Unity Practice Tasks

TASK1 – Setting up your Scene

1. Create a new 2D project and save your scene as <Yourname>Tasks1 (no spaces). Please ensure that you follow this naming convention for all the tasks you do.

   Unity is inherently a 3D game engine. However, by setting the camera to 'orthographic' rather than 'perspective', Unity will render objects uniformly, with no sense of perspective or depth. This means that in Orthographic mode, the Z-position is not relevant to how far or near objects appear to be. However, the Z-position will still dictate how objects appear on top of each other. (E.g. your background must always be furthest from camera, with other objects nearer to the camera, otherwise your background will hide your gameobjects.)

2. Change Camera to 'Orthographic'. Set camera size to **8** and Z-position is **-10**. You scene will consist of 16 world units. (X: from -8 to +8; Y: from -8 to +8)

3. Create a new aspect ratio 1:1. To do this, open the aspect ratio dropdown in Game Window, select + icon and add a new 1:1 aspect ratio. You need to change Type: Aspect Ratio and enter 1 for both width and height.

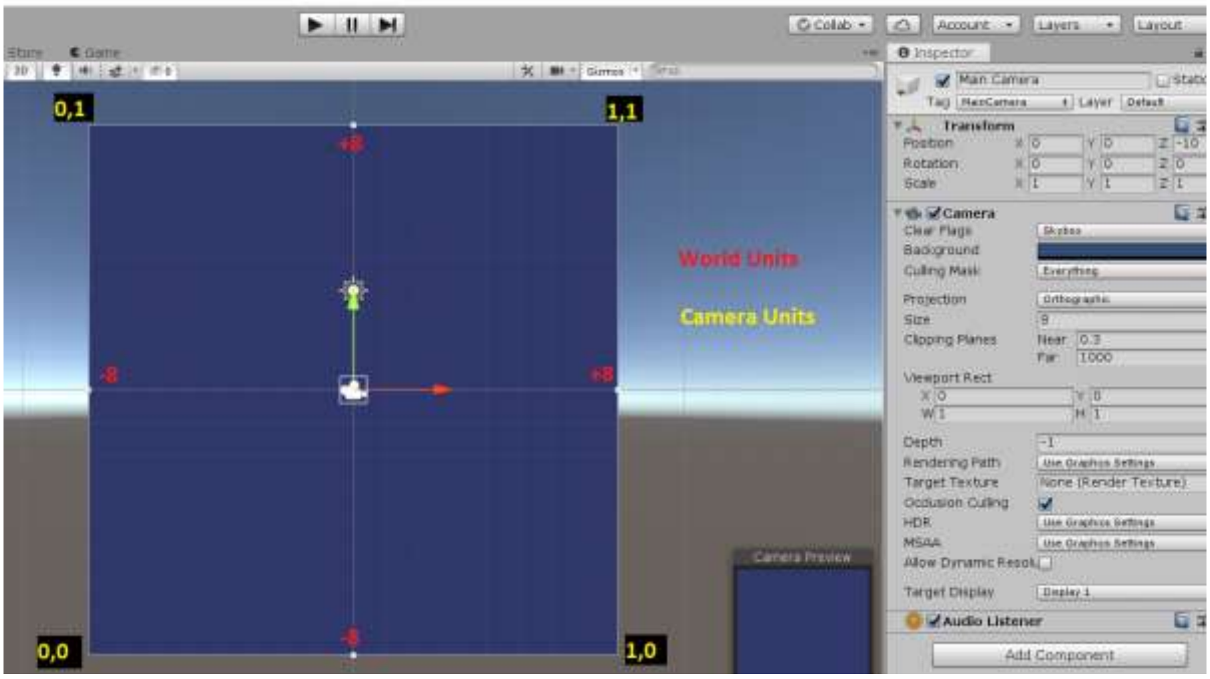
4. Create a BLUE background (2D Box sprite), sizing it to cover the camera field and at z position: 10. Make sure other objects you create have a Z-position between -10 and 10. i.e. between background and camera. Objects created at transform position x=0,y=0 should appear at the centre of the scene.

TASK2 – Units, PreFabs, and Scene Boundaries

1. Use a RED ‘Circle’ 2D sprite to create a Red Ball and add it to the scene. Place the circle object at the center of the screen.

2. Make the Red Ball a PreFab and delete the ball object from the scene.

Camera Viewport coordinates are different from world space coordinates (transform.position). The figure below explains the difference between world space values and The Camera view (viewport) coordinates:



Main Camera
Tag MainCamera Layer Default

Transform

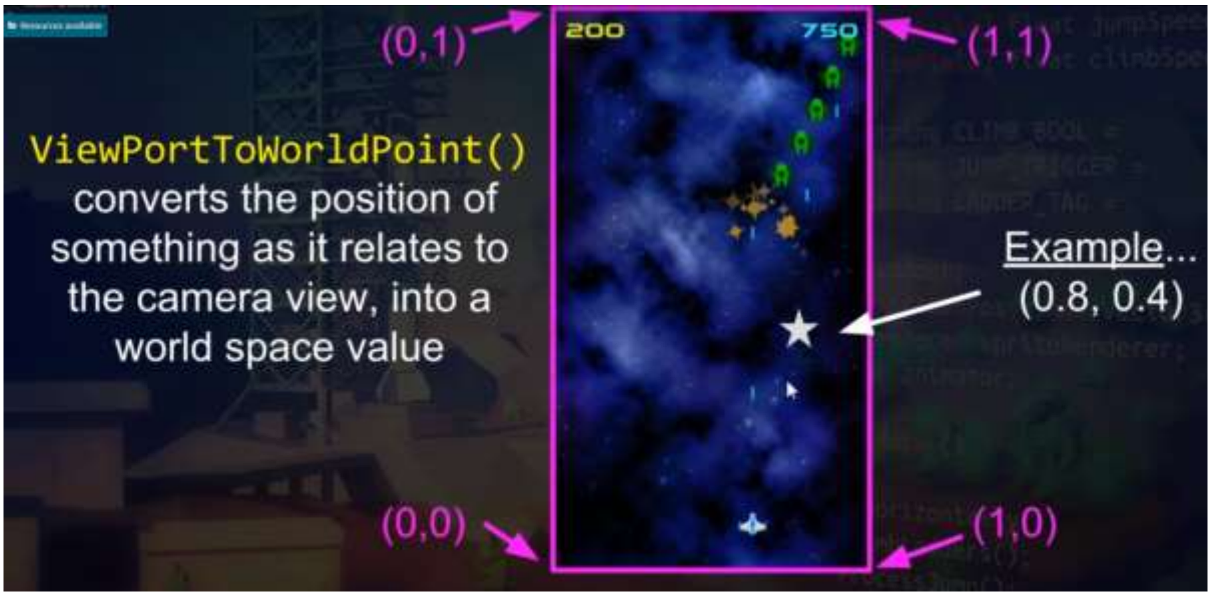
Position	X 0	Y 0	Z -10
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Camera

Clear Flags	Skybox
Background	
Culling Mask	Everything
Projection	Orthographic
Size	8
Clipping Planes	Near 0.3
	Far 1000
Viewport Rect	X 0 Y 0
	W 1 H 1
Depth	-1
Rendering Path	Use Graphics Settings
Target Texture	None (Render Texture)
Occlusion Culling	✓
HDR	Use Graphics Settings
MSAA	Use Graphics Settings
Allow Dynamic Resol.	☐
Target Display	Display 1

Audio Listener

On the other hand game object positions are set using ‘world’ coordinates, through the transform component. Hence, to position objects relative to the camera, a conversion from Viewport (camera) to WorldPoint units is required.

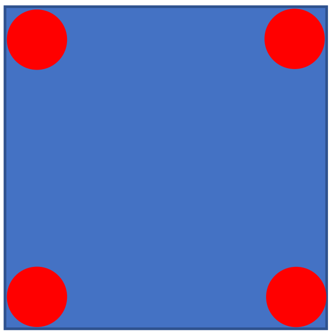


ViewPortToWorldPoint() converts the position of something as it relates to the camera view, into a world space value

3. Research ViewportToWorldPoint() to obtain the FOUR WorldPoint coordinates which correspond to the Camera corners as shown in the above figure. Store the real world boundary limits in **four float** variables (XMin, YMin, XMin, XMax).

4. Put the above code in a separate METHOD called **SetBoundaries()**. You can call the method from Start(). Should the Camera change, this Method will enable you to obtain the new screen boundaries in world space units.

5. Spawn 4 balls at each corner of the Camera view using the created variables + or – a padding, so they appear as in the example below:



TASK 3 – Spawn objects

1. Write a function to spawn a red ball at the mouse position (within the game boundaries) ON a left mouse button down. Call this function from Update().
   (Hint: Research `Camera.main.ScreenToWorldPoint(Input.mousePosition)` to read Screen Mouse Position and convert it to World Units which are required to instantiate object at the correct 'transform position'.

2. Modify the above function to spawn a maximum of 5 balls. (one on each mouse click)

TASK 4 – Mouse ops

Write a function which allows the user to DRAG the circles anywhere on the screen.

TASK 5 - Physics

1. Add **Static** 2D Colliders at the edge of the Camera View.

2. Add Rigidbody2D and CircleCollider2D components to the Ball Prefab and write a function that gives a velocity to the instantiated gameobject as follows. (HINT: research how to assign an instantiated gameobject to a GameObject variable and use gameobject.GetComponent<> to access the rigidbody component.

   ```
   e.g. ballobject.GetComponent<Rigidbody2D>().velocity = new
   Vector2(20f,20f)*Time.deltaTime;
   ```

3. Experiment with Ridigbody properties to understand how the Physics engine works. (e.g. change Body Type between Dynamic, Static and Kinematic to understand these properties)

TASK 6 – Key Controls

Create a Unity application that starts with 1 square on screen. When the arrow keys are pressed once, the square moves one block. The square should not be able to leave the screen.

TASK 7 - Lists

Use a list to create a chessboard pattern on screen. When the space bar is pressed, turn all the white squares to red squares.