IN2026 Coursework Documentation

Mert Ors - ACWW156

Asteroids That Split Into Smaller Asteroids When Hit

I chose to implement asteroids that split into smaller bits. The asteroids turn into smaller asteroids twice and then they disappear in my implementation. The class already had a collision detection for detecting when asteroids get destroyed so all I had to do was to implement a function. To keep track of variables like the size of the asteroid bounding shape and the size of the sprite and how many times the asteroid got destroyed so it would stop splitting I have created variables that are stored in asteroid objects and also gave them initial values for the initial size, scale and times it will split in two.

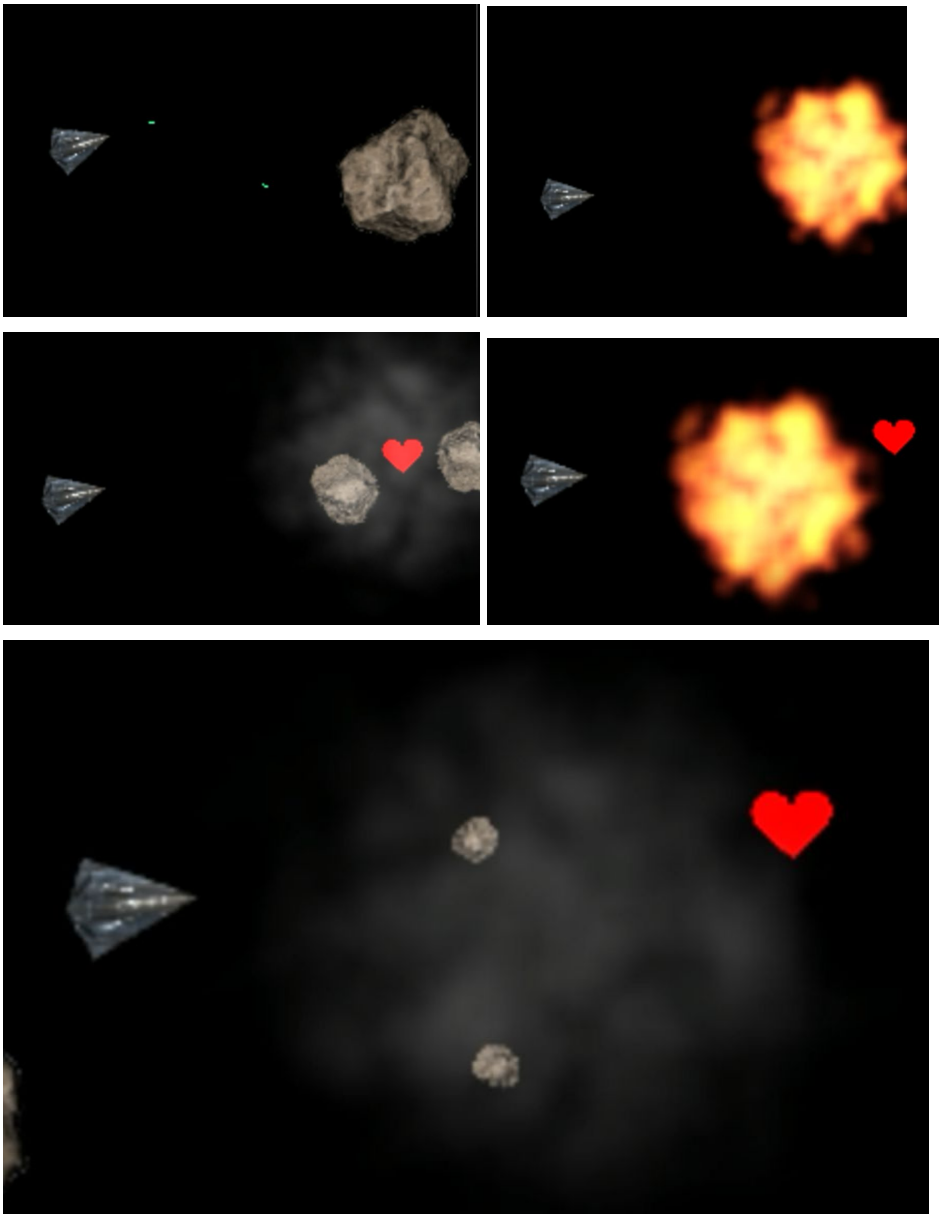
```
//to keep count of previous values for split asteroids
int mCount = 2;
//a member random value because more computational efficiency can be achieved using tricks even though we will use a very little more mem
int mRandom = 0;
//to keep count of previous values for split asteroids
float mScale = 0.1f;
//to keep count of previous values for split asteroids
float mShape = 5.0f;
```

As I have said before there was already a way of tracking collision and so forth to check if the asteroid got destroyed so all I had to do was to create smaller asteroids after the asteroid was destroyed and to accomplish that I have created a function to do that and called it on the collision listener. The function basically creates a new asteroid object and takes the parameters count shape and scale as values from the older asteroid to create the new one and sets them for the new asteroid to half and then sets the location for the asteroid to the location of the previous one. This function only gets called if the mCount is bigger than zero and gets called twice (or can be called in a for loop) for scalability.

```
if (mCount > 0) {
    mCount--;
    AsteroidDestroyed(mScale, mShape, mCount);
    AsteroidDestroyed(mScale, mShape, mCount);
```

```
/** creates an asteroid gets a value scale and shape so we can decide how big our asteroid will be it also gets a value count so it can
keep track of how many times the asteroid got destroyed and sets the asteroids position, rotation and acceleration values to the asteroid
void Asteroid::AsteroidDestroyed(float scale, float shape, int count) {
    //creating the sprite
    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<Asteroid> asteroid = make_shared<Asteroid>();
    //creating a hitbox
    asteroid->SetBoundingShape(make_shared<BoundingSphere>(asteroid->GetThisPtr(), shape));
    //setting the sprite
    asteroid->SetSprite(asteroid_sprite);
    asteroid->SetScale(scale);
    asteroid->mScale = scale/3;
    asteroid->mShape = shape/3;
    asteroid->mCount = count;
    //setting the position of the new asteroids to the old one and the same with rotation and acceleration, left the angle and velocity di
    asteroid->SetPosition(GetThisPtr()->GetPosition());
    asteroid->SetRotation(GetThisPtr()->GetRotation());
    asteroid->SetAcceleration(GetThisPtr()->GetAcceleration());
    // adds the asteroid to the world
    mWorld->AddObject(asteroid);
}
```

I have not set the rotation and acceleration to the previous values as I wanted them to be random (the other way sometimes an asteroid comes on you and you shoot it and keeps coming for you and flying mechanics are a little too flimsy to deal with that). The asteroids splitting can be seen in the screenshots.



Power Ups

I have decided to implement power-ups. Power ups are essentially game objects that add to the gameplay, they are a way of rewarding the player for accomplishing a task in our case destroying asteroids. They add different mechanics to the games and I think a game without power ups would lack flavour so I have implemented them. There are three different power ups in my implementation that all use the Powerup class so I am going to explain them further on but first I would like to explain how my Powerup class works. In my class I just create a basic gameobject and listen for collisions with the spaceship if it gets destroyed our

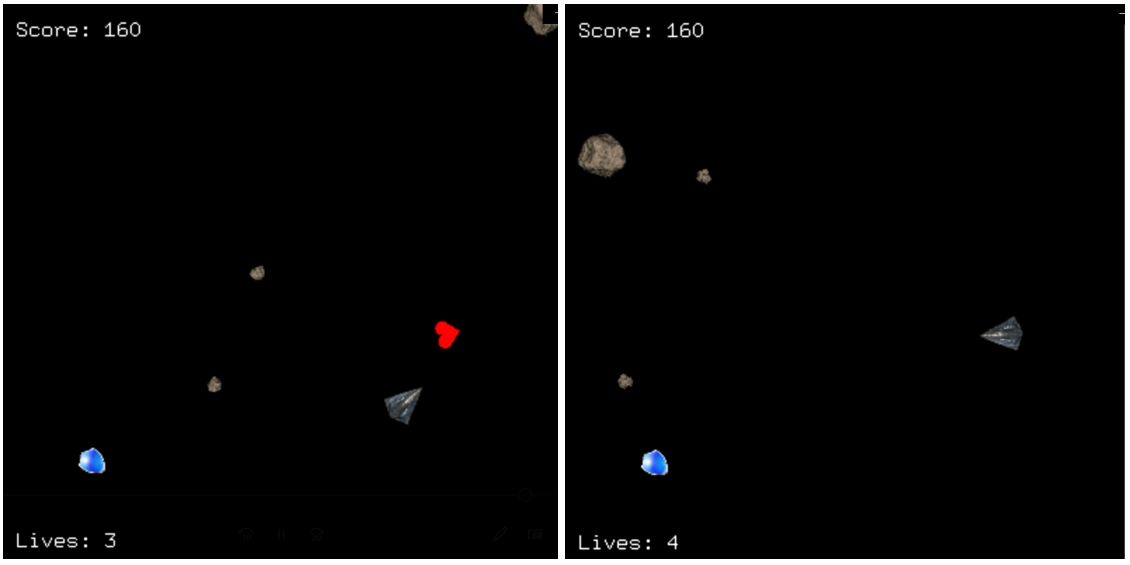
powerups do as well (it gets too messy when we listen to other collisions), I don't add a specific sprite to them or a size and initialize that on creation as it varies on powerup.
I have used a header file here for everything as it is not a big class. You can see that I don't have a variable called type here or a way of storing the type of the powerup as I just use the sizes for the bounding shapes or the scales of their sprites to identify their types.

```
#pragma once

#include "GameObject.h"
#include "PowerUp.h"
#include <stdlib.h>
#include "GameUtil.h"
#include "BoundingShape.h"
#include "AnimationManager.h"
#include "Animation.h"
#include "BoundingSphere.h"


class PowerUp : public GameObject
{
public:
    PowerUp(void) : GameObject("PowerUp"){
        mAngle = rand() % 360;
        mRotation = 0;
        mPosition.x = rand() / 2;
        mPosition.y = rand() / 2;
        mPosition.z = 0.0;
        mVelocity.x = 0.0;
        mVelocity.y = 0.0;
        mVelocity.z = 0.0;
    }
    ~PowerUp(void) {

    }

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

I decided to create power ups whenever asteroids get destroyed and assign them a type and a shape there as well so I have set a random value and used if statements (there are only 3 power ups so there was no need for a switch-case) You can see the function that creates the power ups down there and it just takes a random value as its parameter.

```cpp
void Asteroid::CreatePowerUp(int type) {
    if (type == 1) {
        //setting the sprite
        Animation *anim_ptr = AnimationManager::GetInstance().GetAnimationByName("powerup");
        shared_ptr<Sprite> powerup_sprite
            = make_shared<Sprite>(anim_ptr->GetWidth(), anim_ptr->GetHeight(), anim_ptr);
        powerup_sprite->SetLoopAnimation(true);
        shared_ptr<PowerUp> powerup = make_shared<PowerUp>();
        powerup->SetSprite(powerup_sprite);
        powerup->SetScale(0.025f);
        //setting the hitbox
        powerup->SetBoundingShape(make_shared<BoundingSphere>(powerup->GetThisPtr(), 0.1f));
        //setting the position to where the asteroid was
        powerup->SetPosition(GetThisPtr()->GetPosition());
        // adds the powerup to the world
        mWorld->AddObject(powerup);
    }
    if (type == 2) {
        //setting the sprite
        Animation *anim_ptr = AnimationManager::GetInstance().GetAnimationByName("powerup2");
        shared_ptr<Sprite> powerup_sprite
            = make_shared<Sprite>(anim_ptr->GetWidth(), anim_ptr->GetHeight(), anim_ptr);
        powerup_sprite->SetLoopAnimation(true);
        shared_ptr<PowerUp> powerup = make_shared<PowerUp>();
        powerup->SetSprite(powerup_sprite);
        powerup->SetScale(0.030f);
        //setting the hitbox
        powerup->SetBoundingShape(make_shared<BoundingSphere>(powerup->GetThisPtr(), 0.2f));
        powerup->SetPosition(GetThisPtr()->GetPosition());
        // adds the powerup to the world
        mWorld->AddObject(powerup);
    }
    if (type == 3) {
        //setting the sprite
        Animation *anim_ptr = AnimationManager::GetInstance().GetAnimationByName("powerup3");
        shared_ptr<Sprite> powerup_sprite
            = make_shared<Sprite>(anim_ptr->GetWidth(), anim_ptr->GetHeight(), anim_ptr);
        powerup_sprite->SetLoopAnimation(true);
        shared_ptr<PowerUp> powerup = make_shared<PowerUp>();
        powerup->SetSprite(powerup_sprite);
        powerup->SetScale(0.027f);
        //setting the hitbox
        powerup->SetBoundingShape(make_shared<BoundingSphere>(powerup->GetThisPtr(), 0.2f));
        powerup->SetPosition(GetThisPtr()->GetPosition());
        // adds the powerup to the world
        mWorld->AddObject(powerup);
    }
}
```

One Up



My first power up was a one up! This power up just gives us another life and removes our shields if they are on (as it is too overpowered the other way around) and also updates the health bar every time it is taken. I have implemented this in two different classes. I had to change the Player class to create a oneUp function and I had to call this function and update the health of the player on the gui whenever a power up object was removed as the power ups only collided with the player spaceship so I could use this.

```
void OneUp() {
    mLives++;
}
```

```
if (object->GetType() == GameObjectType("PowerUp")) {

    //whenever the object is a health power up
    if (object->GetScale() == 0.025f) {

        //give the player another live
            mPlayer.OneUp();

            //set the shields for false (I think the other way it was too easy and i would need both a health text and a shields one which takes too much screen space)
            mSpaceship->mShield = false;
            mPlayer.mShield = false;

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

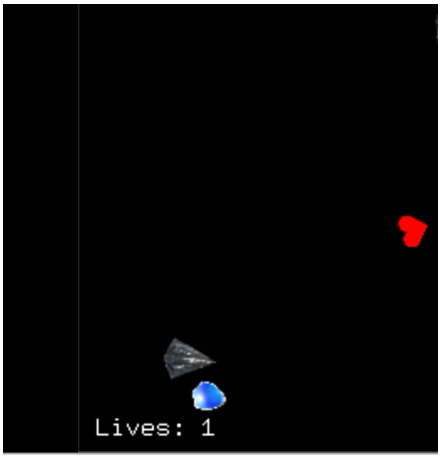
Shields!

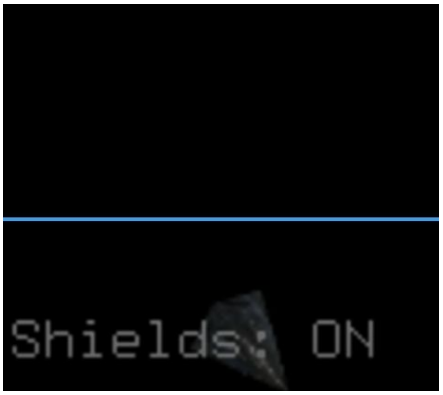
I have implemented shields in the game, the main two differences between a shield and a power up is that you can only keep one shield and when your shield gets broken and your spaceship does not respawn so it keeps its previous location (the second one can be an advantage or a disadvantage regarding the situation but definitely more fun) You also lose your shields when you get a health powerup. I had to extend the Player class for not deducting lives, the spaceship class to avoid removing the spaceship on collision and asteroids to implement the whole functionality for this powerup.

```
//whenever the object is a shield power up
if (object->GetScale() == 0.030f) {
    mPlayer.mShield = true;
    mSpaceship->mShield = true;
    // Format the lives left message using an string-based stream (turns it into Shields ON as I think another text would take too much screen space)
    std::ostringstream msg_stream;
    msg_stream << "Shields: ON";
    // Get the lives left message as a string
    std::string lives_msg = msg_stream.str();
    mLivesLabel->SetText(lives_msg);
}
```

```
void OnObjectRemoved(GameWorld* world, shared_ptr<GameObject> object)
{
    //checks if the collision is with a spaceship
    if (object->GetType() == GameObjectType("Spaceship")) {
        //checks if the shields are not up
        if (!mShield) {
            //if shields are not up you lose a life and die
            mLives -= 1;
            FirePlayerKilled();
        }
        if (mShield) {
            //if shields are up you just live we fire the player killed still but we have tweaked it so that the character does not die
            mShield = false;
            FirePlayerKilled();
        }
    }
}
```

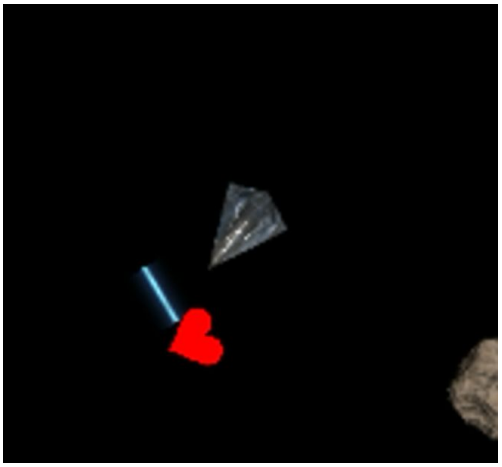
```
void Spaceship::OnCollision(const GameObjectList &objects)
{
    // traverses through the objects if collision is with a bullet an asteroid or the enemy spaceship first checks if the shields are up if not destroys the spaceship
    for (auto object : objects) {
        if (object->GetType() == GameObjectType("Bullet") || object->GetType() == GameObjectType("Asteroid") || object->GetType() == GameObjectType("AlienSpaceship")) {
            if (!mShield) {
                mWorld->FlagForRemoval(GetThisPtr());
            }
            if (mShield) {
                mShield = false;
            }
        }
    }
}
```

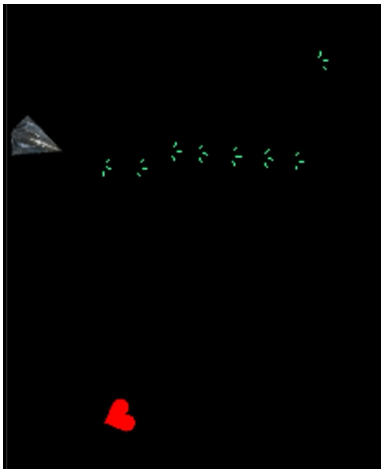




Shooting Upgrade!

The third extension of the powerups is a shooting upgrade, instead of one bullet our spaceship gets to shoot three bullets and that's about it, we keep a variable and set it to true to check if the upgrade is taken and you never lose it! It is quite rare though. To accomplish that I have just created an if statement for the shoot function and created two more bullets inside it that are 60 degrees bent on different sides.





```
//whenever it is a laser power up
if (object->GetScale() == 0.027f) {
    mSpaceship->mSuperShoot = true;
}
```

```
/** 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 * 8);
    // 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, 2000));
    bullet->SetBoundingShape(make_shared<BoundingSphere>(bullet->GetThisPtr(), 2.0f));
    bullet->SetShape(mBulletShape);
    // Add the new bullet to the game world
    mWorld->AddObject(bullet);

    //checks if the shooting powerup is activated
    if(mSuperShoot){
        //creates two new bullets if the powerups is activated with the same position and different angles
        //to give a powerup of three bullets
        shared_ptr<GameObject> bullet
        (new Bullet(bullet_position, bullet_velocity, mAcceleration, mAngle - 60, 0, 2000));
        bullet->SetBoundingShape(make_shared<BoundingSphere>(bullet->GetThisPtr(), 2.0f));
        bullet->SetShape(mBulletShape);
        mWorld->AddObject(bullet);
        shared_ptr<GameObject> bullet2
        (new Bullet(bullet_position, bullet_velocity, mAcceleration, mAngle + 60, 0, 2000));
        bullet2->SetBoundingShape(make_shared<BoundingSphere>(bullet2->GetThisPtr(), 2.0f));
        bullet2->SetShape(mBulletShape);
        mWorld->AddObject(bullet2);

    }
}
```

Alien Spaceship

I have decided to make an alien spaceship that attacks us and follows us around, to do that I have created a class to get instances of. I have created a function inside the alien spaceship class to make it do its alien spaceship activities called shoot (it was intended for just shooting but then ended up being used for path finding which i wasn't planning to implement). This function gets called every 1000 timely thing and it makes our alien spaceship follow us around and shoot at us every now and then. I think using time for following mechanics was a good idea because it gives the alien spaceship a more natural vibe.

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

    if (value == START_NEXT_LEVEL)
    {
    mLevel++;
    // made the game a bit easier since the splitting asteroids make stuff really hard
    int num_asteroids = 2*mLevel;
    CreateAsteroids(num_asteroids);
    //readds the alien to the world on the next level
    mGameWorld->AddObject(mAlien);
    }


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

    //shoots and recursively sets a timer to shoot again for the alien
    if  (value == SHOOT)
    {
        mAlien->Shoot(mSpaceship);
        SetTimer(1000, SHOOT);
    }

}
```

I have called this function recursively from itself to make the spaceship constantly follow us around and shoot. Everything else is not as interesting as they are just setting up sprites velocity etc. but I will also mention the shoot function with more detail here.

```
/** Shoot a bullet. */
void Shoot(shared_ptr<GameObject> o)
{
    //to give a randomised feeling to the shooting I only make 8 out of 10 bullets to actually shoot
    mRandom = rand() % 11;
    if (mRandom <= 8) {
        // Check the world exists
        if (!mWorld) return;

        auto SpaceShipPos = o->GetPosition();

        GLVector3f diff = SpaceShipPos - mPosition;


        // Construct a unit length vector in the direction the spaceship is headed
        GLVector3f spaceship_shooting = diff;
        spaceship_shooting.normalize();
        // Calculate the point at the node of the spaceship from position and heading
        GLVector3f bullet_position = mPosition + (spaceship_shooting * 10);
        // Calculate how fast the bullet should travel
        float bullet_speed = 40;
        // Construct a vector for the bullet's velocity
        GLVector3f bullet_velocity = mVelocity + spaceship_shooting * bullet_speed;
        // Construct a new bullet
        // Set the angle here to a randomised value because otherwise it is impossible to play against
        shared_ptr<GameObject> bullet
        (new Bullet(bullet_position, bullet_velocity, mAcceleration, rand() % 120, 0, 2000));
        bullet->SetBoundingShape(make_shared<BoundingSphere>(bullet->GetThisPtr(), 2.0f));
        bullet->SetShape(mBulletShape);
        // Add the new bullet to the game world
        mWorld->AddObject(bullet);

        //also follows the enemy (I mean it is a part of shooting)
        // it is diff/3 so it slows down as it gets closer rather than a constant slow value which gives it a more realistic way of
        // coming its way to hunt us down
        mVelocity = diff/3;
    }

}
```

I have created the vector variable diff and made it equal to the difference between our and alien spaceships position, then I have created bullets every time with the position pointing towards the friendly spaceship (I had to use pen and paper for this and tried to use tan of the vector but it didn't look as smooth so went back to this idea).
I have also set the velocity of the alien spaceship to diff/3. It is a difference over three instead of the exact same difference so it gives the alien spaceship a natural slowing down effect when it gets closer to us.