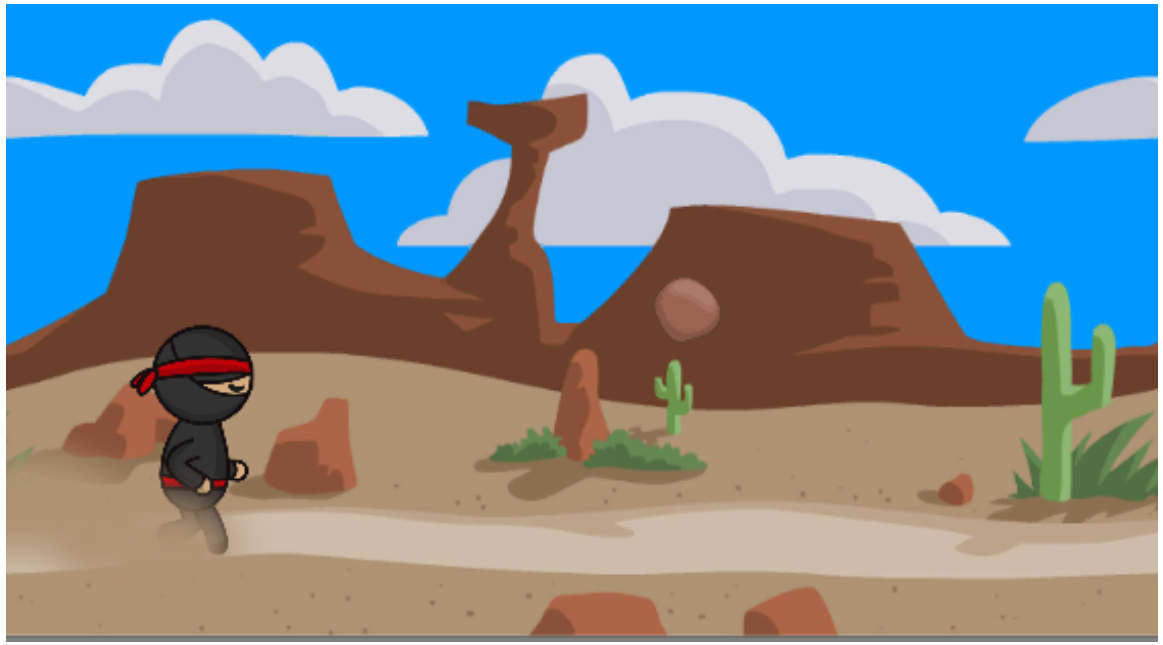
Endless Runner

Scrolling Background



Introduction

- Problem driven
  - Still looking at problems and solutions
  - Problem solving techniques
- Animation in Unity
  - Recap
- Moving objects with code
  - C# Introduction

Today's Problems

1. I want to have an **animated** background in my endless runner game.
2. I want to have obstacles my runner has to avoid.

Problem: I want to have an animated background in my endless runner game

Motivation

- The game has no background, which means the Ninja is running on the spot.
- We need a background image to set the scene.
- We'd like the background to scroll from right to left.

Solution(s)

- This is quite a big problem, so we'll start by deploying our problem solving techniques.

Problem Solving Techniques (Problem 1).

- Divide and conquer
  - Break the problem into smaller problems
    a. How to create a background?
    b. How to move the background image?
    c. How to create the effect that the background is infinite?
- Simplest problem first
  - Lets create a static background
  - Then move it
  - We'll leave the making it 'endless' until last

- Change the problem
  - No need we've already seen a number of similar problems solved.

Problem: How to create a background?

Motivation

- The game currently has no background image, we'd like to set the scene of the game by providing one.

Solution(s)

- Add a background image?
  - Great but what do we mean by background?
- Worksheet (most of this is recapping)

Aside: Backgrounds in 2D Games

- Full-size
  - Self contained images
- Tiles
  - Smaller elements arranged in a mosaic fashion
  - Can be re-used

Backgrounds in 2D Games (cont.)

TABLE 13-1: Comparison of Full-Size Backgrounds and Background Tiles

Game Level Background Technique	*Advantages*	*Disadvantages*
Full-size backgrounds	■ Easy to design and create. ■ Easy to integrate into a game.	■ Can consume large amounts of memory and disk space making their use prohibitive for anything but small games.
Background tiles	■ Extremely efficient in terms of disk and memory space. Can be used to create hundreds of game levels in the same amount of space that a single full-size background might require. ■ Very flexible. Can be used to render many distinct game levels by mixing and matching individual tile elements.	■ Difficult to design. Background tiles require a high degree of precision to create. ■ Require significant design time. Background tiles need more design time in order to ensure that they are created properly.

From: [1]

Backgrounds in 2D Games (cont.)

History

- Background tiles can trace their origins to the earliest arcade/home game systems
- Memory (ROM/RAM) were tight, full size backgrounds were not an option
- Tiling allows the **re-use** of tiles

Creation

- Takes some effort to create tiles, but the compensation is they can be re-used.
- Use a grid tool or specialised program (e.g. Map Editor).

Suggested Background Types

TABLE 13-2: Summary of Game Level Backgrounds in Arcade Games

Arcade Game Sub-genre	*Full-size Backgrounds*	*Background Tiles*	*Comments*
Maze/chase	✓	✓	Background tiles are recommended for maze/chase games with more than five unique game levels.
Pong games		✓	Background tiles should always be used in Pong games.
Puzzlers	✓	✓	Puzzlers with more than ten unique levels should use background tiles instead of full-size background screens.
Shooters	✓	✓	Background tiles are recommended for shooter games with more than five unique game levels.
Platformers		✓	Background tiles should always be used in platformers.

From: [1]

Problem: How to move the background?

Motivation

- We'd like the background to scroll from right to left.

Solution(s)

- Move the background
  - Using an animation curve
  - Move it using code
- Worksheet

Problem: How create an 'endless' background effect?

Motivation

- Moving the background is great, but we can see the edge?

Solution(s)

- Repeat the background image to prevent this
- Worksheet

Problem: I want to have obstacles my runner has to avoid

Motivation

- So far we have a player avatar who can jump and slide.
- The endless runner game requires obstacles to jump over and slide under.

Solution(s)

- This is quite a big problem, so we'll start by deploying our problem solving techniques.

Problem Solving Techniques (Problem 2).

- Divide and conquer
    - Break the problem into smaller problems
        a. How to create copies of the obstacle (boulder).
        b. How to move these towards the player?
        c. How to check for player hit?
        d. How to end the game when the player is hit?
- Simplest problem first
    - Copy the bolder
    - Move the copy
    - Check and respond to collisions.
- Change the problem
    - N/A

Problem: How to create copies of the object?

Motivation

- So far we have one boulder in our scene (just out of frame), we'd like to have more.
- Ideally we'd like to randomly create them at different y coordinates (off screen) before moving them towards the player

Solution(s)

- Copy the object
  - Using code (but what code)?
  - See worksheet.
- Use a prefab
  - We'll cover this later.

Aside: Programming in Unity

- Game-play / Rules / etc.. cannot be created from components
  - We need code
- Unity supports a number of languages
  - Unity Script (Java Script based)
  - C#
  - Boo

Boo

- Object orientated, strongly typed
- Compatible with CLI (Common Language Infrastructure)
- A Python inspired syntax
- Dropped by Unity in 2014 (due to small user base)
- Homepage

Unity Script

- Object Orientated, Weakly typed
  - Some argue its an advantage for web programming
- Can use `#pragma strict` but this violates one of the key principles of JavaScript
- A lot like JavaScript
  - But does not implement the Manuscript specification
    - More like JScript.NET? (see article
  - Similar but different (hence its known as unity script).
- Not close enough to 'real' JavaScript to be useful beyond Unity.

C#

- Multi-paradigm
  - Imperative
  - Declarative
  - Functional
  - Generic
  - Object Orientated (class-based)
- Strongly typed
  - Compiler can catch more errors
- A lot like ... oh wait it is C#
  - But C# is a lot like Java, which is like C.

- Project Cool (C-like Object Orientated Language)
  - Name dropped (C# - increase in pitch over C)
- Some claims it was a Java clone
  - Came just after M$ were sued over J++
  - Designed by the same team as Turbo Pascal and Delphi
  - Increasing divergence over recent years
- More like Java/C/C++ (which we'll be doing in the course)
- Huge industry following / well documented / supported.
- See here

The Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ObstacleMovement : MonoBehaviour
{
    private Rigidbody2D obstacle;

    public float speed = -3.0f;
    public float rockLowerLimit = -4;
    public float rockHigherLimit = -1;
```

- ObstacleMovement - A class derived from MonoBehaviour
- `public / private` - attribute visibility
- In Unity - public = shown in editor.
  -Scope: Class attributes (member variables) visble to all methods (member functions)

```
    // Use this for initialization
    void Start ()
    {
        obstacle = GetComponent<Rigidbody2D>();

        //Invoke the method "SpawnObstacle" every four
        //seconds
        Invoke("SpawnObstacle", 4);
    }
```

- When the game is started `Start()` is called by the engine.
- `GetComponent<Type>()` looks for other components attached to this object of a given type
- `Invoke(...)` calls a method at regular intervals

```
    // Update is called once per frame
    void Update ()
    {

    }
```

- We're not using this yet, but when we get things to animate we'll be updating them once per frame.

```
void SpawnObstacle()
{
    float height = Random.Range(rockLowerLimit,
                                rockHigherLimit);

    Vector3 pos = new Vector3(10, height, 0);

    Quaternion rot = Quaternion.Euler(new Vector3(0, 0, 0));

    Rigidbody2D obstacleInstance = Instantiate(obstacle,
                                               pos,
                                               rot);
    obstacleInstance.name = "Obstacle(Clone)";
    obstacleInstance.velocity = new Vector2(speed, 0);
}
```

- Quaternion (actually a Unit Quaternion) - don't worry.
- `Instantiate` creates a new instance.
- Vector?

Aside: Points and Vectors

Point

- Location in space (in a coordinate system)

Vector

- Displacement between points
- Have a direction and magnitude (length).

Analogy

- Time (instant or datetime) - locations in time.
- Duration - displacement in time.

Problem: How to move the object towards the player?

Motivation

- So we can spawn boulders at a random height, this is great but we want them to move.

Solution(s)

- We saw a per frame update earlier, can we use this?
    - Yes - but we'd probably use a physics based update
- We can add a 2D Physics component and just set the velocity!
    - Didn't you just show us that, with a Vector?
    - Yes :-D

Problem(s): How to check for player hit and end the game?

Motivation

- The boulders move, now we want them to interact with the player.
- We want to detect the collisions and respond, these are part of the same problem (so I combined them).

Solution(s)

- Use a collision detection component
  - Requires that our script contains a `OnCollsionEnter2D()`
  - This is called by the Box Collider 2D component

```
    void OnCollisionEnter2D(Collision2D other)
    {
        //Debugging code, commented out for final version.
        //Debug.Log("We've hit something");

        //If the object we collided with (other has
        //a PlayerMovement (script) object attached,
        //its the Ninja

        PlayerMovement pm =
        other.gameObject.GetComponent<PlayerMovement>()

        //     ^-- get the associated game object
        //         call its GetComponent()
```

- Note debugging code
- Uses `.` notation to extract associated component and call its method.

```
        if( != null)
        {
            Time.timeScale = 0; // Stop time - end game
            Destroy(gameObject);
        }
    }
}
```

- If its the player stop time
  - Not ideal
  - Need a 'something' to track the game state to do more.
- `Destroy(...)` - Removes the game object from the active scene.

Summary

Backgrounds

- Full-size
- Tiled

Scripts

- Code which allows components to interact
- Implements game rules

Next week

- More features
- A little more code

References

[1] Feldman, A. Miscellaneous Topics and Final Thoughts Arcade Game Animation Worldware Publishing, Inc., 2001, 479-487