

College of Arts and Science

School of Science and Technology

# SOFT10101:

# Computer Science Programming

*By*

N0631237: Mohamed Dewidar

Project Space Shooter

**Seminar Tutor:** Dr Patrick Merritt

Contents

[Specification 3](#_Toc449302764)

[Design 4](#_Toc449302765)

[Project Architecture: 4](#_Toc449302766)

[Logic Manager: 5](#_Toc449302767)

[Game State System: 5](#_Toc449302768)

[Object Manager: 5](#_Toc449302769)

[Physics: 5](#_Toc449302770)

[Renderer: 5](#_Toc449302771)

[The Slot Map: 5](#_Toc449302772)

[Separating Axis Theorem: 6](#_Toc449302773)

[Testing 8](#_Toc449302774)

[The birth of Vector2D: 8](#_Toc449302775)

[Deletion of object while iterating a vector: 8](#_Toc449302776)

[Circular dependencies: 8](#_Toc449302777)

[Game Object interaction with other systems: 8](#_Toc449302778)

[Rotation and pointing bullets to the right direction: 9](#_Toc449302779)

[No bullet collision: 9](#_Toc449302780)

[Evaluation 10](#_Toc449302781)

[Bibliography 11](#_Toc449302782)

# Specification

Asteroids is a game where the player controls a ship and shoots waves of asteroids that come towards them from multiple directions. The player uses the forward key to accelerate and the left and right keys to rotate anticlockwise and clockwise respectively. Every few seconds an enemy ship comes flying in from the side and shooting it provides the player with bonus points.

The basic goal of this project is to make a clone of this game. The program will be written using C++ with SDL. The basic game should allow the player to move around, collide with asteroids, die and shoot back at the asteroids.

Stretch goals include having a simple AI that tries to shoot the player back, a menu with options, the ability to choose ships and maybe even a scrollable background with an expanding world, at which case serialization will without a doubt be an important feature that would need to be implemented.

The game will use vector graphics, this means using mathematical concepts such as matrices and vector algebra to draw and move game objects on the screen. For collision detection the game will use the separating axis theorem to provide polygon to polygon collision detection.

# Design

## Project Architecture:

The project consists of 2 main parts; the engine and the game. The game consists of the game objects in the scene and the engine manages those objects. The diagram below shows the engines architecture.

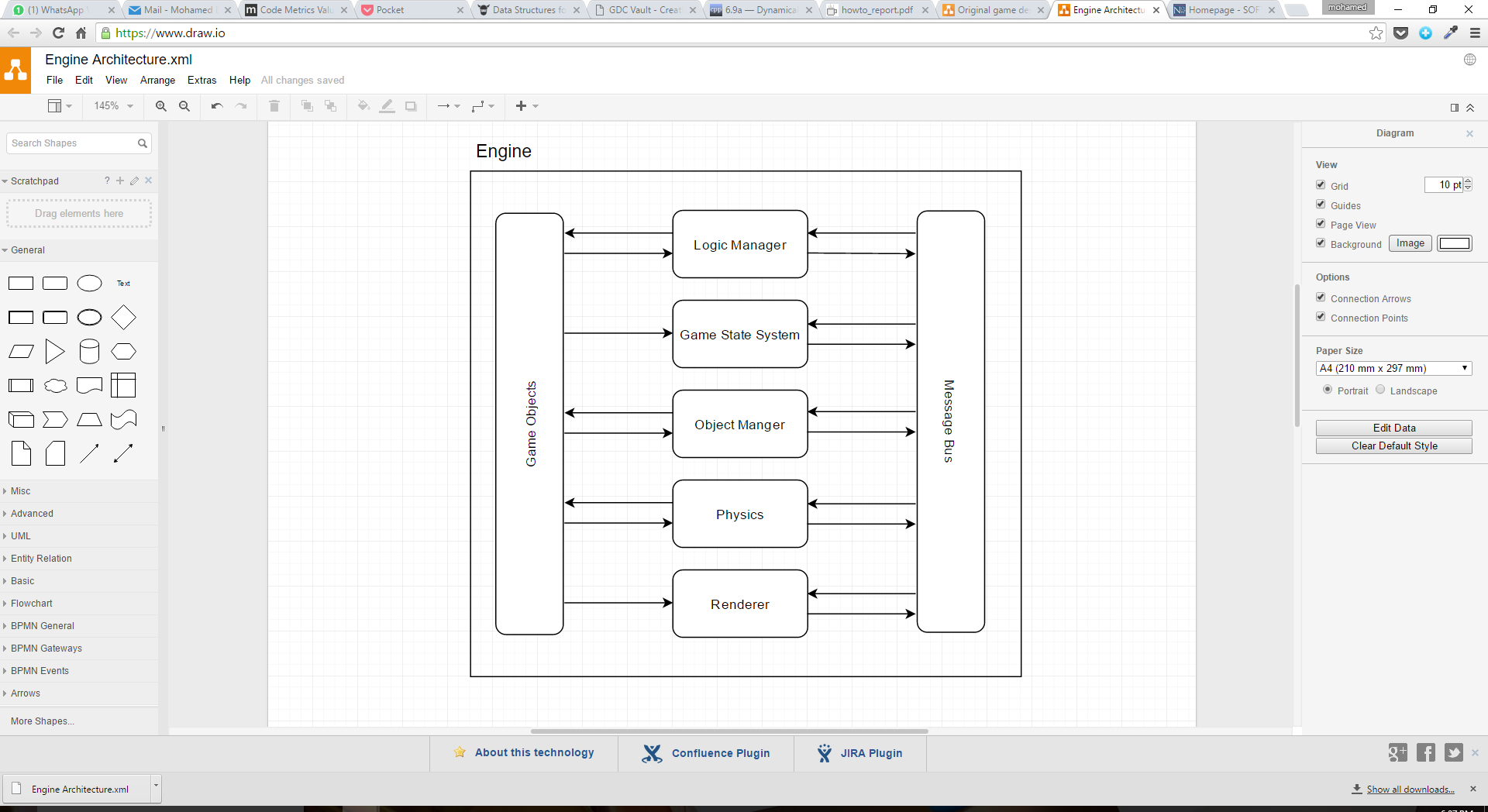


Figure Diagram showing the interaction between the systems in the engine.

The engines architecture follows an Entity-Component System design. This means that the engine consists of many systems, each of which manages or performs its own operations (mostly) independent of others, when required they can send messages across the message bus to communicate with other systems.

For example if the physics system calculates there is a collision between two objects it can create a collision message and send it through the message bus to the other systems, depending on the complexity of the message bus it could either send that message to everyone or to a specific system; in this engine it sends the message to every system it knows off. Each system receives the message and considers whether there is something to be done about the message or not.

In the entity component system, the game objects are entities and they are built up off many components, these components define the functionality of the object. For the basic needs of this project only 2 components were needed; the mesh and the rigid body where the mesh component stores all of the information about what the object looks like on the screen e.g. vertices, while the rigid body component stores the physical information about the object such as position, velocity etc.

### Logic Manager:

The logic manager is the interface through which the game objects communicate with the other systems in the engine. It allows all game objects to perform logic updates.

### Game State System:

It manages the current state of the game, it controls when the game should switch from one screen to another or maybe controls respawning the enemies when they are dead, this could also be thought of as a scene manager, however since the project only has one scene, it doesn’t have any scene functionality.

### Object Manager:

This manages all the game objects that exist in the scene. It performs operations such as creation and deletion of objects and therefore manages their storage in memory. It uses a special data structure that was created to be store all the game objects in the game (see below).

### Physics:

This performs the basic physics operations on the objects in the scene. It moves the objects as per the data in the rigid body component. This system also performs the collision checks on the objects using the data in the rigid body component.

### Renderer:

This system manages drawing things to the screen. It uses the information in the mesh component to draw the objects to the screen using SDL.

## The Slot Map:

Since this is a space game based on asteroids, it is difficult to predict the number of game objects that will be on the screen at any point in time, and having multiple data structures for each type of object (vector for asteroids, vector for bullets etc.) is not flexible, the best way is to have one data structure that stores all of the objects in the scene at once.

The first choice is a vector however a vector is problematic in the sense that deleting or adding objects reshuffles the indices of the object, this means that to delete an object directly by index is no longer possible as its index has “expired”.

The slot map fixes this by not reshuffling the structure of the objects stored but rather marks the index as being free and therefore “frees” space to be available for further use while keeping the indices of all the other objects that exist in the map.

The slot map consists of a vector of fixed sized arrays, the arrays store the game objects that need to be stored. It works as follows:

* Adding a game object is as simple as taking a number off of the vector of free locations in the slot map and inserting the object in that location.
* Accessing a game object is then achieved at a high speed as the index is used directly to point to the location of the object in the slot map.
* Deleting an object includes marking its location as free so that it can be used later.
* Expanding the size of the slot map includes allocating another array of game objects and marking the space as free to use it later.

## Separating Axis Theorem:

The separating axis theorem is one of many theorems that are used to check collision between objects in 3D space. The basic idea behind the theory is that if you can draw a line between 2 objects that separates them, then they are not colliding. Unlike the axis aligned bounding box collision method or circle collision the separating axis theorem allows for any convex polygonal shape, this allows for more detailed collision detection and since the game uses vector graphics the game, this theorem suits it very well as the game is based on vector graphics; therefore the mesh can be used as a polygon for the collision detection resulting in a very accurate collision detection algorithm (similar to “pixel perfect collision” in terms of functionality).

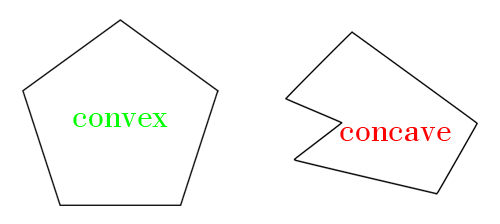


Figure Examples of convex and concave polygons.

The theory uses the vector math concept of projection, this is the idea of calculating a value which represents the “shadow” of one vector onto another. In this case the “shadow” of the vertex in 2D space is obtained on the normal of the edges, recording the max and min values obtained provides a range for both objects, if there is an overlap in these two ranges there is collision between the objects and one must continue the calculations. If there is no overlap then it is impossible to have collision between the two objects.

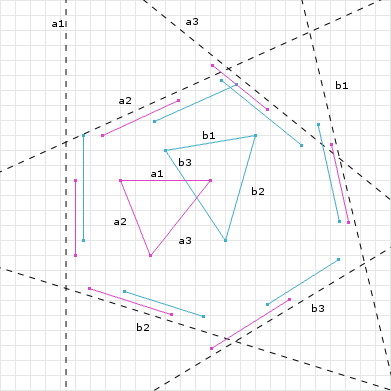


Figure Example of two triangles colliding, overlap in ranges shows collision using SAT

This collision calculation method is as complex as the shapes used. This means that checks with far away objects should be avoided as much as possible, this is done by giving all objects a proximity that other objects have to fall into before the SAT collision is performed on the objects.

The pseudocode of how the collision is detected is available below:

For every game object in the vector

Check whether other objects are within range using circle collision

If the object is within range start SAT

Get all edges for both objects

Find normal for all of the edges of both objects

For every normal

For all of the vertices for each of 2 objects

Find the projection vector of vert onto the normal

Find projection vector scalar

If scalar is greater than current max

It is the new max

If magnitude is lower than current min

It is the new min

If the max for object 1 is greater than the min for object 2

OR the min for object 2 is greater than the max for object 1

NO COLLISION

# Testing

This section doesn’t cover all of the errors that ever happened in the life of the project, they were too many and almost impossible to keep track of. Instead this section only shows the major problems that came up and how they were dealt with.

## The birth of Vector2D:

One of the first problems that occurred in the development process was the inaccurate rotations of the shapes. When the rotation matrix was applied to the shapes they were seemingly changing shape as they rotated. After constant debugging it was discovered that because the SDL\_Point data structure stored the numbers as integers, every time the rotation was applied the numbers where truncated to be fit into the integer format. This now inaccurate value would then be used in the matrix calculation the next time the object was to rotate. This consistent loss of accuracy resulted in the shape changing as it rotated.

The solution to this was to create a new custom data structure to represent a 2D vector/point in the game world. This would require the use of a number format which supports fractions, the project uses doubles. This also allowed the inclusion of other functions that were bound to be written anyway, such as finding the dot product, the magnitude, etc.

## Deletion of object while iterating a vector:

This problem was actually the main problem that resulted in the change to a more structured and system based engine. It encouraged the need to have a better back end for the game. It was also the reason the slot map.

The issue was that when updating the logic/physics for the objects, some objects would need to be killed, deleting them from within the loop would result in the loop breaking as the program went back to go to the next object in the list as the vector had now changed. This lead to the development of the slot map and then the redesign of the engine to be in the state it is now.

The current object manager also records the list of objects to delete and keeps track of them till the next frame and cleans up at the start of the frame. A similar feature was to be implemented for creating objects, however time constraints meant that it couldn’t be implemented.

## Circular dependencies:

This was one of the more annoying issues during the projects development. As the program began to have a more structure back end and the systems started to form, more and more files existed in the solution which required many includes in multiple files. Circular dependencies happen when two files attempt to include each other, resulting in a loop during compilation which ends with over 20 errors which aren’t very descriptive.

The solution to this is to include a forward declaration for the class that is being included in one of the files therefore telling the compiler that the class does exist even though the file itself wasn’t included. This is a problem that I think compilers should be able to discover and avoid completely.

## Game Object interaction with other systems:

This was something that hadn’t been planned for when designing the engine. It was the issue that as part of the logic at any point one object may require another to be deleted (on collision) or maybe created (when player shoots). This meant that the game objects needed to have a way to interact with the other systems in the engine, this lead to the development of the CreateObject and DeleteObject friend functions which given a logic manager will use it to post a message to other systems, this required providing the game objects with a pointer to the logic manager somehow, the simplest way was to pass the logic manager as a parameter to the game objects update function.

## Rotation and pointing bullets to the right direction:

Again this was caused due to an oversight in how the rotation would work for game objects. Objects are rotated by calling one function which rotates the meshes vertices immediately. This means the current rotation of the objects is not stored, which means when bullets are spawned it is not possible to rotate them to have them point in the right direction; the bullets would move in the right direction but their shape would not point in the right direction.

This is a problem that was not fixed due to time constraints.

## No bullet collision:

A way around the issue above was to use bullets as dots instead of shapes (such as lines). This however caused a problem as the collision only works on polygonal objects. Since the bullets don’t have any edges the collision fails and so the bullets pass through the asteroids unaffected.

This is a problem that was not fixed due to time constraints.

# Evaluation

Overall, I have mixed feelings with the final product. On one hand I’ve learnt a lot about designing programs, entity component systems, object oriented programming, collision detection, and C++. However on the other hand the project has not been very successful in terms of functionality. When I set down my stretch goals I really hoped that I would be able to get them done.

Looking back now I wish I had either started off with the engine with my first line of code, that way I could have avoided wasting the time at the start getting something working then struggling to get it to work again.

I have enjoyed the project however, and I will try to work on it more and maybe try to add those extra features I wanted.

# Bibliography

Bittle, W. (2010, Jan 1). Retrieved from dyn4j.org: http://www.dyn4j.org/2010/01/sat/

Middleditch, S. (2013, Jan 5). Retrieved from http://seanmiddleditch.com/data-structures-for-game-developers-the-slot-map/