**Tower Defence Design Document**

**Game Summary**

Tower defence revolving around heavily modifiable structures. The player must build their ark using their invaders resources.

**Player Experience Goals**

Focus on strategy, systems and progression -> all of this is aiming towards one special goal. Building your ark to escape the planet.

Game must be dynamic and slightly unpredictable

**Design Pillars**

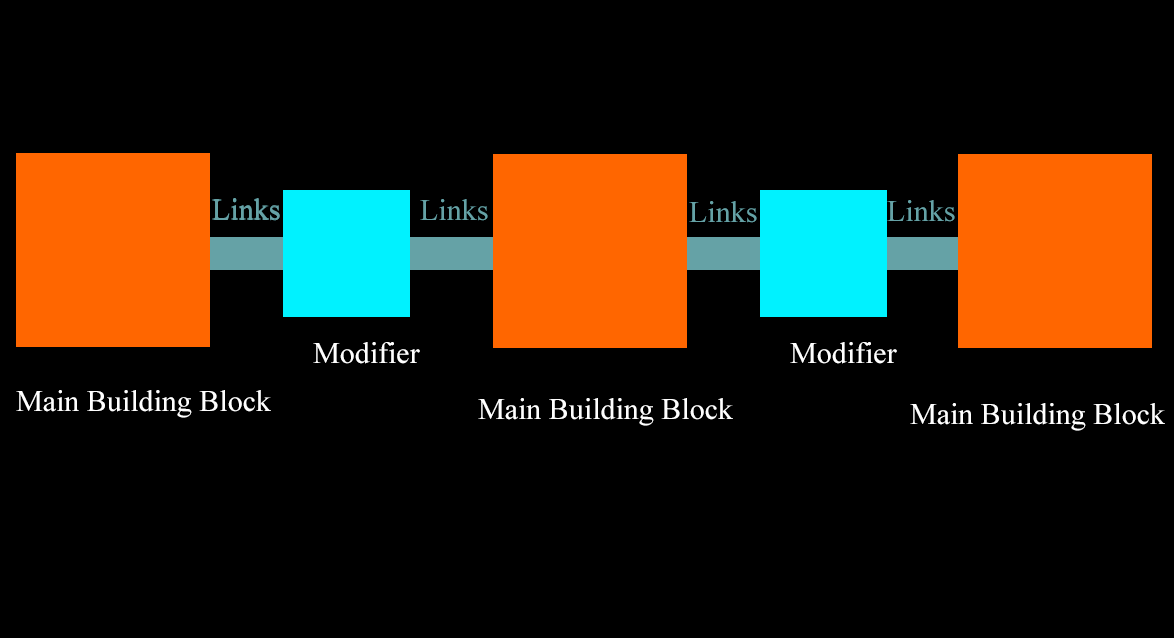
**Main Game Elements**

* Building and modifying towers; *deals with challenges and costs resources*
* Harvesting resources; *give resources, special material but increase the challenge*
* 4 types of 10-wave sets; *give challenge and resources*
* Construct ark; *final goal of the game*
* Ark abilities; *special moves*

**Core Mechanics**

Tower construction

Modifier components (can affect 2 towers at a time)



Modifiers affect only the towers that they are connected to it.

The focus of this mechanic is to combine modules to build the best tower defence system through good combinations.

Deconstruct tower or remove upgrades and get 75% back on resources spent for base structure.

Option to upgrade towers to increase general stats:

* Base damage
* Fire rate

**Modifier Behaviours**

Add pierce damage

Add blunt damage

Add virus

Increase fire rate

Increase base damage

Add disrupter damage (slow down enemies on hit)

**Spawns**

Spawn “types” is determined every 10 waves. There will be different styles of waves that the player needs to adapt to every 10 turns. These will also get gradually stronger as the game goes on.

**Special Materials**

Allows player to build special upgrades, these will affect entire tower sections. Can also be used to increase building of the Ark

**Ark building and abilities**

Place down fields that:

* Add properties to projectiles entering that area (via %)
* Slow down enemies
* Time dilation?
* Laser blast from the sky

**Art**

**Audio**

**Process**

I first began with the basics of tower defence mechanics, as this project was very much about the learning process. My first 2 main questions were:

How do I create an infinite enemy spawning system that gradually increases enemy amount?

How do I create a system that builds towers that have different kinds of behaviours?

How do I create different kinds of damage behaviours?

Modularity is key in this design and was an area within blueprints that I was unfamiliar with.

I began looking into interfaces and tags, as well as separating my objects behaviours and having them as separate but combinable entities.

I first created a base platform that picks up on whether or not the player has clicked it. Once clicked, it sends that information back to the player to let them know which platform they have selected. Depending on player input, another Blueprint structure of their choosing is spawned on top of the platform. This structure acts separately from the platform.

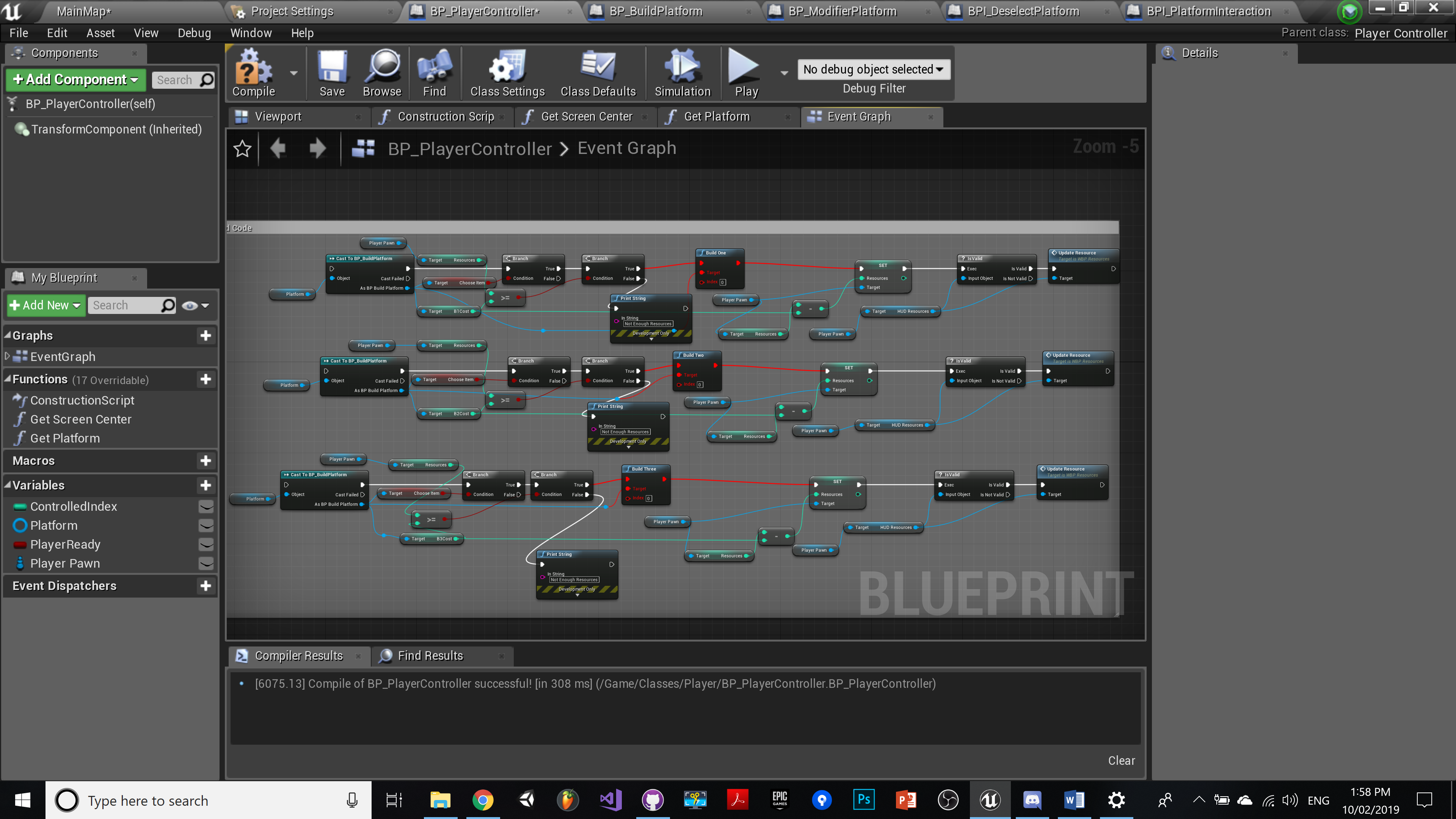
When clicking an actor, the controller checks if the hit actor has a BPI\_PlatformInteraction interface. If so, it then calls a ‘deselect’ interface event on an object reference, then calls a ‘select’ interface event on the newly hit actor, which sets the object reference to that actor instead. This event will check and uncheck a Boolean to let that platform know whether it has been chosen. This is how the game knows what platform is selected at any one time.

For spawning, I created a system in the game mode and “spawn point” blueprints that checked when the player had initiated the game – how many units for the current spawn, and when should they stop spawning and begin spawning again.

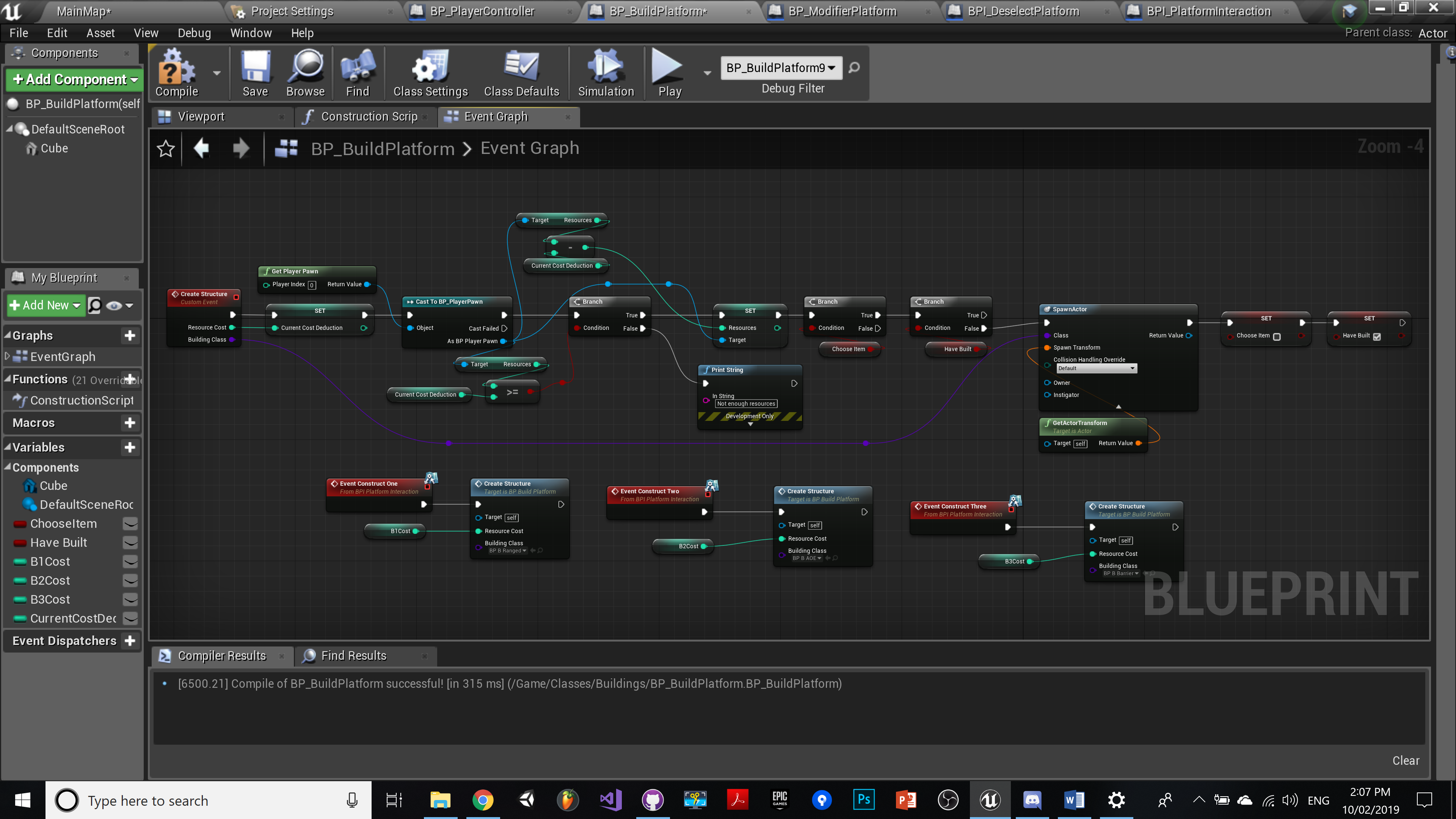
I have 3 types of structures – an AOE tower (affects things over time), a Mortar tower and a “Gunner” tower. Each of these will cover 3 core bases for how the player can affect enemies. The idea is start with a simple base for the game and to then build on top of that by adding more and more complexity and possibilities.

During the project I did a lot of refactoring. For example, when constructing buildings from the platforms, this was my old code:

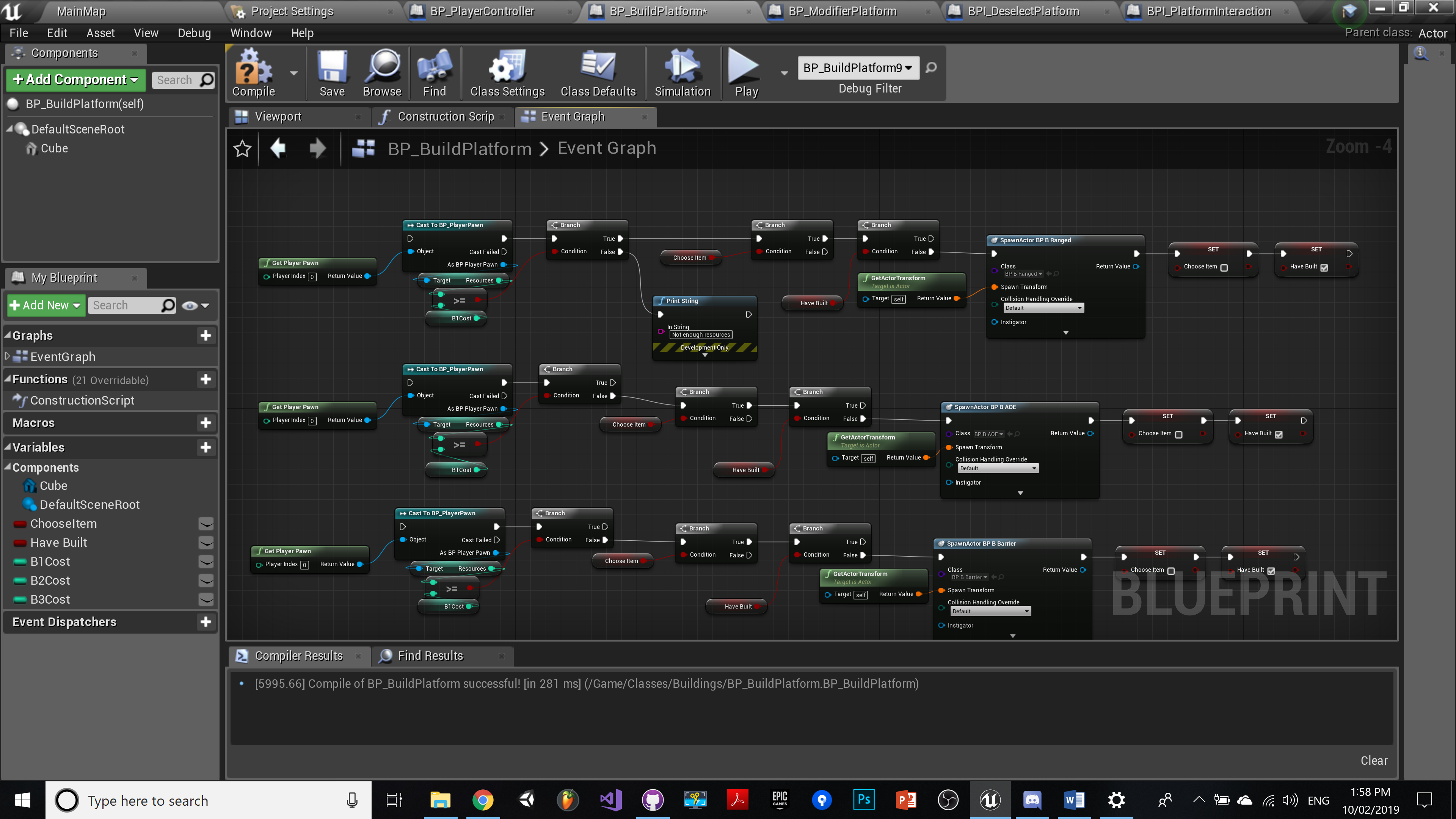
From this:



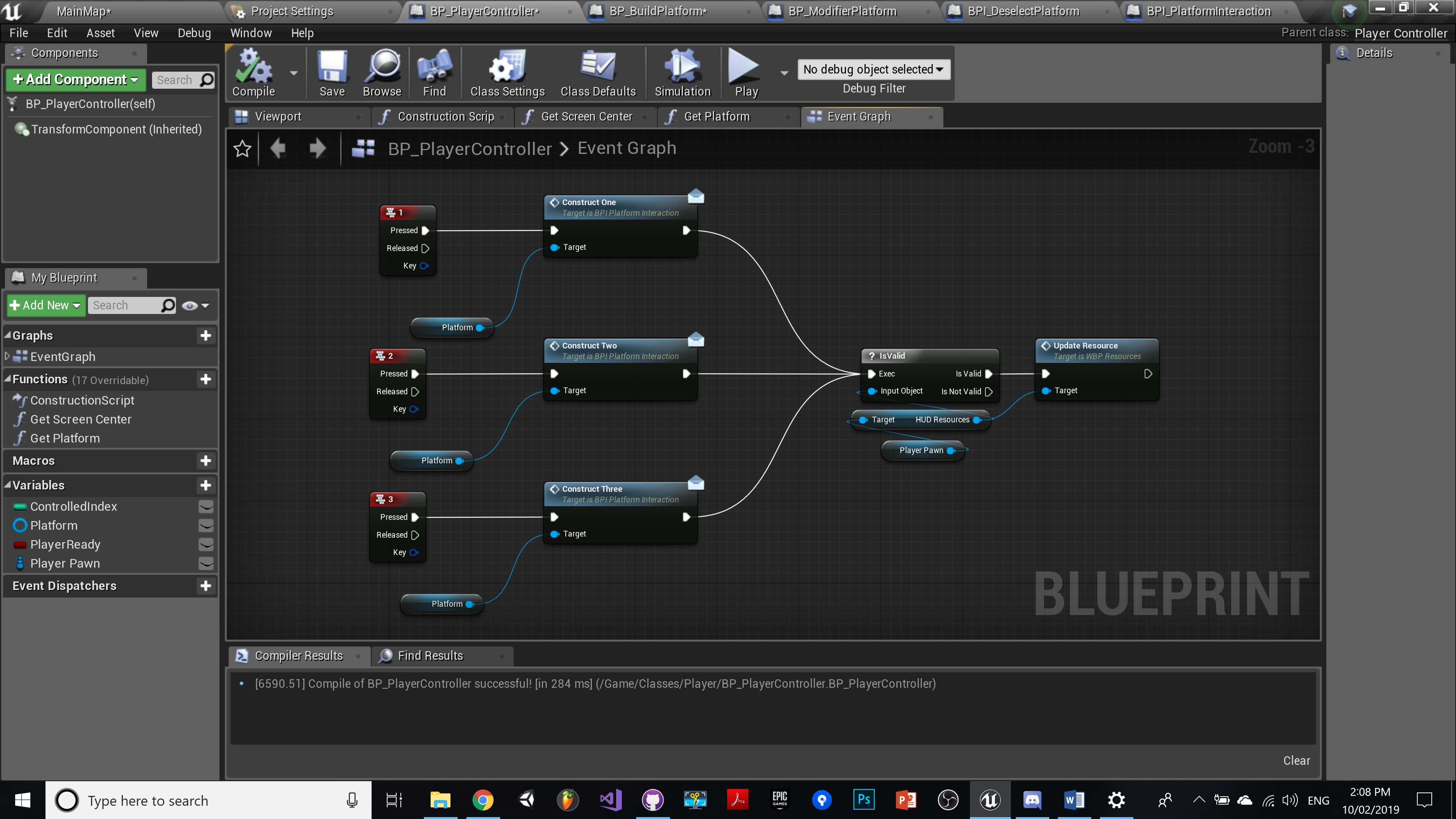
Down to this:



And from this:



Down to this:



(Calls the construction via interface from within the platform script instead)

Refactoring felt great and gave me much more flexibility as at this point I wasn’t going have just one type of building platform now, but at least 2 (and probably more)

The building platform, and then the module platform (which would add upgrades to nearby structures).

The module platform has been successfully implemented and now adds blunt damage to Mortar buildings. The same will need to be implemented for the ranged building as well.

The next step is to be able to upgrade modular buildings.

Current plan is to add same functionality for selecting platforms – basically tell the character controller what object it has selected, and then apply functionality based on input.

This same functionality will be needed for the regular tower structures as well.

Goals:

Upgradable Mods – pierce, blunt and base (up to 3 times to increase base, blunt or pierce damage)

Focus on Blunt mod first, then duplicate or use as base class (base class might be more ideal given that all mods act almost the same way) – this will require some thinking. When a child mod class is created, it will obtain the same functionality as the parent. So on begin overlap it adds structures to an array to keep track of connected structures.

Problem: This works when building before the mod is down, but building after is a problem. When a new building is built, it will be added to an array and mod upgrades can be called on begin play. But when built after, mod upgrades needs to be called for the new building, and only that new building alone. Perhaps in this case an enum state is better, rather than inheriting classes.

Upgradable Structures (2 separate upgrades that give different behaviours)

Once this system is working and shown visibly on the towers, the next step is to add virus damage and mods as well. And also a slow down effect on enemies hit as well (needs thinking on how this will be implemented and where)

**Iteration 2**

I have begun to use inheritance for my Modifier towers – the parent class has a trigger that collects nearby towers and adds them to an array. The child classes will then take from this array and run interface events based on what modifications they want to add.

The effects they can have is increased Pierce dmg, blunt dmg and an additional Virus dmg property (which will hurt enemies over time – basically poison). For the Gunner and HydroCannon (Mortar tower) this was relatively straight forward, however for the Rectifier tower (AOE barrier tower) I was unsure of what effect the modifiers could have on them. I ended up on settling for reducing resistance to piece and blunt damage. This made the most sense and be an easy thing for the player to pick up on.

At this point I am now thinking carefully about the kind of consistent feedback I need to give to the player so that they understand what is happening on screen. UI Will of course give some assistance, but the patterns of what towers do what and what effects are playing out in the scene should be clear to them from the get-go. The term ‘Affordance’ comes to mind – that is to add elements to the game that links well with how people perceive certain things, like poison, danger, speed etc. For instance, when they build a tower that is ideal for piercing heavy armoured targets, the tower needs to look like it was built for that. That means as a designer I need to identify the shape-language, colours and other elements that contribute to someone having that perception.

Virus damage – this was a tricky one – I have to define virus damage behaviour in every AI script (this is where having a base enemy class to inherit from would have been useful) and then I need to add in fire-rate, damage and a boolean for every attack ability initiated from every tower in order to inflict virus damage when the upgrade has occurred.

Lesson learned: For games like this, make use of inheritance. Only 1 level would have been required I.E. a base class for all attack structures, modifiers and enemies, as there are so many common behaviours within these types of objects.