Project Overview

Asteroids is a game where the player controls a ship using “Tank Controls” to shoot waves of asteroids that come towards them from all directions.

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

Future goals include making a scrollable background, adding enemies that have a reasonable amount of intelligence that can attack the player.

Project Analysis

In this section I will be analyzing my game requirements to turn them into technical requirements to help me decide how to design the game code.

**Engine Requirements:**

The initial engine requirements are:

* The engine needs to be able to provide separating axis collision that can be used on both convex and concave polygons. This will require the use of mathematical functions which I can either obtain from the internet or by writing them myself.
* The engine needs to be able to draw vector graphics to the screen and allow me to perform transformations to them efficiently. It will require separation of the object (local) and world coordinates.

The additional engine requirements are:

* The engine needs to be modular in terms of scenes, I need to be able to switch at any point in the game from the main menu to the actual game to the high score scene to whatever other scenes I decide to add later into the game. I will also need to consider more how I want the scenes to interact with each other.

**Gameplay Requirements:**

The most fundamental gameplay requirements are:

* A player ship created out of vector graphics that can move around the screen and rotate using the classic asteroids controls. The player should also be able to shoot at a set rate but without any bullet limits.
* Vector graphics asteroids that move in a set direction till a collision causes them to change course.
* Bullets should be able to damage the asteroids either by splitting it in to or changing its geometry.

The additional gameplay requirements include:

* Increase the space for movement allowing the camera to scroll to reveal more parts of the world.
* Place AI ships that search for the player and shoot at them.
* Procedurally generated world that changes every time you play.

Project Design

For this project I will be using vector based graphics, I will not be utilizing textures or surfaces that SDL provides. I also plan on implementing a collision detection technique called the separating axis theorem, this allows for accurate polygon on polygon collision which is suitable since all game objects in the game will be made out of vectors and will basically be polygons.

**Game Objects Structure:**

In this game, all game objects are vector objects, this means that in code they are to be represented by points and when drawn have to be connected to lines. To represent the vector sprite I must store the points at which the lines change in the image, I also need to store the center coordinate in terms of screen coordinates, I also need to be able to get a “draw able” version of the coordinates since the render points function takes all points relative to the screen coordinates.

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.