**Grunk’s Arena**

**Aaron Camm, Patrick Cardwell, Jeff Huggins, and Steve Moskal**

# 

# 

**Introduction:**

For this capstone we are looking to create a fast-paced local multiplayer FPS Arena shooter. Similar to the Unreal Tournament series but also drawing inspiration from a newer game called *Duck Game*.

**Story:** A group of astronauts get abducted by the infamous space Emperor Grunk. Grunk pits his prisoners against each other in combat….TO THE DEATH!!! His subjects refer to this as Grunk’s Arena.

**Initial Goals:** By the end of the semester we had hoped to have a fast-paced FPS that has support for up to 4 players, each with their own Xbox 360 controllers. Along with that we wanted to have AI bots along with the players, procedurally generated maps, and a visually appealing ingame UI. Allocating each of these fields among our team members, we managed to complete all of the features that we initially planned to implement.

**Rules**

**Win State:**

To win Grunk’s Arena players must be the last player standing at the end of the most rounds, where the number of rounds is determined in the multiplayer lobby.

**Weapons:**

To dispatch opponents players must find and use various weapons spread out randomly across the map.

**Movement:**

Players are able to look and move in any direction as long as they are not obstructed by walls, the ground, and other game objects. Players are also able to jump. While in the air they may jump again to go higher. Or their second jump can be used to perform a wall jump if they are close enough to the wall. Also, players will not suffer damage from anything except for weapons.

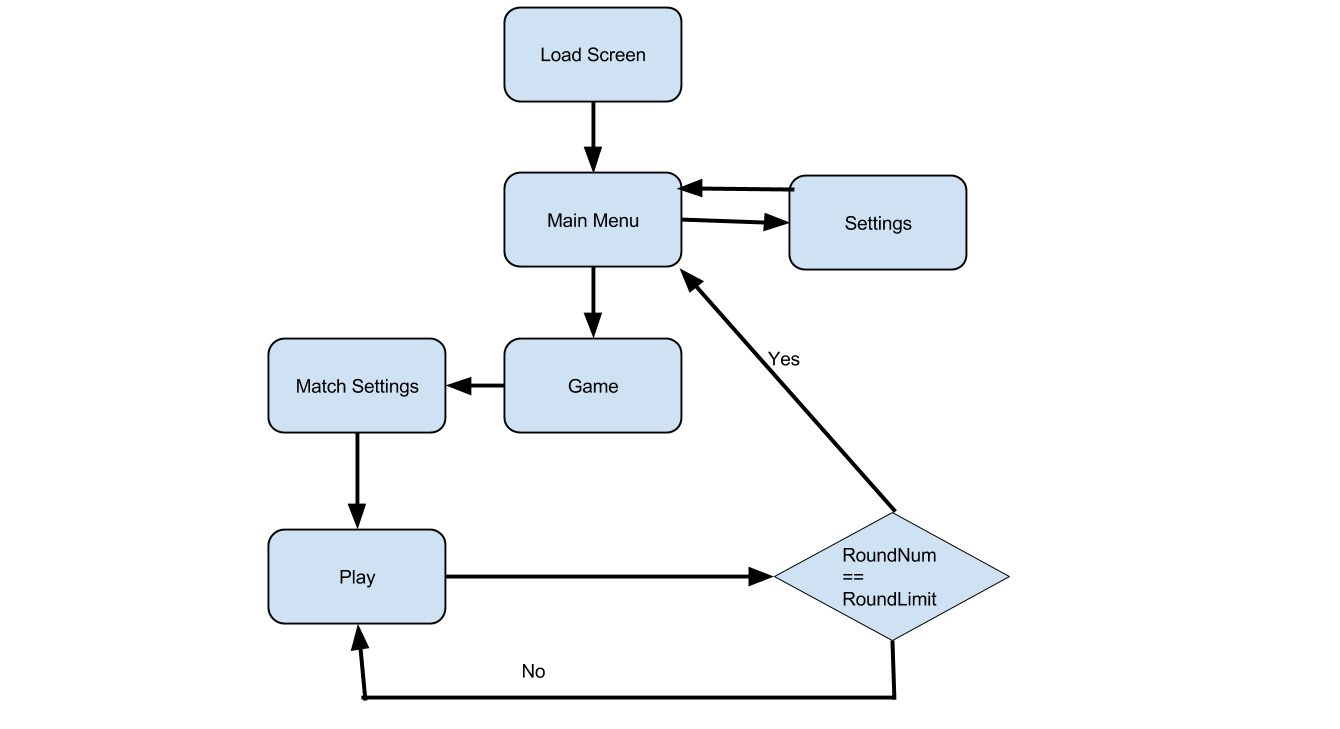
**Health:**

Grunk’s Arena is a fast and dangerous place. All weapons will result in the instant eradication of a player. Once a player is hit and destroyed they will be out for that round and will spawn again at the start of the next.

**Game Flow**

**Overall Game State Logic:**

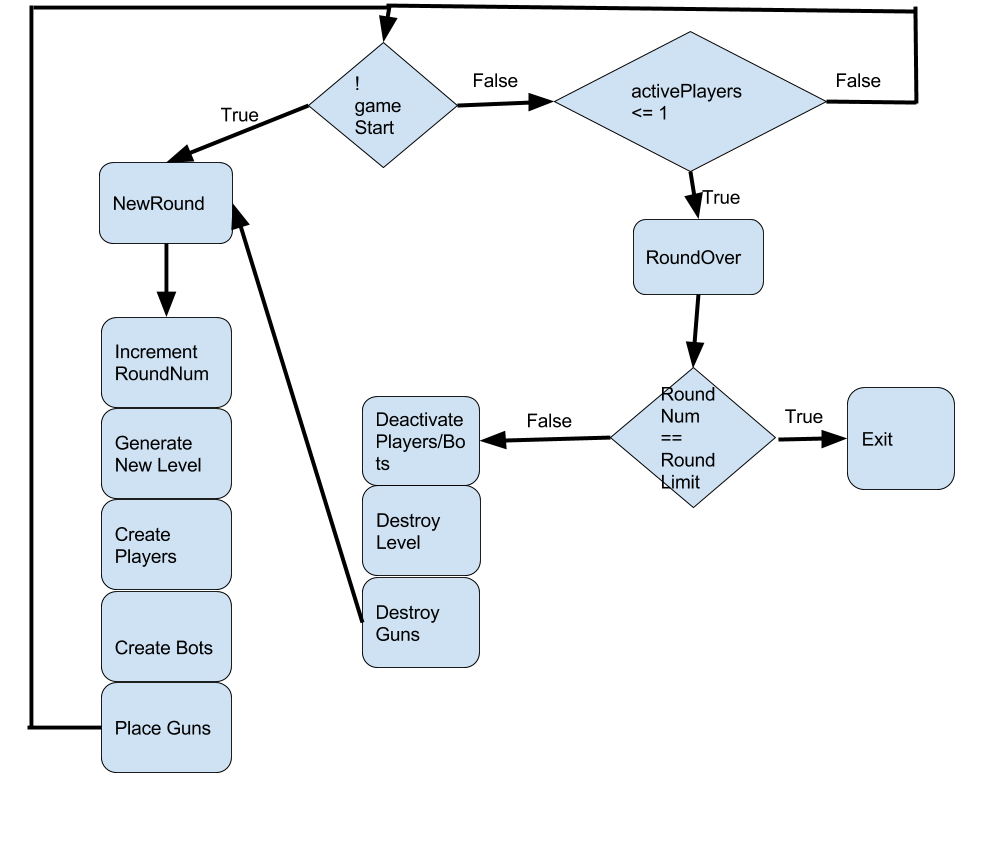
This diagram shows how the game flows from state to state.



So the main menu consists of 4 options: play game, options, credits, and exit game. Settings allow the user to change the music volume and the sound effects volume. Credits shows who worked on creating the game. Exit game just quits the game. And play game brings up the pre game lobby. There the user can change variables to get a different experience. Such as map width, building density, height scale, number of bots and players. Once the user presses start the play state will start, and it will loop over the game play state until the round limit is reached. Then it will go back to the main menu after the game win state.

**Play Game State Logic:**

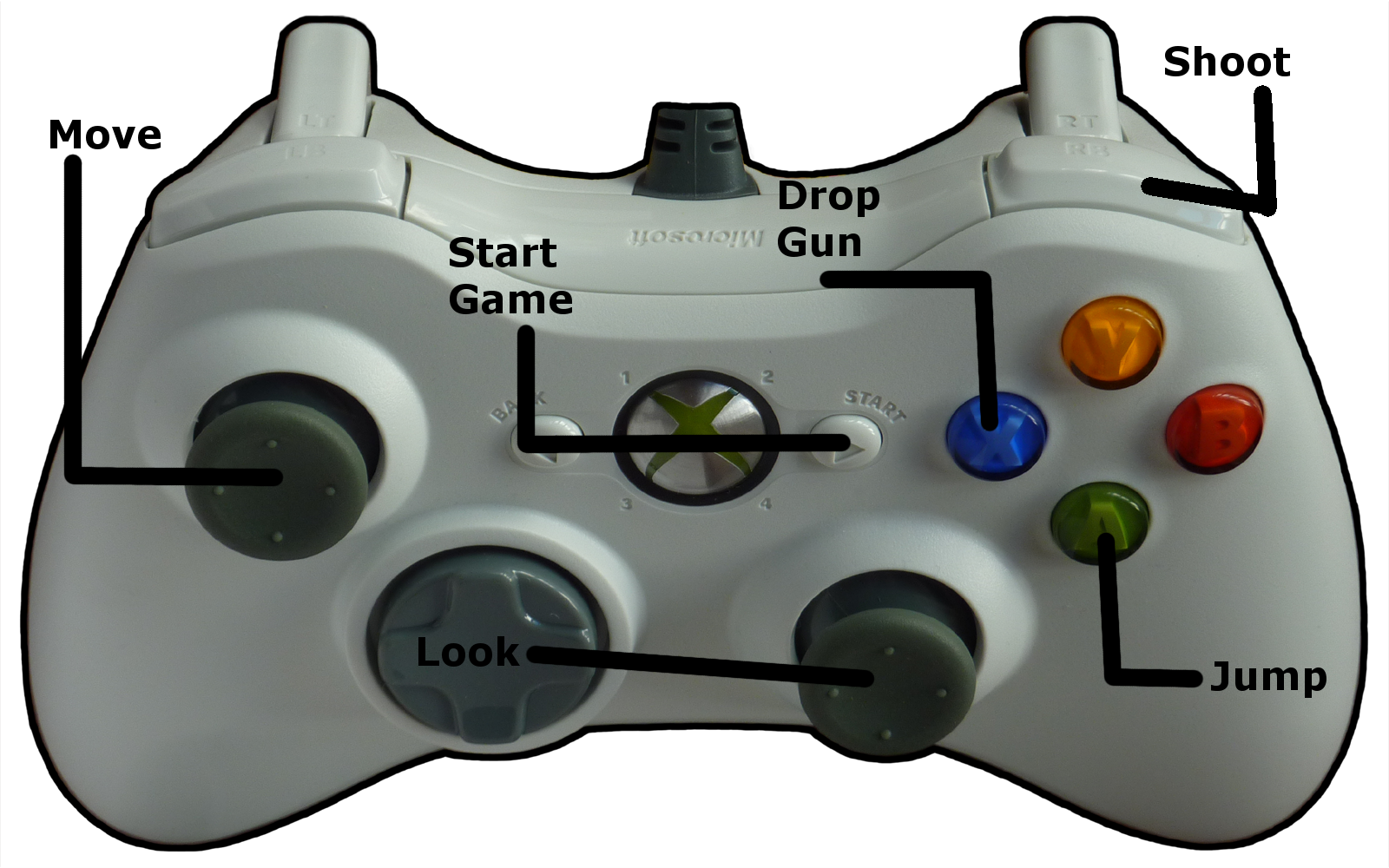
Below is the diagram showing how the PlayerManager script handles starting a new game and starting a new round.



So every update in the PlayerManager script it checks if !gameStart (boolean for if it is the first round or not). If that is true then it will call the NewRound method. Which increments the roundNum, generates the level with the TerrainGeneration script, creates the players and bots, then places the guns. The game then starts. On the other side if !gameStart is false it then checks how many active players there are in the scene. Once there is only one player left the script will then call RoundOver. It first checks if the round limit has been reached, if it has then it will just exit to the main menu. If not then it will start by deactivating all players and bots left, destroy the level, destroy the guns then call NewRound.

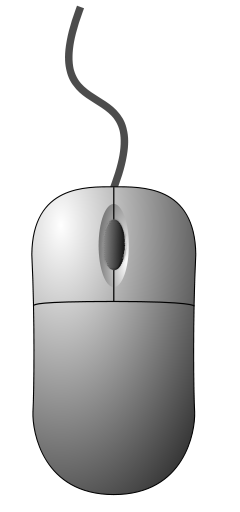
**Input Devices**

**Xbox 360 Controls:**



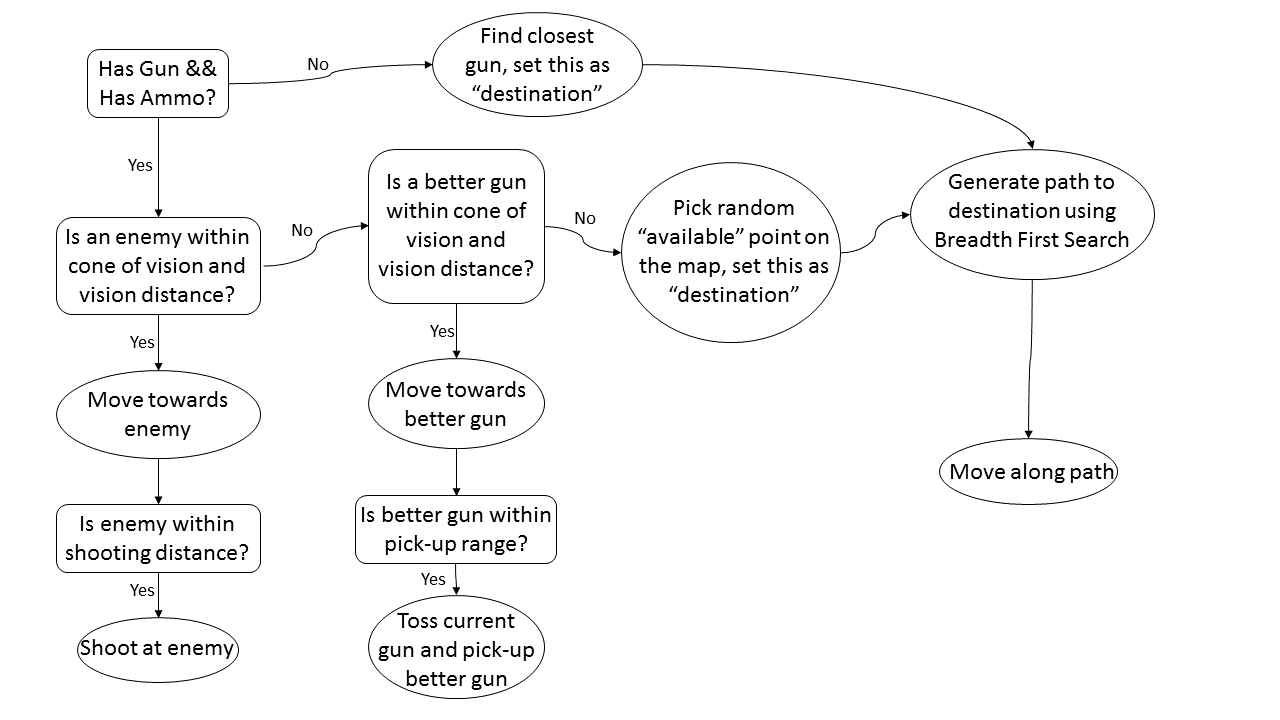
The majority of the xbox controls are for when the game is in the game play state like jumping shooting, moving etc. But when users are in the pre game lobby they use the Xbox 360 controller to press start to join the game.

**Mouse:**

The mouse is used just for navigating and adjusting menu settings. Users are unable to use mice during the gameplay state. 

**AI**

The diagram below details the different checks the bots make every update:



When the terrain is first generated it is covered in a 2D node array. These nodes are listed as “available” if they are not occupied by a wall; this leaves the area outside of buildings, the area inside of buildings, and doorways. These nodes are the ones taken into consideration when searching for a random destination as well as when generating a path to that destination. It should also be noted that if a bot is following the path generated by the breadth-first search, it will not search for another destination or generate a new path until that destination is reached, or until an enemy or better gun enters its cone of vision.

The process of finding a better gun is done by assigning rank values to each gun. The Bit Gun has a rank of 1, the AK47, UMP and M4 have ranks of 2, and the Bazooka and Sonic Guitar have ranks of 3. If a gun in a bot’s cone of vision has a higher rank than its current gun, then that is determined to be a better gun and the bot will move towards it.

As previously described, the pathfinding algorithm used by bots to a random point is breadth-first search. Breadth-first search is achieved by first constructing a BFS tree. This starts by expanding the bot’s current node which adds its adjacent available nodes to the tree. Then subsequent nodes that have been added to the tree will be expanded until the destination node is in the tree and attempts to expand. After the tree has been created, the best path from the current node to the destination node is created using the distance attribute of each node. A node’s distance attribute is set upon its expansion and is equal to how many nodes away it is from the current node.

Another element that was added to the bots was a random degree of inaccuracy when shooting a gun. Since a bot would always be aiming directly at the enemy when they fired, the only inaccuracy they would have would come from the gun itself. Since not even the best player can always have their reticle on their target when they shoot, when a bot gets created they are given a random degree of inaccuracy that adds on to the standard gun inaccuracies.

**Level Generation**

Every map in Grunk’s Arena is randomly generated. This ensures that each time a player enters a map, the gameplay will be a unique experience, and the player won’t be able to rely on knowledge from past maps. The map generation parameters can be modified by players in the pre-game lobby to create even more interesting encounters. This includes options for modifying the base terrain heightmap, the placement and dimensions of buildings, and player/item spawning properties.

**Terrain Parameters:**

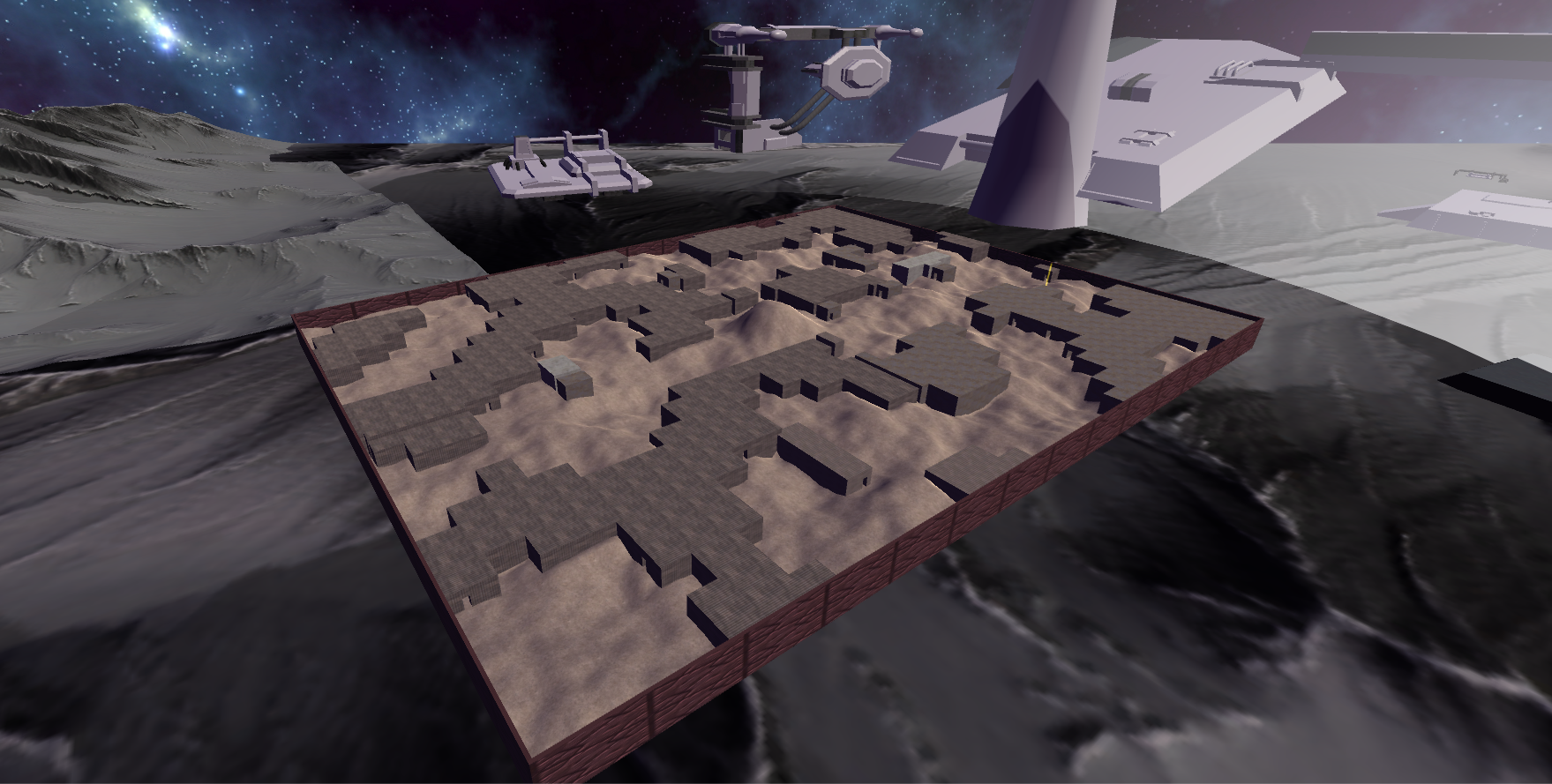
The bumpiness of the the terrain can be modified, as well as the overall dimensions of the map. This includes the length and width.

**Building Parameters:**

The building distribution can be modified to decrease or increase the probability that buildings are placed.

**Spawning Parameters:**

The number of AI players can be adjusted. They can also be removed altogether for competitive player-only action.

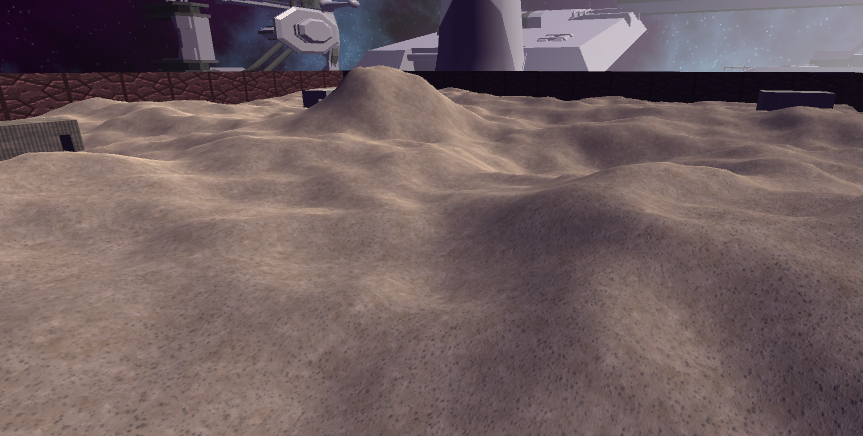


**A standard level in Grunk’s Arena.**

After starting a game from the pre-game lobby, a map is generated based on the parameters. This happens in three primary stages: setting the terrain heightmap, generating the buildings, and placing the spawns in the playable area.

**Creating the Heightmap:**

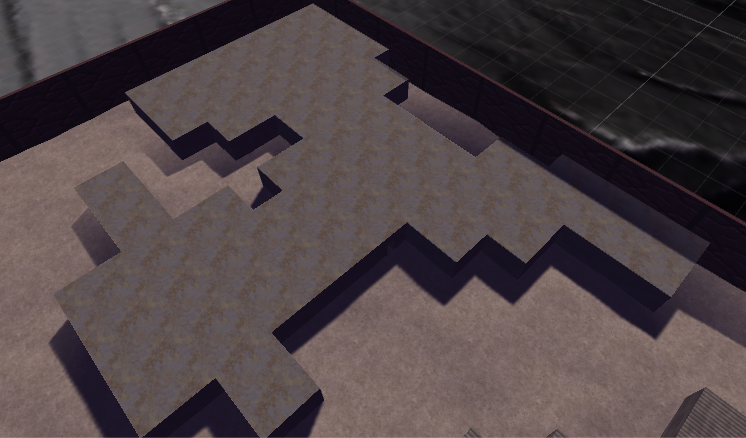
The base of the rectangular map is a 2-dimensional grid of points. These points are connected with triangles to create a walkable ground. The height of each point is determined using the result of a Perlin noise function. Using reasonable input values combined with the modifiable values in the pre-game lobby, different environments can be created. The image below shows a possible result using the default terrain height value.



**Example of terrain created from a heightmap.**

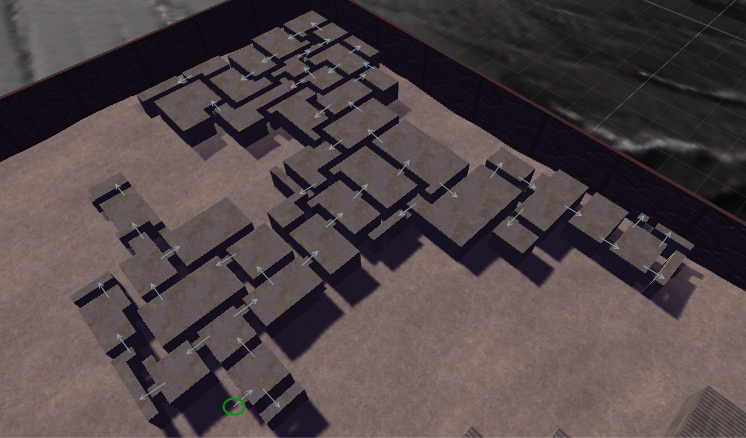
**Creating the Buildings:**

During the heightmap generation, each grid point is marked whether or not it will be a building spot. The decision is made with a separate noise function, testing the result against a threshold. Connected building grid points will end up being used to make a whole building. Next, the area of each building is traced to find where the walls will be. Below is an example of a mesh created using vertices obtained from a traced building.



**Example of the exterior of a building.**

Afterwards, the interior of the structure is populated with rooms. A random wall on the outside of the building is selected to be a door, and using a depth-limited search, a small portion of the inside area is made into a room. Using a recursive approach, the same process is applied to walls on the newly created room, where doors are created on random walls where there is to be another room on the other side. The following image shows an example of the direction this algorithm could go.



**Starting at the green circle, the search results in these separate rooms.**

After the algorithm is complete, the rooms are often far too intricate to traverse due to the structure of how they’re connected, so extra doors are added around the building to make it more open. Once this is complete, randomly placed crates are included in some of the rooms to add more detail.

**Creating the Spawnpoints:**

After the map design has finished, we perform a search from the center of the map outward to find the playable area. This step is necessary because it’s possible for some areas to be blocked off from the rest of the map, such as inside the building’s walls or in an isolated corner. The points that mark the playable area are then marked as walkable nodes for the AI players as well as randomly selected for spawn points. One list is created for player spawn points and another for weapons.

By this point, the level has been fully generated and the player/item spawning parameters have been set. The game begins immediately after this.

.

**UI**

In Grunk’s Arena, every player has their own UI overlay in their player's camera. This show details about the gameplay that the player will need to know. The UI overlay consist of 3 parts: the Radar, the Aiming Reticle, and Ammo details for each player.

**Radar:**

Radar shows the position of nearby other players and bots within a certain distance, internally set to a distance of 50. Radar works by first gets the location of all players with a certain distance, then utilizes linear algebra to map the distance and angle of the player or bot to a radar dot that will appear on the radar image.

**Ammo Text:**

Shows the number of current ammo and max ammo of your currently equipped weapon. If the current ammo ever reaches zero, then the player will know it must either reload or get a different weapon. What is the max ammo may differ for each weapon.

**Aiming Reticle:**

A simple indicator in the middle of the screen that will show roughly as to where any of your weapons will fire towards.

**Weapons**



**AK47**: Automatic, long-range assault rifle. Accuracy decreases drastically after firing more that one shot.



**Bazooka:** Launches a projectile that explodes on impact. The shrapnel that comes from the explosion will destroy players, including the one who launches it.



**Sonic Guitar:** This is not a normal guitar. It has been modified to send out strong sonic waves that will destroy those in its path. Think of it as a shotgun of sorts.



**Bit Gun:** The starting weapon of all combatants in Grunk’s Arena. Not particularly powerful but enough to take down enemies when close up.



**M4:** Standard military assault rifle. A very long-range and incredibly accurate rifle that fires 3-shot bursts. A must have for those who like to dispatch enemies from a distance.



**UMP:** Submachine with a short range but a high rate of fire. Very nice to have when going through buildings.