Unity 3D Third-Person Character and Camera Controller – Documentation

Contents:

Versions:

Current Version = 1.0

Version Differences:

1.0 (Initial Release).

This asset is a part of another 'full' asset which is the **Third-Person Game Foundation Kit**. The Third-Person Game Foundation Kit (TPGFK) is the entirety of this asset along with features such as climbing, swimming, an inventory system, HUD functionality and many more that will be included once it releases as version 2.0. The TPGFK is more expensive but has a lot more content to build a third-person game from; this asset is just the bare camera and character controller, ready to be used in a less restricted scope.

Overview (Project Aim):

KEY:
-'user' refers to the person using this asset,
-'player' refers to the person playing the build,
-'character' refers to the in-game character that the player controls,
-'second analog stick' can also mean 'moving the mouse around',
-'L-Trigger'/'Target Button' can also mean 'right mouse click',
-'FPV Button' can also mean 'mouse scrollwheel click',
-'Action Button' can also mean 'left mouse click',

The main aim for this asset while it was being planned was to make it as similar as possible to the gameplay style of classic 3D platformer titles. Those titles have been the stage I've tried to reach while creating this asset and have achieved something very similar to the way that those games function. It works via a script attached to the character (which is a Capsule Collider with a model as a child transform) known as 'ThirdPersonController.cs' and a script attached to the camera known as 'CameraFollower.cs'.

The main hurdles were the character controller and the camera's movement. Having the character move around the camera while also having the camera follow the character was something I've been trying to implement for a while and wasn't able to find much on the internet about how to accomplish this. It turned out to be a very simple, small amount of code that works flawlessly. Three other camera modes also had to be implemented, one which wasn't present in the games I was trying to emulate. Targeting on certain objects is also a feature that I thought was necessary for this asset. After that, the camera's collision with walls had to be worked on, as well as how to adjust the camera if the player is out of sight. This was something I also couldn't find much on online about however there were some projects/examples out there that helped push me in the right direction.

Finally, it was just a matter of doing animations, adding actions and so on. I wanted to make an asset which can allow the user to just drop in their own character model and animations and have something to play about in. The asset is also intended for being played on a GAMEPAD, however the input can obviously be tweaked in the Input Manager to whatever the user likes and can currently be played with a keyboard and a 3-button mouse.

For a Third-Person game asset with more in-game features, check out my **Third-Person Game Foundation Kit**.

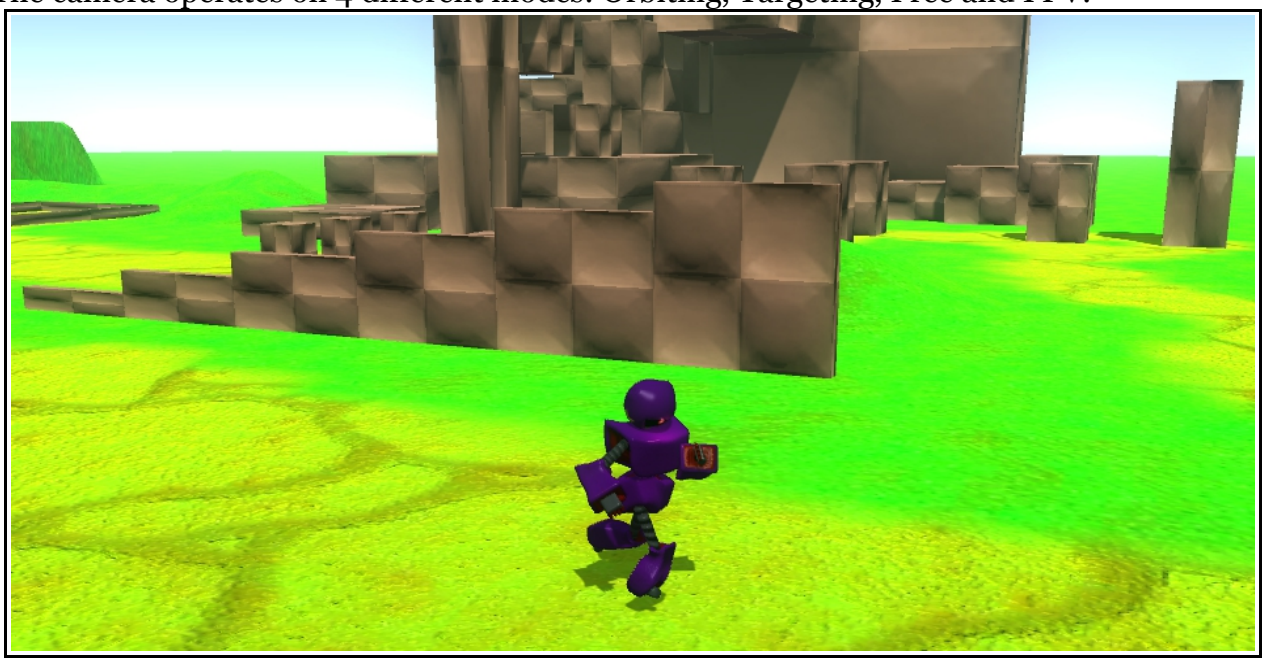
Features:

Here are the list of features included in the current version of this asset (each are described after):
-Camera Modes (Orbiting (follow/behind), Targeting, Free, First Person View (FPV);
-Camera Limits (Collisions and management);
-Character Movement;
-Slope Handling;
-Character Jumping off ledges;
-Ledge Climbing;
-Ledge Grabbing;
-Targeting Objects;
-Rolling;
-Jumping while Targeting;
-Jump on button press;

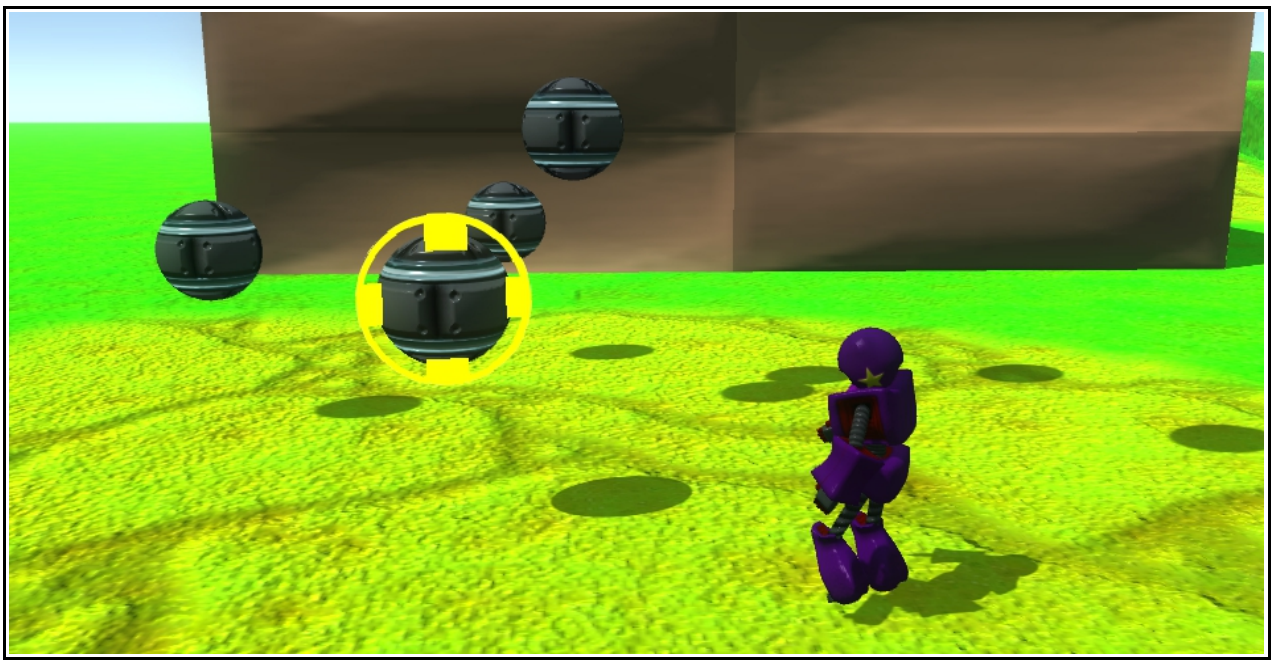
(All images below were taken in from the **Third-Person Game Foundation Kit** which is built on top of this asset with no modifications).

Camera Modes

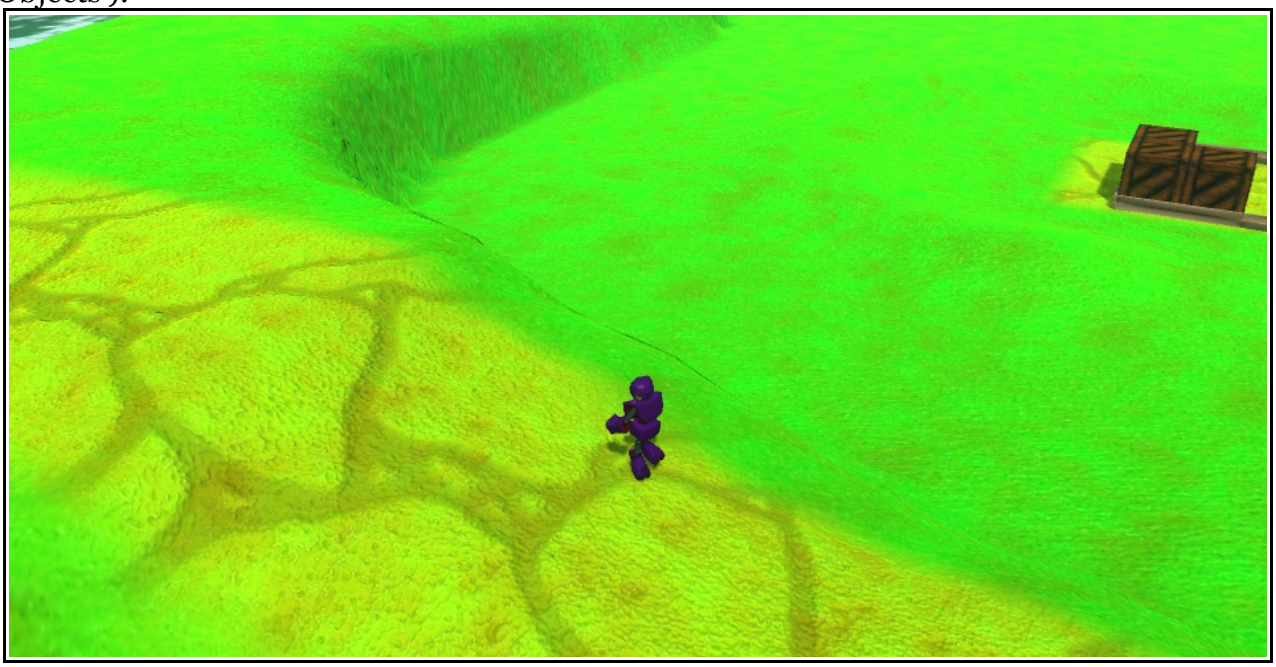
The camera operates on 4 different modes: Orbiting, Targeting, Free and FPV.



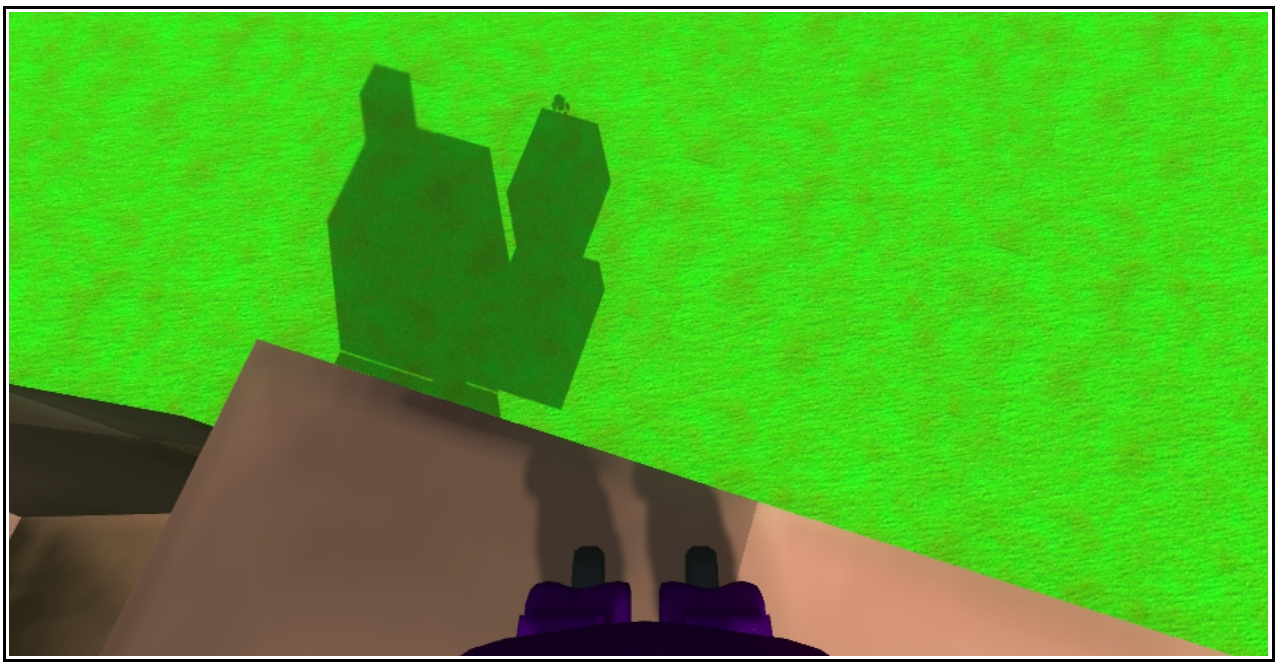
The orbiting (behind) mode is the default mode (like in the most third-person titles) which follows the player and, as the character changes direction, the camera rotates to look at the character. If the character moves to the left/right (if they hold in one of those directions), they will move in full circle around the camera, never leaving its sight. Moving away from the camera will delay the camera following very briefly, and then start following right behind the character. Moving towards the camera will move the camera back while still keeping the character in sight. Explaining it is difficult, but it does work wonders for anyone wanting to make any sort of 3D third-person game in Unity (especially with a gamepad).



The targeting mode focusses directly behind the character while the targeting button is pressed. Pressing it quickly and letting go will focus the camera behind the player (almost instantly) and then revert to orbiting mode. Holding down the button will lock the camera behind the character and have the character move in a straight line in whichever direction the player is holding. The camera will behave differently if there are targetable objects within a certain distance while the targeting button is pressed (more on that in 'Targeting Objects').

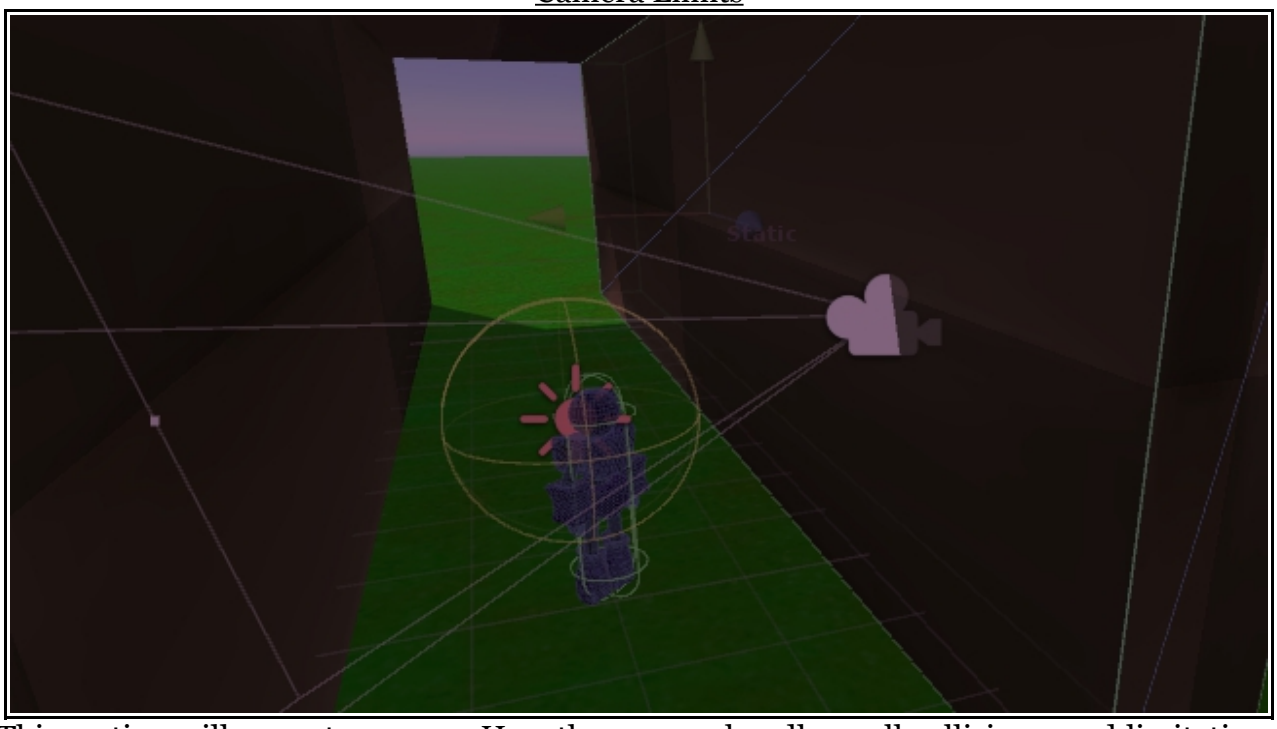


The Free camera mode allows the player to pan the camera away, towards and around the character with a second analog stick. The maximum distance that the player can zoom out to can be changed for the users needs. After zooming out to a position, the camera will stay in that position (of course, relative to the character) until the camera has been switched to either the targeting mode or first-person-view mode. The camera now has added y-axis movement so it can move all around the character in any direction as well as zoom in (however, when playing on a gamepad, the y-axis movement is merged with the zooming in and out).



Finally is the First Preson View (FPV) mode. When the player presses the FPV button, the camera will position itself in front of the character's face. When in this position, the character can move around like normal, and moving a second analog stick will pan the camera around. There is a limit to how much the camera can move up/down, but can fully rotate left/right around the character. While in this mode, the mesh that the characters head is made out of is set to only cast shadows so it cannot be seen by the in-game camera; so if the user would like to change which mesh isn't visible in this mode then make sure that the path to the mesh is changed in the CameraFollower script.

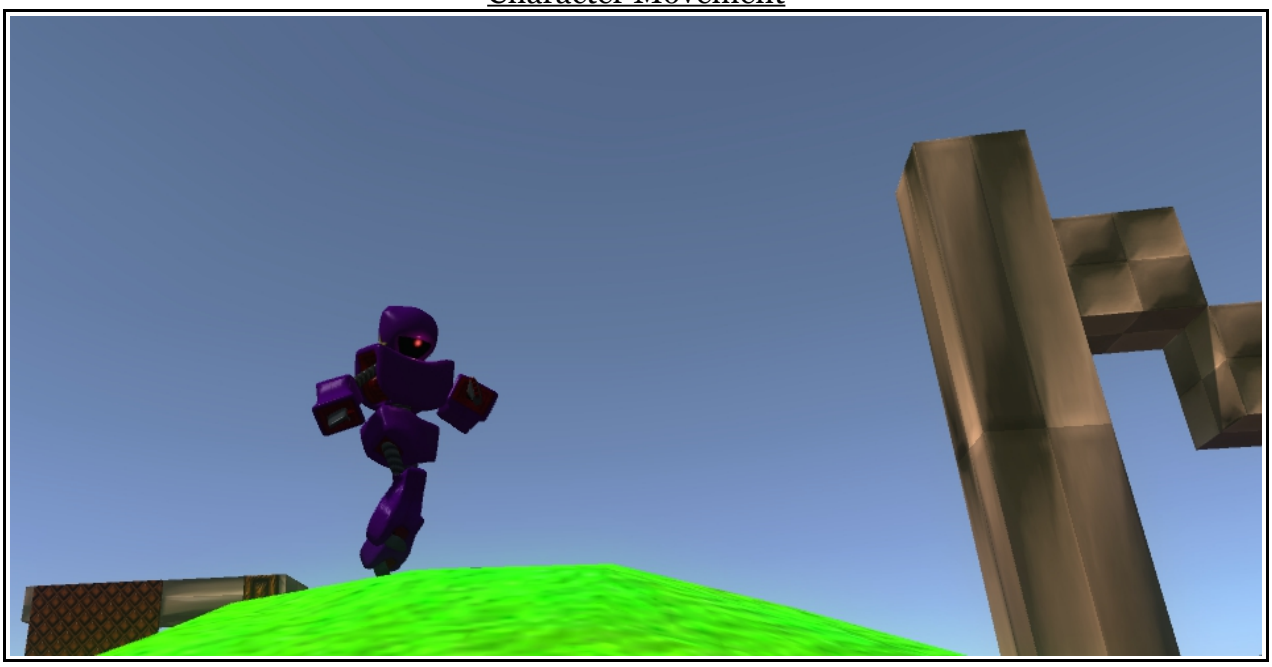
Camera Limits



This section will cover two areas: How the camera handles wall collisions, and limitations to which mode the camera can be in depending on certain circumstances.
Wall collisions with the camera are handled with a single method in the cameras script.

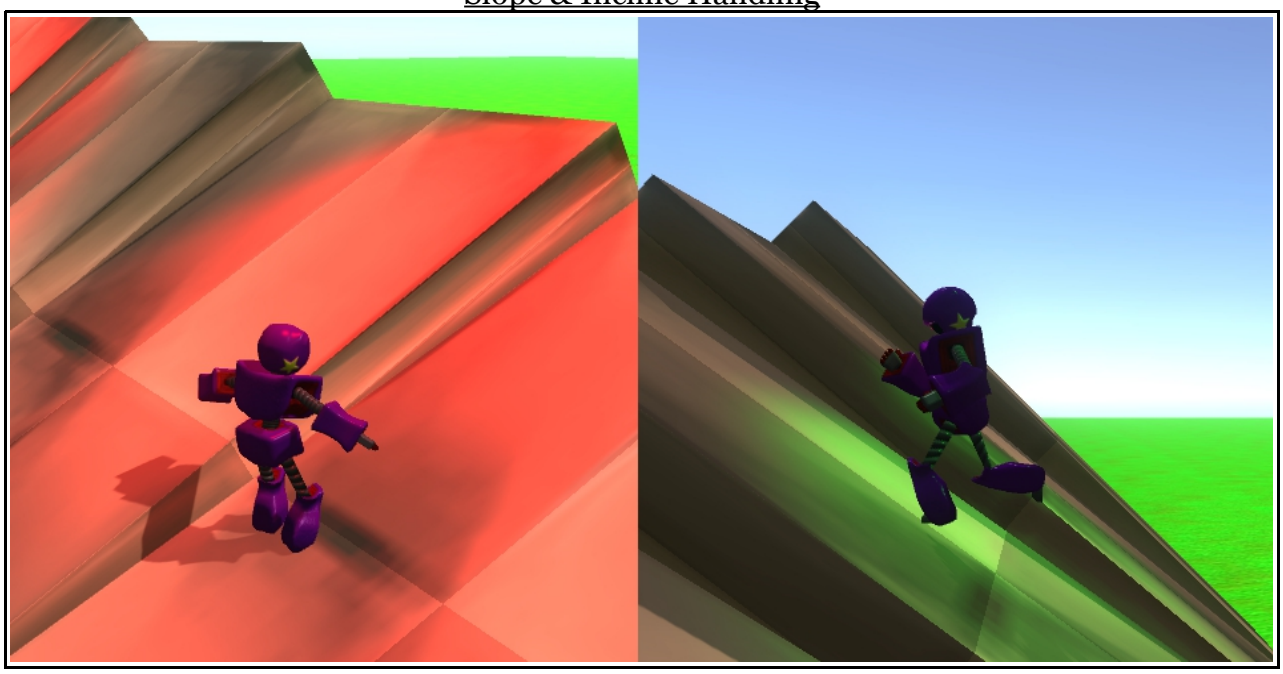
The way that the camera works is that it looks at a 'FollowMe' transform that is a child to the character, it is positioned slightly above the character so that the camera will not adjust it's movement if the top-half of the character is still showing. The method is called just before each update to the cameras position so that it will 'correct' the cameras position before it goes through a wall. It works by checking if there is any collision between the character and the camera. If so, it will move to in-front of that collision. Then it will check behind to see if it can move back to its default distance away. When it can it then checks to see, once it has moved back to its default distance away, if that collision will be obstructing the view again. If so, the camera does not reposition, otherwise it transitions back to its original distance away. Also, the camera rotates slower around the character in Free Mode the closer it is to the character (otherwise it will speed right past the character leaving the movement of the camera very slippery).
Secondly, there are the limitations; simply 'if statements' that check if a camera is in a certain mode, then it cannot access this mode. Or if the camera is in the middle of doing this, it cannot do that. For example, if you are targeting, you cannot also enter FPV mode as the whole point of target mode is to position it directly behind the character.

Character Movement



The characters movements have already been explained in the 'Orbiting Camera' section, however I'll describe it again. The character moves relative to the space that the camera is looking at. Pressing up (as in the positive Horizontal axis) will move the character away from the camera, up the middle of the screen. Pressing down will move the character towards the camera, still in the middle of the screen. Pressing left/right will move the character in a prefect circle around the camera while the camera does not change position. This is how most 3D third-person games function, and precisely how the classic titles function. It works flawlessly and doesn't require root motion to move the character, just code in the script.

Slope & Incline Handling

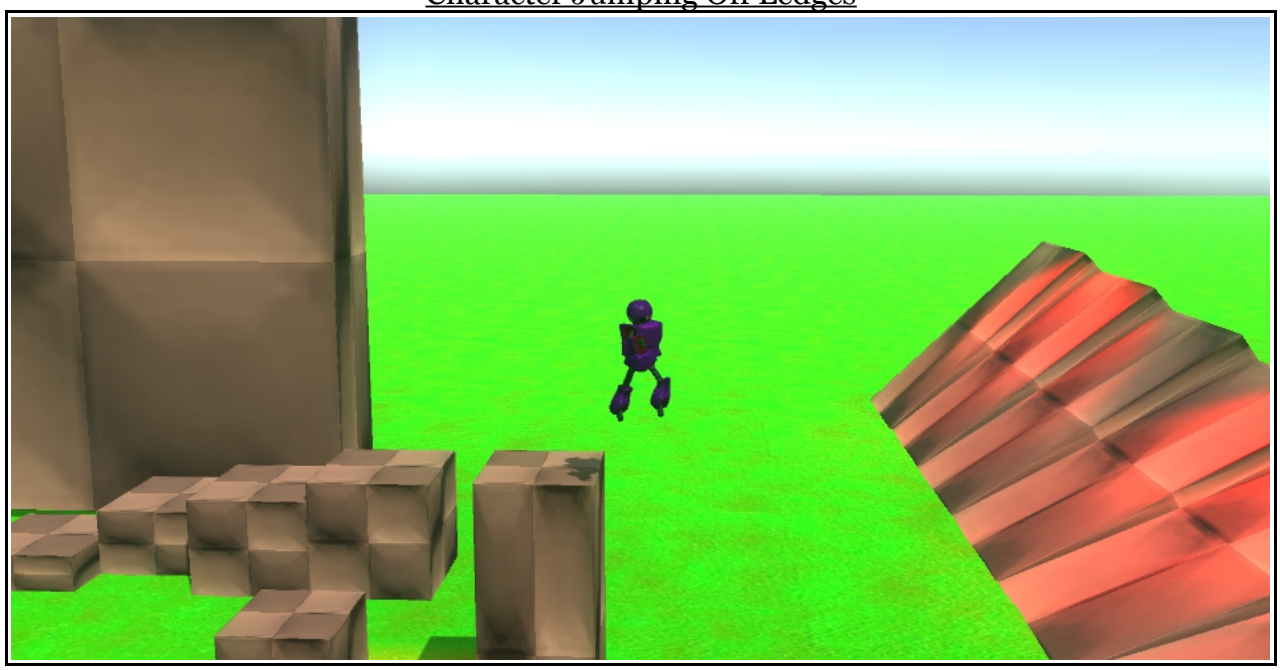


(Slide-down slopes: Left | Walk on inclines: Right)

The way it works now is that it uses the Collider components built in OnCollision() methods to detect any collisions it touches. Depending on the gradient of the ground and the collision layer of the collider it's touching it will act accordingly. If a flat surface, the character will move like normal. If a sloped surface that the character can traverse on, the characters direction of gravity is directed from its centre of mass (the centre of the bottom sphere of the capsule collider) to the point of contact. This allows the character to move around on the sloped ground as if it were flat. If the ground changes gradients, the gravity changes accordingly as to not get the collider stuck on edges between faces of the ground mesh. Finally, if the collider detected is of layer 'Slopes', the character will slide down it regardless of the gradient of the slope.

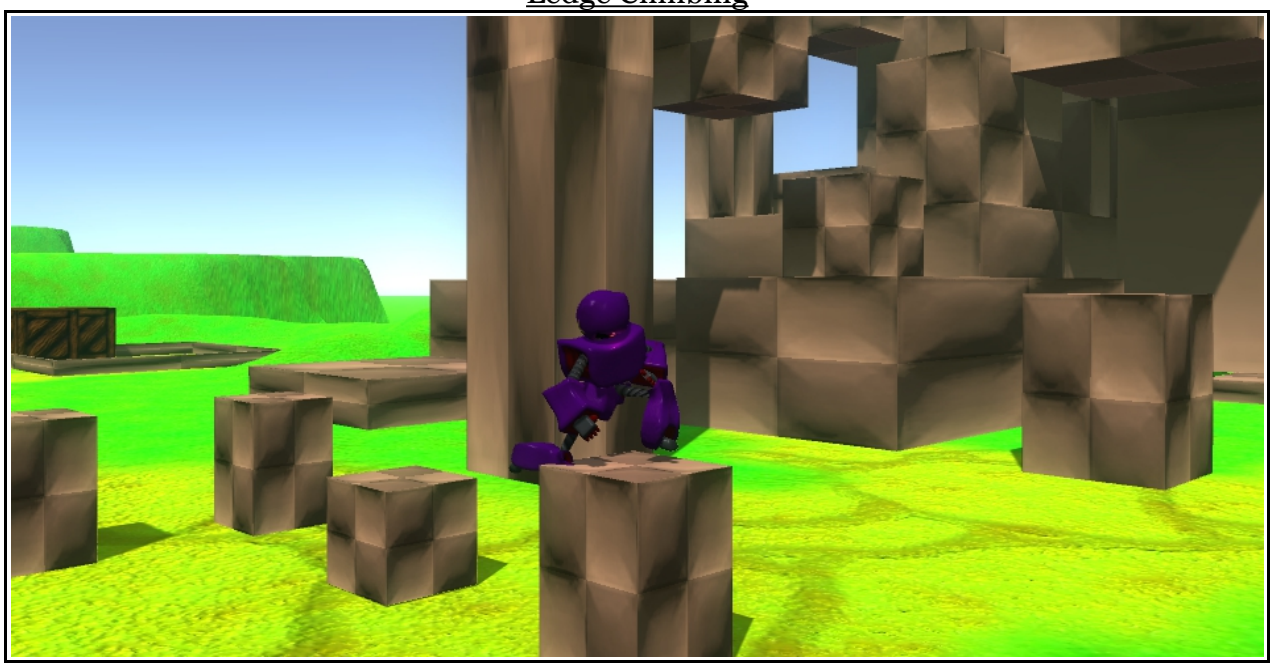
!!-The following features are more about emulating an RPG feel, rather than making a general purpose third-person adventure asset (of course, these can all be removed)-!!

Character Jumping Off Ledges



A line cast checks for ground just behind the character when the OnCollider() methods have set the character to not be 'onGround'. If it's the case that the line hits ground, and if the character is moving fast enough, then a vertical impulse is put on the character and sets a 'jumping' bool. After some time, the character is in a 'falling' state which can be very slightly controlled; also at the point of jumping, the gravity vector is reset to its default direction. Not moving fast enough will cause the character to slowly drop down to grab the ledge and hang off from it, transitioning to a 'ledge grabbing' state (which I will explain after the next feature).

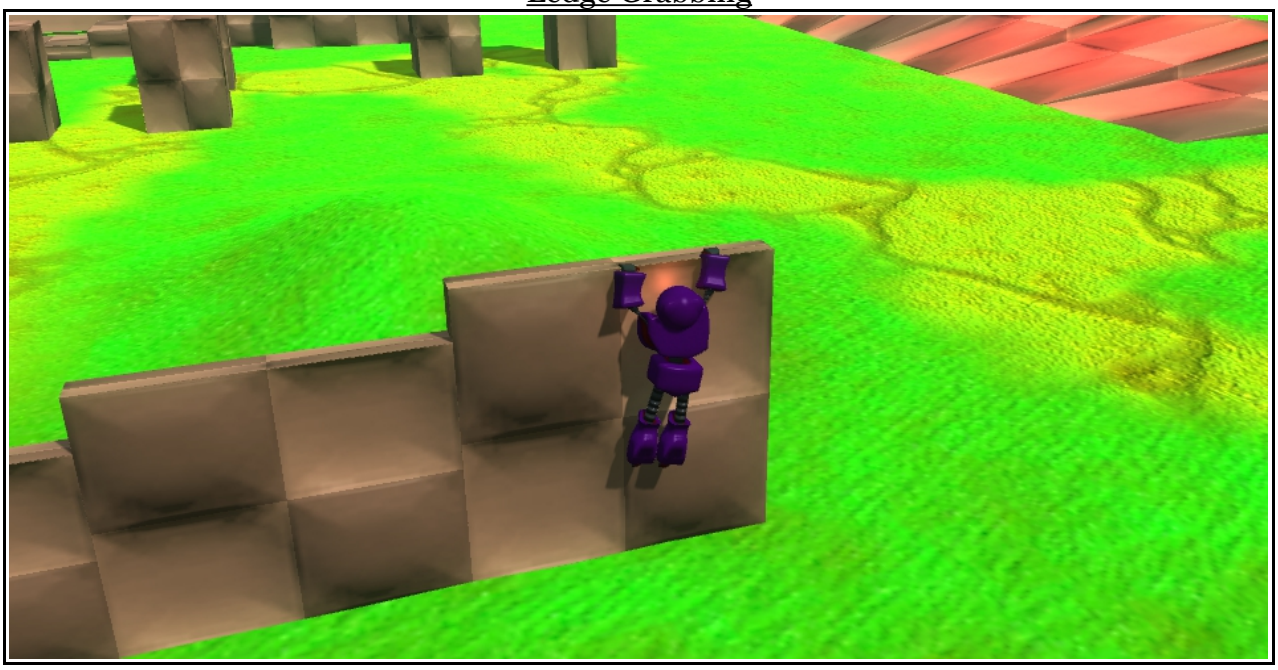
Ledge Climbing



There is a line cast downwards in front and above the character. It detects any even land (land with a certain range of gradients) which the character can climb onto. Depending on the length from where the lines are cast to where they hit, the character performs the appropriate animation (with the character model moving up the ledge and the collider

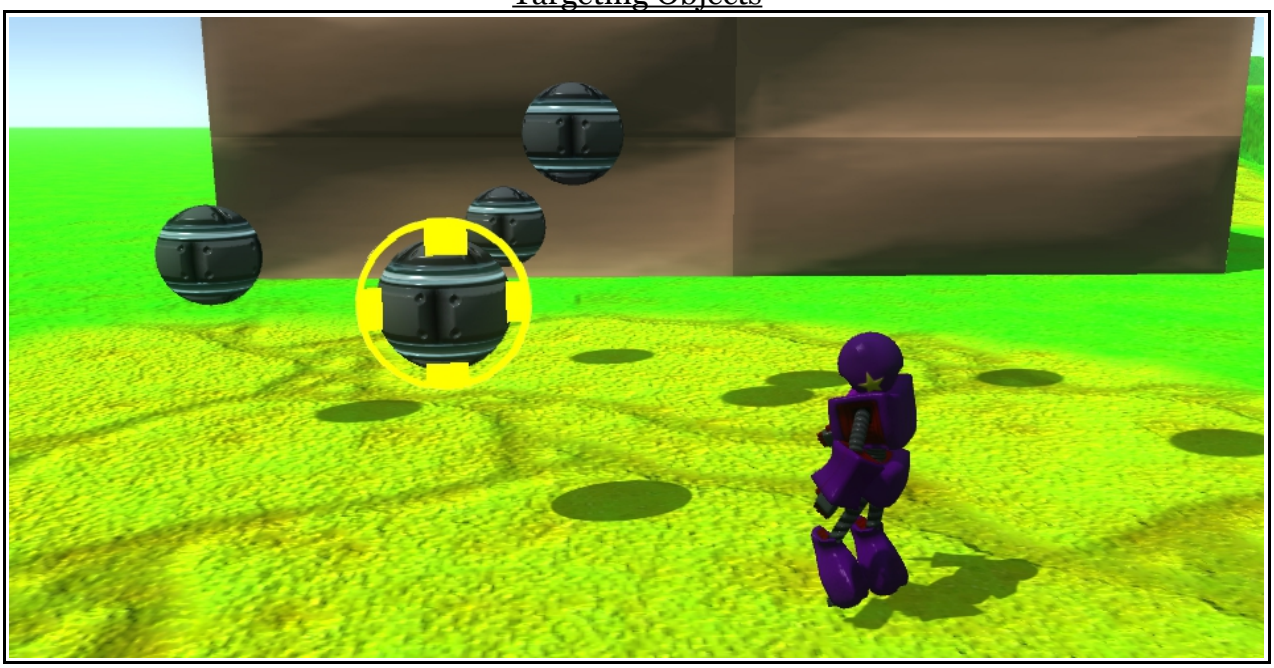
staying in one spot) and then have the collider position itself at the models position at the end of the animation. Ledges that are rather high but still grab-able will have the character jump vertically and grab the ledge (explained in the next section). The animation starts at a fixed height below the ledge that is being climbed, meaning that the lowest possible ledge height for each animation may have the model start the animation within the ground they're standing on; though this is very brief and rare, hence it's barely noticeable.
Note: The animations I have supplied in this package are my own, amateur work. The user (if wishing to use their own, better animations) will have to create animations that are in time with the movements through code; or better yet, create blend trees which alter the height of one animation depending on the length of the linecast.

Ledge Grabbing



As mentioned in the section above, the character can jump up to grab a ledge. This works using the same line casts in the ledge grabbing mechanic, however this places you into a specific spot under a ledge and puts the character into a 'grabbing' state. This works whenever a character is off the ground, is falling downwards, and a grab-able ledge falls into the line casts. When the character is in the grabbing state, they can move along that same ledge (left/right) and can pull themselves up if it's okay to do so (nothing in the way at the top) or can drop with a press of the action button. The character can not move around corners while on a ledge.

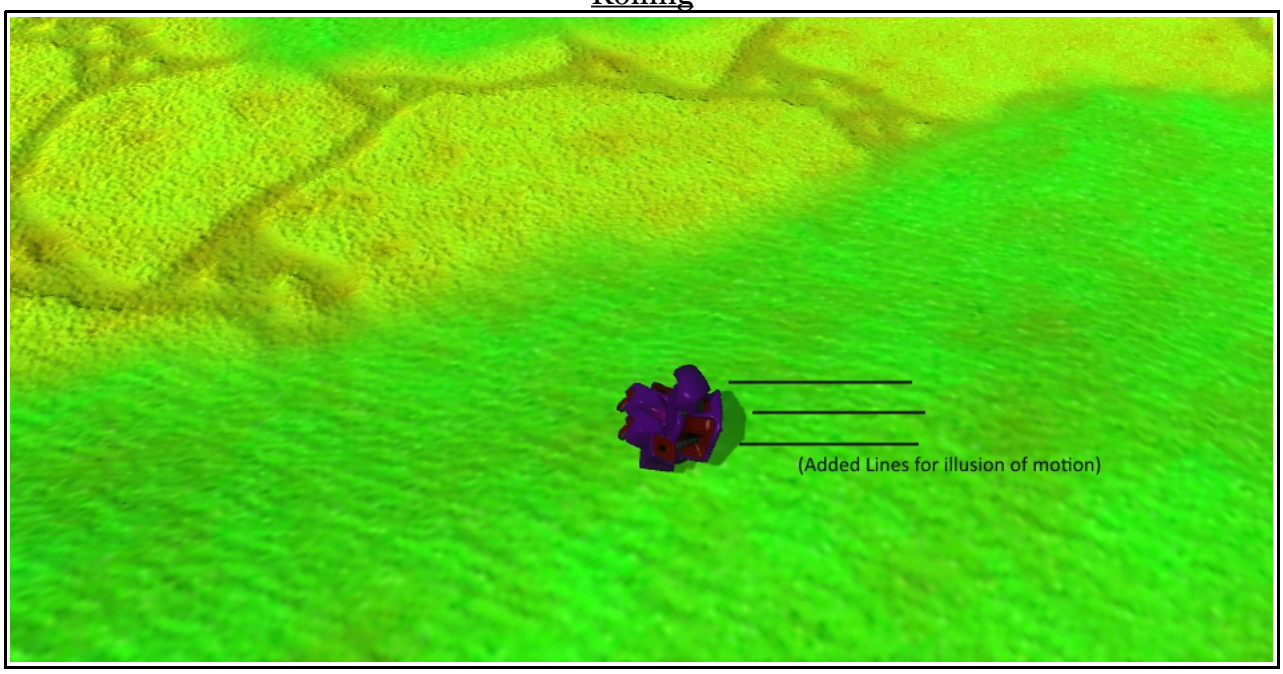
Targeting Objects



Certain objects will have the tag 'Target'; these will be 'targetable' objects. When the L-Trigger is pressed (for focusing/targeting the camera), the script runs through each of the targetable objects in the scene, adding them and sorting them in a list (depending on how far away they are). On each trigger press, the list will go to the next nearest targetable object to focus on. On each press, the camera focuses behind the player at a point midway between the character and the object. A delay has been added to emulate the classic feel to have the character move away from the centre of the screen and around the target. The character can move around, towards and away from the target. Having the character move too far away from the object will cancel this targeting mode and revert to the orbiting camera mode. When a new targetable object becomes the closest one, the list is reset so that pressing the trigger again will focus on that new nearest object. This works flawlessly and can hold up to 128 objects in the list at once (hence this should be altered if you plan on having many more targets in the scene).

Also, there is a bool that acts as the target mode; you can switch between having the mode being in 'Switch Mode' or 'Hold Mode'. Switch mode allows you to cycle between each object on each trigger press (like explained above), and hold mode just focuses on the nearest object while the trigger is pressed down and returns the camera to orbiting mode when released. This mode is managed via the inspector in the editor but can be mapped to a button/menu setting.

Rolling

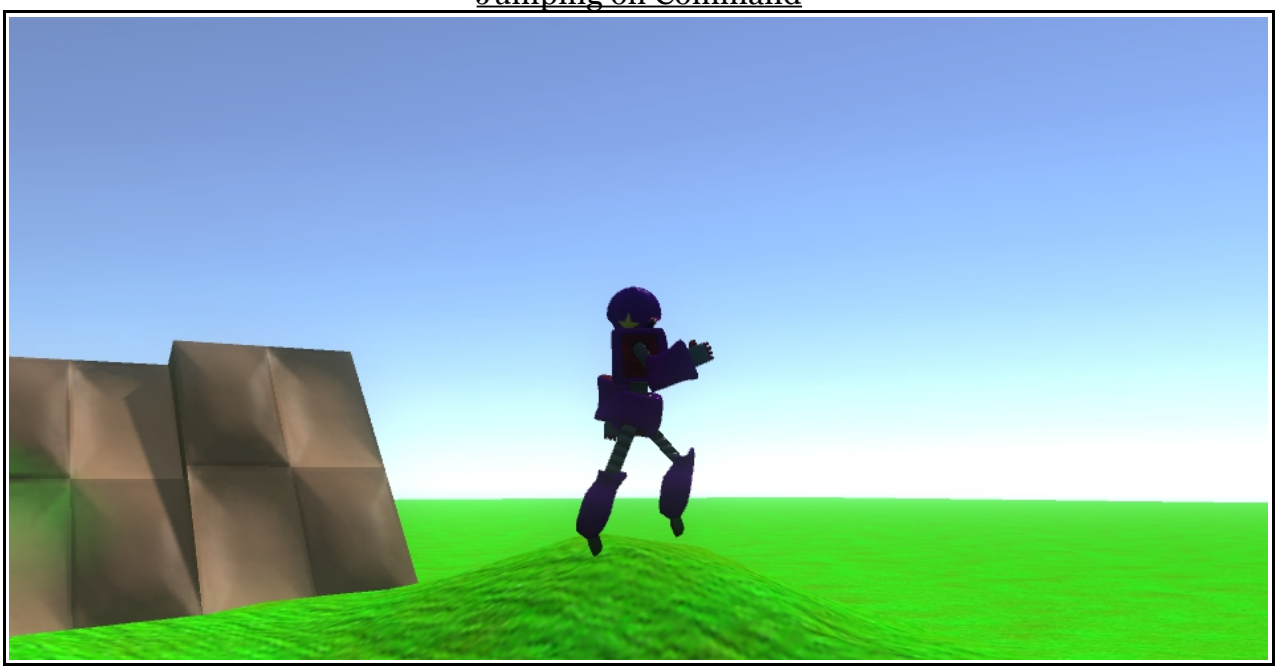


Pressing the action button while moving on ground gives the character a burst of speed. The rolling animation is played at this point and other actions can be added while this is going on. The speed is cancelled when jumping off a ledge to avoid flying off the ledge at insane speeds; only some of the speed is added to increase the jump length. The movement can be controlled while rolling (so the characters roll can strafe left/right), and if there is no input from the player, the character will continue rolling in one direction. Rolling into a collision of a certain gradient (basically, something upright/a wall) will have the character bounce off from it while playing a 'roll knock-back' animation; the character cannot be controlled during this brief animation.

Targeting Jumps

While in target mode, pressing the action button while moving in a certain direction (other than forward which causes the character to roll) will have the character perform the appropriate jump. Holding left/right will have the character side-hop a short distance. Holding backwards will have the character perform a backflip.

Jumping on Command



Pressing the left action button on the controller will have the character jump up on the spot. This uses the same code that is used when the character jumps off a ledge, however more of an impulse is added and the direction that the character is moving in influences the direction of the jump. The character can be slightly controlled in mid air, but no too much to give the realistic feel of momentum.

Support:

If there's anything you need help with in terms of understanding something about the asset or how to use it (even after checking the included User Manual), then please don't hesitate to send me an email via the contact page on my weebly website:

http://zeliggames.weebly.com/contact.html