Documentation

Games Technology

Name: Ioannis Katsikis

Student number: 230019348

1. List of all features implemented

For Part 1 I decided to take the physics/collisions approach with Option B. All features listed have been implemented which includes the following:  
Asteroids that “bounce” off each other when they collide.

Get destroyed and destroy the spaceship when they collide with said spaceship.

Split into smaller asteroids when hit by a bullet and the smaller asteroids get destroyed when hit by a bullet but also bounce when they collide with another asteroid regardless of size.

Score chances depending on which type of asteroid has been hit by a bullet and no score is awarded if the ship collides with an asteroid.

For Part 2 I decided to implement:

Extra lives

Invulnerability for a limited time (20 seconds in this case)

Upgrades to spaceship control. I also bundled that with faster bullets to make shooting easier over longer ranges which also partially ties back to \*\*weapons upgrades as mentioned in the coursework documentation.

There are visual cues on the game for the player to know if the spaceship is temporarily invincible or has been upgraded instead of writing it on the GUI.

2. List of all methods that have changed, new methods, new classes, and the source files where each of these are contained.

To do the powerups I added a class for each one. I added an ExtraLife.cpp and .h for the extra lives choice, Shield.cpp/.h for the temporary invincibility and Upgrade.cpp/.h for the ship upgrades. These 3 classes are repeated classes of Asteroid so for the most part they all share the same code, minus individual tweaks to collision detection between all objects. By themselves these classes cannot do anything, the way it works is that Asteroids registers their destruction and changes game state or behaviour from there.

The smaller asteroids also have their own unique class SmallAst.cpp/.h which shares the same methods as the Asteroid class so they will have for the most part same behaviour and methods apart from tweaks to collision detection

These 4 classes have their own constructors/destructors and the following methods:

void OrderDestruction();

bool CollisionTest(shared\_ptr<GameObject> o);

void OnCollision(const GameObjectList& objects);

The smaller asteroids also have their own unique class SmallAst.cpp/.h which shares the same methods as the Asteroid class so they will have for the most part same behaviour and methods apart from tweaks to collision detection to prevent incorrect destruction of objects. SmallAst however has a modified constructor:

SmallAst(int xPos, int yPos);

So that it spawns in some predetermined coordinates which are near the parent Asteroid.

The Asteroids class has had a lot of methods added to it. (methods which have been commented out indicate that in running they are not used directly but are accessed by the SpawnPowerup() method)

Void OnLifeChanged(int i);

void CreateSmallAst(shared\_ptr<GameObject> parent, const uint num);

void SpawnPowerup();

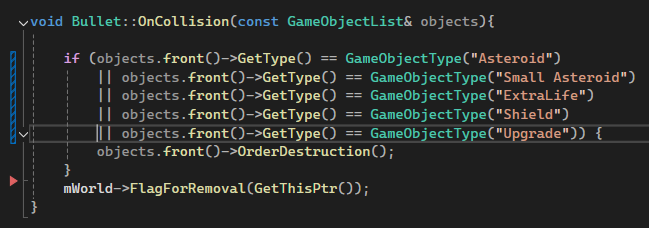
//void CreateUpgrade();

//void CreateExtraLife();

//void CreateShield();

const static uint INVINCIBLE\_TIMER = 3;

The Bullet class has had some modifications done to it, primarily when it detects that it has come in contact with another object, it runs their OrderDestruction method to delete said object. I will explain why I did it this way further in the documentation.



A

For Player.h the constructor has changed, now considers if the player is temporarily invincible or not. Has two new methods:

void increaseLives(int i)

int getLives().

These methods work in tandem with the ExtraLives powerup.

ScoreKeeper.h has had the OnObjectRemoved() method modified to take into account if a small or large asteroid has been destroyed and if the player has been killed.

Spaceship.cpp/.h has had the most methods added to it:

virtual void RightThrust(float t);

virtual void LeftThrust(float t);

virtual void Reverse(float t);

virtual void FastShoot(void);

bool isInvincible;

bool isUpgraded;

void invinciblePupgraded();

const string& chooseBullet();

void applyUpgrades();

void stripUpgrades();

void makeInvincible();

void makeNOT();

void FastThruster();

void NormalThruster();

An id integer was also added to GameObject.h as an attempt to get collisions working. I assumed that the reason that objects of the same type wouldn’t collide was because they both ran the OnCollision method upon contact and cancelled out the effects of the velocity switch that was implemented. I hadn’t noticed that “if (GetType() == o->GetType()) return false;” had been added to the CollisionTest method above, even though I had been looking for it all throughout the Engine as that was the code snipped I theorized could have also been causing issues.

3. Explanation of the implementation of each feature.

Part I

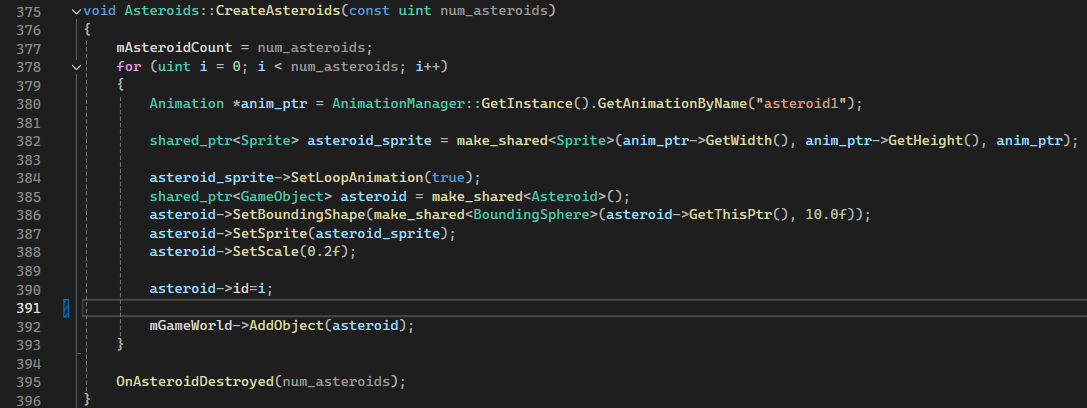
-Asteroids “bouncing” off each other;

My implementation for this is quite simple, when two objects of GameType collide which in this case are Asteroid object, I swap their mVelocity.x and mVelocity.y like so:



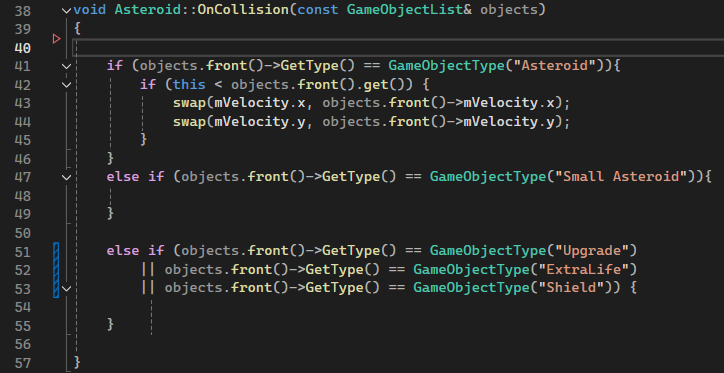
B

You will notice that for Small Asteroid there is also a use of my ID system that I added to the GameObject class that I mentioned earlier. The Asteroids class assigns each Asteroid object an ID sequentially so that when two objects of the same type collide, they can compare IDs first and the one with the smallest one is the one to go ahead and swap velocities, to stop both

objects from swapping so they don’t cancel out the effect. 

C

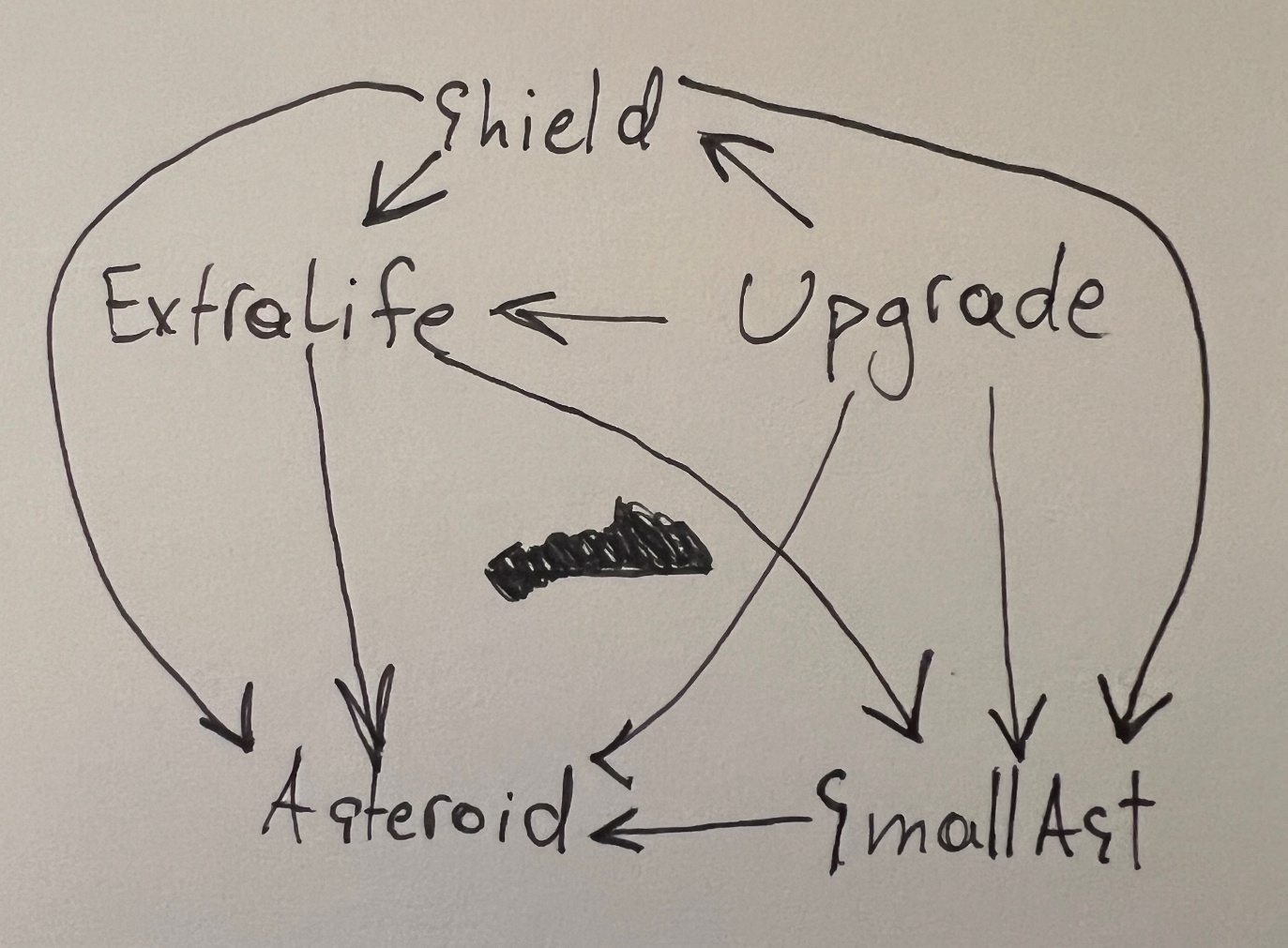
In line 390 you can see where the asteroid is assigned it’s ID and how it’s equal to the current iteration of the loop. All IDs by default are set to 0 and are changed in here. It wasn’t necessary in the end however as when I discovered the line that was preventing my progress I decided to try checking via memory addresses.



D

Both approaches (ID system, memory address comparison) work and I have implemented both, the latter of which you can see in line 42.

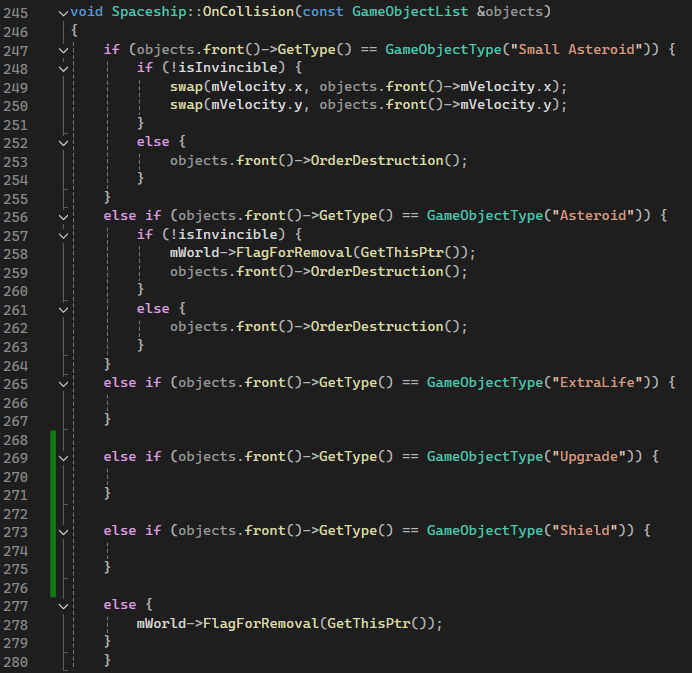
You may have noticed that other objects are being mentioned as well, in particular the powerups but nothing happens in the else if statement. This is so that they don’t get destroyed when they come in contact with the object in person. All the collision checks and calculations have been modelled on paper and shown below:



E

Here you can see which object handles the velocity modification and destruction per combination. In the photos above you can see how the powerups have been added and that they don’t do anythingthat is because the powerups handle the calculations as you can see in the photo above.

-Asteroids getting destroyed and destroying the spaceship, when they collide with the spaceship;

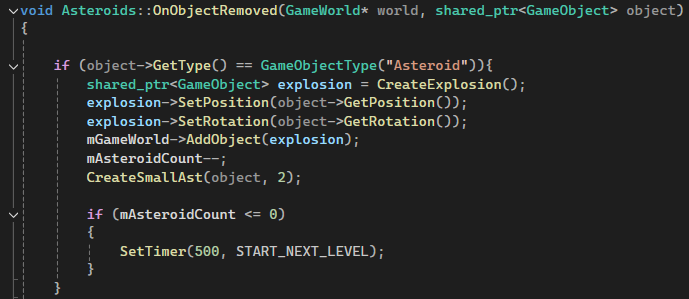


F

In the photo above, line 256-264 you can see how the spaceship runs the OrderDestruction() method of the Asteroid() class after flagging itself for removal.

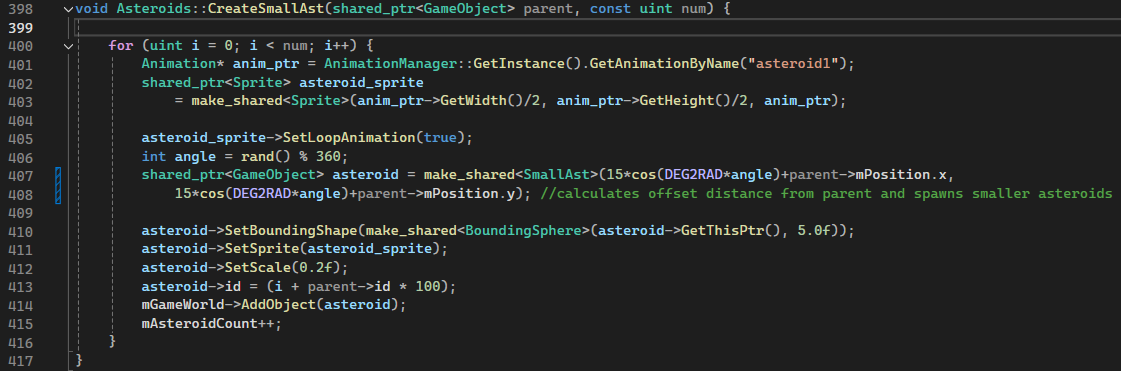
-Asteroids that split into smaller asteroids when hit by a bullet. The smaller asteroids get destroyed when hit by a bullet, but “bounce” when they collide with another asteroid.

The OnObjectRemoved method determines if an Asteroid has been destroyed and then runs the CreateSmallAst method with input parameters the parent object that was destroyed and the amount of smaller asteroids to make.



G

The CreateSmallAst method has largely been modelled after the CreateAsteroids method above with the notable exception of lines 407-408 where the small asteroid constructor is run and input parameters as spawn coordinates are used. In this case the coordinates of the parent are input and normalization is used to spawn the asteroids a certain distance away from the parent.

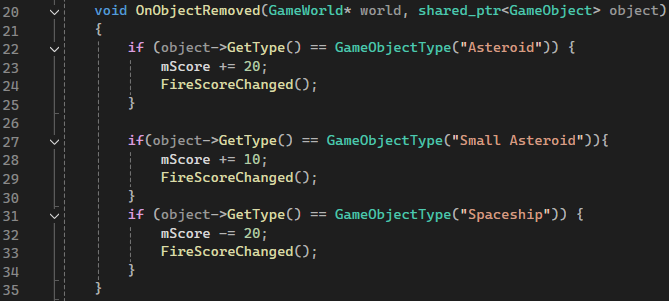


H

The bullet destroys the small asteroid the same way as the spaceship destroys the large asteroid through the OrderDestruction() method, as seen in screenshot F. For collisions the smaller asteroid doesn’t do any of the velocity calculations, the spaceship object does although it has not been added to the image as shown in E.

-The score for hitting a large asteroid with a bullet should be different from the score for hitting a small asteroid. No score should be awarded if the player collides with an asteroid;

This is handled by the ScoreKeeper header where it considers if the asteroid is a large or a smaller one and adjusts the overall score accordingly. The large asteroids give the player more points as they are a threat to the player, whilst the smaller asteroids do not pose any threat. The FireScoreChanged() method is then used to update the score on the player’s screen.



J

In the image above you can also see that when the spaceship is found to have been destroyed, the score is immediately reduced by the amount of points that an asteroid gives, this is because the large asteroid is the only thing that can eliminate the player. Also when the player dies, the asteroid he collides with is also destroyed, this way if the score is reduced by the amount immediately earned, the score remains the same overall.

Part II

-Extra lives