullets position or direction. After changing each variable I found out that changing the Spaceship_heading variable causes the bullet to shoot in different direction. The spaceship_heading uses the angle and I +50 or -50 to the angle variable to make it shoot diagonally. Then I inserted the spaceship_heading variables in the 4 newly created bullet position variables. Each variable will be used for each bullet created. Then I inserted the bullet position into the parameters to pass the values when creating a new bullet.

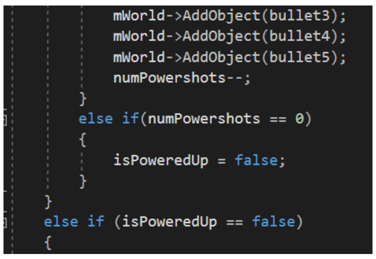


```
// Construct a unit length vector in the direction the spaceship is headed
GLVector3f spaceship_heading(cos(DEG2RAD*mAngle), sin(DEG2RAD*mAngle), 0); // Remove 50 to shoot in straight line
GLVector3f spaceship_heading1(cos(DEG2RAD*mAngle + 50), sin(DEG2RAD*mAngle + 50), 0);
GLVector3f spaceship_heading2(cos(DEG2RAD*mAngle - 50), sin(DEG2RAD*mAngle - 50), 0);
GLVector3f spaceship_heading3(cos(DEG2RAD*mAngle + 100), sin(DEG2RAD*mAngle + 100), 0);
GLVector3f spaceship_heading4(cos(DEG2RAD*mAngle - 100), sin(DEG2RAD*mAngle - 100), 0);
spaceship_heading.normalize();
// Calculate the point at the node of the spaceship from position and heading
GLVector3f bullet_position = mPosition + (spaceship_heading);
// Calculate how fast the bullet should travel
float bullet_speed = 70;
// Construct a vector for the bullet's velocity
GLVector3f bullet_velocity = mVelocity + spaceship_heading * bullet_speed;
GLVector3f bullet_velocity1 = mVelocity + spaceship_heading1 * bullet_speed;
GLVector3f bullet_velocity2 = mVelocity + spaceship_heading2 * bullet_speed;
GLVector3f bullet_velocity3 = mVelocity + spaceship_heading3 * bullet_speed;
GLVector3f bullet_velocity4 = mVelocity + spaceship_heading4 * bullet_speed;
// Construct a new bullet
shared_ptr<GameObject> bullet1
(new Bullet((bullet_position), bullet_velocity, mAcceleration, (mAngle), 0, 2000));
bullet1->SetBoundingShape(make_shared<BoundingSphere>(bullet1->GetThisPtr(), 2.0f));
bullet1->SetShape(mBulletShape);
shared_ptr<GameObject> bullet2
(new Bullet((bullet_position), (bullet_velocity1), mAcceleration, (mAngle), 0, 2000));
bullet2->SetBoundingShape(make_shared<BoundingSphere>(bullet2->GetThisPtr(), 2.0f));
bullet2->SetShape(mBulletShape);
shared_ptr<GameObject> bullet3
```

After I figured that out I then made 2 new variables. A boolean and an Integer. The boolean will be used to check if the power up has been obtained and the int will be the amount of times you can use that power before it wares out (Basically Ammo). So when the bullet power up is removed, we set the boolean to true meaning that the power up has been received. Then in the shoot method I create an if statement so that it will only shoot 5 bullets if it is true else shoot 1 bullet if false. I also had a nested if statement inside which checks whether the integer variable is more than 0. If it is more than 0 then you can still shoot the power up bullets, if not then make the boolean variable false which then results in shooting 1. Where I decreased the ammo count is within the shoot method just after adding 5 of the bullet objects.



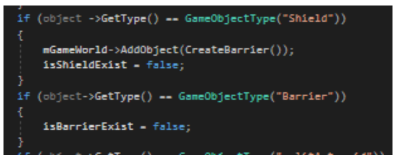
```
void Spaceship::Shoot(void)
{
    if (isPoweredUp == true)
    {
        if (numPowershots > 0) {
            //Insert shoot code here
                // Check the world exists
            if (!mWorld) return;
            // Construct a unit length vector in the direction the spa
            GLVector3f spaceship_heading(cos(DEG2RAD*mAngle), sin(DEG2
            GLVector3f spaceship_heading1(cos(DEG2RAD*mAngle + 50), si
            GLVector3f spaceship_heading2(cos(DEG2RAD*mAngle - 50), si
```



```
            mWorld->AddObject(bullet3);
            mWorld->AddObject(bullet4);
            mWorld->AddObject(bullet5);
            numPowershots--;
        }
        else if(numPowershots == 0)
        {
            isPoweredUp = false;
        }
    }
    else if (isPoweredUp == false)
    {
```

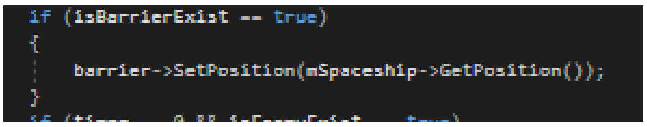
I then moved to creating the shield power up. I made the shield power up spawn just like the hearts giving a random probability of it dropping and having it spawn at the position of the small asteroids when they get destroyed. My initial idea of implementing this was to have a boolean which turns true or false depending on whether the power up was picked up. When it is true send it to the spaceship.cpp in which the if statement in the render function would check to see if it is true or false. If true then render a blue shape around the spaceship indicating a barrier. If false then don’t do anything. The problem with this is that I would have pass the boolean to other classes so that when the shield is true do not decrease the health and don’t remove the spaceship on collisions. So I thought of another way which was simpler.

I created another class called Barrier which has a few of the functions that Spaceship has such as on collision tests and update. It does not need thrust or rotate as all I planned to do was to constantly move the position of the shield exactly where the player spaceship is each frame. I then created the createBarrier function within asteroids.cpp and called this function when the shield power up is removed.



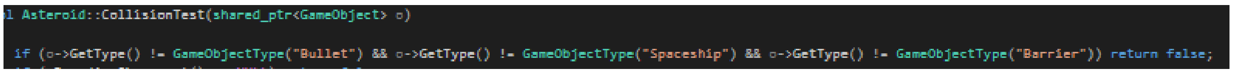
```
if (object ->GetType() == GameObjectType("Shield"))
{
    mGameWorld->AddObject(CreateBarrier());
    isShieldExist = false;
}
if (object->GetType() == GameObjectType("Barrier"))
{
    isBarrierExist = false;
}
```

Within the OnWorldUpdate, I set the barrier position to the player spaceship position only if a boolean is true. This boolean checks whether the barrier has spawned into the world. If not then we don’t have to constantly set the position of the barrier. If yes then set the position.



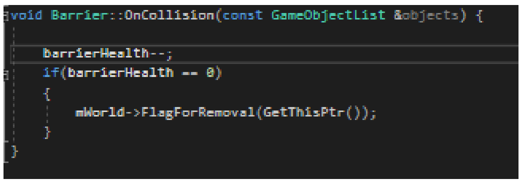
```
if (isBarrierExist == true)
{
    barrier->SetPosition(mSpaceship->GetPosition());
}
```

To make it like a barrier and shield the player from enemy objects I decided to make each object that can destroy the spaceship, be destroyed by the barrier.



```
l Asteroid::CollisionTest(shared_ptr<GameObject> o)

 if (o->GetType() != GameObjectType("Bullet") && o->GetType() != GameObjectType("Spaceship") && o->GetType() != GameObjectType("Barrier")) return false;
```

This is applied to bullets, asteroids, small asteroids. And in the Barrier.cpp, the OnCollision function removes the barrier when the barrier collides with specified object. To make it so the barrier lasts longer instead of getting destroyed after 1 collision I made an Integer variable which is basically the health of the barrier. Every time the barrier collides the OnCollision will get called. Inside it I decreased the barrierHealth by 1 and If the barrier health is equal to 0 then execute the flagforremoval to destroy the barrier.



```
void Barrier::OnCollision(const GameObjectList &objects) {


    barrierHealth--;
    if(barrierHealth == 0)
    {
        mWorld->FlagForRemoval(GetThisPtr());
    }
}
```

There were not that many problems with this part, the biggest problem was rendering the barrier which was solved by just creating a new class. This means that I was able to use any image as a barrier and also not have to go through the trouble of passing through values or making another hitbox for the barrier.

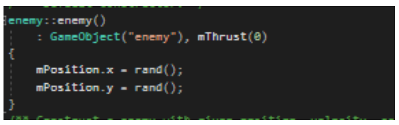
I felt like a small improvement I could have implemented for the shield power up is that instead of being destroyed depending on the barrier health, destroy the barrier depending on time for example after 5 seconds. This barrier could also be used for when the player respawns as in games, when you die and respawn you have a small period of invulnerability.

Part 3

For part 3 I decided to go with an alien spaceship. Mainly because I felt that the demo was more for a menu. So what I intended to do for this part was create an alien spaceship that follows the player and shoot at the player. I would do something similar like the barrier where I would update its position and rotation every frame meaning that I would have to do this somewhere in the OnWorldUpdate. This could also be done in the Enemy class inside the Update function however we would need to pass a shared pointer of the player spaceship to get its exact location.

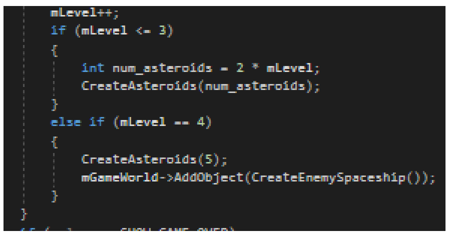
So first of all I create another class. Just like the player spaceship class as it will thurst, shoot and collide with things. I did have to edit some of them because there are functions which change the state of the class depending on if a powerup has been collected or not. I wanted to make an enemy that could not use power ups so I deleted those uneccessary variables and functions. All I left was rotate, thrust, shoot, update and the collision functions.

As for the constructor I wanted the enemy to spawn at a random location. So in the constructor I set values for the position. The values being a number from a random number generator.



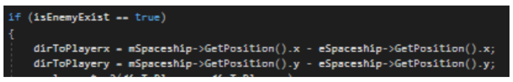
```
enemy::enemy()
    : GameObject("enemy"), mThrust(0)
{
    mPosition.x = rand();
    mPosition.y = rand();
}
```

Within the asteroids.cpp, I added another function which was CreateEnemySpaceship. This will be called whenever I want to spawn the enemy. In my game it is spawned on the last level. To do this I did an if statement saying that if the mLevel is equal to 4 then spawn it. If the mLevel is less than or equal to 3 only spawn asteroids.



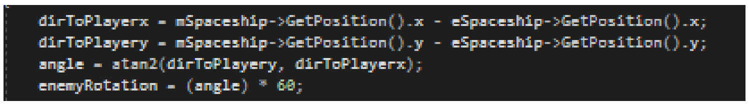
```
        mLevel++;
        if (mLevel <= 3)
        {
            int num_asteroids = 2 * mLevel;
            CreateAsteroids(num_asteroids);
        }
        else if (mLevel == 4)
        {
            CreateAsteroids(5);
            mGameWorld->AddObject(CreateEnemySpaceship());
        }
    }
```

So after testing this and seeing it spawn on the final level but not move I started working on its movement. First of all I needed to make the enemy spaceship face the player and to do this I thought to change the angle of the enemy spaceship but did not know what values to put in. After a lot of research I came across a maths function called atan. This function calculates the arc tangent of 2 numbers that we pass to the function. This will be the distance between playerspaceship x minus enemyspaceship x and will be done the same for y.



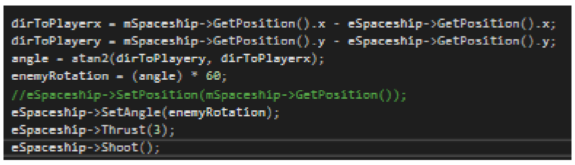
```
if (isEnemyExist == true)
{
    dirToPlayerx = mSpaceship->GetPosition().x - eSpaceship->GetPosition().x;
    dirToPlayery = mSpaceship->GetPosition().y - eSpaceship->GetPosition().y;
```

Now we have the distance in y and x, we pass that in the atan2 function and the result would return the radian. I then set the rotation to the result of the atan2 function. Then my first problem came up which was that the enemy spaceship was rotating extremely slow. I then found out that the radian returned is an extremely small decimal number which is the reason why it was turning so slow. I then did trial and error and multiplied the radian by 50 and kept increasing until the rotation was accurate. The point at where the rotation was accurate was between 58 and 60. I went with 60 as I wanted the enemy spaceship to turn slightly fast.



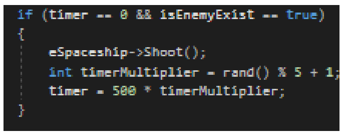
```
    dirToPlayerx = mSpaceship->GetPosition().x - eSpaceship->GetPosition().x;
    dirToPlayery = mSpaceship->GetPosition().y - eSpaceship->GetPosition().y;
    angle = atan2(dirToPlayery, dirToPlayerx);
    enemyRotation = (angle) * 60;
```

Then once I got the enemy to face the player I then had to move it and make it shoot. So I called the thrust function and past in a value. This would be the speed at which it follows the player. I then called the shoot function. The shoot function is different than the player, it shoots 5 out in a cone and does not require a power up to do so.



```
dirToPlayerx = mSpaceship->GetPosition().x - eSpaceship->GetPosition().x;
dirToPlayery = mSpaceship->GetPosition().y - eSpaceship->GetPosition().y;
angle = atan2(dirToPlayery, dirToPlayerx);
enemyRotation = (angle) * 60;
//eSpaceship->SetPosition(mSpaceship->GetPosition());
eSpaceship->SetAngle(enemyRotation);
eSpaceship->Thrust(3);
eSpaceship->Shoot();
```

The problem with this is that it shoots every frame causing the fps to drop. So I had to implement a timer that decreases by 1 every frame. I then did an if statement where if the timer reaches 0 and the enemy spaceship exists then shoot and set the timer to a high number again.



```
if (timer == 0 && isEnemyExist == true)
{
    eSpaceship->Shoot();
    int timerMultiplier = rand() % 5 + 1;
    timer = 500 * timerMultiplier;
}
```

I multiplied the timer by a random number which resulted in the enemy spaceship shooting at different intervals instead of a set interval (Like shoot every 3 seconds). This makes it so the A.i is less predictable and makes it challenging for the player.

The problems I faced during the implementation of part 3 was when adding an enemy spaceship within an if statement like mLevel == 4 the game would open but display a blank screen. This was a problem and to solve it I had to call the createEnemySpaceship function outside the if statement spawning the enemy spaceship on game start. However this is not what I intended to do as I want the enemy spaceship to spawn on the last level as a Boss fight. The problem was that I was constantly changing the enemyspaceship angle in the OnWorldUpdate function despite it not being spawned. So I decided to create a boolean to tell the game that the enemy exists and has spawned and used that boolean within an if statement. If enemy exists then start changing the angle and thrust.

Another problem was that when the newly implemented enemy spaceship is destroyed it continues to fire bullets at the player. To solve this issue I used a boolean to check whether the enemy spaceship exists or not. The initial value is false, but when we call the CreateEnemySpaceship function it changes the value to true as I inserted inside the function a line that changes the value to true. Similarly I did the same for when the enemy spaceship gets destroyed. OnObjectRemoved function I included a line that changes the value to false so when the enemy spaceship is removed it will change the variable value. The IF statement within the OnWorldUpdate checks for the value of the boolean and if it is true then start shooting. If false then do not shoot.

The last problem I faced during part 3 was that the enemy spaceship stopped following the player when player dies once or goes past the enemy spaceship too fast. This was because I made the variable that stores the difference in distance between player and enemy in a variable of type GLVector2f. This worked when the enemy spawns but was really inconsistent as the game played on. So I split the variable into 2 float variables. Distance x and distance y which seemed to solve the problem.

Overall the project was a fun learning experience to see what C++ is capable of and helped me understand more about shared pointers and so on. There are improvements I feel that could be added but due to time constraints it could not be added for example having a homing missile shoot from the enemy spaceship. This would be implemented similarly to the enemy spaceship where it follows the player but it is spawned every so often just like how the shoot function is called depending on the time. Or I could have implemented multiple enemies that follow the player. Despite that I felt like I implemented all 3 parts to the best of my abilities and it works wells with some depth within the gameplay like powerups and boss battle.