# “Mine Empire” Game Design Document

2/3/2020

# Overview

*Build an empire and explore a vast fantasy world…deep underground*.

Fantom, the adder, has enslaved the world with his dark magical forces of undead minions, and poison jungles. You must build a base and travel deep underneath the crust of the world to destroy Fantom, and create world peace.

# Game Design

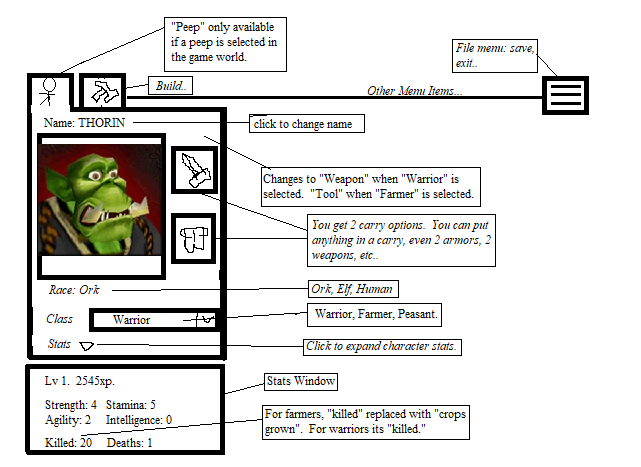
The theme is a modified high fantasy setting. Humans, Orcs, Elves, Dwarfs. You are given the option of having any of these guys in your base.

This game is like Dwarf Fortress. You are given farmers, peons, and warriors, and you must build your fortress, and fight against enemies, in order to reach “Deathstone fortress” and defeat Fantom, the boss of the game.

The game worlds are organized with different ground layers. Each layer of the world is a different game world.

The world editor will be like “legos” where, you’re able to place little blocks in the world that have preconfigured shapes. The world is a collection of density points, which are dependent on cube-like textures. The cubes are marching voxels, but they have slopes that allow us to create pathways for the peeps.

1. Open up the edit menu to enter edit mode.
2. Click a block to place it. Block will show transparent in the game world.
3. To orient the block, Click and hold the RIGHT mouse button then drag the mouse across screen to rotate the block. OR click and drag the rotation slider.
4. To place the block, click the left mouse button.



## Bricks and Blocks

There are only two fundamental bricks. Slopes, and Blocks. Additionally, we may give the user the ability to add density points at will, however, the only *necessary* bricks in the game are slopes, and blocks. Slopes allow you to carve pathways up and down the world for your peeps to move along. Blocks are just structural material. In addition to these two fundamental components, there are also decoration blocks. Those are blocks that have block collision topology, but they have some decoration. They can be anything (sconces for instance). They are also items.

Bricks are collections of these points (like Minecraft blocks, but with slopes). Bricks are bought with money. You get the money from the people in your world paying you taxes. So really, you’re like a government official. While you build things in your world to make it bigger, you’re attacked by a series of enemies whom you must fight off. Enemies can be anything. Probably, zombies, though. You fight enemies by training troops, and building various military tech. Military tech is the primary “new content” portion of this game, in addition to new weapons, ores, and treasures. Military tech evolves over time.

## Gameplay

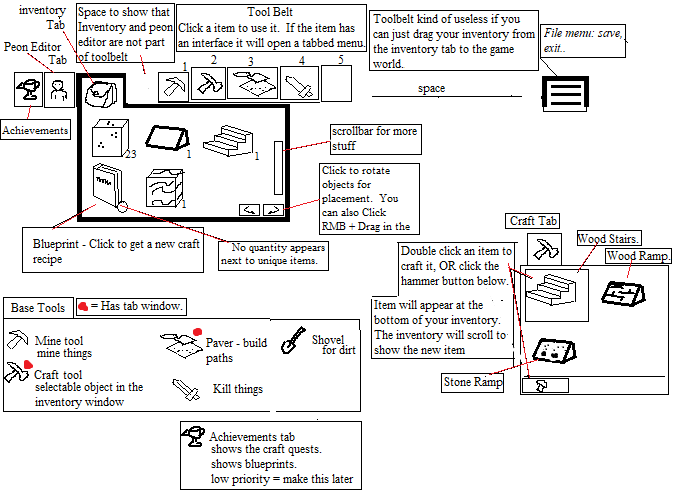
Your base is infinite. As your base grows, you’re able to travel to other parts of the world in search for different minerals. This is the “adventure” portion of the game. You’re able to adventure to different areas, and in these areas you get different materials that let you build your base stronger.

Gating in this game is dependent on the layers in the world. The overworld in the game does have some layers, however, the overall game itself is built on different layers, each layer serves as a gate. In order to get through to the next layer you need the ability to destroy blocks in that layer. To destroy blocks in that layer you need to equip your peeps with pickaxes and such that are strong enough to mine through those blocks. At the start, we’ll use just basic granite, and keep going down, until you get to diamond, and deathstone, and other made-up rock types that are stronger than diamond.

The peeps of your world have different classes. Laborers provide the necessities to grow your world. They provide food and such. Military guys are equipped with bows, and swords, and they’re able to kill off intruders. Then, there are some people that don’t do anything. If they don’t do anything you’re able to turn them into a laborer, or a worker.

# Interface

The interface is like a combination of an RTS and Minecraft where you have your own toolbelt, but you’re also in god mode.



# Implementation Roadmap

1. **Editor**
   1. User Interface
   2. Add the necessary UI components that let you edit the world. This involves fixing the UI. Programming the UI for the items that are available.
   3. Goal
      1. Goal is to create a UI that looks exactly like the nanogui ui. However, I’ve dived into NanoGUI and I want to keep using our own UI, but make changes to it to make it faster and more reliable.
         1. Interestingly, we started making this UI with the intent to give it exceptional detail with texture items. However, it turns out that because Material Design uses simple shapes, textures aren’t really needed in rendering.
         2. For implementation details nanogui’s *Button*::drawBodyTemp
      2. Removing the uDIM parameters, and hard-coding positions will make the UI far easier, and faster to layout.
      3. Remove Image-Centric rendering. See
2. **Peeps**
   1. Create and render simple peep objects in the game. These can be Minecraft block-like guys. It doesn’t need to be fancy.
   2. Get the peeps to travel along paths on the ground.
   3. UI – Peep Editor
      1. Create a “peep editor” screen that lets you edit a peep. The peep editor gives you the abilty to equip a peep with a weapon, and some armor. Peeps have weapons, and armor.
3. **Scenery**
   1. Create Tree Objects in the game.
   2. Update Renderer to handle detail (arbitrary) block geometry.

# Technical Roadmap (DEPRECATED)

# ~~PBR~~

~~The next phase of this graphics engine is to create a graphical PBR renderer like Blender or Unreal.~~

## ~~Tasks~~

~~Scripting Engine will support easier UI layout and game changes without recompilation.~~

~~Add a UI with multiple viewports to support the~~ **~~shader graph.~~**

~~Project file format getting rid of~~ *~~/~~*~~data directory.~~

~~Vulkan Support for faster rendering.~~

~~Add a graphical PBR interface for modern shading.~~

## ~~Phases~~

1. **~~Cmake~~**
   1. ~~Test CMAKE~~
   2. ~~Change the engine to use Cmake as is used by most modern C++ projects.~~
2. ~~Separate SDL window code so that we can have multiple windows.~~
3. **~~C# Script~~**
   1. ~~Embed C# with Mono. Mono readily supports embedding.~~
      1. [~~https://www.mono-project.com/docs/advanced/embedding/scripting/~~](https://www.mono-project.com/docs/advanced/embedding/scripting/)
      2. [~~https://www.mono-project.com/docs/advanced/embedding/~~](https://www.mono-project.com/docs/advanced/embedding/)
   2. ~~Remove “App” layer in place of C# callback function.~~
   3. ~~Remove “Bottle” project in favor of C# language callback.~~
   4. ~~C# callbacks to the C++ API~~
   5. ~~C# Built-In classes~~
      1. ~~Node~~
      2. ~~MeshNode~~
      3. **~~vec3, ivec3 etc..~~**
   6. ~~Compile by calling mono –~~
      1. ~~Save the output CIL to /data/cache/scripts.dll~~
4. **~~User Interface~~**
   1. ~~Description~~
      1. ~~A UI is required to save projects and for editing functionality.~~
      2. ~~The current UI isn’t well implemented and is abhorrently slow.~~
      3. ~~The largest feature change in the UI will be having~~ **~~multiple views~~**~~.~~
   2. ~~UI Design~~
      1. ~~The UI Design will be modeled directly off of Blender’s UI~~
         * ~~Menu – File, Tools, Help~~
         * ~~Toolbar - Left of screen.~~
         * ~~Draggable View Splitter~~
         * ~~Scene Outliner~~
         * ~~Properties Window~~
   3. ~~Views~~
      1. ~~Because we are modelling this off of blender we will create views that show data aspects. We are only going to implement views that correspond to our PBR system.~~
         * ~~Scene View~~
         * ~~Shader Editor View~~
      2. ~~Reorganize drawing code to work with multiple views.~~
5. **~~Project Layer~~**
   1. ~~Project File~~
   2. ~~“Add Project” Menu Item~~
   3. ~~Save the project.~~
6. **~~Vulkan Support~~**
   1. [~~https://vulkan-tutorial.com~~](https://vulkan-tutorial.com/)
   2. ~~Reorganize GL code into generic units (especially “ShaderBase”)~~
   3. ~~Research embedding SPIRV compiler (shaderc)~~
   4. ~~Abstract the Vulkan API through SDL~~
7. **~~Component Infrastructure~~**
   1. ~~We must change the game system to use components.~~
   2. ~~Change Shake to use class aggregation (components) like Unity. Adding components when needed.~~
8. **~~Render Pipeline Changes~~**
   1. ~~Shader Graph Editor~~
      1. ~~Shader Node~~
      2. ~~Shader Node Link line.~~
   2. ~~Create a new pipeline that follows PBR methodology~~
   3. ~~Albedo + Microsurface + Metal~~
9. **~~Physics Engine~~**
   1. ~~Rip out the physics engine in favor of Bullet’s physics engine.~~
   2. ~~Test BtVector3 vs Vec3x<> performance, and optionally, replace the math library.~~