Architecture Document

Milestone 1

Team Name: Sang Rin 
Game Name: MANZO
Score: _____/45

1) List the classes that you'll be using in regards to gameplay with a short description of each class's purpose. (10 pts)

Component

- ParticleManager - manages particles. Not changed from cs230
- Timer - Not changed from cs230.
- Beat - It counts beat and adjusts judgment of beat
- AudioManager - manages sound and SFXs. it loads, gets, sounds.
- Mouse - this class is about the mouse effect. Click effect, follow effect
- Skillsys - It manages the skill of mouse right click. It only
- Collision - It checks the collision of polygon and rectangle. Also, add a draw call to render.
- ShowCollision - it checks whether to show collision and not.
- Background - It is more like a background manager. It contains data of background and add draw call to render.

Game Object

- Fish - Parse fish data from file and update their position.
- Reef - It is Rock. It is currently used to test map data.
- Ship - It is a player. There is move logic, collision resolve, and fuel system

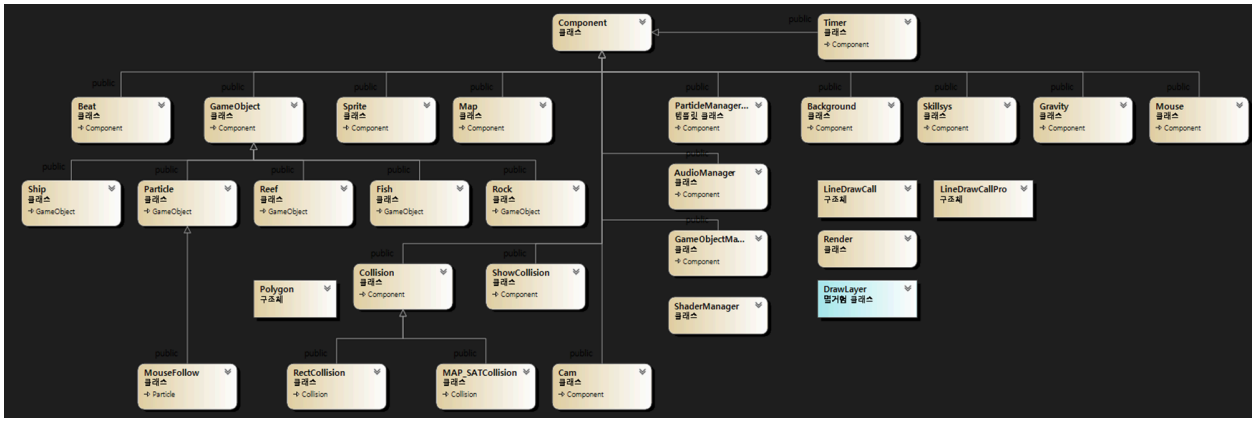
Game State

- Mode1 - stage of sea.
- Mode2 - stage of player house

Others

- Fishgenerator - generates fishes randomly
- Trail - structure to save data of mouse position and its lifetime.

2) Please describe your team's approach to leveraging any interfacing and inheritance to help with development. Additionally, do you foresee these as being useful features in your game's development? (10 pts)



We have refactored many of our engine features into interfaces using inheritance. First, there's the **Collision** class. Although we used it in a similar way before, our current code doesn't support circle collisions. Instead, we use **MAP_SATCollision**, which allows for various polygon collisions. This approach makes it much easier to handle different types of collisions with a single call process.

I also created a **Render** interface to centralize the drawing process. It accepts objects' draw calls and manages everything related to OpenGL rendering, such as matrices, shaders, and settings like glLineWidth, in a more flexible way.

Additionally, I introduced an **AudioManager** for handling music. Since this is a rhythm game, managing the audio is crucial. This system will help ensure smooth transitions between background music tracks.

Lastly, I implemented a **ShaderManager** to simplify shader loading and retrieval. I debated whether to write all the shader code inside Engine.h or create a separate shader manager. In the end, I decided that having a manager would be a better solution, and that's how the ShaderManager came to be.

```cpp
struct LineDrawCall {
    vec2 start;
    vec2 end;
    color3 color;
    const GLShader* shader;
};

struct LineDrawCallPro {
    vec2 start;
    vec2 end;
    color3 color;
    const float width;
    const float alpha;
    const GLShader* shader;
};

class Render {
public:
    Render() { CreatModel(); CreatLineModel(); };

    void AddDrawCall(const DrawCall& drawCall, const DrawLayer& phase = DrawLayer::Draw);
    void AddDrawCall
    (vec2 start, vec2 end, color3 color, float width = 1.0f, float alpha = 255.0f, const GL
    void RenderAll();
    void CreatModel();
    void CreatLineModel();

private:
    // Internal render method
    void Draw(const DrawCall& draw_call);
    void DrawLine(LineDrawCall drawcall);
```

```cpp
class RectCollision : public Collision {
public:
    RectCollision(Math::irect boundary, GameObject* object);
    CollisionShape Shape() override {
        return CollisionShape::Rect;
    }
    void Draw();
    bool IsCollidingWith(GameObject* other_object) override;
    bool IsCollidingWith(vec2 point) override;
    Math::rect WorldBoundary_rect();
private:
    GameObject* object;
    Math::irect boundary;
};

class MAP_SATCollision : public Collision {
public:
    MAP_SATCollision(Polygon boundary, GameObject* object);
    void Draw() override;
    CollisionShape Shape() override {
        return CollisionShape::Rect;
    }

    bool IsCollidingWith(GameObject* other_object) override;
    bool IsCollidingWith(vec2 point) override;
    Polygon WorldBoundary_poly();
private:
    GameObject* object;
    Polygon boundary;
};
```

```cpp
class AudioManager : public CS230::Component {
public:
    AudioManager();
    ~AudioManager();

    bool Init();
    void CleanUp();

    Mix_Chunk* LoadSound(const std::string& filePath, const std::string& name);
    Mix_Music* LoadMusic(const std::string& filePath, const std::string& name);

    void PlaySound(Mix_Chunk* sound, int loops = 0);
    void PlayMusic(Mix_Music* music, int loops = -1);
    void StopMusic();
    //Mix_Music* GetMusic(const std::string& name);
    double GetCurrentMusicTime();

private:
    std::map<std::string, Mix_Chunk*> soundEffects;
    std::map<std::string, Mix_Music*> musicTracks;
};
```

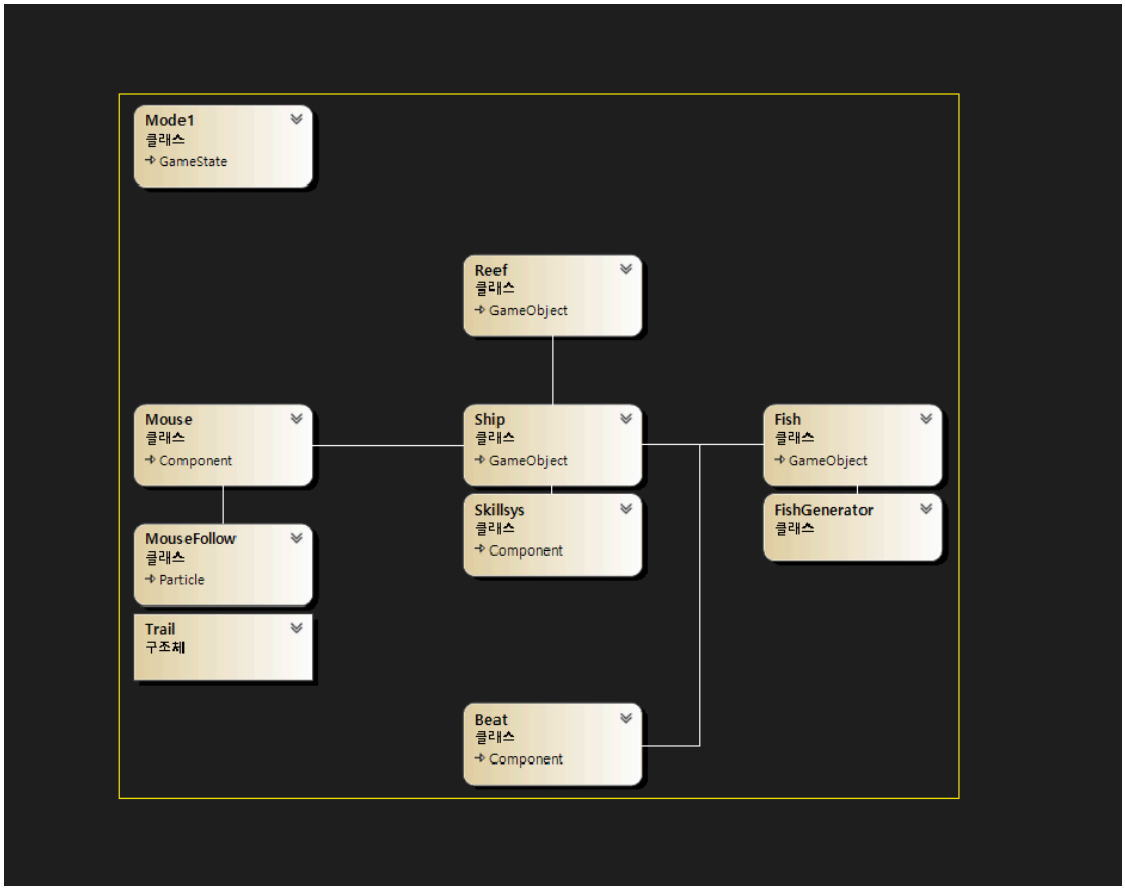
```
class ShaderManager {
public:
    ShaderManager();
    GLShader* LoadShader(const std::string& shader_name, const std::string& vertex_path, const std::string& fragment_path);
    GLShader* GetShader(const std::string& shader_name);
    GLShader* GetDefaultShader();
    void Clear();

private:
    std::unordered_map<std::string, std::shared_ptr<GLShader>> shaders;
    std::shared_ptr<GLShader> default_shader;
};
```

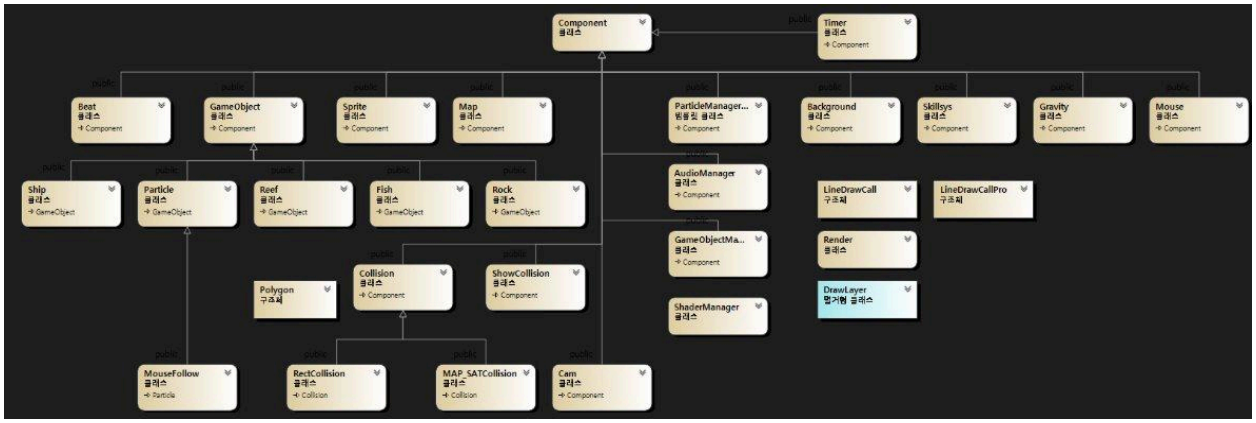
3) Are you using any Factories in this project? If so, please describe them. Additionally, please describe any factories that you think your team may need in the future for this game. (5 pts, even if you don't include them.)

The Fish class embodies characteristics of the factory pattern, serving the purpose of generating various types of fish. In its constructor, it randomly selects a fish type from the fishBook vector and initializes properties such as velocity, scale, and texture file accordingly. This design allows users to create fish objects simply by calling the constructor, without needing to know the details of fish creation, thereby providing abstraction in object creation. Furthermore, when a parent object is present, it initializes properties by inheriting from the parent, allowing for flexibility in generating fish under different conditions. This centralized fish generation logic simplifies future modifications to fish types or attributes. Consequently, while the Fish class may not be a traditional factory class, it operates like a factory by abstracting object creation and facilitating the easy generation of fish with diverse properties.

4) Please include a simplified diagram that helps visually illustrate your gameplay's current architecture. (Think of something like a UML Class Diagram, but a little more simplified.) You must also include a description for the interactions of the systems within this diagram. (20 pts)



In Mode1, Ship(Player) gets input from Mouse, which has an effect represented by MouseFollow and Trail. When it gets the position from Mouse, Ship moves toward that position when the Beats on. And FishGenerator generates several types of Fish. If the Player collides with them, the Player earns money. If the Player collides with Reef, the Player gets damage. This is basic game logic goes.



In the engine, the structure is organized as follows. All managers and main tools are encapsulated within a component inside the engine. This component includes several managers, such as the Map Manager, Sound Manager, GameObject Manager, and Shader Manager, among others. All of these managers are children of the component. The objects used in the game are generated by the Object Manager, meaning that the objects themselves are also considered components.

Map Generation and Management

When you draw the map using an external drawing program, the Map Generator parses the vector positions and creates a Rock object for each shape. The Rock contains a Polygon structure, which has the following member variables:

```
struct Rock {
   Polygon polygon;
};

struct Polygon {
   std::vector<vec2> vertices;
   int vertexCount;
};
```

The Rock is managed by the Object Manager, just like all other game objects. When the manager calls the DrawAll function, it adds a draw call to the rendering class and handles the collision detection for the Rock.

Drawing Game Objects

The drawing process for game objects follows a similar pattern. Each frame, a draw call is added to the render system, which organizes these calls into layers:

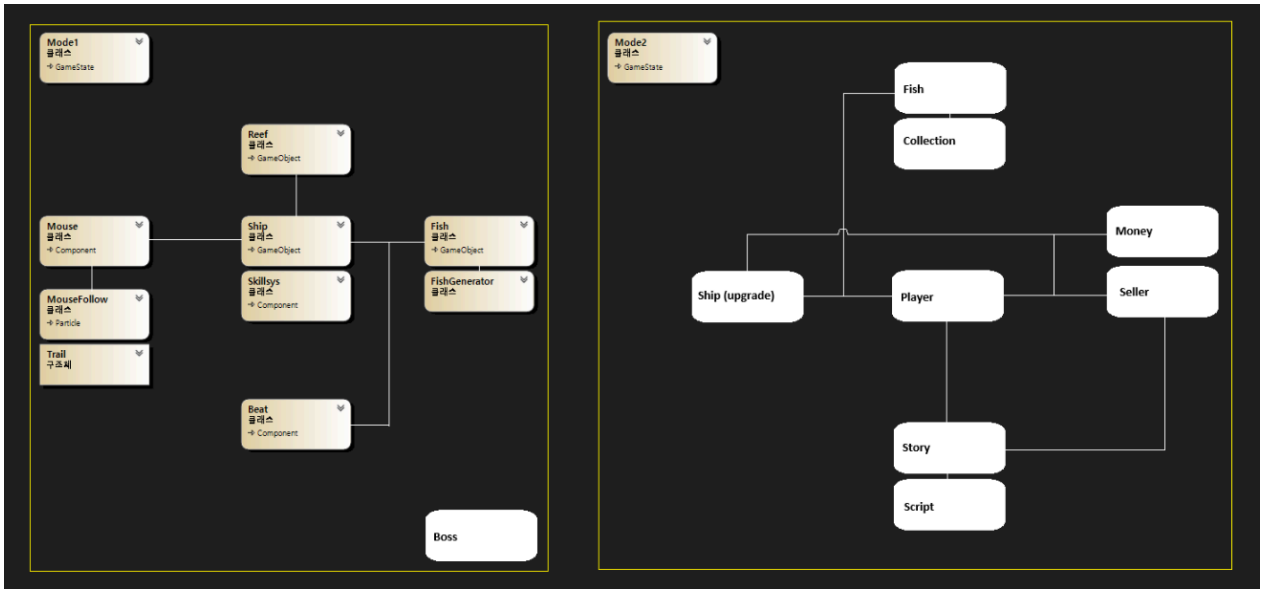
```
std::vector<DrawCall> draw_background_calls;
std::vector<DrawCall> draw_first_calls;
std::vector<DrawCall> draw_calls;
std::vector<DrawCall> draw_late_calls;
```

When all draw calls for the objects are collected, the render system processes them to draw everything within the specified layers.

Beat Class and Object Movement

The Beat class tracks whether the beat is currently active. The Ship takes data from the Beat class to synchronize its movement with the rhythm of the music.

5) Please include a simplified diagram to help visually illustrate what your final project's architecture will look like. (This is just an estimate, so it's okay if your final project does not turn out like this!) (20 pts)



- In Mode1, Ship moves on the beat along the position it gets from the mouse click. If Ship collides with Fish, the player earns money. But If Ship collides with the Reef, Ship gets damaged and bounces off. After all this process, Ship can move down to the deep sea. If the Ship arrives in the boss zone, the Boss fight starts. Along this process, Ship goes deeper and Player earns more money and elements for upgrading. Additionally, there is a Fish object and a FishGenerator, indicating that the player interacts with or collects fish in some capacity. The Reef serves as another game object, possibly tied to the game environment, while a Boss is present, suggesting a major challenge or objective for the player.

- Mode2 is for the player's house. The focus shifts to upgrading and progression. The Player can upgrade the ship here, and sellers visit to sell some items to the player. They cost money or special fish. Also, players can check a collection of fish and exhibit them in a fish tank. Since NPCs are only available in mode2 (player house), all the script about story will interact with mode2