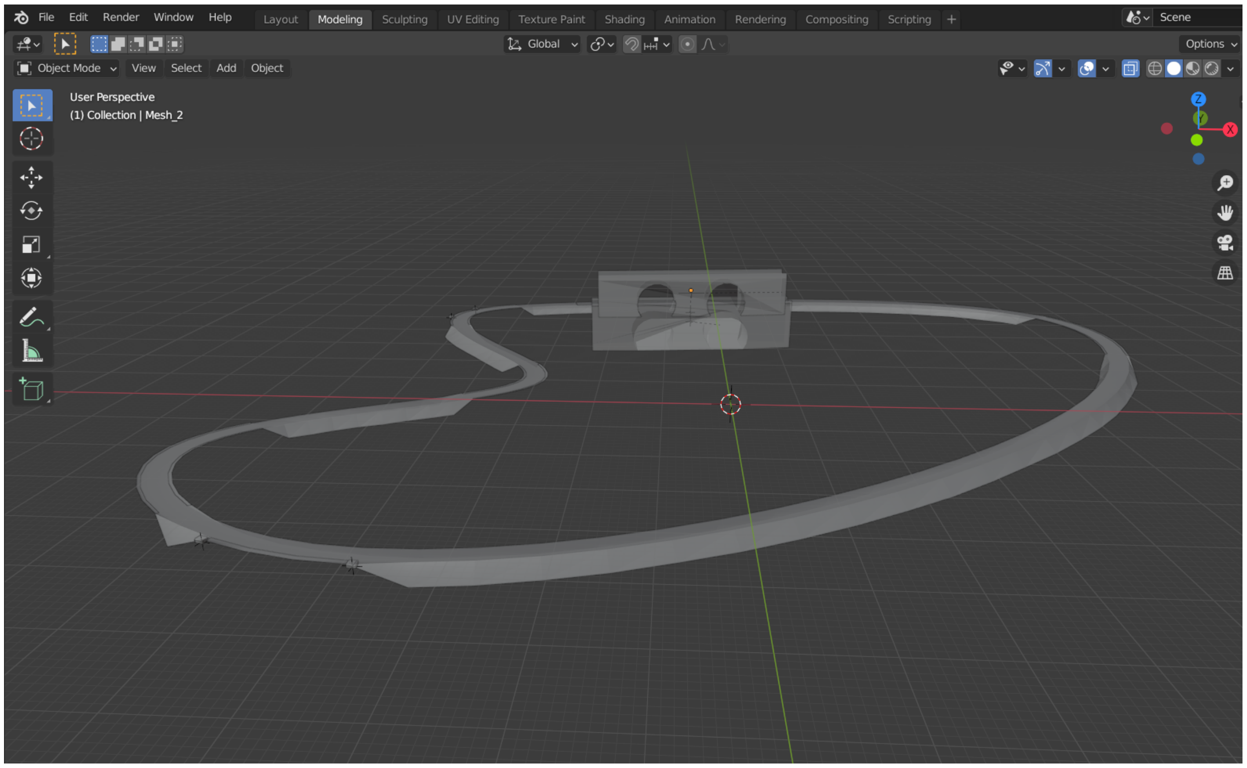
Group 1 Green Shells Technical Details

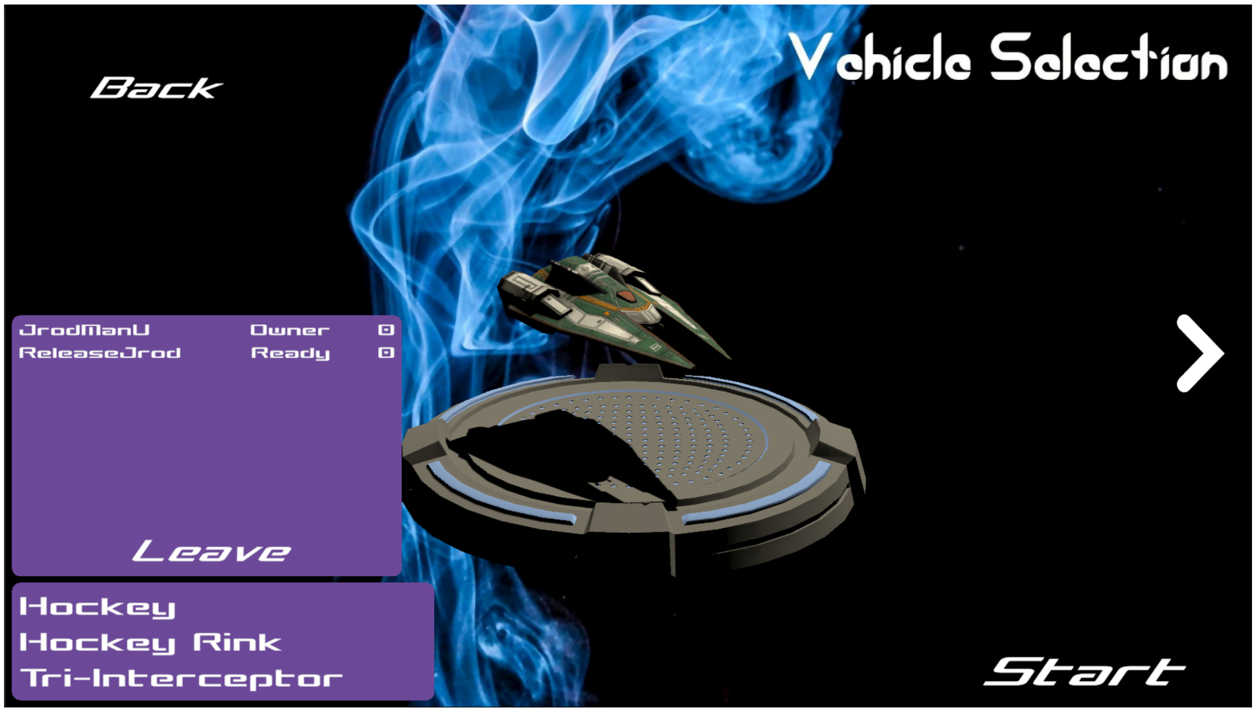
Track Creation

We used both Blender and Unity to create our tracks. Box colliders are used as walls to guide AI and players. In the Radioactive track the barbed wire walls contribute to the theme too. In the Mountain track we modeled invisible colliders in Blender and imported as fbx files back into Unity to ensure the players stayed on track. In the Mars Crater arena, Unity 3D objects with the mesh filters turned off keep the players inside of the main crater.



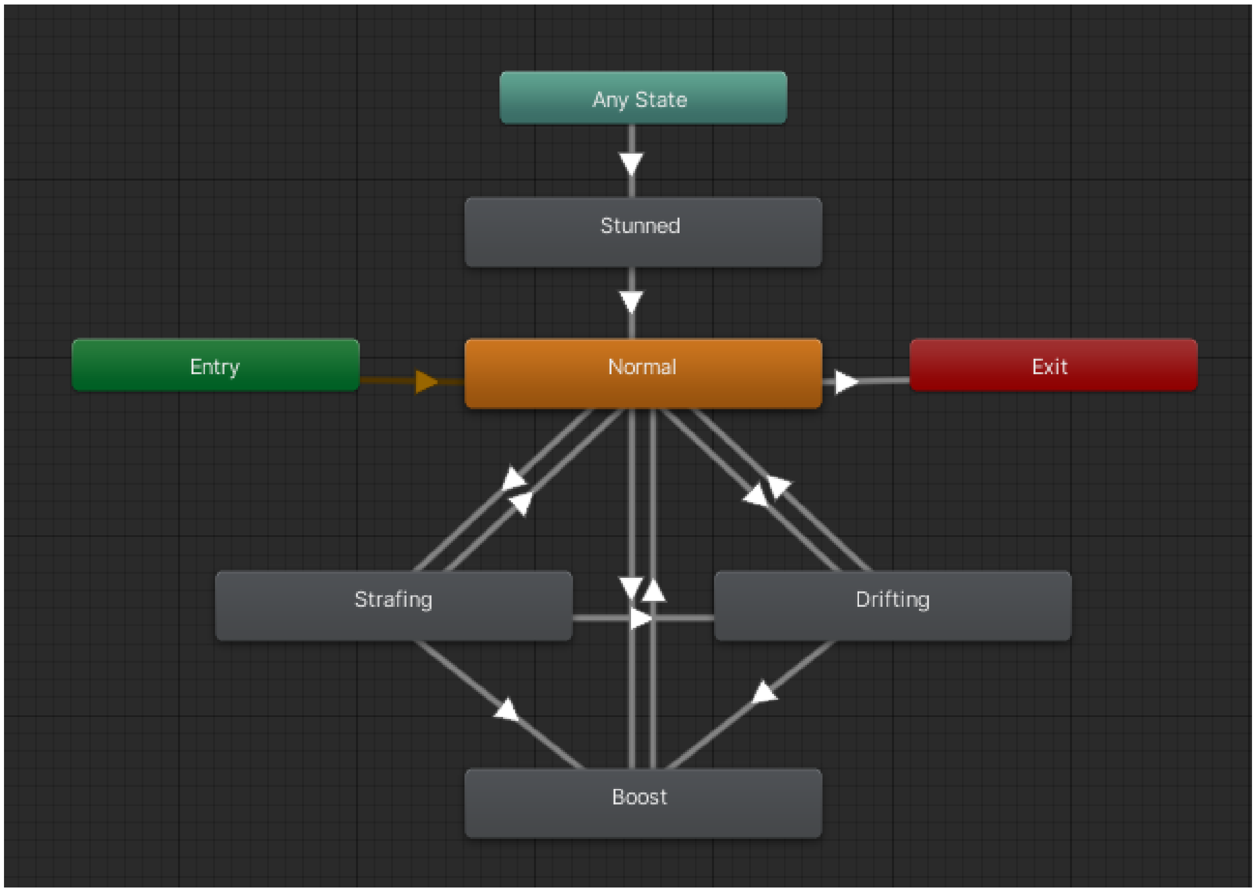
Starting a Multiplayer Race

A lot goes into starting a multiplayer race. First, the lobby has a ready check that all players have to press. This message is sent buffered by Photon so that even new players to the lobby will receive it. Once the master client receives that all players are ready it broadcasts to all that the race has started, and closes off the room to new people. All players will already have a track name from the lobby. They load this using a prefab pool. The track name is the key. After this each player tells the master client to spawn them in with the name of the vehicle as the data. The master client uses Photon's instantiate function that gets called on all clients and then transfers ownership of the vehicle instantiated to the client. Once the master client spawns all the players it will call a function that tells everyone to start the countdown. Additionally, the master client spawns in any AI in the same way. If the master client leaves the new master client will take ownership of the AI and track gameobjects, so the race continues.



Vehicle Movement

The vehicles move primarily based on a Unity Animator state machine. There are 5 states: normal, boosting, strafing, boosting, and stunned. Each state handles the inputs that the player can use, often inheriting the behavior from the base state. In this way, all the stunned state has to do is override the acceleration, turning, and using items input to do nothing. There are also a number of constant behaviors, such as falling, health, and aligning to the surface, which are outside of the state behaviors.

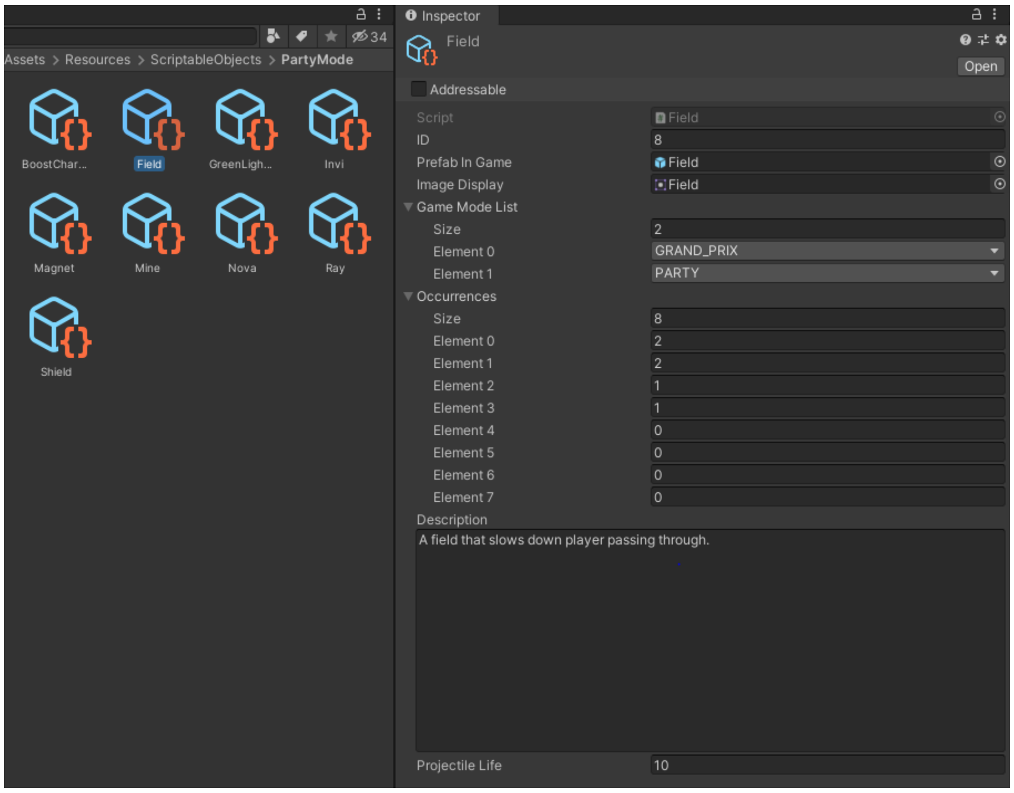


Multiplayer Synchronization

For the vehicle's own changes, such as location, effects and discharging pickup, each vehicle calls the method locally, and then synchronizes it on the screens of other players by photon view or remote procedure call. For changes in the environment and room, such as whether the pickup box in the map is picked up and the location of the energy ball, those changes are sent to all the players in the room by a single manager owned by the master of the room.

Pickups

All the pickups are stored and managed as scriptable objects. Each pickup has its own data and discharge method. When a player hits a pickup box on the map, the pickup manager in the track will be notified and syncs the games of other players. Players will get pickups based on their ranking with probabilities of occurrence for each pickup recorded in scriptable object data.



Addressable

Script	Field
ID	8
Prefab In Game	Field
Image Display	Field
Game Mode List	
Size	2
Element 0	GRAND_PRIX
Element 1	PARTY
Occurrences	
Size	8
Element 0	2
Element 1	2
Element 2	1
Element 3	1
Element 4	0
Element 5	0
Element 6	0
Element 7	0

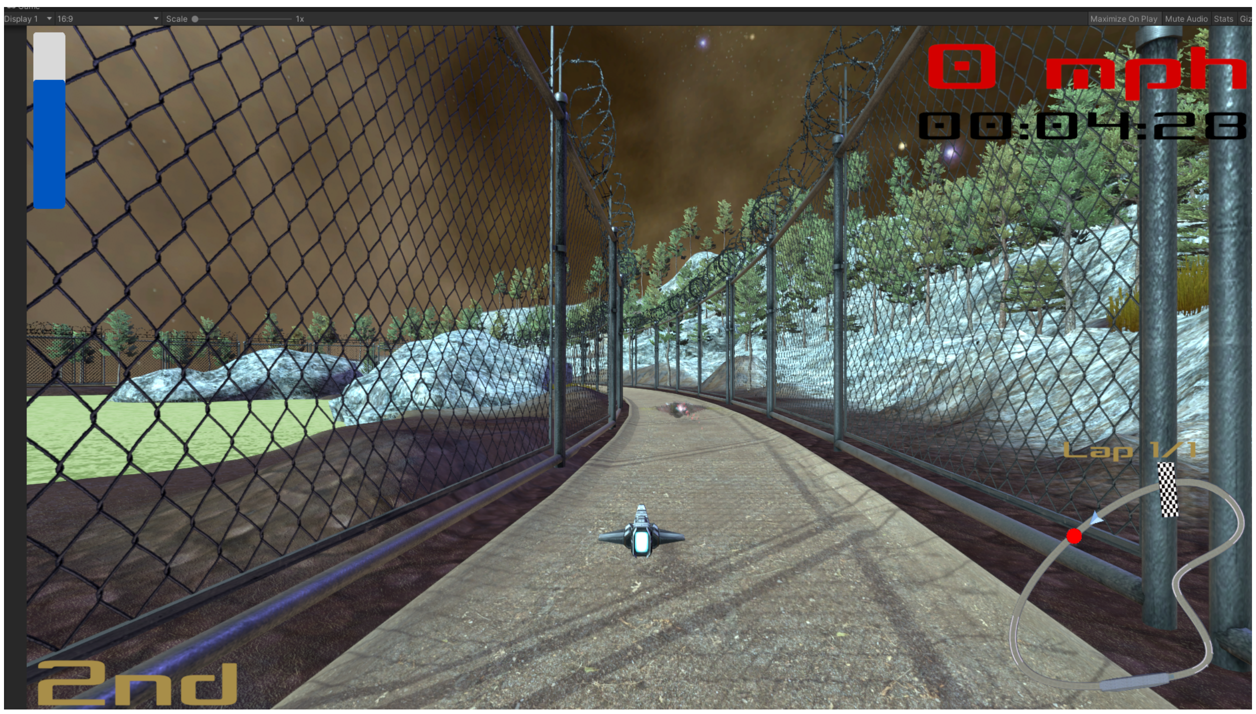
Description

A field that slows down player passing through.

Projectile Life	10

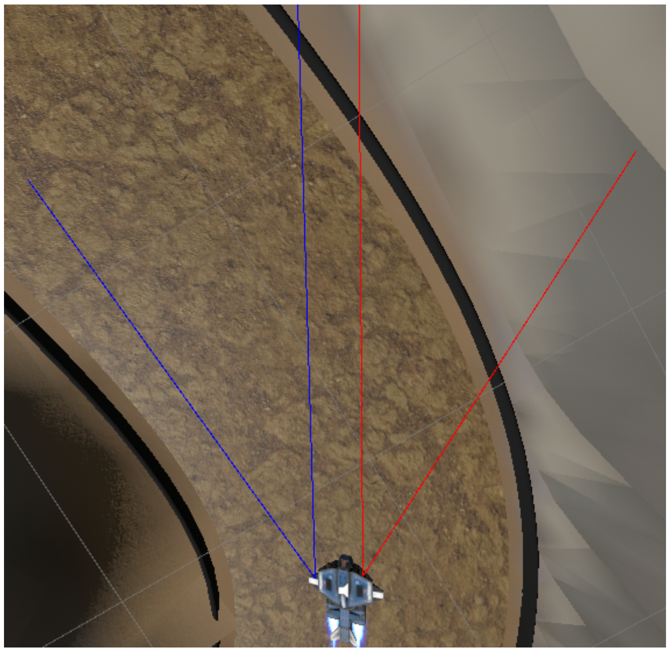
Saving and Loading Player Ghosts

In practice mode, a player races against their own times. As such, having “ghost” players the player can race against helps visually communicate this and make the game mode more engaging. For this, a player’s performance needs to be saved. Since logging a player’s inputs is nondeterministic, we instead log player position and rotation every fixed update. This would then be saved to a .json file containing that logged data, as well as the vehicle used, and the time achieved if it was faster than one of the previous times. For playback, up to 8 ghost records are loaded, then the positions and rotations of those ghost racers are set manually at each time step according to the saved data. Because this is set manually, we can remove colliders on each ghost to prevent them from blocking the player.



AI Vehicle Navigation

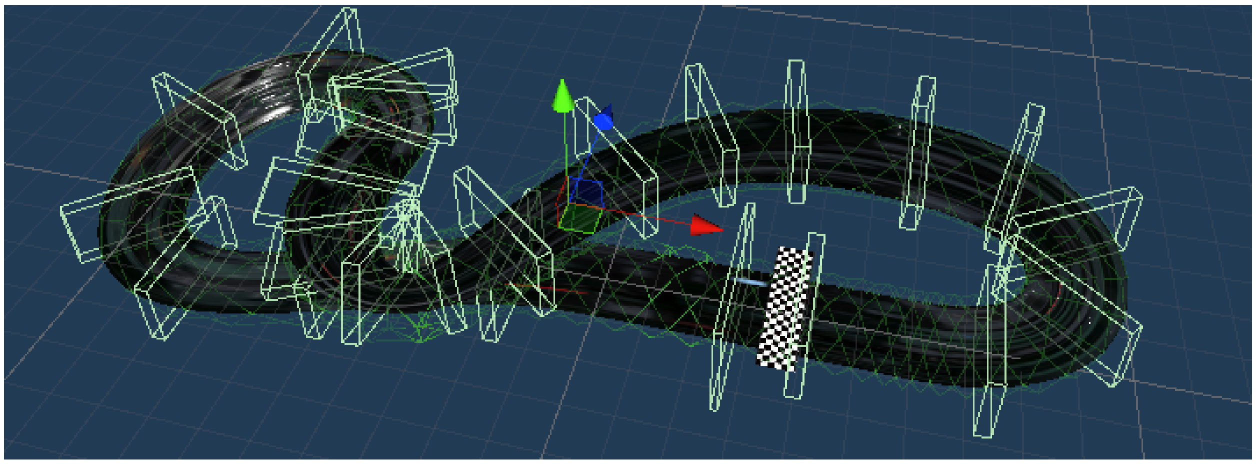
In Grand Prix players can race against AI. Each track has invisible checkpoints. The AI shoot raycasts to find these checkpoints. However, there could be obstacles between the AI and the nearest checkpoint. To handle this we had the AI shoot multiple raycasts from the front of the vehicle, two straight ahead offset to the left and right. Additionally we shot a raycast angled 30 degrees away from the vehicle on the left as well as the right. If a raycast on the left hits something, there’s an obstacle on the left so the AI should turn right. Similarly if the raycast on the right hits something, there’s an obstacle on the right so the AI should turn left. If the left and right raycasts hit something, there’s an obstacle straight ahead so the AI should simply turn in the direction of the next checkpoint. To handle obstacles of different heights, we shoot raycasts in the same directions, but from lower and higher starting positions as well. To become more precise, we added more raycasts on the left and right, same concept but this allowed us to detect narrower obstacles and better navigate when multiple obstacles were present.



Lap Tracker

The track utilizes checkpoints to track the progress and rankings of each player. Each track has multiple numbered checkpoints throughout the map and each player stores the last checkpoint hit. When a player hits a checkpoint, the checkpoint then checks the checkpoint number stored in the player, makes sure that the player did not skip any checkpoint, then updates that player's checkpoint number. Similar logic applies when the player goes backward.

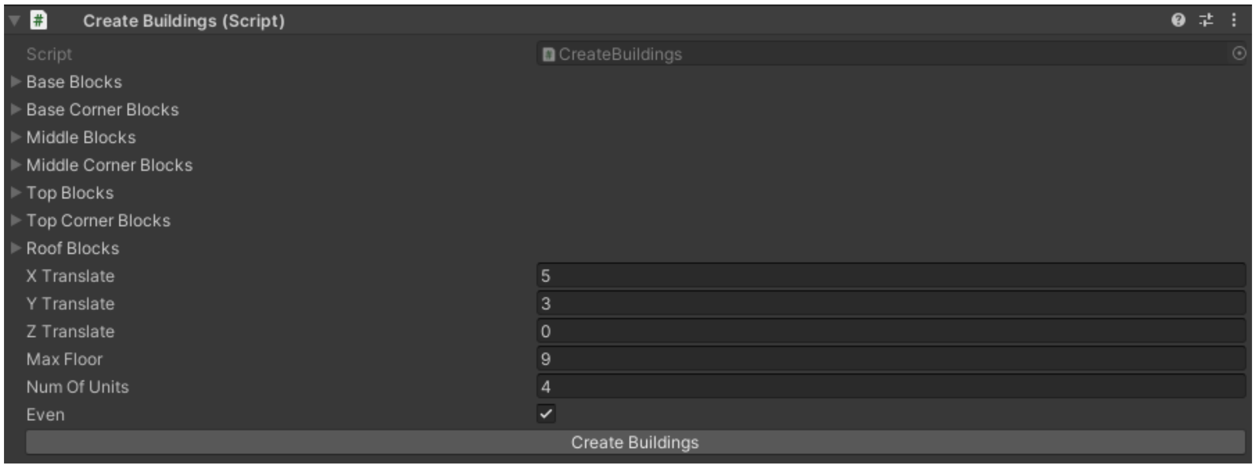
There is a singleton class named LapListener that keeps track of the checkpoint number for all players. Based on the current checkpoint, it calculates which lap each racer is in. It also uses a weighted sum to calculate the ranks of each player using the lap number, checkpoint number and distance to the next checkpoint.



Procedural Generation

Building Generation

Buildings in this game are generated using building blocks. When all the blocks and their corresponding offsets are provided, the script automatically generates a building of a given size.



Create Buildings (Script)

Script	CreateBuildings
Base Blocks	
Base Corner Blocks	
Middle Blocks	
Middle Corner Blocks	
Top Blocks	
Top Corner Blocks	
Roof Blocks	
X Translate	5
Y Translate	3
Z Translate	0
Max Floor	9
Num Of Units	4
Even	✓

City Generation

The city is made out of tiles that are bordered by roads in all four directions. Each tile has some designated building spawn places. When the city is generated, the script randomly stitches tiles together while attaching random buildings on top of the tile. So each time the destruction mode is opened, different tiles and buildings are stitched to create the map.