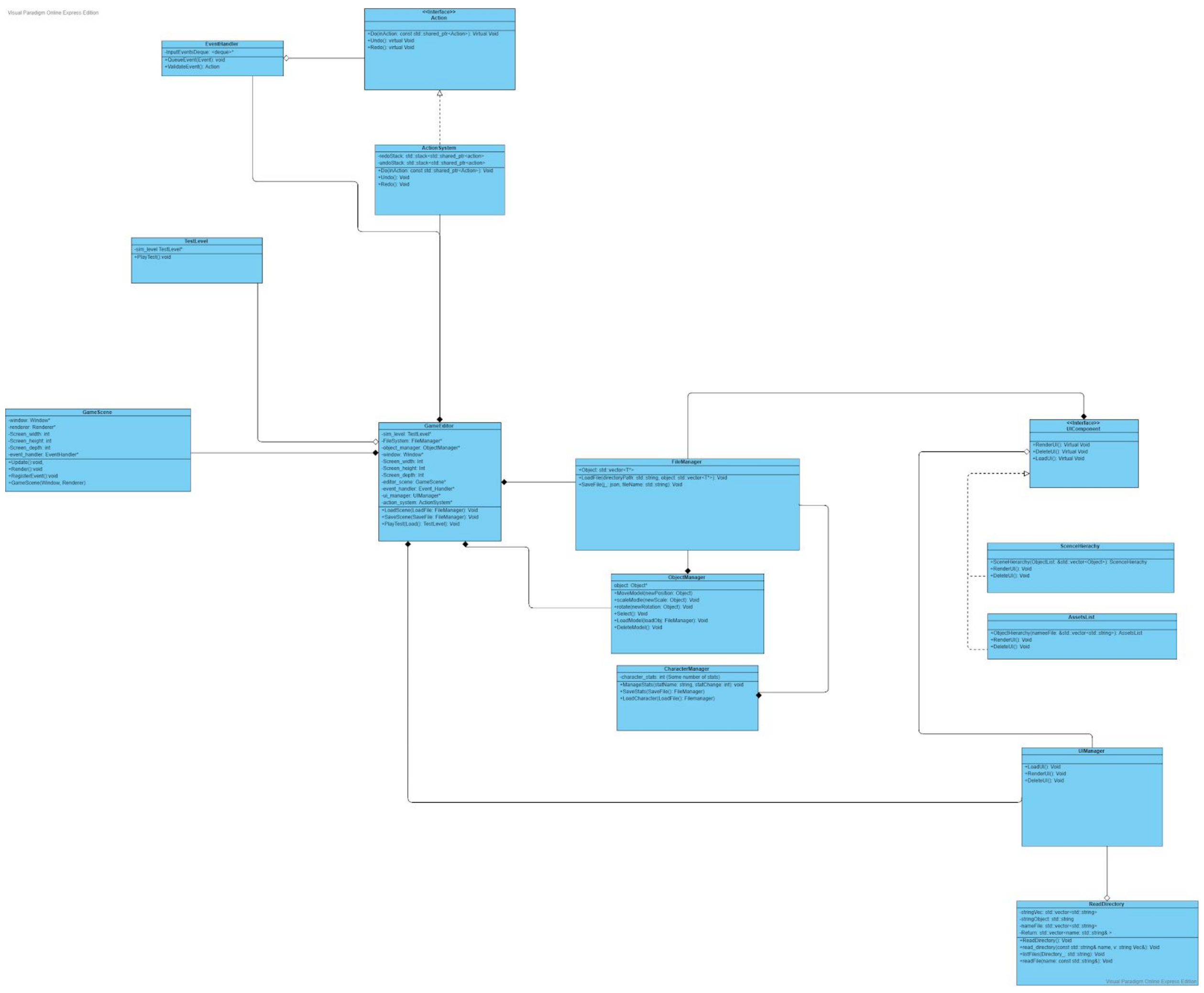
Technical Design Document (Tools)

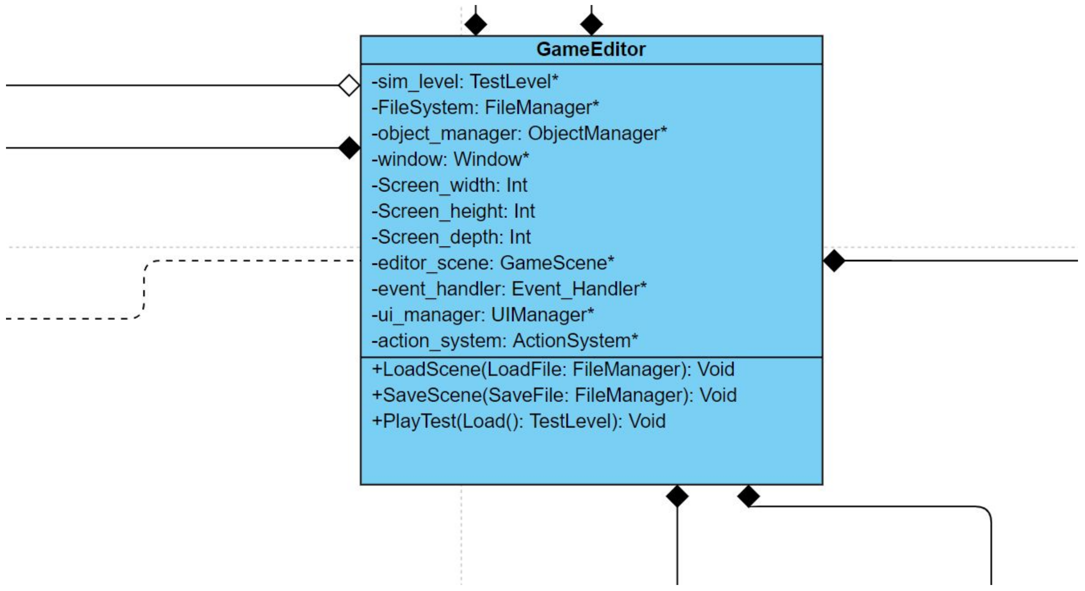
Arifa Tyrell
Sarfaraz Syed
Dong Hyeong Lee
Tyler
Violet
Arielle

Table of Contents

UML Diagram


Visual Paradigm Online Express Edition
<<Interface>>
Action
+Do(inAction: const std::shared_ptr<Action>): Virtual Void
+Undo(): virtual Void
+Redo(): virtual Void
EventHandler
-InputEventsDeque: <deque>*
+QueueEvent(Event): void
+ValidateEvent(): Action
ActionSystem
-redoStack: std::stack<std::shared_ptr<action>
-undoStack: std::stack<std::shared_ptr<action>
+Do(inAction: const std::shared_ptr<Action>): Void
+Undo(): Void
+Redo(): Void
TestLevel
-sim_level TestLevel*
+PlayTest():void
GameScene
-window: Window*
-renderer: Renderer*
-Screen_width: int
-Screen_height: int
-Screen_depth: int
-event_handler: EventHandler*
+Update():void,
+Render():void
+RegisterEvent():void
+GameScene(Window, Renderer)
GameEditor
-sim_level: TestLevel*
-FileSystem: FileManager*
-object_manager: ObjectManager*
-window: Window*
-Screen_width: Int
-Screen_height: Int
-Screen_depth: Int
-editor_scene: GameScene*
-event_handler: Event_Handler*
-ui_manager: UIManager*
-action_system: ActionSystem*
+LoadScene(LoadFile: FileManager): Void
+SaveScene(SaveFile: FileManager): Void
+PlayTest(Load(): TestLevel): Void
FileManager
+Object: std::vector<T*>
+LoadFile(directoryPath: std::string, object: std::vector<T*>): Void
+SaveFile(j_: json, fileName: std::string): Void
ObjectManager
object: Object*
+MoveModel(newPosition: Object)
+scaleModle(newScale: Object): Void
+rotate(newRotation: Object): Void
+Select(): Void
+LoadModel(loadObj: FileManager): Void
+DeleteModel(): Void
CharacterManager
-character_stats: int (Some number of stats)
+ManageStats(statName: string, statChange: int): void
+SaveStats(SaveFile(): FileManager)
+LoadCharacter(LoadFile(): Filemanager)
<<Interface>>
UIComponent
+RenderUI(): Virtual Void
+DeleteUI(): Virtual Void
+LoadUI(): Virtual Void
ScenceHierachy
+SceneHierarchy(ObjectList: &std::vector<Object>): ScenceHierachy
+RenderUI(): Void
+DeleteUI(): Void
AssetsList
+ObjectHierarchy(nameeFile: &std::vector<std::string>): AssetsList
+RenderUI(): Void
+DeleteUI(): Void
UIManager
+LoadUI(): Void
+RenderUI(): Void
+DeleteUI(): Void
ReadDirectory
-stringVec: std::vector<std::string>
-stringObject: std::string
-nameFile: std::vector<std::string>
-Return: std::vector<name: std::string& >
+ReadDirectory(): Void
+read_directory(const std::string& name, v: string Vec&): Void
+listFiles(Directory_: std::string): Void
+readFile(name: const std::string&): Void
Visual Paradigm Online Express Edition

Game Editor


GameEditor
-sim_level: TestLevel*
-FileSystem: FileManager*
-object_manager: ObjectManager*
-window: Window*
-Screen_width: Int
-Screen_height: Int
-Screen_depth: Int
-editor_scene: GameScene*
-event_handler: Event_Handler*
-ui_manager: UIManager*
-action_system: ActionSystem*
+LoadScene(LoadFile: FileManager): Void
+SaveScene(SaveFile: FileManager): Void
+PlayTest(Load(): TestLevel): Void

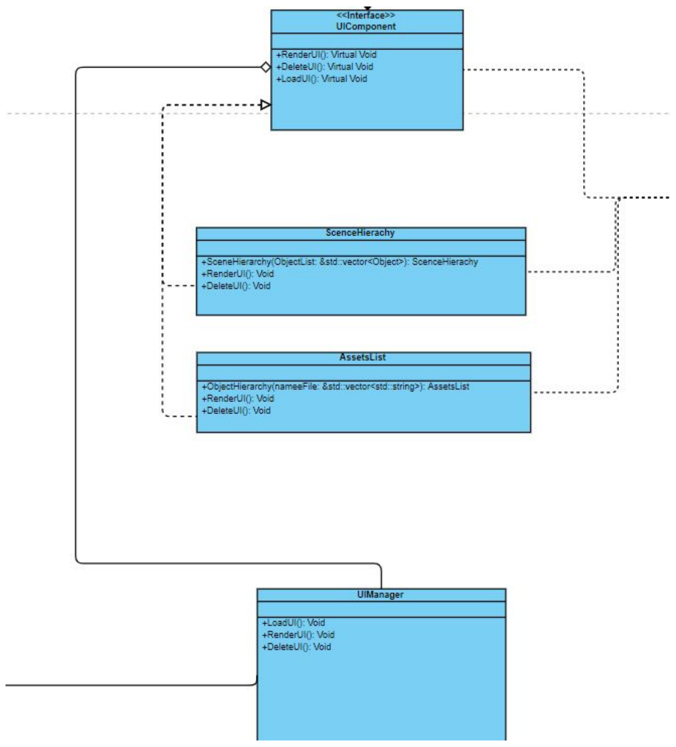
Overview

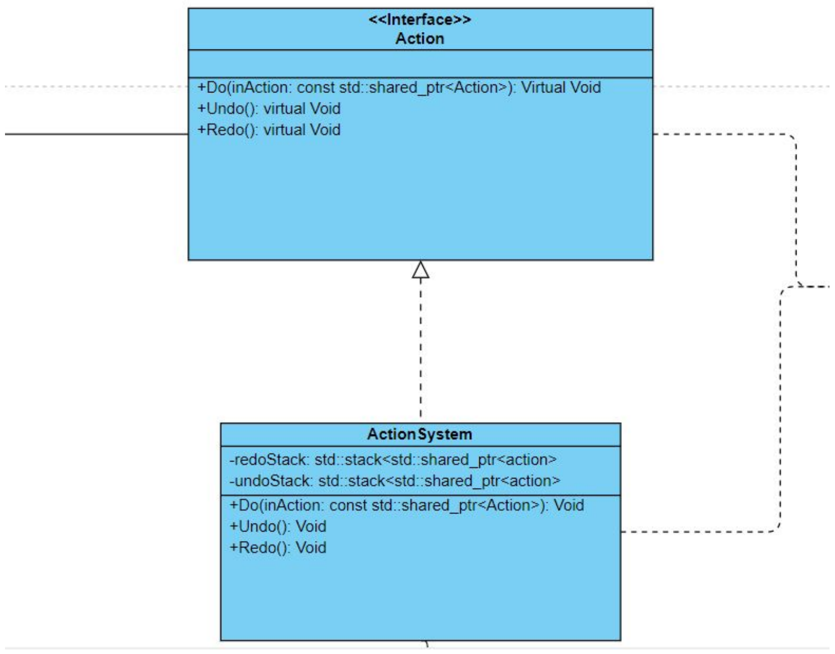
The GameEditor class is used to consolidate all the other classes. When input is processed, the GameEditor class will call methods that act as the main features of the editor, from loading a file by calling LoadFile() in the FileManager to MoveModel() in the ObjectManager.

Main Functionality

- Load in a level by reading a .json file
- Save data to a .json file
- Pass a scene's object hierarchy into an integrated version of the game engine to play test levels on the fly
- Call to undo and redo actions performed by the user

Game Editor Features

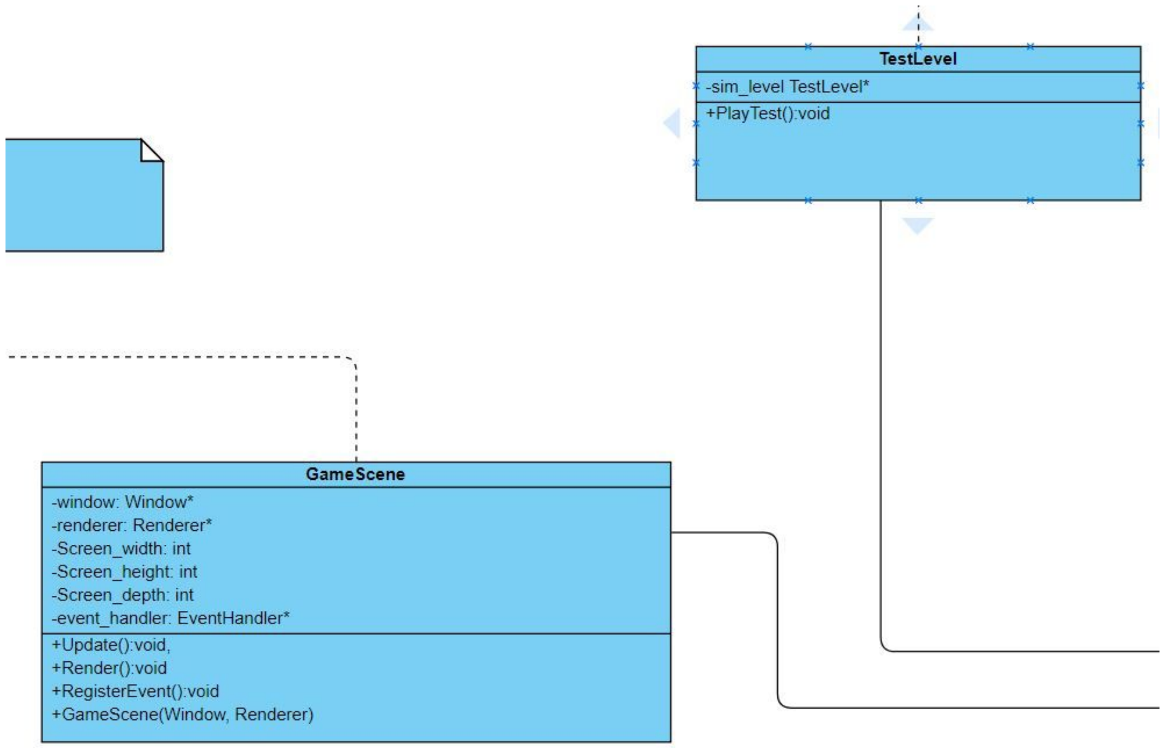



<<Interface>>
Action
+Do(inAction: const std::shared_ptr<Action>): Virtual Void
+Undo(): virtual Void
+Redo(): virtual Void
ActionSystem
-redoStack: std::stack<std::shared_ptr<action>
-undoStack: std::stack<std::shared_ptr<action>
+Do(inAction: const std::shared_ptr<Action>): Void
+Undo(): Void
+Redo(): Void

Game Editor Features

These are some of the classes that will add usability to our game editor. The action system will allow for undo and redo actions to be taken. The actions may include placement, movement, scale, and/or rotation of the object. The UI Component will allow us to view and execute commands, as well as access files in a more organized manner. This allows us to get the placeable objects from the assets file and display it in the Assets List UI which will contain a list of all assets. The Scene Hierarchy will display all the current objects that are in the editor scene.

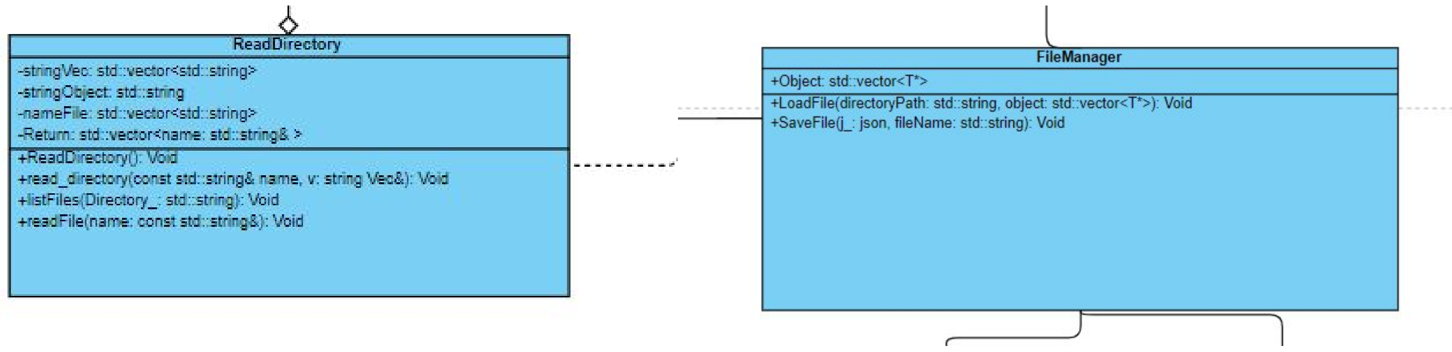
Level Loader



Description

The level loader will take the information from the game editor to load the level into a different window which will then call Render() and take eventHandler from GameScene (test scene from editor). The level loader will enable the user to examine and evaluate the scene. The game editor will place in a list of objects that contain information such as the name, position, scale, and rotation of the object. Once in place, the level loader will go through the list and use the information to instantiate the objects in the scene.

File Reader and Writer


ReadDirectory
-stringVec: std::vector<std::string>
-stringObject: std::string
-nameFile: std::vector<std::string>
-Return: std::vector<name: std::string& >
+ReadDirectory(): Void
+read_directory(const std::string& name, v: string Vec&): Void
+listFiles(Directory_: std::string): Void
+readFile(name: const std::string&): Void
FileManager
+Object: std::vector<T*>
+LoadFile(directoryPath: std::string, object: std::vector<T*>): Void
+SaveFile(j_: json, fileName: std::string): Void

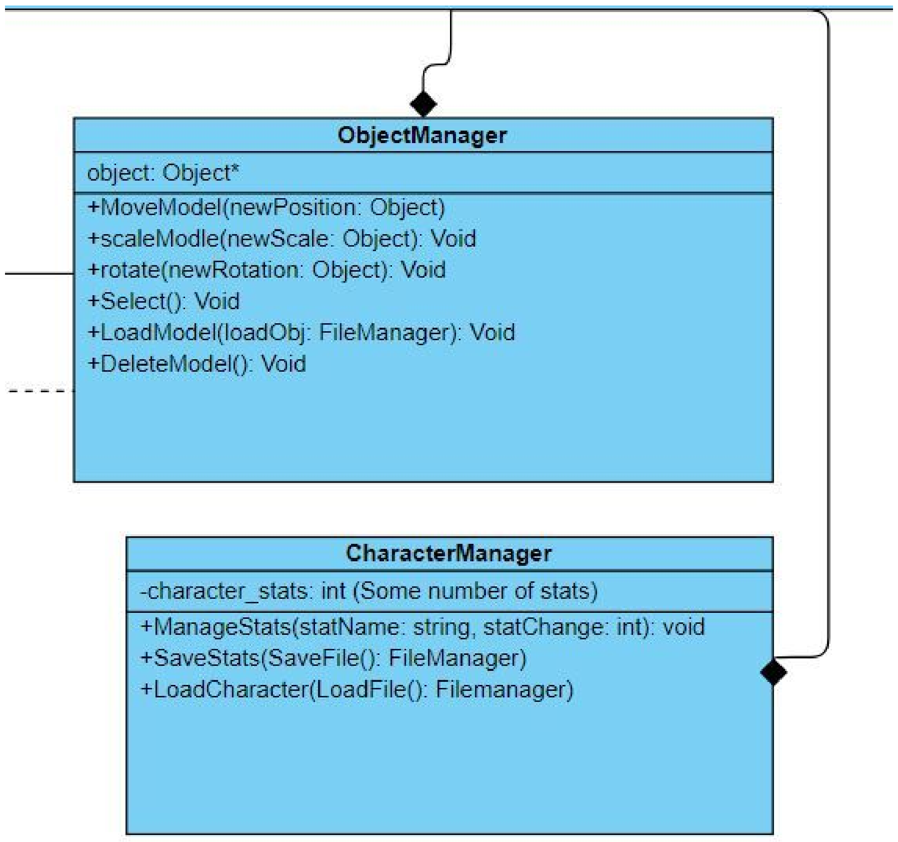
Overview

The file reader and writer loads .json files for the level and character. This allows for quick, easy changes without being required to edit raw code. The class is designed to read object data for use by the scene as well as save and export data to a file for later use in the game or in the editor scene. FileRead and FileWrite will be made to read in any kind of valid file type and reject any types that can't be used by the editor. FileRead and FileWrite are virtual and will be have bespoke implementations for each usable file type.

ReadDirectory

The ReadDirectory enables the ability to freely save and load files to/from an exact location, allowing for easier viewing in the editor scene. It will be given a directory to read and scan its contents for relevant file types, then put them into a data structure to be used by various UI elements. Saving will prompt what directory the file will be saved to as well as a name for the file, or save an already existing file to its current location.

Object & Character Manager


ObjectManager
object: Object*
+MoveModel(newPosition: Object)
+scaleModle(newScale: Object): Void
+rotate(newRotation: Object): Void
+Select(): Void
+LoadModel(loadObj: FileManager): Void
+DeleteModel(): Void
CharacterManager
-character_stats: int (Some number of stats)
+ManageStats(statName: string, statChange: int): void
+SaveStats(SaveFile(): FileManager)
+LoadCharacter(LoadFile(): Filemanager)

ObjectManager

Object Manager is a subsystem from filemanager which manages objects (such as .obj files) in the game editor scene in order to adjust their position, scale, and rotation. This class will also determine if an object has been selected or not. All kinds of subsystems that deal with physical objects have to pass through the Object Manager.

CharacterManager

This will allow for character stats to be individually changed on the fly. It will open up a window and override the original with the newly established one based on the stat given.