

# Unity 3D Workshop: Bean Quest

## 0. Setup

Requirements: Unity Hub, Visual Studio 2019

### 0.1 Installation:

Inside Unity Hub, head to the 'Installs' tab and click on the blue 'ADD' button in the top right. Here you should now select the 'Recommended Release' (as of the time of writing Unity 2020.3.11f1 (LTS)). In the second step you may add additional modules, but the default selection will be enough for this workshop.

Once this version of Unity has finished installing you can now go to the 'Projects' tab and create a new Project by clicking on the 'NEW' button and selecting name, location and template. For this workshop you will need the regular '3D' template.

To ensure that writing Unity Scripts with the help of Visual Studio works well you may have to set it as Unity's 'External Script Editor' and download the required plugins for VS. Both can be done following [this guide](#).

## 1. First Steps

### 1.1 Set the scene

Once you start the project you'll be greeted with a sample scene containing a Main Camera and Directional Light.

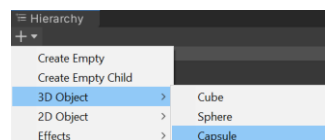
Before we start adding our own object let's do two small things:

- In the Project View go to Assets/Scenes and rename the 'SampleScene' to 'MainLevel'
- Change the rotation of the 'Directional Light' to (90 | 0 | 0). This way the player's shadow will be casted directly below the player later on.

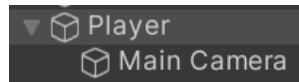
### 1.2 Add a Player

First of all we'll need a player. A little 3D bean should do just fine here.

- Create a 'Sphere' 3D Object in the Hierarchy view and rename it to 'Player'



- Also select the 'Player' Tag on this object
- Move the Main Camera onto our new 'Player' object so that it looks like this:

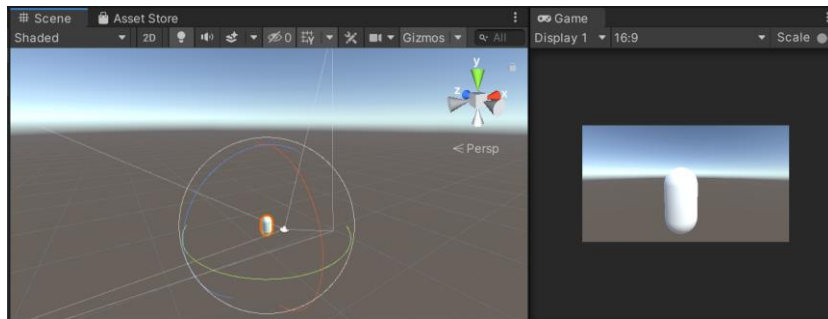


- Now move and rotate the camera so that it has a nice view on the player

Since we made the camera a child object of the Player, it will now automatically orientate around the player and move and rotate alongside him.

We can test this by heading into 'Play mode':

- Press the play button in the middle of the screen to start play mode
- Drag and drop the 'Game' handle to another part of the screen, so you can see the 'Scene' and 'Game' view at the same time



Now you can move and rotate the player around in the scene view and look how it looks in-game inside the 'Game' view.

Watch out: all changes made to the scene while in play mode will always be reset when exiting it. Make sure to not leave it running by accident.

- Press the play button again to exit play mode
- Create a new folder 'Assets/Prefabs' and drag our Player object inside it. This will save it as a reusable prefab

### 1.3 Platforms, Please!

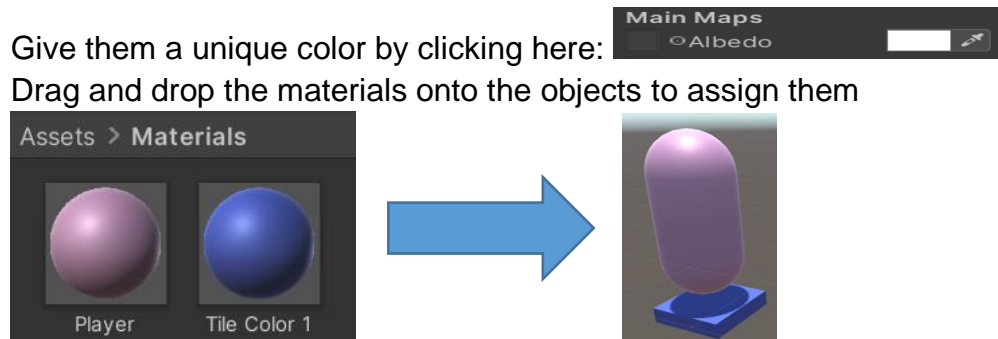
Next let's create some platforms to stand on.

- Create a 'Cube' 3D Object and decrease it's height so it looks somewhat like this:



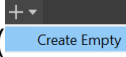
Now follow these steps to add some color:

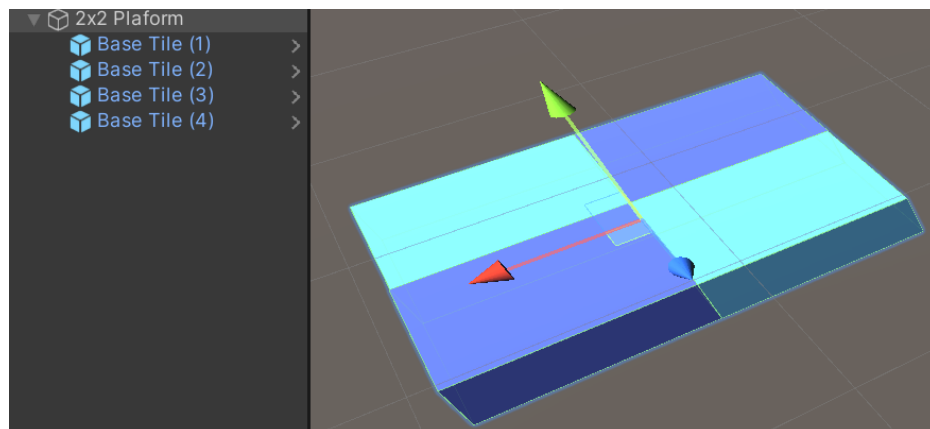
- Create a 'Materials' folder
- Inside the folder create at least two new materials, one for the player and one for the tile
- Give them a unique color by clicking here:
- Drag and drop the materials onto the objects to assign them



Fun Fact: you can also create a material for the sky to change the skybox. If you're tired of the default sky you can check out [this video](#).

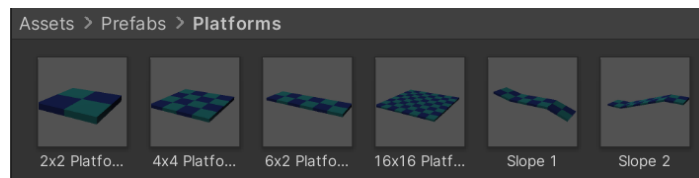
Having the floor made out of small equal-sized tiles can be useful for judging distance. To avoid having to place each tile one by one however, we should do this:

- Save the previously created tile as a 'Base Tile' prefab
- Create a secondary tile color material
- Create an 'Empty' game object (  ) called 2x2 Platform and assign four 'Base Tiles' to them. Then color and move them so they look like this:



- Save your 2x2 Platform as a prefab, so it can be freely reused by simply dragging it into the Scene or Hierarchy
- Repeat the process until you have a good selection of platforms of varying shapes and sizes. Make sure to use previously created prefabs were fit to massively save time. (A 16x16 Platform could consist of four 4x4 Platforms e.g.)
- Last but not least you can use your platforms to create a small obstacle course

Once done your Platform collection could look like this:



## 2. Player Controls

### 2.1 Base controls

Now that you made it past the first steps it is time to write your first scripts to add movement and camera controls for the player. For the next steps you can orientate around [this video](#) to create your first two scripts. Watching it in its entirety is not required as the specific clips for the scripts you must copy will be provided.

- Move the 'Main Camera' inside the Player's head, so that it mimics the first person view of the player
- Add a 'MouseLook' script to the camera, which will allow the player to look around based on mouse rotation. To do this follow [these video instructions](#) (up until 10:20)
- Add a 'Character Controller' component and 'PlayerMovement' script to the player. Write the script by [following this part of the video](#)
  - Make sure that the 'Base Tile' prefab is part of the 'Ground'-Layer (like mentioned [here](#))

### 2.2 Sprint

Thanks to the video you should now have two very nice and useful scripts. But now it's time to add some of your own code by building onto them.

Inside the Player Movement Script do the following:

- Make the 'move Speed' variable private and add two new public variables:

```
[Header("Movement")]
[Tooltip("units per second the player moves by default")]
public float baseMoveSpeed = 4.0f;
[Tooltip("units per second the player moves when sprinting")]
public float sprintMoveSpeed = 7.5f;
```

- Now change the script so that 'moveSpeed' is equal to 'sprintMoveSpeed' while pressing left-shift and equal to 'baseMoveSpeed' when not

## 2.3 Multi-Jump

- Add these two variables to the 'PlayerMovement' script:

```
[Tooltip("number of jumps possible in between touching the ground")]
public int maxJumps = 2;

private int jumpsLeft;
```

- Now change the script so that the player does not have to be on the ground to jump but instead needs to have enough 'jumpsLeft'
  - Jumps are **replenished when landing onto the ground** and used up from jumping

## 2.4 Respawnning

We may have nice movement now but there is one problem: if the player falls of the map, he will just fall forever. To fix this let's add another script to the player called 'RespawnManager'.

- Add these variables to the script:

```
public class RespawnManager : MonoBehaviour
{
    [Tooltip("minum height the object can reach before respawning")]
    public Vector3 minimumHeight = new Vector3(0, -100, 0);
    [Tooltip("if this is checked the player will always respawn were he initially started")]
    public bool useAutomaticRespawnPoint = false;
    [Tooltip("only used if previous box is not checked")]
    public Vector3 respawnPoint;

    private CharacterController controller = null;
```

- Now implement the script so that it checks the player's height for every frame and sets his position to that of the respawn height if he is lower than the minimum height
  - As the CharacterController will prevent the player's position from being altered from other sources, you must disable it before teleporting the player and then enable it right after

## 2.5 Quitting the game

- Add a 'Game Manager' script to the scene (inside an empty object)
  - this script must simply close the Application when pressing 'escape'

Now we can fully build the game by heading into 'File/Build Settings' or pressing Ctrl+Shift+B. Here press 'Add Open Scenes' and select PC, Mac & Linux Standalone.

If you want you can also click on 'Player Settings' to configure the game's name, version, icon etc.

Then you just have to press 'Build' and select an (empty) Folder to build the game into. Once done you will now have a fully working game executable that you can share with anyone (as long as you include the subfolders). Congrats!

### 3. Extra Obstacles (Bonus)

#### 3.1 Collectibles

How about some nice golden beans to collect?

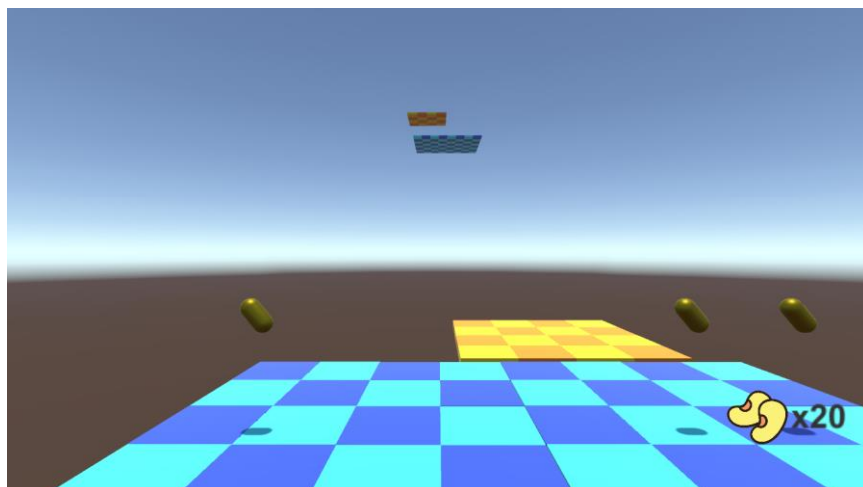
- Add an empty 'Collectible' Game object
- Give it a small capsule that is tilted on the Z-Axis as a child object
- Add a 'Collectible Manager' script to the 'Collectible' Object
- Remove the collider from the capsule and add a box collider going roughly around the Capsule to the 'Collectible' Parent Object. This collider should be marked as 'Trigger'
- Add a public 'collected Beans' variable to the Player

With the help of the script you should make it so that the Collectible will increase the Player's bean count and the self-destruct upon colliding with the player. You should use [this function](#) to check collision.

To give the player more incentive to collect it you should also add a 'roationSpeed' variable to the script and then make the Collectible rotate to this speed (units per second) around the y-axis.

Since we want to know how many beans were collected, it's time to create your first UI elements!

- Add a 'UI->Canvas' child object to your 'Main Camera'
  - The Canvas must at least contain a UI->Text and could also contain a UI->Image with an accompanying bean-Icon
- Then, add a script that takes your 'PlayerController' script and a Text and will update the text based on the 'collectedBeans' variable



Now the Player has something nice to work towards!

## 3.2 Moving Platforms

How about we spice up our gameplay by adding some moving platforms?

- Add a 'PlatformMovement' script, that takes the following variables:

```
public class PlatformMovement : MonoBehaviour
{
    [Tooltip("Initial position of the platform")]
    public Transform pointA;
    [Tooltip("Initial goal of the platform")]
    public Transform pointB;
    [Tooltip("Speed (in u/sec) that the platform has to constantly move between the two points")]
    public float moveSpeed = 3.0f;
```

- Now make it modify the transform.position to move between the two points (tip: [Vector3.MoveTowards](#))

You might notice that the player will not move along with the platform's vertical movement. That is because, thanks to the 'CharacterController', the player can only be moved manually via the 'controller.Move()' function. To fix this you could add a public Vector3 'externalMovement' variable that must be added to the 'controller.Move' call of the player and must always be equal to the movement of any platforms he is standing on. This part is tricky, but if you made it this far you can probably handle it.