Unity 3D Third-Person Game Foundation Kit – Documentation

Contents:

Versions:

Current Version = 1.5

Version Differences:

1.5
+Can turn every feature on or off purely in the Inspector, rather than alter code;
+Double Jumping & longer jumps when holding down the jump button;
+Skidding/Braking/Landing Animations;
+Better Animation Transitions;
+Climbing and Swimming now fully complete;
+Triple Ground Jumps and Sideways Jump when skidding;
+NPC functionality (branching conversations, replies, camera angles);
+Sprinting Button and Acceleration on ground;
+Crawling, Crouching, Roll-to-Crouch, Crouch Backflip;
+Reworked Sliding (can control movement and speed);
+Moved to New GUI System;
+Multiple 'status bars' in HUD functionality;
+Better stair climbing;
+Customizable mid-air control;
+Can drag in certain Transforms in Inspector rather than alter code;
+Capsule Collider can be different sizes;
+Handler.cs script divided into HUD.cs and Inventory.cs for convenience.

Bug Fixes:
-No more 'flicker' when ledge-climbing; it's now a seamless transition;
-GamePadInputs.cs can now read the triggers on a gamepad as either buttons or axes (can be set in the inspector).

Previous Versions:

1.4
+GamePadInputs.cs script for easier Input getting whether using Gamepad or Mouse & Keyboard set-up,
+Handler.cs script of Inventory and HUD features (health, stamina, money, etc.).

Bug Fixes:
-Input Manager has corrected Trigger Input (download available in Support section of the User Manual),
-Still acting as if onWaterSurface when already on land,
-Not jumping on command when in FPV and moving diagonally.

1.3
+Improved Incline handling (will no longer go off ground on little bumps);
+Box-Pushing Mechanic with animations, box prefab and controller prompt;
+Tested on Unity's Terrain system, works well;
+Improved First-Person View (FPV) while swimming and better FPV camera position;
+New character model for more attractive visuals;
+"Cube_001" has been changed to "Head" so it more obviously refers to the head model;
+Cameras 'Field of View' changes between FPV and the other views.

Bug Fixes:
-Swimming glitch when submerging above inclines;
-Climbing glitch when character is behaving as if it is on a climbable object when not;
-Still acting as 'onGround' after jumping in certain circumstances.

1.2
+Added y-axis camera free movement;
+Climbing controls;
+Swimming controls;
+Improved camera collision;
+Grabbing ledges improvements;
+Changed method of turning character on ground;
+Camera moves to behind character when grabbing a ledge if facing the camera;
+The character can pull up onto a ledge after a small period of time rather than waiting for the vertical axis to return to zero.

Bug Fixes:
-Grabbing while falling now working more successfully;
-Fixed positioning when grabbing a ledge beneath;
-Rolling off a ledge into a jump now working more successfully;
-RayA transform position and hence fixed roll jumping;
-Floating on land when jumping down incline;
-Sliding on slopes issue.

1.0 (Initial Release)

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.

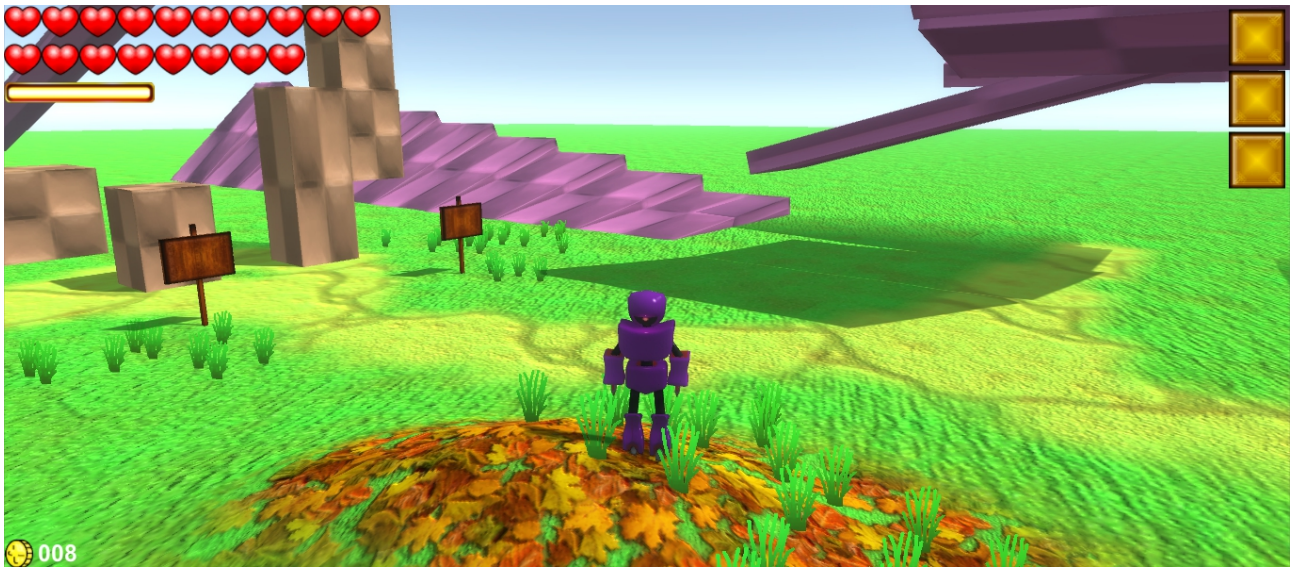
For the first version I just wanted to have the camera and all it's functions working, and the character moving around just fine. For version 1.2 I have added more common third-person gameplay mechanics such as swimming and climbing. For version 1.3 I have made the asset more aesthetically pleasing and added a box-pushing mechanic. For version 1.4 I have added HUD and Inventory functionality. For version 1.5 I have made the asset more customizable and able to use almost entirely from the Unity Editor; I have included different types of jumps, better control, better sliding mechanics, a crouch/crawling function and have moved all the GUI features over to use the new GUI system (available since Unity 4.6).

Features:

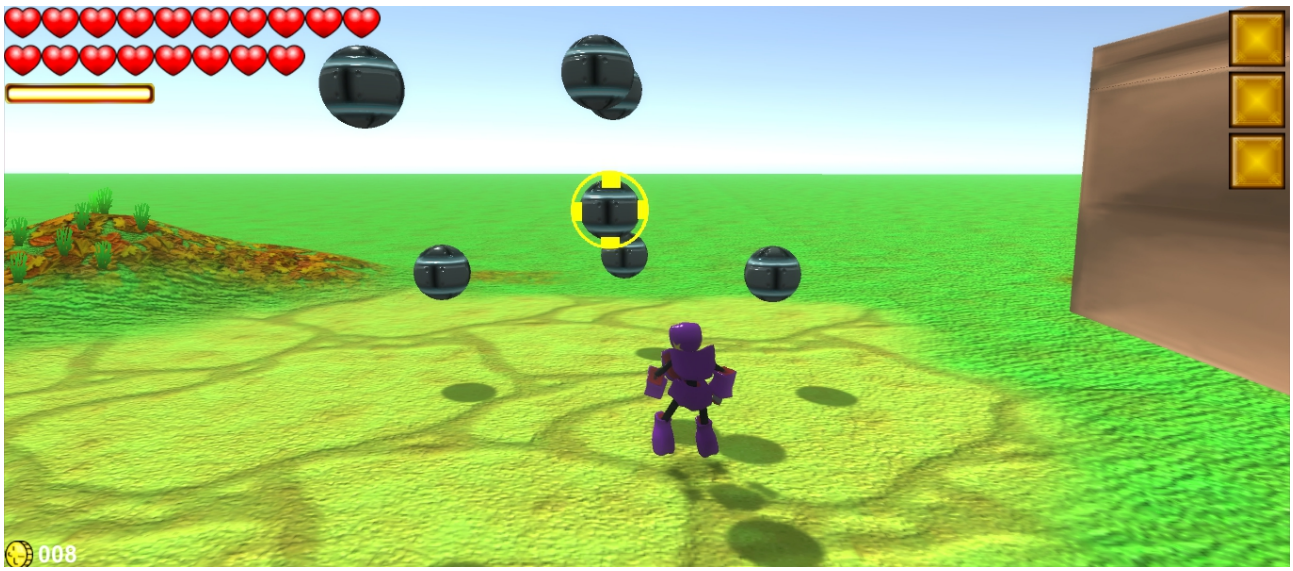
Here are the list of features included in the current version of this asset (each are described after); Since version 1.5, each feature can be turned on/off in the Inspector:

Camera Modes

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

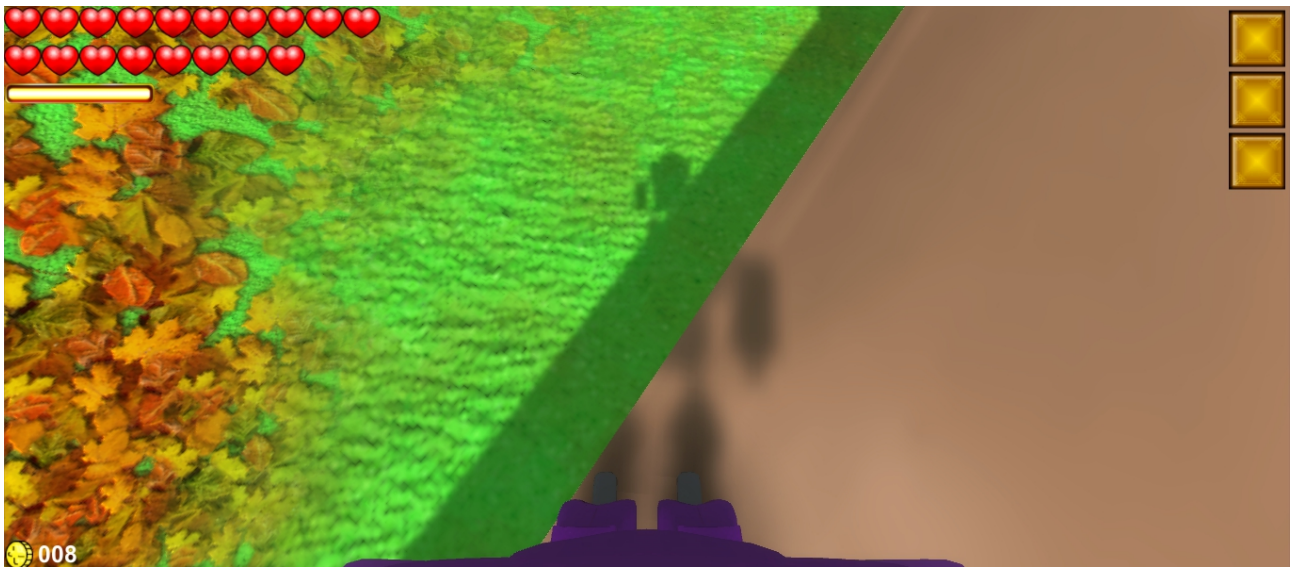


The orbiting (behind) mode is the default mode (like in the most third-person titles) which follows the player and, as the character changes direction, the camera rotates to look at the character. If the character moves to the left/right (if they hold in one of those directions), they will move in full 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 other camera modes can be turned off (via the Inspector) and this will be the only camera mode by default.



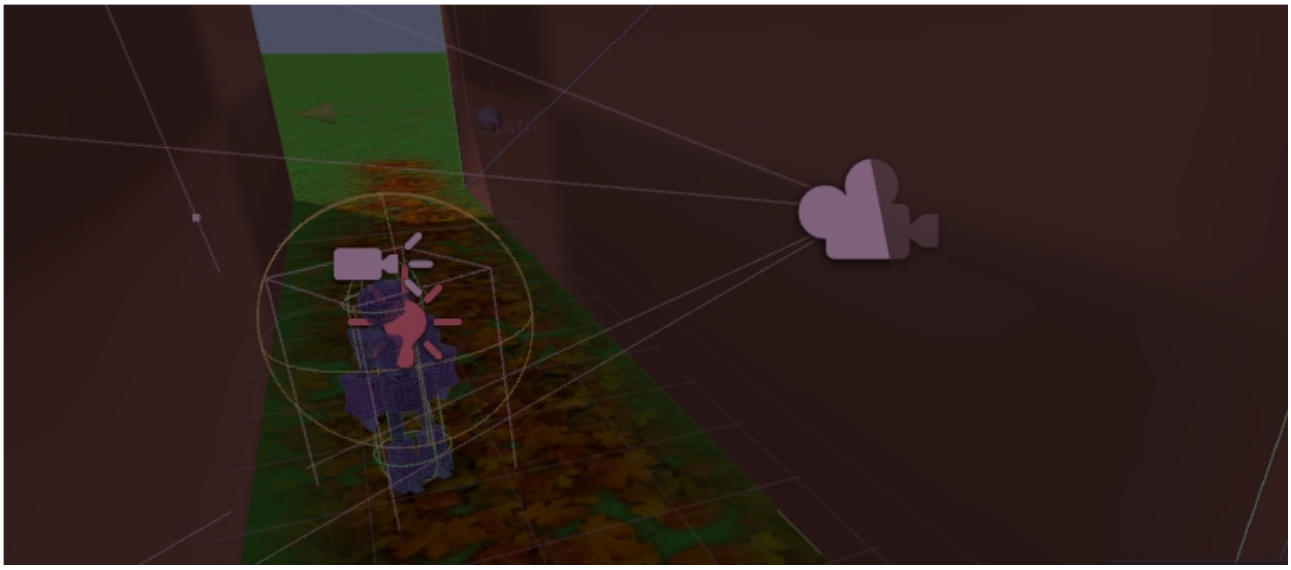
The targeting mode focusses directly behind the character while the targeting button is pressed. Pressing it quickly and letting go will focus the camera behind the player (almost instantly) and then revert to orbiting mode. Holding down the button will lock the camera behind the character and have the 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'). You can toggle whether or not the camera will focus on a target when targeting in the inspector.

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.

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 state, 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.

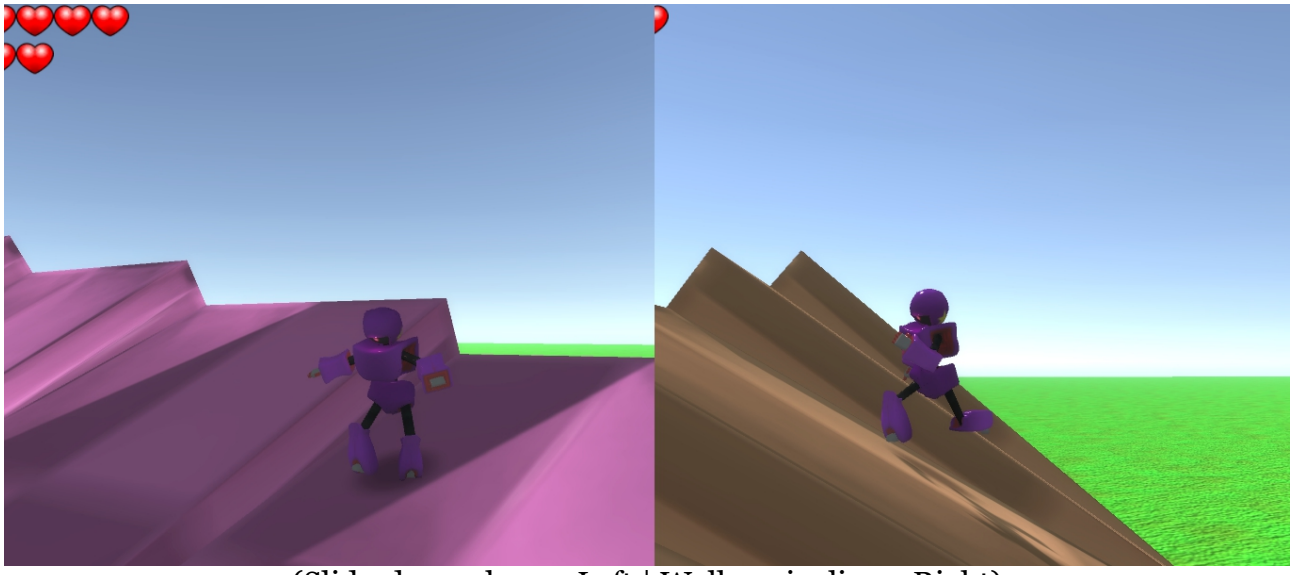
Character Movement

The 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.

Sprinting

When holding down the Left-Ctrl button (or the Left Analog Stick press) the character can move faster than their maximum speed. This additional 'sprint speed' is set in the inspector and as the character picks up speed, their animation changes too.

Incline Handling/Slope Sliding

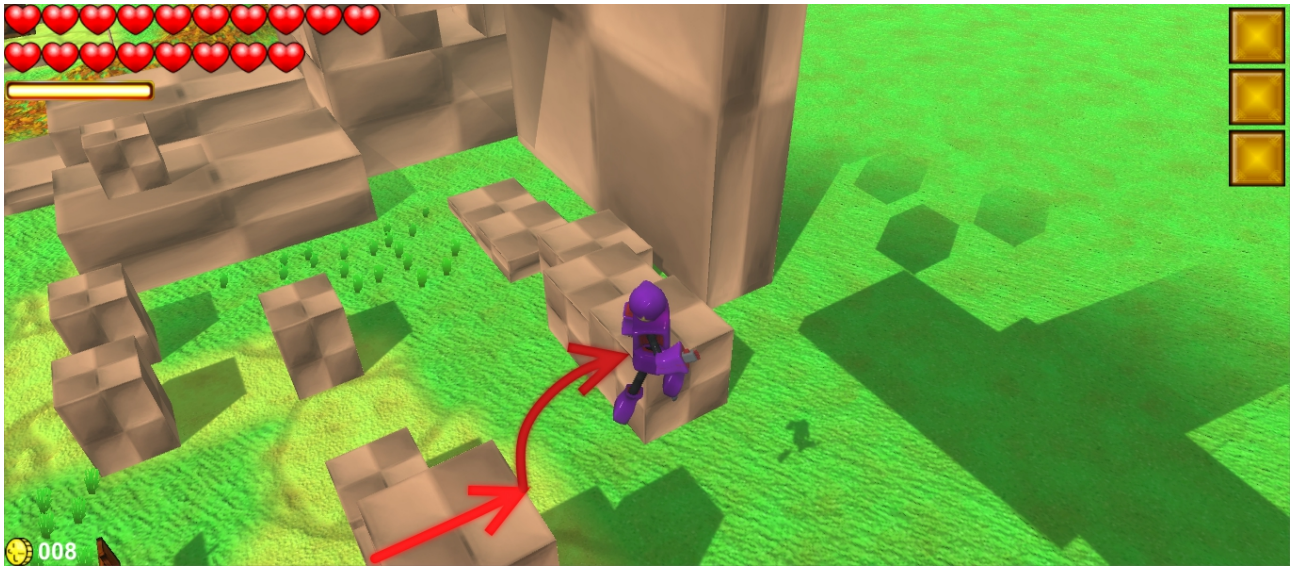


(Slide-down slopes: Left | Walk on inclines: Right)

Depending on the gradient of the ground and the collision layer of the collider (of the ground)
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 movements will adjust to run on the incline as if on flat/horizontal ground. Finally, if the collider detected is of layer 'Slopes', the character will slide down it regardless of the gradient of the slope. The character can run onto slopes, but it won't be long until they begin sliding down the slope. When sliding, the character can manoeuvre left and right slightly as well as hold back (to slow down) or hold forward (to speed up).

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

Character Jumping Off Ledges



The script checks for ground just behind the character when no longer 'onGround' and if that's the case, and if the character is moving fast enough, then a vertical impulse is put on the character and so they jump. After losing vertical momentum, the character is put in a 'falling' state which can be very slightly controlled (the player can still move the character in the (x,z) directions and the sharpness of this movement can be tweaked in the inspector); also at the point of jumping, the gravity acting upon the character in a straight downwards motion. Not moving fast enough will cause the character to slowly drop down to grab the ledge and hang off from it (if the user has allowed the ledge-grabbing function to be 'on'), transitioning to a 'ledge grabbing' state (which I will explain after the next feature).

Ledge Climbing

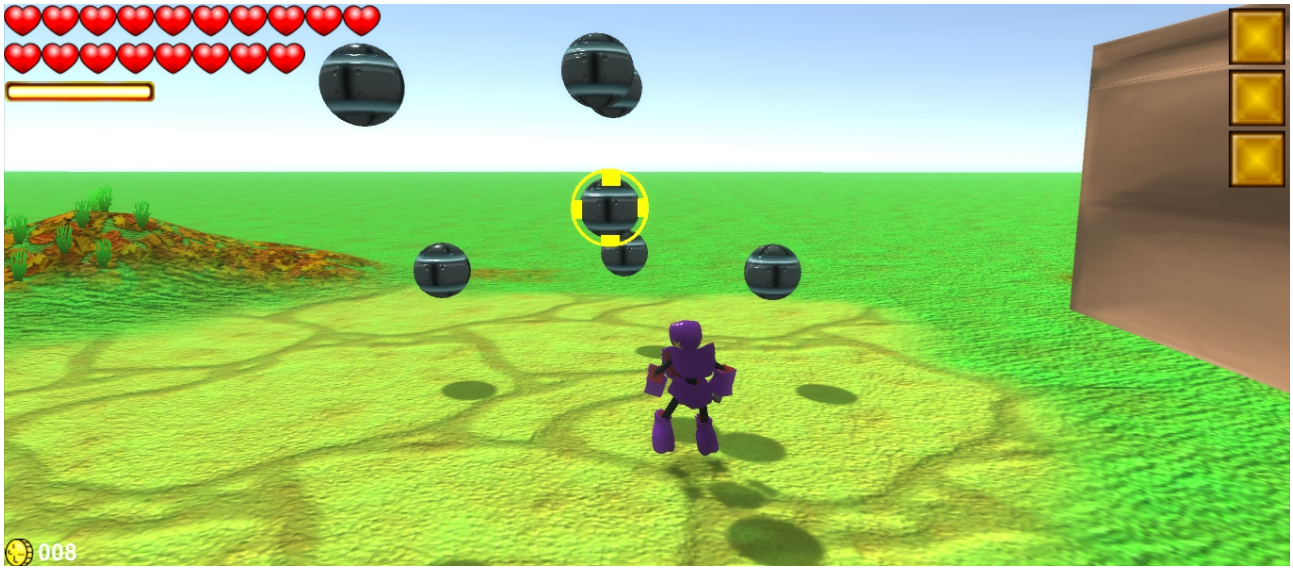
If there's any even land (land with a certain range of gradients) which the character can climb onto in front and depending on how high up this land is, the character performs the appropriate animation and climbs up the ledge. Ledges that are rather high but still grabable will have the character jump vertically and grab the ledge (explained in the next section) before climbing it.
As of version 1.5, there is no longer a 'flicker' that can happen when the character reaches the end of their climbing animation.

Note: The animations I have supplied in this package are my own, amateur work. The user (if wishing to use their own, better animations) can simply replace these.

Ledge Grabbing

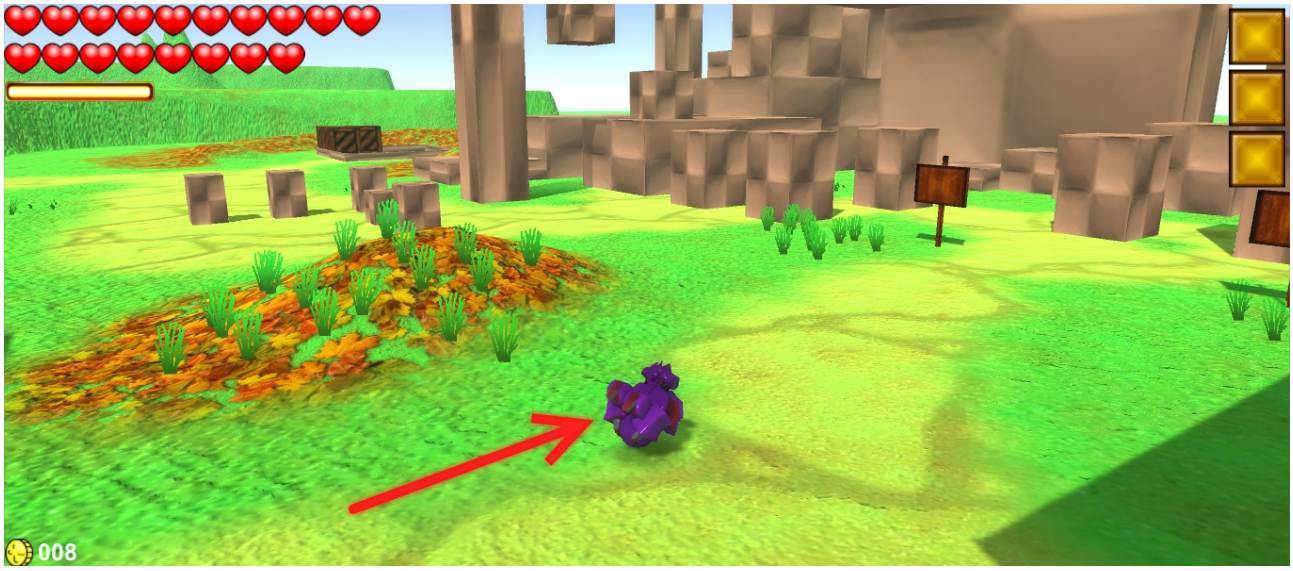
As mentioned in the section above, the character can jump up to grab a ledge. This works whenever a character is off the ground, is falling downwards, and a grab-able ledge falls in front of the character. When the character is in the grabbing state, they can move along that same ledge (left/right) and 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 target button is pressed (and if the feature is turned 'on'), 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.
The character also uses a smaller collider to roll through small gaps. If the character rolls under a collision and the collision above is too low to stand up under, the character will automatically go into a crouch (explained as the next feature).

Crouching and Crawling

If pressing the action button whilst stationary (as opposed to rolling), the character will crouch in place and assume a smaller collider (like with rolling) and can hence crawl around (while holding down the action button and crouching) and crawl through tunnels. If the jump button is pressed while crouching, the character will perform a large back flip jump and will no longer be in a crouching state.

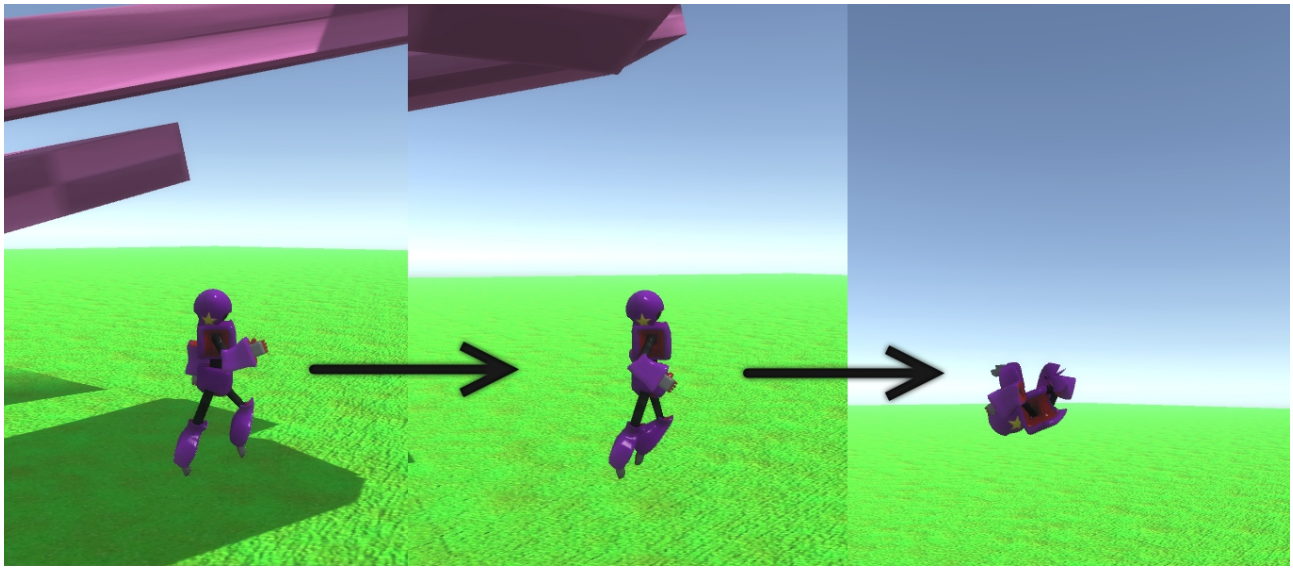
Targeting Jumps

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

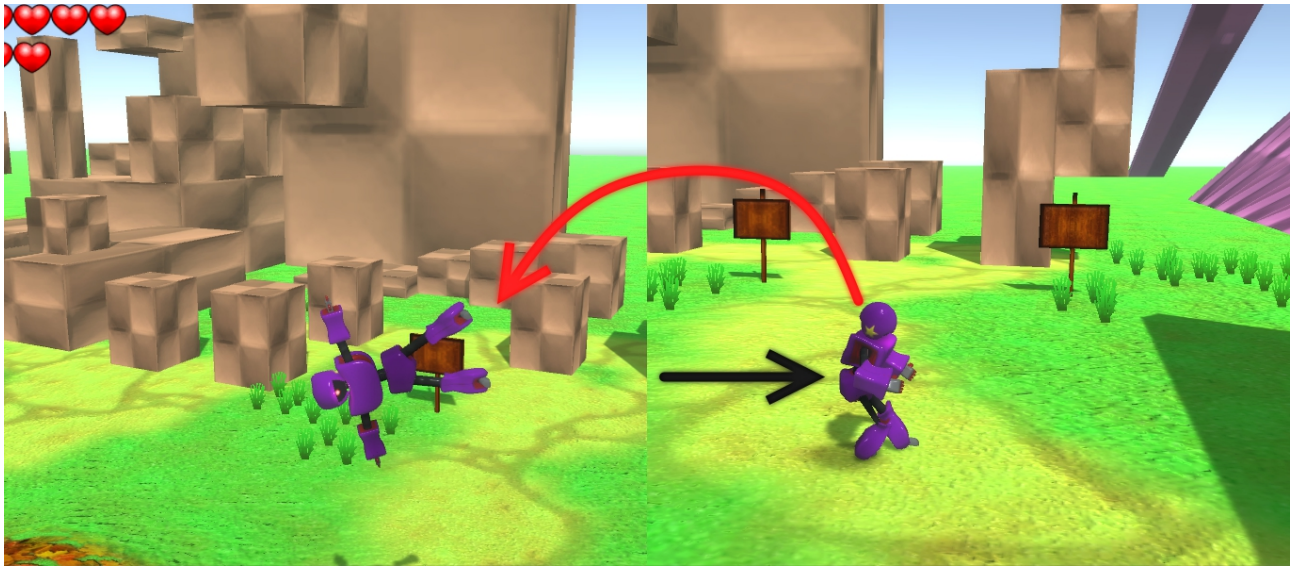
Pressing the left button/space bar will have the character jump up on the spot. This uses the same code that is used when the character jumps off a ledge, however more of an impulse is added and the direction that the character is moving in influences the direction of the jump. The character can be slightly controlled in mid air by default, but this can be customized. The character also performs a longer/higher jump if the jump button has been held down, otherwise the jump will be shorter.

Triple Jump



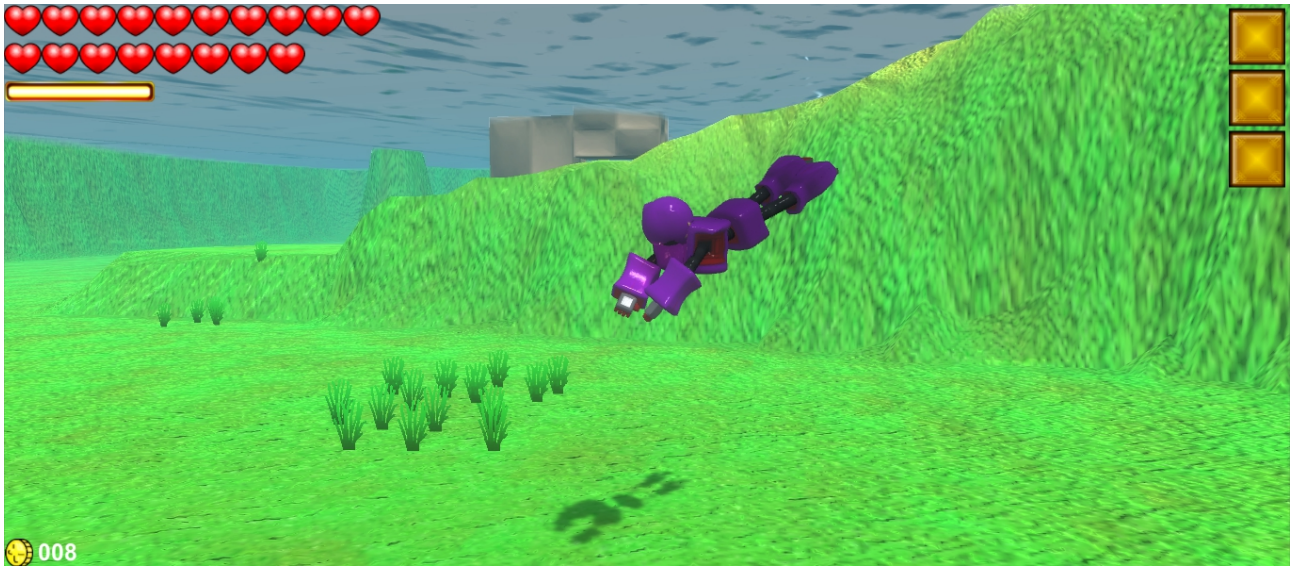
This is a separate feature from the on-command jump that can be turned off. It works by having the character jump when landing and while also still moving forward. This allows the character to perform three jumps in succession, each with a different animation and going higher than the last.

Skidding, Braking and Sideways Jump



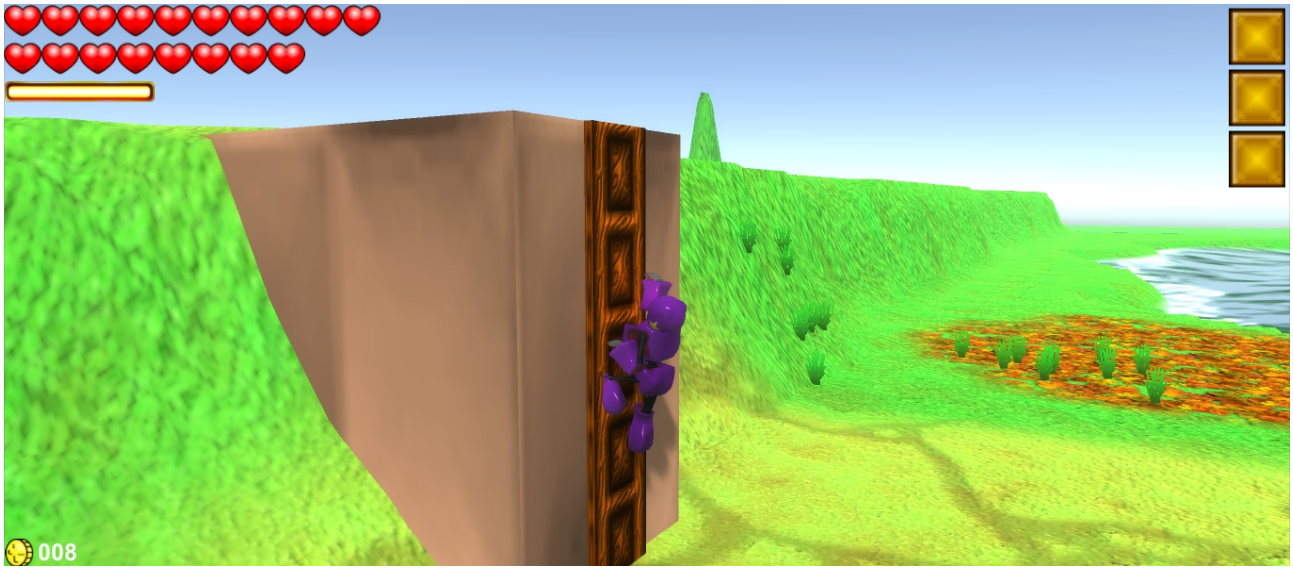
When moving at a high speed, the player may want to suddenly change direction or stop in place. When stopping in place, the character will brake (perform an animation). When changing direction while at a high speed, the character will 'skid' and change direction without concretely stopping. While skidding, if the player presses the jump button (and if this separate feature is on), the character will perform a large 'sideways' jump in the new direction that the player was forcing the character to move at.

Swimming



The character can jump into water and depending how fast they fall into it, the further they'll sink. As soon as they've slowed down, they will start automatically rising to the surface of the water. Simply walking into the water (from an incline) will have the character start floating on the waters surface. Once on the surface, the character can walk out (onto an incline), jump out by pressing the jump button or climb onto a ledge (emerging from the water) that's low enough. The character can either have the ability to swim or not have the ability. If so, pressing the action button will have them flip over and swim into the water slightly. Once submerged, the character can be rotated to face straight up or straight down, or rotate around the Vector3.up axis. Pressing the action button causes an impulse to swim quickly (basically stroke underwater) and keeping the button down has the character swim slowly. The character may also swim slowly by holding down the jump button. If the character cannot swim, they will simply dive for as long as the action button is being held down; letting go will have them reverse and re-surface.
The user can choose to not be able to dive as an alternative at all, meaning that the character can only paddle around on the surface. Additionally, the character can be set to not interact with the water at all and just act like normal (out of water) while submerged.

Climbing



The climbing mechanic works by having either a ladder object (made out of '1 by 1 by 0.1' tiles) or a fence object (also made out of tiles OR by having a large mesh/block) and having the character walk into them and then automatically attaching themselves to it. In the cast of ladders, the character will only attach via the front of the object, not on any other side of the object. Once on, the character may detach by pressing the action button. If the object is a ladder, the character can only climb up or down, and once having reached the top, will pull up onto the ledge at the top. The character will only pull up if there is a ledge behind at the top of the ladder/fence, otherwise they will reach the top and not be able to move up. Climbing down and touching ground will have the character detach and start walking around on the ground. When the character's on a fence, the can climb in all four directions. If there is a ceiling above, the character will not be able to keep climbing upwards. If there are no fence tiles to the left/right, the character will not be able to keep climbing in that direction either. The character can move onto different gradients of a fence if the tiles are attached; fences can have sharp corners or wrap around like cylinders and the character will adjust to these different gradients. If you are coming to a ledge that has a ladder or fence beneath it, the character will perform their 'grab beneath' function and then attach themselves to the climbable object beneath.

Box-Pushing

The box-pushing mechanic works by walking up against the object and pressing the action button (as the prompt appears) and holding it down to be in a 'grab' mode. While grabbing, you can either move towards to or away from the box to either push or pull it respectively. The forward and backward directions are relative to the way the camera is facing (so if the camera was behind the character, pushing up will push; if the camera was to the side, such as in the picture above, pushing left will push). The boxes move an exact amount each time which can be changed in the inspector (as well as the pushing speed) and move at a velocity so that they interact with other colliders if any would get in the way. The boxes cannot be climbed like ledges, however, though they can be grabbed while falling and are fine to walk on/use as ground.

HUD Display

The HUD (Heads-Up Display) in this asset features a classic-styled hearts-for-health system, a magic/stamina bar/meter (supports MULTIPLE bars if the user wishes), money counter and some hotkey buttons on the far right. The user simply has to position a heart, bar(s), money icon and a hotkey icon on the HUD GUI Canvas, drag these elements into the inspector and set all the quantities/arrangements there and running the game will draw out the HUD the way you want it to (e.g. the image above only needed me to put a single heart in the corner, the script drew out the rest in the parameters I specified).
Each of these sections is customizable in the sense that quanities, sizes and positions can be changed via the inspector in the Unity Editor and it will all adjust accordingly:
The maximum number of hearts, hearts acquired and hearts per row can all be adjusted; losing/gaining health will not be an instant visual change but rather you will see the hearts re-fill or deplete. The speed at which this happens can be changed, so it can be instant. There are measures taken that prevent the game from crashing when falling below zero hearts or going above the maximum (the same ideals apply for everything else in the HUD system).
The stamina bars can each be individually set up to have different maximum values, positions, sizes, regeneration speeds, automatic regeneration functions and change (add to/subtract from the values) speeds.
Money can have a maximum amount, and depending on the number of digits in the maximum amount, there will be zeros in front of the amount of money if the amount of money doesn't have as many digits as the maximum (e.g. if the maximum is 1000, and the amount of acquired money is 82, the counter will display '0082'). These values can all be adjusted dynamically and so do not have to stay at one maximum when the game is running (so upgradeable wallets are possible).
The hotkey buttons on the right work with the Inventory system which I will explain as the next feature.
There are also two pick-ups I have included from a 'Pickup' prefab that will regenerate health and money upon collecting them.
Finally, the HUD script contains public methods to use in your own games such as 'AcquireNewHeart()', 'UpgradeMoney()', 'DrawHUD()' and 'HideHUD()'. Each element in the HUD can be turned on/off individually in the inspector so everything I've supplied

doesn't have to be in your game.

Inventory

The Inventory system works as a pause menu as the game freezes in the background and the inventory menu pops-up. If the user wishes not to use the Inventory feature, the hotkeys will not be drawn and the game can still pause without displaying an inventory. Much like with the HUD feature, all you need to create your inventory is to position certain elements on the Inventory GUI Canvas, direct the inspector to these elements and the script will draw the rest at runtime with the parameters the user has set!
The items (represented by a Bow, Bombs, a shovel and some colourful place-holder spheres) will be displayed as GUI Buttons and can be clicked-on and assigned to one of the hotkeys on the right OR the items can be navigated through with the D-Pad on a gamepad (or the IJKL keys) and then pressing the corresponding button will assign it to the corresponding hotkey.
Shown in the image are 3 hotkeys, they can be used by the '1', '2' and '3' keys on the keyboard or via the right bumper, top button and right button on a gamepad. There are three keys so that it doesn't use too many of the buttons on a gamepad, however the number of hotkeys can be increased or decreased to the users liking and assigned to any gamepad button (or none at all if gamepad usage isn't part of your projects plan).
The same item cannot be assigned to more than one hotkey as attempting to do so will swap whatever is in the hotkey it is being assigned to with the hotkey that the item is already on.
Items can be set to either 'hasItem' or not and if not then they will not appear in their space in the inventory until that boolean value is set true. Each item will be given its own place in the inventory menu depending on its place in the item dictionary that the inventory uses. If it isn't acquired, there will be an empty spot where the item should be and will be skipped when navigating through the items via a D-Pad.
The items quantities are shown in the lower-left corner of each item and this carries over to the hotkeys. When in gameplay, pressing the buttons that the hotkeys use will deplete the items quantity until they reach zero. I have included two pickups from a 'Pickup' prefab that will replenish the quantities (of the bow and bombs) until they reach their specified maximum quantity (which can all be customized by the user).

Proper usage of items and weapons (e.g. with animations and projectiles) will be

implemented in the final 2.0 version of this asset.

NPCs / Cutscenes

NPCs simply require the NPC script and the NPC GUI Canvas where you arrange the text-box position, the text position and replies position (much like with the HUD and Inventory as the script will draw out all the reply buttons for you). An NPC can have a 'targetable' part (game object) that you can focus on with the targeting feature and the zone in which you can interact with the NPCs is a Sphere Collider (Trigger) which also checks that you are facing the correct way. A prompt shows up above the sign signalling that the character can interact with it, and once they do, it will display whatever dialogue you have written in the inspector. The script comes with a 'Dialogue' dictionary in which, for each element, the user can (all within the inspector in the Editor):

-Write the dialogue for that specific page,
-Give the camera a position to be at during that specific page of dialogue,
-Give the camera a game objects transform to look at during that page,
-Specify an amount of replies (or alternatively specify the next page of dialogue),
-Write out the content of each reply button,
-Specify the resulting dialogue page for that reply;

This allows for branching conversations and limitless dialogue for each NPC.
The text is also written out at a specified speed (can be set in the inspector) and the user can choose to have the HUD hidden during these 'cutscenes'.

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