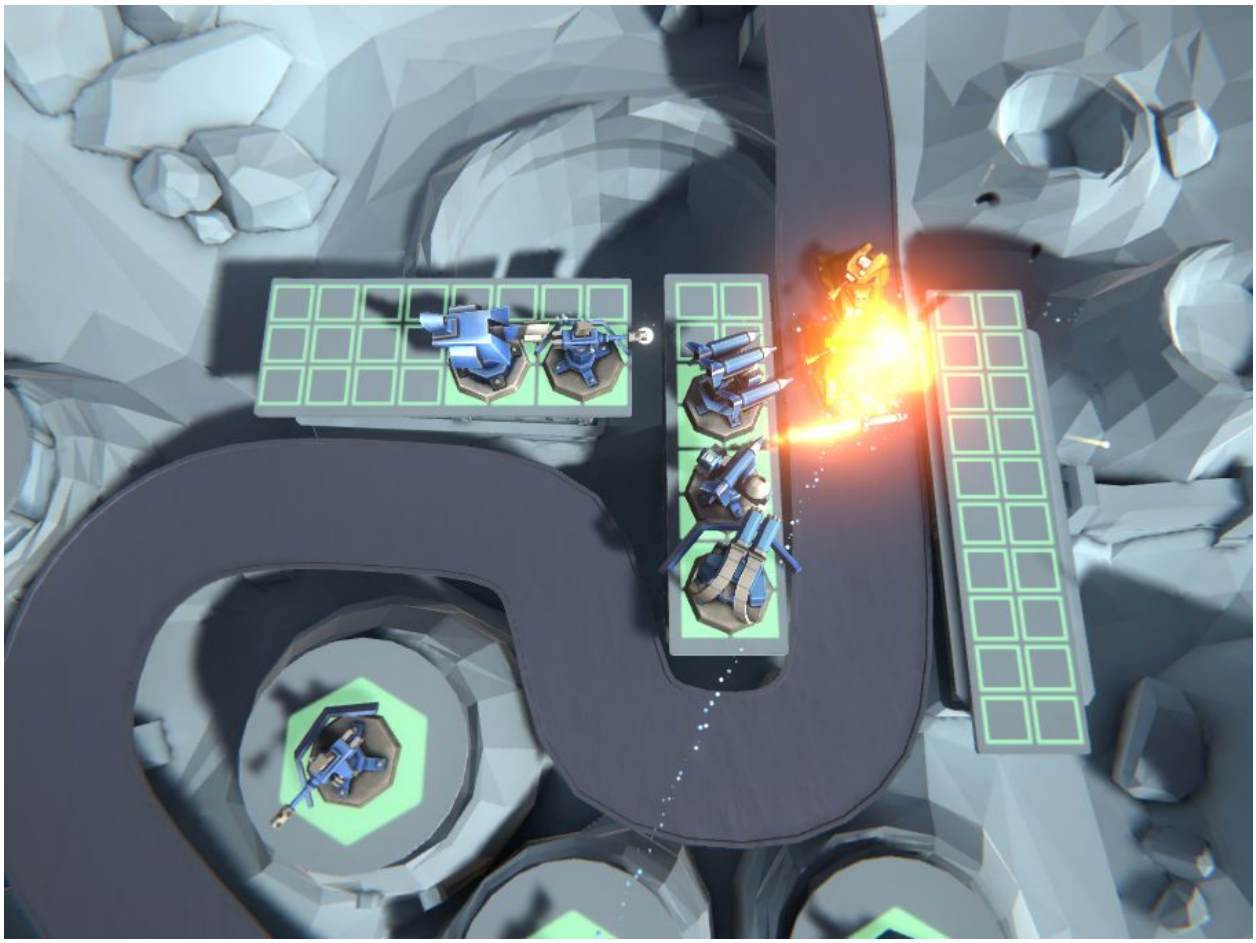
Unity Tower Defense Template



Introduction to the Tower Defense Template and Action Game Framework

About the Tower Defense Template

Unity's Tower Defense Template provides a simple but polished example of a tower defense game. This template serves two purposes: it teaches us best practices using a professional, real-world example and it provides us with a solid base on which we can build a more complex tower defense game of our own for both desktop and mobile platforms.

To understand this template and make our own game, we will need to understand how to work with the three most important parts of the template: towers, enemy agents, and game levels in which those enemies and towers can interact with each other.

The Tower Defense game

Tower defense games understand a tower as an object, fixed in place, that is used to shoot at enemies to stop them from reaching an objective point. Towers can also perform non-attack effects, like slowing enemies down, or generating more currency for a player to buy even more towers. The player will often have the option to upgrade towers to higher levels to make them more powerful or have a longer range. It is not necessary that these objects take the form of towers or turrets in the game. The “tower” part of “tower defense” is more of a historical convention than a rigid genre definition.

The Action Game Framework and the Core Framework

The Tower defense Template is made using two reusable frameworks.

The first framework is called the Action Game Framework. The Action Game Framework includes code that covers concepts needed in action games, such as taking damage and logic for ballistics and projectiles.

The second framework is called the Core Framework. The Core Framework covers concepts that are common to games of all genres, such as game saving, data management, timers, math utilities and more.

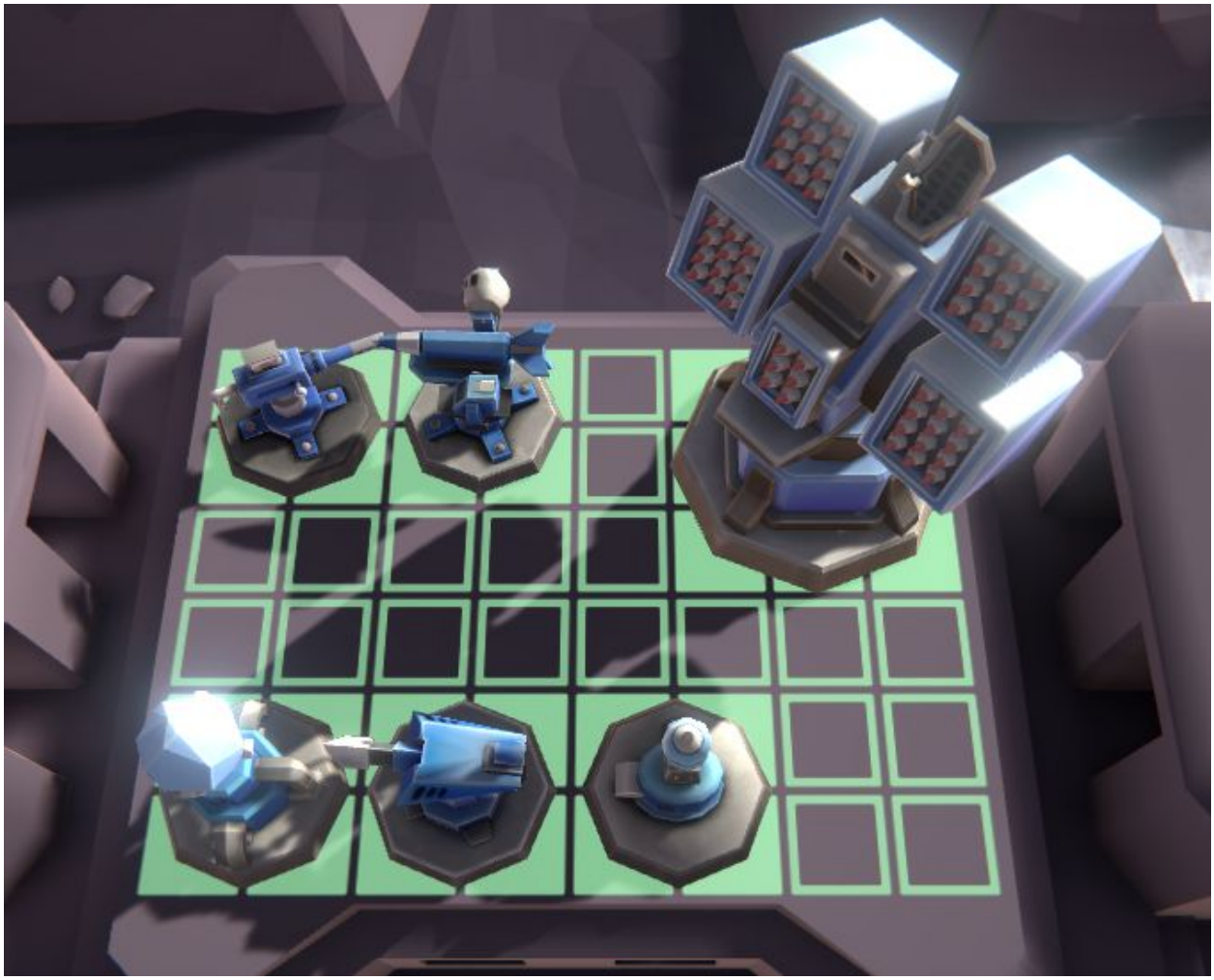
Important concepts in these frameworks will be explained in this document. For a thorough overview of the Action Game Framework, please see the Action Game Framework Reference

Setting up the project

First we will need to install and open the Unity editor. The Tower defense Template can be downloaded from the Asset Store window.

For the sake of this tutorial, we'll use a blank scene. The Starter Kit does come with several pre-made stages that you can view and compare your own work to.

Towers



Introduction

The most important element in the tower defense game is the tower. In the Tower Defense Template, there are two key elements that we use to create a tower. The first is the tower component, which defines basic functionality for the tower. The second is the TowerLevel component, which defines how a tower behaves when it is upgraded.

Creating a Tower

Let's create a new tower from scratch.

Before we create the Tower GameObject itself, we must create the data objects that store data about how it behaves at different levels. These objects are called TowerLevelData ScriptableObjects.

We'll begin by creating a TowerLevelData ScriptableObject to store data about the first level of our tower.

- In the Project window, click on the Create menu at the top right and choose **Create > Tower Defense > Tower Configuration**
- Give the ScriptableObject a name made up of its type and its level number (eg. Laser1)
- Place the ScriptableObject in Data/Towers
- Set values for tower name, health, and price

Now let's repeat this process to create the data object for the second level of our tower.

- In the Project window, click on the Create menu at the top right and choose **Create > Tower Defense > Tower Configuration**
- Give the ScriptableObject a name made up of its type and its level number (eg. Laser2)
- Place the ScriptableObject in Data/Towers
- Set values for tower name, description (shown when upgrading to this level), health, and price

Next, we need to create a TowerLevel Prefab for each of these levels. A TowerLevel Prefab contains a TowerLevel component with a reference to the ScriptableObject associated with that level of the tower. The Prefab will also have child objects that determine how the tower targets and attacks enemies.

Let's create a TowerLevel Prefab for the first level of our tower.

- In the Hierarchy window, click on the Create menu and choose **Create > Create Empty**
- Give the new GameObject a name to indicate its tower type and level (eg. LaserTower_1)
- Add a TowerLevel component to the GameObject
- Drag the ScriptableObject for this tower's level into the Level Data field
- Add meshes that represent the tower as children of the GameObject
- Create a folder for the tower in Prefabs/Towers
- Drag the GameObject from the hierarchy view into the newly created folder to make it a prefab

Now let's repeat this process to create the data object for the second level of our tower.

- In the Hierarchy window, click on the Create menu and choose **Create > Create Empty**
- Give the new GameObject a name to indicate its tower type and level (eg. LaserTower_2)
- Add a TowerLevel component to the GameObject
- Drag the ScriptableObject for this tower's level into the Level Data field
- Add meshes that represent the tower as children of the GameObject
- Create a folder for the tower in Prefabs/Towers
- Drag the GameObject from the hierarchy view into the newly created folder to make it a prefab

Now that we have the TowerLevelData ScriptableObjects and TowerLevel Prefabs we need, we can create the Tower GameObject itself.

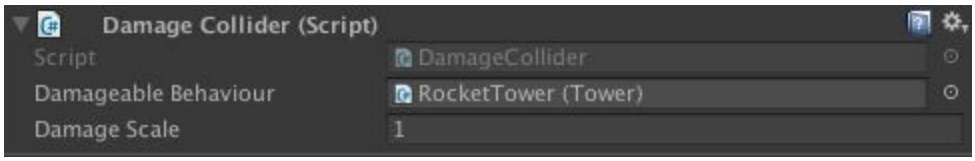
- Open a new Scene
- Create an empty GameObject
- Set the GameObject name to "NewTower"
- Add a Tower component to the NewTower GameObject

The Tower component requires a few other components to function, so we will next add these.

- Add an appropriate Collider component that encapsulates the tower's mesh to the NewTower GameObject

A Collider component is necessary for a tower to be able to determine if it was hit by a projectile. The Collider also establishes an area where a player can click on the tower after it has been placed to select it.

- Add a DamageCollider component to the NewTower GameObject



The DamageCollider component will handle collisions with objects that have Damagers (see Action Game Framework Reference).

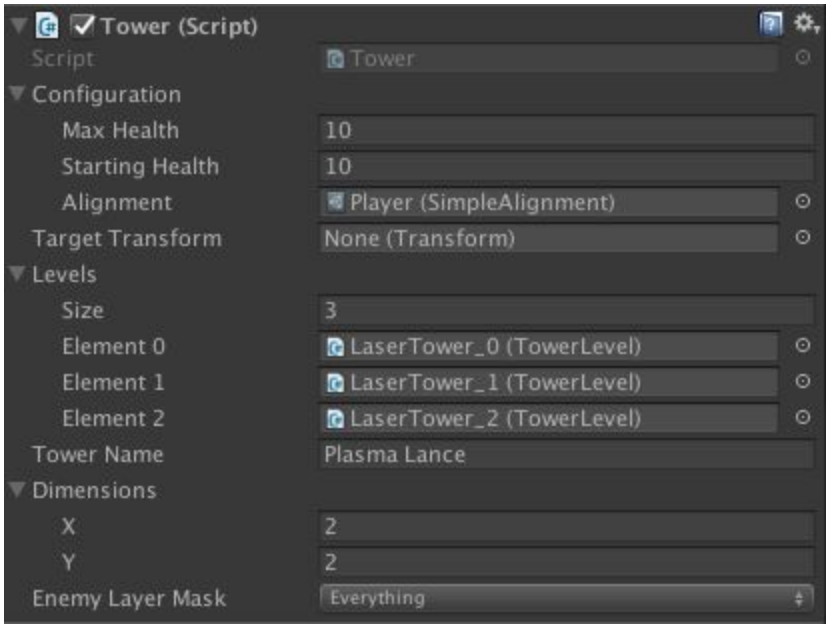
Now we can configure the settings on the Tower component on our NewTower GameObject.

- Drag in the Player SimpleAlignment ScriptableObject into the Alignment field
- Set the Target Transform if necessary
- Set the Size field under Levels to the number of TowerLevel objects required
- Drag the TowerLevel Prefabs for this Tower into the Element fields
- Name the Tower in the Tower Name field
- Set the X and Y values under Dimensions to set how much space the Tower requires to be placed on the placement grid
- Add a Health bar object as a child of the Tower GameObject

The Tower prefab will contain all visual information for that particular level of the tower. The Tower prefab will also contain the tower's affectors and launchers (described below).

Understanding the Tower component

This Tower component will allow us to set values that are common to all levels of the tower, and to set references to TowerLevel Prefabs that contain data for individual tower levels.



Configuration

Max Health

The maximum health of the tower.

Please note that if a TowerLevel Prefab is assigned for level 1 of the Tower, the value set here will be overridden by the MaxHealth value of that TowerLevel component.

Starting Health

The starting health of the tower.

Please note that if a TowerLevel ScriptableObject is assigned for level 1 of the Tower, the value set here will be overridden by the StartingHealth value of that TowerLevel ScriptableObject .

Alignment

Alignment determines which potential target the tower will fire at. We set this with a reference to a SimpleAlignment ScriptableObject.

The starter kit provides two SimpleAlignment ScriptableObjects:

- Player - the tower will fire at enemy units.
- Enemy - will fire at anything with a Player alignment.

A SimpleAlignment ScriptableObject is a simple implementation of how to represent teams for a game. A SimpleAlignment is configured using a list of other SimpleAlignments that it considers enemies and can attack and damage. The Tower defense Template includes the two alignments above, but it is possible to create other custom SimpleAlignments.

To create more alignments, from the Project window choose **Create > Starter Kit > Simple Alignment.** Alternately, we can implement our own version of the IAlignmentProvider script if we need more complicated logic.

Target Transform

Ordinarily when Towers or Agents target each other, they will aim for the origin of their transforms. The TargetTransform allows us to set an alternate point to fire at.

Levels

Here we can set the number of levels the tower has, and also associate a TowerLevel Prefab (more detail on this below) with the Tower component.

Tower Name

This field allows us to enter a name for the Tower that is displayed to the player.

Tower Description

This field allows us to enter a short description for the Tower that is displayed to the player. We can use this to tell the player about the tower's purpose and abilities.

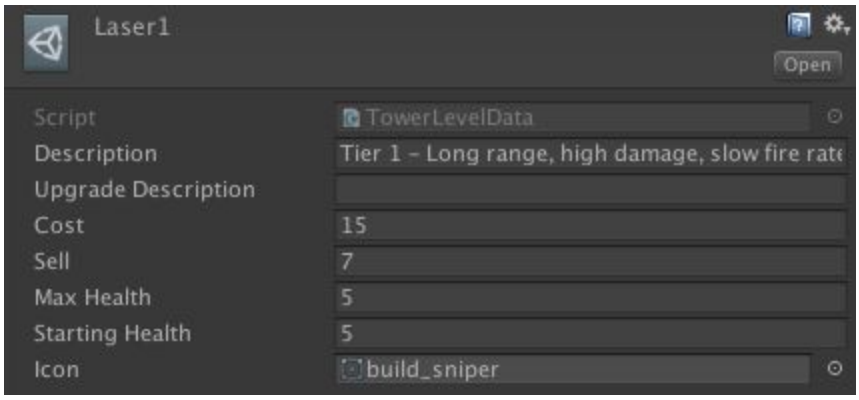
Dimensions

Here we can set how much space the tower takes up on the placement grid.

Enemy Layer Mask

This determines which physics layers the tower can target.

Understanding the TowerLevelData ScriptableObject



Description

This is a short string that will appear on the UI when a user selects a tower of this type.

Upgrade Description

A short string that informs the user how the tower will change when upgraded. Important to note is that the text is displayed when upgrading to this tower level, so the first level will not require one.

Cost

Cost refers to how much it will cost for the player to buy or upgrade to this level of the tower.

Sell

Sell refers to how much money the player will receive for selling the tower during the game.

Sell is usually lower than Cost, but it doesn't have to be. Note that in the example project the sell value for higher level towers takes into account the accumulated currency the player has spent on all tower levels.

Max Health

The maximum health of the tower, when it is upgraded to this level.

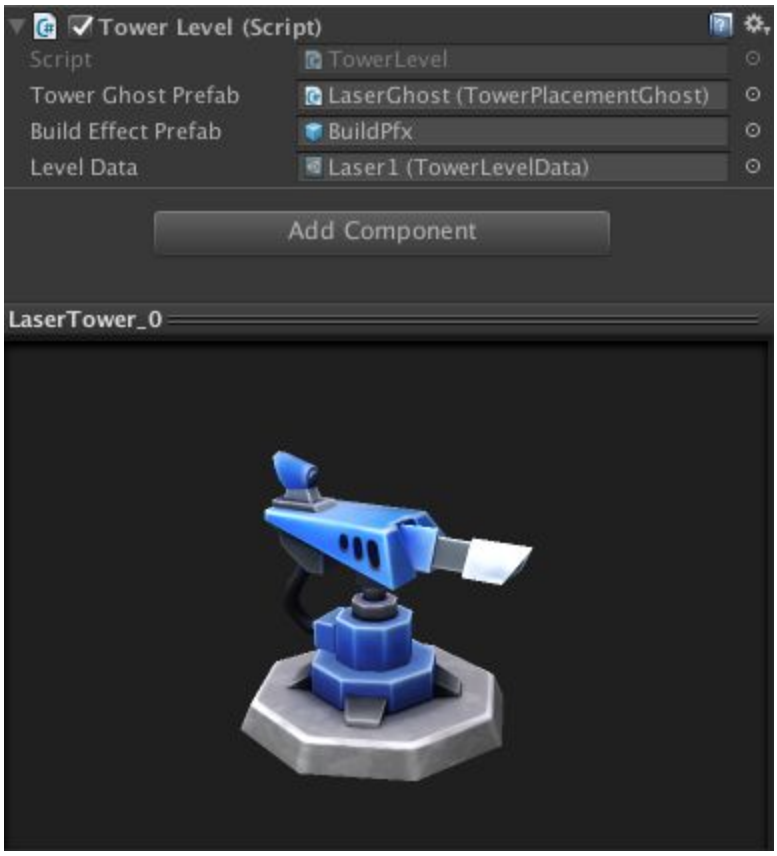
Starting Health

The starting health of the tower, when it is upgraded to this level.

Icon

The Icon field can be used to assign a sprite to represent the tower in UI. More information on this and how it ties into the list of available towers for each stage is available in the **Setting Up a Stage** section.

Understanding the TowerLevel component



Tower Ghost Prefab

The preview model shown when a player is placing a tower.

Build Effect Prefab

The particle effect Prefab assigned here will be instantiated whenever the tower is built or upgraded, along with its associated sound effects.

LevelData

The TowerLevelData ScriptableObject that contains the data for this level.

Agents

Introduction

Now that we have a tower, we will need something for the tower to defend against. This next section will explain how to create enemy Agents that will try to attack the player's home base. A large part of the process of getting them to navigate around a stage will be covered in **Setting Up a Stage**, but we will not be able to do that without first setting up an Agent GameObject with the necessary components.

Note that we will be using the term 'Agent' instead of 'Enemy' to refer to these autonomous actors.

Creating an Agent

The first step is to make an Agent Configuration.

- In the Project window, click on the Create menu at the top right and choose: **Create > Tower Defense > Agent Configuration**

We now have an Agent Configuration, where we can set values for the agent.

Next, we must create the Agent GameObject.

- Create an empty GameObject
- Name the GameObject NewAgent
- Add an AttackingAgent or FlyingAgent component to the GameObject
- Drag the Enemy SimpleAlignment ScriptableObject into the Alignment field
- Add a mesh to visualise the Agent as a child object of the Agent GameObject
- Create an empty child GameObject of the Agent and name it Targetable. This object will ensure that Towers view the Agent as a valid target of attack

- Select the Agent in the Hierarchy window and drag the Targetable into the Target Transform field of the AttackingAgent component

Note that a NavMeshAgent component is added automatically upon adding an AttackingAgent or FlyingAgent component. The NavMeshAgent component is required to ensure the Agent can move around the level. The values defined within it specify how it does that.

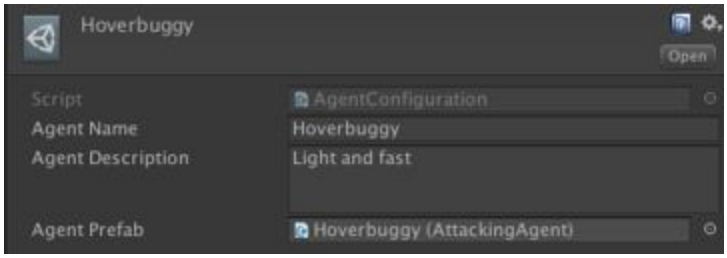
As is the case with towers, an Agent cannot be defined simply by attaching a AttackingAgent or FlyingAgent component to a GameObject. Some other components will be required to ensure that the Agent carries out its required behaviours.

- Add an appropriate Collider component that encapsulates the Agent's mesh to the NewAgent GameObject

A Collider is necessary for an Agent to be able to determine whether or not it was hit by a projectile.

- Add a DamageCollider component to the NewAgent GameObject

Understanding the AgentConfiguration



Agent Name

This is the name of the Agent.

Agent Description

A description string to describe the Agent and its capabilities.

Agent Prefab

The Prefab that contains the AttackingAgent or FlyingAgent component for this Agent.

Understanding the AttackingAgent and FlyingAgent components

The Starter Kit includes two different Agent components: AttackingAgent and FlyingAgent. These two agents respond differently if their path is blocked.

AttackingAgent will stop and attack nearby towers until a path becomes available and depends on an AttackAffector to operate (see **Targeting and Firing** below). FlyingAgent will simply fly over obstacles directly to its destination, ignoring usual NavMesh behaviour.

Both of these components extend the Agent base class, and are configured in exactly the same way. We can also implement our own logic for how agents behave by creating scripts that extend the Agent base class.

Max Health

The maximum health of this Agent.

Starting Health

The starting health of this Agent.

Alignment

Alignment defines which "team" the Agent is on, and therefore what the Agent targets and is targeted by. When creating enemy Agents, this will usually be set to Enemy.

Target Transform

Ordinarily when Towers or Agents target each other, they will aim for the origin of their transforms. The TargetTransform allows us to set an alternate point to fire at.

Applied Effect Offset

If a Tower applies an effect to the Agent, this value allows us to adjust the position of that particle effect.

Applied Effect Scale

If a Tower applies an effect to the Agent, this value allows us to adjust the size of that particle effect.

Targeting and Firing

Introduction

The previous sections provide instructions on how to create Agents and Towers, but don’t cover how to actually have them fire at each other. This is, in large part, because the same components and processes are required for both enemies and towers. We will go through these in the following section.

Creating a Projectile

Before making Agents and Towers capable of shooting, we'll need to make the projectiles for them to shoot at each other. The amount of damage they'll do is a property of the projectile, rather than the Tower or Agent itself.

- Create an empty GameObject and give it a name made up of its Tower/Agent type, the word 'projectile', and the level it will be associated with, if appropriate (eg. LaserTowerProjectile_1)
- Add a Damager component to the new Projectile object
- Set the Damage field to indicate the amount of Health that should be removed from Towers or Agents the projectile hits
- If the projectile is for a Tower, drag the Player SimpleAlignment ScriptableObject into the Alignment field. If it is for an Agent, drag the Enemy SimpleAlignment ScriptableObject into the Alignment field

Further steps depend on what kind of projectile is being created. For a hitscan projectile, which doesn't rely on ballistics or collision detection:

- Add a HitscanAttack component to the Projectile GameObject.
- Set the Delay field to a desirable number of seconds.

For a projectile such as a rocket, where we check to see if and where it collides with an object to determine damage:

- Add a BallisticProjectile component to the Projectile GameObject
- Add a RigidBody and CapsuleCollider component to the Projectile GameObject to ensure that collisions between the Projectile and the environment can be detected
- Add a ContactDestroyer component, which will destroy the Projectile after a collision

If the projectile is meant to do damage in a wider radius, take the following steps:

- Add a SplashDamager component to the Projectile GameObject
- Set the desired radius for the area of effect in the Attack Range field
- Set the value of the Damage Amount field to the amount of damage that Agents/Towers not directly hit by the Projectile should take

Adding Targeting to a Tower

The following steps will enable targeting on both Towers and Agents. The Starter Kit uses a separate child GameObject for the AttackAffector on Towers, whereas the component is attached to the Agent Prefab itself. This is purely an organisational difference, and does not change how the AttackAffector functions.

- Create an empty child GameObject of the Tower level/Agent Prefab and name it Affector. This object will ensure that Towers and Agents can fire at each other
- Add an AttackAffector component to the Affector GameObject
- Drag a projectile Prefab into the Projectile field
- Add a Projectile Point as a location from which the Tower/Agent will fire the projectile
- Add an appropriate Launcher component to the Affector GameObject
- Set the rate of fire for the Tower level/Agent
- Create an empty child GameObject of the Tower level/Agent Prefab and name it Targetter. This object will ensure that the Tower is able to find Agents to fire at
- Drag the Tower level's turret Transform into the Turret field. The object set as the turret will point towards Agents
- Set the Tower level's search rate
- Set the Tower level's collider to be a sphere or capsule
- Set the Radius and Vertical Range fields
- Select the Affector GameObject in the Hierarchy window and drag the Targetter GameObject into the Tower Targetter field in the AttackAffector component
- Apply changes to the Tower prefab

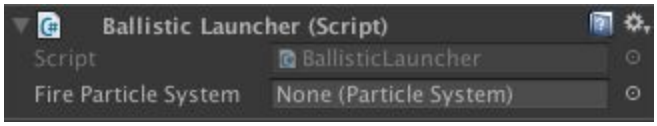
The above steps will ensure that an Agent can fire at Towers that block its path to the player's home base. In order to make the Agent damage the home base as well, take the following steps

- Add a HomeBaseAttacker to the Agent Prefab
- Set the number of seconds the Agent should spend charging up before attacking when it reaches the home base

Understanding the Launcher components

There are a number of launchers available in the Tower defense Template.

Ballistic Launcher

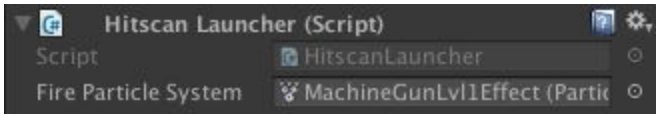


The BallisticLauncher calculates the trajectory of the projectile using physics and launches a BallisticProjectile. More information on the BallisticProjectile can be found in the Action Game Framework Reference.

Fire Particle System

The Particle System that should play when the tower fires using this launcher.

Hitscan Launcher

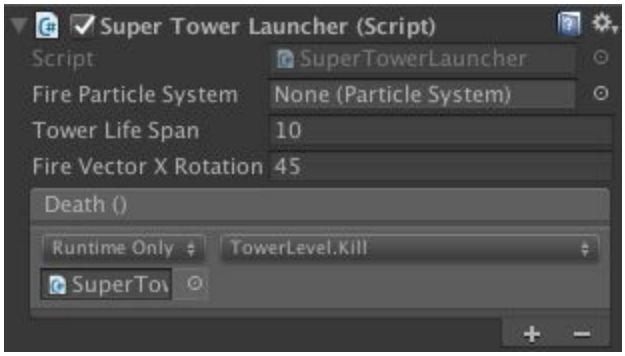


The HitscanLauncher immediately places a HitscanAttack on a target. The HitscanAttack will then (with an optional delay) damage the target.

Fire Particle System

The Particle System that should play when the tower fires using this launcher.

Super Tower Launcher



The Super Tower appears in the example project, it automatically destroys itself after a certain amount of time.

Fire Particle System

The Particle System that should play when the tower fires using this launcher.

Tower Life Span

The number of seconds before the Super Tower self-destructs. This field is included to balance the powerful Super Tower by preventing it from being in play for too long.

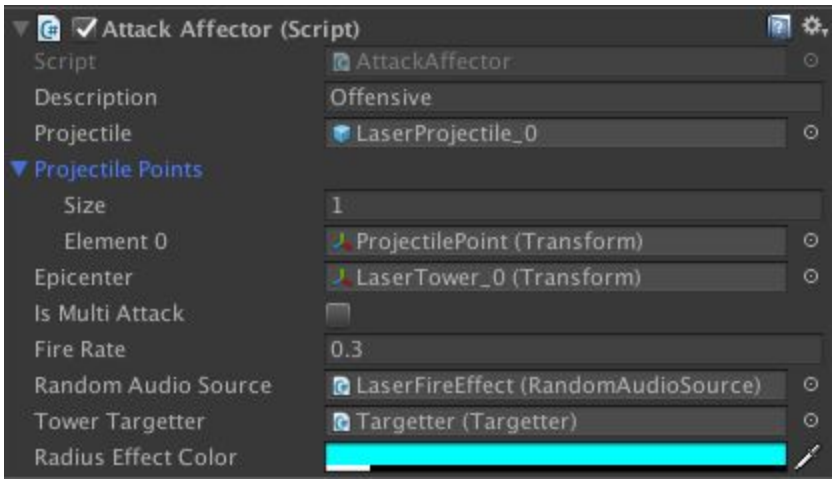
Fire Vector X Rotation

FireVectorXRotationAdjustment represents the amount that the SuperTower's turret rotates based on the direction the SuperTower's projectiles (see WobblingHomingProjectile in Action Game Framework Reference).

Death()

The function the SuperTower uses to destroy itself is exposed to allow for configurable handling of the Super Tower's self-destruct function. This allows us to prevent the Super Tower from killing itself, for example.

Understanding the AttackAffector component



The AttackAffector component allows a Tower or Agent to fire at each other.

Projectile

The Prefab that is spawned when this AttackAffector fires at a target. The specific type of Projectile will depend on the Launcher component.

Projectile Points

A list of Transforms. When a Transform is added to this list, projectiles may spawn at the position of that Transform. This is useful for towers or enemies that have multiple turrets.

Is Multi Attack

If set to true, the AttackAffector will be able to find multiple targets if they are in range. If set to false, the Affector will target one enemy at a time until either the enemy is destroyed or the enemy moves out of range.

Fire Rate

This controls how often the AttackAffector will tell the Launcher to launch a projectile.

Random Audio Source

A list of AudioSources. A random one will play each time the AttackAffector fires.

Tower Targetter

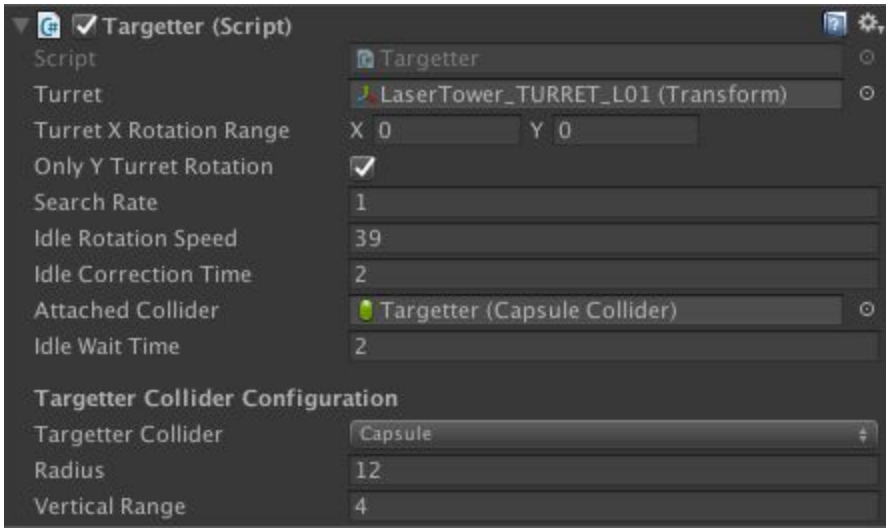
A reference to the Prefab that contains the Targetter component for this Tower level.

The Targetter keeps track of the number of targets in range and specifically the closest target to the Targetter's epicentre. For feedback, it also controls the turret mesh which is a child of the Tower level Prefab and aims it at the nearest target to communicate to the player that it is the closest one.

Radius Effect Color

This changes the color of the range visualizer that shows how far the tower can fire.

Understanding the Targetter component



Turret

The turret mesh attached to the prefab for this tower level.

Turret X Rotation Range

Value in degrees that determines how far the Tower can look up and down.

Only Y Turret Rotation

Whether or not the Tower can rotate around the X axis.

Search Rate

The Targetter's SearchRate configures how often a new closest target is selected.

Idle Rotation Speed

How fast the tower rotates when it has no target to fire at.

Idle Correction Time

The time in seconds the Tower will wait after losing a target before it returns to its neutral position.

Idle Wait Time

The time in seconds the Tower will wait after losing a target before it begins its idle rotation again.

Attached Collider

A reference to the Targetter Collider set below. This is automatically set when using the Targetter Collider field.

Targetter Collider Configuration

The Targetter's search volume can either be a capsule or a sphere. When the collider is a sphere, the radius can be configured. When the volume is a capsule the vertical range of the capsule can also be configured.

Targetter Collider

Whether the Targetter's collider is a capsule or sphere.

Radius

The Targetter's firing range in standard Unity units.

Vertical Range

How far above or below itself the Tower is able to fire.

Understanding the HomeBaseAttacker component

Home Base Attack Charge Time

This defines the amount of time (in seconds) between the Agent reaching the final node in the level, and it reducing the health of the player's home base.

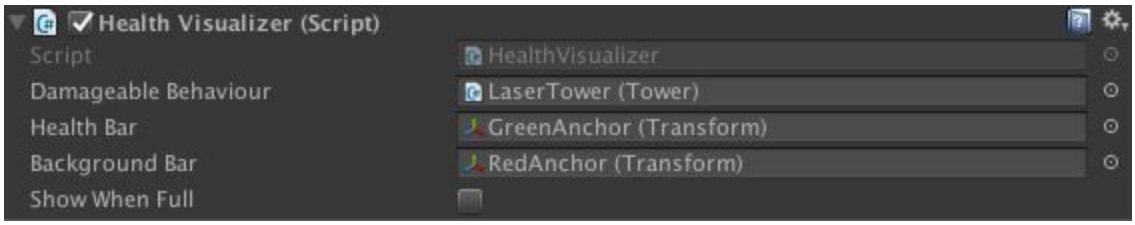
Other Effects, Visualizers and Behaviours

Adding Visualization for Health

To show the player how much health an Agent or Tower has remaining, we can do the following:

- Drag in the HealthBar Prefab from Prefabs/UI and make it a child of the GameObject with a Tower, AttackingAgent, or FlyingAgent component
- Select the HealthBar object and drag its parent GameObject into the Damageable Behaviour field

Understanding the HealthVisualizer component



It is also common to have health bars appear over objects in the game that have health. To visualise the health of a Damageable or DamageableBehaviour, we can add a HealthVisualiser to the object.

Note that the Starter Kit comes with a Health Bar prefab, which can be used instead of making assets from scratch.

Damageable Behaviour

Here we can set a reference to a Prefab, and the HealthVisualiser script will attach the health bar to the related object.

Health Bar

This is the bar that will show how much health the Agent or Tower has remaining. It will become smaller as the object it is attached to takes damage and loses health.

Background Bar

This bar appears behind the health bar, and becomes visible when the object takes damage. This is useful to show the difference between the unit’s current health and its maximum health.

Show When Full

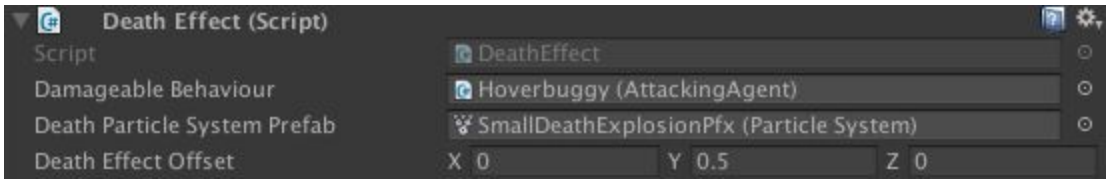
If this is turned off, the Tower or Agent will only display its health bar after it has first taken damage.

Adding Visualization for Death

Currently, Towers and Agents will simply be removed from the stage when they are destroyed. We can make this process more dramatic and noticeable to the player by adding effects when they are destroyed.

- Add a DeathEffect component to a GameObject with a Tower, AttackingAgent, or FlyingAgent component
- Drag the Tower or Agent Prefab into the Damageable Behaviour field
- Drag a Particle System that should play on the unit's death into the Death Particle System Prefab field
- If the Particle System's position is not quite where it should be, use the Death Effect Offset to adjust it

Understanding the DeathEffect component



Damageable Behaviour

Here we can set a reference to a Prefab, and the DeathEffect script will create a particle effect when that object is killed.

Death Particle System Prefab

This can be used to assign a particle prefab that will be created when the Tower or Agent is destroyed.

Death Effect Offset

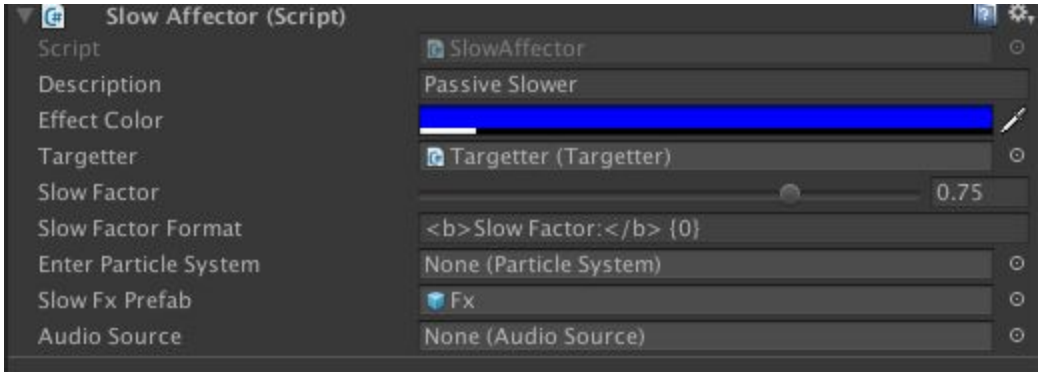
This can be used to adjust the position of the Particle System if it appears in the wrong place in relation to the unit's death position.

Adding a Tower that debuffs Agents

We can also add the ability for a Tower to slow Agents down when they are near it. This can work on its own or in addition to the Tower attacking the Agents.

- Add a SlowAffector component to the Affector GameObject
- Drag the Tower's Targetter Game Object into the Targetter field
- Set the Slow Factor value to a desirable amount
- Drag an effect Prefab into the Slow Fx Prefab

Understanding the SlowAffector component



Description

A short string to describe what the script's effect is.

Effect Color

The color of the range visualizer for the Tower ghost.

Targetter

References the Tower's Targetter component. This ensures that the SlowAffector effect can be applied to Agents targeted by the Targetter.

Slow Factor

The modifier applied to the Agent's speed when it is in the Tower's radius.

Slow Factor Format

A string that shows the player how much Agents in the Tower's effect radius are slowed down.

Enter Particle System

The Particle System that will play when the Agent first enters the Tower's effect radius.

Slow Fx Prefab

The Particle System Prefab that will play while the slow effect is applied to an Agent.

Audio Source

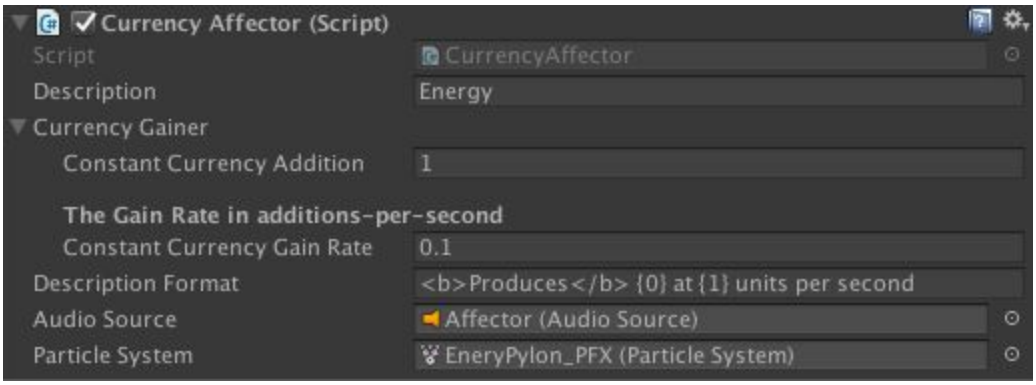
The Audio Source that should play while the slow effect is applied to an Agent.

Adding a Tower that passively adds currency

We can also add the ability for a Tower to passively generate currency over time:

- Add a CurrencyAffector component to the Affector GameObject
- Set the Constant Currency Addition field to the amount of currency that should be added at set intervals
- Set the Constant Currency Gain Rate to the number of times the addition should be made per second

Understanding the CurrencyAffector component



Description

The name of the currency.

Currency Gainer

Constant Currency Addition

The number of currency units added to the player's total every time the script adds currency.

Constant Currency Gain Rate

The number of times per second the script will add currency to the player's total.

Description Format

String to show players what the currency is and how much is added.

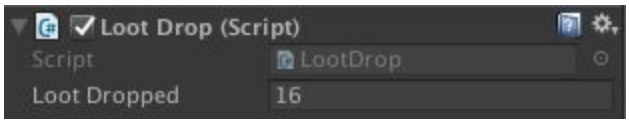
Audio Source

The Audio Source that will play whenever the Tower adds currency to the player's total.

Particle System

The Particle System that plays whenever the Tower adds currency to the player's total.

Adding rewards for destroying Agents



In order to give the player more currency when their Towers destroy enemy Agents, we can take the following steps:

- Add a LootDrop component to the Agent Prefab
- Set the value of the Loot Dropped field to the amount of currency we want to add to the player's total when they destroy the enemy

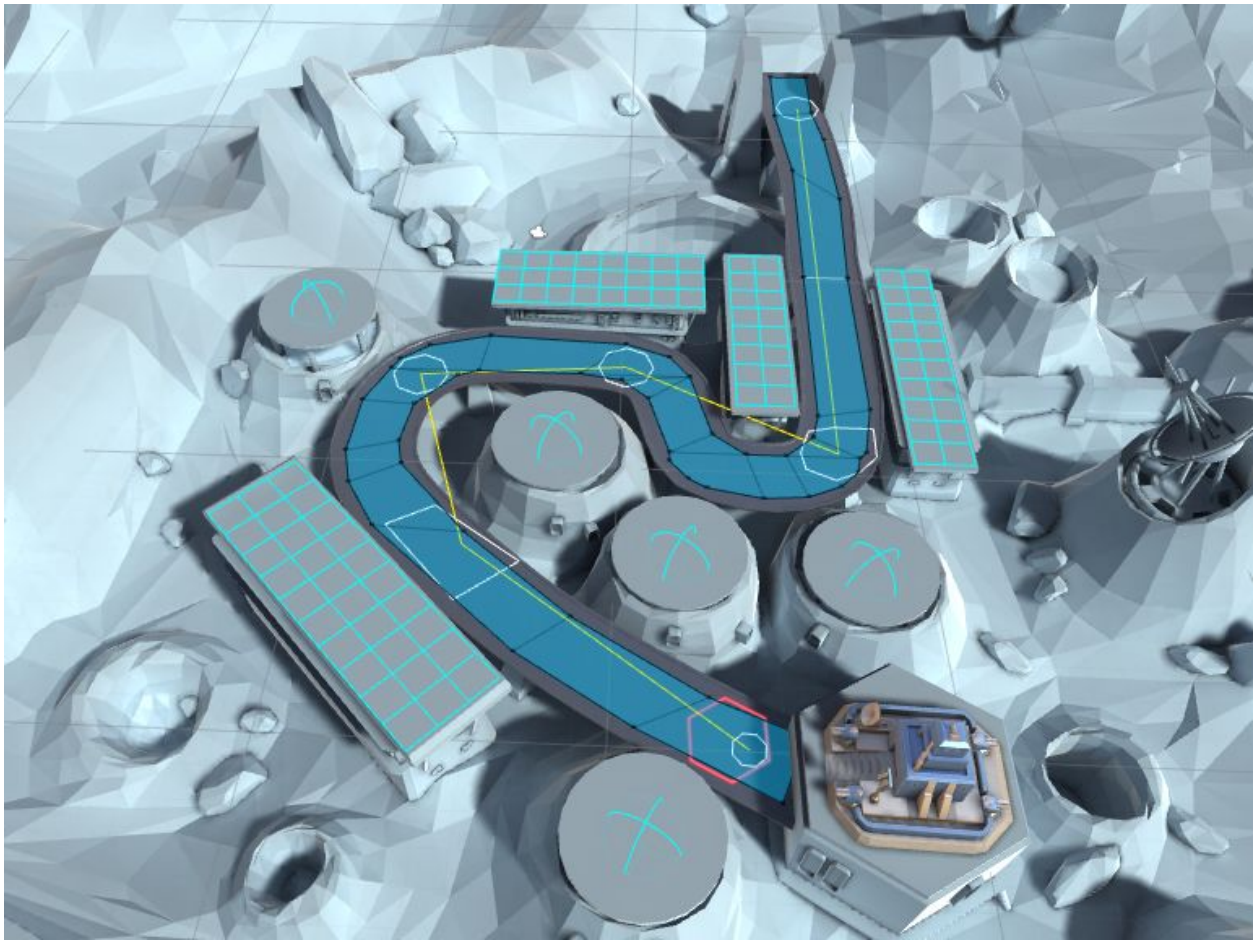
Understanding the LootDrop component

We might choose to start the player off with only enough currency to place one or two Towers, and have them build up more over time by destroying enemies. This functionality is made possible by the LootDrop component

Loot Dropped

This is the amount of currency that will be added to the player's total when the Agent is destroyed. Conventionally, weaker enemies will give smaller amounts of currency.

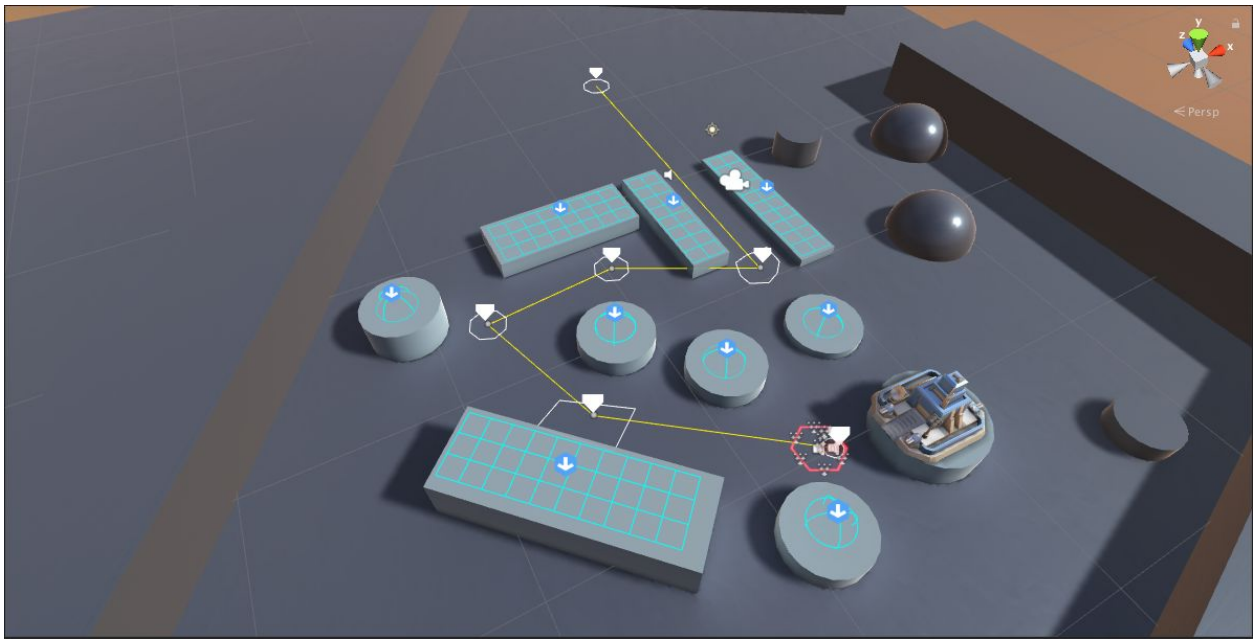
Setting Up a Stage

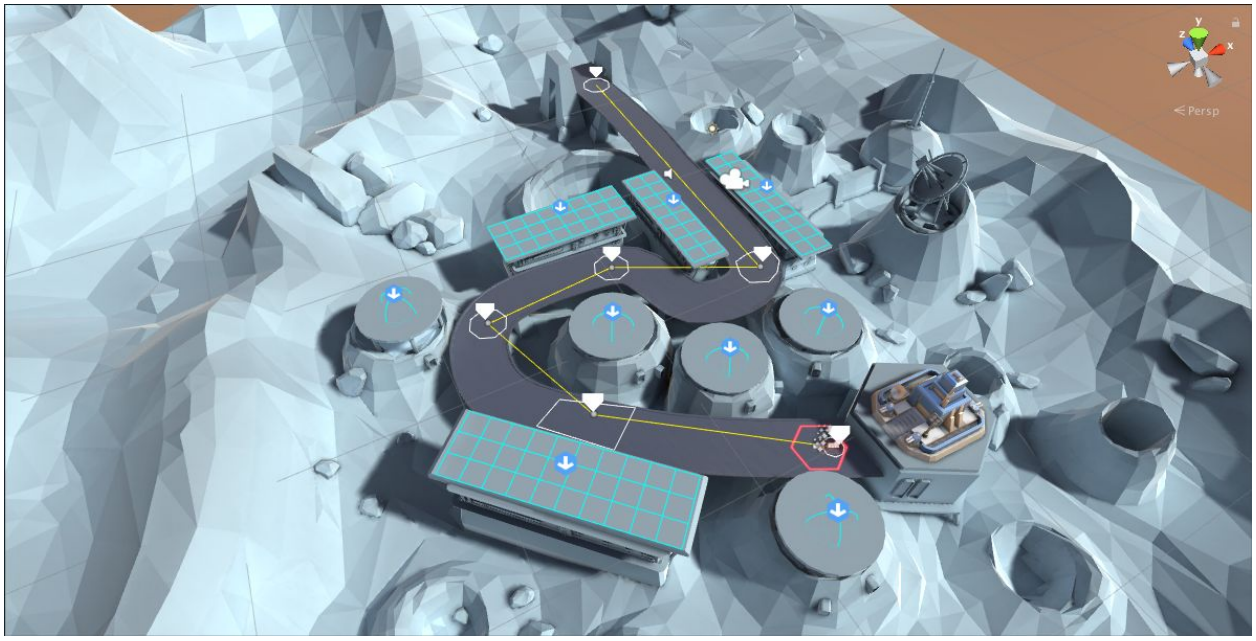


Introduction

Now that we have created towers and enemy Agents with a variety of effects, the next step is to set up a stage where players can build towers to defend a location from those enemies.

Prototype Level





When creating a game, it is often worthwhile to create placeholder art assets to playtest early design work. This allows us to ensure that systems and level design work as intended before committing time and effort to making high quality art that needs to be changed in the event that the design needs to change.

To show this stage of the development process, the starter kit includes a prototype version of stage 1. The prototype stage is roughly the same as stage 1, but the environment art is much simpler than the final version that is playable in the game. Instead of the complex world geometry of the final stage, the prototype is made up entirely of 3D primitives available in the Unity editor.

When going from the prototype to the final stage, the following technical considerations were made to produce a release-quality stage:

Mesh Extension

The stage mesh was extended to ensure that players would never be able to see outside of the bounds of the playing area.

World Mesh Combination

The stage mesh was broken up into chunks to make sure that it is not too large. Meshes much larger than the camera frustum are inefficient because many verts are drawn offscreen. It is best to try to avoid this while keeping as much of the world combined into fewer meshes to minimize draw calls and frustum culling costs.

Collision Mesh

A single approximate collision mesh was created. This is so that projectiles will hit the world objects, as well as to project the cursor into the world when placing towers.

Removing Non-Visible Faces

Non-visible faces are removed from world objects were removed. For the most part, this includes faces at the bottom of objects. Removing these faces reduces overdraw which can reduce performance, especially on lower-end devices.

The above steps should be considered when transitioning from prototype art assets to final, higher fidelity assets. Following them will result in a stage that is performant and easy to manage. A stage made using these steps will contain all required objects and components to ensure that the stage works correctly.

The rest of this tutorial focuses on the step-by step creation of a stage from a blank Unity scene, and does not extensively cover the process of creating high quality art assets.

Level List

If we want our stage to be accessible to players through the menu systems, we will need to create a level list and add the stage to it. We can achieve this by following these steps:

- From the Project window select **Create > Starter Kit > Level List**
- Add the details of the stage to the LevelList ScriptableObject

The stage will now appear in the Level Select menu in the game.

Understanding the LevelList ScriptableObject

The level list allows us to set the number of stage to be shown in the main menu, and we can set the following information for each stage:

ID

The order in which the stage will appear in the menu.

Name

A title for the stage.

Description

A brief string to give players an idea of what to expect from the stage.

Scene Name

The name of the scene that contains the stage.

Setting Up Geometry

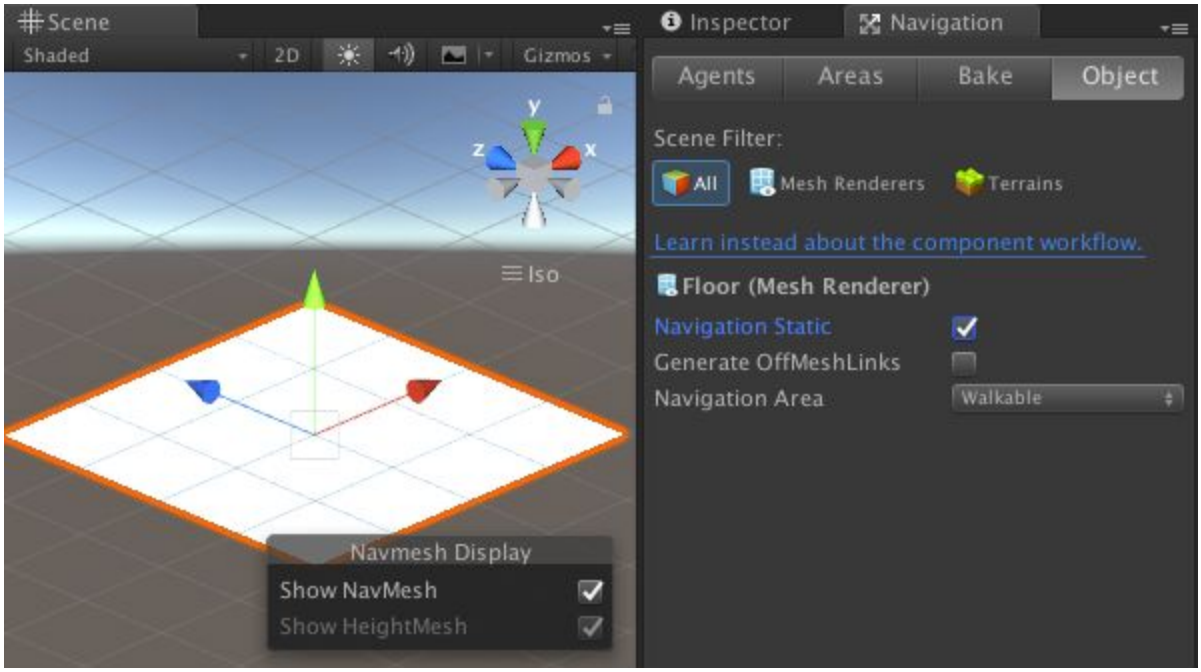
- Add a large plane to the blank scene created earlier. From the Hierarchy window, **Create > 3D Object > Plane**
- Ensure that the plane's transform position is set to (0, 0, 0)
- Add 3D objects to the stage to establish the layout. Cubes and cylinders are best for this. From the Hierarchy window, **Create > 3D Object > Cube/Cylinder**

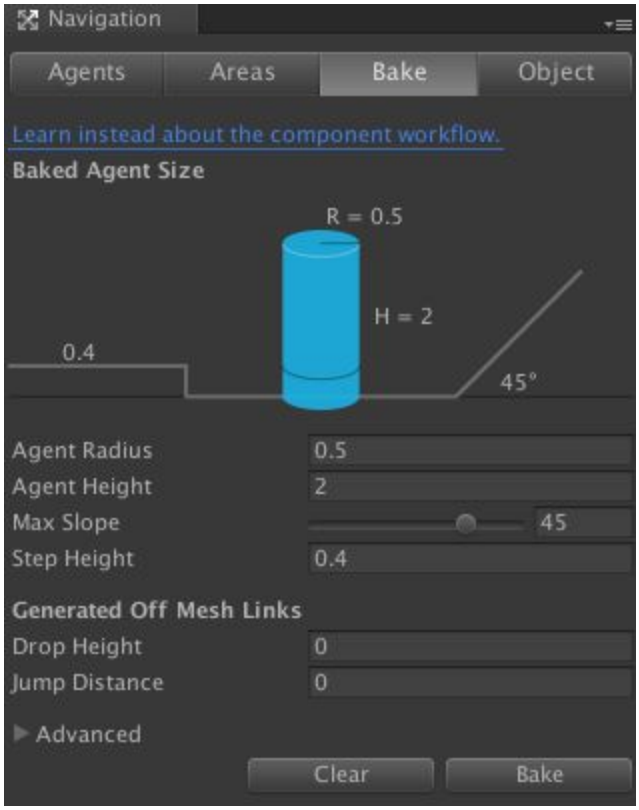
If we already have a more complex mesh created, we can bypass this step and use that instead.

NavMesh Setup

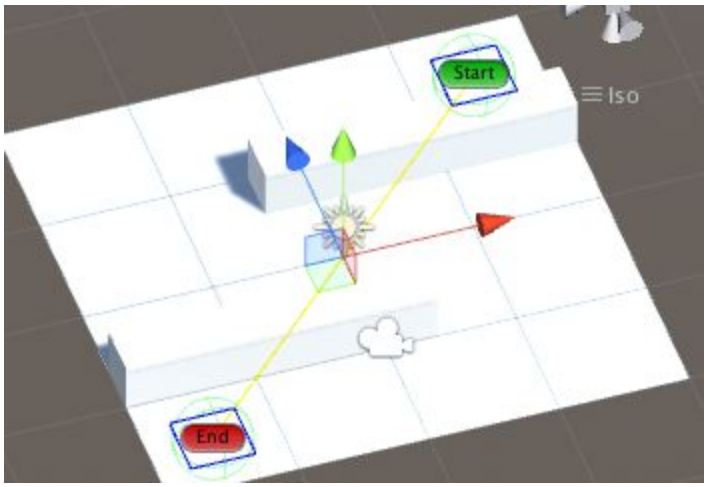
In order to make the stage geometry navigable, we will need to bake the NavMesh.

- Select the plane or stage mesh
- Open the Object tab in the Navigation window
- Check the Navigation Static option
- Set the Navigation Area to Walkable
- Select the 3D objects that Agents shouldn't pass through
- Open the Object tab in the Navigation window
- Check the Navigation Static option
- Set the Navigation Area to Non-Walkable
- Bake the NavMesh


Scene
Shaded
2D
Gizmos
y
z
x
Iso
Navmesh Display
Show NavMesh
Show HeightMesh
Inspector
Navigation
Agents
Areas
Bake
Object
Scene Filter:
All
Mesh Renderers
Terrains
Learn instead about the component workflow.
Floor (Mesh Renderer)
Navigation Static
Generate OffMeshLinks
Navigation Area
Walkable


Navigation
Agents
Areas
Bake
Object
Learn instead about the component workflow.
Baked Agent Size
R = 0.5
H = 2
0.4
45°
Agent Radius
0.5
Agent Height
2
Max Slope
45
Step Height
0.4
Generated Off Mesh Links
Drop Height
0
Jump Distance
0
Advanced
Clear
Bake

Navigation Nodes

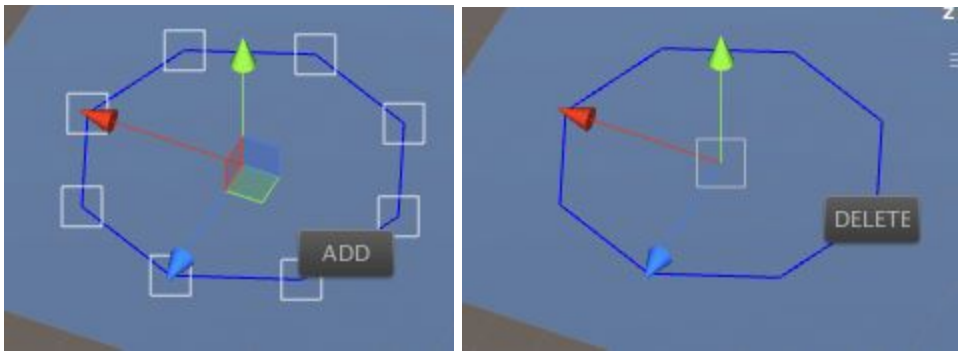


For enemies to move around the stage, we will need to set up nodes for them to be able to go from point to point. Nodes are also used to define spawn points and points at which the player home base will take damage, should an agent reach it.

We can create a node by adding a Node component to an empty GameObject. This lets us create a mesh which defines where agents will spawn and provides the agents with a way to select a random point within the Node area to navigate to. Being able to select a random point helps ensure that agents do not cluster to a single point.

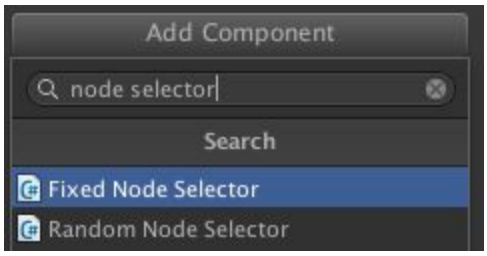
- Create an empty GameObject and name it Navigation Nodes or something similar
- Create an empty GameObject and make it a child of Navigation Nodes, name it StartNode
- Add a Node component to the Start GameObject
- Click the Select/Add Mesh button
- Adjust the child GameObject mesh to be the desired shape
- Add a Sphere or Capsule Collider component to the StartNode GameObject
- Set the Is Collider checkbox to true
- Ensure that the Collider encapsulates the node mesh

The Node component allows us to add a new mesh, which will create a child object with an Area Mesh Creator component, where we can define preset shapes or add points to the mesh by clicking between its corners in the Scene view. Points can be removed by holding shift and clicking the points.



In order to define the next node in the sequence, we will need to add a node selector component, either a Fixed Node Selector, which sends the enemies to a specific node, or a Random Node Selector, which will select from a weighted list of possible nodes.

- Create another node using the steps listed above
- On the StartNode GameObject, add a FixedNodeSelector component
- Add an element to Linked Nodes and drag the new node into the created field



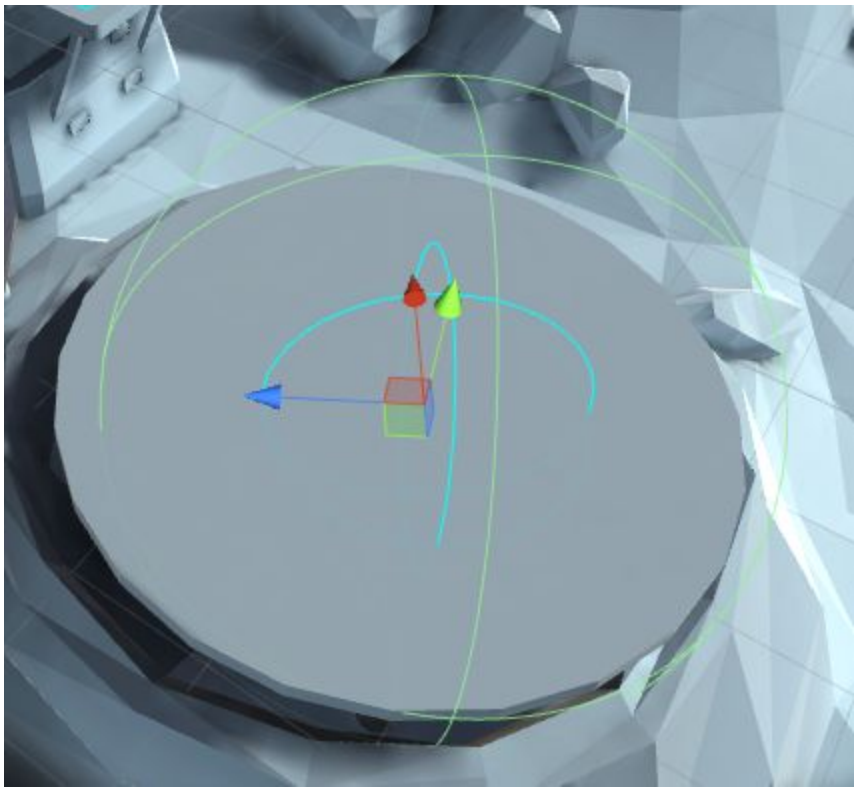
The above steps can be repeated as necessary, linking nodes in the order Agents should travel to them, to make sure that they will follow the desired path through the level.

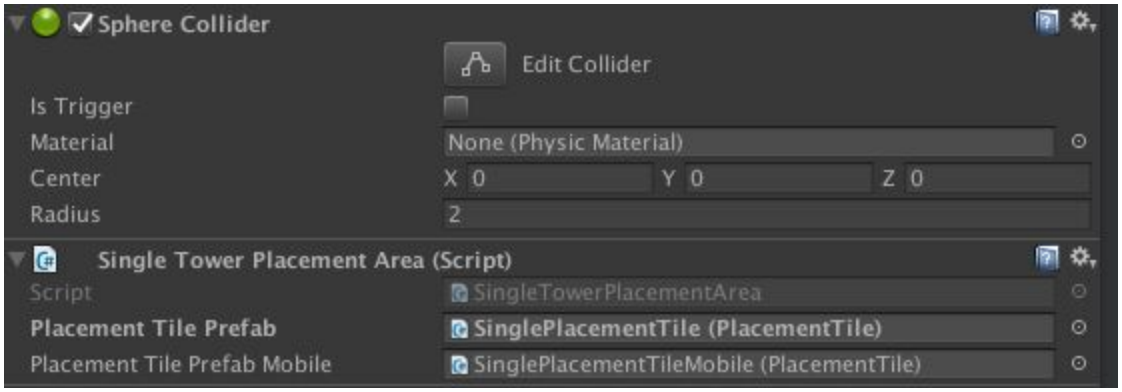
Tower Placement Areas

Towers in the Starter Kit require IPlacementAreas for a player to be able to place them within a stage. After adding an implementation of IPlacementArea to an empty GameObject, we will be able to define places where towers can be placed.

The starter kit contains two implementations of IPlacementArea:

SingleTowerPlacementArea



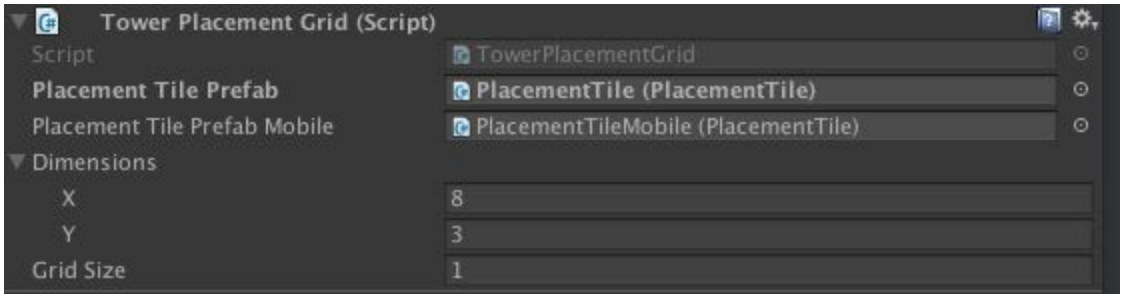


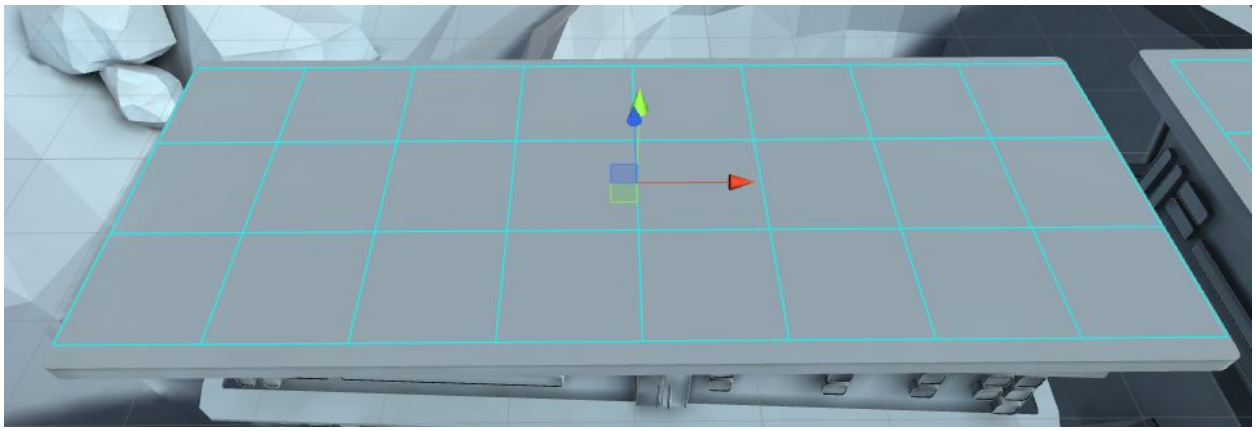
SingleTowerPlacementArea is a IPlacementArea that can contain only one tower.

- Create a new GameObject and call it TowerPlacementArea
- Add a Sphere Collider to the TowerPlacementArea object
- Add a SingleTowerPlacementArea component to the TowerPlacementArea object
- Drag the SinglePlacementTile prefab from **Prefabs/UI** to the Placement Tile Prefab field

- Set the TowerPlacementArea's layer to PlacementLayer

TowerPlacementGrid



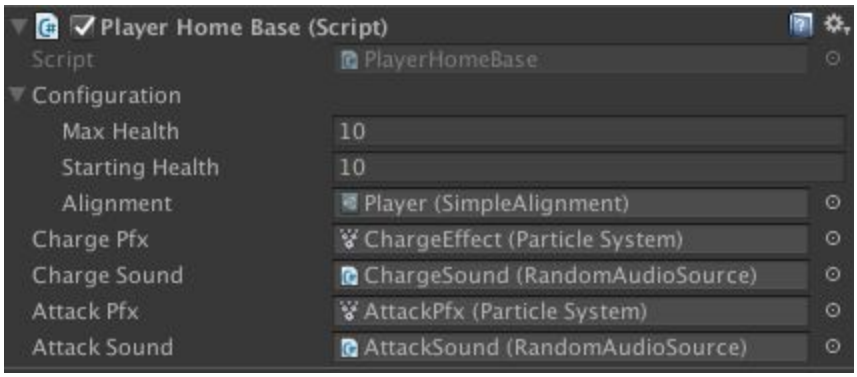


A TowerPlacementGrid represents a gridded area where the player can place multiple towers.

- Create an empty GameObject and name it PlacementGrid
- Add a TowerPlacementGrid component
- Set the desired dimensions for the grid
- Drag the PlacementTile prefab from **Prefabs/UI** to the Placement Tile Prefab field

Unlike the SingleTowerPlacementArea, automatically creates a collider for itself set to the correct size, for input handling.

Adding a Home Base



Enemies do not damage the player's health when reaching the final node. To remedy this, we'll need to make some modifications to the final navigation node.

- Add a PlayerHomeBase component to the final node GameObject
- Set the home base's Max and Starting Health
- Drag the Player SimpleAlignment ScriptableObject into the Alignment field

The Player Home Base has fields to assign Particle Systems. One for a charge effect and another for the attack effect. These are not required.

Understanding the PlayerHomeBase component

Configuration

Max Health

The maximum health of the player's home base.

Starting Health

The amount of health the home base will start with.

Alignment

Refers to the ScriptableObject that defines which objects can target and damage the home base.

Charge Pfx

The Particle System will play while an Agent is charging its attack on the home base.

Charge Sound

The sound will play while an Agent is charging its attack on the home base.

Attack Pfx

The Particle System will play while an Agent is done charging and attacks the home base.

Attack Sound

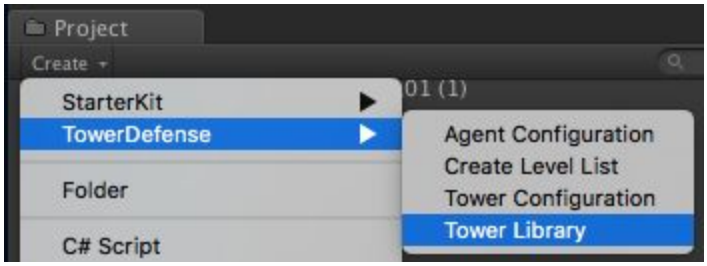
The sound will play while an Agent is done charging and attacks the home base.

The attack effect will be played at the conclusion of the charge, when damage is applied, if assigned.

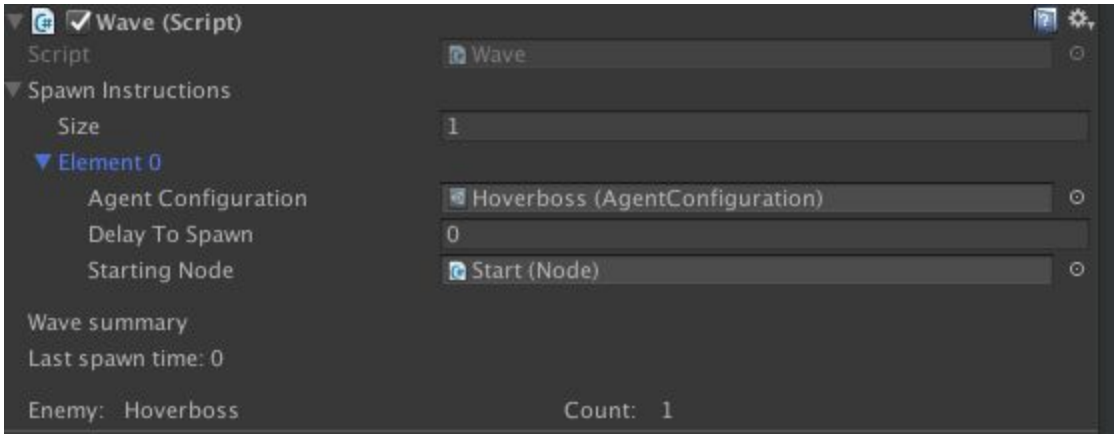
Adding a Tower Library

Now that we have created areas where towers can be placed, we need to set up which towers are available in our stage.

- From the Project window: **Create > Tower Defense > Tower Library**
- Drag in Tower prefabs that will be available to the player in this stage

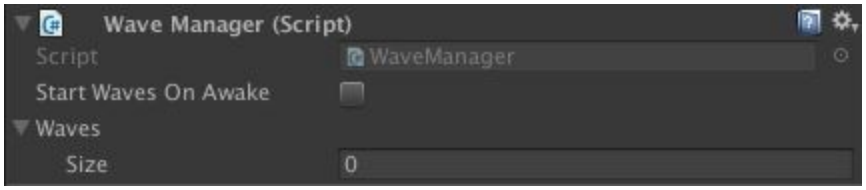


Setting Up Waves



For enemies to spawn, we'll need to set up waves for them to appear in.

- Create an empty GameObject and name it Wave Manager
- Add a WaveManager component to to the Wave Manager object
- Set the Size field to the number of waves we plan on including.



In order to create waves:

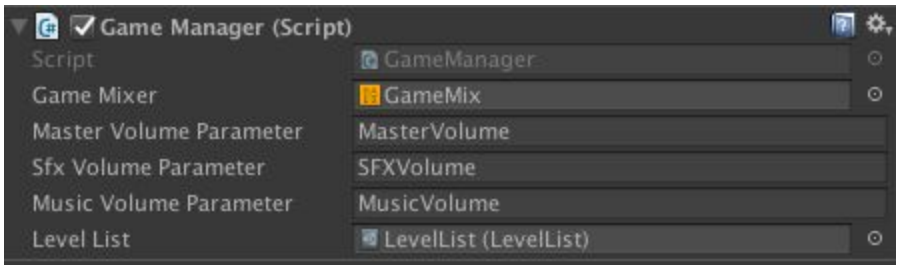
- Create an empty GameObject and name it Wave1 (replace the number as necessary)
- Add a Wave component to Wave1
- Set the Size field under Spawn Instructions to the number of Agents that should appear in the wave
- For each Agent:

    - Drag the AgentConfiguration ScriptableObject for the type of enemy into the Agent Configuration field
    - Set the Delay To Spawn field to how many seconds should pass after spawning the previous agent before spawning the current one
    - Drag the node that the Agent should spawn at into the Starting Node field

The regular Wave component will only complete the wave once all Agents are destroyed. We can also use a TimedWave component, which provides a Time To Next Wave field, where we can set a fixed time before Agents from the next wave start spawning.

After the waves have been created, add them to elements under Waves in the WaveManager.

Adding a Game Manager



A stage requires a GameObject with a GameManager component in order to end the stage when the player has destroyed all enemies or had their base health depleted.

- Drag the default Starter Kit GameManager prefab into the Hierarchy view from Prefabs/Managers.
- Drag the LevelList ScriptableObject into the Level List field

Understanding the GameManager component

Game Mixer

This contains a reference to the audio mixer that the game uses.

Master Volume Parameter

The name of the parameter in GameMixer that controls the overall volume of the game.

Sfx Volume Parameter

The name of the parameter in GameMixer that controls the volume of the sound effects in the game.

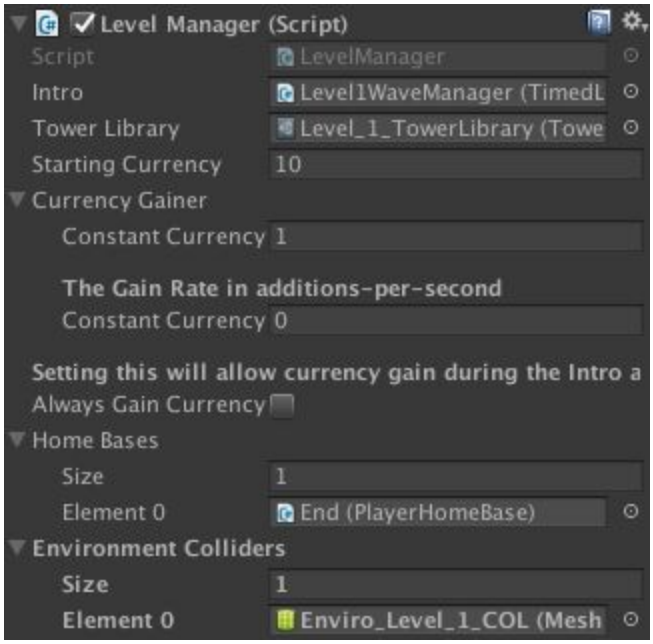
Music Volume Parameter

The name of the parameter in GameMixer that controls the volume of the music in the game.

Level List

Here we can set a reference to the list of stages that are to be included in our game.

Adding a Level Manager



The LevelManager manages the state of the game, and is where we can set up some other important features of our stage, such as how much currency the player is given at the start. To add one:

- Add a LevelManager component to the Wave Manager GameObject made earlier
- Set the Starting Currency field
- Drag the TowerLibrary ScriptableObject to the Tower Library field

Understanding the LevelManager component

Intro

The LevelManager can optionally enter a special state before the game starts. For example, this could be used to display an establishing cutscene for the stage.

The Starter Kit includes TimedLevelIntro, a very simple timed implementation of the intro that waits for a period of time before beginning the game.

To implement custom behaviour, create a new script that extends from Intro and perform our logic in `Start`. For example, we might launch a cutscene created using Timeline. Once our intro is complete, we can call `SafelyCallIntroCompleted` to instruct the LevelManager to continue.

Tower Library

Here we can assign the TowerLibrary scriptable object we made earlier to the stage. Doing this will set the available towers for the stage.

Starting Currency

The Starting Currency field defines the amount of in-game currency the player starts with to spend on towers.

Currency Gainer

We may want the player to be able to earn currency over time while playing the game. The fields in the Currency Gainer are where we can set this up.

Constant Currency Gain Addition

Constant Currency Gain Addition is how much currency will be added to the player's total each time an addition is made.

Constant Currency Gain Rate

Constant Currency Gain Rate is how many times per second the currency addition occurs.

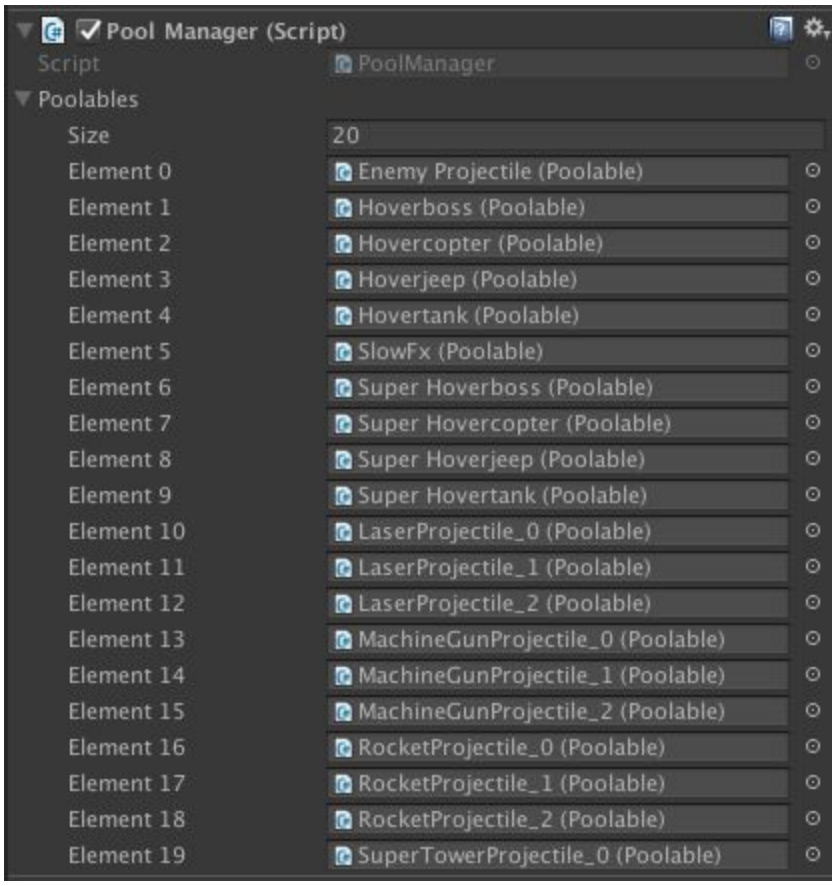
Home Bases

This field defines the player home base. Enemies will reduce the player's health if they reach this point. This needs to be set to a reference to the final navigation node, which has a PlayerHomeBase component on it. More on this below.

Environment Colliders

Any colliders set here will be ignored by ballistic projectiles briefly. This can be useful to ensure that they do not collide with small parts of the environment between a Tower and an Agent and explode without hitting an enemy.

Pool Manager



Constantly creating and destroying instances of enemies, projectiles, and passive effects is an expensive operation. Instead of doing this, we can use the Starter Kit's pool manager. The pool manager is a class that will create and store duplicates of objects so that they can be reused.

- Add a Poolable component to the Projectiles, Particle Systems, and Agents
- Create an empty GameObject and name it Pool Manager
- Add a PoolManager component to the Pool Manager GameObject
- Drag Prefabs for Projectiles, Particle Systems, and Enemies into the Poolables fields if they will appear in the stage
- Ensure that the Transform scale of the Pool Manager is set to (1, 1, 1)

Note that spawned enemies are made children of the GameObject with the PoolManager, which means they will inherit its transform values. It is likely that we will want the scale to remain (1, 1, 1), but this can be used should we wish to increase the scale of all enemies.

Camera

Introduction

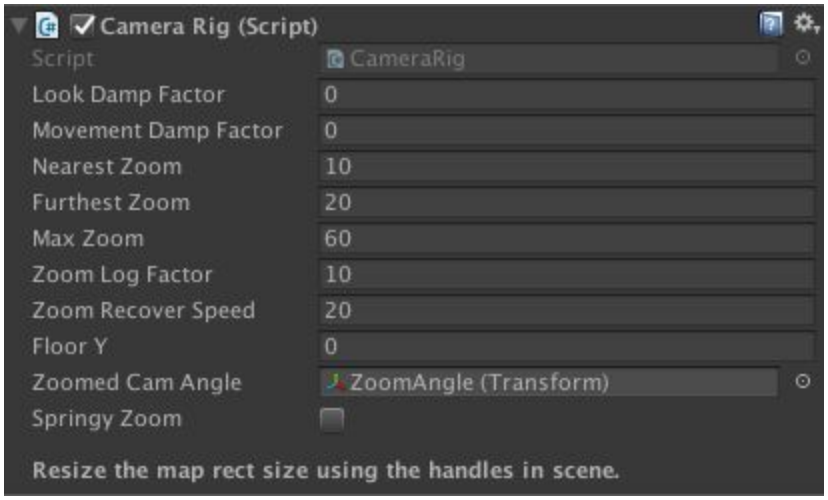
In this section we will look at the CameraRig component that controls the way in which the camera moves. In later sections we will focus on the UI and input components attached to the GameCamera object.

Adding the Starter Kit camera

The Unity Tower defense Template comes with a predefined camera already set up to use in Tower Defense games.

- Delete the default camera from the scene Hierarchy window
- Drag the GameCamera prefab from Prefabs/Player/ into the Hierarchy

Understanding the CameraRig component



The GameCamera GameObject has a component called CameraRig, which provides us with a number of ways to adjust how the camera works:

Look and Movement Damp Factors

These two fields control how springy the camera movement will be. The higher this is, the smoother the camera motion will be.

Nearest Zoom, Furthest Zoom, Max Zoom

These fields control the zoom extents of the camera.

Max Zoom is only relevant if Springy Zoom is checked on below. It controls how much further than the Furthest Zoom the player is able to zoom out before zoom is clamped entirely.

Zoom Log Factor

Only relevant if Springy Zoom is checked on below. Determines how “strong” the rubber band effect is.

Zoom Recover Factor

Only relevant if Springy Zoom is checked on below. Determines how quickly the zoom returns to its normal maximum extents after the player releases.

Floor Y

This camera assumes the ground is at a fixed Y level, and this value sets where that plane is in the world.

Zoomed Cam Angle

A reference to a transform that has the camera’s angle when fully zoomed in.

Springy Zoom

Whether zoom rubber bands at the near and far zoom points.

Pan bounds

When the CameraRig is selected, a rectangular area will appear in the scene view with draggable handles at its edges. The camera cannot move outside of this rectangle. We can use the handles to make this area bigger or smaller.

User Interface

Introduction

We will want to make certain pieces of information about the current game state available to the player. This includes how much health their base has remaining and how much currency they have to spend on towers.

This section covers how to make sure that information is available to the players, primarily through the use of a UI Canvas and the GameUI component.

UI Canvas

For the UI, create a canvas by right clicking in the Hierarchy view and selecting **UI > Canvas**. Next, navigate to Prefabs/UI in the Project window and attach the following prefabs as children of the canvas object:

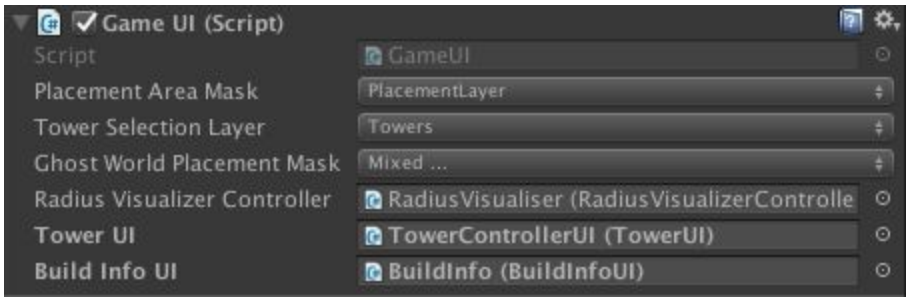
- TowerControllerUI - This will open a popup when the player clicks the tower, which will give them information about the tower and give them the ability to sell or upgrade the tower.
- Build Menu - This gives the players a selection of towers to build.
- PlayerBaseLife - A UI element that displays the Home Base's current health and how much currency is available to the player.
- WaveContainer - Displays the number of waves in the current stage.
- Game Over - When the player destroys all the enemies or has their Home Base's health reduced to zero, the endgame popup will appear.
- PauseMenu - Allows the user to pause the game and restart/exit the stage.

- TowerPlaceConfirmationUI - Allows the player to confirm tower placement on mobile devices.
- TowerPlaceInvalid - Indicates to players using mobile devices that the location they have placed a tower is not a valid placement area.

The GameUI component on the GameCamera object (added in the previous section) will need references to some of these Prefabs:

- Drag the RadiusVisualiser Prefab to the Radius Visualizer Controller field
- Drag the TowerControllerUI Prefab to the Tower UI field
- Drag the BuildInfo Prefab to the Build Info UI field

Understanding the GameUI component



We will need to attach a GameUI component to the GameCamera GameObject. The GameUI component defines UI components that are shown when the player selects, buys, or sells a tower.

Placement Area Mask

This allows us to select the layer that Tower Placement components are on. This allows the game to filter out other layers when placing a tower.

Tower Selection Layer

This is the layer that towers that are already placed exist on. This allows us to filter out other layers when selecting a tower on the grid.

Ghost World Placement Mask

This defines which layers tower ghosts can collide with. When moving tower ghosts around, we want to be able to see the ghost on certain stage colliders as well as on the placement grids themselves.

Radius Visualiser Controller

This holds a reference to a visualiser that displays a tower's radius when selecting/placing it.

TowerUI

This holds a reference to the TowerUI that displays a tower's level, DPS, and the buttons to upgrade or sell the tower.

Build Info UI

This references the sliding panel that appears when selecting a tower from the build menu.

Input

Introduction

In previous sections we focused on setting up the game camera and the UI. Now we will need to make it possible for users to move the camera and interact with the user interface using both keyboard/mouse and touch input systems.

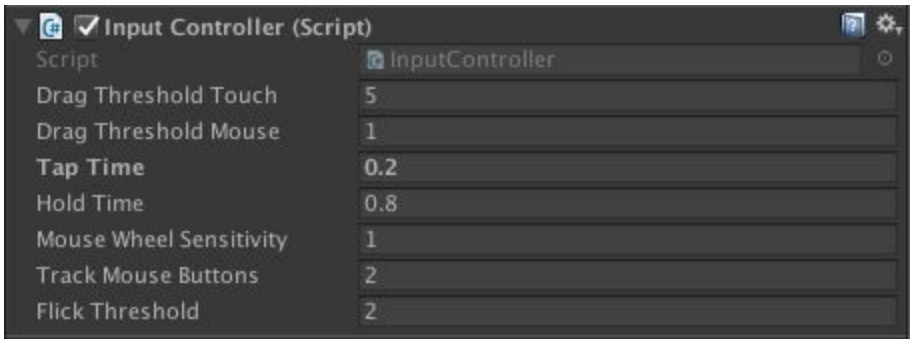
Implementing desktop and mobile control schemes

It is likely that some stages will be too large for a single screen, and players will want to pan the camera to view other parts of those stages. The Starter Kit comes with components that make this possible using both keyboard and mouse controls, and mobile touch controls. To add the desktop controls, take the following steps:

- Delete the default camera from the scene Hierarchy window
- Drag the GameCamera Prefab from Prefabs/Player/ into the Hierarchy

The premade Starter Kit GameCamera Prefab has components that enable multiple different input schemes. They are discussed below.

Understanding the InputController component



The InputController component determines which input system is active. It also fires off events made on input received in any given frame. The GameCamera object has this component attached to it. We will go through its fields below:

Drag Threshold Mouse

This will determine how many pixels must be dragged before the camera pans when using mouse controls.

Drag Threshold Touch

This will determine how many pixels must be dragged before the camera pans when using touch controls.

Tap Time

TapTime sets the maximum time for input to be registered as a tap.

Hold Time

HoldTime sets the minimum time for input to be registered as a hold.

Mouse Wheel Sensitivity

This sets how fast the camera will zoom when the mouse wheel scrolls.

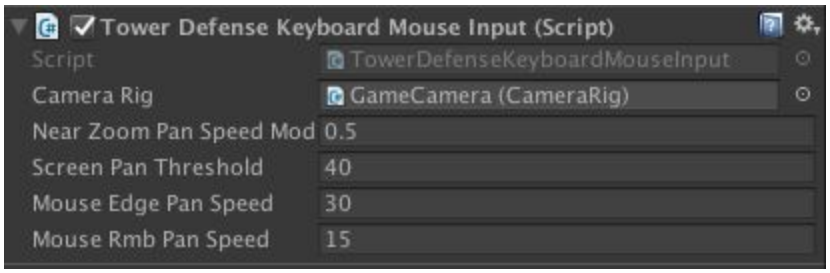
Track Mouse Buttons

This determines how many mouse buttons to track.

Flick Threshold

This defines how far the mouse or touch input needs to move to register a “flick” when panning the camera.

Understanding the TowerDefenseKeyboardMouseInput component



The KeyboardMouseUI component provides functionality for panning the camera using the mouse and keyboard.

Near Zoom Pan Speed Mod

This field controls the keyboard pan speed difference between being fully zoomed in versus fully zoomed out.

Screen Pan Threshold

ScreenPanThreshold sets the distance the mouse must be from the screen edge before the camera starts to pan.

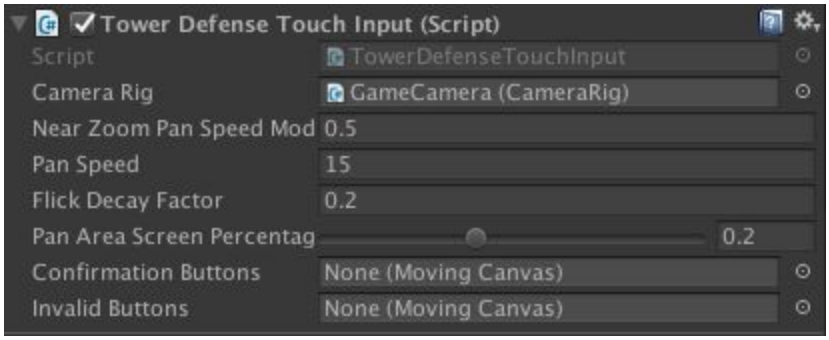
Mouse Edge Pan Speed

MouseEdgePanSpeed defines how fast the camera pans when scrolling at the edge of the screen.

Mouse Rmb Pan Speed

MouseRMBPanSpeed defines how fast the camera pans when moving the mouse with the right mouse button held.

Understanding the TowerDefenseTouchInput component



The TouchInputUI component provides functionality for moving the camera using mobile touch input and interacting with the UI for the mobile version of the Template.

Near Zoom Pan Speed Mod

This field controls the keyboard pan speed difference between being fully zoomed in versus fully zoomed out.

Pan Area Screen Percentage

This defines how much of the area around the middle of the screen can be used to pan. When the user is dragging a tower within this area of the screen, the camera will pan.

Pan Speed

This defines how fast the camera moves while panning using touch input.

Flick Decay Factor

How long it takes for the camera to stop moving after a flick.

Confirmation Buttons

This contains a reference to the UI prefab that we want to show players when they have dragged a tower visualizer to a valid placement location.

Invalid Buttons

This contains a reference to the UI prefab that we want to show players when they have dragged a tower visualizer to an invalid placement location.