Beat Em' Up Engine for Unity Read Me

By: GameStar18

Introduction

Thanks for your interest in my beat em' up engine! This engine has been remade and improved much more than the previous one I made. With this engine you will be able to get an understanding of how to put together a beat em' up style game using setups that I have put together. I also have a nice and easy to use menu system setup that shows one way to setup a menu using the new UI system that came with Unity 4.6. It allows for a good amount of customizability. The scripts are heavily commented explaining bits of code. Each script also has a summary at the top explaining its overall purpose.

It would be cool to see what you make with this engine! I plan to use it, or parts of it for many projects to come.

There are words used that have different colors to signal what they are, Unity related. The colors are shown below.

Color Coding

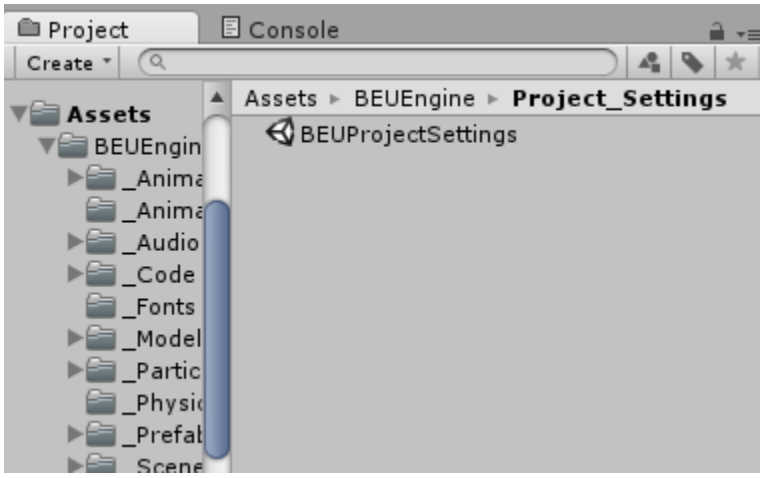
- *Color* – script names.
- *Color* – methods and coroutines.
- *Color* – variable or parameter names.
- *Color* – game object names.
- *Color* – animation state or animation names.
- ***Color*** – bold for other things like component names, scene names, etc.

Table of Contents

I. Introduction 1
   a. Color Coding 1
II. Importing the Project Settings 3
III. How the Project's Game Works 3
IV. Project Overview 4
   a. Level Layout 4
      i. Battle Zones and the End Boundary. 4
         1. End Boundary 6
      ii. Item Spawners and Item Containers 7
      iii. Boundaries and Waypoints 8
      iv. Nav Mesh and Terrain Things 9
   b. Character Setup. 11
      i. Character Prefab 11
         1. Attacking and Hitboxes 12
         2. Item Grabbing Mounts 14
         3. Dead Prefab 15
         4. Animator Setup 16
         5. Edited Animations 19
         6. Character AI 20
         7. Character Voice Animation Events . . 21
      ii. Input 22
   c. Items 23
      i. Collectables 23
      ii. Throwables 24
      iii. Weapons 24
   d. Scenes 25
   e. Mobile Input 26
V. Update Log 27
VI. Where to go From Here 28
VII. Questions/Comments? 28
VIII. Credits 28

Importing the Project Settings

After importing the project into a new Unity project, locate the Project_Settings folder. Inside of here you will find a Unity package file which contains the project settings for this project. Simply double-click it to open it up and then import it. You will then be set! After importing it, you can then delete this Project_Settings folder if desired.



How the Project's Game Works

Here I just simply describe what the game is like in the project for if you were to hit the play button and just play through.

- ***Scenes Used***:
  - **Scene_MainMenu** – choose characters, number of players, number of human players, difficulty and item appear rate before starting the game, or simply quit. The number of human players determines if you will have an AI controlled companion joining you. Say if you choose 2 for number of players and only 1 human player, then the other character will be AI controlled. If you want an easier game, other than making the difficulty set to easy, you can choose to have the item appear rate to its highest setting so you'll get the most amount of items.
  - **Scene_Canyon_0** – The first area you start in. Complete the battles here to move onto the next area after running off the far right side…
  - **Scene_Canyon_1** – The next area from the previous scene which has the end stage sequence. After the characters have reached the Boundary_End_Win game object's trigger, it will signal the end of the stage. This means that the results will appear showing how the player(s) did. After that, the characters will run off to the far right side of the scene to a waypoint gameObject far off to the right called Waypoint_End while the screen fades out and the game restarts.
  - **Scene_Demo** – This is simply the demo level used in my demo created for this project. The level has a simple menu allowing you to choose the number of players, humans, and difficulty.
- ***Item and Enemy Spawners***:
  - These are setup in a way so that for items, you will be able to get some in battle areas and in-between them. The enemy spawners are setup so that you will not be able to see the enemies spawn. The enemies run into the area where the player(s) are. You will not see items spawn either. Right now for items I only have crates spawn, large or small.

Project Overview

In this huge section, I will go over the project itself, in detail.

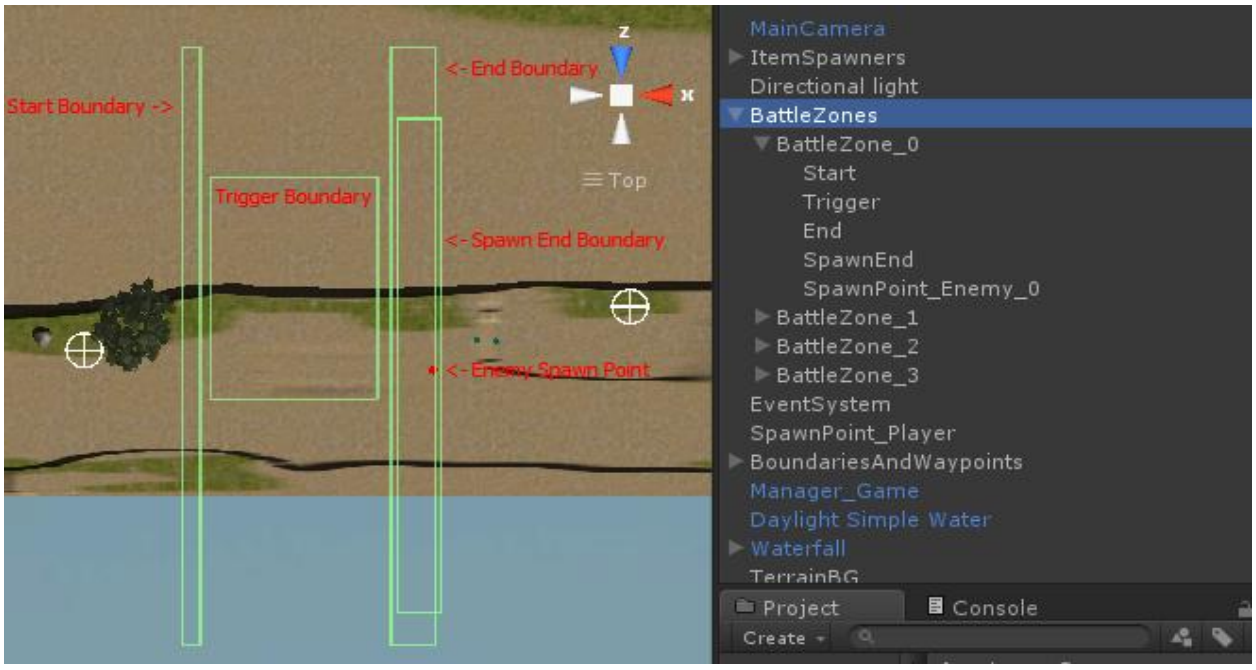
Level Layout

Quick Note: Do not start a level without the Canvas_Overlay gameObject in the hierarchy! An error will occur. Your starting scene will be the only one that needs it since it doesn't get destroyed and will carry over into scenes during the game. So don't start the canyon scenes normally. Always start the Main Menu scene first, or the demo one since that has the Canvas_Overlay as well.

I will start by going over how the battle zones are setup which explains the main layout of the level scenes…

Battle Zones and the End Boundary

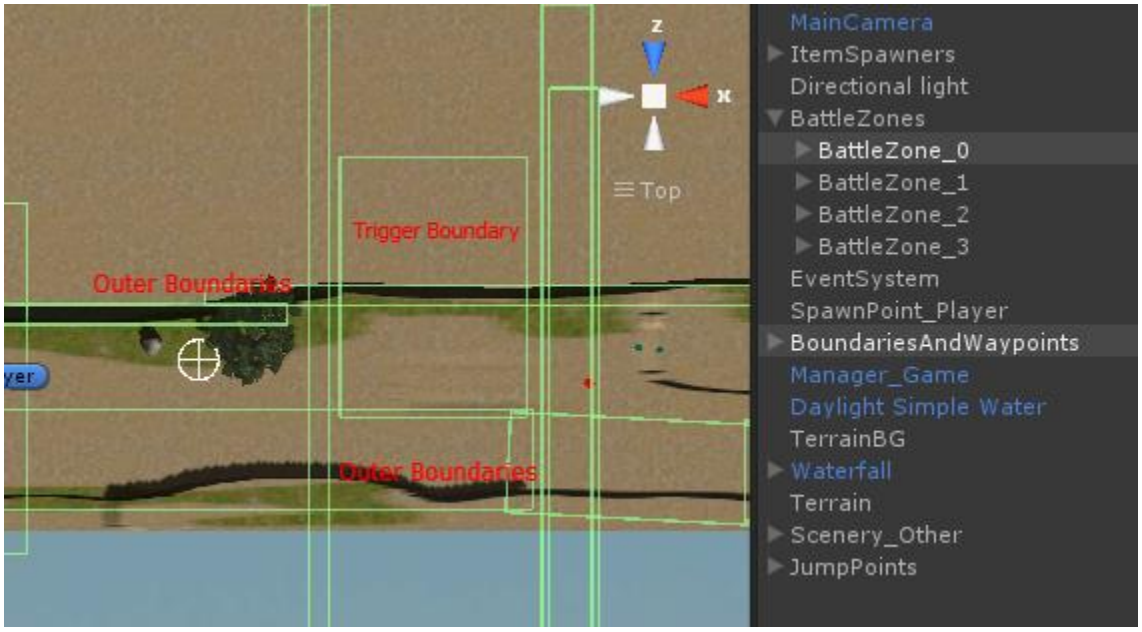
The battle zones are setup which explains the main layout of the level scenes. I would first suggest looking at the starting scene: "**scene_Canyon_0**" located in the "_Scenes" folder. Open that up and take a look at how the level is laid out. I have colored wired spheres for certain objects of interest such as the item spawners, enemy spawn points, and a few others. The Battle zones are shown below. Looking at it from above:



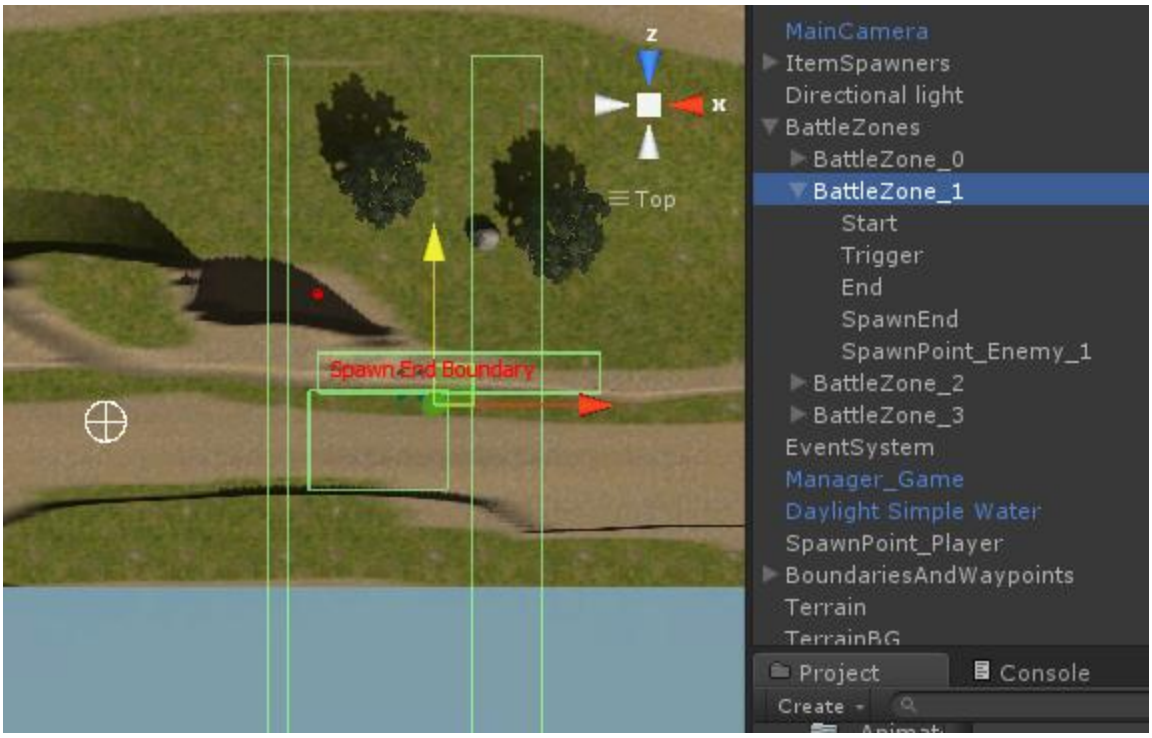
I have each battle zone laid out with a number at the end representing their current area number. This is important since I not only use it for organization, but the Battle Zone Manager (Manager_BattleZone) will organize each in numerical order starting with that parent gameObject (BattleZone_0, BattleZone_1, etc) for its children (explaining those next). Check the Manager_BattleZone script for more info on that. Each battle zone has a bunch of boundary gameObjects - Start boundary, Trigger, End, enemy spawn end boundary (SpawnEnd), and an enemy spawn point. I will explain them in detail below:

- The Start boundary represents the area that will be blocked off when the battle starts in that area.

- The Trigger is what initiates the battle after all players are inside of it, thus activating the Start boundary which is not active until after the battle starts in its area and then remains active for the rest of the scene to prevent you from going back. The Camera bases its min x position off of the Start boundary's collider's max bounds x and its max x position off of the End boundary's collider's min bounds x. Check the camera script Camera_BEU in its SetupRanges method for more information on that. The camera bases its z (forward and backward) position based off of the Trigger collider's min bounds z. With that in mind, how far down it goes is important, since the camera will be at a different distance when in battle. Also, you should make sure the trigger takes up the whole area between the outer boundaries of the level so that the player(s) will always go into the trigger area and not be able to move around it. In the below example I show the outer boundaries and the battle zone together so that you can see what I mean by making sure the trigger takes up the space that the player(s) can navigate so that they won't be able to move around it.



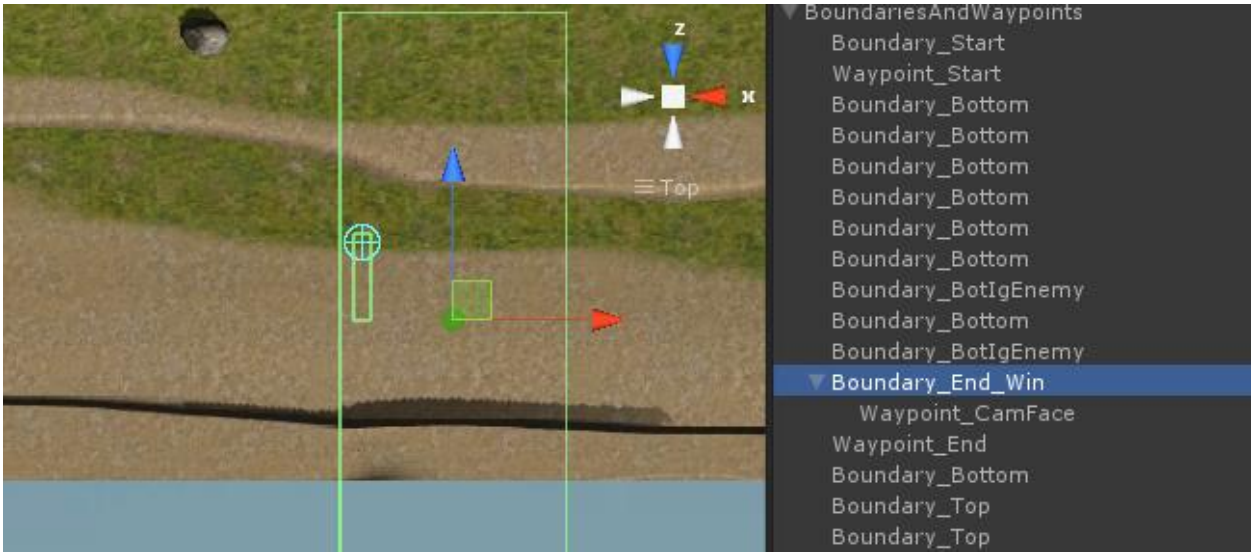
- The End boundary indicates the far edge of the battle area and is usually where the enemies spawn from since you won't be able to see them spawn off screen. The Camera's position gets clamped in between the Start and End boundaries with an offset that you can change called xPosClampOffset in Camera_BEU. The players can't go past these during a battle but the enemies can since the enemies layer called **Enemy** does not collide with the boundary's layer, **IgnoreEnemy**.
- The SpawnEnd indicates when the enemy will end being in its "Spawning" state and start targeting players as soon as it leaves that boundary's collider. This is done on the OnTriggerExit message so it will only happen when they leave the collider, not enter. I suggest making these boundaries end slightly before the End boundary starts so that they will be done spawning before the players can get to them.
- The SpawnPoint_Enemy gameObjects are the tiny red wired circles you see in the image and are indeed where the enemies spawn from. Make sure the blue arrow (forward direction) is facing where you want the enemies to move to after being spawned since the enemies will be created with the rotation of the spawn point they are spawned from. Just rotate the enemy spawn point so it faces the battle/trigger area. There are two different ways enemies can spawn: a normal walk using the rotation of the spawn point, or where they use their **NavMeshAgent** to run in from anywhere, like shown below:



*In the above image, I have a part where the enemies will spawn behind a wall and run into the area simply by navigating using their **NavMeshAgent**. As soon as the enemies run through and exit that SpawnEnd boundary, they will not be "Spawning" anymore and will start targeting players. That goes for SpawnEnd boundaries in general, not just the above example.*

Check the Manager_BattleZone script for additional information on how battles work.

End Boundary



When "Win" is in the name like in the image above, that indicates that it is the final scene of the stage and will result in the players going into their win animation as well as the results being displayed. Otherwise they will just run off the screen and the next scene will load shortly after.

I have a Waypoint_CamFace child gameObject in there. It is the light blue wired sphere you see in the image in the scene view. The wired sphere area is where the main position of it is and that is where player one will run to. Player two will run slightly in front of its forward direction from that spot and any player after two will run a little further in front than that. With more players you'll want to make the small green trigger box below it slightly longer. When the players collide with that small box trigger on it, they will stop running and start to face the camera. Also a side note, the camera will stop moving relative to the bounds of the Boundary_End / Boundary_End_Win gameObject so be sure to make the size of it accordingly. What I have now works well so you can still see the characters after they run into that waypoint trigger. Waypoint_End is the waypoint gameObject that the player(s) will run to after the results have finished displaying.

Items Spawners and Item Containers

I use a white sphere to visually represent these. The sphere radius is the range of the max distance of how far an item can be spawned from this point. If an item is created in an outer boundary, I have code setup for it to be placed at the edge of the boundary it is inside closest to the play area so that wouldn't be an issue.

With item spawners, you will be able to choose what items will be created by the item storer/container gameObject that gets created from these. You can choose to have items randomly created taking a chosen amount from all items available, or you can specify which items you want the object created to carry. I have large and small crates that get created. The small ones will only have half as many items as the large ones. If you choose manual items for the item spawner's crates/item container/storer gameObject, then random ones from the given list will be removed. Check the ItemStorer and ItemSpawner scripts for more information.

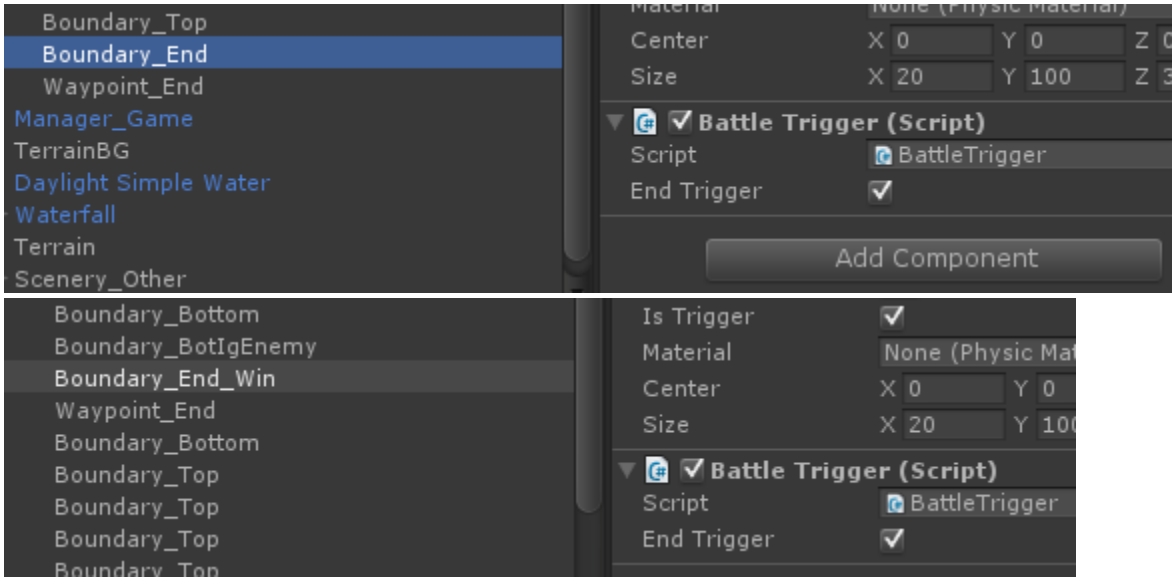
Also, based on the ItemAppearRate of Manager_Game chosen, small and/or large crates will be created containing fewer or more items inside. For example, if the ItemAppearRate is very low, only one or two small crates will be created. However on very large, you will always get at least one large crate, plus two small or another large crate.

Boundaries and Waypoints:

If you look in the BoundariesAndWaypoints gameObject you will find the outer boundaries as well as waypoints for the start and end of the level. I will explain specific objects below:

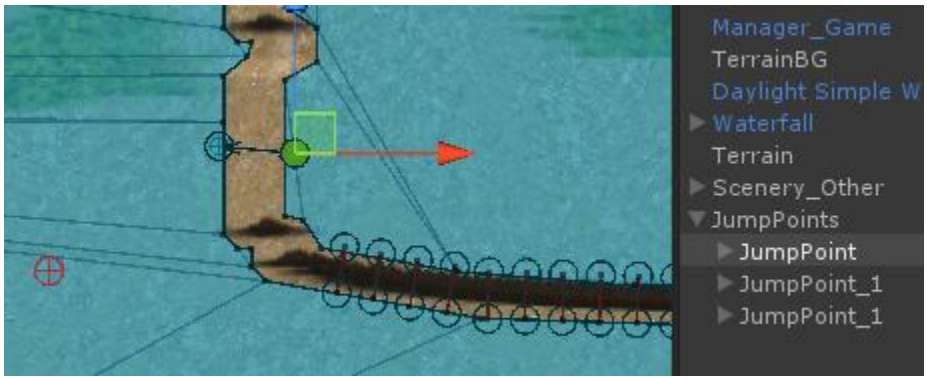
- Boundary_Start is the gameObject that refers to the very beginning of the scene. You can't go past it so make sure the SpawnPoint_Player (spawns the players) gameObject is not inside of it. Boundary_Start marks a clamp for the camera so it will stop before getting too close to it. Same goes for Boundary_End.
- Waypoint_Start is the waypoint the player(s) will run to at the start of the scene. Once they touch it, you will be able to control them.
- Boundary_Bottom and Boundary_Top are just simply the names I gave to all of the boundaries located on the corresponding part of the play area. The name is important as the ItemStorer script checks if it is colliding with one of these and if so, moves to the corresponding spot at the edge of it inside the play area. That script checks the name and sees if "Bot" or "Top" is in the name and moves to the correct border based on that. Check that script's OnCollisionEnter() method for more info on how it's done.
- Boundary_BotIgEnemy is just another Boundary_Bottom with the only difference being, enemies won't collide with it allowing them to pass through. It is just to block players. There are a few areas I didn't want to have a large Boundary_Bottom collider for. I used Boundary_BotIgEnemy for areas where enemies spawn from the bottom of the screen. These boundaries have a layer called **IgnoreEnemy** which ignores collision with the "Enemy" layer, which indeed is the layer the enemies are on. Check Edit -> Project Settings -> Physics to see the layer listing of which objects collide with which layers.
- Boundary_End and Boundary_End_Win are triggers that signal the end of the scene when all players have entered it. You just need to check the End Trigger checkbox on its script BattleTrigger for the end stage one. It uses the same script for the battle zone triggers, simply since that is useful for waiting until all players are in the trigger area. If all of the players are not inside of an End Trigger one, all players who have entered will stop and wait for the other remaining players to enter before they all proceed and run off. If "Win" is in the name, that signals to the script that it is end of the stage, so it alerts Manager_Cutscene to start the Stage Clear cutscene, thus the players will go into a win animation (I just

used a different idle for my win animation), face the camera and then run off screen after the results are done displaying and you press the "Continue" button. I just have the game restart after this. If "Win" is not in the name, then the players will simply just run off the screen and the next scene will load up.

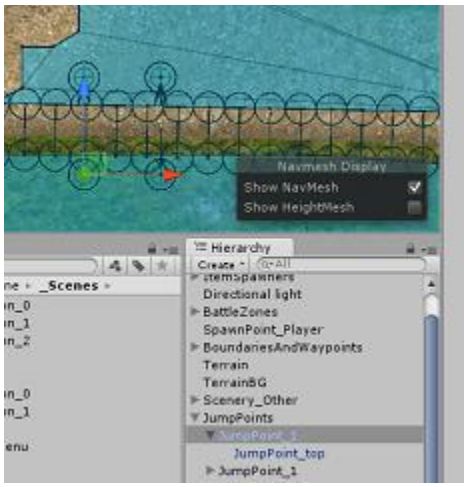


- Waypoint_End is simply where the players will run to after all passing through Boundary_End and completing the scene basically. The only thing you need to make sure of is to place the position/pivot part of it at about ground level and have it in a good place for the players to run to when running out.

Nav Mesh and Terrain things



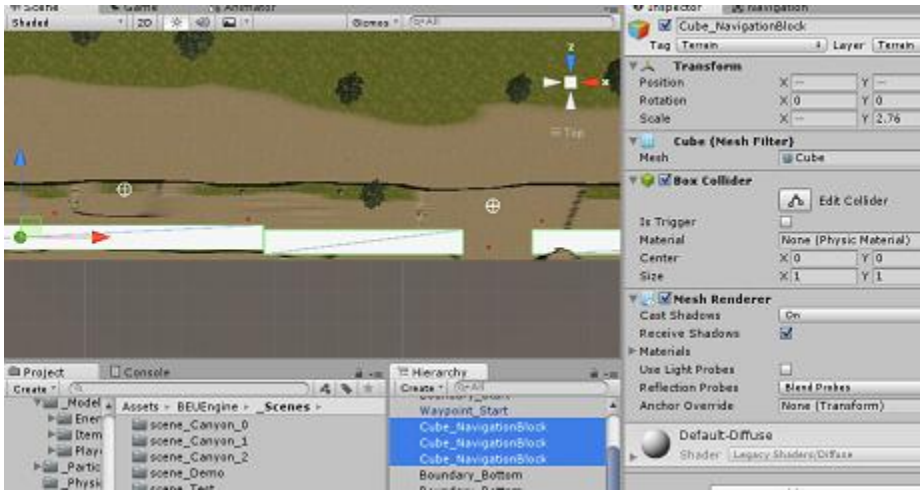
I created a few jump points for use with the NavMesh simply for areas where auto-generated off-mesh links weren't created so that the AI can make it over. These jump points are all inside of a parent gameObject called JumpPoints.

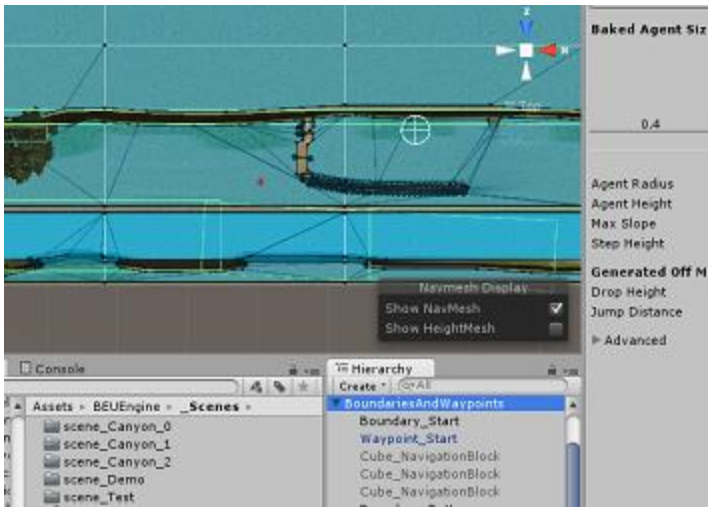


*Each JumpPoint gameObject has a child called JumpPoint_top which is the top position used. These two gameObjects are connected to one another in the **OffMeshLink** component.*

Lastly, not important enough for its own section but, Scenery_Other is simply used for other scenery not used with the terrain. I gave those static for being used with Occlusion Culling so they won't be rendered when not visible by the camera. Everything used by the terrain already takes care of things like that.

Another thing to mention are the Cube_NavigationBlock gameObjects which are not active by default. I activated them here to show them. I used these to change where the nav mesh gets created when baking it since I don't want it to create it all over the terrain since the AI will try to go to places they shouldn't, such as outside certain boundaries. These cube blocks are used to create a gap that the AI has no way to cross.





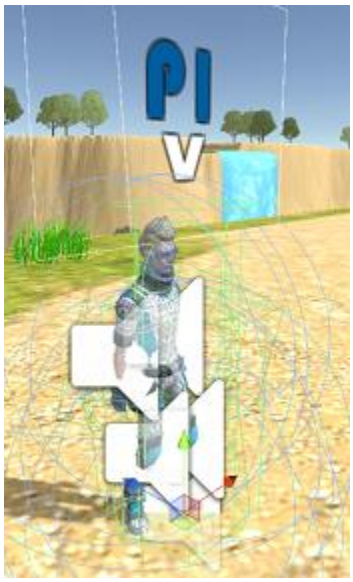
Here you can see the gaps that the AI will not be able to cross since I don't allow a path across with a jump point. This way, the AI will not try to move outside the colliders (green lines).

If you create your own environment, you can of course detach areas where you don't want nav mesh to be created. Since I used Unity's terrain for this project, this is one way to prevent the AI from trying to go to other areas.

Character Setup

Since the enemies and the players are setup in the same way, I can explain them both in this section.

Character Prefab



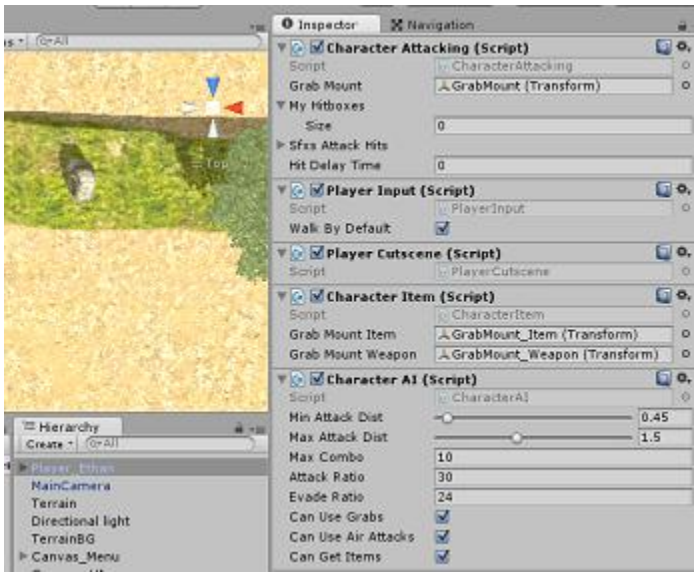
The main player selected which shows all of the things he uses and has attached.

The player prefab is the most complex prefab in the entire project. Here is where I will describe the key things on it in more detail.

Attacking and Hitboxes

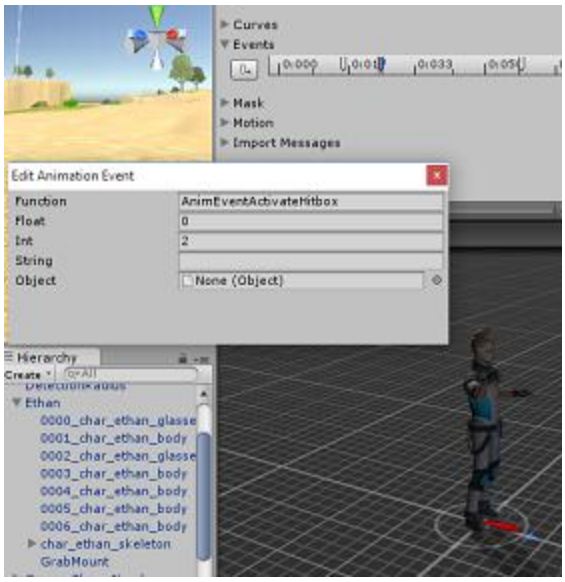
The right forearm selected on the player.

The players and enemies hitbox is simply a body part with an Is Kinematic rigidbody, a collider, and a HitboxProperties script attached. Each attack animation uses an animation event which activates a chosen hitbox in the CharacterAttacking script based on which animation is being used.



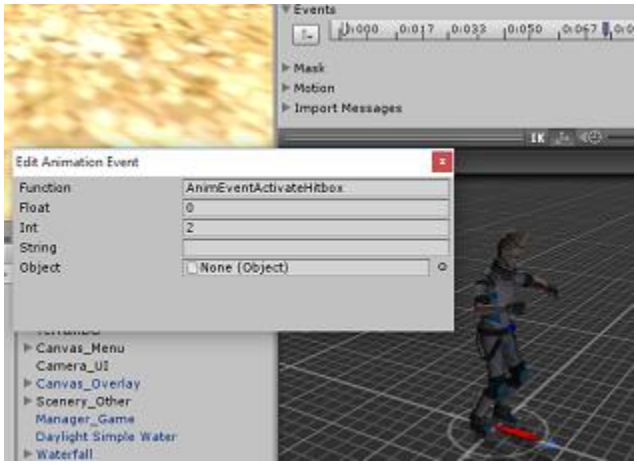
Things shown in the inspector on the player prefab

The CharacterAttacking script has a variable called _myHitboxes which will hold all of the HitboxProperties scripts on the character. Right now I simply have CharacterAttacking find all of them on the player in Start() by using **GetComponentsInChildren**<HitboxProperties>(). You can assign them in the inspector instead if you wish if you want them to be in a specific order. If you assign any in the inspector, you'll need to assign them all that way since the script checks if the list is empty, and if it is, it will find them all. The order they are in inside of the _myHitboxes list is important since you need to know which hitbox you're activating during an attack which is done in SetupHitboxStats of CharacterAttacking.



The first attack animation. Showing the animation event.

In the above screenshot, you can see the animation event used for attacks. It's called AnimEventActivateHitbox and the CharacterAttacking script has that method/message. The parameter for int determines what to do with the chosen hitbox where 0 = deactivate, 1 = activated but at weakest strength, 2 = activated and at full strength. In CharacterAttacking I have a method called SetupHitboxStats which checks the current state to see what hitbox to use on the character as well as what stats/properties to give it for the attack such as push force, strength, etc. SetupHitboxStats gets called during AnimEventActivateHitbox. Check CharacterAttacking for more information on attacks and hitboxes.

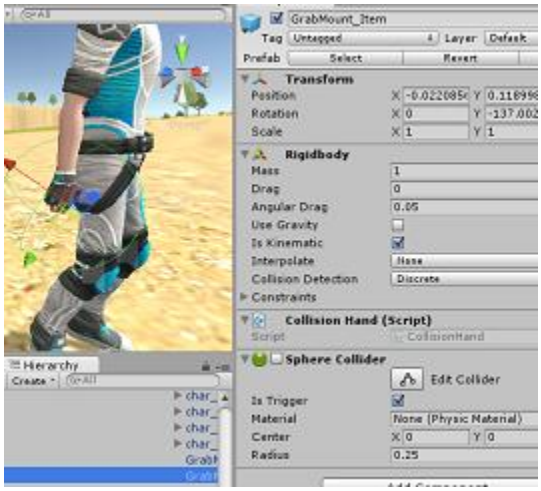


Grab animation showing the grab animation event.

Grabbing is used much the same way as it uses the same animation event and uses the same integer numbers to activate and deactivate it where 2 = activate and 0 = deactivate. When an opposing character is inside the trigger used for this, they will be grabbed as their animator parameter "HurtOther" will become 4. When it becomes 4, the character is inside the WasGrabbed() coroutine on CharacterStatus. Note when grabbed, the character has a trigger collider which is used so they can be hit while grabbed since their regular capsule one gets disabled to prevent unwanted collisions while grabbed. The trigger collider is on CharacterMotor and its called grabTriggerCol.

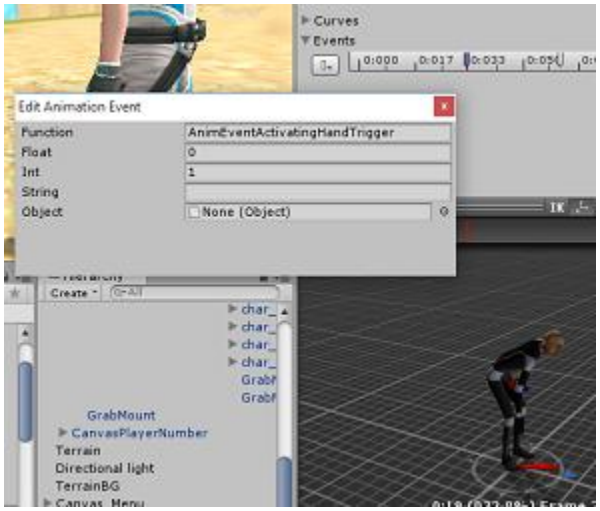
When you have a character grabbed, press your attack button when in the Grab_Idle state to attack, which gets called from PlayerInput, which when holding a character, calls the GrabAttackSetup method on CharacterAttacking which also prevents the grabbed character from escaping as seen in the WasGrabbed coroutine of CharacterStatus during its while(anim.GetInteger("HurtOther") > 3) loop. Holding the attack button when you have a character grabbed will make you throw them instead. Check the WasGrabbed, WasThrown, and GotHit coroutines on CharacterStatus as well as the area where the grabbing takes place on HitboxProperties (check for OnTriggerStay looking for 'if(_isGrabBox)" for even more information on grabbing characters.

Item Grabbing Mounts



The player's grab mount for throwable items selected.

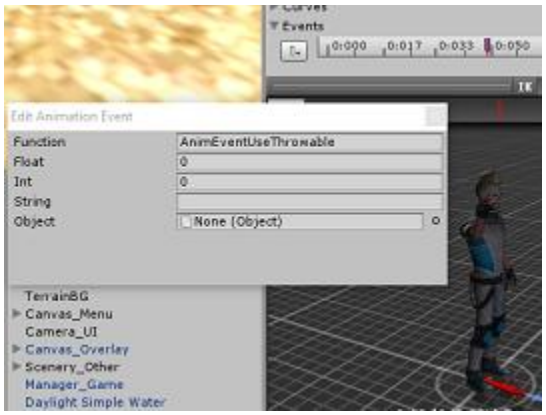
This is the mount used for grabbing throwable items. This item mount as well as the one for weapons use the script CollisionHand which gets activated during the pickup animation. The correct mount gets activated based on which item is being attempted to be picked up. CharacterItem is the script that takes care of this. There is an animation event used on the pickup animation as seen below:



Showing the animation event for grabbing items.

The animation event is called AnimEventActivatingHandTrigger and is used in CharacterItem. When the int value on this event is 0, it disables each item grab mount that is enabled. When it is 1, it enables the correct one based on which type of item is attempted to be grabbed. When the collider trigger is activated, the item mount for

throwables can then pick up throwable items only, as the weapon one can then pick up only weapons. After an item has been collected, it will become a child of the correct item mount if it is not a collectable item such as a healing item; those items will get used up right away. CollisionHand is the script that has the OnTriggerEnter event for the item being touched which sends the message to Base_Item on the item being touched. This holds the WasGrabbed method that then gets called on the item, resulting in it being "grabbed" by the character.

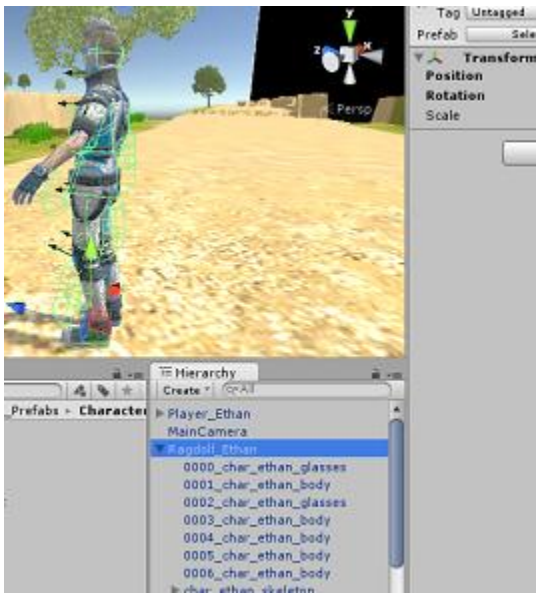


Animation event shown for throwing items.

Throwing items is done with the above animation event method called AnimEventUseThrowable which is used on the CharacterItem script. This event sends the message "WasThrown" to the item which makes it not be a child to the character's item mount anymore and gives it velocity.

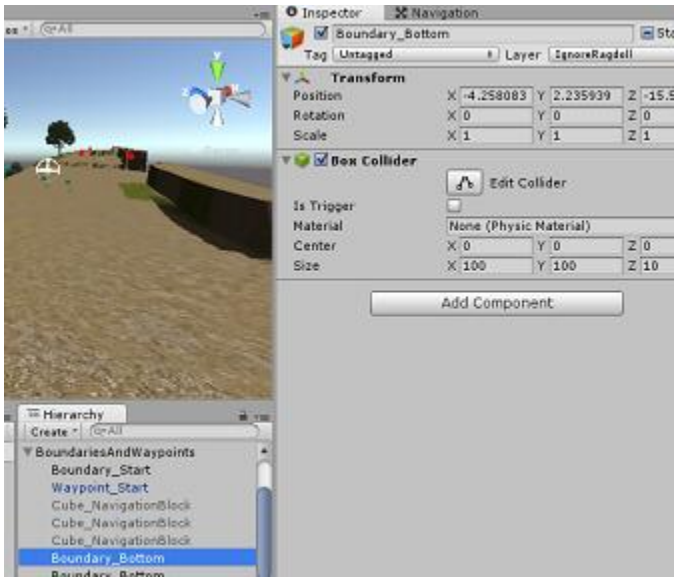
Dead Prefab

The characters have an option slot found on CharacterStatus for a prefab to use when the character dies. For this I created a ragdoll of each character model which gets instantiated upon dying. The first rigidbody source is found on its root child gameObject and is used for pushing the ragdoll in the direction that the character died in. If no ragdoll is assigned to the deadPrefab variable, then a dying animation will be used instead.



Ethan's ragdoll selected in the hierarchy.

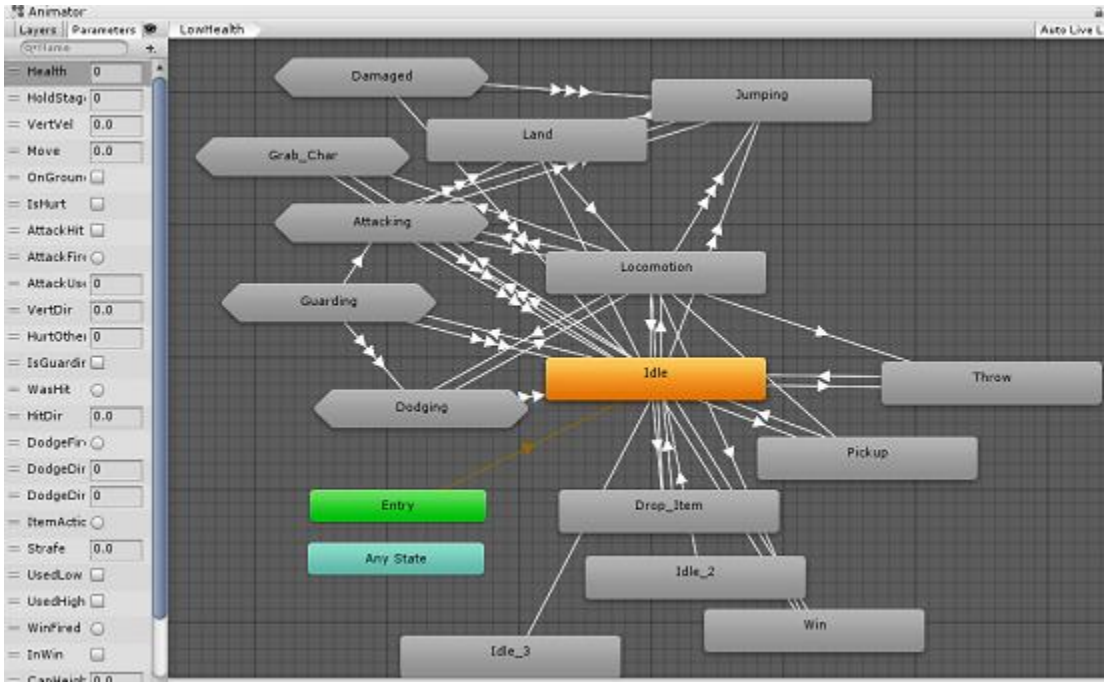
Check the Die coroutine of CharacterStatus for more information. Note that the ragdolls have their own layer called “Ragdoll” which I gave them so that I could make them not collide with boundaries by giving the boundaries a layer name called **IgnoreRagdoll** as shown in the image below. Thus, if a character dies and has a ragdoll assigned, it will allow them to fall out of the main area, or even off the level, haha fun stuff. I used that feature in my battle game found on my YouTube channel.



A boundary selected and has the “IgnoreRagdoll” layer.

My Animator Setup

The enemies and players use the same Animator controller:



A note on the sync layer "LowHealth": *I also use the "timing" checkbox on this layer which results in the animation always going at a steady pace.*

I will explain the sub-state machines and give a brief overview:

- **Damaged**: here is where all of the animations for being hit, grabbed, and thrown are stored.
- **Grab_Char**: here is where all of the animations for grabbing a character are stored, such as the grab attempt (Grab state name), Grab_Idle, hitting a character while they are in your grasp, losing them out of your grasp, and throwing the character that you have in your grasp.
- **Attacking**: all of the attacks are held here and I have it setup how I prefer combos to be aligned but, as with most things in this project, you can tweak it however you want. I will explain all of the Animator parameters in the next part.
- **Guarding**: All of the guard states such as starting to guard, Guarding (guard idle), and hit while guarding. Un-guarding is unnecessary since the transition back to idle from guarding does that nicely. You can dodge while guarding which goes to the next thing…
- **Dodging**: All of the different dodge moves you can do are here. I use four, one for each direction – left, right, up and down. The air evade one I simply just use the left or right one since I use a side flip animation which looks pretty good even while in the air.
- **Layers used**: I use a low health layer which uses wounded animations for idle and moving/Locomotion when the character's health is below 25% of their max. This layer is synced with the base layer so the only changes made are those two animation states (Idle and Locomotion). I activate this layer by setting its weight value to 1 right after the player health reaches 25% or lower. The Hold Item layer is used for when you are holding a throwable item. I use this so the player will hold out their right hand. The right arm is the only thing used for this layer for that purpose.

Other than those, the Jumping state is a blend tree with jumping and falling. You can randomly transition to different idles when in idle, but I do that in code (CharacterStatus script). The other animation states should be self-explanatory.

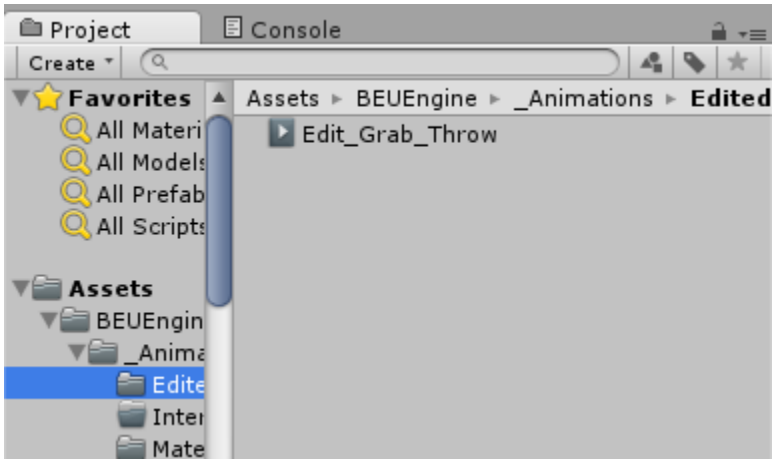
Now for the Animator parameter descriptions:

Key – int = integer, bo = Boolean, fl = float, tr = trigger

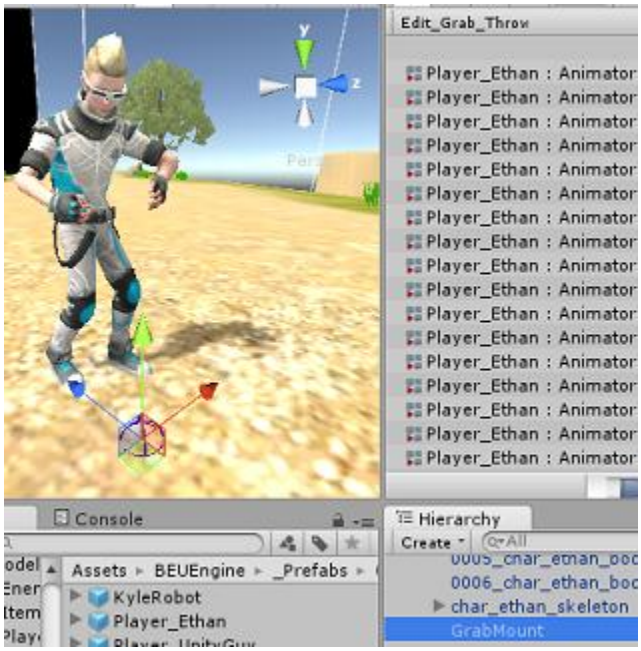
- **Health(int):** the character's current HP/health.
- **HoldStage(int):** determines if the character is holding an item. 0 = none, 1 = holding throwable, 2 = throwing a throwable, 3 = holding a weapon.
- **VertVel(fl):** the character's vertical velocity (y).
- **Move(fl):** how much the character is moving. 0 = idle, 1 = running. Negative values mean that you are walking backwards which can be done when holding your targeting button when you have an enemy targeted.
- **OnGround(bo):** is this character touching any ground?
- **IsHurt(bo):** if the character currently hurt and in hit stun. They won't recover until this gets set back to false.
- **AttackHit(bo):** did the character successfully connect an attack? Used to combo.
- **AttackFired(tr):** a trigger used for going into an attack state.

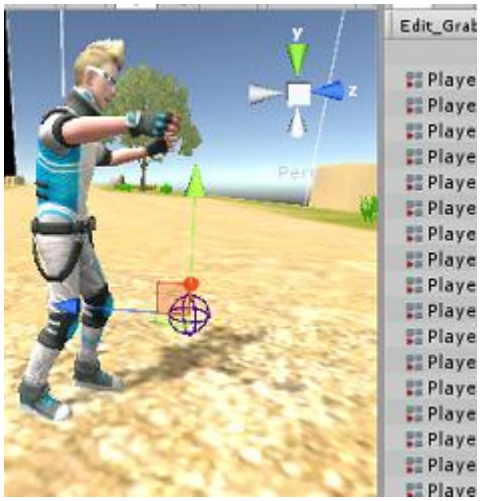
- **AttackUsed(int):** this is an integer value which indicates which attack was used, whether it is a normal(1) or heavy(2) attack. 0 = none.
- **VertDir(fl):** vertical direction for attack input. Used with **AttackUsed** to specify if a high or low attack was used.
- **HurtOther(int):** An integer value for the variety of ways to be hurt. 0 = regular hit, 1 = hit hard/more stun, 2 = knock down, 3 = dizzy, 4 = grabbed, 5 = thrown. This gets set back to 0 along with IsHurt at the end of being in hit stun before returning back to idle.
- **IsGuarding(bo):** just that, whether we are guarding or not. When this is true, we can enter our Guarding state.
- **WasHit(tr):** the trigger used to bring us into a hurt state.
- **HitDir(fl):** used to determine if we were hit from the front or from behind based on the character that attacked us so we know whether to do a hit forward or hit backward animation.
- **DodgeFired(tr):** a trigger attempt for dodging.
- **DodgeDirHor(int):** horizontal input for a dodge direction.
- **DodgeDirVert(int):** vertical input for a dodge direction.
- **ItemActionFired(tr):** a trigger attempt for picking up an item or throwing it.
- **Strafe(fl):** a float value for whether we are strafing left or right. You can strafe when holding your targeting button when targeting an enemy by pressing left or right. You will circle around the enemy in this way.
- **UsedLow(bo):** bool stating whether we have used our low attack in the current combo we are in or not. Resets to false when the combo ends. This is used so we can't continue to use this move repeatedly in the same combo.
- **UsedHigh(bo):** same as above only for our high attack.
- **WinFired(tr):** a trigger for going into our win animation at the end of a scene.
- **InWin(bo):** if we are currently still waiting for the end cut scene to finish after completing the stage. This will become false afterwards, it which the character will then run out of the scene.
- **CapHeight(fl):** It just gives more precise height for the character's capsule when they are knocked down and get back up. Check the ScaleCapsule method in the CharacterMotor script to see how it's done.
- **MinMovement(bo):** The minimum movement required to be considered going back into idle if in locomotion or going into locomotion if in idle. This gets updated in the Update() method of CharacterMotor.
- **IsGrabbing(bo):** If we currently have a character held in our grasp. When we do, and this becomes false, it means they have been released from our grab/grasp.

Edited Animations



In the end, I only ended up having one edited animation. I made this folder for character animations that needed some kind of edit. In this case, I edited my Grab_Throw animation. In Grab_Throw I edit the mount (GrabMount) where the character is held to move it where I want them to go when being thrown.





The above images show that the GrabMount gameObject (where the character is held) which is a child gameObject of the character, moves during the Edit_Grab_Throw animation which was done manually when editing this animation. I gave it a dark purple wired sphere so you can see it better.

Character AI

The enemies and the players use the same AI so I can describe them both here. The CharacterAI script handles the AI's behavior so I recommend checking that for more information. Here I will just mention how they behave and a few key things to know about them since the CharacterAI script handles the rest. The DetectionRadius holds all of the lists of things the AI can use to search for, such as items and characters; so that would be another good script to look at, since it allows the AI to know what's around them. I also have a bool for AI called canGetItems from CharacterAI which determines if the AI is even able to get items, and if not, don't bother pursuing them.

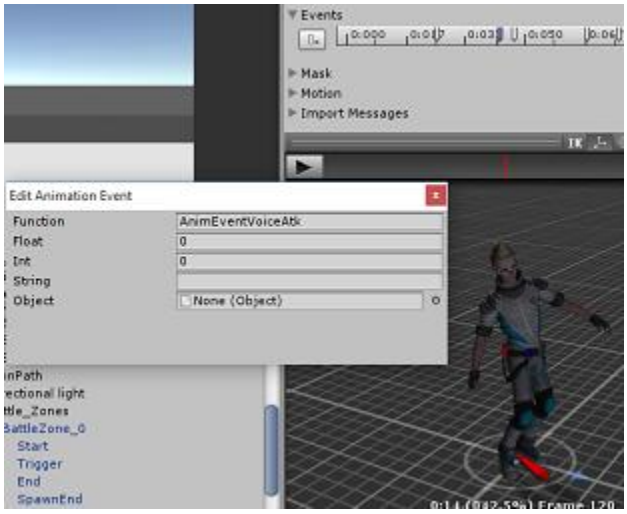
- **Player AI**
  - *In Battle*
    - Players will only go after healing items if player 1's health is less than theirs - 2. They allow the other player to get it, how nice of them. They will only pursue items if no enemies are nearby them, in their DetectionRadius' inCloseRangeChar list.
    - Attacking wise, the players will always be able to do all of the different attack things: grabbing and throwing, air combos, and high combo counts. You can however, make it so they can't grab nor do air attacks if you make the bools which allow them to use those things, false. Those bools are canUseGrabs, and canUseAirAttacks, which are found in CharacterAI. The player AI's attack and evade ratio will always be the same; what you set it to for them in the inspector.
  - *Not In Battle*
    - If there are any item containers/crates around, the AI will always go after them and try to break them. As for healing items, if their health is less than 90%, they will get rid of any item they are holding, if any, and go grab those first since you can't pick up another item when holding one; if player 1's health is less than theirs - 2, then they won't pursue the healing item. How nice of them. For any other type of item, they will choose randomly.
    - If no items are available, or the player AI is done with the ones available, the AI will follow player one.
- **Enemy AI**
  - *In Battle Only*
    - Enemies will come in from the right side of the screen, from behind, or from the front. They will come in using one of two different enter battle area types: simply walking in their spawner's direction, or using their nav mesh agent to get into the area from anywhere. I use the nav mesh one when I want them to move around a wall to come into the battle area. After they have entered the area, they will target the player nearest to them.
    - Enemies will go after items in the same manner as players, except of course they do not care if a certain player has less health than theirs. So for the same as players, they will only pursue items when no players are nearby and they go after healing items first unless none are available, then they will choose one randomly.
    - Attacking wise, the enemies will only be able to do grabs when in hard, and air combos when not in easy. Their attack ratio and evade ratio also changes so they do both slightly more in a higher difficulty. Check SetupDifficultyThings on CharacterAI to see this.

- **Both**
  - *In Battle*
    - AI will first find the nearest opposing character to target. After targeting them, they will move towards them and attack when within their minAttackDist and maxAttackDist which can change depending on if they are holding an item or not. They will only pursue items if there is not a single opposing character close by, in their DetectionRadius' inCloseRangeChar list.
    - They will go into targeting mode when less than 3 units away from their targeted enemy and not retreating. The player will start slowing down when less than a set value away from their target they are going after, whether it is an item or a character. This set value is a variable called distToSlowDown which changes depending on if they are pursuing an item or not. If pursuing an item, they will move even slower. If greater than distToSlowDown, the AI will run to their target.

Character Voice Animation Events

I have made a couple of animation events for the character to use a voice clip if available. I use this for dodges and attacks. In CharacterAttacking there is a voice array for regular attacks and one for heavy attacks. There is also a voice array for dodge voices.

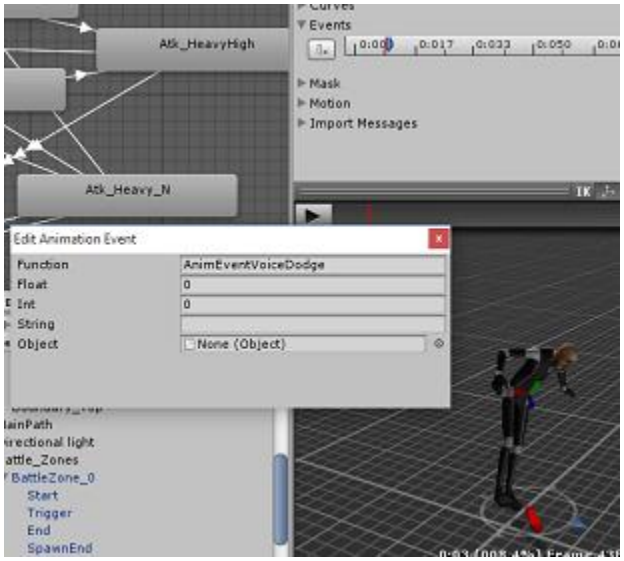
There is an animation event on all attack animations except for the weakest attacks, called AnimEventVoiceAtk which gets called on CharacterAttacking. In CharacterAttacking there is a voice array for regular attacks (voicesAtk) and one for heavy attacks (voicesAtkHeavy).



Animation event shown for an attack voice.

In the animation event AnimEventVoiceAtk, the int parameter is used. When it is 0, the regular attack voice collection is used, otherwise if it is 1, the heavy attack voice collection is used. After choosing a collection, a random voice clip will be played from it. On a side note, my high attack animation (shown in the image above) is the same animation used for several attack states. I check to see which one we are in when this event gets called, and play a heavy voice clip instead if in the Atk_Counter or Atk_HeavyHigh animation state, since those use the same animation and are heavy attacks.

There is also a voice array for dodge voices. It is used in an animation event called AnimEventVoiceDodge.



Animation event for a dodge voice.

No parameters are used for this animation event. A random voice clip is chosen from the voicesDodge array unless air evading, which in that case, a particular one is chosen. In my case when air evading, I use the one at the end of the array so I make sure not to use that when dodging normally.

Input

This section will explain the input settings and what each button does. Note that I will explain the name, not including the P1 or P2 since one is simply for player 1 and the other is for player 2. The "JS" simply means that it is for joystick input instead of keyboard.

- "*Horizontal*" – Horizontal keyboard movement.
- "*HorizontalJS*" – Horizontal joystick movement.

- "*Vertical*" – Vertical keyboard movement.
- "*VerticalJS"* – Vertical joystick movement.
- "*Action"* – If not holding an item and not near one, this will attack. If you are running, this will do a dash slide attack instead of regular attacks. Hold it to do a heavy attack (not for running), or if you currently have an enemy held, it will throw them. Tap this after guarding an attack to counter. If near an item and not holding one, this will attempt to pick it up. If holding an item, this will swing/throw it. This is also the submit button on menus.
- "*Jump*" – Yep, make the character jump when on the ground and not busy.
- *"Guard"* – Hold to guard. Tap before being hit in the air after being in hit stun long enough to air evade. Check the air evade time in CharacterStatus in Update(). Tap while walking or running to dodge. Press a direction while guarding to dodge in the chosen direction.
- *"Target"* – Hold to stay facing a chosen target. Tap to change targets or pick the closest one if there is currently no target set.
- *"Submit"* – I left this in here just in case you would want to use a different key or button for menus.
- "*Cancel*" – I left this in here also just in case you would want to use a different key or button for menus.
- "*Pause"* – Pause the game and bring up the pause menu.
- "*Run"* – Hold this button to run. While running you can do a dash attack after pressing the action button.

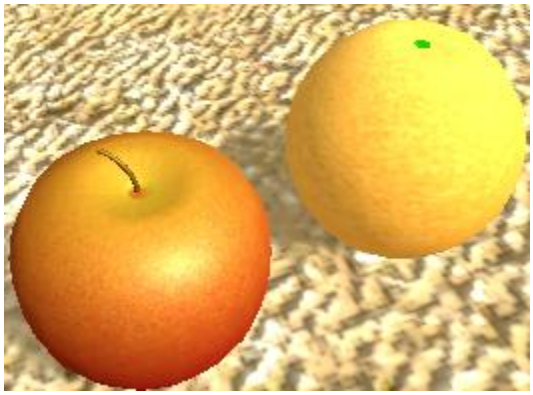
You can check the PlayerInput script for all additional input information.

Items

A brief overview of each item type will be described here. Note that all items have a script called FlashAway which will make them flash after a set time and shortly after, disappear (get destroyed).

Collectables

These are items that get used up right away after being grabbed. All I have for these are healing items. The healing effect will take place after the character who grabbed one of these returns back to idle. Check the method UseCollectable on CharacterItem and WasHealed on CharacterStatus for more information. These items all have the ItemCollectable script so I recommend checking that for more information on these items.



The collectable healing items, an apple and an orange.

Throwables

Throwables are items that can be carried and thrown at enemies to damage them. These will break after their hit points reaches 0. I have a broken prefab for these which will get instantiated upon breaking. Instead of these items getting used up after the grabber returns to idle from grabbing them, these will get parented to the character's grabMountItem transform found on the character's CharacterItem script. The character's throw animation uses an animation event called AnimEventUseThrowable which sends the message "WasThrown" to the item, resulting in it not being a child of the character's item mount anymore and gives it velocity. These items all have the ItemThrowable script, so I recommend checking that for more information on these items.

The throwing items, a rock and a throwing knife.

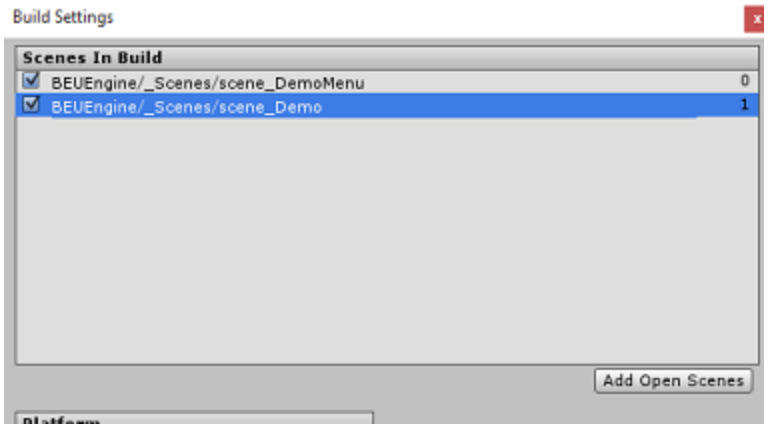
Weapons

Weapons are items that are extra strong and give more damage with fewer hits. These will also break after their hit points reach 0. I also have a broken prefab for these which will get instantiated upon breaking. After the character who grabbed one of these returns to idle, the weapon will become a child of the character's grabMountWeapon transform found on CharacterItem. When holding a weapon, you will use a different attack animation and can only combo up to two hits. The weapons are quite strong and do more damage & stun. They receive damage in HitboxProperties in OnTriggerStay upon hitting an opposing character. StruckHit is the method that gets called on ItemWeapon which is what deals the damage given to it and also makes it break after all of its hit points are gone.

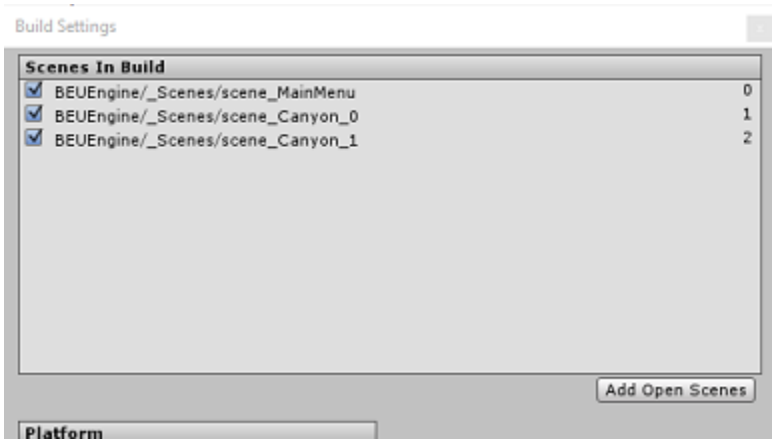
The weapons, a sword and a staff.

Scenes

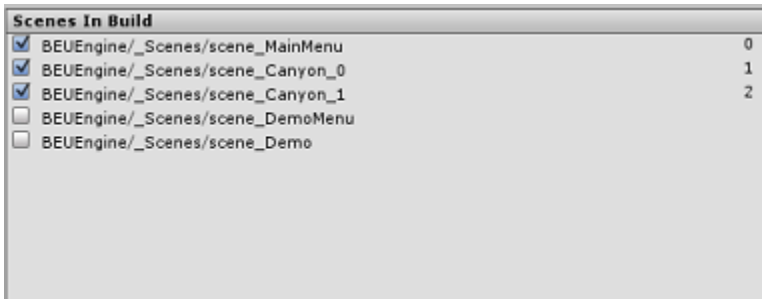
Here I will simply show the build settings needed in order to play the certain set of scenes. There is the demo scene and menu as well as the regular canyon and main menu. When using the demo scenes, just make sure that **scene_DemoMenu** is placed right before **scene_Demo**. When using the canyon and main menu scenes, just make sure that **scene_MainMenu** is placed right before either **scene_Canyon_0** or **scene_Canyon_1**.



Using the demo menu and demo scenes.



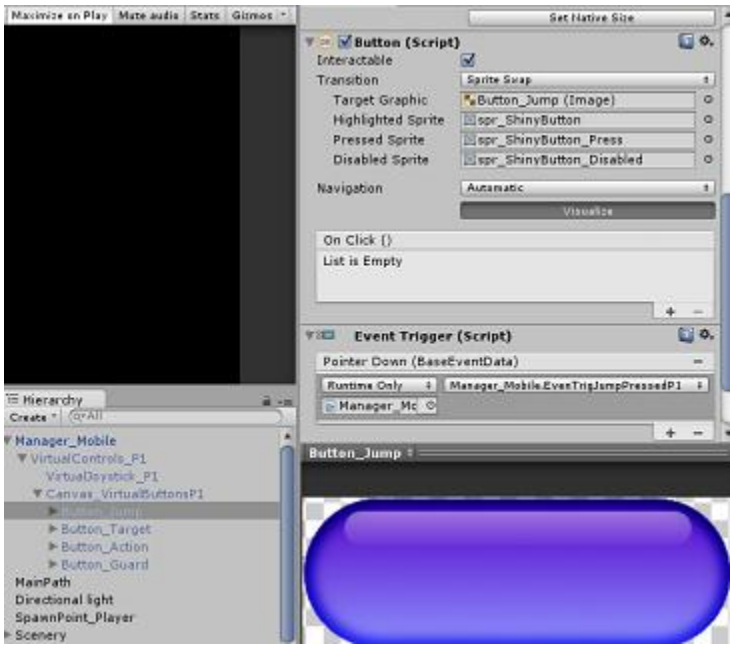
Using the main menu and canyon scenes.



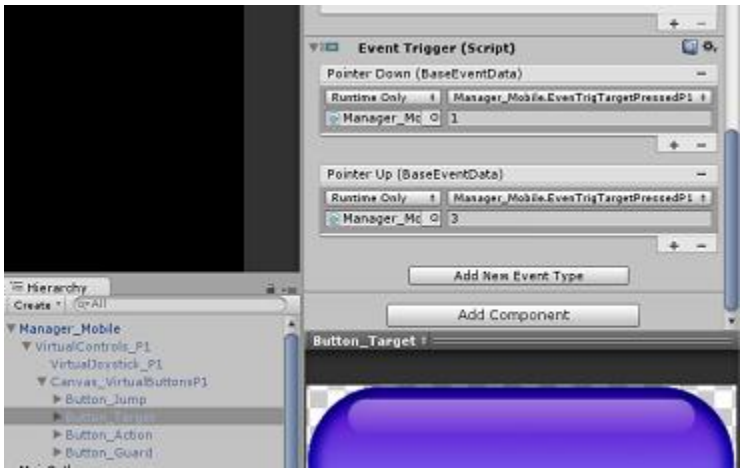
An alternate way, only check ones to use. This is the preferred way so you don't need to keep deleting and putting scenes back into the build settings.

Mobile Input

Here I will explain the things added for mobile input. There is a new manager script that has been added called, Manager_Mobile which is attached to a new game object with the same name. Manager_Mobile receives virtual button interaction and has methods for each one: EvenTrigActionPressedP1, EvenTrigTargetPressedP1, EvenTrigGuardPressedP1, and EvenTrigJumpPressedP1.



The jump virtual button game object selected.



The target virtual button game object selected.

As you can see from the above images, the virtual buttons use one of the event methods mentioned above. You can see that the jump one does not pass in any parameters since that button is only needed to check if we have just pressed it since you always jump the same height and no other actions are available regarding the state of the button being pressed or released. The target button and the others pass in an integer parameter where 1 = pressed down and 3 = released. PlayerInputMobile for P1 receives these messages and does the corresponding action accordingly, just like with a normal keyboard or joystick button press. P2 does not have mobile input support since that would require networking.

PlayerInputMobile has its receiving method messages: ActionButtonPressed, TargetButtonPressed, GuardButtonPressed, and JumpButtonPressed. PlayerInputMobile has variables that get updated from these which helps for doing the required action. They are actionButtonPressed, targetButtonPressed, guardButtonPressed, and jumpButtonPressed. Manager_Mobile checks upon starting to see if the usingMobile bool of Manager_Game is true and that the currently loaded scene is not a menu. If that is true, it will remain active and mobile input is enabled. Otherwise it will disable the virtual joystick and itself to completely disable mobile input. Check PlayerInputMobile and/or Manager_Mobile for more information.

Update Log

- ✓ *Version 1.02:*
  - Added mobile input support.
    - PlayerInputMobile is a new input script added to each player prefab which will take note of when virtual mobile buttons have been interacted with on-screen by way of receiving messages from a new manager script, Manager_Mobile.
    - A new game object prefab Manager_Mobile has been created which holds the virtual joystick and virtual buttons used for mobile input and will only be activated when the usingMobile bool of Manager_Game is true.
    - Added another script, VirtualJoystick which is on a child game object of the Manager_Mobile prefab which is what PlayerInputMobile uses to help determine what direction is being used for movement.
    - Created a few graphics for the mobile virtual controls, a joystick and a button.
    - Added a "Mobile Input" section to this document.
- ✓ *Version 1.03:*
  - Fixed the jerky motion of the camera when following the characters:
    - No more interpolation is done for the characters on their rigidbody component so I removed that code from CharacterMotor where it was being changed in the Move method. The camera simply now uses FixedUpdate instead of LateUpdate in Camera_BEU in order to be on par with the players better. I noticed this after experimenting with it.
- ✓ *Version 1.05:*
  - Three and four player support added! Along with that came the following:
    - Updated Manager_Game, Manager_UI, Manager_Menu. Changed the AssignUI method name on various character scripts (CharacterAttacking, CharacterItem, and CharacterStatus) to CreatedSetup.
    - Updated CameraBEU by making it check for all players being far enough away before removing any added distance away from them.
    - Updated FaceTransform and Manager_Targeting with offsets for the player target cursor's x position depending on how many players have an enemy targeted.
    - Updated the demo scene canvas as well as the main Canvas_Overlay prefab with player 3 and player 4 slots for the things it needs in the inspector.

Where to go from here:

I highly suggest playing the game and getting a feel for it before diving into the scripts and the layout of the game. That will definitely help in getting even more of an understanding of how everything works. After doing so, you can look at each section, one part at a time until you see how it works and change or edit anything to your heart's desire.

Check the comments on all of the scripts and take note that any script mentioned in a section on here most likely holds more information on what is being talked about. The Manager scripts are well explained describing how they function and what they do also. Start small, and learn things one step at a time! If you ever have any questions feel free to email me anytime and I will get back to you as soon as I can. Check below for my email.

Questions/Comments/Suggestions/Feedback/Bug Findings?

Feel free to email me at GameStar18@yahoo.com. I have fixed a countless amount of bugs and weird issues but it is true that there are no games or projects out there that do not have bugs. If you find anything you feel is worth mentioning please let me know on that too so I can fix it/them for another update. Thanks!

Credits:

- ***Unity Technologies:*** *for their character models, other assets, and of course Unity!*
- ***DrPetter and increpare –*** *DrPetter created Sfxr and increpare created a different version based off of it which he called Bfxr. These programs create sound effects and I used them for a number of sound effects.*
- ***GoldWave –*** *program used to change my voice and sound effects in a variety of ways.*
- ***GIMP and Paint:*** *for image editing.*
- ***Autodesk 3DS Max****: for my item models I created and creating animations for this project.*
- ***Myself:*** *For spending all the time to put this project together. Much trial and error, experimenting, research, and sticking with it even during the worst bugs.*