Technical Design Doc
Graphics Team

Gabriel K

Matthew F

Ruan

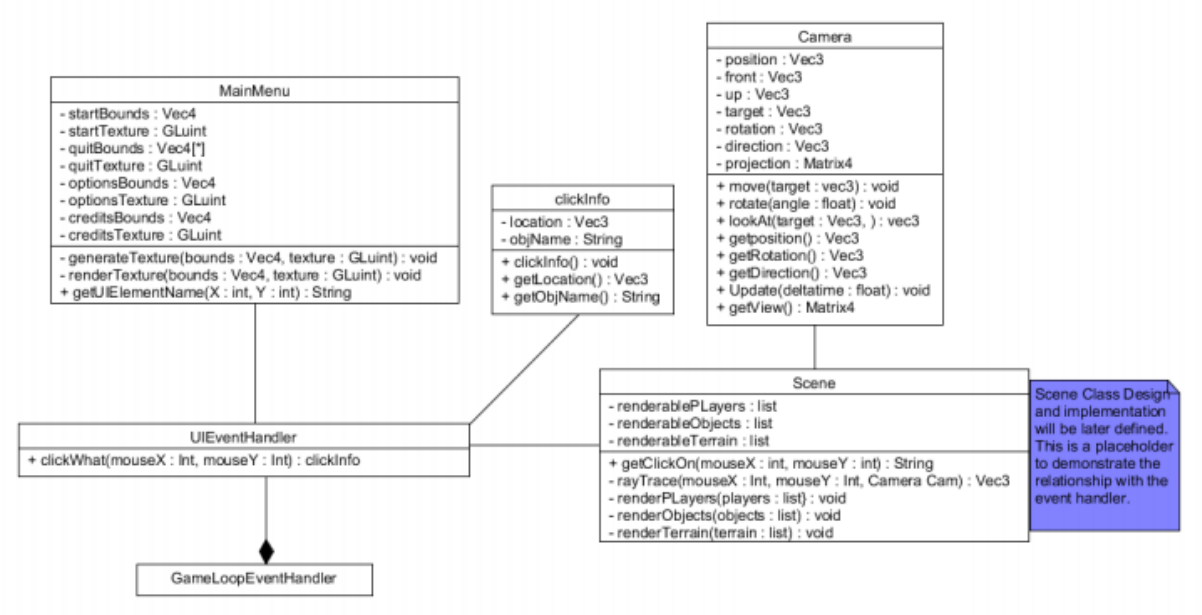
Peter A

Gabe G

Marco C

Brandon P

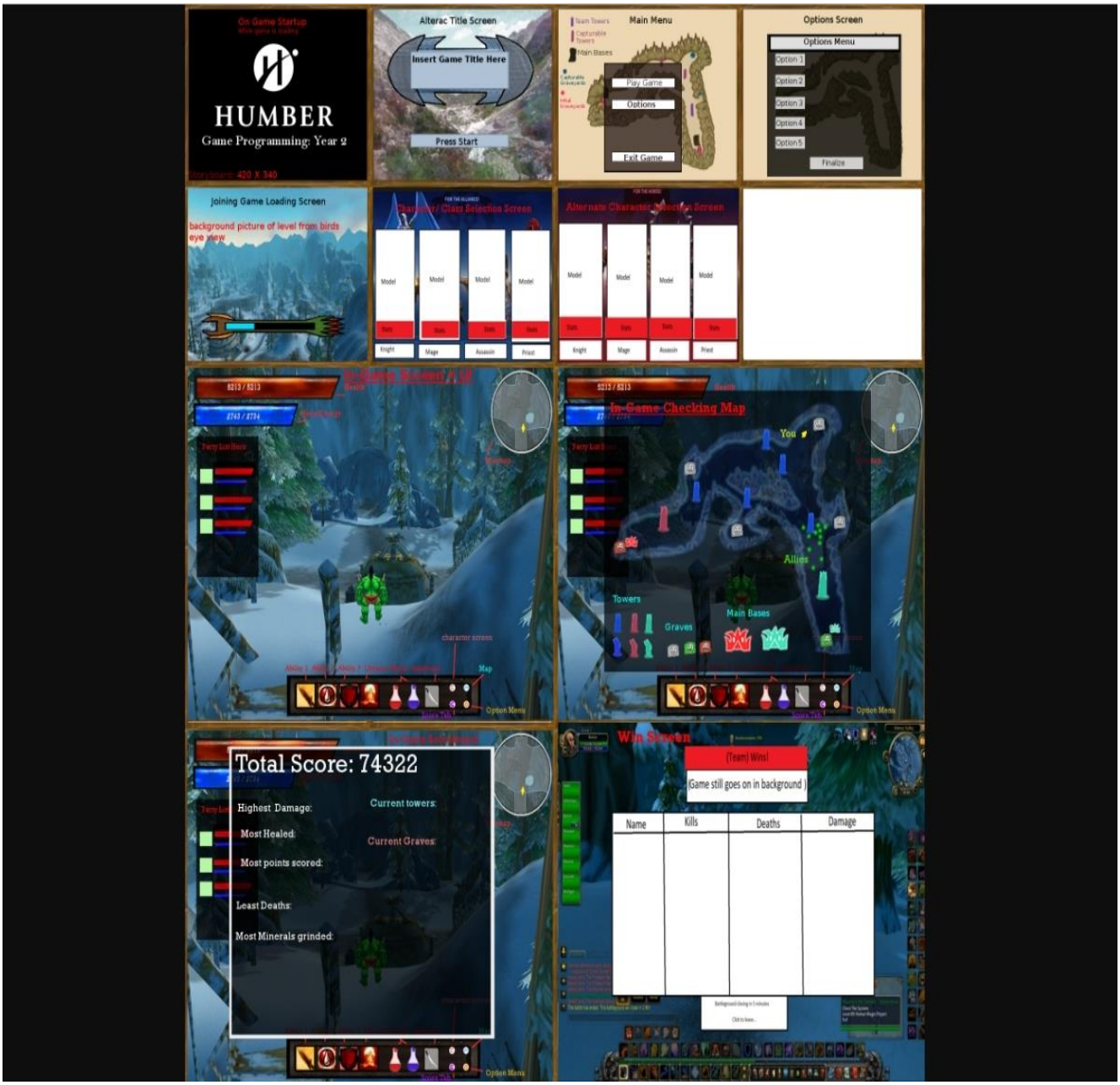
EventHandler UML



Description

The Eventhandler will check input and on mouse click, it will take the mouse's coordinated and compare it with the UI elements. It will then return The UI element's name, or raytrace for an object in the worldspace and return the name of that object. The game loop can use this name (or ID) to determine what action to take.

Story Board Template



UI Events and Functionality

Scene Class:

- needs to have a UI object defined
- calls UI->OnCreate() in Scene::OnCreate()
- calls UI->Update() in Scene::Update()
- calls UI->Render() in Scene::Render()
- call UIManager->HandleEvents() when the player clicks on an interactable UI element or presses the button to pull up a sub menu

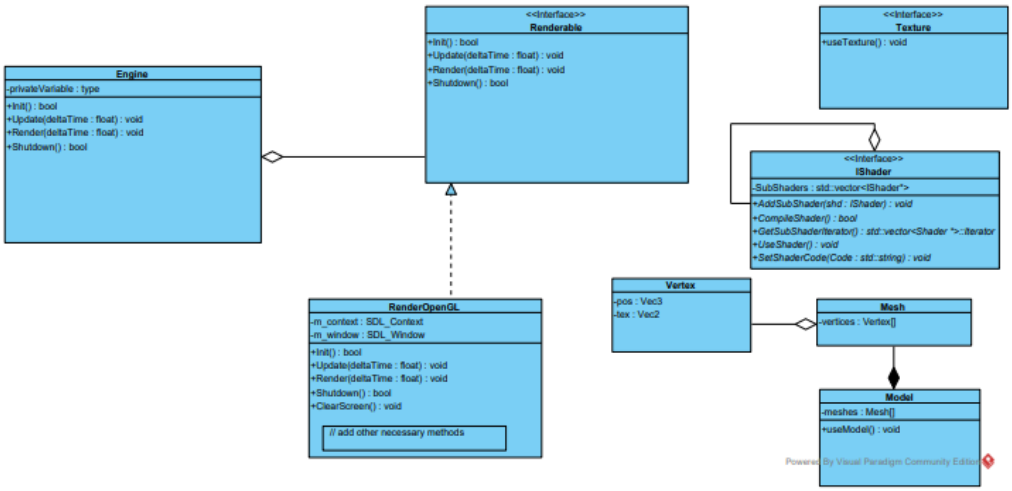
UI Manager Class:

- manage the UI changing an element(s) when:
    1) A mouse event (i.e. OnMouseOver(), OnMouseDown, and OnMouseUp()) occurs
    2) Key event (OnKeyDown(), OnKeyUp()) occurs when the player has been hit or killed
    3) When the player uses an item or ability
- manage the UI completely changing when loading between the game and menu
- have a sub menu when the player presses a certain button

Description

The Scene class treats the UI as an object that and calls its methods in the corresponding classes. The scene creates, updates, renders and destroys the UI. The UI Manager class is called by the scene and affects the UI based on the input it receives. It affects what parts of the UI are rendered on the screen such as a larger map.

Renderable Protocol UML Class



Description

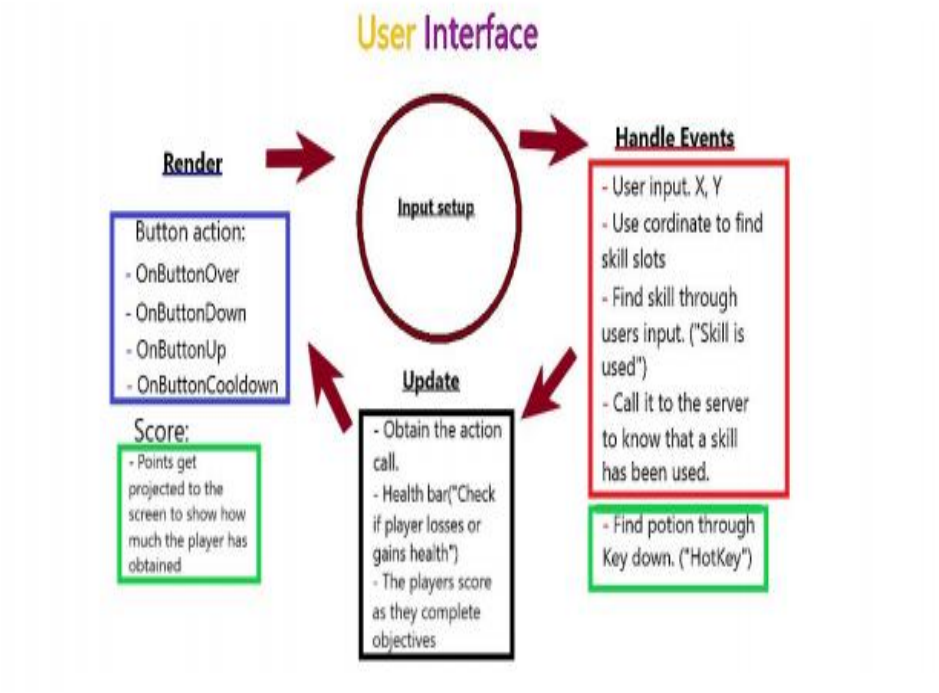
Renderable Protocol:

This section is the renderable protocol class diagram that holds the interface for all the other classes to implement. The Interface class IRenderable will be a pointer by the Engine. The RenderOpenGL is a class that can implement IRenderable but can also be used by another graphics API like DirectX. The other interface classes are for each member of the protocol (Texture, Shader, Vertex and Mesh) all of those are used on the Model class to be rendered therefore.

Shader tree:

We decided to use a shader tree to achieve a rich dynamic shader system. Essentially the root shader would have the main GLSL function and inside that function, forward declare several common GLSL functions like GetPos() GetColor(). In your leaf node include a GLSL shader that define the previously forward declared shader. The process can end here but one can extend this ShaderA uses ShaderB and ShaderC and ShaderB uses ShaderD and shaderE and etc

User Interface



Description

This is the main loop for the game. What i have done for the group is make a Chart for the loop regarding the User Interface. This is what the game would need for the UI to work as the game continues to progress.