


Stuart Heath
Grads In Games 2021-2022
Programming
Stuart@Stuartheath.co
[LinkedIn](#)
[Game Link\(Itch\)](#)

[Trello](#)
[Report](#) - Latest live version(This docx created 22/01/22)
[GIT](#)  **132 hrs 43 mins**
[Youtube preview](#)

Concept Outline

I thought through a number of different ways to extend the basic mechanics, penalty for failure, extra lives, unlocks different ways of performing a “unlocking minigame”, collectables etc however, I wanted to do something a little different. I’ve made rendering engines and physics engines using OpenGL previously and have spent some time researching game engine development but have not yet had a good opportunity to take this further. I’m currently working as a tools engineer so spending some time to understand more of what goes on with the data under the hood, seemed like a good use of my time.

Planning and Pre-production

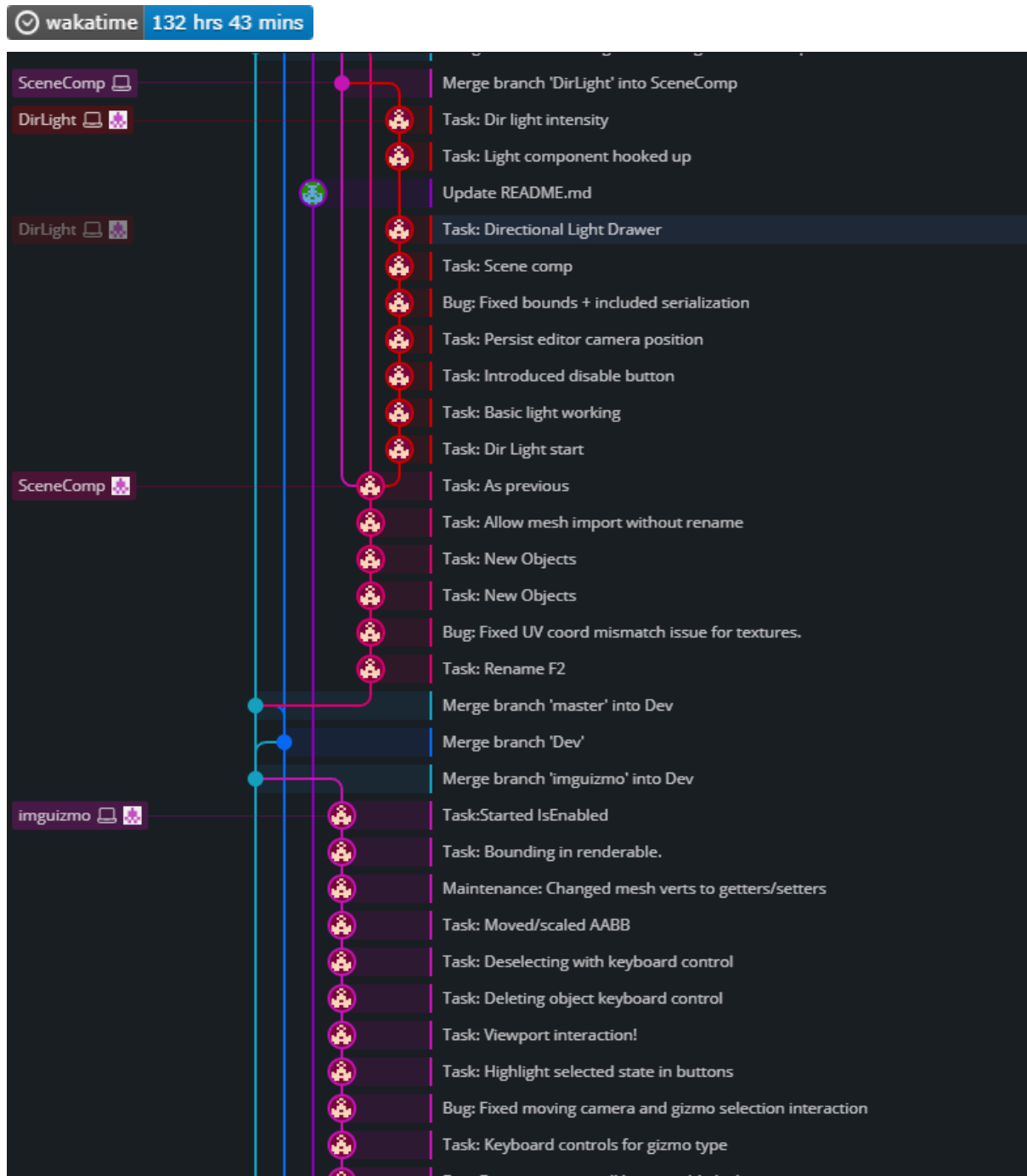
I collected a list of everything I was interested in adding to my Trello Board so that I could track my initial ideas as well as anything else identified on the way, including bugs/optimisations and improvements. After first getting to terms with the basic layout of the existing code base and having only worked with OpenGL in the past from a rendering perspective, I spent my first week or so researching DirectX and making small adjustments to the existing project to solidify my understanding of what was going on. Adding a new renderable, changing their positions, moving the “camera” etc.



(A snippet of my trello board at time of writing)

Source Control

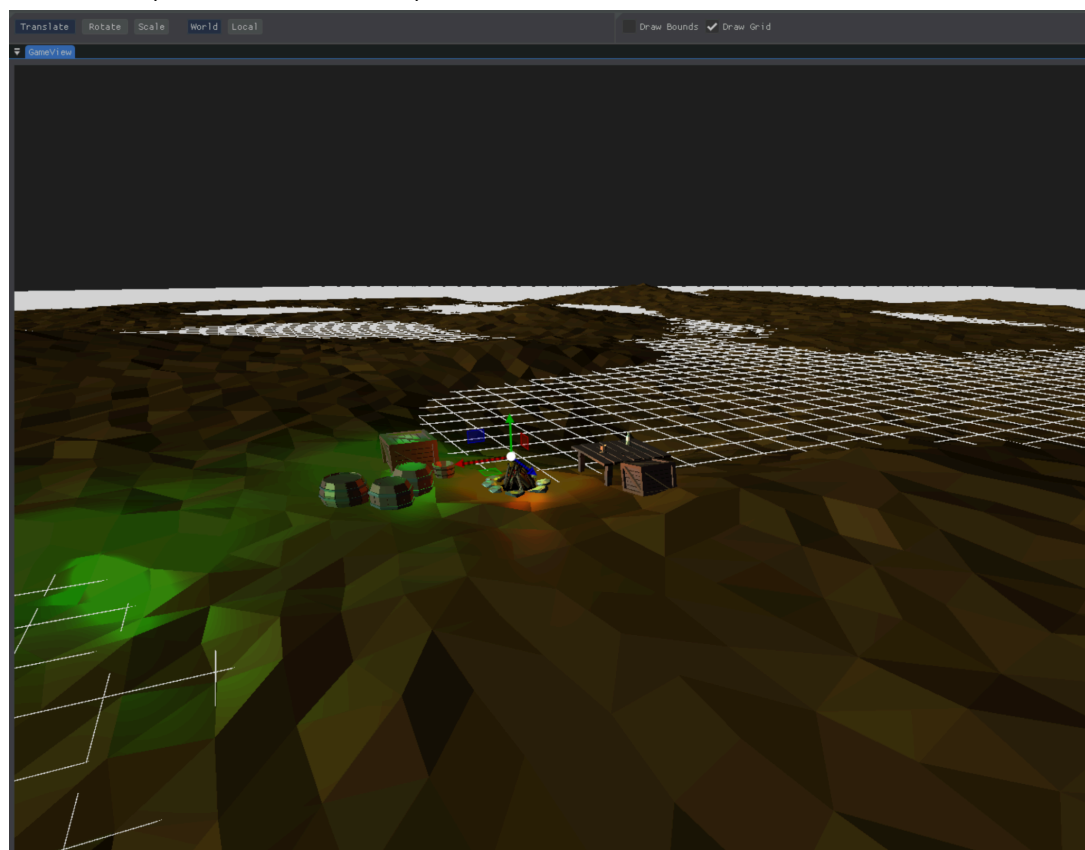
Source control is always key to any project. Although I am happy using command line, I do enjoy GitKrakens UI. There was a number of instances where I really benefited from being able to develop on separate branches as well as going back to earlier commits to understand what had changes (thus helping to resolve a few bugs along the way) I usually try to use accurate naming for commits although sometimes I end up just writing something like Fix which isn't great, but at least I have consistent commits I can reference. At the time of writing, I've performed 297 commits, over 21 branches between 11/11 & 14/1.



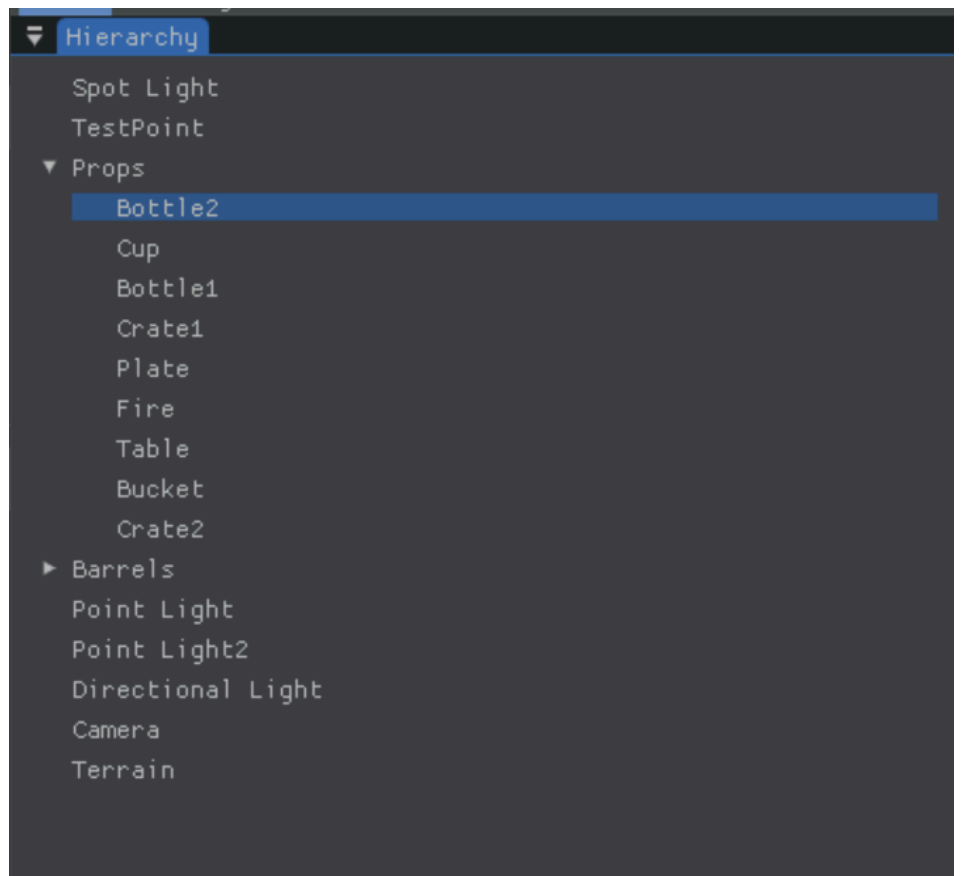
(Small extract of commits)

Key Functionality

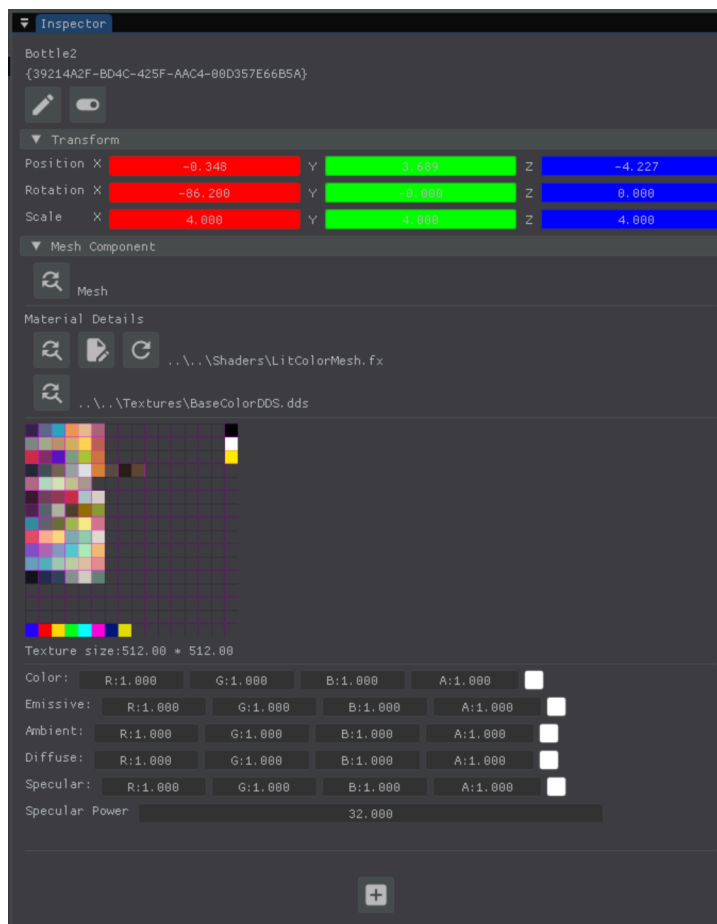
- 4 Projects(Engine/Editor/Game/Gamebase)
- Engine (Static Library)
 - 3D Rendering with basic Phong lighting
 - GameObjects
 - Components(Tranforms/Lights/Mesh)
 - Materials/Shader/Textures
 - Mouse/Keyboard input
 - Camera (Perspective + Ortho)
 - SceneManager
 - Gameobject/Component factory / “Builders”
 - Logging with Console/File and ImGui Sinks
 - Mesh Serializer
 - Scene Serializer
 - Material Serializer(In progress as stand alone asset, currently captured in gameobject serialization)
- Editor (.Exe)
 - ImGui Editor
 - GameView(with editor camera)



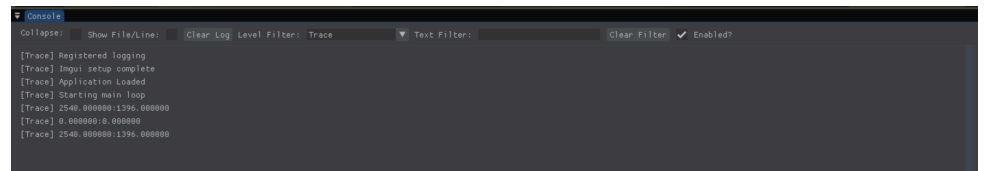
-
- IMGUIZMO controls with picking objects from sceneview using AABB/Ray
- Hierarchy
 - Add/Move/Remove gameobjects



Inspector



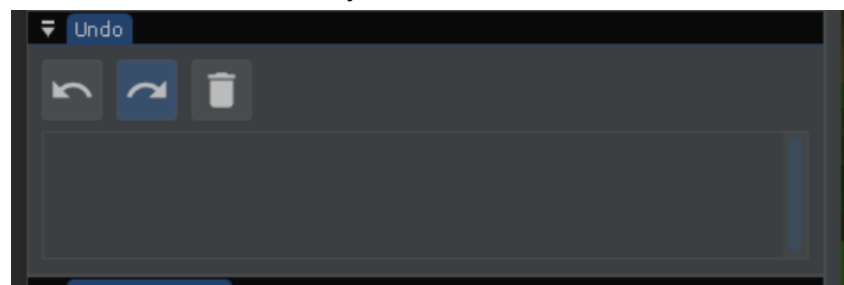
- Transform
- GUID
- Rename
- Component inspectors
 - Mesh
 - Sprite
 - Material
 - Texture
 - Shader
 - Hot reload of shaders during running
 - Directional Light
 - Point Light
 - Spot Light
- Add/Remove Components
- Debug Log/Console



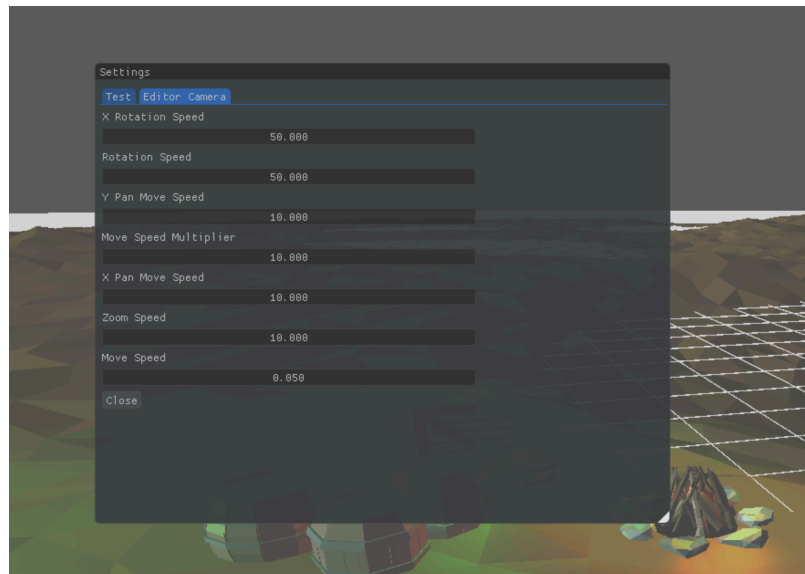
-
- Editor Camera Dedicated inspector



- Mesh Importer Tool
 - Using ASSIMP, converts FBX files from blender into my own binary filetype
- Input stats/ Rendering Stats/ Time Stats
- Undo/Redo
 - Using Command/Memento to record state for editor interaction to allow undo/redo with keyboard shortcuts and UI.



- Editor Settings Panel (Persistent editor state)



- GameBase(.Exe)/Game(Static library).
 - These two projects have the basic application framework for running a game. As it stands, it's only job is fire up the required engine functionality, resource library, input, scene deserialization. The game simply has reference to a scene file and loads this via the Gamebase.
 -

Project Difficulties

Although I've used C++ on various projects, I've always had more experience and prefer using C#, so my initial hurdle was getting back to grips with the language and I am in no doubt, not using the language to its full potential and trying to shoehorn concepts familiar to be in C# into C++. I've made extensive use of smart pointers and where appropriate avoid pointers altogether and use references, but there are still drawbacks to this. With the introduction of undo/redo, I used the memento pattern to capture the do/redo states, but this proved much more difficult in C++ compared to C# where you really don't need to worry and simply pass in copies of your undo data to revert to. Smart pointers both helped and hindered in this regard and certainly feel like I have a lot more to learn on this subject to be able to continue using this pattern in this situation in C++.

Another shortcoming in my knowledge is C++ build systems, I've had so many issues getting projects to compile correctly, with splitting the project into 4, implementing new libraries etc and if I did this again/when I carry on with this project, I will need to look into learning a build system such as cmake/premake to make this process a lot simpler.

Final Reflection

Whilst I have clearly gone a little off-piste with my submission, I'm hoping that the time and care I have taken in producing this little engine is at least considered. I've learnt a great deal, in terms of Dx11, engine architecture and C++ and I am very much looking forward to continuing its development as I head into my final year and beyond.

Thanks for taking the time to review my work and for the opportunity to participate.

Third Party/Premade Assets

- Libraries:
 - <https://github.com/nlohmann/json>
 - <https://github.com/ocornut/imgui>
 - <https://github.com/CedricGuillemet/ImGuizmo>
 - <https://github.com/assimp/assimp>
- Other resources
 - <https://fonts.google.com/icons>
 - <https://www.3dgep.com/texturing-lighting-directx-11/>
 - <https://github.com/microsoft/DirectXTK/wiki/Collision-detection>
 - <https://github.com/microsoft/DirectXTK/wiki/DebugDraw>