Instructions for using Object Collection and Trigger mechanics in Unity

Jason Fritts
jfritts@slu.edu

Note for Unity 2018+

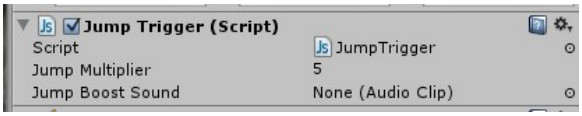
For Unity 2018+, the developers dramatically changed the Character Controller scripts, which eliminated our ability to access the movement variables such as jumpSpeed and moveSpeed. These are needed to create the jump and speed boosts, so we need to use an alternate character controller from the Asset Store in order to use these mechanics.

As a result, instead of **FPSController**, **RigidBodyFPSController**, or **ThirdPersonController**, we'll be using the **First Person Controller** from the *Modular First Person Controller* package. If you have one of the former character controllers in your scene, you'll need to disable it, and add a **First Person Controller** in its place. The *Modular First Person Controller* package is available on the Asset Store, but you'll also find the prefab for the **First Person Controller** in the SLU Prelim Assets package, under the Assets/SLU Prelim Assets/Modular First Person Controller/ folder.

1 Super Jump Trigger

- Change the Tag of **First Person Controller** to Player
  - Select **First Person Controller** object in the Hierarchy window.
  - In the Inspector window, change the **Tag** to Player. This tag should already exist, but if not you will need to create it by selecting the **Add Tag...** option from the drop-down menu, clicking on the + symbol under Tags, entering the name Player in the pop-up window, and clicking the Save button.
- Add one or more **SuperJumpTrigger** prefab objects into your Scene at the desired places
  - To facilitate ease of use, we created a Prefab for the **SuperJumpTrigger**. However, this Prefab was easy to make – it is nothing more than an empty object with a Box Collider, an Audio Source, and an attached **JumpTrigger** script.
  - In the Project window, select the Assets/SLU Gameplay Mechanics/Prefabs/ folder. Inside you will find the **SuperJumpTrigger** prefab object.
  - For each location you desire to have a jump trigger in your scene, move your camera to that position and then drag the **SuperJumpTrigger** prefab object from the Project window into the Scene window. Each time you drag one into the Scene, it will create a unique instance of the jump trigger.
  - After creating an instance of the **SuperJumpTrigger**, you may position and re-size it as desired, either through the Inspector window, or via the Unity selection, translation, rotation, and sizing tools available in the upper left-hand corner of the Unity editor (see below):

  - In addition, the **SuperJumpTrigger** includes a **JumpTrigger** script, which has the following paramters:



– The parameters in the **JumpTrigger** script are: Jump Multiplier and Jump Boost Sound. Like the speed boost, Jump Multiplier indicates how many times higher the the player can jump when inside the jump trigger area, while Jump Boost Sound is an optional sound that plays if the Player jumps when inside the jump trigger area (but won't play if the Player doesn't jump).

  Below shows an example parameterization that will increase the jump height by 5× and will play the **door_powerup** sound when the Player jumps (while inside the jump trigger area).

- **NOTE: Do NOT add the audio file as the** Audio Clip **variable in the** Audio Source **component** of the **SuperJumpTrigger** prefab – Audio Clip should remain set to None. The audio file should only be specified in the **JumpTrigger** script. The Audio Source is only there for adjusting the volume, distance roll-off, other other aspects of playing the sound.

2 Audio Trigger

- Change the Tag of **First Person Controller** to Player

  – Process is identical to the first bullet in **SuperJumpTrigger**, above.

- Add one or more **AudioTrigger** prefab objects into your Scene at the desired places

  – Process is again identical to the second bullet above, where you added instances of the **SuperJumpTrigger** prefab element into the game. The only difference is that the **AudioTrigger** has its own script, with different parameters, as shown below:

  – The parameters in the **AudioTrigger** script are: Audio File, Play Only Once, and Multi Play Delay. The Audio File parameter is obviously the sound file that will be played when the Player enters the trigger area. The Play Only Once option is a checkbox that indicates whether the audio clip can be played more than once per game – if checked, the audio clip can be played only once during the entire game; if unchecked, it will play again whenever the Player re-enters the trigger area.

  The last parameter, Multi Play Delay is only relevant if Play Only Once is unchecked. It's purpose is to prevent the audio trigger from re-playing the audio clip too frequently by enforcing a minimum wait period until the audio clip can be played again.

  Below is an example parameterization that will play the **thunder3** sound when the Player enters the trigger area. It will play this sound each time the Player enters the trigger area, provided at least 15 seconds has passed since the sound file was last played.

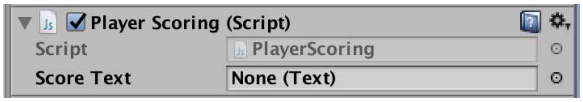
  – Similar to the **SuperJumpTrigger**, it is not necessary to add an Audio Source component to the **First Person Controller** in order to play the audio clip. The **AudioTrigger** prefab already contains the requisite Audio Source component.

  - *Note:* Again, if you haven't already, be sure to remove the Audio Listener component on the **Main Camera**. A scene can only have one Audio Listener for audio to work correctly. Since both the **Main Camera** and the camera in the **First Person Controller** have an Audio Listener, you need to remove/delete one. While playing the game, the player should hear the audio from the perspective of the player controller, so the Audio Listener on the **Main Camera** is the one that should be deleted.

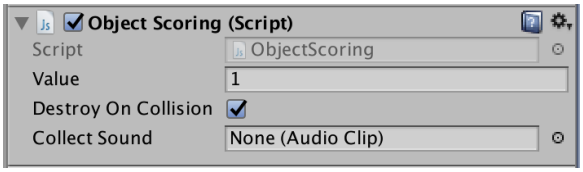
- Optionally modify the **AudioTrigger**'s play parameters, in the attached Audio Source component
  - The **AudioTrigger** includes an Audio Source component that specifies further parameters regarding how the audio clip is played. These include whether it is 2D or 3D audio, whether it's looping, the attenuation characteristics for 3D audio, etc. Modify these as needed for each individual audio trigger.

3 Object Collection and Scoring

- Change the Tag of **First Person Controller** to Player
  - Process is identical to the first bullet in **SuperJumpTrigger**, above.
- Add **PlayerScoring** script to **First Person Controller**
  - Select **First Person Controller** object in the Hierarchy window.
  - In Inspector window, click on the Add Component button.
  - In the drop-down menu, select Scripts → **PlayerScoring**. You should now see the following script component attached to **First Person Controller**.

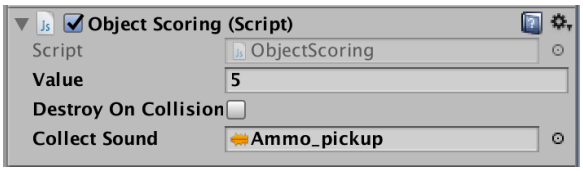


- Prepare each of the objects you want to collect for scoring
  - For each object that you want to collect for scoring, first select it in the Hierarchy window.
  - In the Inspector window, make sure that it has a Collider component. If not, add one and size it as appropriate for the object.
  - In conjunction with the Collider component, either: *A)* check IsTrigger checkbox, or *B)* add a Rigidbody component, **but NOT both**. The former makes a non-solid collectible, whereas the latter make a solid collectible.
  - In the Inspector window, change the object's Tag to CollisionObject. If this Tag does not exist yet, you will need to create it by selecting the **Add Tag...** option from the drop-down menu, clicking on the **+** symbol under Tags, entering the name CollisionObject in the pop-up window, and clicking the Save button. If for some reason you can't create a new Tag, use the Finish Tag.
  - In the Inspector window, click on the Add Component button. Then in the drop-down menu, select Scripts → **ObjectScoring**. You should now see the following script component attached to the object.



  - The parameters in the **ObjectScoring** script are: Value, Destroy On Collision, and Collect Sound. Value is the number of points added to the game score when the player collides with this object. Destroy On Collision dictates whether the object is deleted from the game upon collision – if the box is checked it will be deleted from the game; if not checked, it remains in the game. Finally, Collect Sound is an optional audio file that is played when a collision occurs.

    Below shows an example parameterization that upon collision will add +5 points, will not delete the object from the game, and will play the **Ammo_pickup** sound.



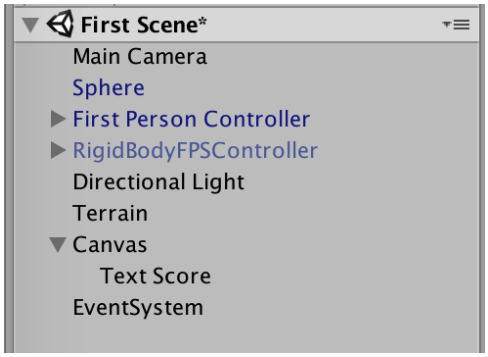
  - Optionally, if you're creating many objects with the **ObjectScoring** script, you can make a Prefab object that already has the appropriate Tag, **ObjectScoring** script, and Collider component (with IsTrigger checked or a Rigidbody component). You can then quickly create instances of this Prefab by dragging them into the game.

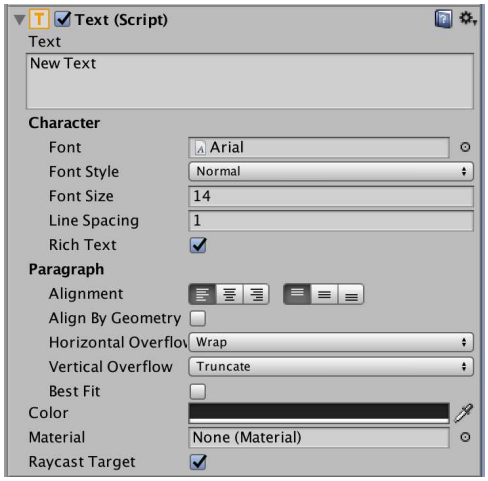
- Optionally add an Audio Source to the **First Person Controller**

  - If you want an audio sound played when the player collects a CollisionObject, you need to add an Audio Source component to the **First Person Controller**.
  - In the Inspector window, click on the Add Component button.
  - In the drop-down menu, select Audio → Audio Source. When the player now collides with an object, it will play the sound file specified by that object's Collect Sound setting in its ObjectScoring script.
  - *Note:* If you haven't already, be sure to remove the Audio Listener component on the **Main Camera**. A scene can only have one Audio Listener for audio to work correctly. Since both the **Main Camera** and the camera in the **First Person Controller** have an Audio Listener, you need to remove/delete one. While playing the game, the player should hear the audio from the perspective of the player controller, so the Audio Listener on the **Main Camera** is the one that should be deleted.

- To display the game score, add a **Text** object to the Scene

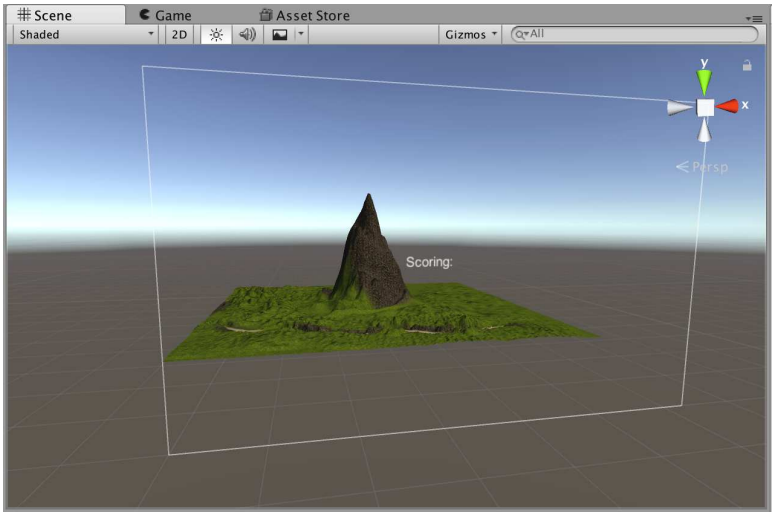
  - In the Hierarchy window, click on the Create button and select UI → **Text**. Once you've done this, you'll see three objects added to your scene – a **Canvas**, a **Text** object under **Canvas**, and an **EventSystem**, as shown below.
  - The **Canvas** object defines the HUD (Heads-Up Display) overlay that is added on top of the game view, when playing the game. The **EventSystem** controls communication between the game and the HUD, and the **Text** object is the object we'll use for displaying the score from object collection.
  - For reference, you may want to change the name of the **Text** object to something more descriptive like **Text Score**, though this is not required.
  - To modify the parameters of the **Text** element, select it in the Hierarchy window, and you will see the following in the Inspector window.
  - To make the text more visible, in the Inspector window, change Font Size to a larger value like 20 or 24.



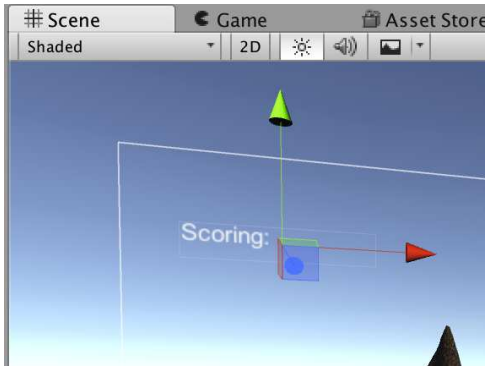


  - You may also want a more visible color for the text than black, so change Color to a bright color like white, yellow, or other more visible color.
  - And while not completely necessary (because we'll be changing it in the code), you may want to change the text to "Scoring:".
  - Finally, in order for the displayed score to be updated each time another object is collected, attach the **DisplayScore** script to the **Text** object.

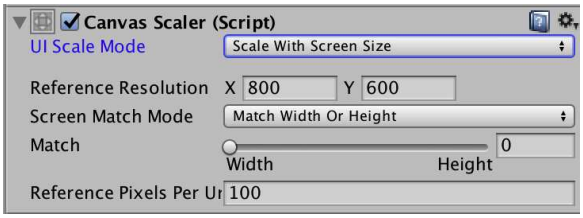
- Next, adjust the position of the **Text** score within the HUD display



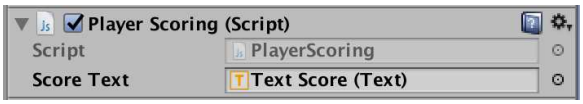
- When you added the **Text** (and **Canvas**) object in the Hierarchy window, you may have noticed that a large white rectangular box appeared in your Scene window. To get a better view of it, adjust your view position in the Scene window so that you can see the whole rectangular white box, as shown below.
- This is your visual display for what the HUD will look like while playing the game. Notice that you can select the **Text** object and move it within the canvas area, as shown here. I suggest moving it to one corner of the Canvas/HUD.



- In order for this positioning to work correctly across different screen sizes, I also recommend selecting the **Canvas** object in the Hierarchy Window, and then adjusting its parameter for UI Scale Mode to Scale with Screen Size in the Inspector window, as shown.



- Finally, link the **Text** score object to the **PlayerScoring** script
  - In the Hierarchy window, select the **First Person Controller**. Scroll down in the Inspector window until you see the parameters for the **PlayerScoring** script.
  - Click and hold on the **Text** element that displays the score. While holding down the mouse button, drag the **Text** element into the Score Text parameter in the **PlayerScoring** script.
  - After completing this, the **PlayerScoring** script should show the name of your **Text** element after the Score Text parameter. This links the two objects together, so now the score should display correctly when you play the game.



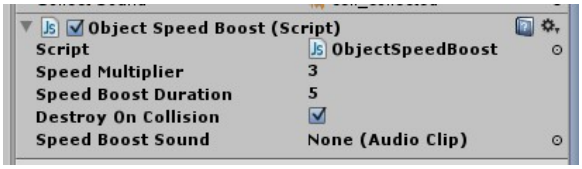
4 Speed Boost from Object Collection

- Add **PlayerSpeedBoost** script to **First Person Controller**
  - Process is identical to the first bullet above, where you added the **PlayerScoring** script to the **First Person Controller**. Here you simply use the **PlayerSpeedBoost** script instead.

- The only difference is that the **PlayerSpeedBoost** script is simpler; it does not have any parameters that need to be defined.

- Add **ObjectSpeedBoost** script to each of the Rigidbody objects you want to give a speed boost upon collection

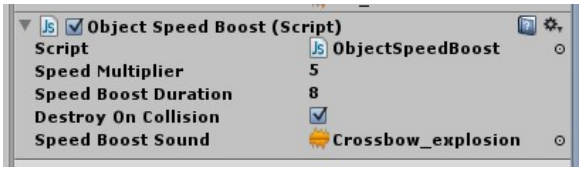
  - Process is identical to the second bullet above, where you added the **ObjectScoring** script to the Rigidbody objects. Here you're adding the **ObjectSpeedBoost** script instead, which has a few different parameters, as shown below:



  - The parameters in the **ObjectSpeedBoost** script are: Speed Multiplier, Speed Boost Duration, Destroy On Collision, and Collect Sound. Speed Multiplier indicates how many times faster the player becomes (e.g. 3 would indicate three times faster). Speed Boost Duration indicates how many seconds the speed boost will last. Destroy On Collision and Speed Boost Sound are identical to **ObjectScoring**, determining whether the object will be deleted and which sound (if any) will be played upon collision.

    Note: If you don't want an object deleted after a collision, and both the **ObjectScoring** and **ObjectSpeedBoost** scripts are attached to an object, then Destroy On Collision must be unchecked in both scripts.

    Below shows an example parameterization that will increase speed by 5× for 8 seconds, will delete the object from the game, and will play the **Crossbow_explosion** sound.



  - Optionally, if you're creating many objects with the **ObjectSpeedBoost** script (and/or **ObjectScoring** script), you can make a Prefab object that already has the appropriate Tag, Rigidbody component, and one or both scripts, and then quickly create instances of this Prefab by dragging them into the game.

- Optionally add an Audio Source to the **First Person Controller**

  - Just like bullet three for **PlayerScoring**, if you want to play an audio sound when the player collects a CollisionObject, you need to an Audio Source component to the **First Person Controller**.

5 Player Shooter Script

- The **Shooter** script we created in the *Wall Shooter* assignment has been adapted for you to function on the player controller. Simply add the **PlayerShooter** script to the **First Person Controller**. Be sure to place it directly on the **First Person Controller**, and not on the **Main Camera** sub-object within it.

  - Select **First Person Controller** object in the Hierarchy window.
  - In the Inspector window, click on the Add Component button.
  - In the drop-down menu, select Scripts → **PlayerShooter**. You should now see that script attached to the **Main Camera** under **First Person Controller**.
  - Select appropriate parameters for the **PlayerShooter** script, just like we did in the *Wall Shooter* assignment.

6 New Mechanics

- A few new, but similar, gameplay mechanics have been added to the asset package in recent semesters, including:

  - A **TeleportTrigger** script and prefab – It operates identically to the other trigger scripts and prefabs, but you need to specify the teleport location via another GameObject. I recommend using the menu option of GameObject → **Create Empty** to create an empty (non-visible) object, setting its location to the teleport target, and linking it to the TeleportTrigger script.
  - A WinTrigger script and **WinArea** prefab – It operates identically to the other trigger scripts and prefabs, but you similarly need to create a **Text** element in the **Canvas** for displaying the win message. The win message will only display once the player has entered the trigger area, and then the Scene will optionally reset after a short delay.

    Note: This trigger can likewise be used to designate a trigger area for losing the game, simply by replacing the message and sound effect as appropriate. For example, it might be used to encompass a lava-like ground surface area, causing the player to die should he/she fall into it.
  - A UnderwaterPrefab script and three water prefabs based on **Water4Prefab**, **WaterProDaytime**, and **WaterProNighttime** – These give the sense of being "underwater" when the player goes below the water level. To use these prefabs you simply replace your existing water object with the corresponding prefab in the *SLU Gameplay Mechanics/Prefabs/* folder. You can you the Copy Component and Paste Component options to position and size the replaced water objects identically.