IN2026 Coursework

GitHub Link

https://github.com/CitySefti/Games-Tech-Coursework

Part 1: Start Screen

Objectives:

- Create screen with title and key prompt using GUILabels.

An alternative would be to create a "texture"/image that is on the display at first before transitioning to the game world.

Outline of changes:

- Two new labels in Asteroids.h/.cpp
- One new Key press implementation
- Set initial visibility of lives and scores labels to false.

Implementation:

Created two new GUILabels alongside the already existing ones in Asteroids.h.

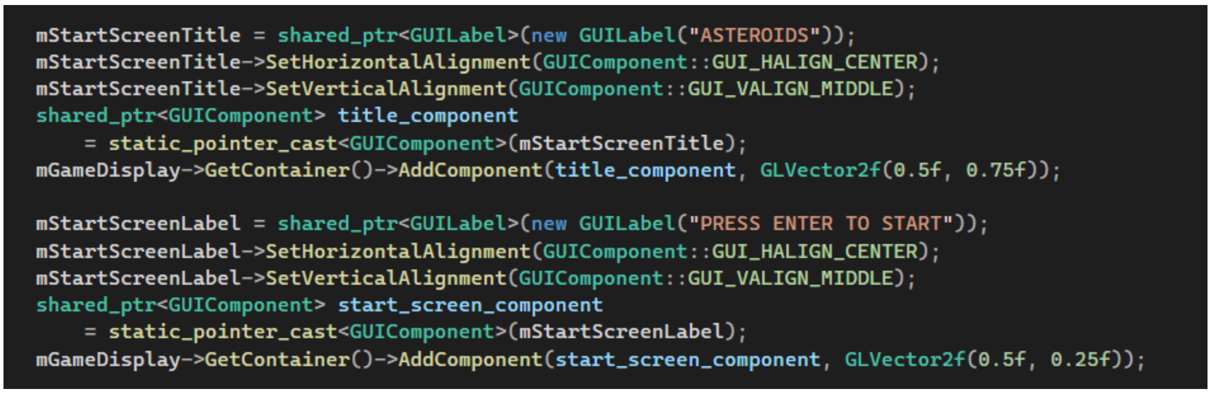
```
shared_ptr<GUILabel> mStartScreenTitle;
shared_ptr<GUILabel> mStartScreenLabel;
```

Next, I created the GUI.

First, I set the visibility for the score and lives labels to false, this is to make sure they are not present on the start screen.

```
mScoreLabel->SetVisible(false);

mLivesLabel->SetVisible(false);
```

Then I made the GUI components for the start screen: the Title and Key prompt in `Asteroids::CreateGUI`.



```
mStartScreenTitle = shared_ptr<GUILabel>(new GUILabel("ASTEROIDS"));
mStartScreenTitle->SetHorizontalAlignment(GUIComponent::GUI_HALIGN_CENTER);
mStartScreenTitle->SetVerticalAlignment(GUIComponent::GUI_VALIGN_MIDDLE);
shared_ptr<GUIComponent> title_component
    = static_pointer_cast<GUIComponent>(mStartScreenTitle);
mGameDisplay->GetContainer()->AddComponent(title_component, GLVector2f(0.5f, 0.75f));

mStartScreenLabel = shared_ptr<GUILabel>(new GUILabel("PRESS ENTER TO START"));
mStartScreenLabel->SetHorizontalAlignment(GUIComponent::GUI_HALIGN_CENTER);
mStartScreenLabel->SetVerticalAlignment(GUIComponent::GUI_VALIGN_MIDDLE);
shared_ptr<GUIComponent> start_screen_component
    = static_pointer_cast<GUIComponent>(mStartScreenLabel);
mGameDisplay->GetContainer()->AddComponent(start_screen_component, GLVector2f(0.5f, 0.25f));
```

The visibility of these is automatically true and don't need to be set to false as they should be present on the initial display.

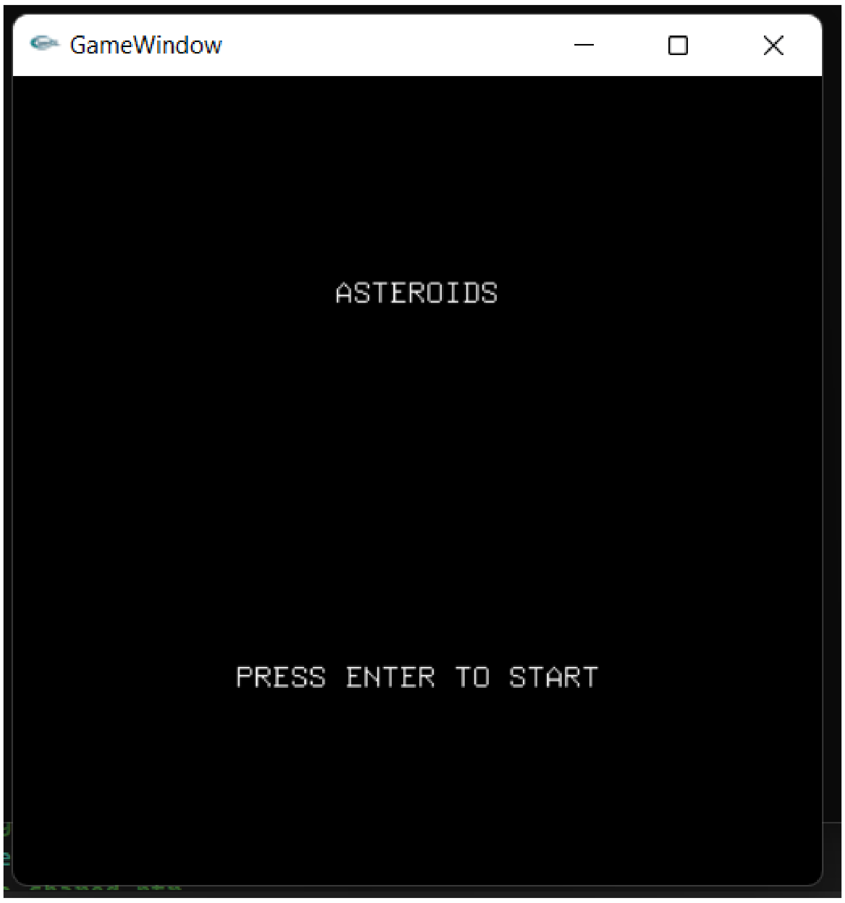
Finally, I made it so to start the game, the player needs to press the enter key. I did this by expanding the `Asteroids::OnKeyPressed method.`

```
switch (key)
	{
	case ' ':
		mSpaceship->Shoot();
		break;
	case 13: // Enter
		mStartScreenTitle->SetVisible(false);
		mStartScreenLabel->SetVisible(false);
		mLivesLabel->SetVisible(true);
		mScoreLabel->SetVisible(true);
		mGameWorld->AddObject(CreateSpaceship());
		CreateAsteroids(10);
	default:
		break;
	}
```

The tricky part here for me was seeing what could/should be in `Asteroids::OnKeyPressed` and what should remain in `Asteroids::Start`.

In the end, the code for creating the spaceship and asteroids on screen were moved to the key pressed method – this is so that nothing besides the title and key prompt are on screen for the start.

The result looks like this:



ASTEROIDS

PRESS ENTER TO START

Part 2: Power-Up System

Objectives:

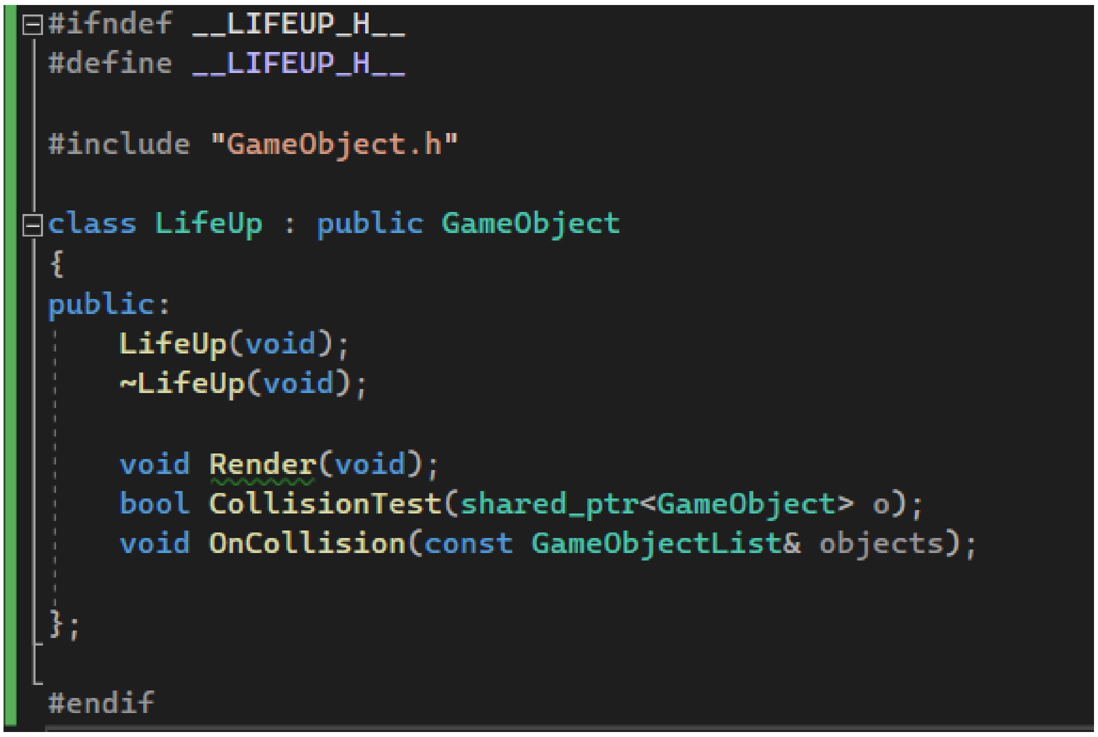
- Implement a power-up (Extra Lives)
- Update collisions.

Outline of changes:

- One new Header file and accompanying cpp.
- Changes to collision throughout the game. (e.g., bullets)
- New sprite
- New listener method for player listener.

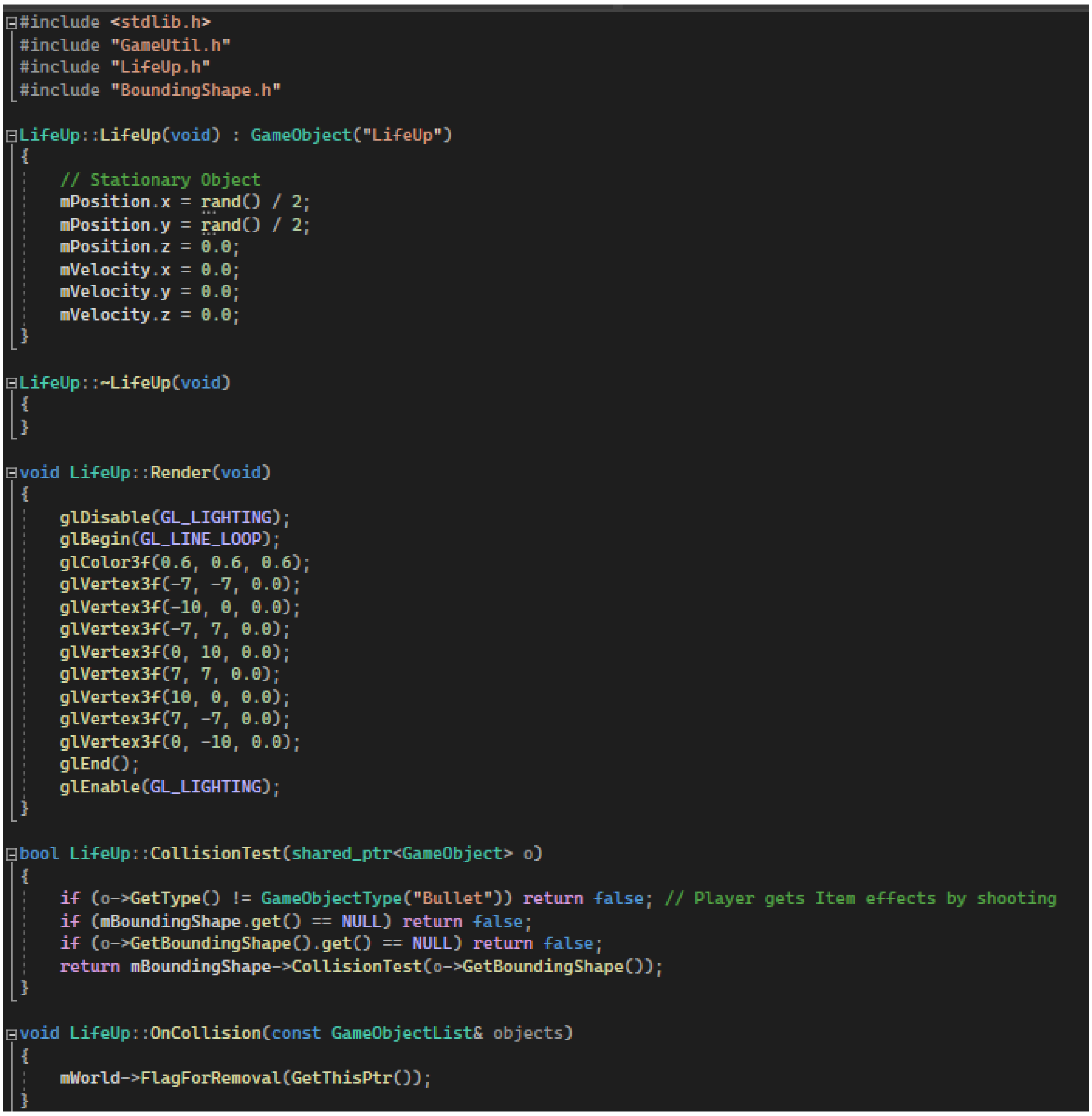
Implementation:

First, I made LifeUp.h.



```cpp
#ifndef __LIFEUP_H__
#define __LIFEUP_H__

#include "GameObject.h"

class LifeUp : public GameObject
{
public:
    LifeUp(void);
    ~LifeUp(void);

    void Render(void);
    bool CollisionTest(shared_ptr<GameObject> o);
    void OnCollision(const GameObjectList& objects);

};

#endif
```

Followed by LifeUp.cpp (Note: I did use tutorial 2 as a base for this)



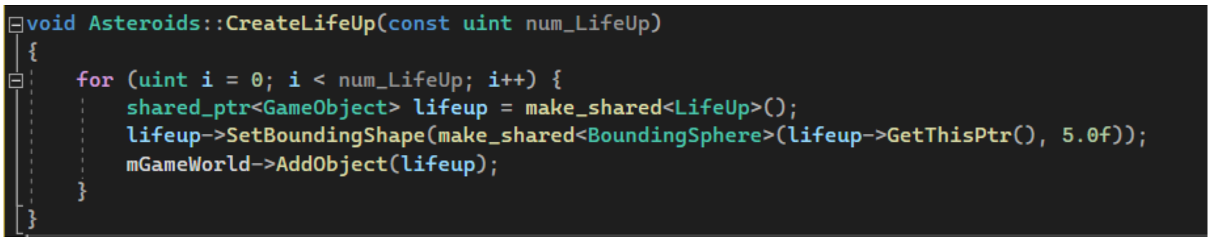
```cpp
#include <stdlib.h>
#include "GameUtil.h"
#include "LifeUp.h"
#include "BoundingShape.h"

LifeUp::LifeUp(void) : GameObject("LifeUp")
{
    // Stationary Object
    mPosition.x = rand() / 2;
    mPosition.y = rand() / 2;
    mPosition.z = 0.0;
    mVelocity.x = 0.0;
    mVelocity.y = 0.0;
    mVelocity.z = 0.0;
}

LifeUp::~LifeUp(void)
{
}

void LifeUp::Render(void)
{
    glDisable(GL_LIGHTING);
    glBegin(GL_LINE_LOOP);
    glColor3f(0.6, 0.6, 0.6);
    glVertex3f(-7, -7, 0.0);
    glVertex3f(-10, 0, 0.0);
    glVertex3f(-7, 7, 0.0);
    glVertex3f(0, 10, 0.0);
    glVertex3f(7, 7, 0.0);
    glVertex3f(10, 0, 0.0);
    glVertex3f(7, -7, 0.0);
    glVertex3f(0, -10, 0.0);
    glEnd();
    glEnable(GL_LIGHTING);
}

bool LifeUp::CollisionTest(shared_ptr<GameObject> o)
{
    if (o->GetType() != GameObjectType("Bullet")) return false; // Player gets Item effects by shooting
    if (mBoundingShape.get() == NULL) return false;
    if (o->GetBoundingShape().get() == NULL) return false;
    return mBoundingShape->CollisionTest(o->GetBoundingShape());
}

void LifeUp::OnCollision(const GameObjectList& objects)
{
    mWorld->FlagForRemoval(GetThisPtr());
}
```

Next, I moved onto making changes in Asteroids.h and .cpp.

In Asteroids.h, I put a method to create the new object in game world.

```cpp
void CreateLifeUp(const uint num_LifeUp);
```

In Asteroids.cpp, I added: `#include "LifeUp.h"`, so that I can define the method.



```cpp
void Asteroids::CreateLifeUp(const uint num_LifeUp)
{
    for (uint i = 0; i < num_LifeUp; i++) {
        shared_ptr<GameObject> lifeup = make_shared<LifeUp>();
        lifeup->SetBoundingShape(make_shared<BoundingSphere>(lifeup->GetThisPtr(), 5.0f));
        mGameWorld->AddObject(lifeup);
    }
}
```

I can test this by putting CreateLifeUp(1) in Asteroids::Start(). This is only for testing purposes, by putting it in start I am making it so that is it visible on the start screen.

When I tried running the whole game, I noticed an issue with asteroid collisions, if an asteroid ran into the new object, it gets destroyed, but the spaceship does not.

So now I looked to make changes to other object collisions to see how other objects should interact with items.

To fix the bug with asteroid collisions, I changed this line:

```
if (GetType() == o->GetType()) return false;
```

to:

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

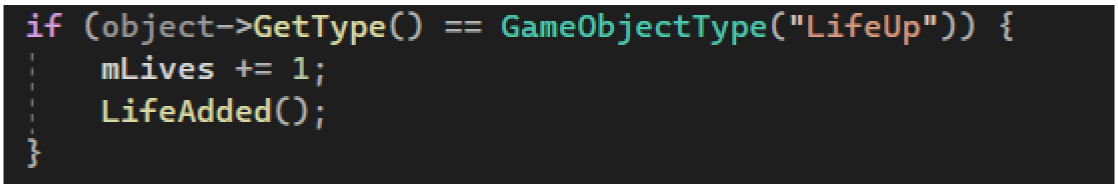
However, whilst this did make it so that asteroids did not get destroyed by the new object, it made it so that previous collisions with bullets and the spaceship did not work. To fix this I swapped out the || (OR) for && (AND) which did resolve the issues.

Bullet needed to be altered so that they actually destroy the new object. (using && from before) (`Bullet::CollisionTest`)

```
    if (o->GetType() != GameObjectType("Asteroid") && o->GetType() !=
    GameObjectType("LifeUp")) return false;
```

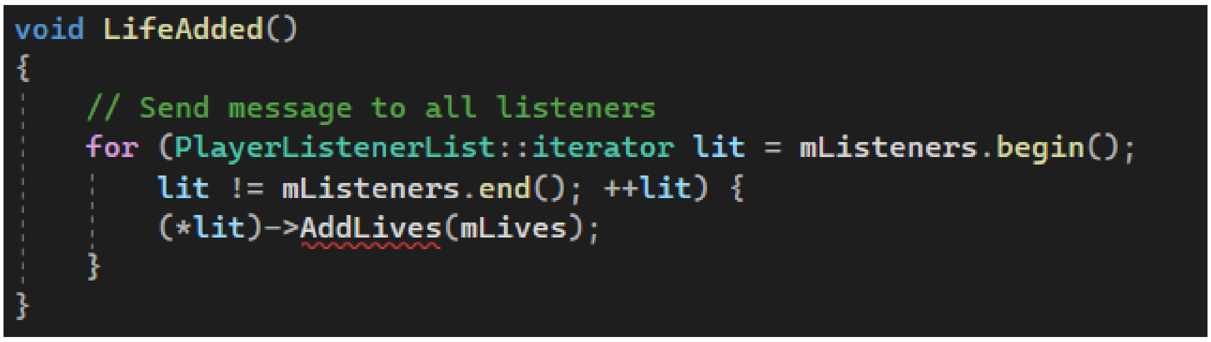
The next thing to do is to give the new object an effect. To start, I went to player.h to add a new condition to `void OnObjectRemoved` and add a new method to send a message to all listeners.

New condition:



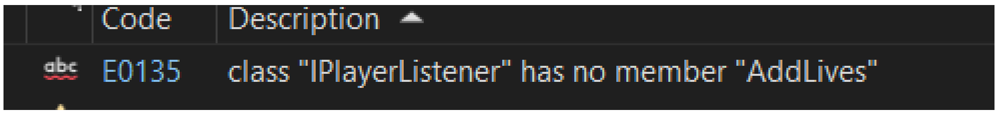
```
if (object->GetType() == GameObjectType("LifeUp")) {
    mLives += 1;
    LifeAdded();
}
```

New method: (this is basically the equivalent of “FirePlayerKilled” for adding lives)



```
void LifeAdded()
{
    // Send message to all listeners
    for (PlayerListenerList::iterator lit = mListeners.begin();
        lit != mListeners.end(); ++lit) {
        (*lit)->AddLives(mLives);
    }
}
```

As you can see, there is an error. This is caused by :

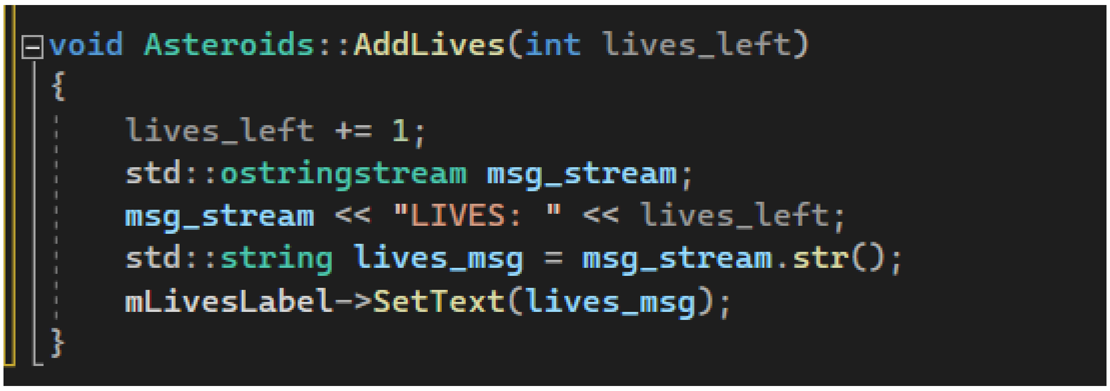


```
Code     Description
E0135    class "IPlayerListener" has no member "AddLives"
```

To fix this I added: `virtual void AddLives(int lives_left) = 0;` , to IPlayerListener.h.

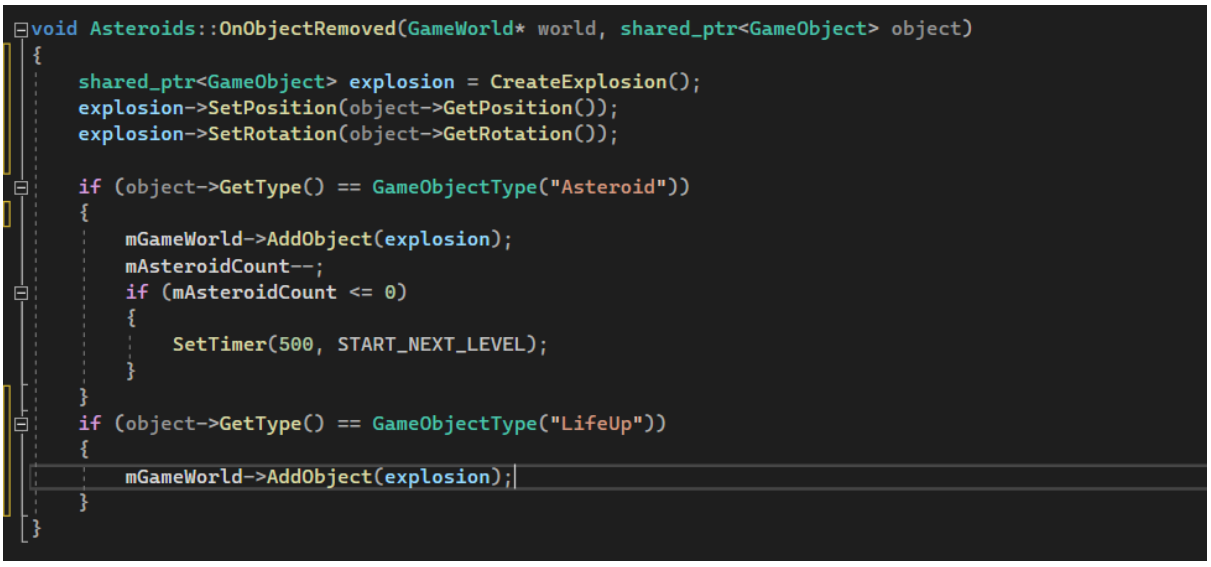
I then had to define it in Asteroids.h and .cpp.

`void AddLives(int lives_left); -> Header file`



```cpp
void Asteroids::AddLives(int lives_left)
{
    lives_left += 1;
    std::ostringstream msg_stream;
    msg_stream << "LIVES: " << lives_left;
    std::string lives_msg = msg_stream.str();
    mLivesLabel->SetText(lives_msg);
}
```

Next was to give the removal of the object an explosion. For this I Altered `Asteroids::OnObjectRemoved`.

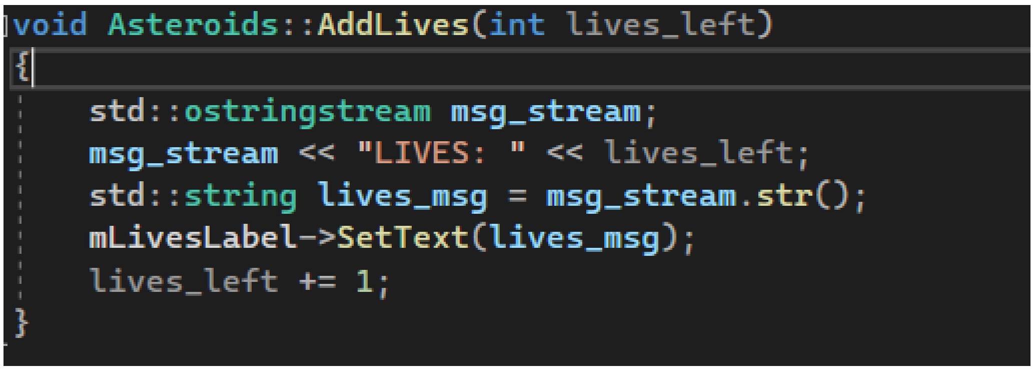


```cpp
void Asteroids::OnObjectRemoved(GameWorld* world, shared_ptr<GameObject> object)
{
    shared_ptr<GameObject> explosion = CreateExplosion();
    explosion->SetPosition(object->GetPosition());
    explosion->SetRotation(object->GetRotation());

    if (object->GetType() == GameObjectType("Asteroid"))
    {
        mGameWorld->AddObject(explosion);
        mAsteroidCount--;
        if (mAsteroidCount <= 0)
        {
            SetTimer(500, START_NEXT_LEVEL);
        }
    }
    if (object->GetType() == GameObjectType("LifeUp"))
    {
        mGameWorld->AddObject(explosion);
    }
}
```

When I tried running it, it mostly worked – just 1 bug fix needed.

When destroying the object, the player does only get 1 extra life, but the displayed life goes up by 2.

Fixed this by taking lives_left += 1 from the top to the bottom of the AddLives() method.

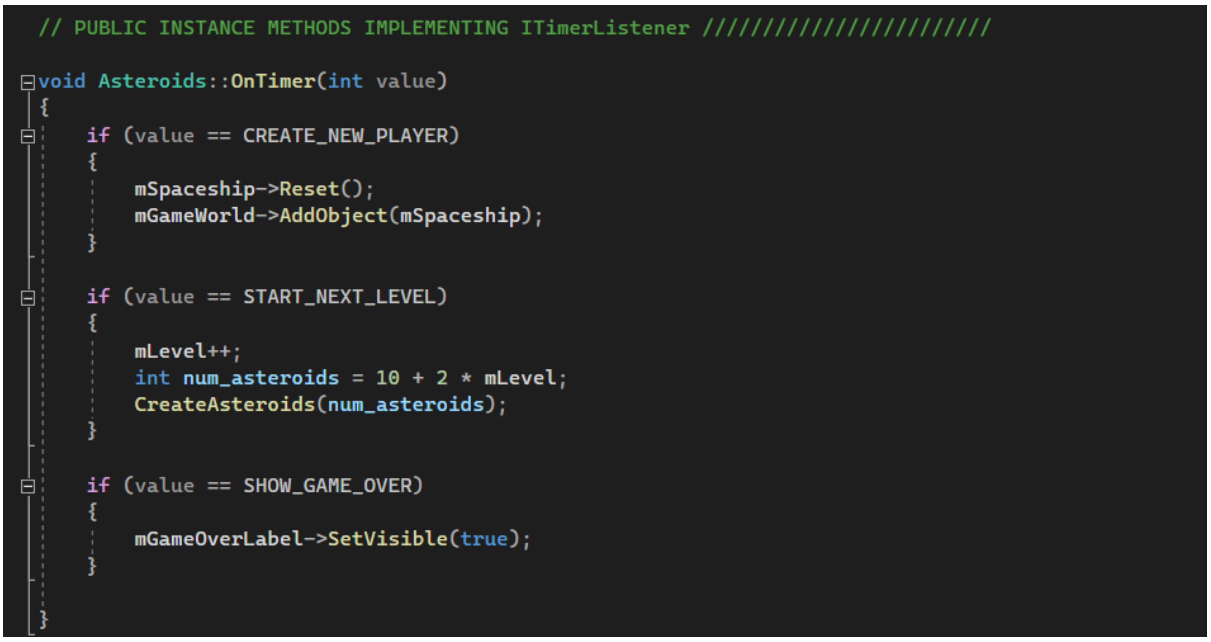


```
void Asteroids::AddLives(int lives_left)
{
    std::ostringstream msg_stream;
    msg_stream << "LIVES: " << lives_left;
    std::string lives_msg = msg_stream.str();
    mLivesLabel->SetText(lives_msg);
    lives_left += 1;
}
```

Two things left: sprite and implementation into game play loop. (Sprites covered later)

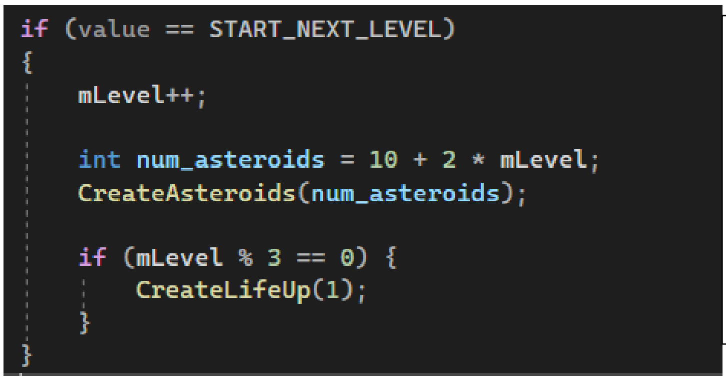
Up until now, I have merely created the object at the start for the purpose of testing, but this isn't how a power up would work in an actual gameplay loop. I can't handle it the same way the game handles the actual asteroids as that would be very imbalanced (a lot of extra lives).

For this I looked into how the game handles it's levels.



```
// PUBLIC INSTANCE METHODS IMPLEMENTING ITimerListener ////////////////////////

void Asteroids::OnTimer(int value)
{
    if (value == CREATE_NEW_PLAYER)
    {
        mSpaceship->Reset();
        mGameWorld->AddObject(mSpaceship);
    }

    if (value == START_NEXT_LEVEL)
    {
        mLevel++;
        int num_asteroids = 10 + 2 * mLevel;
        CreateAsteroids(num_asteroids);
    }

    if (value == SHOW_GAME_OVER)
    {
        mGameOverLabel->SetVisible(true);
    }

}
```

The START_NEXT_LEVEL condition keeps track of the number of levels. So, for my implementation I:



```
if (value == START_NEXT_LEVEL)
{
    mLevel++;

    int num_asteroids = 10 + 2 * mLevel;
    CreateAsteroids(num_asteroids);

    if (mLevel % 3 == 0) {
        CreateLifeUp(1);
    }
}
```

Every 3 levels, a LifeUp will spawn.

The CreateLifeUp(1) in Asteroids::Start() is still there, comment and uncomment as needed.

Second Power Up: Upgrade Bullet Life.

Objectives:

- Implement a power-up (Bullet life extension)

Outline of changes:

- One new Header file and accompanying cpp.
- New sprite
- New definition and method in Asteroids.h and .cpp
- Additions to Spaceship.h and .cpp.

Implementation:

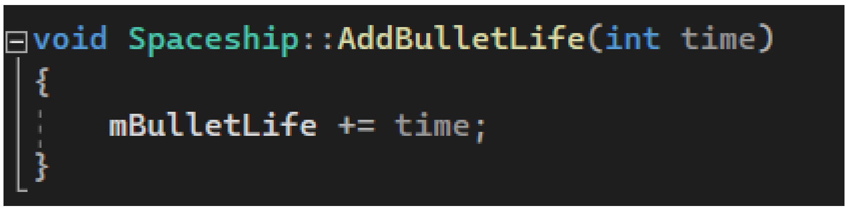
First, I looked into the shoot() method in Spaceship.cpp. I confirmed that when shooting a new bullet is created. So, to start, I added a new variable and method int the header and cpp file for spaceship.

Header:

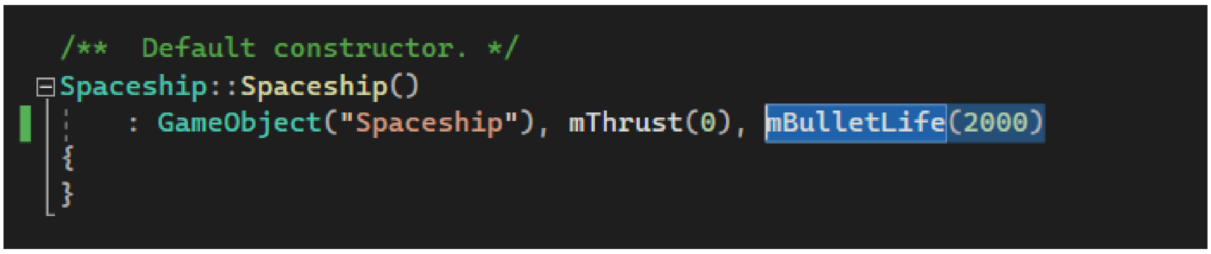
```
int mBulletLife;
```

```
virtual void AddBulletLife(int time);
```

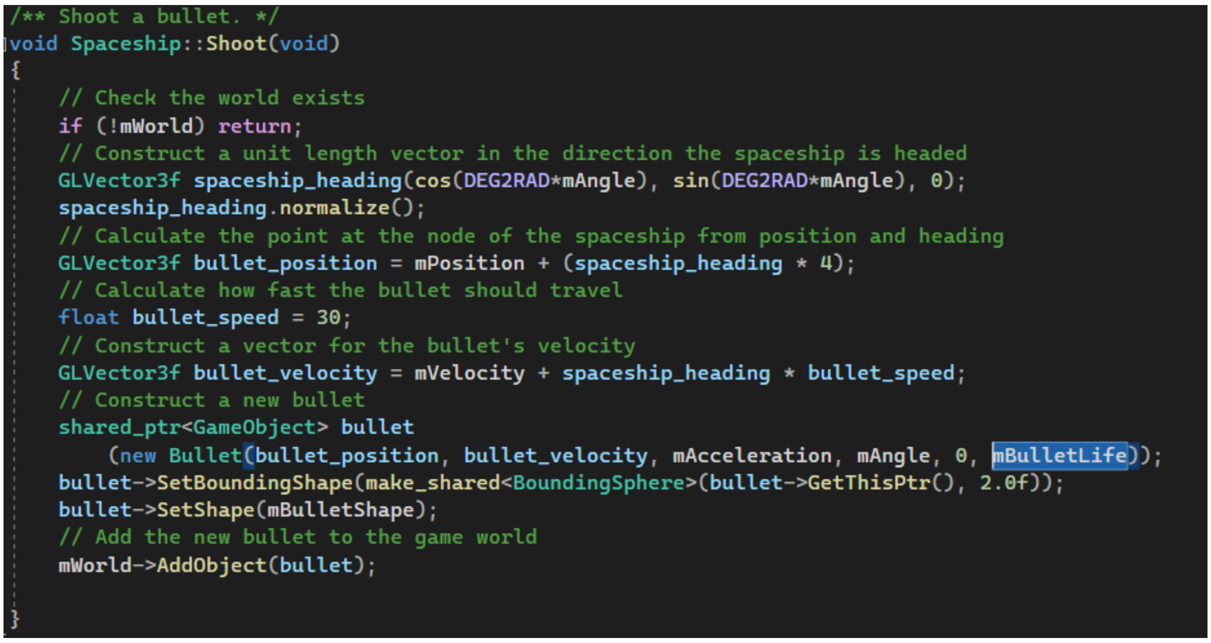
cpp:



```
void Spaceship::AddBulletLife(int time)
{
    mBulletLife += time;
}
```

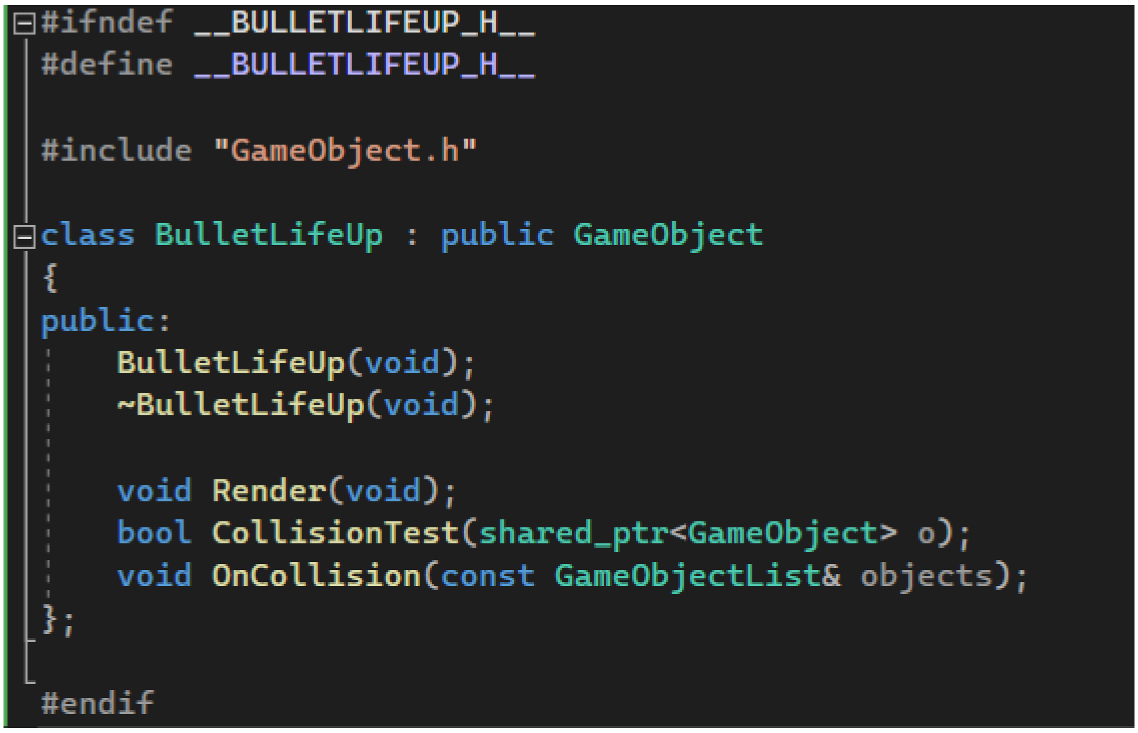


```
/**  Default constructor. */
Spaceship::Spaceship()
    : GameObject("Spaceship"), mThrust(0), mBulletLife(2000)
{
}
```

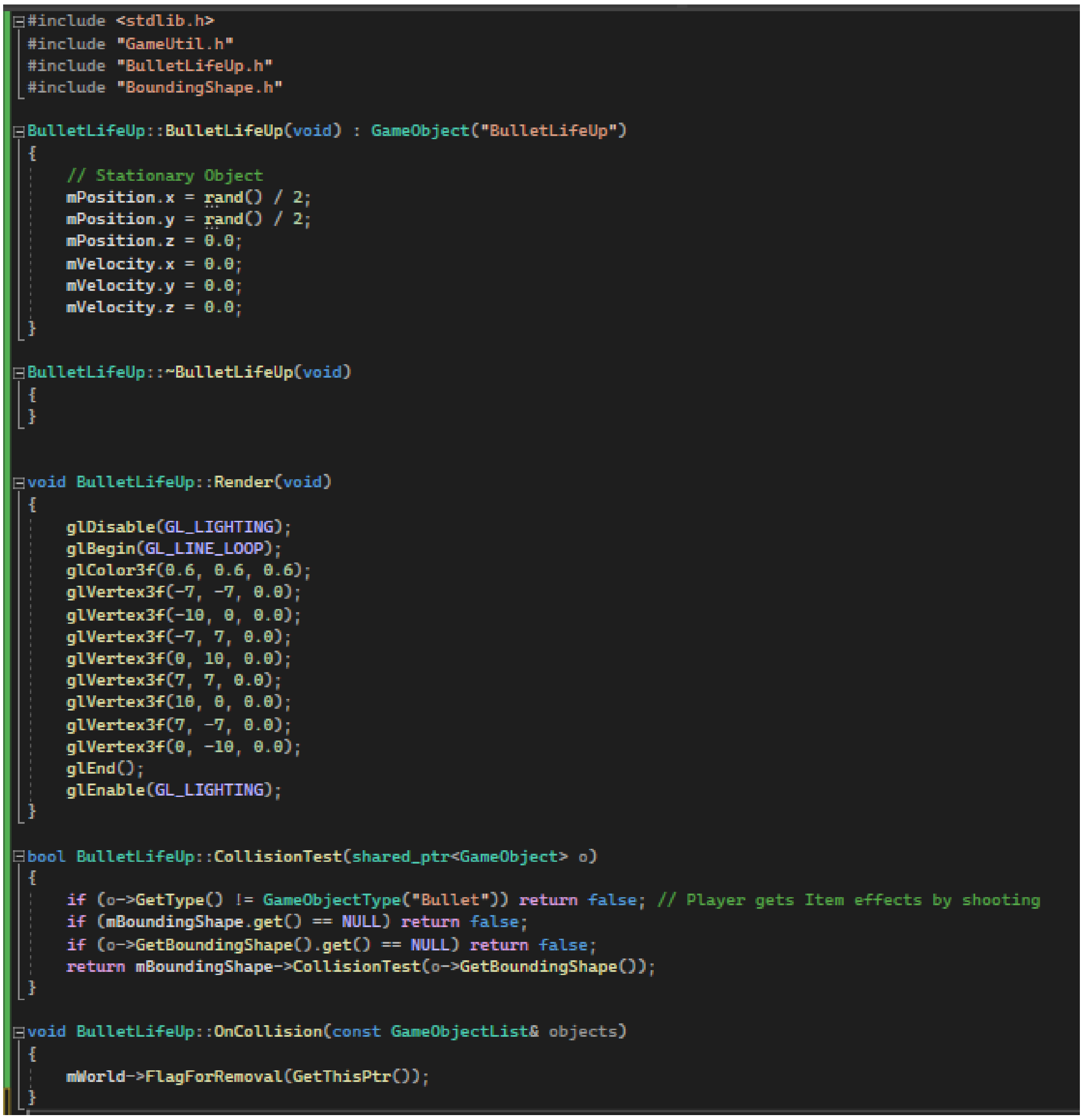


```
/** Shoot a bullet. */
void Spaceship::Shoot(void)
{
    // Check the world exists
    if (!mWorld) return;
    // Construct a unit length vector in the direction the spaceship is headed
    GLVector3f spaceship_heading(cos(DEG2RAD*mAngle), sin(DEG2RAD*mAngle), 0);
    spaceship_heading.normalize();
    // Calculate the point at the node of the spaceship from position and heading
    GLVector3f bullet_position = mPosition + (spaceship_heading * 4);
    // Calculate how fast the bullet should travel
    float bullet_speed = 30;
    // Construct a vector for the bullet's velocity
    GLVector3f bullet_velocity = mVelocity + spaceship_heading * bullet_speed;
    // Construct a new bullet
    shared_ptr<GameObject> bullet
        (new Bullet(bullet_position, bullet_velocity, mAcceleration, mAngle, 0, mBulletLife));
    bullet->SetBoundingShape(make_shared<BoundingSphere>(bullet->GetThisPtr(), 2.0f));
    bullet->SetShape(mBulletShape);
    // Add the new bullet to the game world
    mWorld->AddObject(bullet);

}
```

After that, I created the header and cpp files for the new power up. This was simple as I could just use the life up code as a template.



```
#ifndef __BULLETLIFEUP_H__
#define __BULLETLIFEUP_H__

#include "GameObject.h"

class BulletLifeUp : public GameObject
{
public:
    BulletLifeUp(void);
    ~BulletLifeUp(void);

    void Render(void);
    bool CollisionTest(shared_ptr<GameObject> o);
    void OnCollision(const GameObjectList& objects);
};

#endif
```



```
#include <stdlib.h>
#include "GameUtil.h"
#include "BulletLifeUp.h"
#include "BoundingShape.h"

BulletLifeUp::BulletLifeUp(void) : GameObject("BulletLifeUp")
{
    // Stationary Object
    mPosition.x = rand() / 2;
    mPosition.y = rand() / 2;
    mPosition.z = 0.0;
    mVelocity.x = 0.0;
    mVelocity.y = 0.0;
    mVelocity.z = 0.0;
}

BulletLifeUp::~BulletLifeUp(void)
{
}


void BulletLifeUp::Render(void)
{
    glDisable(GL_LIGHTING);
    glBegin(GL_LINE_LOOP);
    glColor3f(0.6, 0.6, 0.6);
    glVertex3f(-7, -7, 0.0);
    glVertex3f(-10, 0, 0.0);
    glVertex3f(-7, 7, 0.0);
    glVertex3f(0, 10, 0.0);
    glVertex3f(7, 7, 0.0);
    glVertex3f(10, 0, 0.0);
    glVertex3f(7, -7, 0.0);
    glVertex3f(0, -10, 0.0);
    glEnd();
    glEnable(GL_LIGHTING);
}

bool BulletLifeUp::CollisionTest(shared_ptr<GameObject> o)
{
    if (o->GetType() != GameObjectType("Bullet")) return false; // Player gets Item effects by shooting
    if (mBoundingShape.get() == NULL) return false;
    if (o->GetBoundingShape().get() == NULL) return false;
    return mBoundingShape->CollisionTest(o->GetBoundingShape());
}

void BulletLifeUp::OnCollision(const GameObjectList& objects)
{
    mWorld->FlagForRemoval(GetThisPtr());
}
```

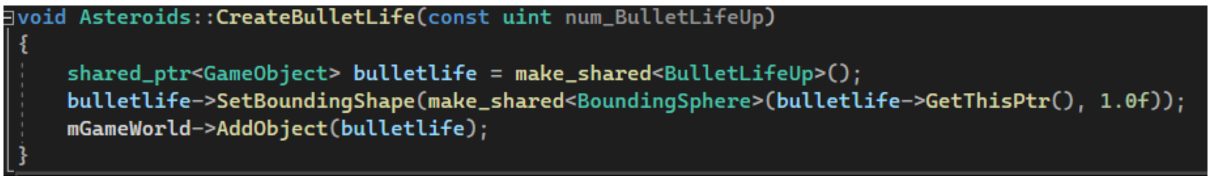
Next step is to include and implement in Asteroids.h and .cpp.

In Header file:

```
void CreateBulletLife(const uint num_BulletLifeUp);
```

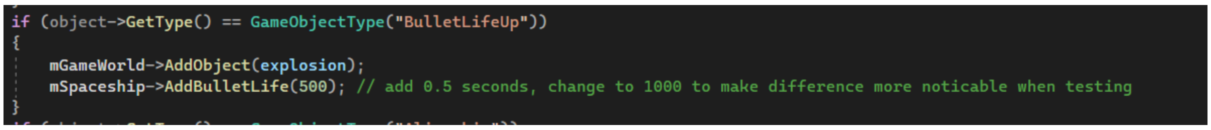
In cpp:

```
#include "BulletLifeUp.h"
```



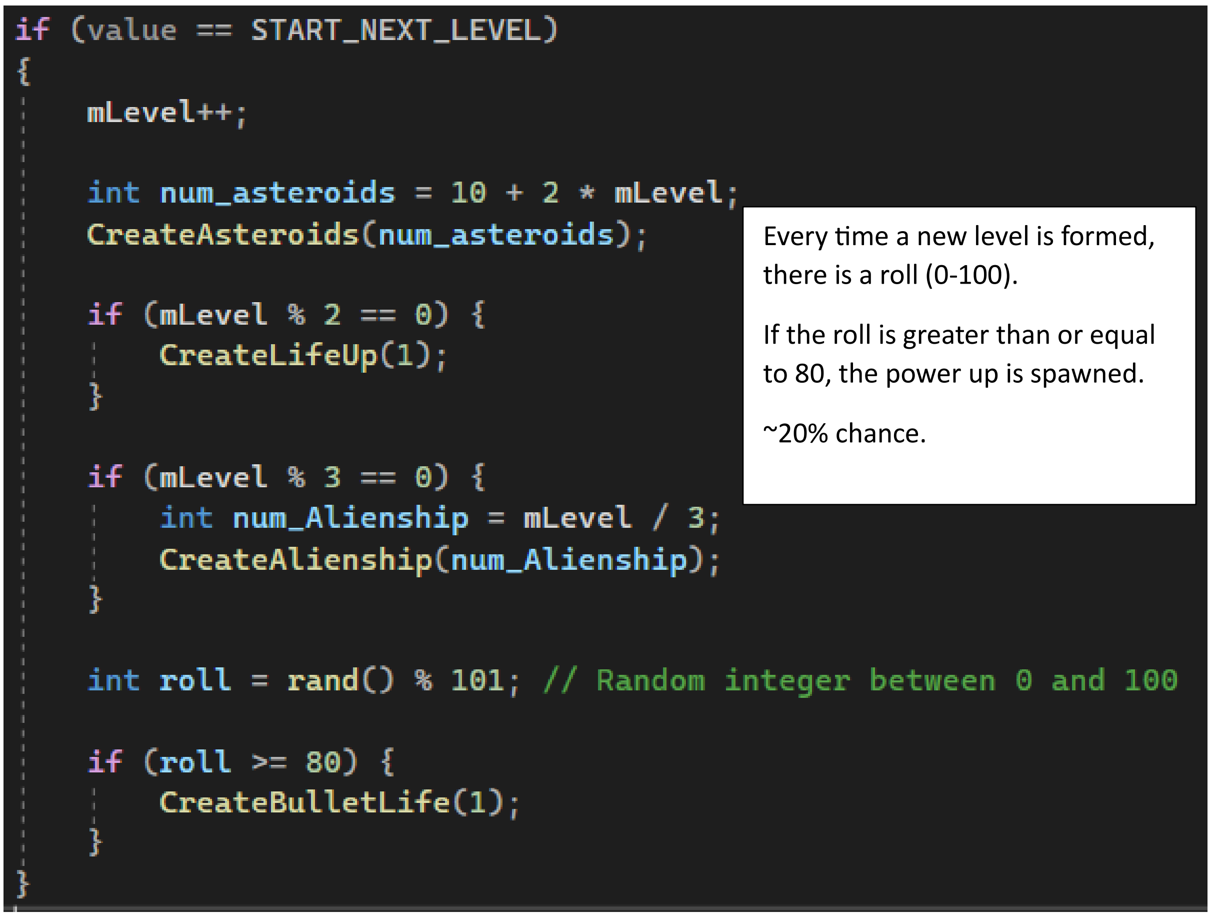
```
void Asteroids::CreateBulletLife(const uint num_BulletLifeUp)
{
    shared_ptr<GameObject> bulletlife = make_shared<BulletLifeUp>();
    bulletlife->SetBoundingShape(make_shared<BoundingSphere>(bulletlife->GetThisPtr(), 1.0f));
    mGameWorld->AddObject(bulletlife);
}
```

OnObjectRemoved():



```
if (object->GetType() == GameObjectType("BulletLifeUp"))
{
    mGameWorld->AddObject(explosion);
    mSpaceship->AddBulletLife(500); // add 0.5 seconds, change to 1000 to make difference more noticable when testing
}
```

For game play implementation, I thought it wouldn't be a good idea to make this power up spawn every x level as it is a permanent statistical change. (Just like before, there is a line that spawns on session start for testing).



```
if (value == START_NEXT_LEVEL)
{
    mLevel++;

    int num_asteroids = 10 + 2 * mLevel;
    CreateAsteroids(num_asteroids);

    if (mLevel % 2 == 0) {
        CreateLifeUp(1);
    }

    if (mLevel % 3 == 0) {
        int num_Alienship = mLevel / 3;
        CreateAlienship(num_Alienship);
    }

    int roll = rand() % 101; // Random integer between 0 and 100

    if (roll >= 80) {
        CreateBulletLife(1);
    }
}
```

Every time a new level is formed, there is a roll (0-100).

If the roll is greater than or equal to 80, the power up is spawned.

~20% chance.

Part 3: Alien Spaceship

Objectives:

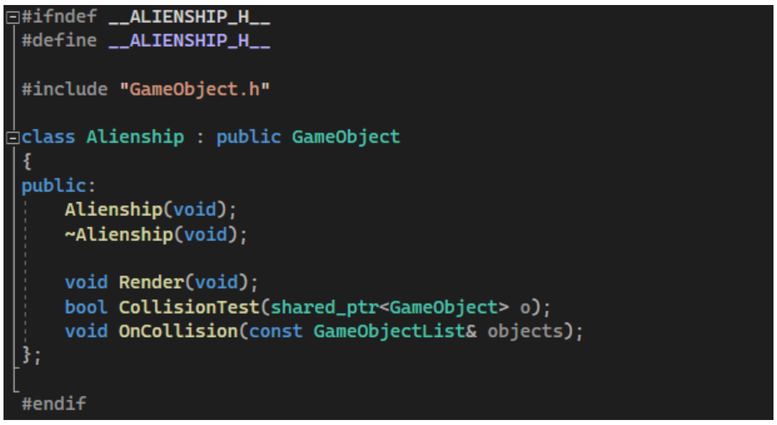
- Add an enemy into the game.
- Implement a computer control system.
- Sprite

Outline of Changes:

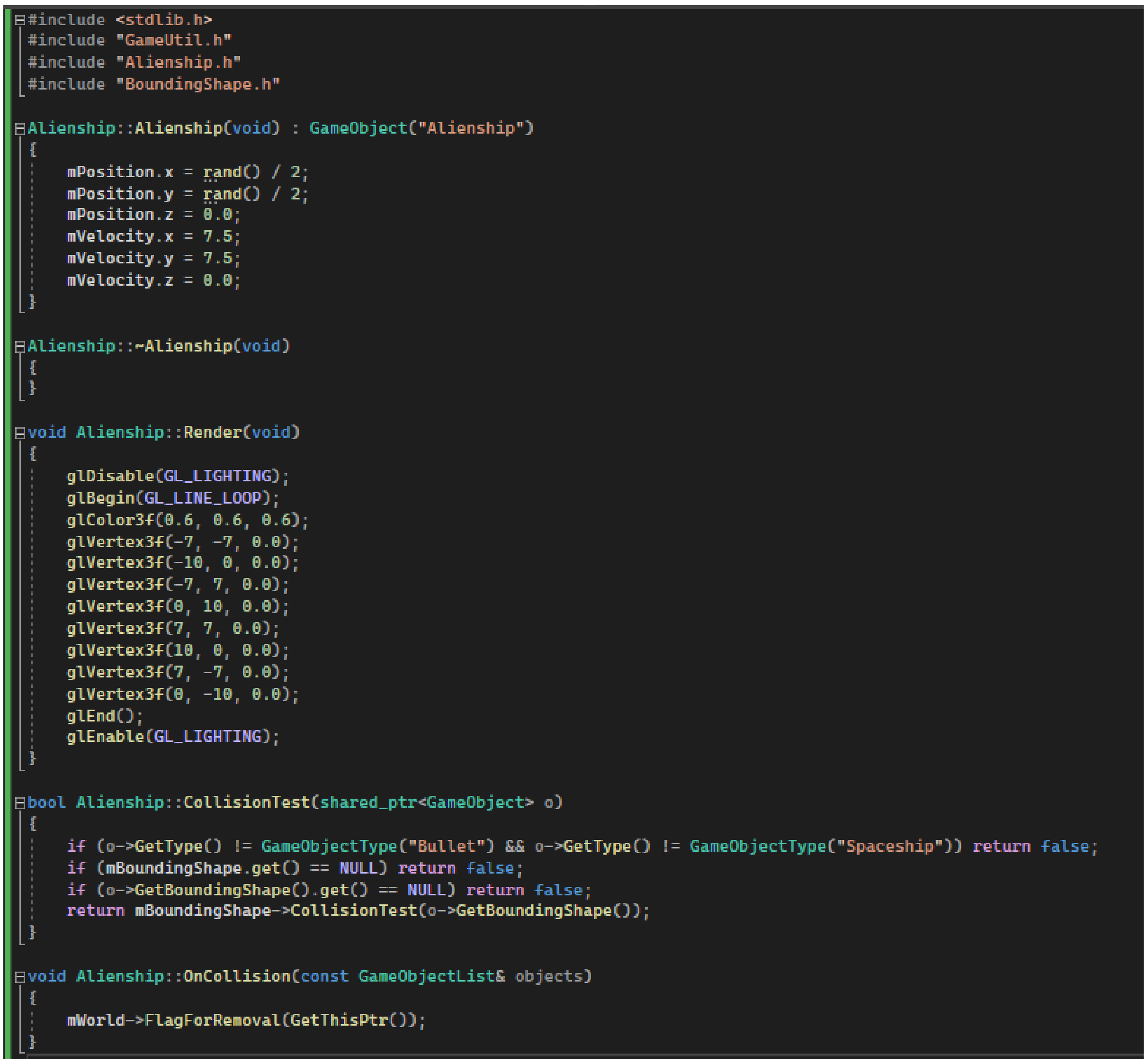
- One new header file and accompanying cpp.
- Sprite

Implementation:

Like the previous task, I started with the header and cpp file for the enemy. The behaviours is similar to asteroids (in terms of collisions), so I used asteroid.h and cpp (as well as tutorial 2, just like the previous task) as a base.



```
#ifndef __ALIENSHIP_H__
#define __ALIENSHIP_H__

#include "GameObject.h"

class Alienship : public GameObject
{
public:
    Alienship(void);
    ~Alienship(void);

    void Render(void);
    bool CollisionTest(shared_ptr<GameObject> o);
    void OnCollision(const GameObjectList& objects);
};

#endif
```



```cpp
#include <stdlib.h>
#include "GameUtil.h"
#include "Alienship.h"
#include "BoundingShape.h"

Alienship::Alienship(void) : GameObject("Alienship")
{
    mPosition.x = rand() / 2;
    mPosition.y = rand() / 2;
    mPosition.z = 0.0;
    mVelocity.x = 7.5;
    mVelocity.y = 7.5;
    mVelocity.z = 0.0;
}

Alienship::~Alienship(void)
{
}

void Alienship::Render(void)
{
    glDisable(GL_LIGHTING);
    glBegin(GL_LINE_LOOP);
    glColor3f(0.6, 0.6, 0.6);
    glVertex3f(-7, -7, 0.0);
    glVertex3f(-10, 0, 0.0);
    glVertex3f(-7, 7, 0.0);
    glVertex3f(0, 10, 0.0);
    glVertex3f(7, 7, 0.0);
    glVertex3f(10, 0, 0.0);
    glVertex3f(7, -7, 0.0);
    glVertex3f(0, -10, 0.0);
    glEnd();
    glEnable(GL_LIGHTING);
}

bool Alienship::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 Alienship::OnCollision(const GameObjectList& objects)
{
    mWorld->FlagForRemoval(GetThisPtr());
}
```

For collisions, the enemy will behave similarly to asteroids. Player must avoid contact whilst trying to destroy via shooting. So, the collisions for bullets and the spaceship had to be updated.

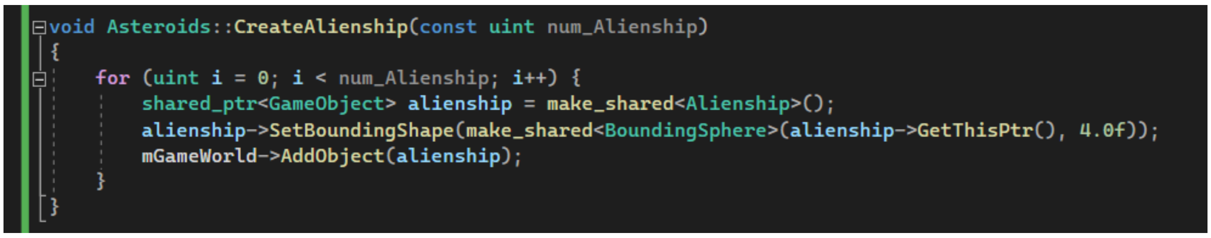
Spaceship: `if (o->GetType() != GameObjectType("Asteroid") && o->GetType() != GameObjectType("Alienship")) return false;`

Bullets: `if (o->GetType() != GameObjectType("Asteroid") && o->GetType() != GameObjectType("LifeUp") && o->GetType() != GameObjectType("Alienship")) return false;`

Next, asteroids.h and .cpp. In the header file:

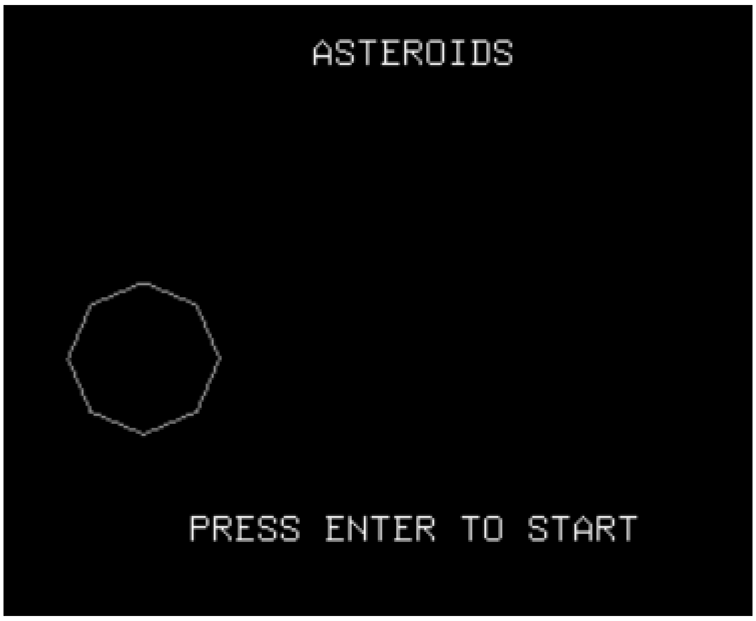
```cpp
void CreateAlienship(const uint num_Alienship);
```

For cpp, include Alienship.h and define the method.

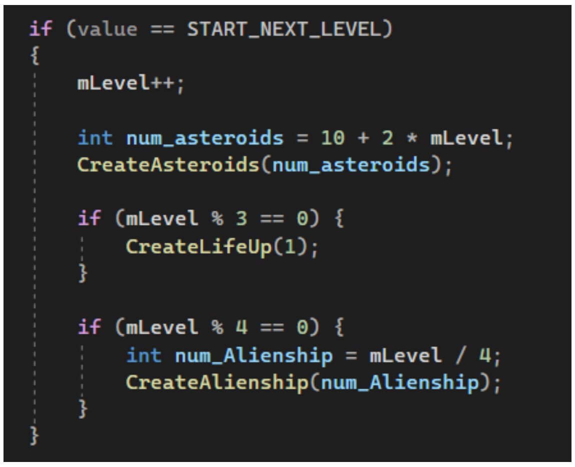


```cpp
void Asteroids::CreateAlienship(const uint num_Alienship)
{
    for (uint i = 0; i < num_Alienship; i++) {
        shared_ptr<GameObject> alienship = make_shared<Alienship>();
        alienship->SetBoundingShape(make_shared<BoundingSphere>(alienship->GetThisPtr(), 4.0f));
        mGameWorld->AddObject(alienship);
    }
}
```

To test I just used the method in Asteroids:Start, just like how I tested the extra lives. (the extra lives creation is commented out)



Since it's not linked to much and can be done easily, I worked on implementing it into the game loop now. I did it the same way; by altering Asteroids::OnTimer:



```
if (value == START_NEXT_LEVEL)
{
    mLevel++;

    int num_asteroids = 10 + 2 * mLevel;
    CreateAsteroids(num_asteroids);

    if (mLevel % 3 == 0) {
        CreateLifeUp(1);
    }

    if (mLevel % 4 == 0) {
        int num_Alienship = mLevel / 4;
        CreateAlienship(num_Alienship);
    }
}
```

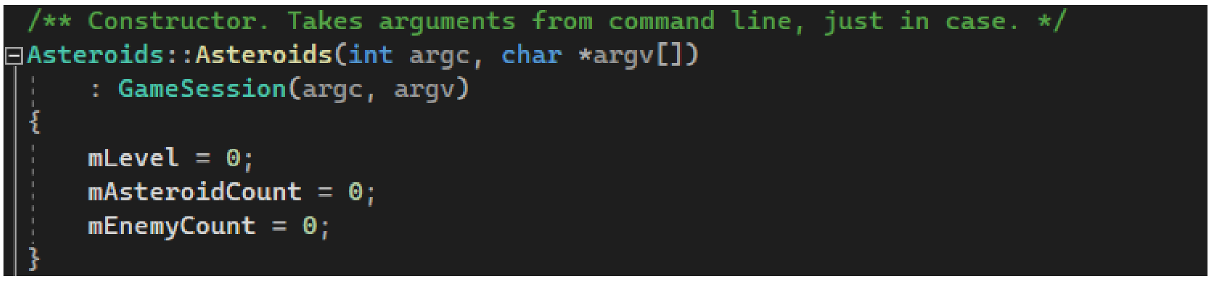
Every 4 levels, the enemy is spawned.

The total number is basically just (number from previous instance) + 1.

Now to make it so that levels with enemies only "end" when all asteroids AND enemies are destroyed.

Add to Asteroids.h: `uint mEnemyCount;`

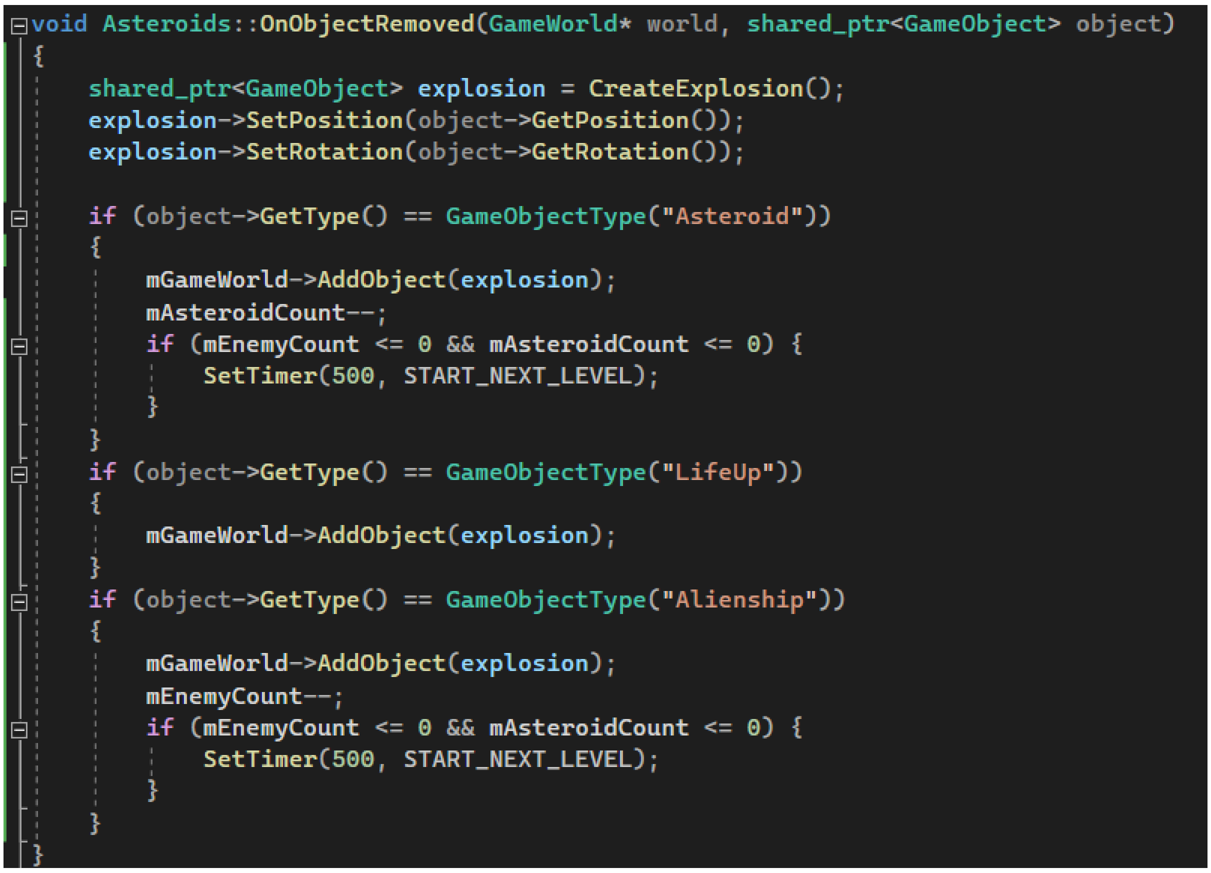
Now all necessary changes in the .cpp file.



```
/** Constructor. Takes arguments from command line, just in case. */
Asteroids::Asteroids(int argc, char *argv[])
    : GameSession(argc, argv)
{
    mLevel = 0;
    mAsteroidCount = 0;
    mEnemyCount = 0;
}
```

In Asteroids::CreateAlienship: `mEnemyCount = num_Alienship;`

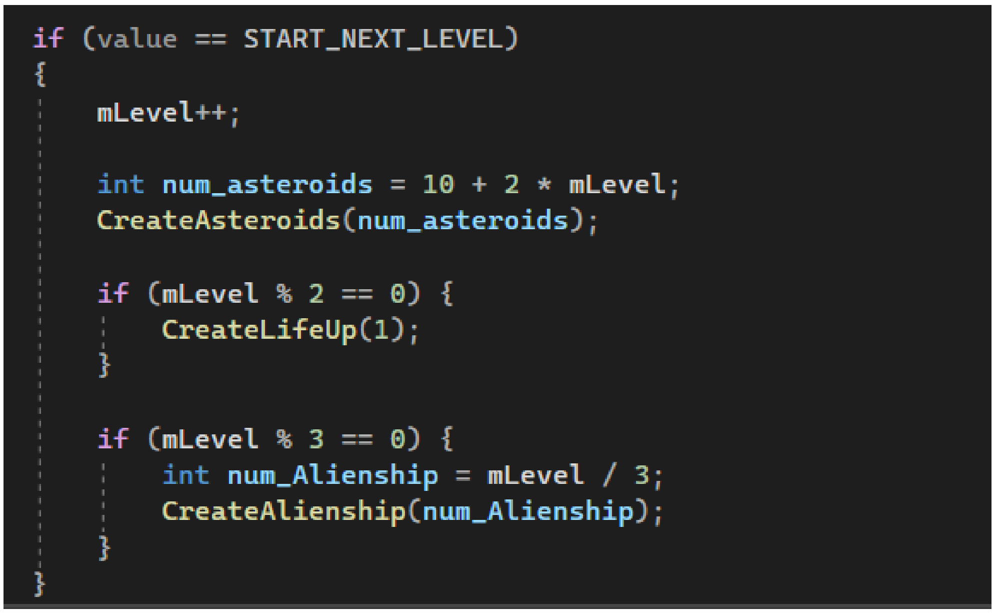
Asteroids::OnObjectRemoved, After a lot of trial and error:



```
void Asteroids::OnObjectRemoved(GameWorld* world, shared_ptr<GameObject> object)
{
    shared_ptr<GameObject> explosion = CreateExplosion();
    explosion->SetPosition(object->GetPosition());
    explosion->SetRotation(object->GetRotation());

    if (object->GetType() == GameObjectType("Asteroid"))
    {
        mGameWorld->AddObject(explosion);
        mAsteroidCount--;
        if (mEnemyCount <= 0 && mAsteroidCount <= 0) {
            SetTimer(500, START_NEXT_LEVEL);
        }
    }
    if (object->GetType() == GameObjectType("LifeUp"))
    {
        mGameWorld->AddObject(explosion);
    }
    if (object->GetType() == GameObjectType("Alienship"))
    {
        mGameWorld->AddObject(explosion);
        mEnemyCount--;
        if (mEnemyCount <= 0 && mAsteroidCount <= 0) {
            SetTimer(500, START_NEXT_LEVEL);
        }
    }
}
```

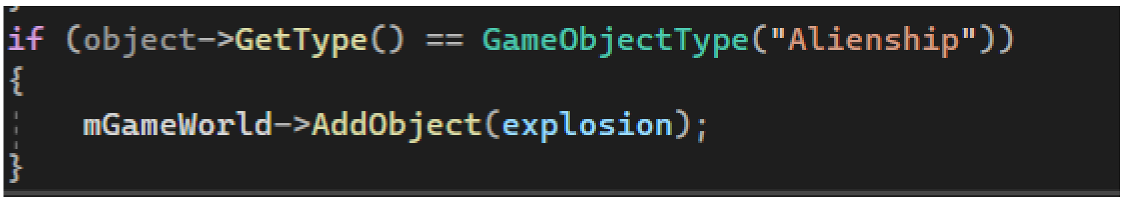
Whilst I was testing, I realised that I made a mistake when altering the level spawns. My intent was every 3 levels for extra lives, and every 4 levels for enemies. However, I forgot that mLevel starts at 0, so my implementation using the mod function makes it so that its: extra lives every 4 levels, enemies every 5 levels.

Alterations to fix this:



```
if (value == START_NEXT_LEVEL)
{
    mLevel++;

    int num_asteroids = 10 + 2 * mLevel;
    CreateAsteroids(num_asteroids);

    if (mLevel % 2 == 0) {
        CreateLifeUp(1);
    }

    if (mLevel % 3 == 0) {
        int num_Alienship = mLevel / 3;
        CreateAlienship(num_Alienship);
    }
}
```

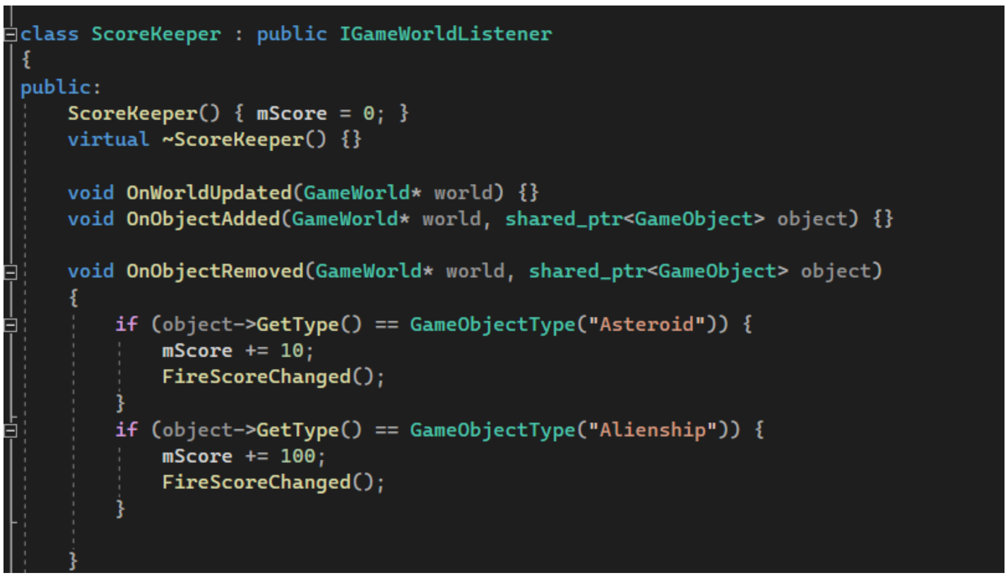
Changes to Asteroids::ObjectRemoved to add explosions.



```
if (object->GetType() == GameObjectType("Alienship"))
{
    mGameWorld->AddObject(explosion);
}
```

For the sprite, I first looked up a free to use sprite, downloaded one and put it in the ASSET folder.

Next, to make destroying enemies adds to the score by adding to scorekeeper.h.



```
class ScoreKeeper : public IGameWorldListener
{
public:
    ScoreKeeper() { mScore = 0; }
    virtual ~ScoreKeeper() {}

    void OnWorldUpdated(GameWorld* world) {}
    void OnObjectAdded(GameWorld* world, shared_ptr<GameObject> object) {}

    void OnObjectRemoved(GameWorld* world, shared_ptr<GameObject> object)
    {
        if (object->GetType() == GameObjectType("Asteroid")) {
            mScore += 10;
            FireScoreChanged();
        }
        if (object->GetType() == GameObjectType("Alienship")) {
            mScore += 100;
            FireScoreChanged();
        }

    }
```

<u>Sprites:</u>

Sprite: lifeup.png (Source: https://opengameart.org/content/heart-pixel-art)

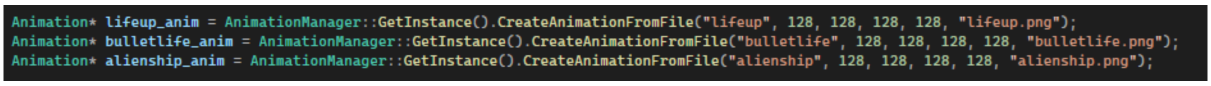
Sprite: bulletlife.png (Source https://opengameart.org/content/missle-bullet)

Sprite: alienship.png (Source: https://www.pngwing.com/en/free-png-ztrlr)

First, I re-sized the images to 128x128 and then removed the backgrounds using free online tools.

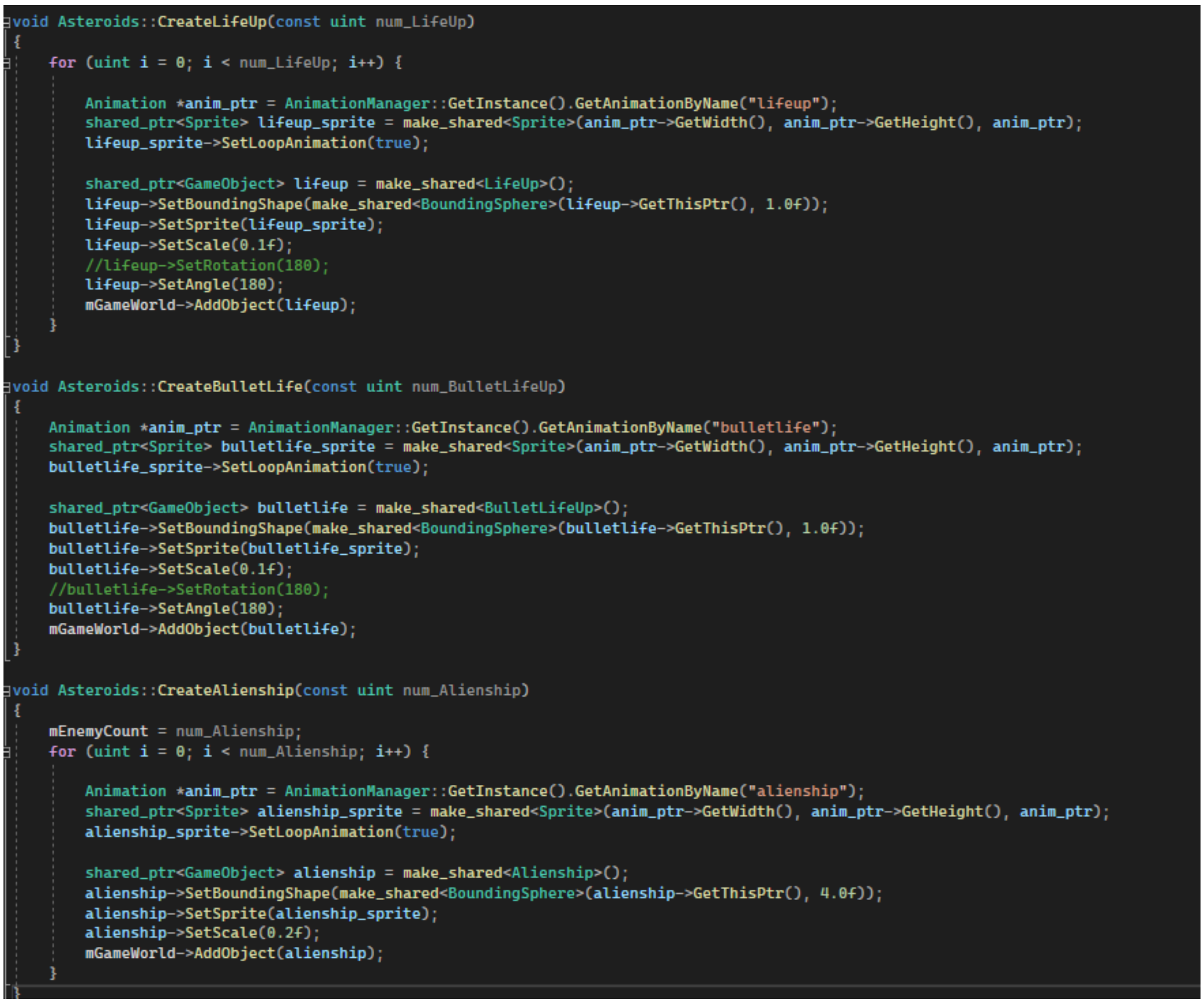
Then I got rid of the Render() method from each powerups .cpp and header file.

In Asteroids.cpp, I first created the instances for each "Animation" in Asteroids::Start().



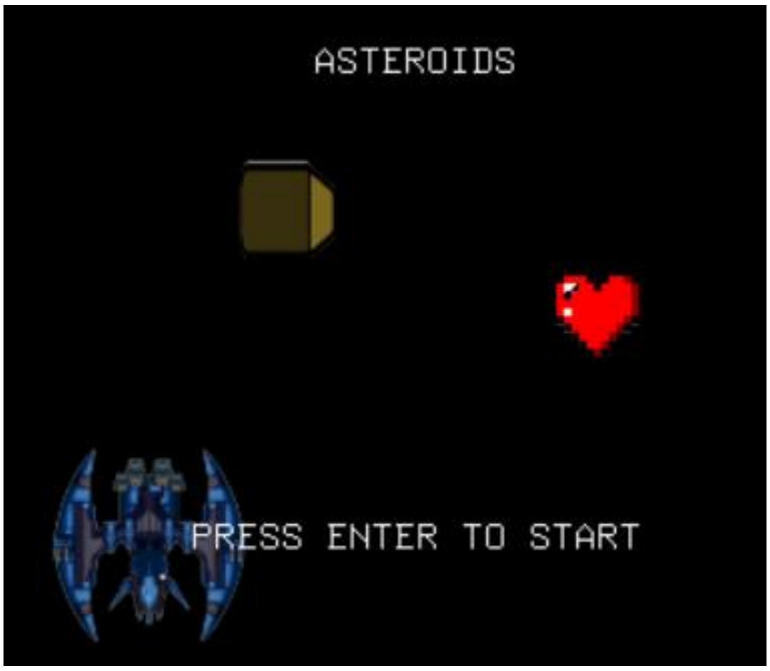
```
Animation* lifeup_anim = AnimationManager::GetInstance().CreateAnimationFromFile("lifeup", 128, 128, 128, 128, "lifeup.png");
Animation* bulletlife_anim = AnimationManager::GetInstance().CreateAnimationFromFile("bulletlife", 128, 128, 128, 128, "bulletlife.png");
Animation* alienship_anim = AnimationManager::GetInstance().CreateAnimationFromFile("alienship", 128, 128, 128, 128, "alienship.png");
```

I then expanded the code in their respective Create methods. Following the blueprint from objects already in the game (asteroids, player ship)



```
void Asteroids::CreateLifeUp(const uint num_LifeUp)
{
    for (uint i = 0; i < num_LifeUp; i++) {

        Animation *anim_ptr = AnimationManager::GetInstance().GetAnimationByName("lifeup");
        shared_ptr<Sprite> lifeup_sprite = make_shared<Sprite>(anim_ptr->GetWidth(), anim_ptr->GetHeight(), anim_ptr);
        lifeup_sprite->SetLoopAnimation(true);

        shared_ptr<GameObject> lifeup = make_shared<LifeUp>();
        lifeup->SetBoundingShape(make_shared<BoundingSphere>(lifeup->GetThisPtr(), 1.0f));
        lifeup->SetSprite(lifeup_sprite);
        lifeup->SetScale(0.1f);
        //lifeup->SetRotation(180);
        lifeup->SetAngle(180);
        mGameWorld->AddObject(lifeup);
    }
}

void Asteroids::CreateBulletLife(const uint num_BulletLifeUp)
{
    Animation *anim_ptr = AnimationManager::GetInstance().GetAnimationByName("bulletlife");
    shared_ptr<Sprite> bulletlife_sprite = make_shared<Sprite>(anim_ptr->GetWidth(), anim_ptr->GetHeight(), anim_ptr);
    bulletlife_sprite->SetLoopAnimation(true);

    shared_ptr<GameObject> bulletlife = make_shared<BulletLifeUp>();
    bulletlife->SetBoundingShape(make_shared<BoundingSphere>(bulletlife->GetThisPtr(), 1.0f));
    bulletlife->SetSprite(bulletlife_sprite);
    bulletlife->SetScale(0.1f);
    //bulletlife->SetRotation(180);
    bulletlife->SetAngle(180);
    mGameWorld->AddObject(bulletlife);
}

void Asteroids::CreateAlienship(const uint num_Alienship)
{
    mEnemyCount = num_Alienship;
    for (uint i = 0; i < num_Alienship; i++) {

        Animation *anim_ptr = AnimationManager::GetInstance().GetAnimationByName("alienship");
        shared_ptr<Sprite> alienship_sprite = make_shared<Sprite>(anim_ptr->GetWidth(), anim_ptr->GetHeight(), anim_ptr);
        alienship_sprite->SetLoopAnimation(true);

        shared_ptr<GameObject> alienship = make_shared<Alienship>();
        alienship->SetBoundingShape(make_shared<BoundingSphere>(alienship->GetThisPtr(), 4.0f));
        alienship->SetSprite(alienship_sprite);
        alienship->SetScale(0.2f);
        mGameWorld->AddObject(alienship);
    }
}
```

Results:



Alienship AI

Whilst I was not able to implement it properly in time for the deadline, I can supply my idea and pseudo-code.

- Use an FSM. Constantly have the enemy check where the player is and update state accordingly.

States: RIGHT, LEFT, UP, DOWN

The current state is determined by the position of the player i.e., if the player is above the enemy, state is shifted to UP which would set the Y velocity of the enemy ship to +- [velocity].

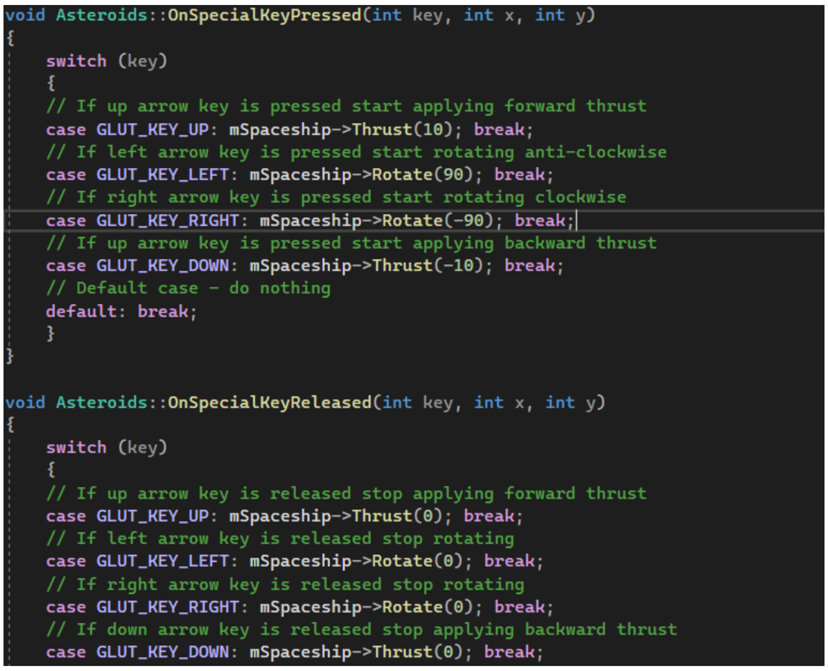
Example code:

```
if (Enemy_Position.x < Player_Position.x) then
      SET Current_State TO UP

^ - This would be required to check if enemy x > player x, also check the Y positions as well.

if (Current_State == '[STATE]') then
      Enemy.Velocity.[x || y] = speed;

(if current state is UP (above) then set y velocity to + 10.f)
```

This would mean that enemies would need to make us of the player listener.

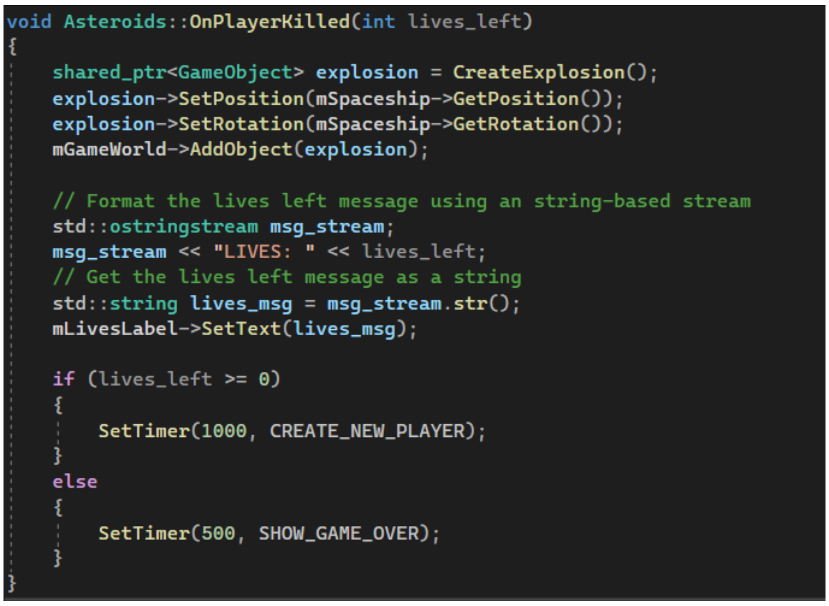
GENERIC CHANGES

- Added implementation for GLUT_KEY_DOWN for special key presses and release.



```
void Asteroids::OnSpecialKeyPressed(int key, int x, int y)
{
    switch (key)
    {
    // If up arrow key is pressed start applying forward thrust
    case GLUT_KEY_UP: mSpaceship->Thrust(10); break;
    // If left arrow key is pressed start rotating anti-clockwise
    case GLUT_KEY_LEFT: mSpaceship->Rotate(90); break;
    // If right arrow key is pressed start rotating clockwise
    case GLUT_KEY_RIGHT: mSpaceship->Rotate(-90); break;
    // If up arrow key is pressed start applying backward thrust
    case GLUT_KEY_DOWN: mSpaceship->Thrust(-10); break;
    // Default case - do nothing
    default: break;
    }
}

void Asteroids::OnSpecialKeyReleased(int key, int x, int y)
{
    switch (key)
    {
    // If up arrow key is released stop applying forward thrust
    case GLUT_KEY_UP: mSpaceship->Thrust(0); break;
    // If left arrow key is released stop rotating
    case GLUT_KEY_LEFT: mSpaceship->Rotate(0); break;
    // If right arrow key is released stop rotating
    case GLUT_KEY_RIGHT: mSpaceship->Rotate(0); break;
    // If down arrow key is released stop applying backward thrust
    case GLUT_KEY_DOWN: mSpaceship->Thrust(0); break;
```

- Game over when player lives reaches negative.



```
void Asteroids::OnPlayerKilled(int lives_left)
{
    shared_ptr<GameObject> explosion = CreateExplosion();
    explosion->SetPosition(mSpaceship->GetPosition());
    explosion->SetRotation(mSpaceship->GetRotation());
    mGameWorld->AddObject(explosion);

    // Format the lives left message using an string-based stream
    std::ostringstream msg_stream;
    msg_stream << "LIVES: " << lives_left;
    // Get the lives left message as a string
    std::string lives_msg = msg_stream.str();
    mLivesLabel->SetText(lives_msg);

    if (lives_left >= 0)
    {
        SetTimer(1000, CREATE_NEW_PLAYER);
    }
    else
    {
        SetTimer(500, SHOW_GAME_OVER);
    }
}
```