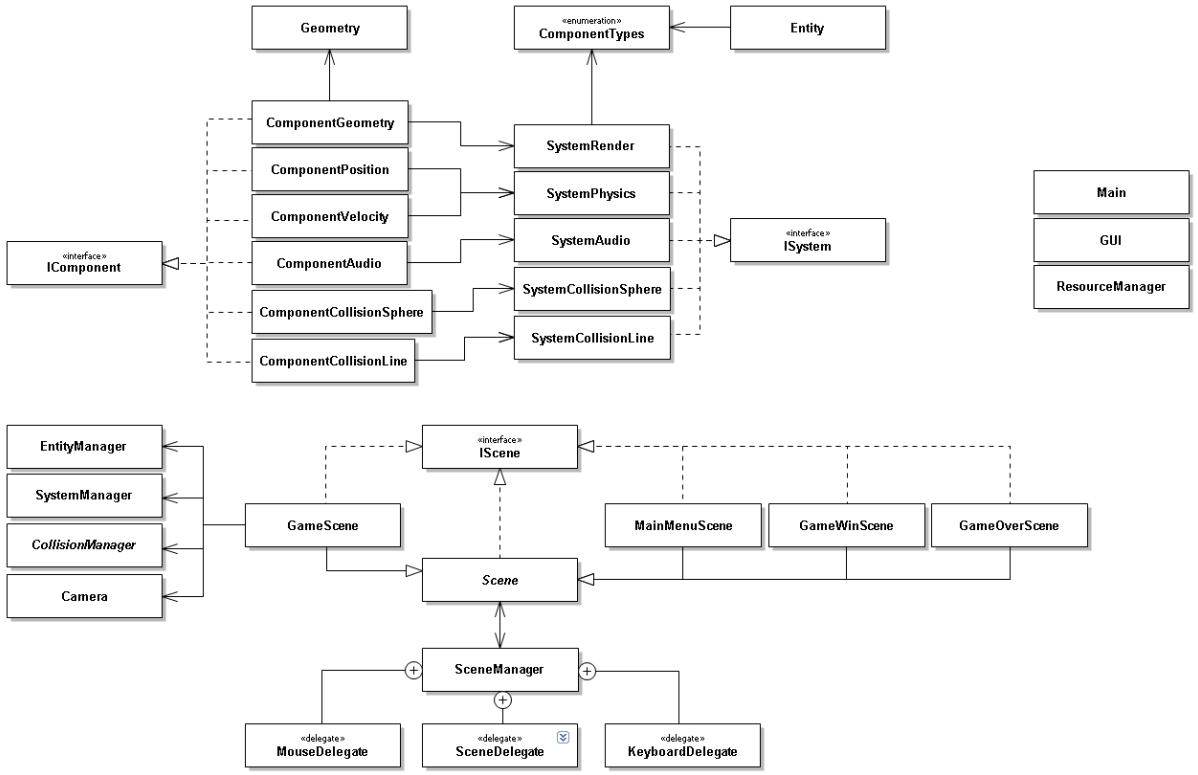
Games Architecture: Game Engine Design & Critique

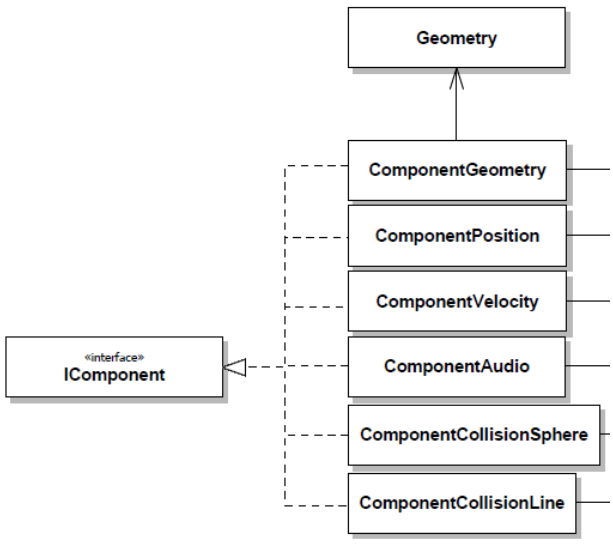
This document will cover the design of the game engine aspect of the assignment; discussing the inheritance, associations and overall structure between classes, This document will also discuss the various entities used for the assignment and compare and discuss the entity component system with an example inheritance-based design system.

UML Diagrams – Entity Component System



Entity Component System Description

Components Overview



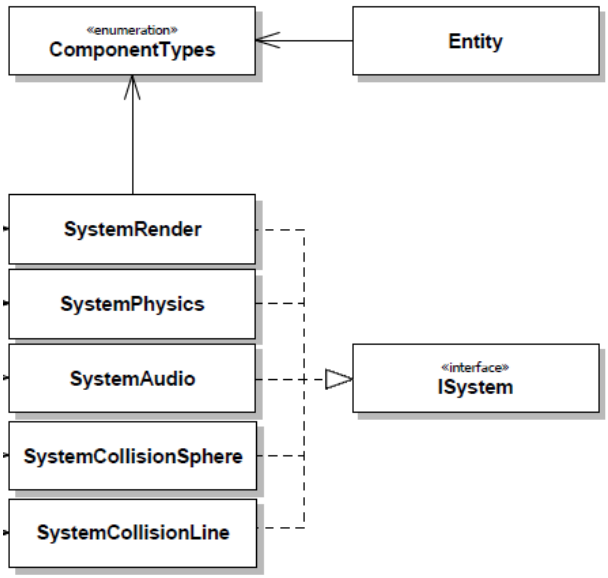
The various component classes provide entities with various properties to use within the game. The classes themselves contain variables to be used by the systems and GameScene. The components are identified by other methods through their specific enumerator type, these are declared within IComponent. Having the various attributes split into different component classes allows me when programming to provide entities with only the values that they need. The various component classes are as followed:

- **ComponentGeometry:** Takes in an object file and sends that data to the Geometry class. Used to physically render objects within the scene.
- **ComponentPosition:** Holds a Vector3 which corresponds to the where the entity is positioned in physical space. This information is passed over to the SystemPhysics and SystemRender to render entities in that position should the entity also have ComponentGeometry.
- **ComponentVelocity:** Holds a Vector3 which corresponds to the direction of movement an entity may move within the environment. This information is passed over to the SystemPhysics and SystemRender to render entities in that position should the entity also have ComponentGeometry.
- **ComponentAudio:** Takes in an audio file and contains various methods to play and manipulate the audio from the entity, sends that data to SystemAudio.
- **ComponentCollisionSphere:** Takes in a float value which represents the radius of the entity. This information is sent to the system of the same name to calculate simple sphere collisions. ComponentCollisionSphere also contains methods to manipulate the properties of the entity's collision such as turning collision on or off.
- **ComponentCollisionLine:** Takes in two Vector2 values in order to create a line. These values are sent to the system of the same name to calculate collisions.

Entity Matrix Table

Entity	Components				
	CollisionLine	**CollisionSphere**	**Geometry**	**Position**	**Velocity**
Drone		X	X	X	X
Key		X	X	X	
Portal		X	X	X	
Wall	X				

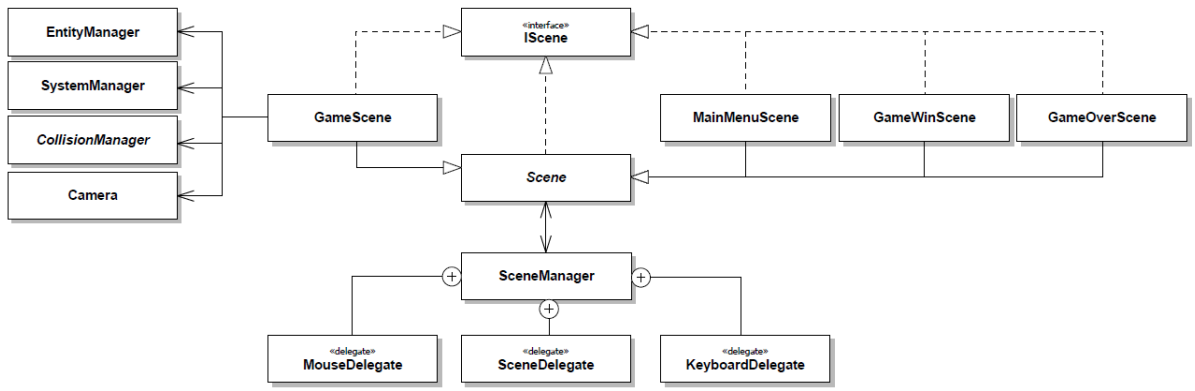
Systems Overview



Systems are used to perform the calculations of the entity components. Each system searches through each entity to find the specific components the systems using maks, when successful accesses the entity's component by reference. After the calculations, the components are then updated to correspond and/or sent to corresponding managers.

- **SystemRender:** SystemRender handles the vertex and shader calculations of the object file sent through ComponentGeometry. This data afterwards is sent back to the Geometry class to be rendered.
- **SystemPhysics:** SystemPhysics calculates the movement of an entity using ComponentPosition and ComponentVelocity. The completed calculations are then sent back and update the two component classes.
- **SystemAudio:** SystemAudio updates the position of the audio source using the entity's ComponentPosition and ComponentVelocity.
- **SystemCollisionSphere:** SystemCollisionSphere calculates whether an entity has collided with the camera, if successfully collide, the information is sent to the CollisionManager.
- **SystemCollisionLine:** SystemCollisionLine calculates whether an entity has collided with the camera, if successfully collide, the information is sent to the CollisionManager.

Scenes Overview



The classes associated with scenes can be split into two categories, scenes and managers. Scenes are classes for the environments for the program, such as menus and the game scene. These are

accessible by other classes through the enumerator SceneTypes. Managers, primarily working with the GameScene handle various backend elements such as sending messages across the program and adding the created systems and entities to the environment.

Scenes

- **Abstract Class – Scene:** The Scene abstract class lays out the various methods and variables that are used by every scene. The class is built to access the SceneManager class to initialise scenes made from the abstract class.
- **MainMenuScene:** Starting scene when the program runs. Clicking from this point takes you into the game scene.
- **GameOverScene:** End scene when the player runs out of lives. Clicking from this scene will take you back to the MainMenuScene.
- **GameWinScene:** End scene when the player successfully completes the game. This works functionally the same as the GameOverScene.

Manager

- **EntityManager:** EntityManager class handles the entities present in the program. This includes holding the list of entities and containing the methods to access and manipulate the list.
- **Abstract Class – CollisionManager:** CollisionManager is an abstract class holding methods for creating and manipulating a list of collisions. The class also has an override class for processing the collisions that are in the list. Created classes that inherit from the CollisionManager can create game specific systems. In my program, there is a messaging system to send back to the game scene.