Nicolas LaCognata
CAP 4053

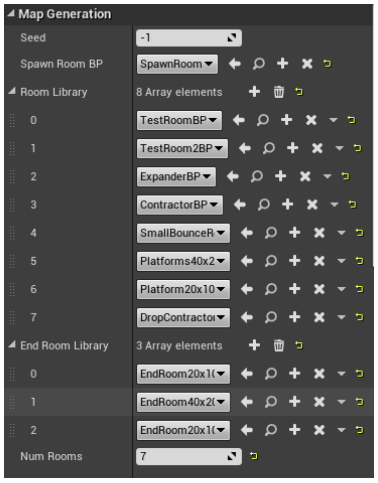
Summary of Contribution

Level Generation

Overview:

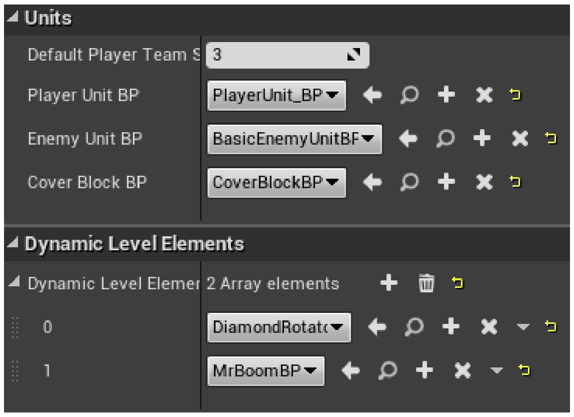
The level generator for Master Blaster is one of the larger systems of our game. It is used to spawn and build another Actor, the Combat Map. The combat map is basically a thin wrapper for the list of rooms the level generator has selected to make a level. The level generator delegates the spawning and initialization of each room to the combat map.

The level generator was made to be extremely flexible, and it allows for rapid iterations. A level designer can add, modify, and remove rooms from the level generation process without changing the generator's code.



Map Generation

Seed	-1
Spawn Room BP	SpawnRoom
Room Library	8 Array elements
0	TestRoomBP
1	TestRoom2BP
2	ExpanderBP
3	ContractorBP
4	SmallBounceR
5	Platforms40x2
6	Platform20x10
7	DropContractor
End Room Library	3 Array elements
0	EndRoom20x1(
1	EndRoom40x2(
2	EndRoom20x1(
Num Rooms	7



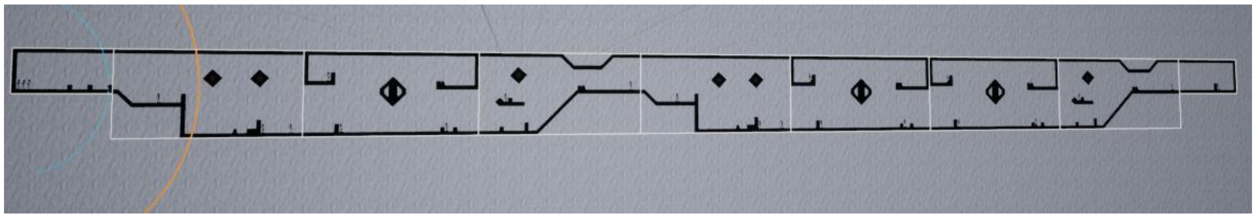
Units

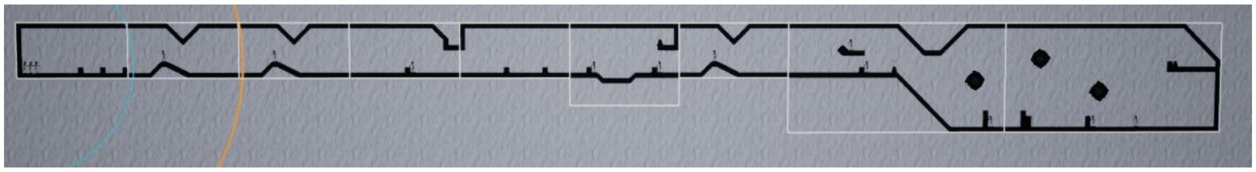
Default Player Team S	3
Player Unit BP	PlayerUnit_BP
Enemy Unit BP	BasicEnemyUnitBF
Cover Block BP	CoverBlockBP

Dynamic Level Elements

Dynamic Level Elemer	2 Array elements
0	DiamondRotat
1	MrBoomBP

Output:





Nicolas LaCognata
CAP 4053

Algorithm:

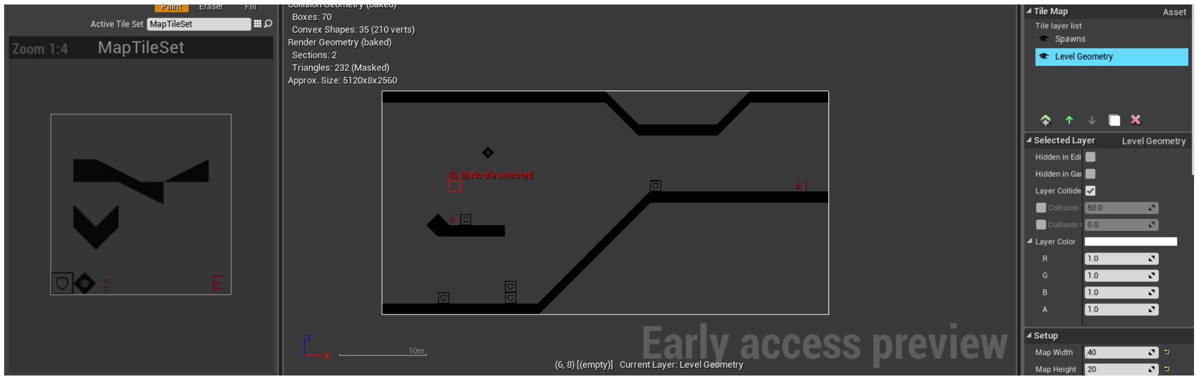
The logic of the level generator is very simple. It takes a list of rooms, starting with a static spawn room. It adds the requested number of rooms to the map by following the same process for each room.

1. Look at the size of the exit of the previous room
2. Get a list of all the rooms in the room library that connects to that room
    a. This comparison is a comparison of the base of the exit (from the top of the map) and the height of the exit. If both conditions match up, the room can be added to the list.
3. Pick an element from that list and add it to the map.

After the map is generated, the level generator instructs the combat map to begin spawning in Dynamic Level Elements, which will be discussed later.

Rooms:

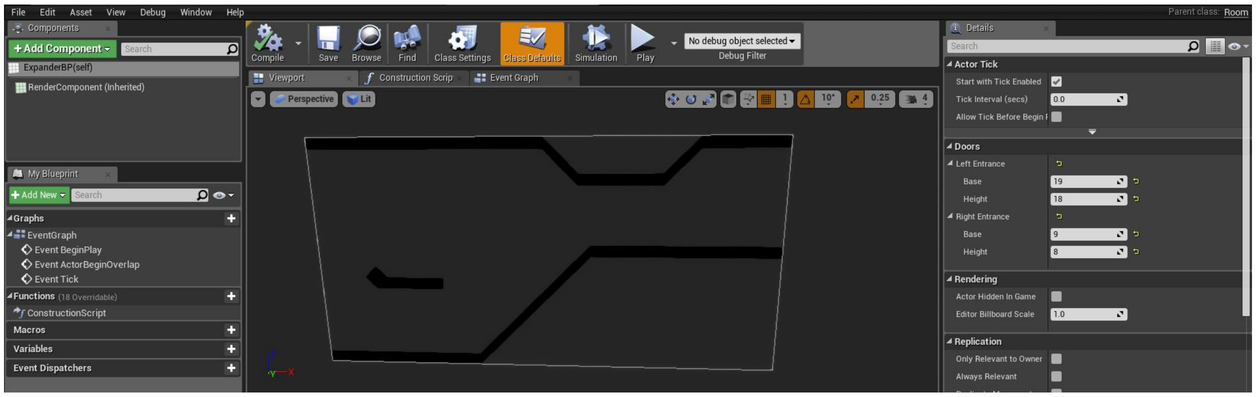
The rooms that are fed into the level generation are tile maps. These maps are 2x2 arrays of tiles specified by a tile set. The tile set was designed in Photoshop, and rigged up in Unreal with collision boundaries. Rooms are desinged using Unreal’s tile map editior. Unreal’s tile map implementation is currently in early access, so finding doucmentation on the matter proved to be difficult at times.



```
Active Tile Set MapTileSet
Collision Geometry (baked)
  Boxes: 70
  Convex Shapes: 35 (210 verts)
Render Geometry (baked)
  Sections: 2
  Triangles: 232 (Masked)
Approx. Size: 5120x8x2560
(6, 8) [(empty)]  Current Layer: Level Geometry

Selected Layer    Level Geometry
Hidden in Edi     ☐
Hidden in Gar     ☐
Layer Collide     ✓
Collision         50.0
Collision         0.0
Layer Color
  R               1.0
  G               1.0
  B               1.0
  A               1.0
Setup
Map Width         40
Map Height        20
```

The rooms the level generator receives are the same tile maps wrapped in Unreal Paper Tile Map Actor Blueprints. Wrapping the raw tile maps in these blueprints allow the maps to be placed in the world like any other actor. The blueprint includes the tile map it represents, and the size of the map’s entrance and exit.



Parent class: Room

Actor Tick
Start with Tick Enabled ✓
Tick Interval (secs) 0.0
Allow Tick Before Begin I

Doors
Left Entrance
Base 19
Height 18
Right Entrance
Base 9
Height 8

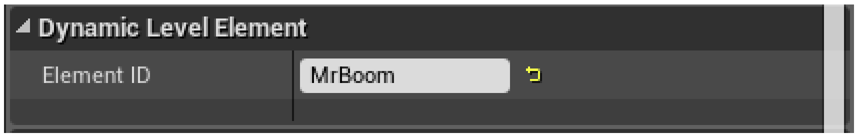
Rendering
Actor Hidden In Game
Editor Billboard Scale 1.0

Replication
Only Relevant to Owner
Always Relevant

Dynamic Level Elements (DLE):

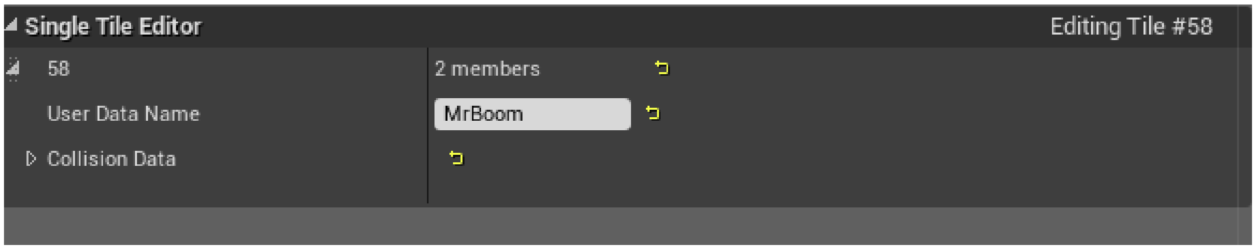
The most important feature of the level generator, aside from generating levels, is handling the inclusion of gameplay actors into the levels. This is achieved through the Dynamic Level Elements system.

A DLE in our game is any actor that implements the Dynamic Level Element Interface. The interface includes a function to retrieve the actor's element ID, and a class member to store that ID.



Dynamic Level Element
Element ID: MrBoom

This ID is matched against tiles in the tile set used to make rooms. If a tile on the spawn layer of a room has a User Data Name that matches an Element ID of an actor in the level generator's list of DLEs, the appropriate actor is spawned at that tile's position in world space.



Single Tile Editor — Editing Tile #58
58: 2 members
User Data Name: MrBoom
Collision Data

Elements can be added and removed from this system without opening visual studio. The only two steps are adding the spawn tile in the tile map, and matching the spawn tile's ID when creating the actor.

Nicolas LaCognata
CAP 4053

Issues:

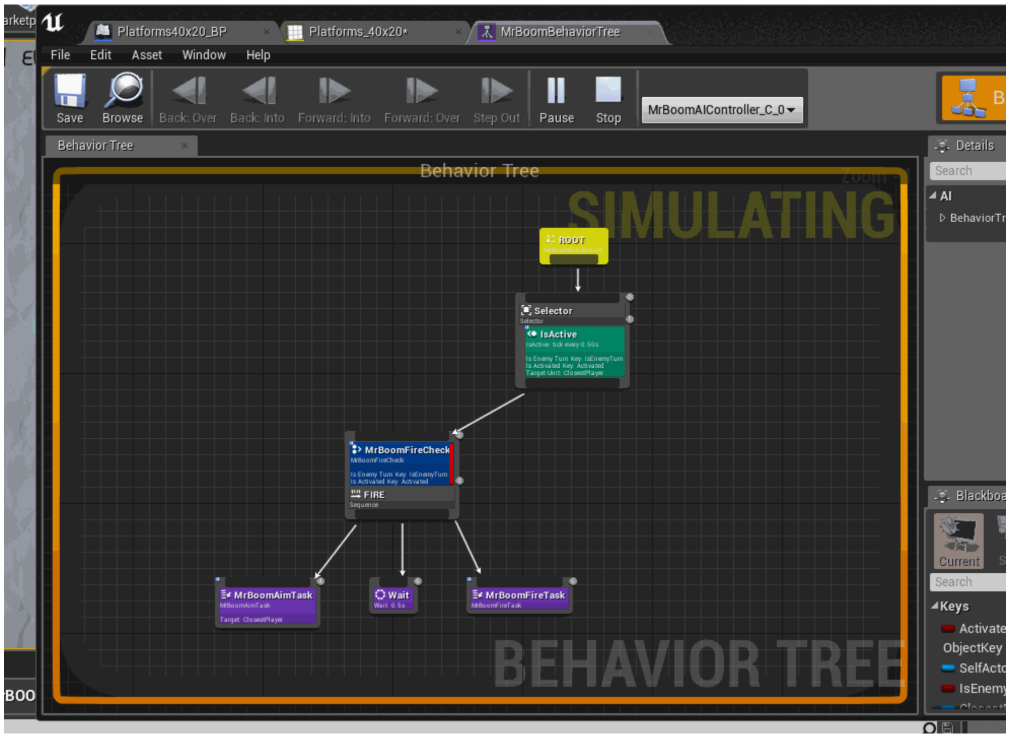
While I am extremely proud of this level generator, it does have some drawbacks. Currently, the level generator does not validate the designer's supplied room set to make sure that there are no impossible room combinations. Namely, if a room has a uniquely sized exit, it does not verify the list to ensure that another room with an appropriate entrance is also provided. This situation currently leads to the level generator terminating prematurely and placing the generic ending room after the offending room even if it does not fit. Early in development while I was making rooms, this issue caused a crash when no room could be found. Exiting level generation was the band-aid fix for that.

Nicolas LaCognata
CAP 4053

Mr. Boom

Overview:

Mr. Boom is a turret spawned into the level as a DLE. He uses a behavior tree to conduct his business. The behavior tree made programming his actions extremely easy and flexible. Each of his behaviors are broken up into tasks that are activated based on the rules set in the behavior tree. Because of his design and the nature of our level generator, making Mr. Boom and integrating him into our game so late into development was extremely easy.



Behaviors:

Mr. Booms behaviors are deliberately simple. He is the only entity in the game that can fire more than one shot at a time, so his accuracy is significantly reduced. He cannot move once he is set in the level. He merely fires directly at the nearest player unit in range with no regard for obstacles. The intention is that most of the damage Mr. Boom inflicts is to the cover the player is using to protect his units.

Nicolas LaCognata
CAP 4053

The Player System

Overview:

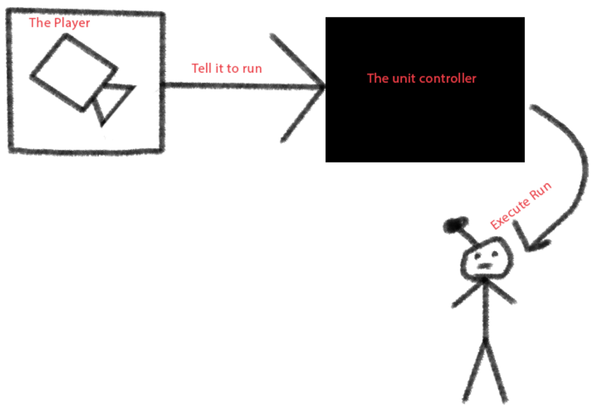
The notion of the "Player" in our game is much different from most of the other games we have seen in this class. Our player is a spectator to the game, rather than an element of it. The player has no representation in the game world, and the player pawn is simply a camera that moves through space. The player does not manually control any of the game elements. The player issues commands to his units.

Implementation:

The command structure of the game is as follows:

1. The player picks a unit
2. The player executes a command
3. Commands are passed to the unit controller of the selected unit for processing
4. The unit controller processes the player's request and decides whether to forward the player's command to its unit.
5. The unit receives a command and runs the code necessary to carry out that action.

This system allows the game to filter out invalid commands and restrict player control of units that are out of actions for the turn.

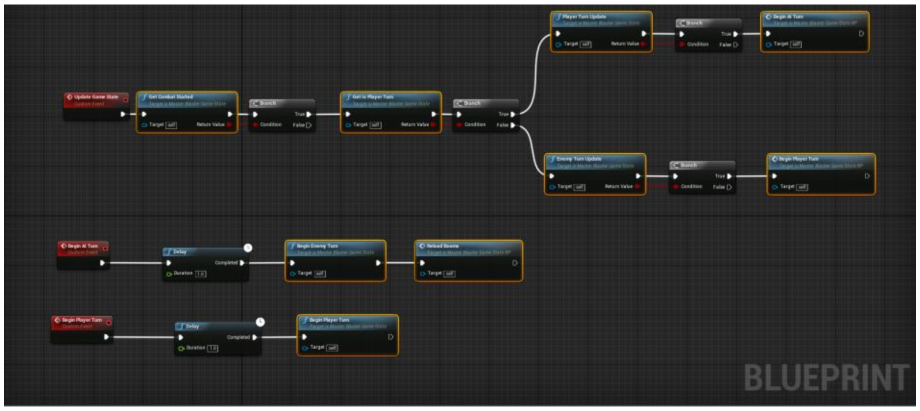


Nicolas LaCognata
CAP 4053

The Game State

Overview:

In our game, we make heavy use of the game's state to manage the transitions between the player turn and the CPU turn. We implement it through Unreal's built in game state system. The game state's most important function is maintaining lists of the units on the field. These lists are accessed at the start of each turn to refresh the active team's action points.



Issues:

The design of the game state system has been a fatal flaw of our game. It is in some ways a collection of glorified global variables, and it has introduced some extreme issues with segmentation faults when elements of the lists are destroyed. It also difficult to maintain and update without using a programmer's time. We never made a proper interface to standardize refreshing units on their turn, which has been a huge failure in keeping our code clean and maintainable.