Unity game examples

By Mike Hergaarden from M2H (www.M2H.nl)

Introduction

Welcome to my second unity tutorial! This tutorial has been originally created for *Creative Technology, Universiteit Twente* and *Multimedia, VU Amsterdam*. The tutorial is aimed at users new to Unity scripting. This time I'm showing you a collection of game examples to demonstrate how basic unity games work. This tutorial comes with a premade example project that you can download at http://m2h.nl/unity/.

I assume you do know the basics of the Unity engine such as using prefabs and moving around the editor. If not, please start with the video tutorials right here: http://unity3d.com/support/documentation/video/.

It's important to note that this entire set of game examples have been scripted in C#. Javascript/UnityScript is a bit easier for starters, but in the end C# proves to be a bit more mature/professional. We (*M2H*) have only recently switched to C# ourselves, that's why our previous examples/tutorials used Javascript. It's up to you to choose your preferred language. If you have no/limited coding experience Javascript might be best as most community tutorials/snippets use javascript. If you can't really choose, choose C#. *(I'm hereby ignoring the Boo language completely as we have no experience with it whatsoever. I've never even met a Boo coder either..)*. Since the two languages aren't too different this tutorial is also still useful to Javascript users.

The very first game example is very basic. Make sure you understand it (and therefore also the Unity basics) before moving on to the other examples. Some of the code has been made as clean/neat as possible, but therefore the code might be harder to grasp if its all new to you.

Two more tips related to the coding languages:

1. Need to convert your Unity Javascript scripts to C#? We made a converter to help speed up this process greatly, see: http://m2h.nl/unity/
2. Make your life easy and use of one of the following free code editors:

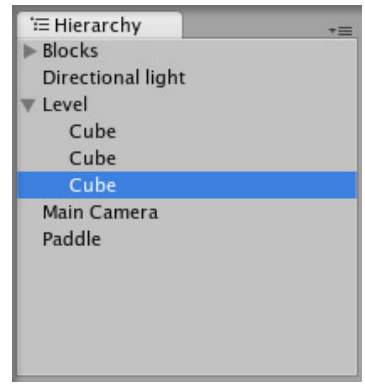
- For Javascript: MonoDevelop
- For C#: MonoDevelop or VisualStudio(C# Express/Pro). At the moment I find Visual studio(Pro) the best of these two, MonoDevelop is improving a lot recently.

Let's get started by checking out the first game!

Mike Hergaarden
M2H.nl

A brief introduction to Unity scripting

Scenes
A scene is a "world"; or when thinking about games, it's a level (ignoring the fact that you can also divide a level over several scenes and streaming those). All contents of a scene is shown in the Hierarchy. This hierarchy contains GameObjects (which all have a Transform component).



GameObjects and Transforms
Every object you have in your scene is a gameobject. Gameobjects are the entities that make up the scene. A game object could be a blank 'node'. However, you can add components to your gamenodes: Graphics/Sounds etcetera. A gameobject can as many components as you want.
Transforms describe position/rotation/parents and childs of objects. Every gameobject has a transform(and visa verse). You can use transforms to find the children/parents of your transform(/gameobject).

MonoBehaviours
"MonoBehaviour is the base class every script derives from". Simply put: Your scripts can use all of the "MonoBehaviours" default functions (http://unity3d.com/support/documentation/ScriptReference/MonoBehaviour.html). The most important functions are:

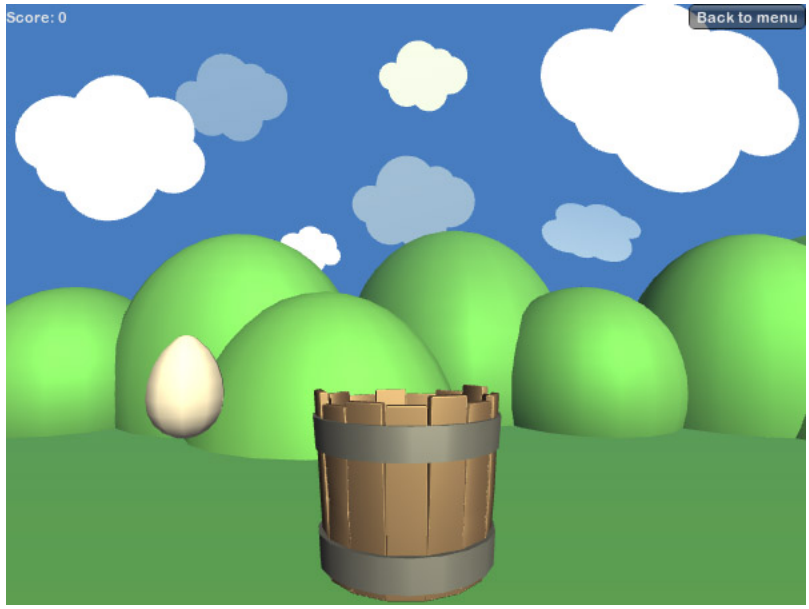
- Awake, called only once when you load a scene, before all other functions.
- Start, called at the start of a scene, after Awake.
- Update, called every frame. Use it for...anything
- FixedUpdate, called every "physics frame". Use it for physics related work.
- OnGUI, You can only use Unity GUI functions in OnGUI or functions that have been called via OnGUI. Don't put non-GUI functions in here, that's a waste of performance.

To use a script in unity create a new script, add a new gameobject to your scene and finally attach the script.

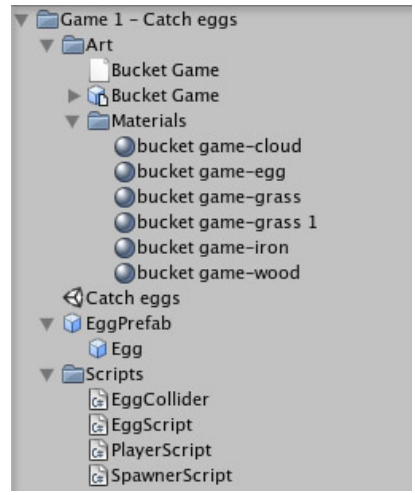
Further help
I highly recommend reading the scripting introduction by Unity, see http://unity3d.com/support/documentation/ScriptReference/index.html. After reading Unity's introduction you should be ready to continue as you've at least seen everything once by now. To fully understand everything you can see and try everything in practice.

Game 1: Catch eggs

Open the scene at "*Game 1 - Catch eggs/Catch eggs*" and run the game. Eggs fall from the sky, move your character to catch the eggs and gain score. Have a look at the scene's contents and the entire contents of the "*Game 1 - Catch eggs*" folder, figure out what's going on!



How it works



Basically four things are going on in this scene for each of which we use a script: The player (Bucket) movement, egg spawning, egg movement, collecting eggs. The assets you see to the right are all the assets that are used in this game.

Player movement: *Playerscript.cs*
The GameObject "Bucket" has the PlayerScript attached to it which is moving the bucket gameobject to the left and right using the "Horizontal Input" key that has been defined via the Unity input system as the horizontal arrow keys and the AD characters. We use this value to move the transform that the script has been attached to. After moving we check and restrict the movement on the X axis between the values -2.5 to 2.5. Do note that we also (ab)use this script to show the score GUI.

Egg spawning: SpawnerScript.cs
Using our own timer we spawn (Instantiate) the Egg prefab. Furthermore we speed up the spawn process until we spawn an egg every 0.3 seconds.

Egg movement: Eggscript.cs
The Eggscript moves the egg down, and destroys it if it's too low. We're not using any physics here(!).

Collecting eggs: EggCollider.cs

The bucket contains a hidden collider that we use as a trigger to detect eggs. Because the EggCollider gameobject has a box collider attached (marked as Trigger) we can use trigger functions. In the EggCollider script we use OnTriggerEnter to detect eggs. Since the eggs are the only physics objects in this scene we do not need to check what object we're colliding with.

Things you could improve

1. Add a game over event and allowing restarting of the game. Hint: *Application.LoadLevel(Application.loadedLevel)*;
2. Add a random "bomb"/death-item that the player should avoid instead of catch.
3. Separate the GUI from the Playerscript and create a GameGUI script instead.
4. Improve the egg movement how you see it fit (it "lags" when it touches the bucket which moves). You could use real physics to drop the egg. Don't forget to restrict the egg movement to the X and Y axis only (See Game 2).

Code snippets

EggScript.cs *Moving and possibly Destroying the egg*

```
void Update () {
    float fallSpeed = 2 * Time.deltaTime;
    transform.position -= new Vector3(0, fallSpeed, 0);

    if (transform.position.y < -1 || transform.position.y >= 20)
    {
      //Destroy this gameobject (and all attached components)
      Destroy(gameObject);
    }
        }
```

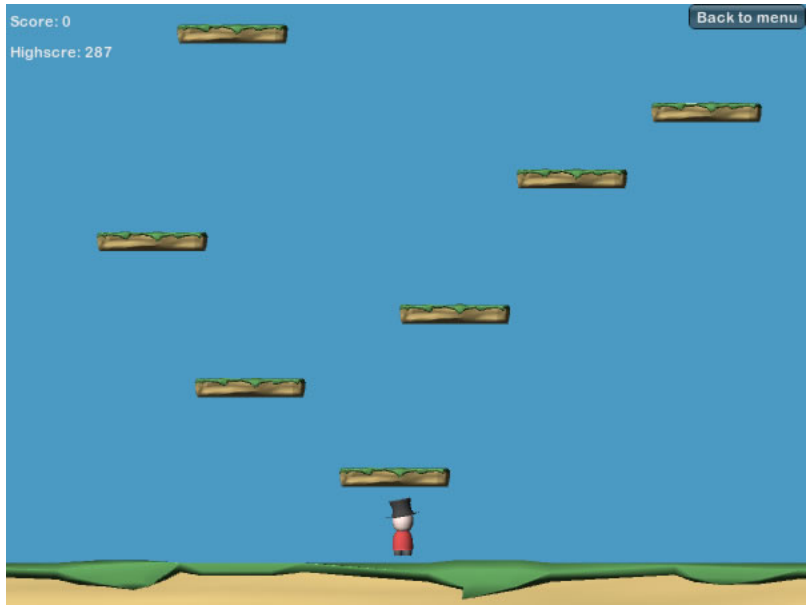
Playerscript.cs *Moving the player horizontally and clamping it between two values (no-physics)*

```
void Update () {
    //These two lines are all there is to the actual movement..
    float moveInput = Input.GetAxis("Horizontal") * Time.deltaTime * 3;
    transform.position += new Vector3(moveInput, 0, 0);

    //Restrict movement between two values
    if (transform.position.x <= -2.5f || transform.position.x >= 2.5f)
    {
      float xPos = Mathf.Clamp(transform.position.x, -2.5f, 2.5f); //Clamp between min -2.5 and max 2.5
      transform.position = new Vector3(xPos, transform.position.y, transform.position.z);
    }
        }
```

Game 2: Platform jumper

In this game the player tries to jump as high as possible, with no way back (down). New platforms are generated on the fly. The most interesting concept in this game is the platforms that are automatically spawned and deleted as you progress in the game. A simple (local) highscore is implemented using PlayerPrefs. The player is locked to a 2D world (X and Y) using a Joint and finally the game uses raycasting to check whether a player is standing on a platform.



How it works

Again, we'll go over the individual scripts of this mini-game to explain how the bigger picture works.

EdgeTrigger.cs
The camera has two child's (Trigger box colliders with the EdgeTrigger script attached). However, you can ignore these for now. These have been added as placeholder for you to implement and do not affect the game right now.

GameGUI.cs
We now have a dedicated script for the GUI of the game, combined with a basic highscore system.

Playermovement.cs
The movement on the X axis is the same as the bucket game. However, besides the left and right movement we now also have jumping, this is implemented using physics on the Y axis. We disable the physics (velocity) on the other two axes to prevent strange movements since we are using our own movement for the X axis.

The Player gameobject has a bit more to it than just the Playermovement script. We use a Rigidbody for the Y axis Physics. We use the boxcollider for collision with the platforms and finally we use a Configurable joint to restrict movement to the X and Y axis only. This configurable joint might look quite scary, for now briefly look at it and feel free to assume that it 'just works'.

GameControl.cs
This 100 line script is the heart of the game; It controls the game state and spawns "procedurally generated" random level by spawning platforms. Furthermore it also controls the camera height. Everything but the Platform spawning should be straight forward. It's not necessary to fully understand the platform spawning yet.

Things you could improve

1. Allow jumping to the far left and appear on the right (and vice verse)
2. Allow jumping through platforms (from the bottom only)
3. Improve the platform generation script. At the moment the player can get stuck.

Code snippets

GameControl.cs *Maintaining the game state and spawning platforms*

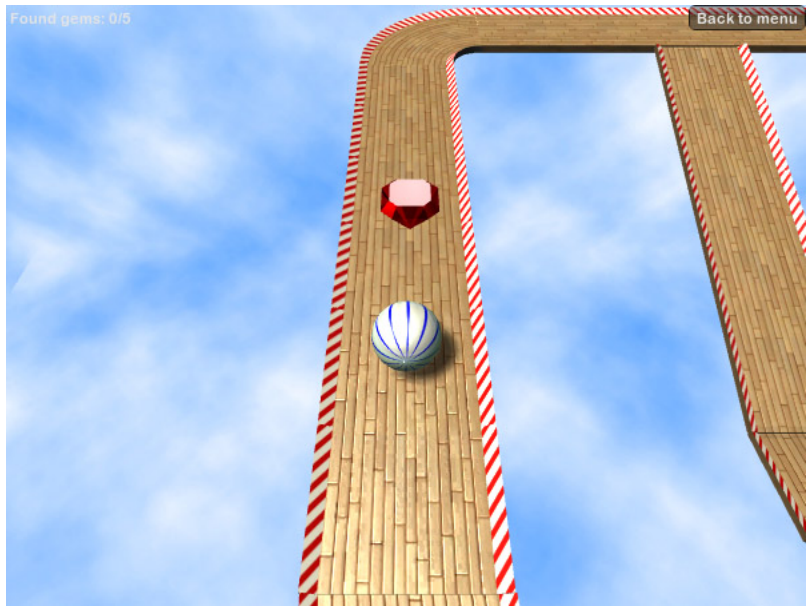
```
        void Update () {
    //Do we need to spawn new platforms yet? (we do this every X meters we climb)
    float playerHeight = playerTrans.position.y;
    if (playerHeight > nextPlatformCheck)
    {
       PlatformMaintenaince(); //Spawn new platforms
    }

    //Update camera position if the player has climbed and if the player is too low: Set gameover.
    float currentCameraHeight = transform.position.y;
    float newHeight = Mathf.Lerp(currentCameraHeight, playerHeight, Time.deltaTime * 10);
    if (playerTrans.position.y > currentCameraHeight)
    {
       transform.position = new Vector3(transform.position.x, newHeight, transform.position.z);
    }else{
       //Player is lower..maybe below the cameras view?
       if (playerHeight < (currentCameraHeight - 10))
       {
          GameOver();
       }
    }

    //Have we reached a new score yet?
    if (playerHeight > GameGUI.score)
    {
       GameGUI.score = (int)playerHeight;
    }
        }
```

Game 3: Marble game

A (marble) balancing game where you guide a rolling 'ball' trough a level while collecting items. It's amazing how little scripting this game requires in Unity. This game has a Game manager that keeps track of game states (Playing, Won, Lost). This is the first game of this package that's "truly" 3D as the first two games hid the third dimension in the gameplay.



How it works

Compared to the previous game the code behind this Marble game is surprisingly easy.

GameoverTrigger.cs
This script simply sets the game to a GameOver state as soon as something touches it. This trigger is used on an big invisible box collider at the bottom of the level in case you fall off.

MarbleCamera.cs
This simple camera script follows the position of a target object, but ignores the rotation.

MarbleControl.cs
Surprisingly, the physics driven marble movement takes just two lines. What's might be new to you in this script is the use of gameobject tags to detect what the marble is colliding with; if we're colliding with a "Pickup" object we know we've found a game and act accordingly.

MarbleGameManager.cs
This script combines a our game management and GUI in one. Once again we make smart usage of tags to automatically count the total amount of gems we need to find in this level.

Things you could improve:

1. Expand the level
2. Add interactable objects (A pad that launches the ball?)
3. Certain "triggers" that unlock new paths. (Switches, keys, doors)

Code snippets

MarbleCamera.cs *Follow the position of a target only, no rotations*

```
public Transform target;
        public float relativeHeigth = 10.0f;
public float zDistance = 5.0f;
public float dampSpeed = 2;

        void Update () {
  Vector3 newPos = target.position + new Vector3(0, relativeHeigth, -zDistance);
  transform.position = Vector3.Lerp(transform.position, newPos, Time.deltaTime*dampSpeed);
        }
```

MarbleControl.cs *Controls the marble and* checks for item pickups/collision

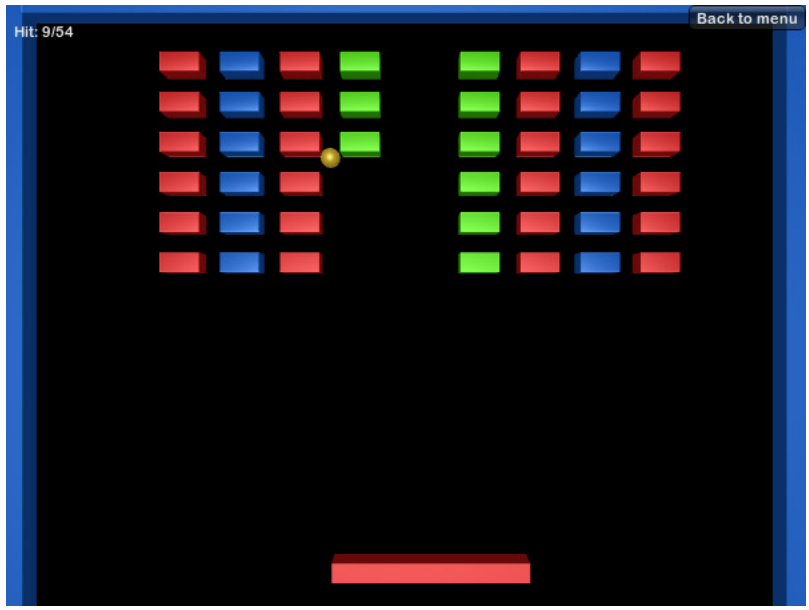
```
public float movementSpeed = 6.0f;


        void Update () {
  Vector3 movement = (Input.GetAxis("Horizontal") * -Vector3.left * movementSpeed) + (Input.GetAxis("Vertical") *
Vector3.forward *movementSpeed);
  rigidbody.AddForce(movement, ForceMode.Force);
        }

void OnTriggerEnter  (Collider other  ) {
  if (other.tag == "Pickup")
  {
    MarbleGameManager.SP.FoundGem();
    Destroy(other.gameObject);
  }
  else
  {
    //Other collider.. See other.tag and other.name
  }
}
```

Game 4: Breakout

The legacy game Breakout remade in Unity. While this game is not really testing Unity's capabilities it is a perfect game to learn the basics of games. Now with unity we can make the game use real physics, you could even have the game make use of a 3D world; That would make a pinball table.



How it works

For the breakout game we have three main objects: The paddle, blocks and the balls.

Block.cs
We only use this script to detect a hit and destroy this block.

Ball.cs
This script doesn't do much either. It ensures the ball always has a velocity between 15 and 20 and checks if the ball is below z -3. If so, it's game over.

Paddle.cs
Quite similar to the previous player movement, we move the paddle to the left and right and clamp it between two X values. One bit of magic is the "OnCollisionExit", this adds a left or right movement to the ball when it leaves the paddle. Without this the ball would go up and down forever in a straight vertical line, unless you miss it.

BreakoutGame.cs
While this is yet a pretty important script since it's the games backbone, nothing in this script is new compared with the previous games.

Things you could improve:

1. Add upgrades/power ups

2. Add scoring (combos?)
3. Create a Pinball game out of this game

Code snippets

Ball.cs *Enforce a min and max speed at all times.* Also checks for gameover

```
        void Awake () {
    rigidbody.velocity = new Vector3(0, 0, -18);
        }

        void Update () {
    //Make sure we stay between the MAX and MIN speed.
    float totalVelocity = Vector3.Magnitude(rigidbody.velocity);
    if(totalVelocity>maxVelocity){
       float tooHard = totalVelocity / maxVelocity;
       rigidbody.velocity /= tooHard;
    }
    else if (totalVelocity < minVelocity)
    {
       float tooSlowRate = totalVelocity / minVelocity;
       rigidbody.velocity /= tooSlowRate;
    }

    //Is the ball below -3? Then we're game over.
    if(transform.position.z <= -3){
       BreakoutGame.SP.LostBall();
       Destroy(gameObject);
    }
        }
```

Paddle.cs *Clamped horizontal movemen. Adds a swing to the ball when it's being hit by the edge of the paddle.*

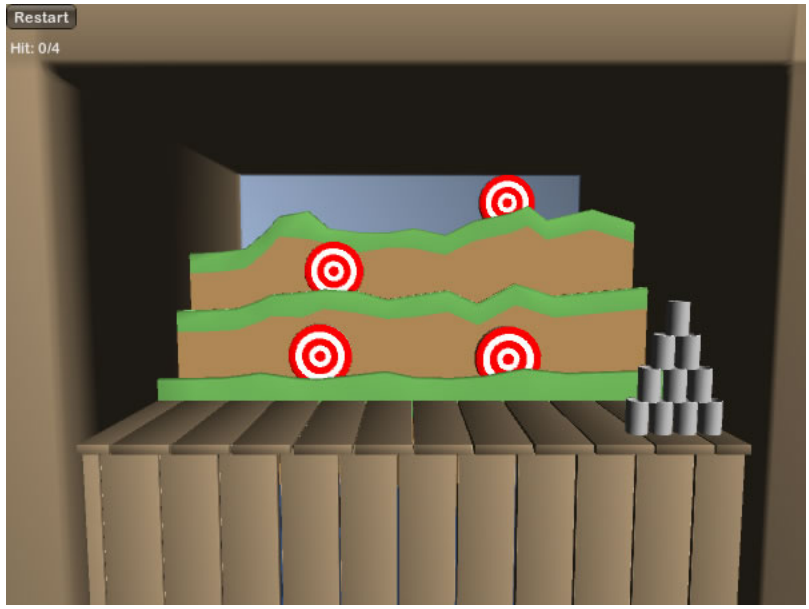
```
  void Update () {
    float moveInput = Input.GetAxis("Horizontal") * Time.deltaTime * moveSpeed;
    transform.position += new Vector3(moveInput, 0, 0);

    float max = 14.0f;
    if (transform.position.x <= -max || transform.position.x >= max)
    {
       float xPos = Mathf.Clamp(transform.position.x, -max, max); //Clamp between min -5 and max 5
       transform.position = new Vector3(xPos, transform.position.y, transform.position.z);
    }
        }

  void OnCollisionExit(Collision collisionInfo ) {
    //Add X velocity..otherwise the ball would only go up&down
    Rigidbody rigid = collisionInfo.rigidbody;
    float xDistance = rigid.position.x - transform.position.x;
    rigid.velocity = new Vector3(rigid.velocity.x + xDistance/2, rigid.velocity.y, rigid.velocity.z);
  }
```

Game 5: Shooting gallery

A pretty important feature you probably haven't really used so far is raycasting (the platform game uses it for ground detection). Therefore I'm introducing this shooting range which uses raycasting to shoot targets. Furthermore this game also introduces basic sounds.



How it works

GameManager.cs
The game manager shows the simple GUI, maintains a list of targets, moves these targets and removes these targets.

ObjectSpawner.cs
For nicer looking code we're using the DirectionEnum, however this could be replaced by using a regular int instead. We also have a tidy class for the MovingObject data. The rest of the script is straightforward spawning of targets using a timer, registering the spawned objects at the GameManager.

PlayerInput.cs
This is where the two new features are introduced: sounds and raycasting. Every time the (left) mouse button is pressed we cast a ray and detect whats hit. When we hit a Target ("ShootingObject" tag) we play the audioclip of the audiosource attached to the gameobject of this script. We also set a random pitch to spice up the sound. The GameManager is used to remove the targets right away. If we hit a can("Can" tag) we play the AudioClip that was passed via the inspector, this does not interfere with the audioclip attached to the audiosource. We use a fancy unity physics function to blow all cans away on a hit.

Things you could improve:

1. Spawn a particle effect when targets are hit for smooth removal of these targets.
2. Add some goals to the game (Scoring, Game over event)

Code snippets

PlayerInput.cs *Shoot at Cans and targets + audio*

```
void Update () {
    if(Input.GetMouseButtonDown(0)){
        Ray ray = Camera.main.ScreenPointToRay (Input.mousePosition);
        RaycastHit hit;
```

```
        if (Physics.Raycast (ray, out hit, 100)) {
            if (hit.transform.tag == "ShootingObject")
            {
                audio.pitch = Random.Range(0.9f, 1.3f);
                audio.Play();
                GameManager.SP.RemoveObject(hit.transform);
            }else if(hit.transform.tag == "Can"){
                audio.PlayOneShot(canHitSound);
                Vector3 explosionPos = transform.position;
                hit.rigidbody.AddExplosionForce(5000, explosionPos, 25.0f, 1.0f);
            }
        }
    }
            }
```

ObjectSpawner.cs *Spawn new targets*

```
public enum DirectionEnum{left = -1, right= 1}

void Update () {
    if ((lastSpawnTime + spawnInterval) < Time.time)
    {
        SpawnObject();
    }
            }

 void SpawnObject()
 {
    lastSpawnTime = Time.time;
    spawnInterval *= 0.99f;//Speed up spawning

    DirectionEnum direction = spawnDirection; //-1 or 1 (left, right)
    Transform newObj = (Transform)Instantiate(objectPrefab, transform.position, transform.rotation);
    MovingObject movObj = new MovingObject(direction, newObj);
    GameManager.SP.AddTarget( movObj );
 }
```

The main menu

This project comes with a main menu scene that allows you to easily browse and play all games. It uses the unity function "DontDestroyOnLoad(this);" to make the attached gameobject persistent; it won't be removed when loading a new scene like all objects normally are. So when we start a game from the main menu, this script will be part of the games scene as well; We use this to add an GUI 'overlay' to the game to be able to return to the main menu. When we do return to the main menu from inside a game, we remove the persistent object so that there won't be two persistent objects when you load the main menu scene again.
Note: you need to add all scenes to the build menu for the menu to work!

Code snippets

MainMenuScript.cs *A persistent object*

```
void Awake(){ //Make this script persistent(Does not destroy when loading a new level)
   DontDestroyOnLoad(this);
}

            void OnGUI () {
   if (Application.loadedLevel == 0){//Detect if we're in the main menu scene
      MainMenuGUI();
  else
      InGameGUI();
           }

void InGameGUI(){
   if (GUILayout.Button("Back to menu")){
      Destroy(gameObject); //Otherwise we'd have two of these..
      Application.LoadLevel(0);
   }
}
```

Congratulations!

Maybe you've skimmed trough this tutorial, *or* you've explored every detail of Unity that this tutorial touched. Either way; You must have learned quite a bit to continue your (game/)multimedia development.

There's loads more to explore, now that you feel comfortable with the basics of Unity it's all downhill from here. Good luck and enjoy your adventure!

Mike Hergaarden

PS: Loved this tutorial? Check out our other assets or buy us a beer at Unite ;)!