Ben Saylor
CSC 831 – Multiplayer Game Development
Spring 2015

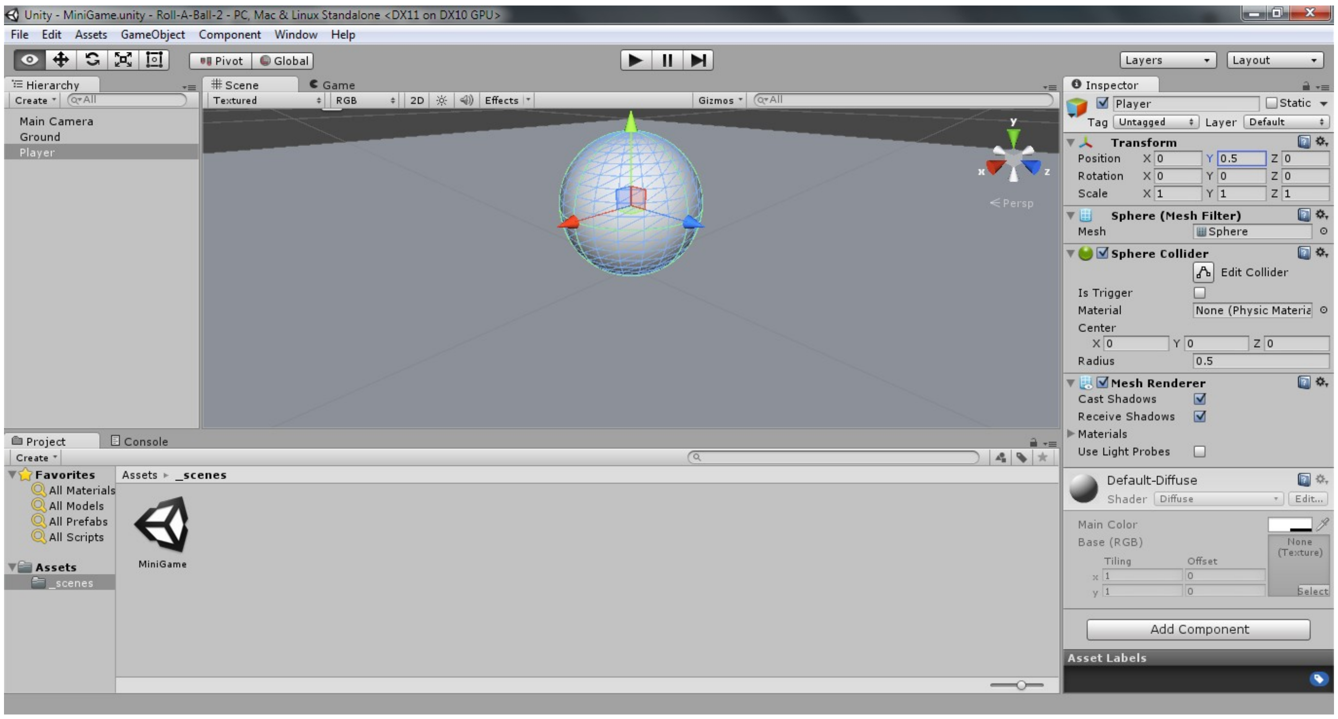
Project 1: Client Development Part

Play with Unity3D

1. I chose one of the official video tutorials, Roll-A-Ball.

2. I created the game as described in the tutorial, but I rotated the walls to form ramps up which the player could roll the ball to collect pick-up objects. I then built the game for the Unity web player. The Unity project is included.

Screenshot after creating the ground and player objects:



Player ☐ Static
Tag Untagged Layer Default

Transform
Position X 0 Y 0.5 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

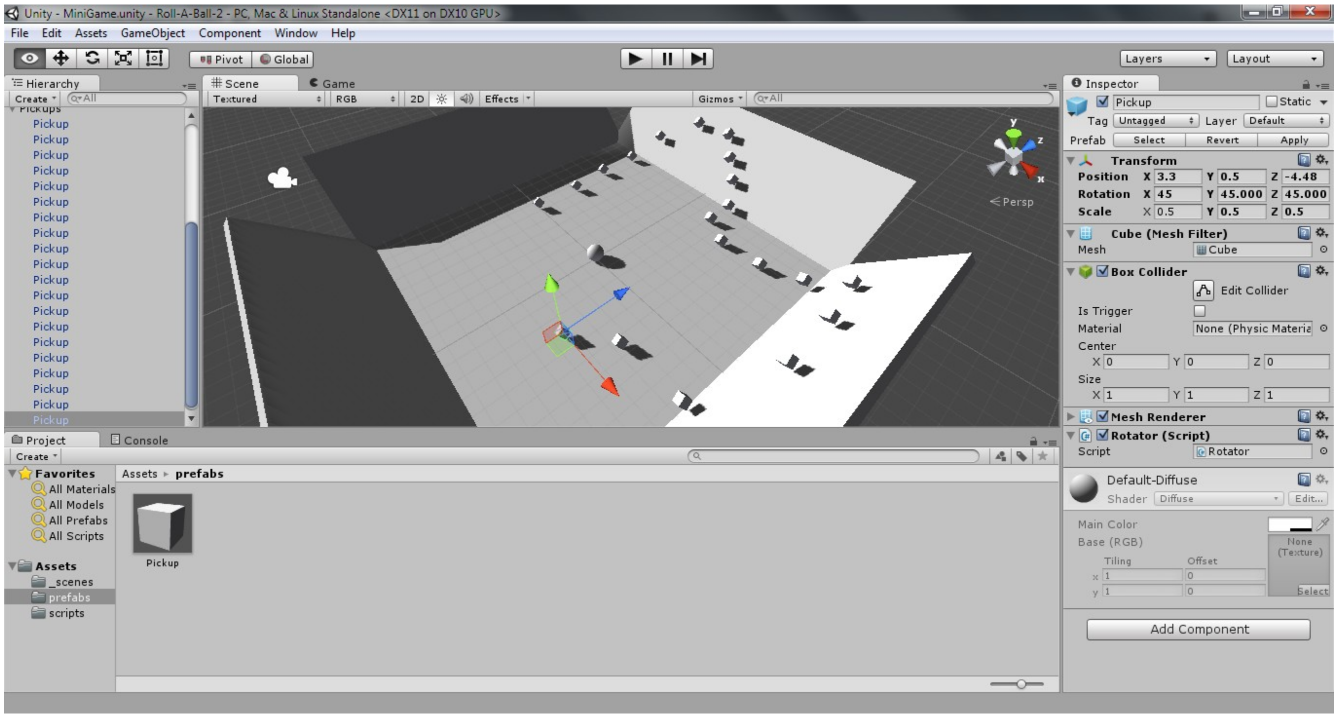
Sphere (Mesh Filter)
Mesh Sphere

Sphere Collider
Is Trigger ☐
Material None (Physic Materia
Center X 0 Y 0 Z 0
Radius 0.5

Mesh Renderer
Cast Shadows ✓
Receive Shadows ✓
Materials
Use Light Probes ☐

Default-Diffuse
Shader Diffuse Edit...
Main Color
Base (RGB) None (Texture) Select
Tiling x 1 y 1
Offset x 0 y 0

Screenshot of scene view with pickup objects placed along the ramps (my variation):



Pickups
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup
Pickup

Pickup ☐ Static
Tag Untagged Layer Default
Prefab Select Revert Apply

Transform
Position X 3.3 Y 0.5 Z -4.48
Rotation X 45 Y 45.000 Z 45.000
Scale X 0.5 Y 0.5 Z 0.5

Cube (Mesh Filter)
Mesh Cube

Box Collider
Edit Collider
Is Trigger ☐
Material None (Physic Materia
Center
X 0 Y 0 Z 0
Size
X 1 Y 1 Z 1

Mesh Renderer

Rotator (Script)
Script Rotator

Default-Diffuse
Shader Diffuse Edit...
Main Color
Base (RGB)
Tiling Offset
x 1 0
y 1 0
None (Texture) Select

Terms Learned

- Project: a collection of assets that are compiled into a game – essentially, the source code of a game
- Scene: typically a game level; could also be an entry or transition screen
- Assets: can include scenes, prefabs, scripts, textures, models, etc.
- Inspector: The GUI tab that displays and allows editing of the properties of a GameObject and its Components
- Component: defines an object or behavior associated with a GameObject
- Transform component: defines a GameObject's position, rotation, and scale
- GameObject: A container for Components and other GameObjects
- Vector3: A 3D vector
- FixedUpdate: A script method that is called every frame just before performing any physics calculations
- Rigidbody: A Component that allows Unity's physics engine to control the movement of a GameObject
- Prefab: A blueprint for instantiating GameObjects with predetermined parameters and components

- Collider: A Component that allows a GameObject to collide with other objects
  - Static colliders
  - Dynamic colliders
  - Trigger colliders
- Tag: A user-defined string associated with a GameObject, used to obtain a reference to the object within a script. A GameObject can only have one tag.
- Kinematic rigid body: will not react to physics forces. It is useful for objects with colliders that need to be animated or moved by their transform, because it saves Unity from having to recalculate collider volumes every frame.

3. Going through the Unity Overview section of the Unity Manual, here are the additional terms I learned:

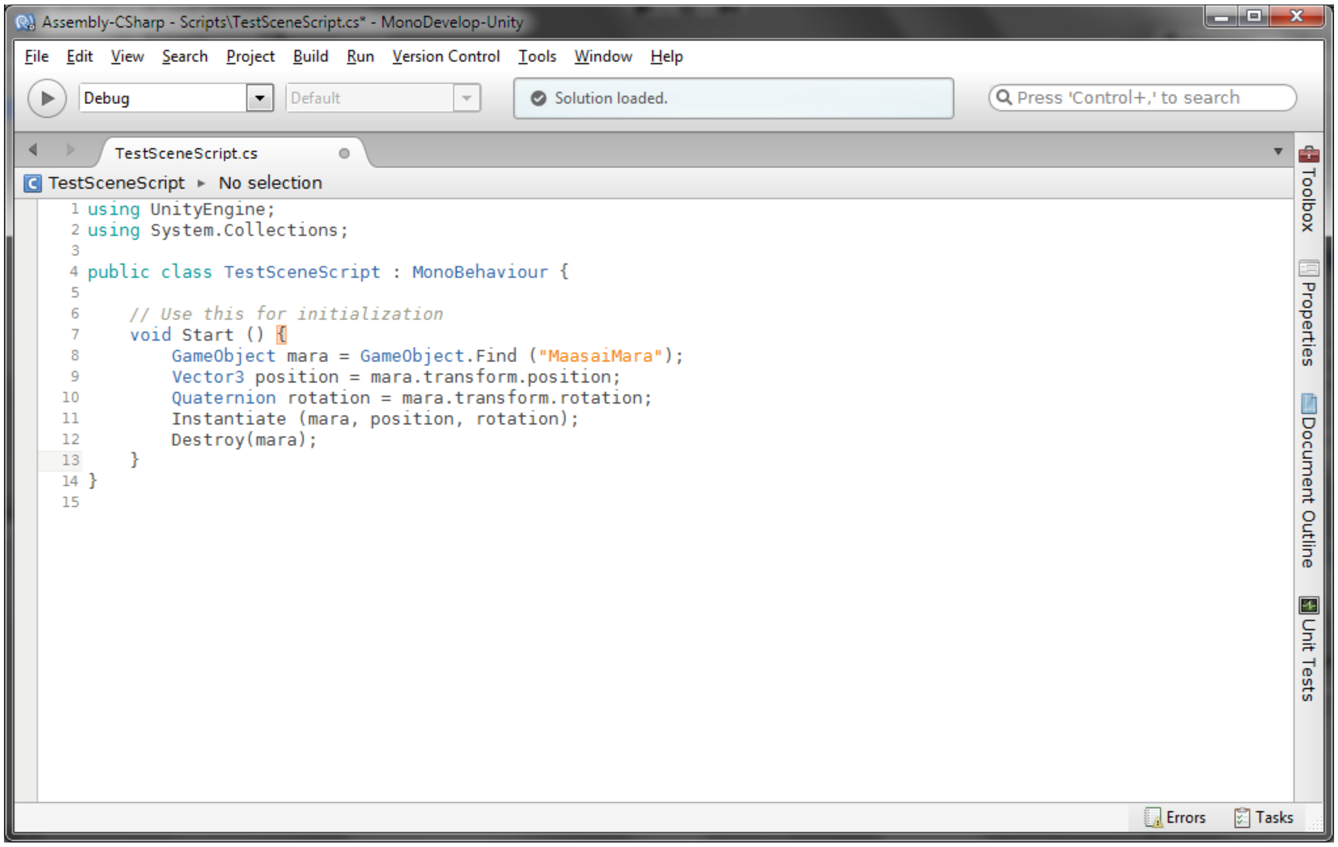
- Views: Panels in the GUI interface, such as the Toolbar, Inspector, Hierarchy, and Project
- Layers: group parts of a scene for separate rendering, lighting, or raycasting.
- Flythrough mode: A useful mode of navigating the scene view by holding down the right mouse button and using the keyboard to move in any direction.
- Gizmo: An object displaying axes of an object or the scene (Scene Gizmo), allowing manipulation of the object's transform or the current view of the scene, or otherwise helping to identify/visualize game objects
- Asset Labels: Text labels that can be applied to assets to make them easier to find and organize
- Material: A component to which a texture can be applied
- Static GameObject: A GameObject can be set to “static” to indicate that it will not move, allowing various performance optimizations to be applied to it.
- Preset Library: a collection of user-defined presets (for a gradient or a curve, for example)
- Snapping: Unity provides Unit Snapping Surface Snapping, Look-At Rotation, and Vertex Snapping to allow quick and precise positioning of GameObjects.
- Overdraw: A view mode in which objects are displayed as transparent silhouettes
- Mipmaps: A view mode that indicates how well textures fit the objects to which they are applied
- Lightmap: A precalculated map of brightness across surfaces
- Rendering Paths: Rendering methods that vary in the quality level of lighting and shadows and in hardware demands
- Local coordinates: Transform coordinates of a GameObject relative to its parent

Creating a Scene

Steps 4-6 were straightforward.

7. I wasn't sure at first how to attach a script to the scene, but I figured out that I could create a new empty GameObject (TestSceneScriptObject) and attach a new script component to it.

8. I put the code in the Start() method of the script so it would be run when the game started. I used the code posted in the hint on iLearn. I found that destroying the terrain object before instantiating the copy resulted in a scene with no visible textures. Instead, I destroyed the original after instantiating the copy.


Assembly-CSharp - Scripts\TestSceneScript.cs* - MonoDevelop-Unity
File Edit View Search Project Build Run Version Control Tools Window Help
Debug
Default
Solution loaded.
Press 'Control+,' to search
TestSceneScript.cs
TestSceneScript ▸ No selection
Toolbox
Properties
Document Outline
Unit Tests
Errors
Tasks

```
using UnityEngine;
using System.Collections;

public class TestSceneScript : MonoBehaviour {

    // Use this for initialization
    void Start () {
        GameObject mara = GameObject.Find ("MaasaiMara");
        Vector3 position = mara.transform.position;
        Quaternion rotation = mara.transform.rotation;
        Instantiate (mara, position, rotation);
        Destroy(mara);
    }
}
```

Populating Scene with Species

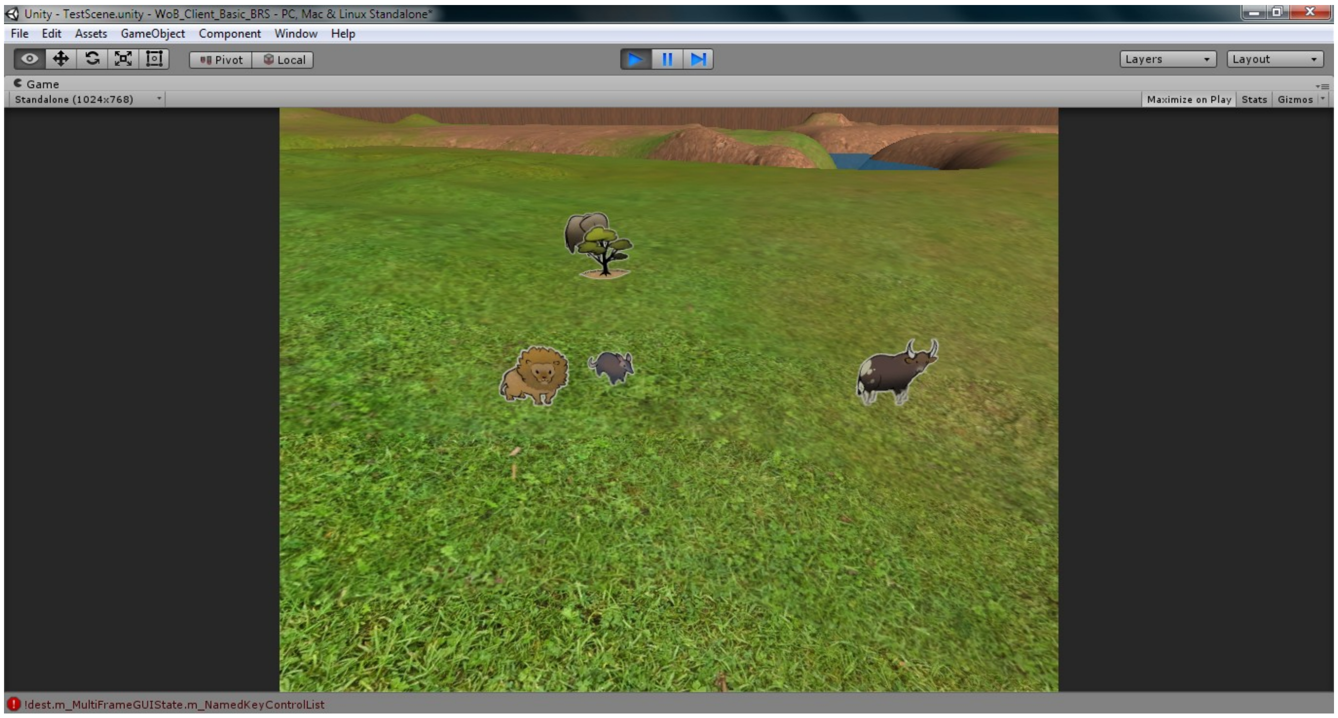
1. I completed this as in step 7.

2a. I positioned the camera to point at HQ and the surrounding area. I used the hint you posted on iLearn, and inferred from it that we needed to create the Materials for the species. I figured out that in order to obtain a reference to "dummy", I needed to create a public variable and drag the dummy prefab into the slot in the Inspector for the script. I spent some time debugging the following error:

> NullReferenceException: Object reference not set to an instance of an object
> TestSceneScript2.Start () (at Assets/Scripts/TestSceneScript2.cs:12)

It turned out that apparently, Unity had not recompiled my script since my last edit – hitting the Play button resolved the issue, but the following new error appeared: “Failed to create agent because there is no valid NavMesh”. I resolved this issue by deactivating the Nav Mesh Agent in the Inspector for Dummy. However, the test animal I'd created in the script still did not appear. After manually duplicating the dummy and setting its material to the elephant material, I discovered that the reason it was invisible was the camera was looking at it from behind – the side of the plane that is not displayed. After resolving this, creating the Materials for the additional species and instantiating them at random positions was straightforward.

2b. This was straightforward – I just had to learn about the Input.GetKeyDown() method. The keys to create animals are the numeric keys 1, 2, and 3.


Unity - TestScene.unity - WoB_Client_Basic_BRS - PC, Mac & Linux Standalone*
File Edit Assets GameObject Component Window Help
Pivot
Local
Layers
Layout
Game
Standalone (1024x768)
Maximize on Play
Stats
Gizmos

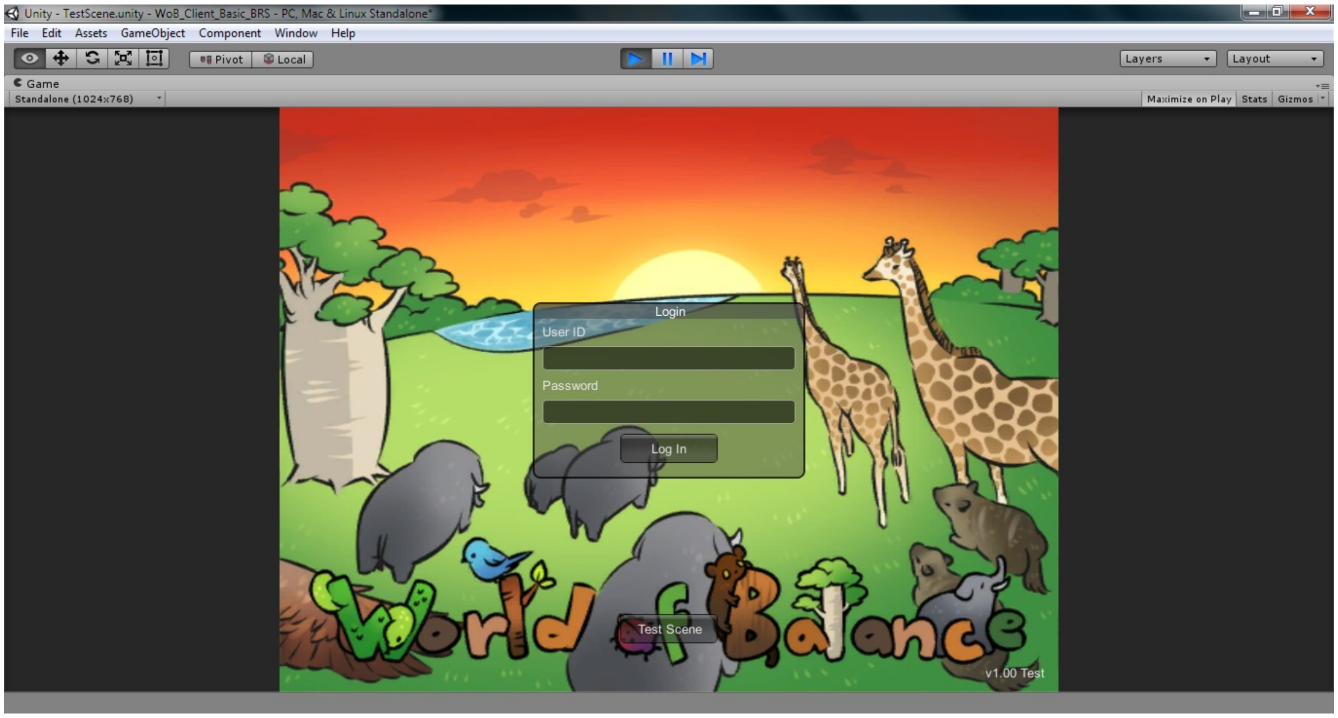
```
!dest.m_MultiFrameGUIState.m_NamedKeyControlList
```

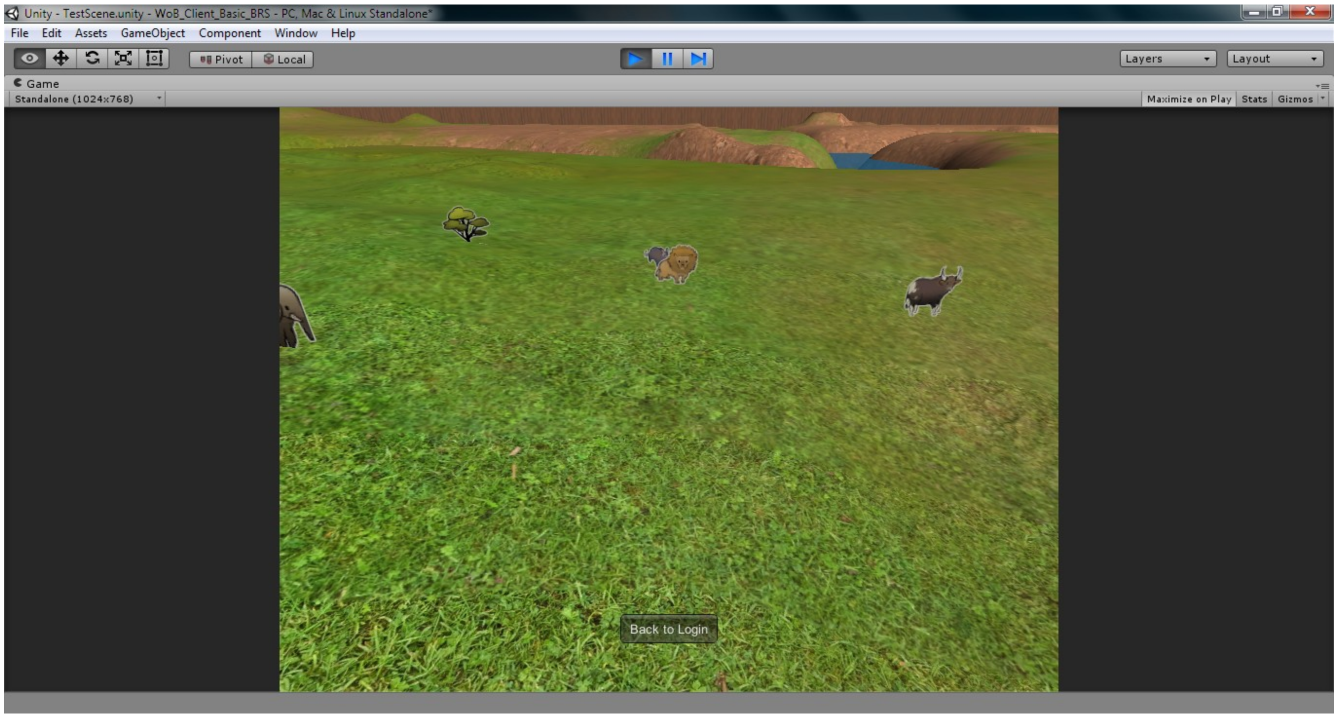
Creating a Simple GUI Element

1. First, I had to resolve a NullReferenceException that occurred due to the lack of a MainObject instance. Since it appeared the MainObject was only needed to submit the login request to the server, I simply commented out any references to it.

2. A hint posted on iLearn helped me figure out how to load the test scene when the button is clicked. The error message resulting from my first attempt helpfully informed me how to add TestScene to the build settings.

3. This was straightforward.




Unity - TestScene.unity - WoB_Client_Basic_BRS - PC, Mac & Linux Standalone*
File Edit Assets GameObject Component Window Help
Pivot
Local
Layers
Layout
Game
Standalone (1024x768)
Maximize on Play
Stats
Gizmos
Back to Login

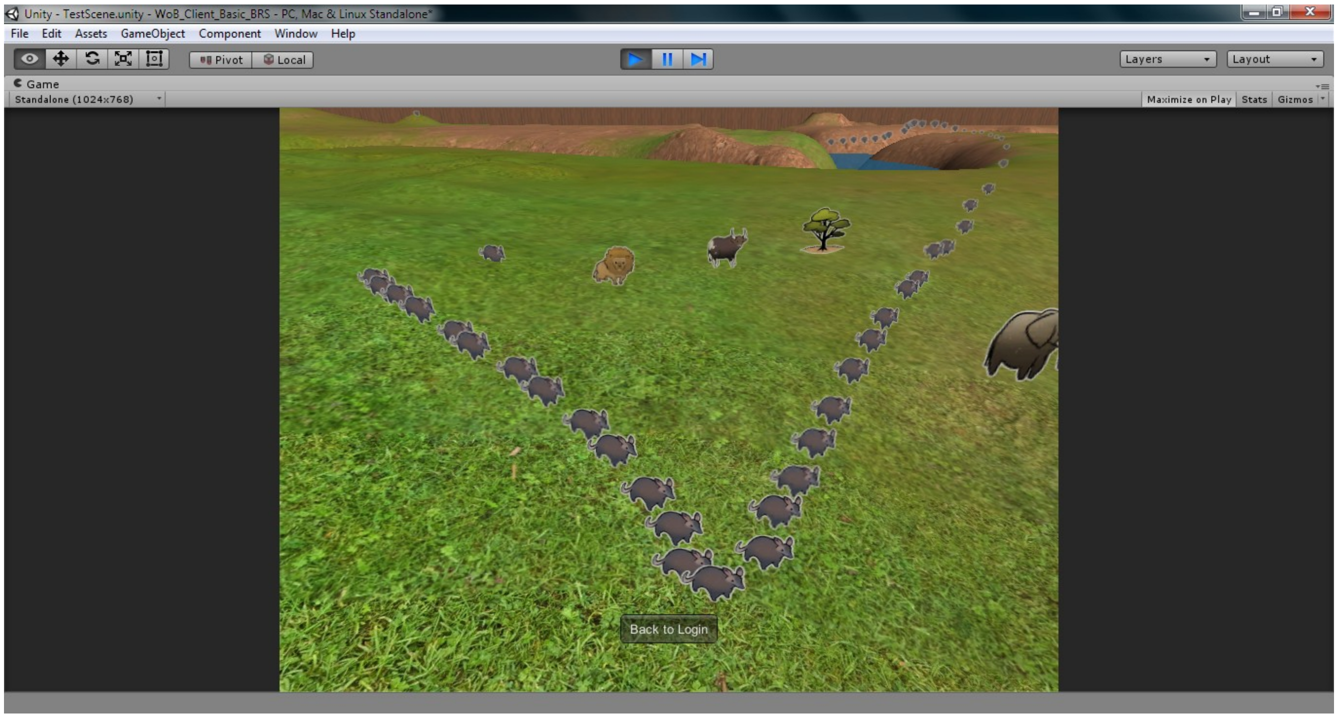
Using Collision Rays to activate mouse click on the scene

Steps 1 and 2 were straightforward.

3. I was able to write the raycasting code after reading the references someone posted on iLearn (http://stackoverflow.com/questions/20583653/raycasting-to-find-mouseclick-on-object-in-unity-2dgames and http://answers.unity3d.com/questions/761210/raycast-mouse-click-on-specific-objects-only.html), as well as the API documentation for Physics.Raycast.

4. I encountered the issue that I wanted to call my method createAnimal(), which was defined in the previous script, TestSceneScript2. I figured out how to get a reference to the instance of TestSceneScript2 based on http://tiku.io/questions/391232/in-unity-how-can-i-pass-values-from-one-script-to-another, made createAnimal() a public method, and it worked.

However, I noticed that animals created near each other were being "stacked" in an odd way. I determined the cause was that the rays being cast were colliding with other animals, not just the terrain. I found the Layers chapter in the manual, which described how to use a layer mask to cast rays selectively. I created a Terrain layer, assigned the terrain to it, and used a layer mask when calling Physics.Raycast().


Unity - TestScene.unity - WoB_Client_Basic_BRS - PC, Mac & Linux Standalone*
File
Edit
Assets
GameObject
Component
Window
Help
Pivot
Local
Layers
Layout
Game
Standalone (1024x768)
Maximize on Play
Stats
Gizmos
Back to Login