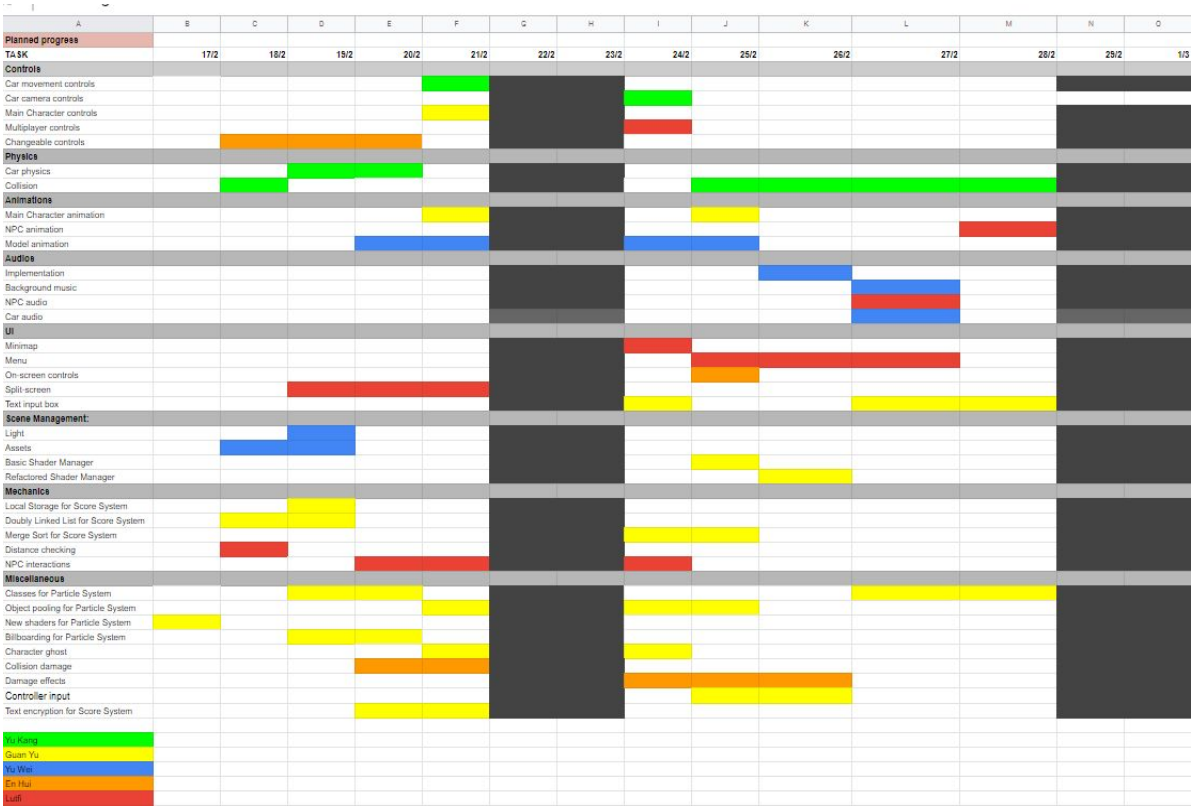
DM 2199 Computer Graphics Project

Project Report

Ou Yu Kang (190721A)
Ling Guan Yu (193541T)
Lutfi Dini Bin Azman (192614Y)
Lee Yu Wei (192848R)
Lim En Hui (194013J)

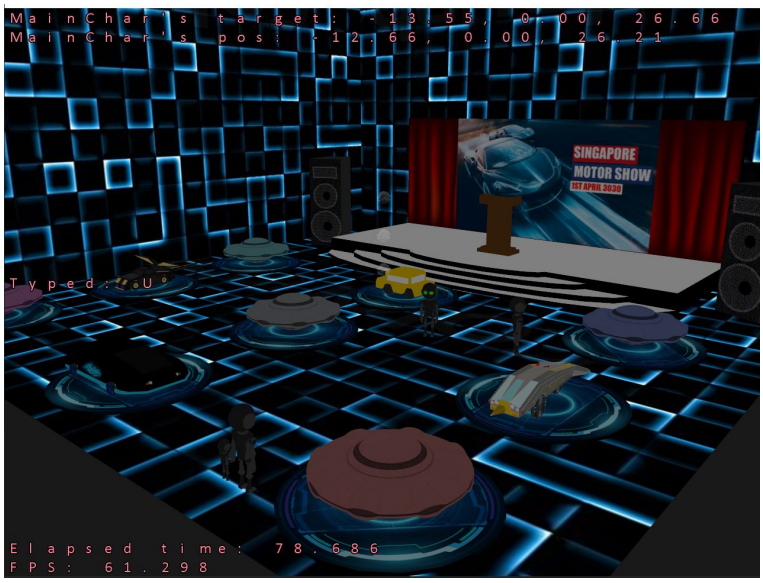
Planned progress:



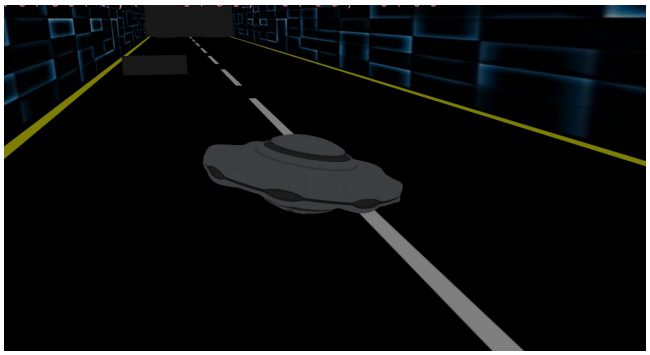
Actual progress:

Screenshots:

MotorShow Scene

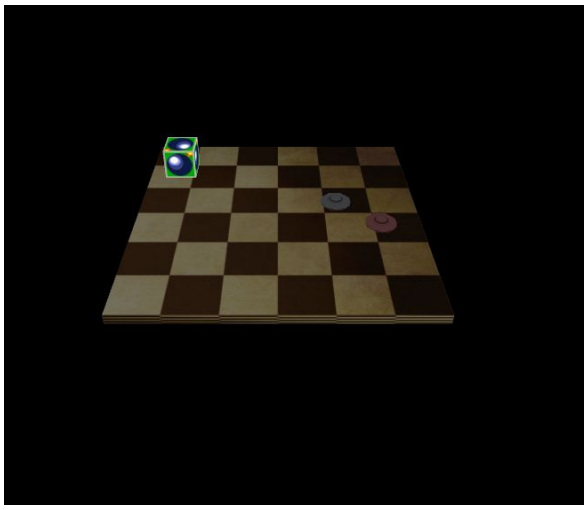


Minigame 1 Scene



Menu

Minigame 2 Scene



Implemented Features (knowledge applied listed)

- Car (UFO)
    - Physics(acceleration, velocity, displacement)

- Main Character and Main Character ghost
    - Hierarchical Modelling
    - std::pair
    - STL vectors
    - Transformations like translation and rotation

- NPCs with interactions
    - (Math P) Pythagoras theorem for distance check,
    - (Math P) Dot product for angle check and 3d billboarding
    - (Compg) Hierarchical modelling & looped animations

- Collision Detection
    - Knowledge applied: Separating Axis Theorem(SAT) https://www.jkh.me/files/tutorials/Separating%20Axis%20Theorem%20for%20Oriented%20Bounding%20Boxes.pdf
    - A lot of math specifically vectors

- Camera
    - Vector sum
    - Vector difference
    - Transformations like translation and rotation

- Local Multiplayer
    - Split-screen rendering

- Particle System
    - Dot Product (For Billboarding)
    - 3D Vector knowledge
    - Object pooling
    - Bubble Sort (For rendering order)
    - Physics (Concepts from Kinematics like acceleration)
    - Object-Oriented Programming (OOP)
    - Transformations (translation, rotation, scaling)

- UI
    - (ComG) Rendering 2d animations & Alpha Blending
    - Getting cursor position

(https://stackoverflow.com/questions/6423729/get-current-cursor-position/6423739)

- Mini-games
    - Knowledge applied: Data Struct (queue/stack)

- Scene Management
    - Knowledge applied: (CG & VART) different light types, proper loading of obj and TGA into the scenes, materials for objs, modelling and UV mapping.

- Sounds
    - Knowledge applied: (research on irrklang) usage of external lib and understanding how the individual functions work eg play3d(), play2d(). (https://www.ambiera.com/irrklang/tutorials.html)

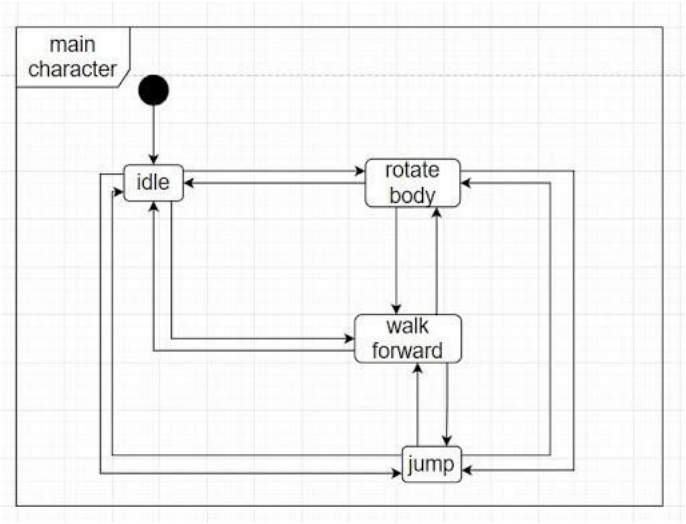
- Health bar for car
    - Rendering 2d animations with controlled frames
    - Photoshop to edit gif into 1 TGA file for animation

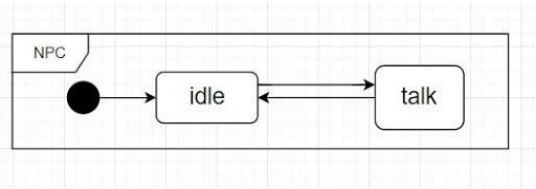
- Score System
    - File stream (Read and Write, Input and Output)
    - Encryption
    - std::pair
    - Doubly Linked List
    - Merge Sort (Includes Floyd's Algorithm)
    - Object-Oriented Programming (OOP)
    - Render Text On Screen

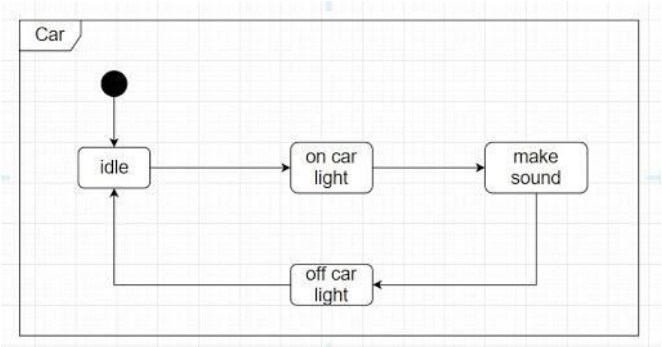
- Scene Manager
    - Singleton
    - Polymorphism
    - Type casting (static, dynamic)
    - Object-Oriented Programming (OOP)

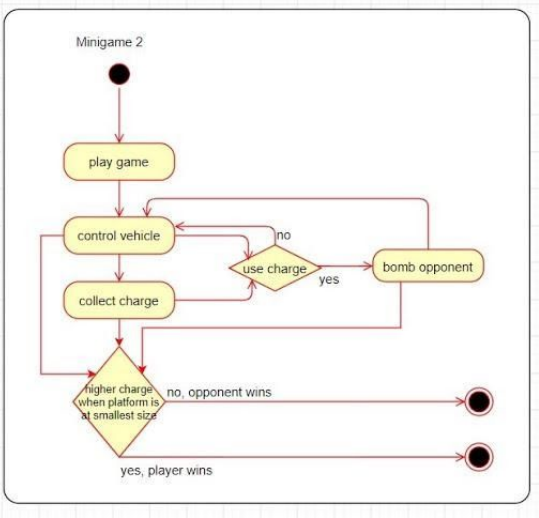
- Controller and Mouse input
    - https://www.glfw.org/docs/latest/input_guide.html

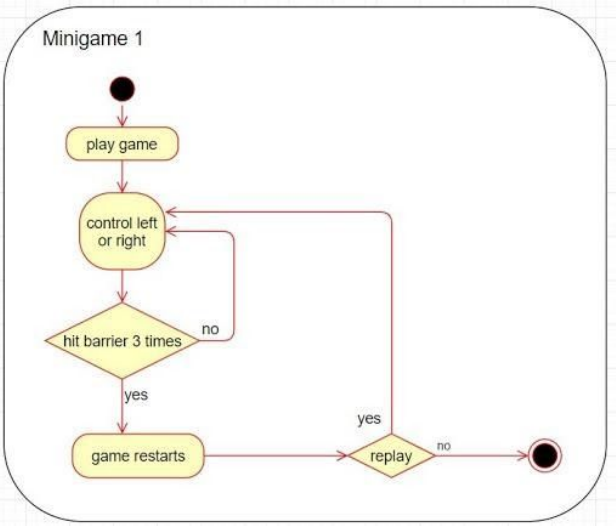
Description of implementation

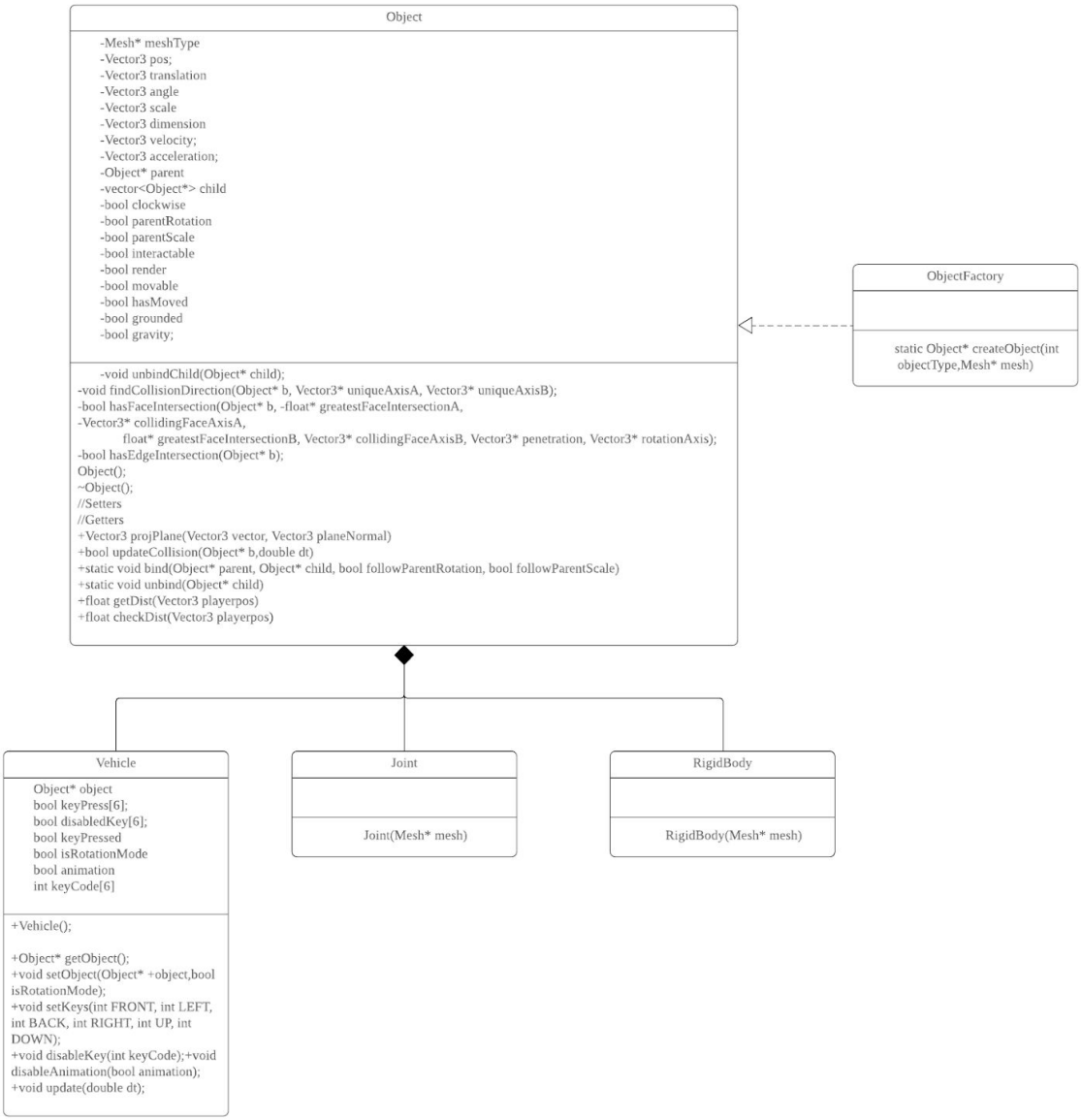




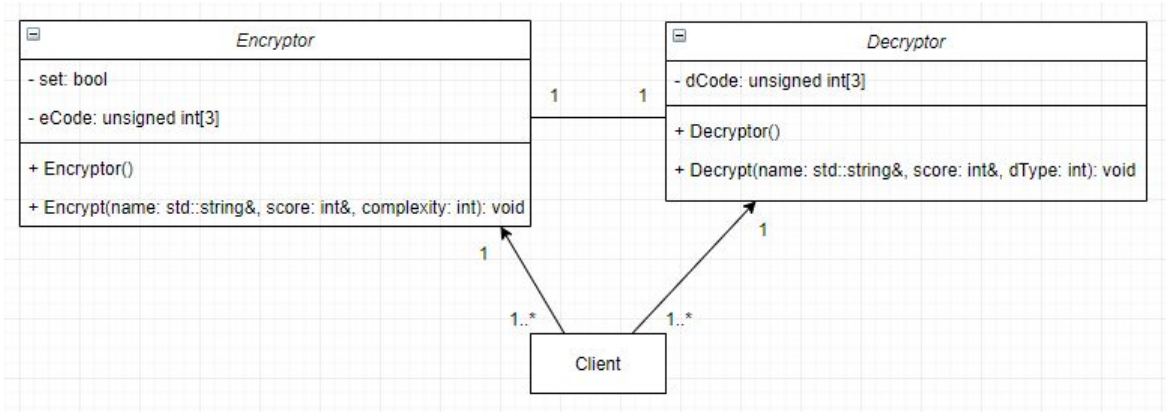




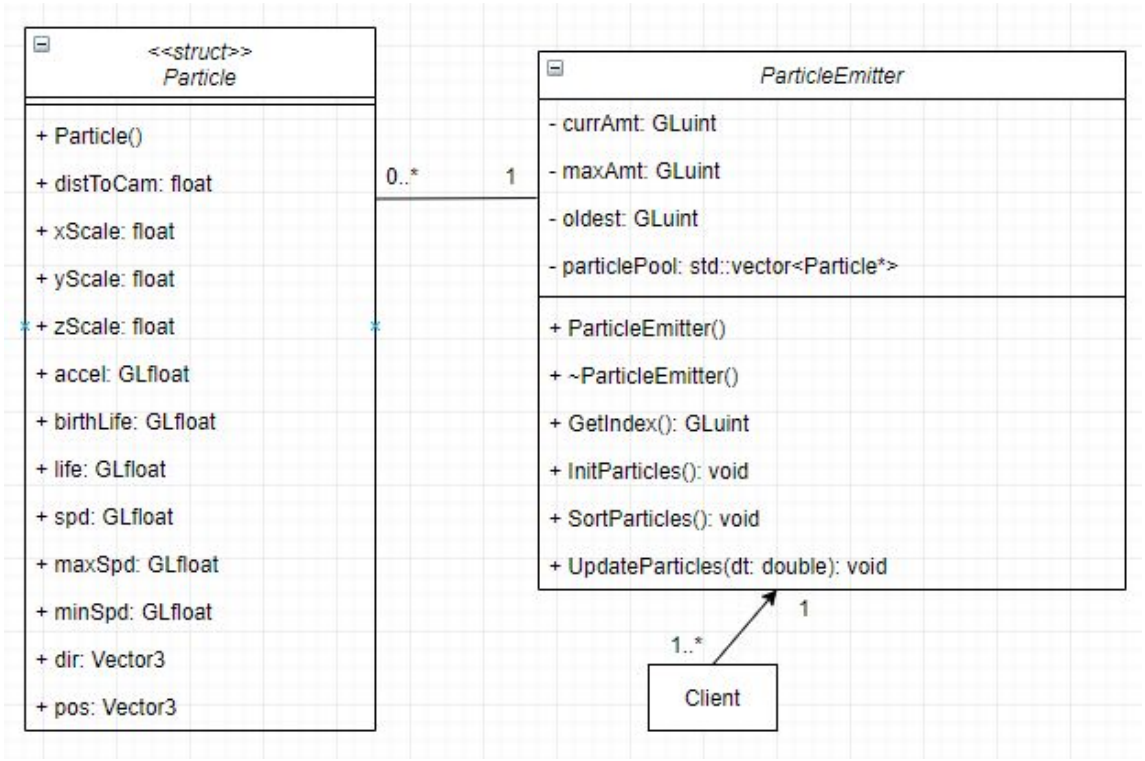




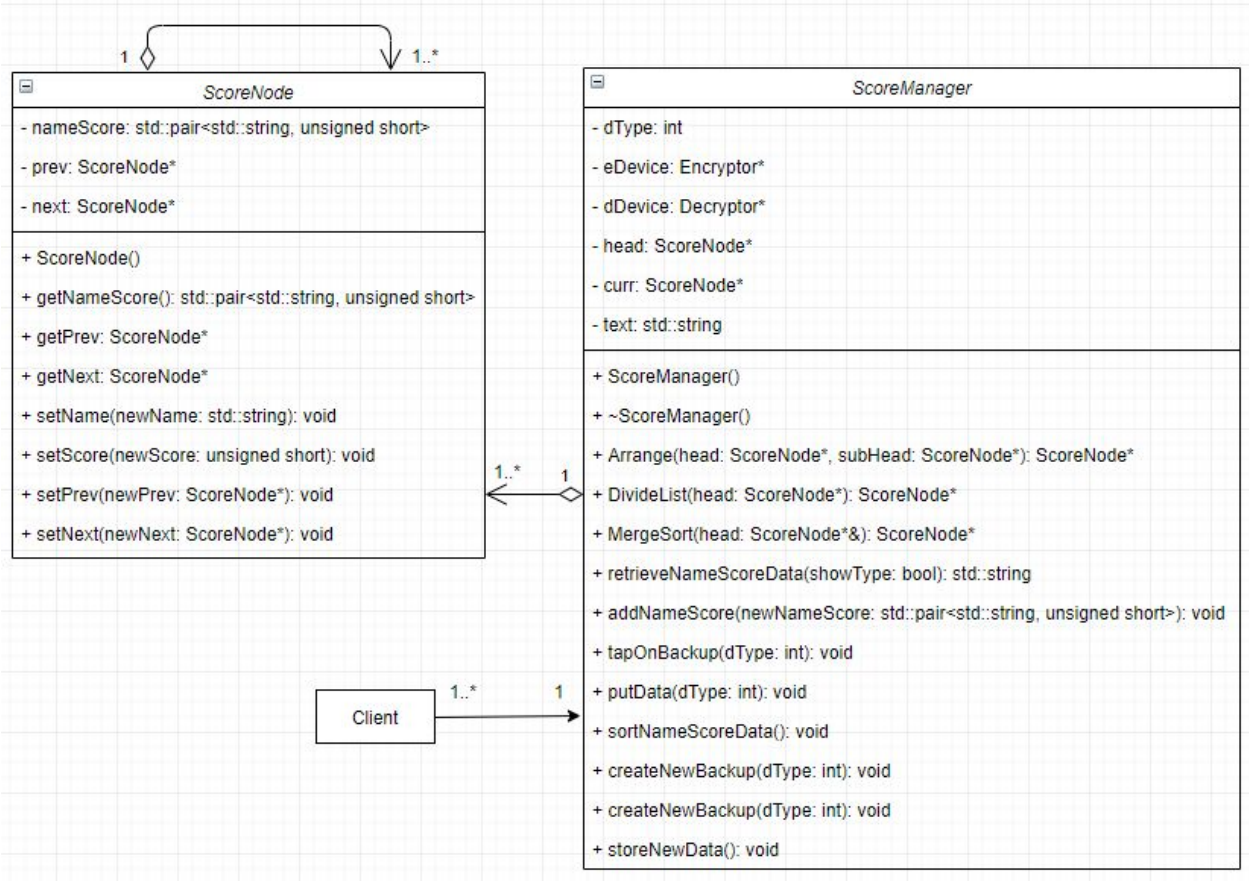
Explanation: Object class store values needed for physics, the Vehicle, Joint and Rigidbody class all have a is a relationship with Object class. Instances of these children classes can be created with the ObjectFactory class which returns different instances of the child classes based on the input into the static function createObject(), allowing for more types of “Object” to be added easily. The physics calculation in object class is also split into smaller modules so that it can be reused in the future for other projects. All values are private and only accessible via setters and getters.



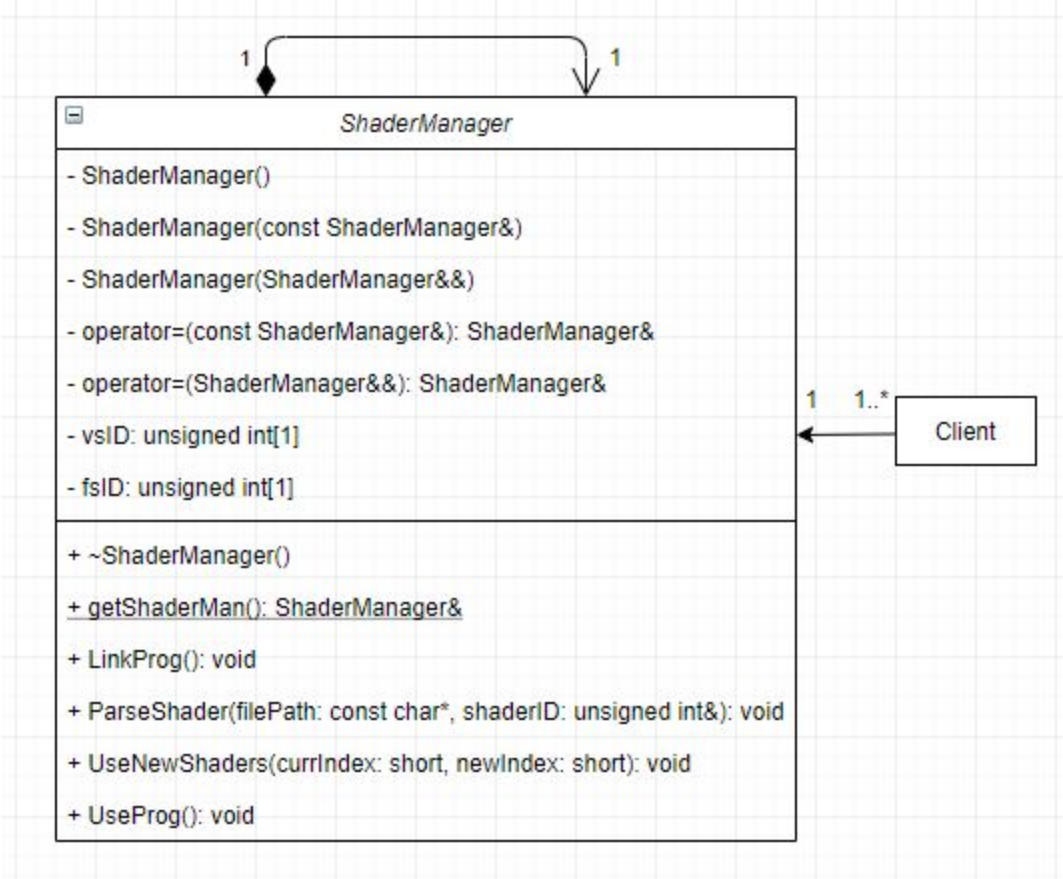
Explanation: Classes Encryptor and Decryptor are defined in Crypto.h with all their members declared there. Their data members are initialised and their function members are defined in Crypto.cpp and both classes are associated with each other with both having a multiplicity of 1. Also, 1 or more Clients are related to 1 Encryptor object and 1 Decryptor object.



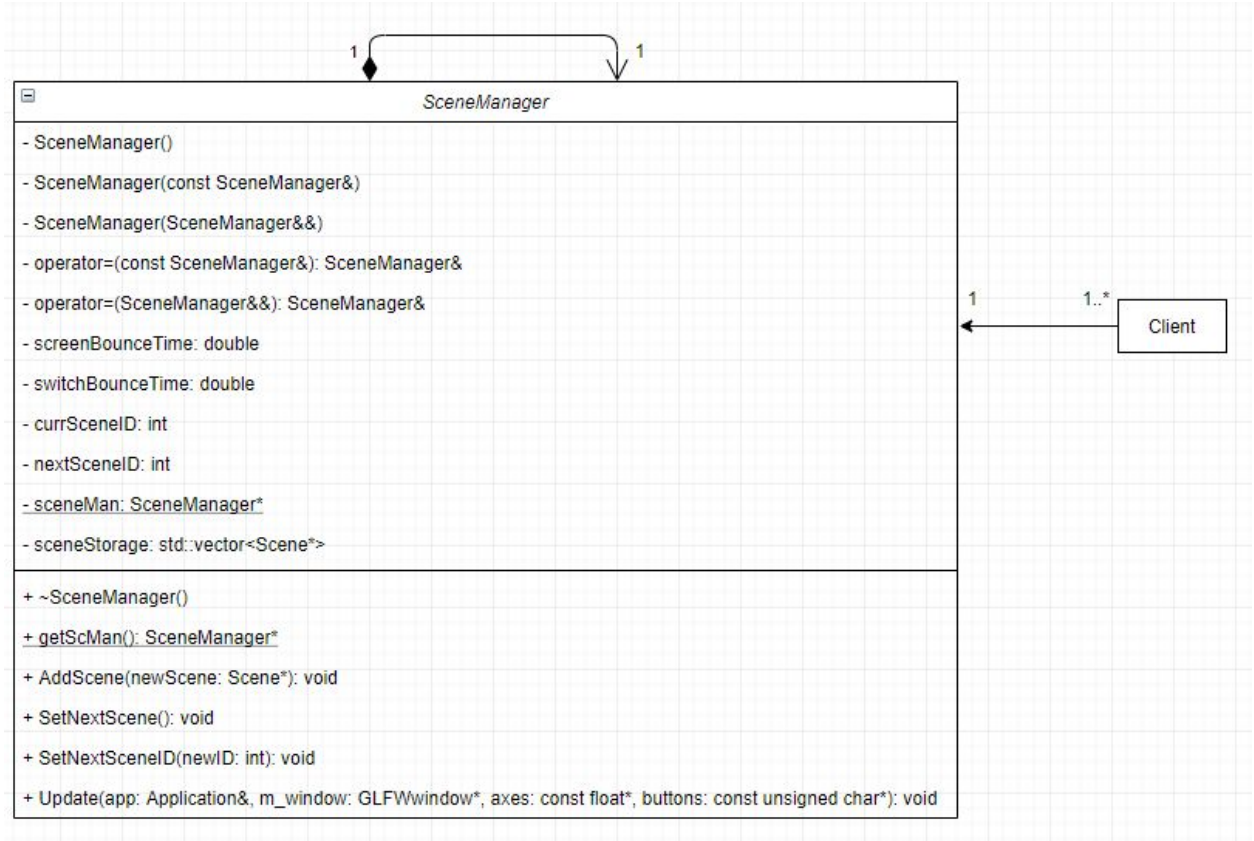
Explanation: Class ParticleEmitter and struct Particle are defined in ParticleSystem.h with all their members declared there. Their data members are initialised and their function members are defined in ParticleSystem.cpp and both classes are associated with each other with class ParticleEmitter and struct Particle having a multiplicity of 1 and 0..* respectively. Also, 1 or more Clients are related to 1 ParticleEmitter instance.



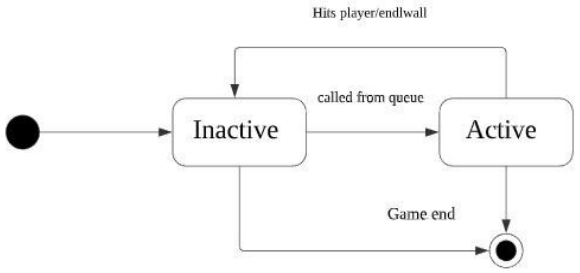
Explanation: Classes ScoreManager and ScoreNode are defined in ScoreSystem.h with all their members declared there. Their member variables are initialised and their member functions are defined in ScoreSystem.cpp and ScoreManager object contains one or more instances of ScoreNode. Also, a ScoreNode object contains 1 or more ScoreNode instances. Furthermore, 1 or more Clients are related to 1 ScoreManager instance.



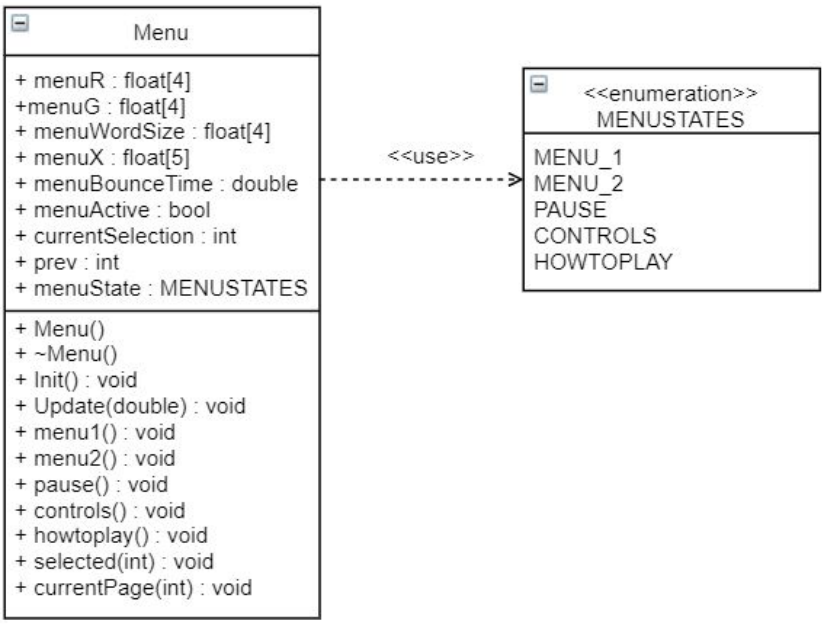
Explanation: ShaderManager is a Singleton defined in ShaderManager.hpp with all its data members and function members declared there. All its member variables and member functions are initialised and defined respectively in ShaderManager.cpp and it is composed of one and only one ShaderManager global instance. The ShaderManager class consists of a private default constructor, a private copy constructor, a private move constructor, a private copy assignment operator (overloaded) and a private move assignment operator (overloaded). This is to prevent instantiation of more than one instance of ShaderManager and thus fulfils the Singleton software design pattern. Also, the ShaderManager class contains a static getter which allows the client(s) to access the single global instance it contains. 1 or more Clients are related to 1 ShaderManager object.



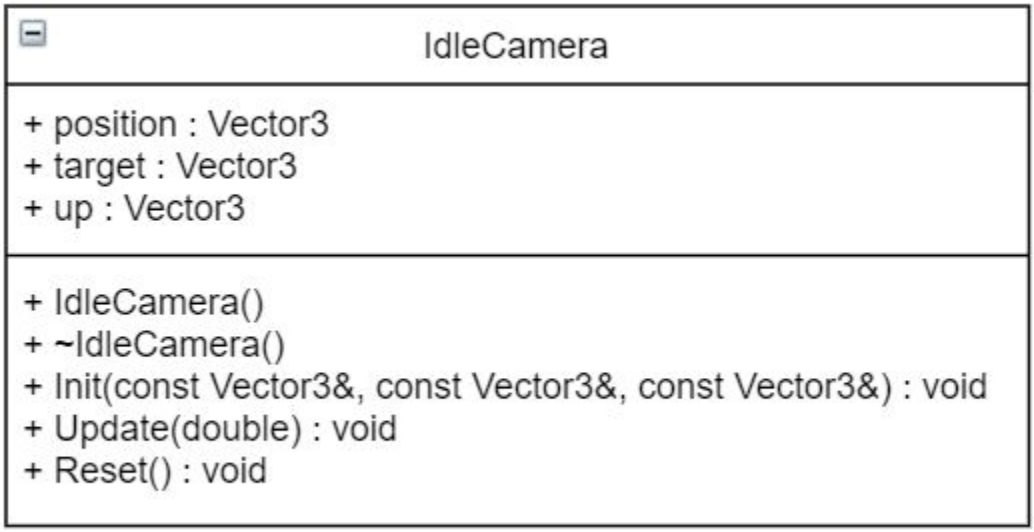
Explanation: SceneManager is a Singleton defined in SceneManager.h with all its members declared there. All its member variables and member functions are initialised and defined respectively in SceneManager.cpp and it is composed of a sole SceneManager global instance. The SceneManager class consists of a private default constructor, a private copy constructor, a private move constructor, a private copy assignment operator (overloaded) and a private move assignment operator (overloaded). This is to disallow instantiation of more than one instance of SceneManager and thus fulfils the Singleton software design pattern. Also, the SceneManager class contains a static function which allows the client(s) to access the single global instance it contains. 1 or more Clients are related to 1 SceneManager object.



Explanation: The obstacle management in minigame 1(endless runner) is based on a queue system. At first, all obstacles are inactive, after a specific time has elapsed, the first inactive object is dequeued and put into the active list of obstacles which is rendered in the scene and can interact with the environment. Once the obstacle interacts with the player or goes out of range, it is placed to the back of the inactive queue.



Explanation: Menu class is declared in menu.h. Menu responds to mouse control by detecting the current cursor position and keyboard control by detecting if up and down arrow keys are pressed. The update function contains a switch statement that calls a function respective to the current menuState. Functions “selected()” and “currentPage” are functions that alter the float variables that are used to render the menu and its selections in MotorScene.



Explanation: IdleCamera class is declared in IdleCamera.h. This class renders a stagnant MotorScene while the menu is active, such that key inputs do not affect this camera's position. The "Update()" function causes a constant yaw of the camera around a fixed target.
