University of Reading
School of Mathematical, Physical and Computational Sciences
Module Code:CS2JA16
Student Number:27020363
Finish date:18/01/2020
Hours Spent on the Assignment : 18 hrs

Major assignment 1 : Gui version

Abstract

In the following assessment , we would attempt to create a simplistic 3D simulator of drones ,projected into a 2D plane, written in java 13 and openfx 2.0 (an open javafx implementation)

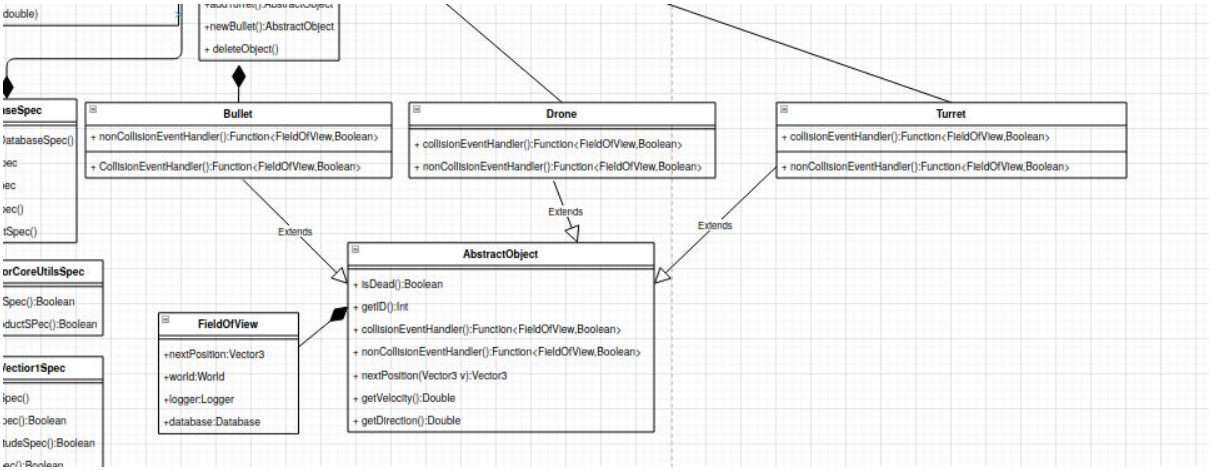
Introduction

In this assignment we will attempt to create a pseudo 3D drone simulator , projected into a 2D plane.a lot of care has been given to the structural hierarchy as well as in the collision system. We will use a grid system to detect collisions ahead of time with extreme accuracy. We will use our custom Vector classes throughout the game as well as we will make the drones rotation to depend on their speed and direction using a little bit of trigonometry to achieve that.Of course, extra care will be given to efficiency , using advanced structures such as hashmaps to achieve that. Additionally , extensive use of Stream api is expected as well as Threading Support. Finally an extensive testing facility is expected to be developed in order to ensure the integrity of our system

OOP Design

Due to the strict size limit of 10 pages for this report , we would have a look in some important areas of interest in our design . Although further reading can be done in the included javadoc , as well as in the full uml diagram in the appendix A at the bottom of this report

1:The AbstractObject class cluster



Our brief analysis starts from the AbstractObject class. This is an abstract class that is inherited from every drawable entity of this simulator . As we can see there are 3 concrete implementations , named Bullet , Drone and Turret.A brief explanation of each is given below

Drone

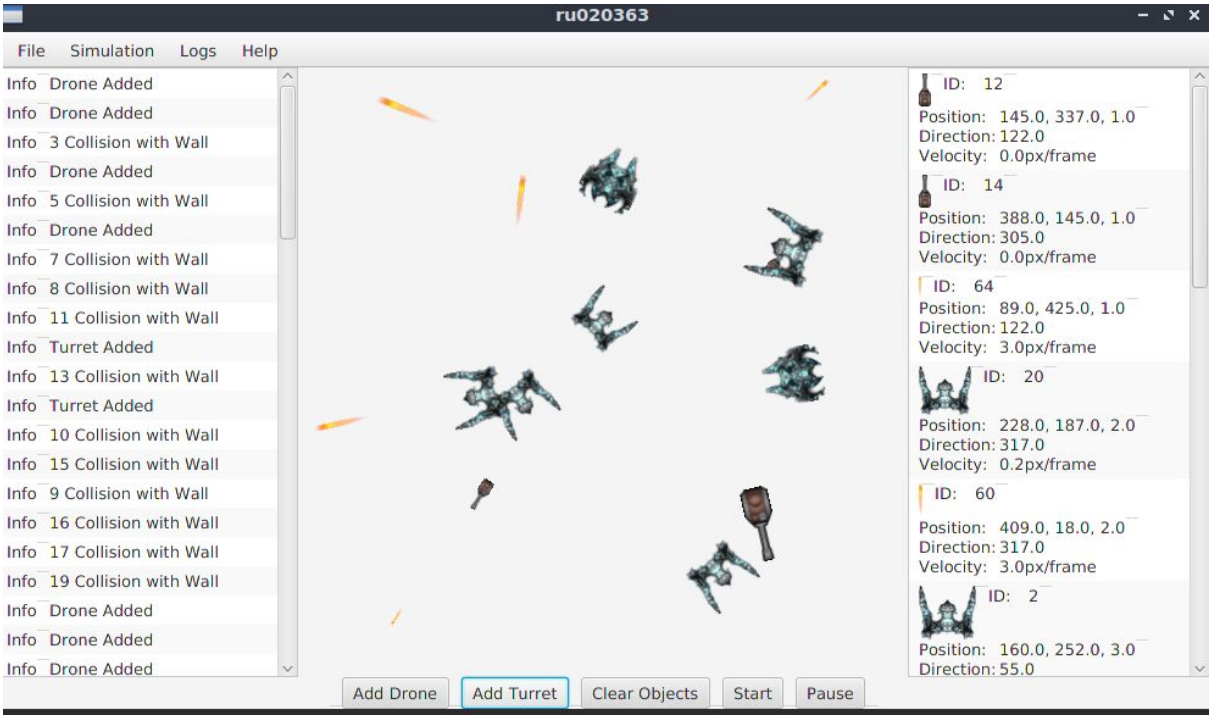
This is the first concrete class , it represents one of The 3 drones given at the right of this page.
The selection for the representation for each drone Is happened at the initialization based on a random number . each drone has velocity and direction , and the physics engine calculates their next location in each frame. Each drone is operating at different heights (thus the pseudo 3D projected at the 2D plane). To represent height , each drone has 5 different sprites , with different size to represent height.Each drone moves randomly by the plane and shoots with approximate 10% probability per frame.It also has 10 lives, which can lose either by bumping to another AbstractObject or by the limits of the World.

Bullet

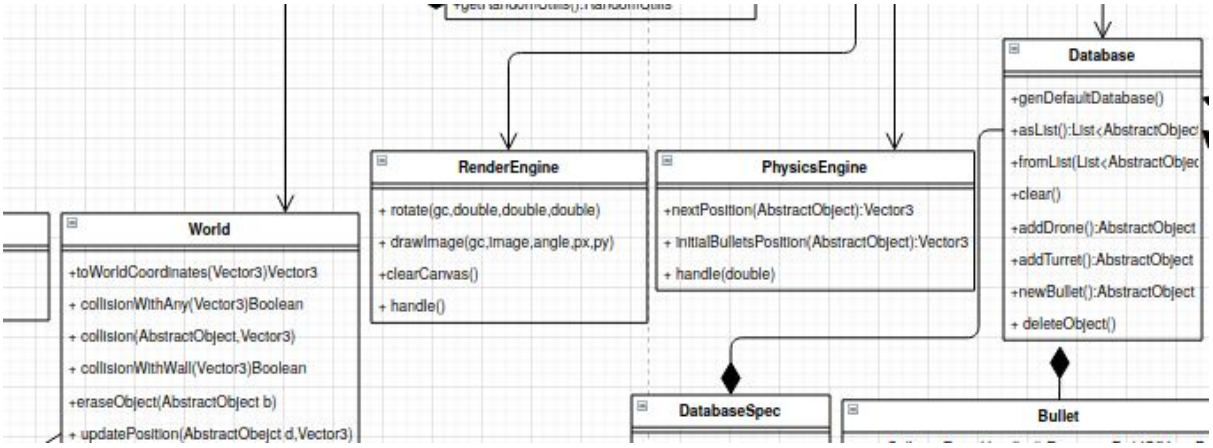
This is the second concrete implementation of the AbstractObject. It represents a bullet fired from any AbstractObject.The velocity of the bullet is constant at 3.0 pixels/frame.If a bullet hits a Drone, Then the drone loses one life, and if a bullet is fired from a drone (or a turret , see below) then it has the height of this particular drone and it can only hurt drones at that height.

Turret

This is the 3rd and final concrete implementation of AbstractObject. A turret is a static(non-move-able) object which fires bullets to drones . it has unlimited lives and it has the ability to steer itself(rotate) in order to shoot drones.every turret is operating in a specific height and it can shoot drones only if the respective drones are operating in the same height.



The Engines class cluster



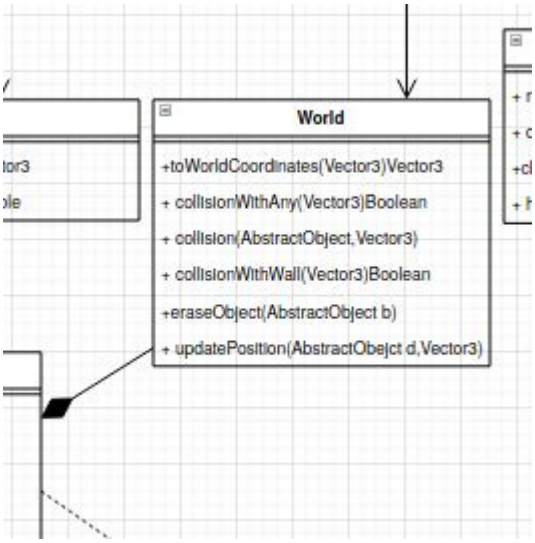
The engines are distinct operating modules working independently in different thread .There are 2 of them , named PhysicsEngine and RenderEngine(Renderer).

PhysicsEngine

The physics engine , runs in every frame(extends AnimationTimer) and calculates every objects next location , detects (with the help of the ‘’World” class) possible collisions and runs the respective handlers for each object(collision and non-collision event handlers).When we attempt to ‘Pause’ the simulation , effectively we just stop the Physics thread , but leaving untouched the render thread(in order for us to be able to see the paused drones , as we will see in the next section)

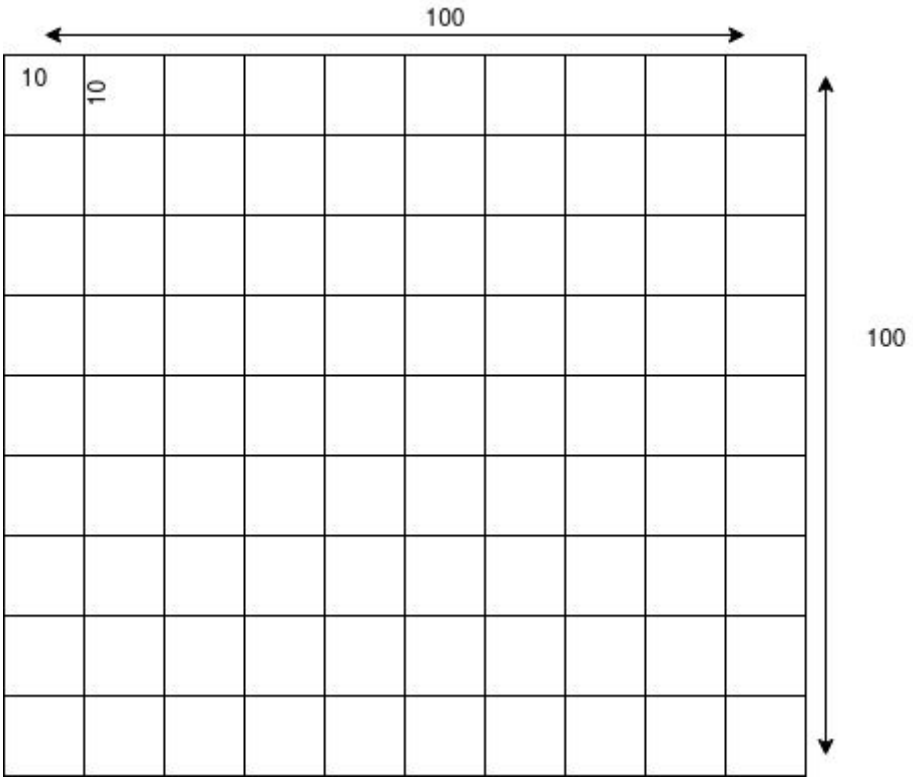
RenderEngine

The render engine, runs in every frame(extends AnimationTimer) and just draws the objects in each frame in their respective location with the proper sprite , depending on their particular height, it also handles the GraphicsContent properly (in order to create the effect of rotation)



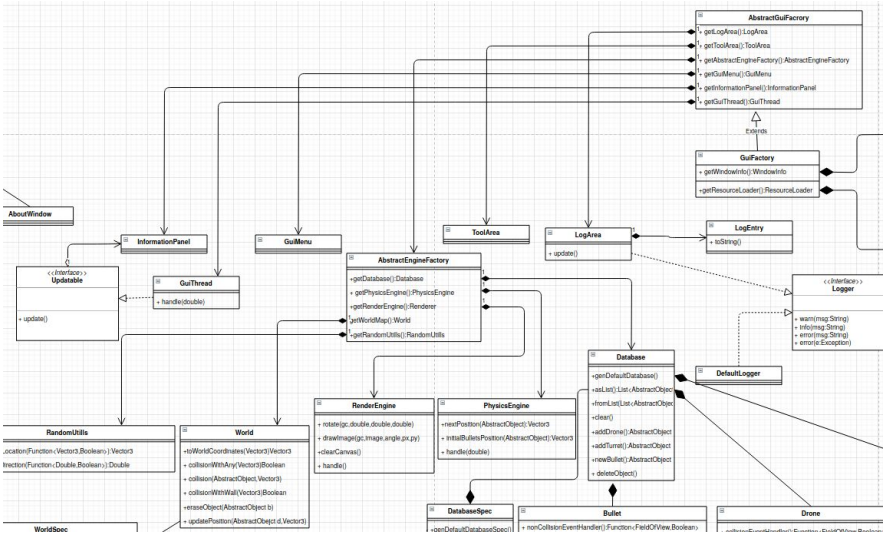
The collision system , the World class

The world class handles every request for change location .It is used by the PhysicsEngine to keep track of possible collisions between objects. Internally , a HashMap<Vector3,AbstractObject>. Is used In order to avoid the scenario of the sprite pass very close to another object , but a collision is never detected due to not colliding exactly at the middle . the World class uses a transformation to achieve that.
This transformation translates from real coordinates to a grid system of squares , just like the image below



Let's say we have a window 100x100 pixels , and we choose a grid system with squares of length 10. Each drone can move anywhere it wants , but internally the real coordinates is translated into grid coordinates , only one drone can be in a square at one time. By choosing the right square size we have a very effective collision detector. About the height, each AbstractObject can be operated at 5 different heights , 1 to 5 . each height level has its own plane (not really a plane though , just a single HashMap does the job perfectly).So effectively we keep the memory complexity linear and the search complexity proportional to the number of the objects involved.With this technique we can achieve very high frame rates indeed.

The factories class cluster



The factories keep track of the various dependencies that our classes may have , injecting them into the runtime(is the easiest and most effective Dependency Injection(DI) approach as far).We have 2 factories here , keeping the pure game engine and the javafx code as separate as possible.

AbstractEngineFactory

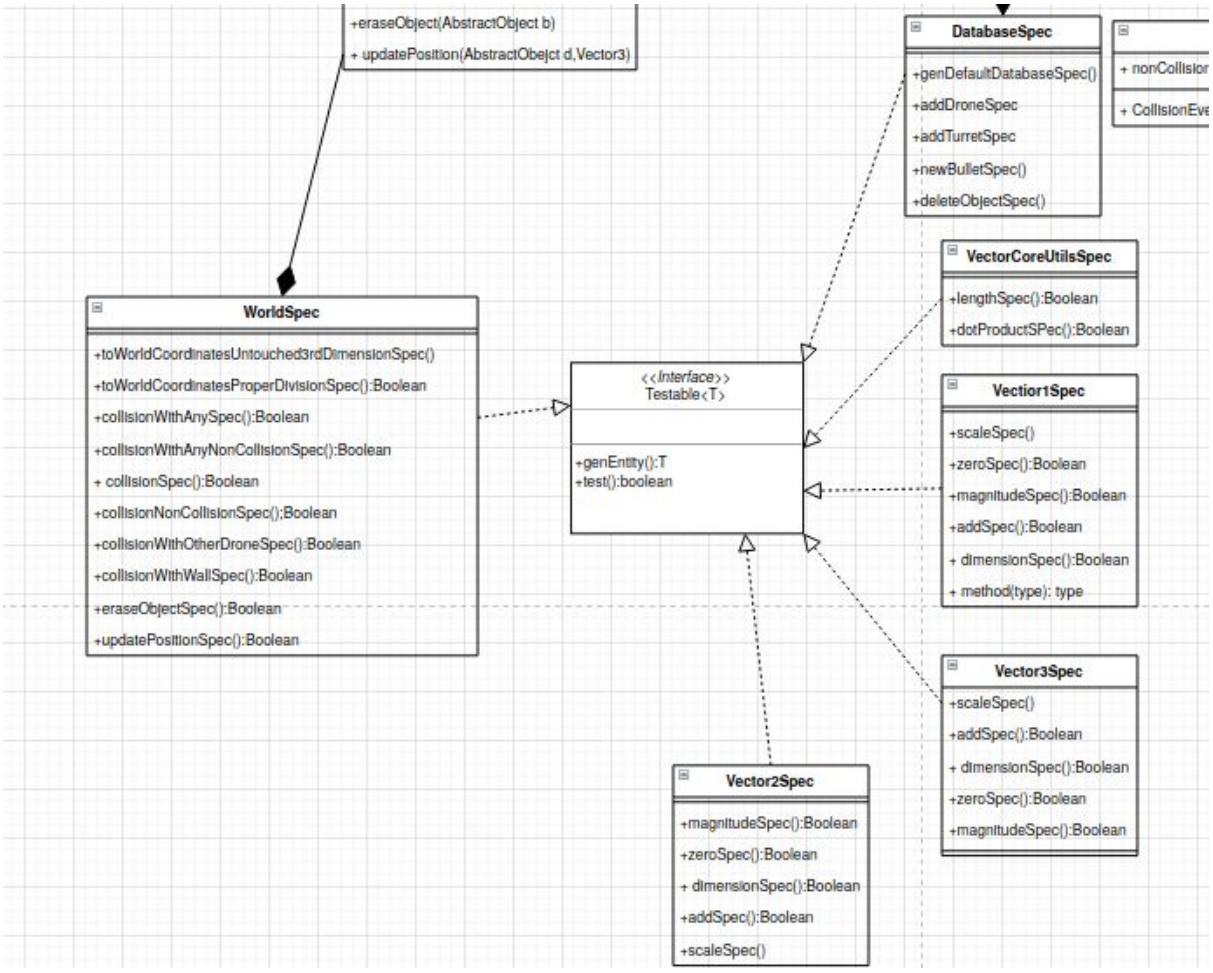
The AbstractEngineFactory and their implementation , EngineFactory , constructs , injects and maintains the set of classes that belong to the pure Game engine itself.It provides the following classes :Database,PhysicsEngine,RenderEngine,World and RandomUtils.

AbstractGuiFactory

The AbstractGuiFactory and their implementation , GuiFactory , constructs , injects and maintains the set of classes that belong to the javafx part of the game.It provides the following classes :LogArea,ToolArea,EngineFactory,GuiMenu,InformationPanel and GuiThread.

Tests

This project has embedded testing facility , with the following hierarchy



There are tests in every essential part of this project , In the collision system , in the custom vector classes as well as in the Database object. The following table contains all the tests

ClassName	Test class	Method name	Description
World	WorldSpec	.toWorldCoordinatesUntouched3rdDimensionSpec()	The world transformation must not transform the 3rd dimension(height) because the height is already discrete
World	WorldSpec	.toWorldCoordinatesProperDivisionSpec	The world transformation must return correct results
World	WorldSpec	.collisionWithAnySpec()	The world must not allow multiple drones at the same square
World	WorldSpec	.collisionWithAnyNonCollisionSpec()	The world must allow drones to co-exist at different squares
World	WorldSpec	.collisionSpec()	The world must allow a drone to change position in the same square
World	WorldSpec	.collisionNonCollisionSpec()	The world must not raise false-positives
World	WorldSpec	.collisionWithOtherDroneSpec()	The world must detects collisions
World	WorldSpec	.collisionWithWallSpec()	The world must detects collisions with the borders
World	WorldSpec	.eraseObjectSpec()	The world need to delete objects from their registry when requested
World	WorldSpec	.updatePositionSpec()	The world need to proper update the registry when a drone moves from one location to the other
Database	DatabaseSpec	genDefaultDatabaseSpec()	The Database must be able to generate a default database(need for a default arena feature)
Database	DatabaseSpec	addDroneSpec()	The Database need to be able to add a drone into their registry , and inform the World object for this event
Database	DatabaseSpec	addTurretSpec()	The Database need to be able to add a turret into their registry, and inform the World object for this event
Database	DatabaseSpec	newBulletSpec()	The Database need to be able to add a bullet into their registry, and inform the World object for this event
Database	DatabaseSpec	deleteObjectSpec()	The Database need to be able to remove a object from their registry if requested

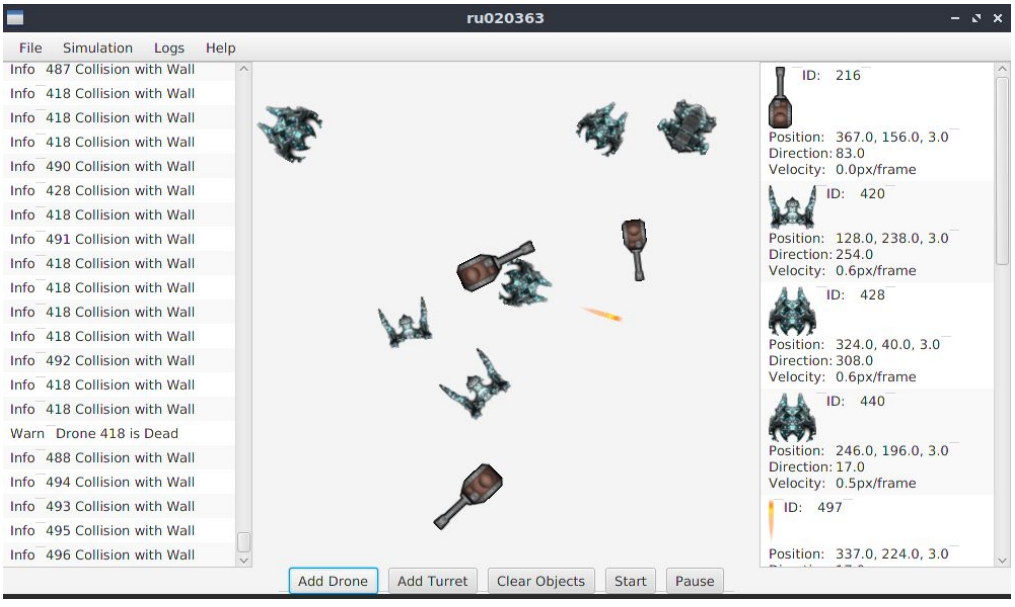
VectorCore	VectorCoreUtillsSpec	lengthSpec()	The VectorCore class needs to be able to calculate the length of a a vector properly
VectorCore	VectorCoreUtillsSpec	dotProduct()	The VectorCore class needs to be able to calculate the dot product of two vectors

And the following tests , whitch are made in every implementation of Vector class(Vector1,Vector2,Vector3)

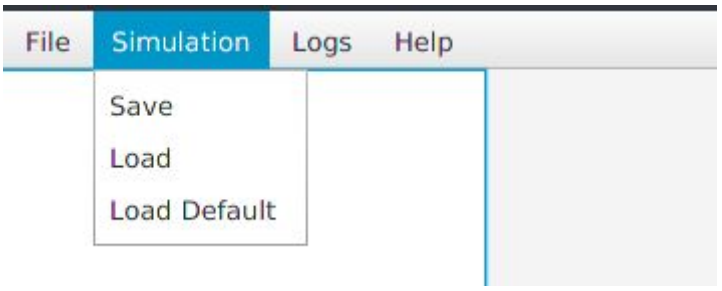
ClassName	TestClassName	methodName	Brief description
Vector1,Vector2,Vector3	Vector1Spec,Vector2Spec,Vector3Spec	.scaleSpec()	The scale operator needs to return proper results
Vector1,Vector2,Vector3	Vector1Spec,Vector2Spec,Vector3Spec	.zeroSpec()	The .zero operator needs to return a zero vector of same dimension
Vector1,Vector2,Vector3	Vector1Spec,Vector2Spec,Vector3Spec	magnitureSpec()	The magnitude operator needs to return proper results
Vector1,Vector2,Vector3	Vector1Spec,Vector2Spec,Vector3Spec	addSpec()	The addition operator needs to return proper results
Vector1,Vector2,Vector3	Vector1Spec,Vector2Spec,Vector3Spec	dimensionCheck()	The dimension operator needs to return proper results

User manual

Lets have a first meeting with the enviroment

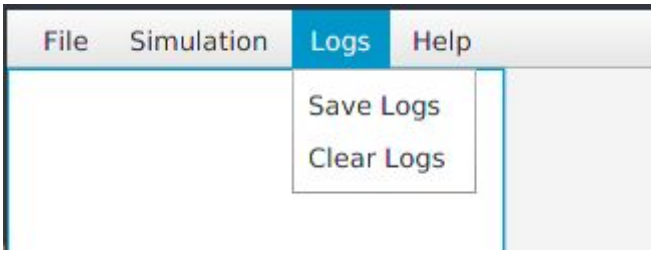


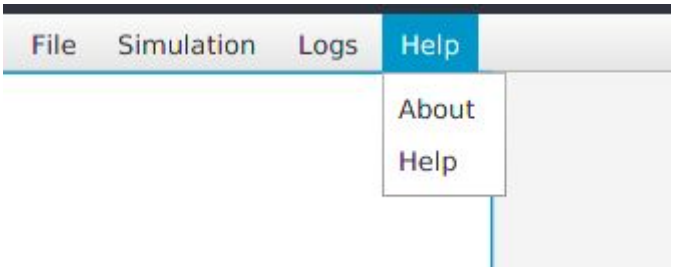
From left to right , we have 3 main areas of interest. The LogArea , which prints all the logs in the screen, we have the main drawing area , which all the simulation are displayed , and we have the Information panel , which informs us for the position,direction and velocity for every object in the screen.We have also at the bottom a Tool Area whitch allow us to introduce new objects in the simulation .The ‘AddDrone’ inserts a new drone in the drawing area , and the ‘Add Turret’ adds a new turret.we can also clear the screen and start/pause the simulation.
In the top we have our menubar.
Under the ‘File’ menu we can find the ‘Exit option’. This option terminates the programm correctly by waiting all the threads to close and clear all the relevant streams that may have been opened during the execution of the programm



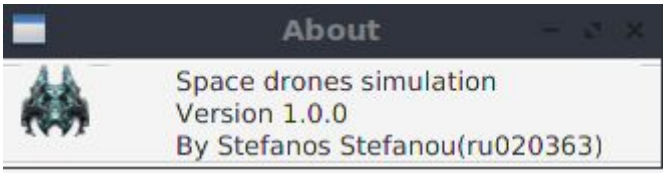
Under the ‘Simulation menu we can find our save/load capabilities’ as well as the support for loading a ‘Default’ arena with 3 drones

Under the ‘Logs’ menu we can find the support for storing the logs for future use. We can also clear the logs in order to start fresh a new simulation





Finally , under the ‘Help’ menu we can find useful information about our simulator in the ‘Help’ section and information about the developer and the version under the ‘About’ section



Discussion

I am aware that this design maybe was an <<overkill>> , but there are some serious arguments about this design. I wanted to be flexible and optimised. Using just some additional classes i made this possible.I also wanted to use as much as possble my own classes (that's why i have, for example , a custom Vector Class) for educational purposes(re-inventing the wheel is not a good idea in a working enviroment).Given a little more time I could have implement it in real 3D , but this isn't a problem. I will attempt it in my free time.

Conclusion

I knew from last years courses about the importance of OOP but I never had the opportunity to do a full project employing this paradigm.It is clear to me that a well structured codebase can actually save you a ton of time when additional features are needed quickly.Additionally i had a first contact with advanced structures such as hashmaps and i figured out how fast they are compared to more traditional data structures (such as binary trees , lists , heaps etc).Finally it was my first touch with the stream API and i can say that is very convinient

Personal reflection

It was a nice assignment because I finally put my whole new knowledge into action and I created something . it is way different from knowing the theory from actually applying it and I applied it effectively in this project. So i am happy with the result

Appendix : the full design