NAFM Design Document

Alex Hicks, Brian Huntley, Matthew Neal

Intro to 3D Game Development

Andy Harris

IUPUI

December 7, 2017

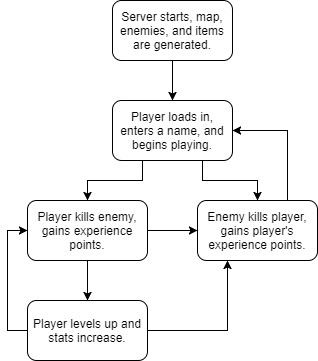
**Introduction**

**Not Another F^\*%!#@ MUD** (NAFM) was started and birthed as an idea for a multiplayer roguelike, hosted on a server with support for multiple different client machines, built in Typescript on the eta framework. Since Typescript was being used, it was necessary to build all functionality not available within eta directly for the game, including user input, rendering, entity management, and many other key components of a game. Due to the nature of a multiplayer game, SocketIO was utilized to form handshakes between client and server and make sure that multiplayer functioned properly. Due to it being hosted server side, all that’s needed to play is the link to the server, which is <https://alexhicks.net/nafm/>. Movement is handled either via WASD or arrow keys, and combat occurs by pressing an arrow key in the direction of an enemy. Overall, we were able to create a multiplayer dungeon, complete with AI enemies, items, and a boss, with a map that is randomly generated and populated each time. Enjoy!

**Engineering Plan**

Our original engineering plan was to start with Alex getting our server set up and configured, with our basic engine placed on top, and then have Brian and Matt get map generation and map based events and assets written, while Alex worked on server and client based communication and interaction. The idea was to start this the weekend after Thanksgiving, however, this ended up having to be put off until mid-first week of December. While we were wanting to spend more time planning and designing initially, we ended up using a more Agile approach and only designing out the minimum pieces necessary to proceed, and then designing what we needed afterwards. Our design began with the understanding of wanting a multiplayer roguelike, with a randomly generated dungeon. Once we were able to get the minimum viable dungeon assets and multiplayer functionality, we then designed combat, enemies, items, and the Binary Space Partition algorithm for further map generation individually, as we progressed and saw what needs might arise. Since we had planned to start earlier, we had initially expected to be able to spend more time on each part of the game, designing and planning more before testing implementations, however, in the end, with our reduced amount of work time due to unforeseen circumstances, we ended up growing our game as opposed to building our game, which worked in our favor in many aspects. Since we built the map and players first, we were able to get a good grasp on how many types and number of enemies we should create, since we then knew how big our map would be. From there, it was simply a matter of using our already built renderer and inserting all Entities into the map. Our initial timeline was to have our basic server and engine completed by the beginning of December, with map generation complete by December 1st, and then taking the time from the 2nd until the 6th to design and create enemies, items, combat, and any other complexities we wanted to add. However, our actual timeline ended up with our server being complete December 2nd, due to other school and work projects, and map generation was finally completed on December 5th, meaning all of the 6th was spent designing and implementing the basic game functionality, such as combat, enemies, and items.

**State Change Diagram**



**Assets**

**Player:** Born when a **Player** loads the webpage from the server. Has invincibility while entering in the **Player’s** name. Moves via WASD or arrow keys. Attacks either other **Players** or **Enemies** by pressing the directional movement key towards that unit once adjacent.Dies once hitpoints reach 0 or below, but respawns at the beginning of the **Map**, back at level 1, post mortem. **Players** are visually represented by an @ symbol with a random color chosen per each **Player**. **Players** may also type in the **Chatbox** and communicate with other **Players**, as well as read information regarding **Enemy** and **Player** stats.

**Enemies**: Born when the **Map** is generated. AI functionality ensures that they are always headed towards the nearest **Player** using a breadth-first-search on the array to find the nearest **Player** and route towards them. Attacks **Player** characters upon each collision, doing calculated, random amounts of damage to the **Player** character by randomly generating a number between 1 and 20 and comparing it to the **Player’s** armor class, if the attack roll exceeds the armor class, each enemy has a pre-defined amount of damage they deal. There are five classes of **Enemy**, tier 1, 2, 3, 4 ,5, bosses, and raid bosses. Each level has a boss, and a 15% chance to spawn a raid boss. When the **Map** is created, there is a 60% chance that each spawned **Enemy** will be from tier 1, an 18% chance it will be a tier 2, 12% chance for tier 3, 7% for tier 4, and 3% for tier 5. This helps ensure that there aren’t an excessive amount of **Enemies** that are too difficult for a level 1 **Player** to beat, though the generation throughout the **Map** is random, and so may create difficulty for single **Players**. **Enemies** die once their hitpoints are reduced to or below 0, and are removed from the **Map**, granting experience points to the **Player** that dealt the final blow. Each **Enemy** has a different symbol and color, with bosses having capital letters and basic **Enemies** being lowercase letters or symbols. Raid bosses also randomly will choose a color to depict different types of high level bosses making each raid more unique.

**Items:** Born when the **Map** is generated. Instantly consumed when a **Player** character overlaps an **Item** space. There are 3 kinds of **Items** on the **Map**, health potions which restore a **Player’s** hitpoints to max value and are the most common throughout the **Map**, attack potions which slightly increase the amount of damage a **Player** deals during combat and are rare, and defense potions which dramatically increases the armor class of a **Player** and are very rare. All potions are permanent until death, and do not respawn on the **Map**.

**Map:** Generated upon server start. Uses a Binary Space Partitioning algorithm to create randomness in generation so that it is unlikely 2 **Maps** will ever be identical. Doors between rooms are also created randomly. Walls are defined by hashtags and are game entities, while floor is just filler and not actually an entity.

**Chatbox:** While not necessarily a sprite or **Map** object, this key piece allows for each **Player** to see exactly what combat actions are taking place during the game, as well as showing remaining hitpoints for **Players** and **Enemies** that are fighting. This is also configured to allow **Players** to communicate so that they can coordinate to fight bosses or clear the dungeon more efficiently.