Tutorial – Game Mechanics

Open the Teleport Click Example provided.

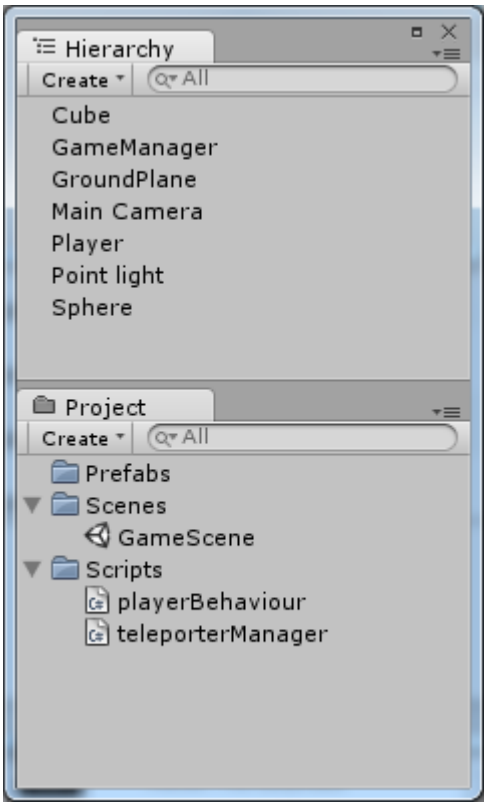
There are 2 game mechanics in play.

- Random active tile
  Rules:
  - Only one tile can be active at a time
  - An active tile will change colour when the mouse is over
  - If a tile is clicked on, it will activate the Teleporting Box.
  - A new box should be randomly selected every 2 seconds, and not be the same box twice in a row.
- Teleporting box
  Rules:
  - Can only teleport when the mouse clicks on an active tile
  - The box will teleport from any location to the location of the box.
  - Any force / momentum applied to the box before teleporting will remain after its teleported. Think portal ("speedy thing goes in, speedy thing comes out")

Have a play with the sample project and experiment with how this mechanic works. In this tutorial we are going to walk through step-by-step to achieve the same result.

Setting Up Your Project:

1. Create folders in your project, "Scenes", "Prefabs", "Scripts"
2. As a rule of thumb, you should always save your scene every chance you get. Save the scene as "GameScene", and place it in the "Scenes" Folder
3. Create a plane in the hierarchy, call it "GroundPlane". Make sure its position is set to the origin(0, 0, 0)
4. Create a cube in the hierarchy, call it "Player" Make sure its position is set to (0, 0.5, 0)
5. Add a light into the scene, we won't be doing anything special with the light, I used a point light with default settings, position set to (0, 5, 0)



© AIE 2015

6. Finally add an empty game object into the scene, call it “GameManager” we will be dragging scripts onto this object for managing various object groups within the game.

7. Create 2 C# scripts, called “playerBehaviour”, “teleporterManager”.
   in this project, Behaviour scripts will be applied to various game objects, Manager scripts will be applied to the “GameManager” object

Scripting the Teleporter Manager:

The teleporterManager script will be responsible for telling the player to teleport to a location. We instruct the script to teleport the player to the intersection location of whatever object the mouse happens to click on.

Drag the “teleportManager” script onto the “GameManager” object and then open the script for editing.

Create a function called “DetectTeleportLocation” that returns a Vector3 the function will return false if no location was found, otherwise, the teleport location will be returned through the parameters.

Refer to the code, and comments below on how to achieve this.

```
// returns true if a valid teleport was found
// the location to teleport is returned through the out Vector3 result
bool DetectTeleportLocation(out Vector3 position, out Vector3 normal)
{
	position = new Vector3();

	// if the left mouse button has been pressed
	if (Input.GetMouseButtonDown(0))
	{
		// define a ray moving into the screen from the mouse position
		Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
		RaycastHit hit;

		// Physics.Raycast returns true if there was an intersection
		// with any game object, the results are put into the hit
		// variable above
		if (Physics.Raycast(ray, out hit, 100))
		{
			// make the result the intersection location of the ray
			position = hit.point;

			// retunr true as we do want to teleport
			return true;
		}
	}

	// return false as we do not want to teleport
	return false;
}
```


© AIE 2015

Every update, we will test if a valid teleport location was found, and tell the player to teleport. For the moment, we will just print the position to the console. Later we will send a message to the player informing it where to teleport.

```
// Update is called once per frame
void Update ()
{
	// if detectTeleportLocation teturns true, than teleport the player
	Vector3 teleportTo = new Vector3();
	if( DetectTeleportLocation(out teleportTo ) )
	{
		// we are just going to print to the console window for now
		print ( teleportTo );
	}
}
```

Hopefully you should have a project that compiles. Run the game and click on any object within the world, the teleport to position should be printed to the screen.

Scripting the PlayerBehaviour:

The playerBehaviour script will be applied to the player object, it will be responsible for updating the player and making sure it behaves as intended, in our case, teleporting to desired location.

Click and drag the player behaviour script onto the player object in the hierarchy, and then open the script to edit.

Create a function in called “Teleport” that that takes in a vector3 for the position. For now we will directly move the position of the player, just to make sure it works. Later we will modify it so we have a teleport out animation and a teleport in animation.

Add the following to the playerBehaviour script

```
void Teleport(Vector3 location)
{
	// move the position of the player to the teleport location
	transform.position = location;
}
```

That was easy, jump back to the teleportManager script, and add a public GameObject variable to represent the player.

Highlighted areas refer to code that has been added or changed.

© AIE 2015

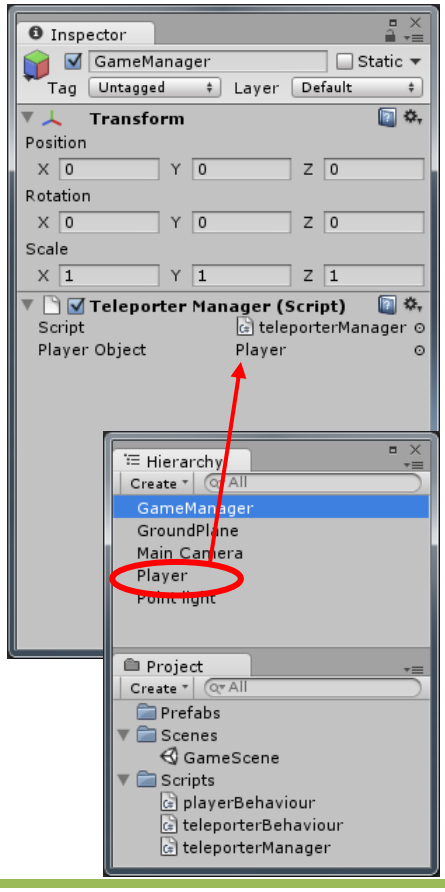
```
public class teleporterManager : MonoBehaviour {

	public GameObject playerObject = null;

	// Use this for initialization
	void Start () {

	}

	// Update is called once per frame
	void Update ()
	{
		// if detectTeleportLocation teturns true, than teleport the player
		Vector3 teleportTo = new Vector3();
		if( DetectTeleportLocation(out teleportTo) )
		{
			// you can send a message to the player object.
			// this will automatically call the players teleport
			// function that we defined earlier, as long as it exists
			if( playerObject != null )
			{
				playerObject.SendMessage("Teleport", teleportTo );
			}

			// we are just going to print to the console window for now
			print ( teleportTo );
		}
	}

 ...
}
```

One more thing we need to do to make sure the script works.

The player Object variable that we described above has a default value of null...

Select the GameManager Object, then click and drag the Player object from the hierarchy onto the Player Object in the inspector. Refer to image for clarification.

Run the game to make sure everything works as intended, you should be able to click on any object in the scene and have the player teleport to that location.



© AIE 2015

Extending the PlayerBehaviour script:

Time to make the teleporting actually look like its teleporting...

We will have a few internal states that affect the player in slightly different ways. We will define 3 states: “BeginTeleport”, “EndTeleport”, and “Normal”.

If the player is in BeginTeleport State, than we will shrink the player over time, change its position and then change to EndTeleport state, where the player will grow back to its normal size.

Below we define an enumeration to describe the player’s state. We also create a variable to record the position we want to teleport the player to, as we want the player to animate out, then move and then animate back in.

```
public class playerBehaviour : MonoBehaviour {

	// Enumeration describing the states of the player
	public enum EPlayerState
	{
		NORMAL,
		BEGIN_TELEPORT,
		END_TELEPORT
	}

	private EPlayerState		playerState = EPlayerState.NORMAL;
	private Vector3			teleportToPosition;

	// Use this for initialization
	void Start (){

	}

	// Update is called once per frame
	void Update ()
	{
		if( playerState == EPlayerState.BEGIN_TELEPORT )
		{
			// insert begin teleport update code
		}
		else if( playerState == EPlayerState.END_TELEPORT )
		{
			// insert end teleport update code
		}
	}

	void Teleport(Vector3 location)
	{
		// move the position of the player to the teleport location
		transform.position = location;

		// change the state to the ‘begin teleport’ state
		playerState = EPlayerState.BEGIN_TELEPORT;
	}
}
```


© AIE 2015

Make sure the project still compiles and runs, the player will not teleport now, we will add that back in soon.

We are going to animate the box scaling out and then sailing back in to its original scale value.

To do this we need to record the player's original position, and the scale we want the player to transition toward over time, (teleportOutScale).

teleportOutScale will be public so that we can modify the value in the inspector window, the original scale will be private.

We also require a float value to describe how far between the 2 scales the player is currently at.

Define the required variables in your code, as described below.

```
public class playerBehaviour : MonoBehaviour {

	// Enumeration describing the states of the player
	public enum EPlayerState
	{
		NORMAL,
		BEGIN_TELEPORT,
		END_TELEPORT
	}

	private EPlayerState		playerState = EPlayerState.NORMAL;
	private Vector3			teleportToPosition;

	// vector representing the scale we want the player to lerp out to.
	public Vector3 teleportOutScale;

	// record the origional scale of the object
	// so we know where to move back to.
	private Vector3 origionalScale;

	// refers to how fast the animation will play
	public float teleportOutSpeed = 1.0f;

	// record the % between lerp states
	private float teleportOutAmount = 0.0f;

	// Use this for initialization
	void Start (){

	}
```

Ok, update the teleport function, the player will no longer be able to teleport if he is halfway through the transition.

© AIE 2015

```
	void Teleport(Vector3 location)
	{
		// only teleport if we are not already teleporting...
		if( playerState == EPlayerState.NORMAL )
		{
			// record the location we want to teleport to
			teleportToPosition = location;

			// change state to begin teleport
			playerState = EPlayerState.BEGIN_TELEPORT;

			// record out origional scale
			origionalScale = transform.localScale;

			// reset the teleport amount
			teleportOutAmount = 0.0f;
		}
	}
```

Add code for the begin and end teleport animations

```
	// Update is called once per frame
	void Update ()
	{
		if( playerState == EPlayerState.BEGIN_TELEPORT )
		{
			// Update the teleportOutAmount
			teleportOutAmount += Time.deltaTime * teleportOutSpeed;

			// set the local scale
			transform.localScale = Vector3.Slerp(origionalScale,
				teleportOutScale, teleportOutAmount);

			// if teleport out amount is > 1.0 than we have finish
			// move to the end teleport state, and move the position
			// of the player.
			if( teleportOutAmount > 1.0f )
			{
				transform.position = teleportToPosition;
				playerState = EPlayerState.END_TELEPORT;
			}
		}
		else if( playerState == EPlayerState.END_TELEPORT )
		{
			// Update the teleportOutAmount
			teleportOutAmount -= Time.deltaTime * teleportOutSpeed;

			// set the local scale
			transform.localScale = Vector3.Slerp(origionalScale,
				teleportOutScale, teleportOutAmount);

			// if teleport out amount is > 1.0 than we have finish
			// move to the end teleport state, and move the position
			//of the player.
			if( teleportOutAmount < 0.0f )
			{
```

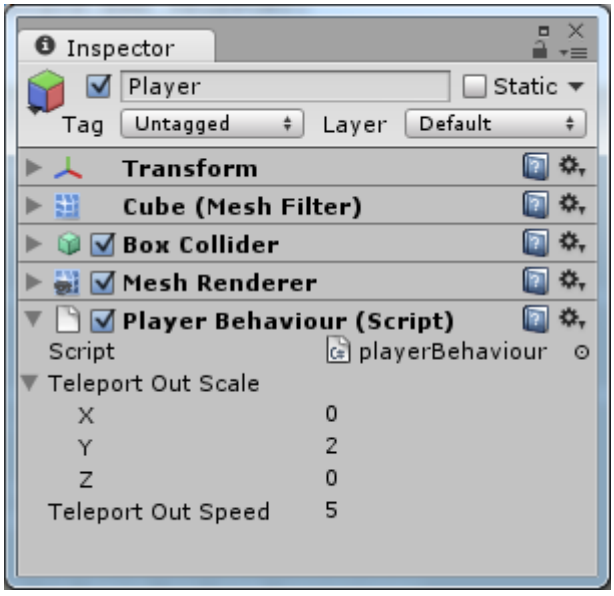

© AIE 2015

```
                transform.position = teleportToPosition;
                playerState = EPlayerState.NORMAL;

                // make sure the scale is back to the origional
                transform.localScale = origionalScale;
            }
        }
    }
```

Run and test the game, hopefully no errors have occurred... if there are errors, make sure to fix before continuing.

This looks ok, but it's not quite real enough, back in Unity, click on the player object, you should be able to change the teleport speed, and teleport out scale. Change those values just a little.

I used a Teleport out Speed of 5 and a Teleport Scale of (0, 2, 0)



Adding Some Realism:

Make sure the player object is selected and add a Rigid Body component, use the default settings...

Run and test the game again, obviously we have a problem, the player falls through the ground most of the time...

Now when we teleport, we want to move the player away a few units from surface that we clicked on, as currently we are spawning at the intersection location of the mouse and the surface we clicked on...

This means we need to get the direction of the face we click on, otherwise known as the normal.

Open the teleportManager Script; we need to re-define the DetectTeleportLocation function. This time we will also return the surfaces normal.

© AIE 2015

```
	// returns true if a valid teleport was found
	// the location to teleport is returned through the out Vector3 result
	bool DetectTeleportLocation(out Vector3 position, out Vector3 normal)
	{
		position = new Vector3();
		normal = new Vector3();
		// if the left mouse button has been pressed
		if( Input.GetMouseButtonDown(0) )
		{
			// define a ray moving into the screen from the mouse position
			Ray ray = Camera.main.ScreenPointToRay( Input.mousePosition );
			RaycastHit hit;

			// Physics.Raycast returns true if there was an intersection
			// with any game object, the results are put into the hit
			// variable above
			if( Physics.Raycast( ray, out hit, 100) )
			{
				// make the result the intersection location of the ray
				position = hit.point;
				normal = hit.normal;

				// retunr true as we do want to teleport
				return true;
			}
		}

		// return false as we do not want to teleport
		return false;
	}
```

Unfortunately, we are unable to send a message with more than 1 parameter... so instead we can define a custom structure, and store our parameters in the struct.

At the top of the TeleporterManager script add the following structure to define the parameters we need to send...

```
using UnityEngine;
using System.Collections;


public struct TeleportMessageParams
{
	public Vector3 position;
	public Vector3 surfaceNormal;
};

public class teleporterManager : MonoBehaviour {

	public GameObject playerObject = null;

	// Use this for initialization
	void Start () {

	}

...
```


© AIE 2015

We can now modify the update function, and then we will modify the teleport function for the player.

```
public class teleporterManager : MonoBehaviour {

	public GameObject playerObject = null;

	// Use this for initialization
	void Start () {

	}

	// Update is called once per frame
	void Update ()
	{
		// if detectTeleportLocation teturns true, than teleport the player
		Vector3 teleportTo = new Vector3();
		Vector3 surfaceNormal = new Vector3();
		if( DetectTeleportLocation(out teleportTo, out surfaceNormal) )
		{
			// you can send a message to the player object.
			// this will automaticly call the players teleport
			// function that we defined earlier, as long as it exists
			if( playerObject != null )
			{
				TeleportMessageParams teleportParams =
					new TeleportMessageParams();
				teleportParams.position = teleportTo;
				teleportParams.surfaceNormal = surfaceNormal;
				playerObject.SendMessage("Teleport", teleportParams );
			}

			// we are just going to print to the console window
			print ( teleportTo );
		}
	}
```

Finally we can now fix the players teleport function. Open the playerBehaviour script.

© AIE 2015

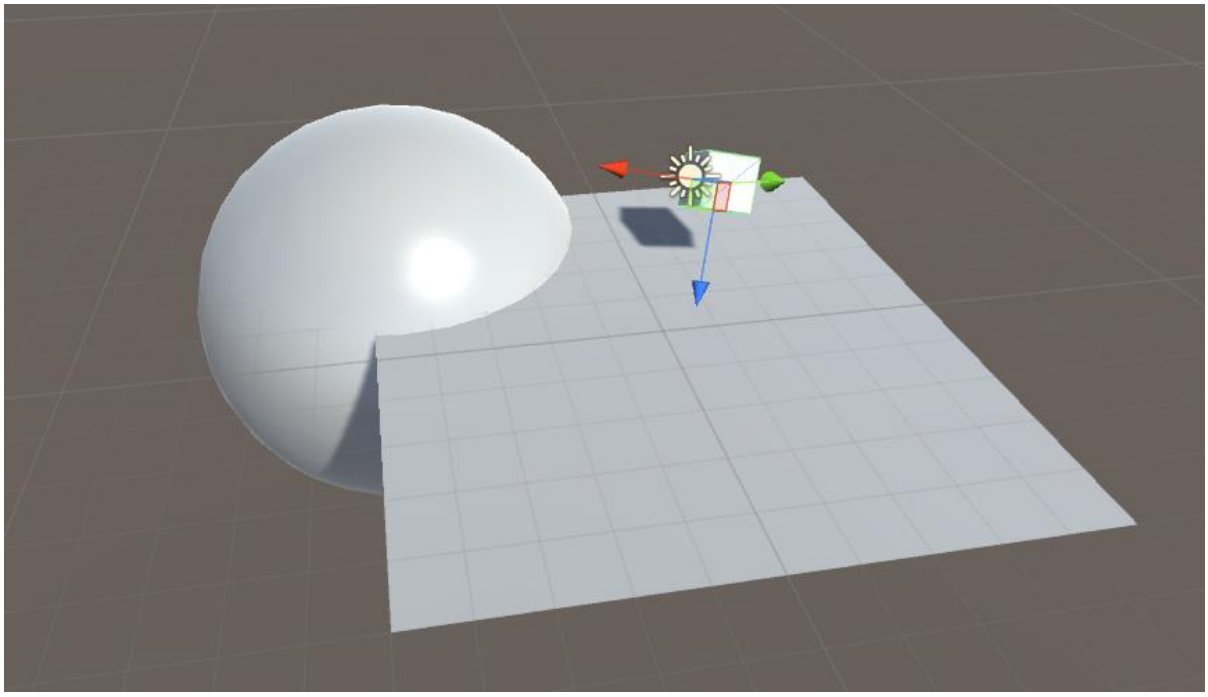
We now need to tell it to take in the TeleportMessageParams struct that we have just defined.

```
...

        void Teleport(TeleportMessageParams parameters)
        {
                // only teleport if we are not already teleporting...
                if( playerState == EPlayerState.NORMAL )
                {
                        // record the location we want to teleport to
                        teleportToPosition = parameters.position +
                                (parameters.surfaceNormal *
                                (parameters.surfaceNormal.magnitude/2));

                        // change state to begin teleport
                        playerState = EPlayerState.BEGIN_TELEPORT;

                        // record out origional scale
                        origionalScale = transform.localScale;

                        // reset the teleport amount
                        teleportOutAmount = 0.0f;
                }
        }
}
```

Once again, run the project; make sure there are no errors.

That's the Teleport mechanic completed, feel free to extend this project, include some form of environment, and add other mechanics etc.

Here's a screenshot of my version.



© AIE 2015