Complete Top-Down Controller

Introduction

Thank you for purchasing the Complete Top-Down Controller. This asset was created with the intention of providing you with a huge head-start in creating your very own top-down game. Get started quickly and easily with all the core mechanics that you'll need to navigate and interact with your game world.

After a short initial set up, you'll be able to add an animated Player Character, Touch Screen Mobile Controls, Teleport Tiles, Interactable Objects, Pushable Blocks, Music, and more to your game! Use the "WorldMap" prefab as a template or use your own existing tilemap to start adding items and creating an interactive world for your players!

Included

Included in this package is an Example Scene, Animation Controller Templates, Sprite and Tilemap Images, Sounds, and Prefabs. The prefabs' sprites can, and should, be changed to fit your game's aesthetic better. The images and sounds included for the examples are NOT exclusive to this package. See below for more details:

Images & Sounds

- Player Character – Created by AxulArt and free for any project. There are also 2 other player sprite variants available.
- Tilemap and Music – Created by pixel-boy and free for any project. This asset includes a VERY small sample of what is freely available for download.
- Touch Screen Input – Created by greatdocbrown and free for any project. The full pack contains variants for all major consoles.

Prefabs

- **Player** - This single prefab can support either Rigidbody or Grid movement and can be modified with custom sprites and animations.
- **WorldMap** - This is a template to help show how layers should be set up in your existing tilemap. You can also use this as a start for a new map if desired.
- **LocalTeleportTile** - A teleport tile that will send the player to another location in the same scene. Can be used to change music as well.
- **SceneTeleportTile** - A teleport tile that will send the player to another location in a different scene. Can be used to change music as well.
- **Item-Click** – This item needs to be walked up to and interacted with in order to execute code.
- **Item-Placeholder** – This item contains everything needed to create a new item, just change the sprite and place it in your scene
- **Item-Push** – This item can be pushed around the WorldMap by the player, does not display a message when interacted with, and will not run code.
- **Item-PushClick** – This item can be pushed around the WorldMap, displays a message, and runs code when interacted with.

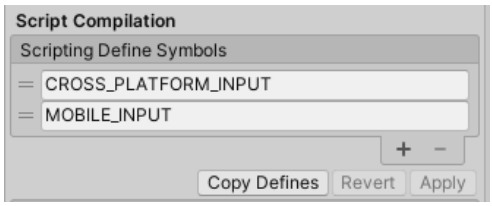
- **Item-PushSign** – This item can be pushed around the WorldMap and displays a message when interacted with but will not run code.
- **Item-Sign** – This item cannot move at all, blocks player movement, and displays a message when interacted with, but will not run code.
- **Item-Touch** – This item will execute code as soon as the player walks into it.

Part 1: Initial Set-Up

Once you've imported the asset, there are a few additional steps to get everything up and running. In order to simplify this process, it is broken down into multiple sections and sub-sections so that you can reference each as needed. While there is a lot of documentation here, it should solve any questions or issues that you may have. For a simplified version of these instructions, view the Quickstart ReadMe instead.

Important Notice

This asset uses Unity's old input system. When playing your game for the first time, you may see an error stating, "Exception: This is not possible to be called for standalone input." This can be resolved by selecting: ***Edit -> Project Settings -> Player -> Other Settings*** and adding 2 more "Scripting Define Symbols" to your project. Ensure they appear exactly as shown in the image to the right and click the "Apply" buton.



Step 1: Tags

Three tags need to set up for this script to work properly: ***Player / Walls / Item***
These tags are used by the various scripts to find certain objects, or to confirm the object that the player is interacting with. These tags should be added automatically when the asset is imported, however the package relies on them, so they deserve a mention. The parent player object should be tagged as "Player". Anything that uses the "TDC_InteractableObject" script should be tagged as "Item". And a single layer on your Tile map should be tagged as "Walls". Everything else in your game, including Teleport Tiles and other Tilemap layers, can be left untagged.

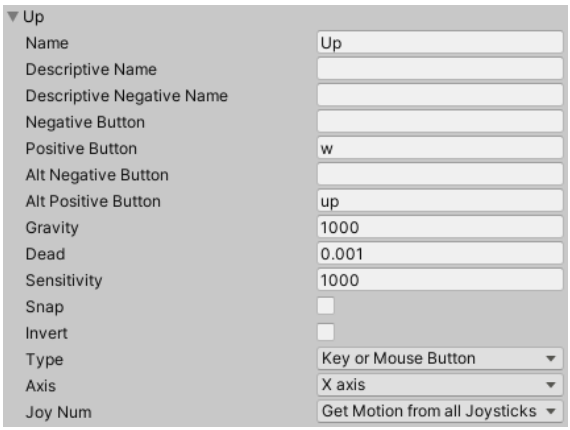
Step 2: Inputs

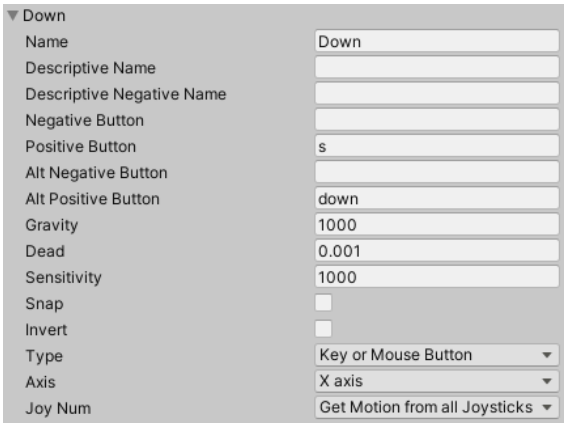
There are two options for adding Inputs to your game. The first option is to manually add each button's control input to the Input Manager. This method should be used if you already have a partially completed game, as it will not overwrite any existing Inputs that may have been added already. The second option is to download the "InputManager.asset" file and overwrite the existing file in your project. This is quick and simple if starting a project from scratch, or if you don't care to overwrite your existing Inputs.

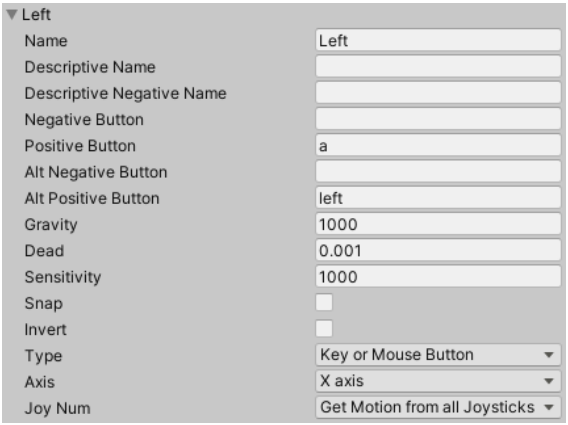
Method 1: Manual Entry

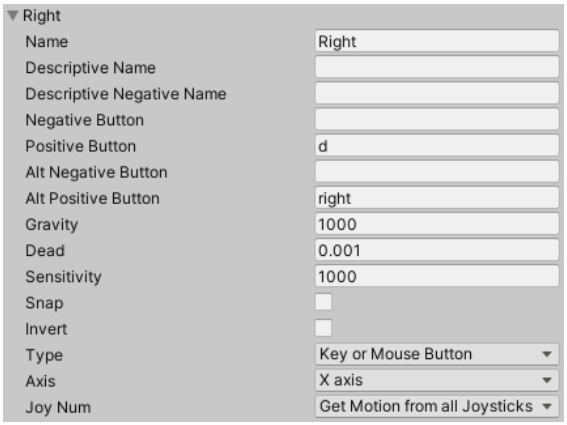
If you choose to add each input manually, then you will need to add 10 additional Inputs to your game. This can be done by

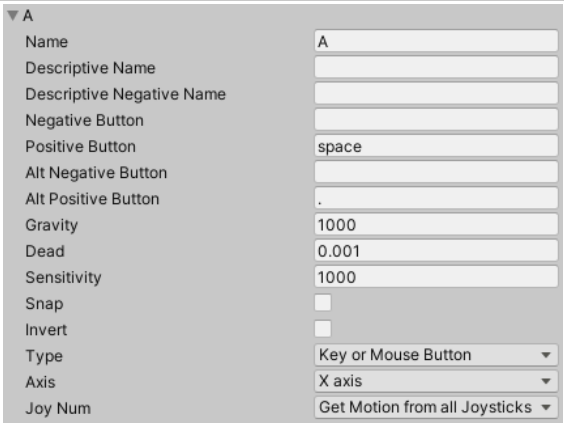
navigating to: ***Edit -> Project Settings -> Input Manager***. In this screen you will need to add 10 to the existing "Size" variable. In the example, it will increase from 30 to 40.
The 10 additional Inputs will be added to the bottom of the Input list and the values of each can be adjusted. Below are images to show what these values should be set to. Notice that both the "Exit" and "Return" inputs have no actual values set in their fields. This is because they are not tied to any actual keystrokes. They exist only as onscreen buttons, but still need to have an input on this list in to avoid any error messages. The 'Positive Button' for any of the other inputs can be any button you choose. The values set below are just what was found to be the most comfortable on PC. Once all of these inputs are updated, you can move on to Step 3.

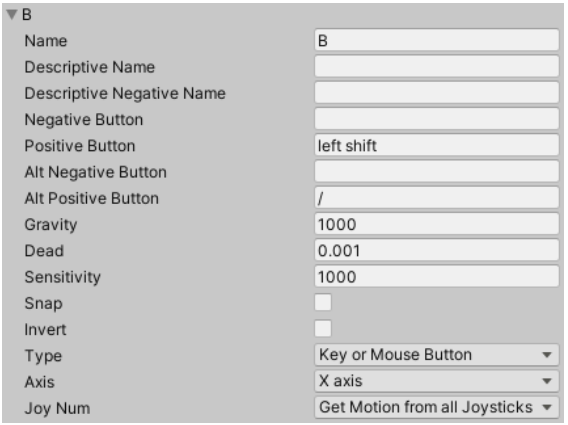








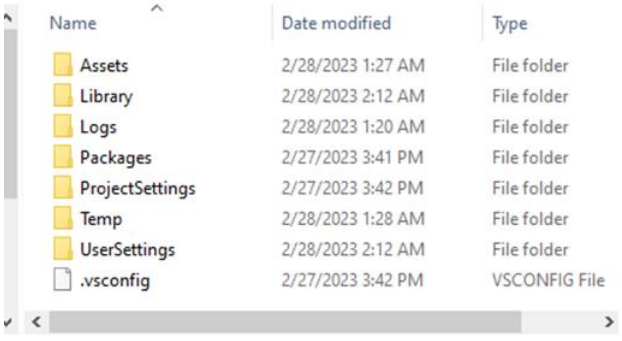




Method 2: "InputManager.asset" file

For quicker setup, the "InputManager.asset" can be downloaded and placed in your project folder. This file can be used to automatically set up inputs for your game without adding each individually. The only downside of this is that it will overwrite any existing Input data you may have added previously. To use this method you will have to go outside of the Unity Editor.

1: Right-click anywhere in your project window and select "Show In Explorer" to view your project folders outside of the Unity Editor. Navigate up through the parent folders until your window looks like the image to the right.



2: Download the "InputManager.asset" from here and place it in the folder named "Project Settings". Overwrite any existing files and replace them with this new downloaded file.

3: Now, inside of the Unity Editor, navigate to: ***Edit -> Project Settings -> Input Manager*** and verify that you have all of the inputs that are mentioned above in Method 1. If everything is there, then you're ready to move on to Step 3.

Step 3: Player

Now that you have your Inputs set up, you can drag and drop the "Player" prefab object into your game scene. You can choose for the player character to use either Grid-Based or Rigidbody-Based movement styles. Make sure that there are no other Cameras in your game scene or, if your scene requires additional cameras, make sure that they do not have an "Audio Listener" component attached to them.

Grid-Based movement is selected by default and will lock the player into moving in 4 basic directions, Up, Down, Left, and Right. The player will also move one full block with each movement input that is given. This ensures that the player character is always standing on a whole number coordinate when not moving. Grid-Based players do not have the option to use the touch joystick for input.

When the player is using Rigidbody-Based movement, they can free roam the map in any direction and are not locked into a squared grid or forced to make a full unit of movement each time. Rigidbody-Based players can have a BoxCollider of any size greater than 0 (although it is recommended to cover the left and right sides of the sprite fully to prevent clipping.) The touch joystick was created specifically for the Rigidbody-Based player to control the direction of movement more easily. Keep in mind that the D-Pad can still be used if preferred.

Another main difference between these options is the way that they handle pushing items. The Rigidbody-Based controller can push objects freely in any direction. The player's difficulty pushing these items will depend on the mass and drag options for the rigidbody components attached to both the player and the object being pushed.

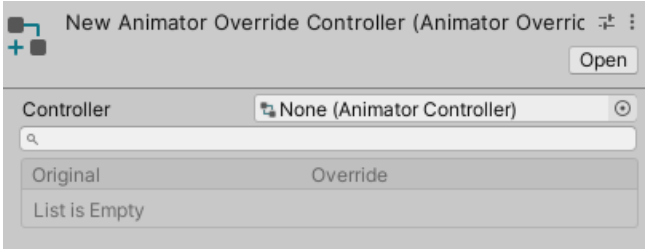
Grid-based players will move items a single block at a time. Walking into a pushable object will push that object one square in that direction while the player stays in the same position. The player can not, however, push two blocks at the same time. In this case neither block would move until one is pushed out of the way. For Grid-Based players, pushable objects work the same way that boulders did in the original Pokémon games. It is reccommended to use the Grid-Based Controller if you plan to make puzzle sections in your game.

You can now add the "WorldMap" prefab to your scene if you haven't already (or add the "Walls" tag to an existing tilemap layer) and press the play button to freely move your player around the map. The world map will be discussed more in depth in step 5. At this point, you'll probably want to change the way that the character looks which leads us into…

Step 4: Animations

Animations will control the way that your player character looks while moving and standing still. The only time that the "Sprite Renderer" component on the "Player" prefab will be used is when the game is first loaded, and before the player makes any movements. This can be changed to whatever sprite you would like to appear when the player first starts the game. After the player moves however, the Animator Override Controller will take over and determine the way that the player character looks by using its specified animations.

Each player type has a different Animator Override Controller that they use. These can be seen by toggling the "UseRigidbody" option on the "TDC_Movement" script. As you change this option, you will see the Animation Controller variable change as well. To customize the look of your character, you will need to create an Animator Override Controller, as well as different animations to assign to the controller.



To create a new Animator Override Controller, right-click anywhere in the project window and select ***Create -> Animator Override Controller.*** Once you have created the new controller, click it and view the properties in the inspector. Choose the appropriate controller (GridAnimationController or RBAnimationController) for your player type under the "Controller" variable and you will be presented with all the animation clips that are connected to that controller. Use this as a guide to see what animations will need to be created.

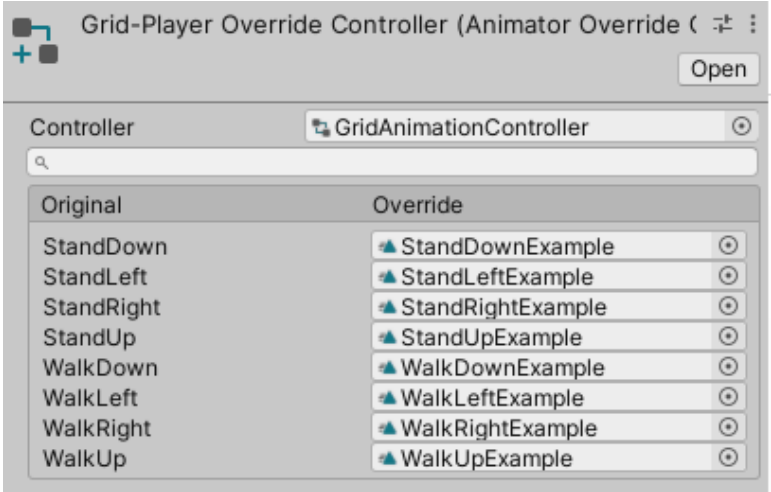
There are 8 animation clips that will be shared between both the Grid and Rigidbody player types. There are also 8 additional animation clips that are only used by the Rigidbody player for diagonal movement. The chart to the right will more simply explain which animation clips are needed for which player type.

	Grid	RigidBody
Walk Up	X	X
Walk Down	X	X
Walk Left	X	X
Walk Right	X	X
Stand Up	X	X
Stand Down	X	X
Stand Left	X	X
Stand Right	X	X
Walk Up Right		X
Walk Up Left		X
Walk Down Left		X
Walk Down Right		X
Stand Up Right		X
Stand Up Left		X
Stand Down Right		X
Stand Down Left		X

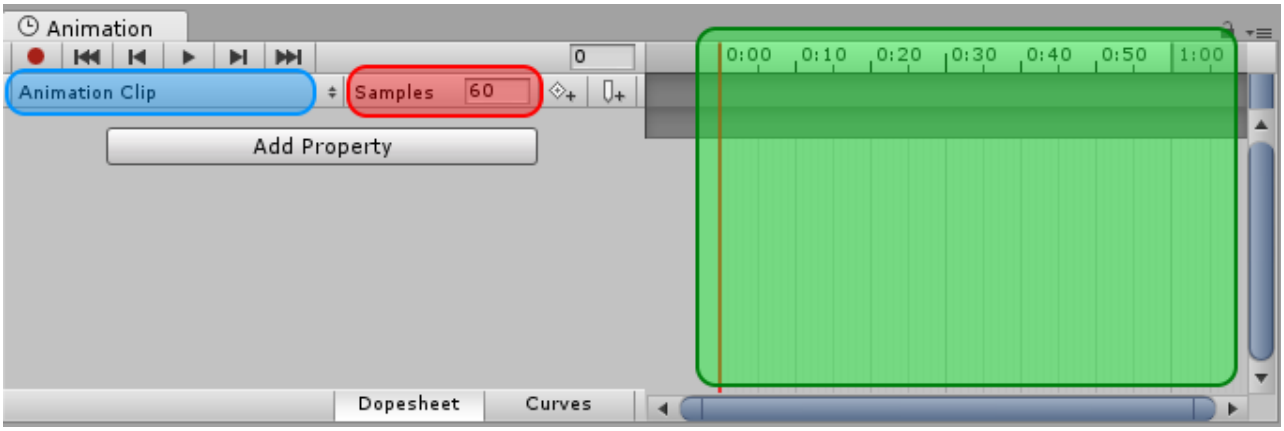
An animation will be needed for each movement listed, based on the movement type of the player. Animation clips can be created by right clicking anywhere in the project window and selecting ***Create -> Animation.*** It's recommended to create a new folder to store all of these animations and controllers so that things don't get too messy and you can easily locate the files for your player.

After you've created a new animation clip and named it appropriately, (it can be named anything as long as you can identify it) you can assign it to the appropriate slot in the Animator Override Controller. Repeat this step for each of the animation clips that are needed for your desired movement type. Don't worry about adding any actual animation frames yet, just create the empty placeholders and assign them to the override controller for now.

When you finish it should look like the image to the right (depending on your Controller type). Make sure to double check that the “Controller” value is correct based on the type of movement you decide to use for the player. You can now click on the “Player” in your scene and drop your Animator Override Controller into both the “Controller” slot on the “Animator” component, and in the “Anim Override Controller” variable on the “TDC_Movement” script.



Once you have added the Animator Override Controller to the player in your scene, you can begin editing the animation clips with your custom sprite images. Double-click any of the animation clips that you created to open the Animation Window. In this window you can grab individual sprite images from your project folder and add them to the timeline to create an animation.



In the above image, the blue section represents the name of the selected animation that is being edited. Having the player selected in the hierarchy will allow you to use this drop down menu to see all of the animations being used by the Animation Override Controller. The red section is the number of animation frames that will be shown per second. And the green section is the animation timeline where you will drag and drop your sprite images to make up your animation. These values can be set to whatever you prefer and can be adjusted as needed to find what looks best for your game.

As a reference point when creating animations, the “Player” prefab only uses 3 frames of animation (Left Step, Center Step, and Right Step) for each type of movement and the animations are played at a sample rate of only 12. Each of the “Stand” animations uses only a single frame to give the impression that the player is standing still. This may or may not be what your game requires. For example, you may have much more complex walking animations or full idle animations rather than a single frame.

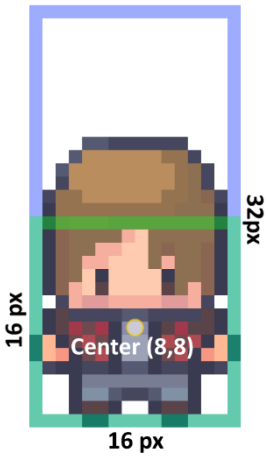
Once all of your animations have been edited, you can press play and see your new character working in your game. If you notice any issues with the look of your character, then keep reading below, otherwise you can move on to step 5.

Additional Notes

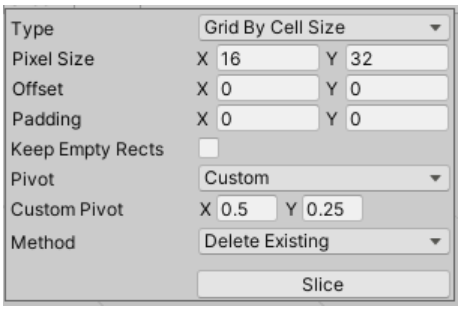
1.) A few things to consider when editing your animation clip is the movement speed of your player, the sample rate of the animation, and the framerate of your game. All of these can affect how your player looks when moving. For example: if the framerate is too high, then your player will look as if they're gliding around the map and not playing any animations at all. If the animation plays too quickly (high samples value) or if the player reaches the next square too quickly (high movement speed), then the animation may cut off half-way through when the player stops, or only play a few frames of the animation before being called to play the animation again.

The player's movement speed and the game's target framerate can be adjusted manually under the "TDC_Movement" script that is attached to the player. Ideally, the player should complete a full animation cycle, or close to it, in the time that it takes for them to move to the next grid square. This isn't as important for Rigidbody-Based players but should still be considered if you're having issues with the way your animations look.

2.) If the player seems smaller or larger than expected, then check on the "Pixels Per Unit" value that can be found when selecting your spritesheet in the project window. This defaults to a value of 100, but that's most likely not what it should be for your game. This value should be set to the width of your player's sprite.



3.) If you find that the feet of your sprite are appearing on top of walls when you are standing behind them, then the center of your sprite needs to be adjusted. This is most likely occurring because your sprite is not a perfect square, and the center is not where it should be.



In order to correct this, you can open the Sprite Editor tab in the Unity Editor and change the "Pivot" of your sprite to Custom. This will allow you to change what is considered the center of your sprite. If you have many sprites to correct, you can do it quickly by replacing all of the existing sprites on your spritesheet. Use the image above as an example of how to measure your character.

Take the width of the sprite and create an imaginary square at the bottom of it. Then find the center point of the bottom square by dividing the width and length by 2. Now divide this center point by the total width and height of your sprite to get the "Custom Pivot" values for the image to the left. For example: X = 8/16, Y = 8/32. This can then be simplified to a decimal as: X = 0.5, Y = 0.25

Step 5: World Map

Now that you have your player moving, you can start editing the game world. If you already have a tilemap in your scene, then you can simply add the "Walls" tag to the layer that contains objects the player can't pass through. If you don't have any sort of map set up, then you can drag and drop the "WorldMap" prefab into your scene. This prefab contains 5 different layers that are meant to show you how to set up the layers that will make up your tilemap. The chart to the right shows the "Order in Layer" number for each layer as well as for the player.

Layer Name	#
Ground	0
Ground Decorations	1
Walls	2
Wall Decoration	3
Player	5
Above Player	6

This variable can be found on the "Tilemap Renderer" component ("Sprite Renderer" for the player) of each layer and can be adjusted if needed. You can either follow the prefab as a guide to configure your existing tilemap or use the "WorldMap" prefab and edit it. If you choose to build from the prefab, then you should right-click it in you hierarchy and select: ***Prefab -> Unpack***. This will keep the prefab for you to use again later if needed but allow you to edit the WorldMap freely in your scene.

Note: the "WorldMap" game object has its transform set to (x: -0.5, y: -0.5, z: 0), while the rest of the child objects are set to (x: 0, y: 0, z: 0). This is done so that the player and items will always move to a whole number coordinate if needed, rather than a fraction. The movement script relies on this regardless of when movement type you choose to use. If you are using an existing tilemap that was created, make sure to adjust its transform to match this.

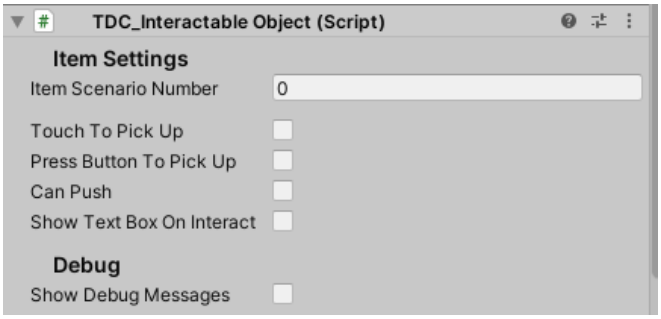
Part 2: Interactable Objects

Now that you have completed the initial set-up, you can begin adding items for your player to interact with. There are many prefabs that you can choose from to quickly add items to your scene, but you can also create your own with the "Item-Placeholder" prefab. All of the item prefabs have the same 2 scripts attached to them, "TDC_InteractableObject" and "TDC_ItemScenarios".

"TDC_InteractableObject" should **NOT** be edited in any way at all, as it is the core behavior script for in-game items. "TDC_ItemScenarios" can be edited as needed to add more functionality to items. Read below for a more in-depth explanation of both of these scripts.

TDC_InteractableObject

This script should be attached to any items in your game that the player can interact with. It is used to determine how the objects should move throughout your game scene as well as what the player needs to do in order to pick up the item, if they can at all.



To the right is an example of how the script appears when attached to a game object. These options can be mixed and matched to make the item behave as needed in your game. The only exception to this is the "Touch To Pick Up" option. If this option is

selected, then the “Press Button To Pick Up” and “Can Push” options will be automatically marked as false when the game starts. Read below for an explanation on the “Item Scenario Number” variable.

TDC_ItemScenarios

This script is used to determine exactly what should happen when the player interacts with an item. An integer is set in the inspector of “TDC_InteractableObject” which is sent to the “TDC_ItemScenarios” script when the item is picked up. This number is then used by “TDC_ItemScenarios” to decide which function should run. Open “TDC_ItemScenarios” in your script editor and you will see additional notes on how to add functionality to your script. Item scenario 0 is the default option that is set for all items. Scenario 0 will remove the object from the scene and state that the number should be changed away from the default scenario.

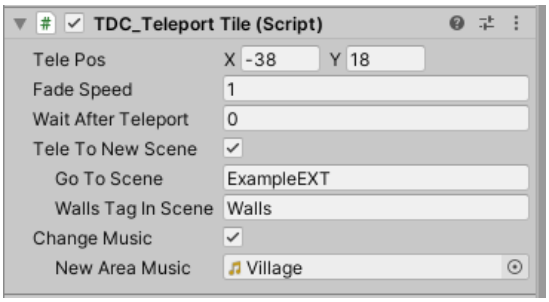
At the top of this script there are 2 commented areas that are labeled: Scenario A and Scenario B. These templates intentionally use letters instead of numbers so that an error will be thrown if they are not changed. These commented scenarios should not be edited at all. You should instead copy the template, paste it, uncomment it (Ctrl+K+U), and then begin making your edits. After you paste the template, locate the line that begins with: ***if (ScenarioNumber == …*** and change the letter to an integer number value that is not being used already by another scenario.

This has been done once already for you as an example in the script, but can be done as many times as needed for as many items as you’d like to add to your game. You can also add any public variables that may be needed the same way you normally would for any other script.

Once all of your edits are done, you can close the script editor and update the “Item Scenario Number” on the “TDC_InteractableObject” script that is attached to the item that you want to change. This will make your new function be executed whenever this item is picked up by the player. Additional notes for this process can be found in the comments of the “TDC_ItemScenario” script itself.

Part 3: Teleporting

Another prefab available to use is the Teleport Tile. This object is a trigger box that will fade the screen, change the music, and teleport the player to a specified location. These can be used to move the player to another scene as well if needed. Teleport tiles can be placed anywhere in your scene, but should be near doors, or other clear entrances for the player.

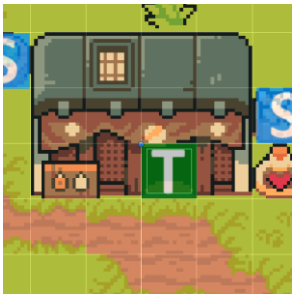


Because these tiles are able to teleport the player between scenes, there is no need to add a “Player” prefab to each scene that makes up your level. The player only needs to be added once to the starting scene of the level. You may notice a warning in the console regarding 2 audio listeners when returning to your starting scene. This is normal due to the player being active in your starting scene already, but is corrected automatically by the script and can be safely ignored.

When setting up teleport tiles, you will notice two separate prefabs in the folder, “LocalTeleportTile” and “SceneTeleportTile”. These both use the same script and are essentially the same object. The only real

difference is the color of the placeholder icon. Green is used for teleportation inside of the same scene and red is used for moving the player to a different scene. These are only used for identification for the user so that tiles can easily be identified. The image used has NO bearing on where the player can teleport, that is determined by the script itself.

Additional Notes



1.) When setting up the "Tele Pos", make sure that you do not set it to the coordinates of another teleport tile. This will trap your player in a continuous loop and will most likely crash your game. For example: when exiting the house in the image, the interior's exit Teleport Tile should move the player to the square directly under the Teleport Tile that is shown.

2.) In the image to the right, the teleport tile is set to be on top of the doorway. The tile underneath the Teleport Tile needs to be set as walkable in order for the player to be able to use it.

3.) When editing an interior space, the exit's teleport tiles should be embedded in the wall instead of in front of it. If the exit is in front of the wall, then the player may accidentally walk into the side of the trigger box, leaving the building when it is not intended.

4.) Note that in the image to the left, the teleport tile is stretched to be 2 blocks wide. This should be done anytime that multiple teleport tiles would be used next to each other. If the player walks between 2 teleport tiles, it will activate the teleport functions twice and attempt to move the player both times, which will crash your game. Instead, use a single Teleport Tile and make it wider to fit your needs.

Part 4: Adding Buttons

More UI buttons can be added easily to your game along with custom functionality. In the following steps, 2 buttons, X and Y, will be added for the player to use in your game. This process can be repeated to add any buttons to the UI that you'd like. They can be called whatever is suitable and they can use any images. Also, if you don't plan to use any mobile input, then you can safely skip to step 3.

1: Duplicate A and B buttons

To begin, locate the player character in your scene hierarchy. Right-click on the player and select: Prefab -> Unpack. Now locate the game objects named "btnA" and "btnB" that are children of the "Touch Controls" game object. Select both of the buttons and press Ctrl+D to duplicate them. Once they are duplicated rename them to "btnX" and "btnY" and move the buttons above the A and B in your game scene. When youre finished, the scene hierarchy should look like the image to the left.

2: Adjust the Inspector

Now that the buttons have been duplicated, some inspector values must be changed to show that these are new buttons. Select one of the new buttons that were created and locate the "Image" component. The "Source Image" value represents the image that will be shown when the button is not pressed. Next, locate the "Button" component and change both the "Highlighted Sprite" and the "Pressed Sprite" values to the image that should be used when the button is being pushed down. Last, find the "TDC_ButtonHandler" component and change the "Button Name" to the name that will be used for the button. This script can also be used to trigger button press and release noises if desired which can be unique for each button.

3: Add Inputs to Manager

After adding the buttons to the UI, they will need to be added to the Input Manager. This process will be the same as in step 2, but it will be recapped quickly here for convenience. Navigate to: ***Edit -> Project Setting -> Input Manager*** and add 2 additional inputs for our new X and Y buttons. Copy the same settings that are used for the A button, but change the "Positive Button" to a new value and rename the input to the name you assigned in step 2 (or a new input name if step 2 was skipped)

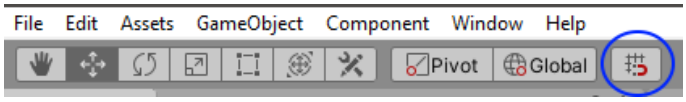
4: Add Button to Script

Locate the player in your scene hierarchy, select it and open the "TDC_Actions" script for editing. Once opened, you will notice 2 templates at the top of this script that can be copied and pasted under the Custom Button Scripts area. Note that the scripts don't have to be put here, this is just done to keep things organized. For this example, we will only worry about the Get Button Down Template. Copy the entire function, paste it below and un-comment it by selecting the text and pressing Ctrl+K+U. Now locate the phrase "CustomButton" in the pasted function and replace it with the name from steps 2 and 3. There are 4 instances of the phrase "CustomButton" in each template that will need to be replaced with your buttons actual name.

Additional Notes

- This asset is set up to assume a single 1x1 square on the grid represents a single tile. If this is not the case for your project, then you may need to adjust the size of tiles in your game to accommodate this.

- There is a magnet snapping tool available in your Unity Editor that will automatically snap your cursor to the closest whole number if enabled. This can be a very useful tool when editing items on the map.



- Objects that can be pushed should not be placed directly in front of teleport tiles if the scene will be reloaded when using the teleport tile again. This will cause the player to teleport on top of an object that they shouldn't be able to walk on top of and may result in the player having the ability to walk through other walls or getting stuck in the level colliders.

- Do not rename any children of the player game object (except the way that is specifically explained in "Part 4: Adding Buttons"). Some scripts look for children of the player game object based on their names, and you may see errors if they are changed.

- If you notice lines in between your tilemap while playing your game, this can be solved by creating a Sprite Atlas. To do this, right-click anywhere in your project window and select: ***Create -> 2D -> Sprite Atlas.*** You can then add your tilemap sheet to the "Objects for Packing" list and select "Pack Preview". Visit this page for more information.

- The sound effects heard in the game are found under the player character. You can edit the music audio clip here to adjust what music will play when the game starts. You can also change this value in code if desired. Otherwise, music should be changed using the Teleport Tiles.

- If you find your sprite images are blurry, this is most likely due to your filter type. To fix this, select your spritesheet and change the "Filter Type" in the inspector to "Point No Filter" and click the "Apply" button.

- The touch joystick will only work for mobile builds. When trying to use it inside the Unity Editor on PC, if you try to click and drag with your mouse, it will drag off the screen.

- Interactable objects should be on the same "Order in Layer" number as walls and the same z coordinate as the player. This will ensure that the player's sprite will appear in front of the wall if necessary.

- Pushable items will have their location automatically reset when the scene is re-entered, but items that have been removed will not respawn. Pushable items also have a function that can be called manually to reset their position. You can call "TDC_InteractableObject.ResetPosition()" as a public method from any script.

Conclusion

I hope that this has documentation has been helpful in setting up your Complete Top-Down Controller. If you have any issues with this package, find a bug, or need help with set up then please feel free to reach out to me via email at FrodoUndead@gmail.com I will do my very best to help you with any problems you may encounter. Thank you again for your purchase. Happy development! 💻