**Student:** Julian Nodarse, Florida International University  
**Mentor:** Wei Zeng, Assistant Professor  
**Instructor:** Masoud Sadjadi, Florida International University

---

**2- System Overview**

- Current multiplatform game engines do not have intuitive options for new game developers to experiment with.
- There is a lack of complete open source game engines.
- Goal of this project is to create an accessible easy to use, free game engine.

**2- Problem**

- There is a need for the user to interact with game assets in an easy way without worrying about the implementation of an actual game asset.
- The system does not have a way to keep track of all the actions and events that take place as the game engine is running.

---

**3- Current System**

- **Ogre 3D** is an open source rendering engine. While it provides extensive rendering functionality, it does not provide many of the other features for game design.
- **Moai** provides open source game engine that allows the user to create the game with Lua. This forces the user to only use Lua paired with the unique architecture makes it impractical for larger scale games.
- **The Unreal Engine** is a very popular game engine used throughout the industry. However, its very high cost typically make it an unreasonable choice for smaller studios.

---

**4- Requirements**

Requirements accounted for by my role

- The system shall implement an asset management subsystem which will allow data to be loaded onto game engine memory in an efficient manner.
- The system shall include a debug printing and logging subsystem that will display a data log in a text file that include errors and messages for the user.

---

**5- System Design**

**Singleton** – Used by Logger and AssetManager. This will allow unnecessary productions of extra classes.

**Singleton Abstract Factory** – Used by AssetManager. Supports user friendliness with no having to worry about how Assets are implemented. The AssetManager allows users to not have to worry about how Assets are loaded.

---

**6- Object Design**

**Asset**

- `dataName`: String
- `dataRefCount`: int

**AssetManager**

- `assetStorage`: Map `<String, Asset>`
- `currentAssetName`: String
- `currentAssetLocation`: String

---

**7- Implementation**

- We created our project using **C++**. We used **CMake** to be able to use our code in multiple platforms.
- With **generics**, the user does not have to know which type of asset is being loaded.
- Assets will be used by the Audio Management, Scripting, and Rendering subsystems. These subsystems will inherit the Asset's load method and will implement their own way to load onto memory.
- The **AssetManager** will keep track of which assets are being used and which are not needed.
- The **Logger** can be used by the users and all other subsystems to record messages or error logs.

---

**8- Verification**

Google Test was used for testing out system. We created our tests at runtime and were developed as implementation of the system was taking place.

By having a bottom-up approach, we could identify bugs and issues early on before proceeding.

---

**9- Summary**

Creating a new game engine will be the approach our team has taken in order to allow flexibility in meeting our goals of user-friendliness, cost efficiency and accessibility. The game engine requires an easy way for users to load game assets such as sounds and images. The AssetManager is a singleton abstract factory for users to not need to worry about the implementation of each asset. The manager will handle the loading of each asset and will ensure no redundancies are in the system in order to improve memory efficiency. The Logger is responsible for giving the other subsystems as well as the user a way to print error messages or any data log. By using a bottom-up approach with google tests we were able to test our system as we progressed.

---

**Acknowledgement**

Special thanks to my team: Michael Weschler, Antonio Diaz, Brian Lara, and Robert Law