# Project 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.

# Project 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.

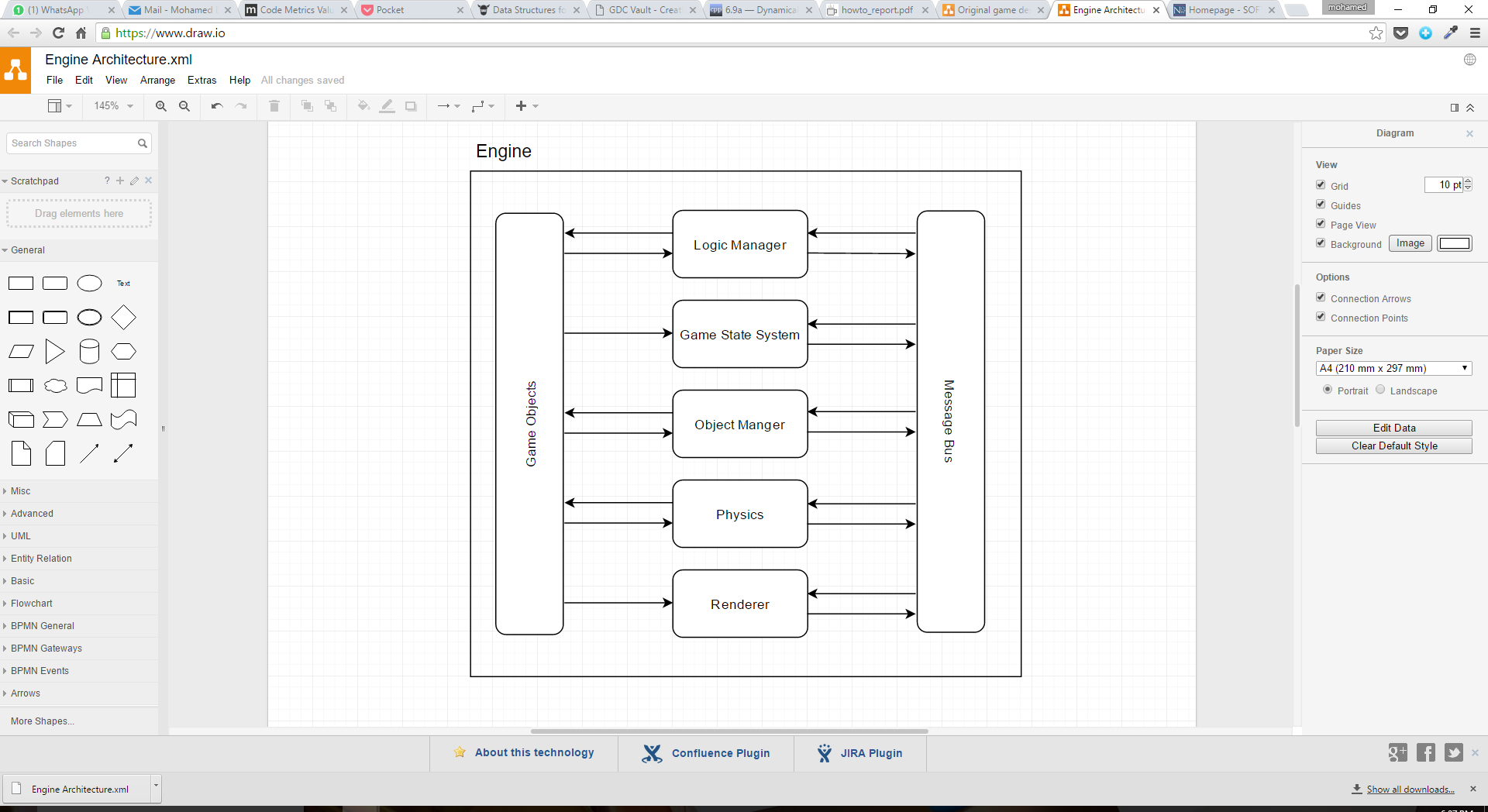


Figure 1 Diagram showing the interaction between the systems in the engine.

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.

### 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.

Need to make provide all game objects with a handle to allow distinction between other objects, such that when I delete an object from the heap I can remove all pointers pointing to it. However removing an object from the vector shifts all the objects in the vector invalidating all the game object handles.

Need to rethink code design!!!!

Should the message bus deal with each message that comes immediately when it happens or should it store them all and then let the systems deal with them in their own update functions? Since I’m using a broadcasting system for now both will have loops through the messages unnecessarily. However having each deal with the messages in its own update period means messages pushed onto the queue later may not be dealt with by systems which have already had their update period, e.g if a destroy object message is posted by the physics system, all other systems which have updated before the physics system will not deal with the message till the next frame. This also poses the problem of keeping track of which messages on the queue have been dealt with and which haven’t. Solution? Manage them all the moment they are triggered. So when a message is posted the message bus goes to every system and announces the message to them and they can choose to handle or not.

Circular dependencies!!! What a pain!!!

Can’t put the sdl initialization in the constructor for the engine, the constructor for the members in the engine class get called before the constructor of the engine class itself meaning the renderer has already tried to create a window, which causes problems.

Now that the engine works, will need to redesign the game object class. I could either try and go for an entity component system, or continue to patch it up like this. Either way I need to find a way to separate physics from the main logic. This means I will need to either provide every game object (or component) with a physics update function or publicly expose all of the variables. I could add a physics update function to the base game object class and therefore it can’t be changed without effecting all objects, however it still means someone can change it. Using the component system only means that the code of the component needs to be changed instead of the code of the game object. The only benefit for the component system would be modularity. The disadvantage of having a function means it can be called from anywhere. The solution to this is the friend specifier!!! In this case a component system is best since then only the public members can be accessed and the physics engine could be a friend and access any more private members.

Too meet basic requirements I need the following components:

* **Transform:** Has 2D coordinates and 2D rotation (though storing rotation is not something I need now), would also have handle in the future.
* **RigidBody:** Would have everything physics related such as mass, movement direction, velocity, acceleration, max and min velocity.
* **Mesh:** which would define the shape of the object and its forward direction.