

# Experience Design

## Assignment 2 : Shoot-em-up

Conor O'Kane, 2013

[conor.okane@rmit.edu.au](mailto:conor.okane@rmit.edu.au)



This work is licensed under a Creative Commons Attribution 3.0 Unported License.



# Resources

[shmup-dev.com](http://shmup-dev.com)

A forum and news site for shoot-em-up developers. The forum archive has a wealth of information on techniques for programming shmups. This is a good place to get feedback from other developers.

[shmups.system11.org](http://shmups.system11.org)

A forum for shoot-em-up players and fans. Also has a development board. This is a good place to get feedback from dedicated shmup players.



# Recommended Reading

## [A Beginner's Guide to 2D Shooters – Racketboy.com](#)

Essential reading for anyone not familiar with the genre.

## [Games that Defined the Shmups Genre – Racketboy.com](#)

A list of some key titles all shmup developers should play.

## [Foundataions of Interactive Game Design: Shmups](#) - Prof. Jim Whitehead, UC Santa Cruz [PDF]

Lecture on the history, themes and evolution of shmups.



# Assignment Brief

Create a shmup playable in the Unity web player.

Your game can be an original creation, or it can be based on the provided shmup template.

**Your game must feature a novel scoring mechanic of your own design.**



# Research Time!

Read the 'Recommended Reading' list

Play some shmups!

# Scoring Mechanics Shot Counting



SCORE<1> HI-SCORE SCORE<2>  
0000 0000 0000

PLAY

SPACE INVADERS

\*SCORE ADVANCE TABLE\*

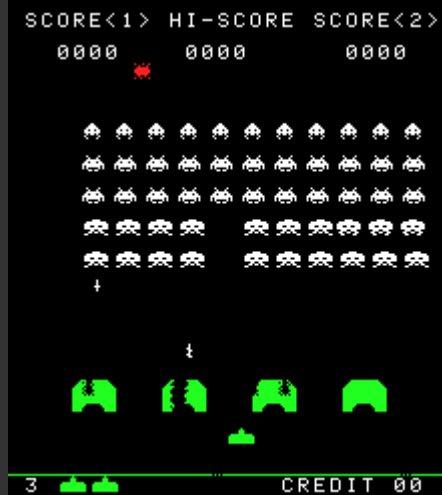
- UFO = ? MYSTERY
- Alien = 30 POINTS
- Alien = 20 POINTS
- Alien = 10 POINTS

CREDIT 00

## Space Invaders

First game to save the player's score.

To score maximum points from the UFO you must hit it with your 23<sup>rd</sup> shot the first time it appears. Thereafter you must hit it with your 15<sup>th</sup> shot.



SCORE<1> HI-SCORE SCORE<2>  
0000 0000 0000

Gameplay showing the UFO at the top, a grid of alien invaders, and the player's ship at the bottom. The UFO is the first enemy to appear.

3 CREDIT 00



# Scoring Mechanics

## Timed-chain multiplier



### DonPachi

Maintain your score multiplier by killing enemies within a short time of each other.

This requires leaving some enemies alive to bridge the gap between enemy waves and keep your combo going.

# Scoring Mechanics

## Counted-chain multiplier



### Ikaruga

Maintain your score multiplier by killing enemies in groups of three of the same color.

Killing one or two of one color followed by one of the other color resets the multiplier back to 1.

This requires careful aiming and memorization of enemy patterns.



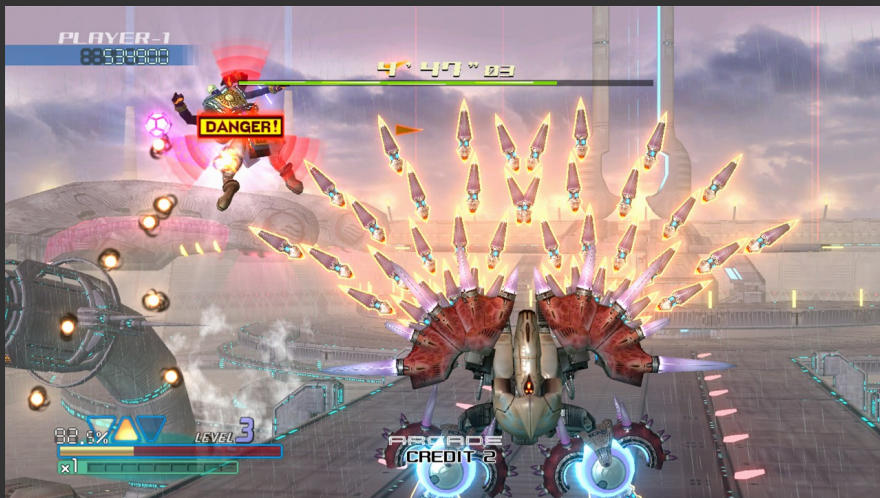
# Scoring Mechanics

## Boss Milking



### Omega Five

Rather than killing the end-of-level boss quickly, increase your score by keeping it alive longer.



This allows you to continue scoring from its attacks, only killing it at the last moment.



# Scoring Mechanics

## Grazing



## Bangai-O

Grazing means deliberately getting close to enemy bullets, without getting killed.

In many games grazing awards points directly.

In Bangai-O, triggering your counter-attack while very close to many enemy bullets greatly increases its power.



# Scoring Mechanics

## Collecting



## Trouble Witches

Collecting usually involves picking up items that enemies drop when they die, or converting their bullets into points.

In Trouble Witches players can convert enemy bullets into gold, which they then collect and spend on power-ups.

# Genre Conventions

Your game is expected to conform to many of the following shmup 'genre conventions':

## **The player is powerful, but fragile.**

The player's main weapon should be capable of destroying small enemies quickly and should be satisfying to use. Often the player has a secondary weapon which is very powerful, but limited in use.

A single hit from an enemy bullet should damage the player. In most shmups this destroys the player craft, and causes them to restart from an earlier position or lose their power-ups and score multiplier.

# Genre Conventions

## Precise Controls

Shmups require accurate movement. The player's craft or avatar should feel responsive and precise.

In traditional arcade shmups, digital 8-way controls are used.

In more modern shmups, made for console platforms, support for full analog control and movement is common.

It is very rare for a shmup to feature any inertia or damping on the player's movement; this is generally disliked by fans of the genre. Care must be taken to set up the inputs correctly, as damping is on by default in Unity.



# Genre Conventions

## Waves and Bosses

Smaller enemies should arrive in groups (called waves). Enemies that can be killed with one or two hits (popcorn enemies) should be interspersed with slightly tougher enemies.

It is common for the level to end with a large enemy (the boss fight) who takes longer to kill, and attacks the player with varying bullet patterns.

Longer levels may also features mini-bosses; unique enemies that show up mid-level to provide some variety to the gameplay.

# Genre Conventions



Boss Fight  
R-Type



Mini-boss  
R-Type Dimensions

# Genre Conventions

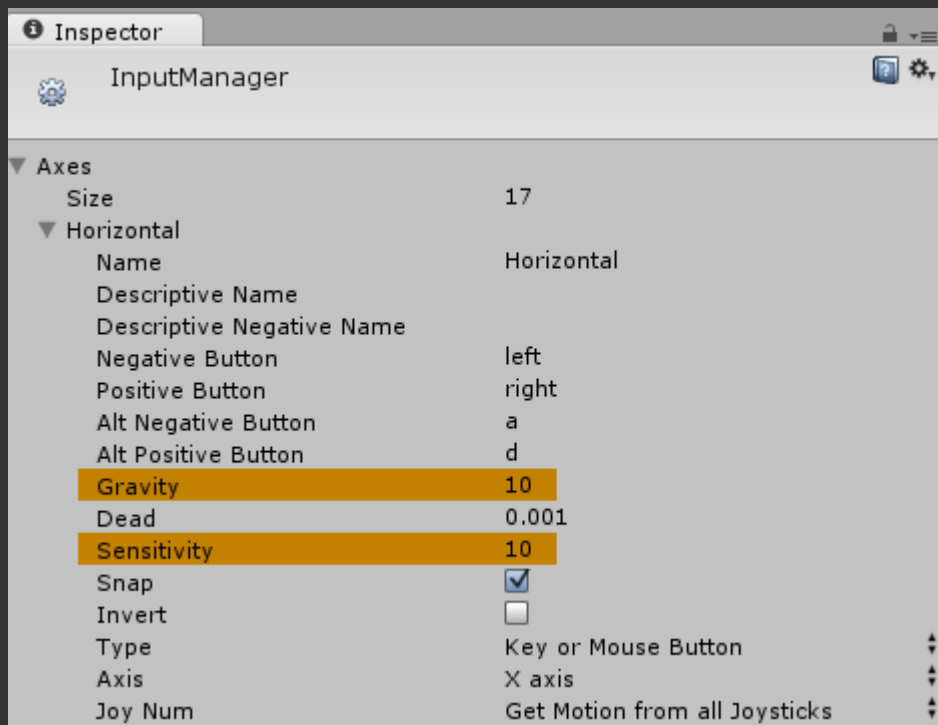
## Pacing

The gameplay should be frantic, however it is advisable to have short breaks between waves occasionally to allow the player to rest and recover.

Breaks are typically given before a mini-boss and before the final boss fight.

Enemies should be introduced in increasing order of difficulty, and if the player can collect weapons or power-ups, these should appear gradually over the first few levels, allowing the player time to learn how to use them.

# Unity Shmup Template Optimizations

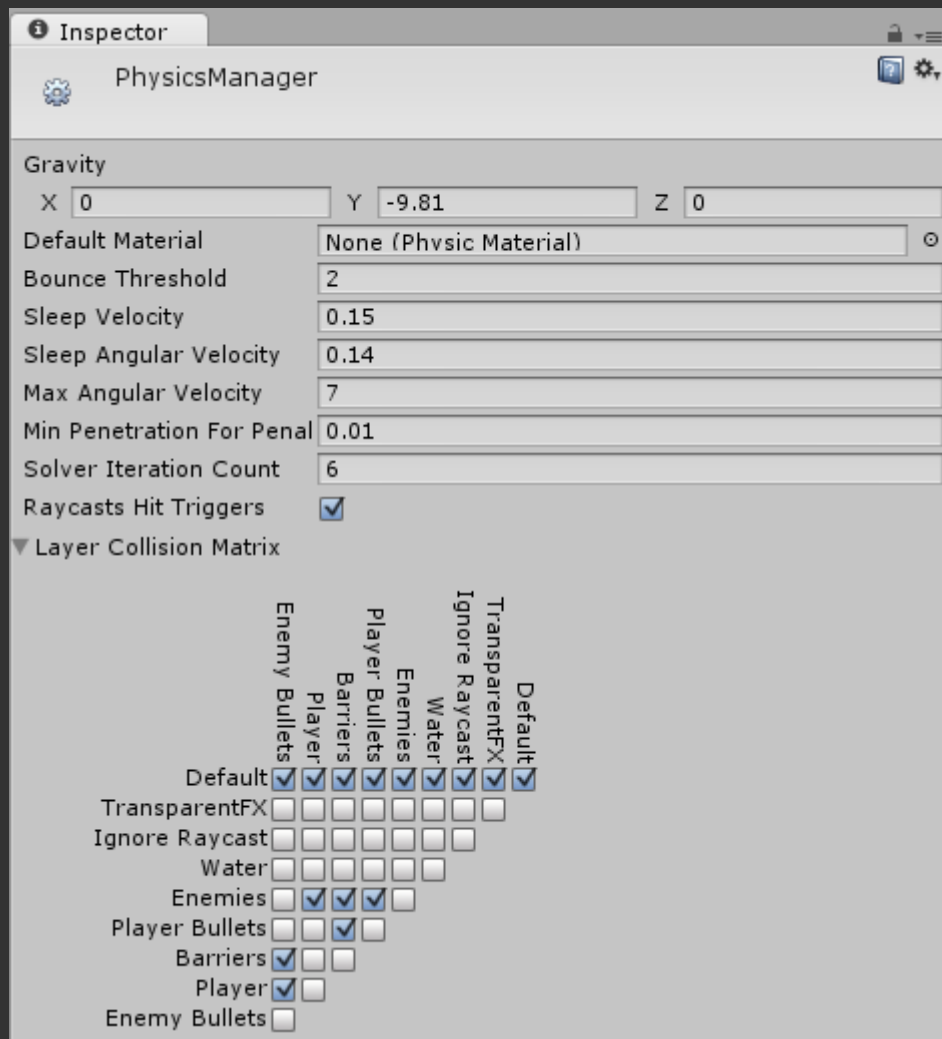


InputManager settings.

Horizontal and vertical  
input axes: gravity and  
sensitivity are set to 10.

This makes the controls  
responsive and fast.

# Unity Shmup Template Optimizations



PhysicsManager settings.

Collision layers are configured so that only essential collisions are detected.



# Unity Shmup Template Optimizations

```
static var playerBulletStack = new Stack.<Bullet>(); // a stack to store all the player bullets  
static var enemyBulletStack = new Stack.<Bullet>(); // a stack to store all the enemy bullets
```

Bullets are not instantiated at run-time.  
Instead many bullets are stored on a  
stack during start-up.

```
// shooting  
if (Input.GetButton ("Fire1") && Time.time > nextFire)  
{  
    // delay the next shot by the firing rate  
    nextFire = Time.time + fireRate;  
  
    // get a bullet from the stack  
    var newBullet = gameManager.playerBulletStack.Pop();
```

When needed, bullets are taken from the  
stack and 'fired'.

# Unity Shmup Template Optimizations

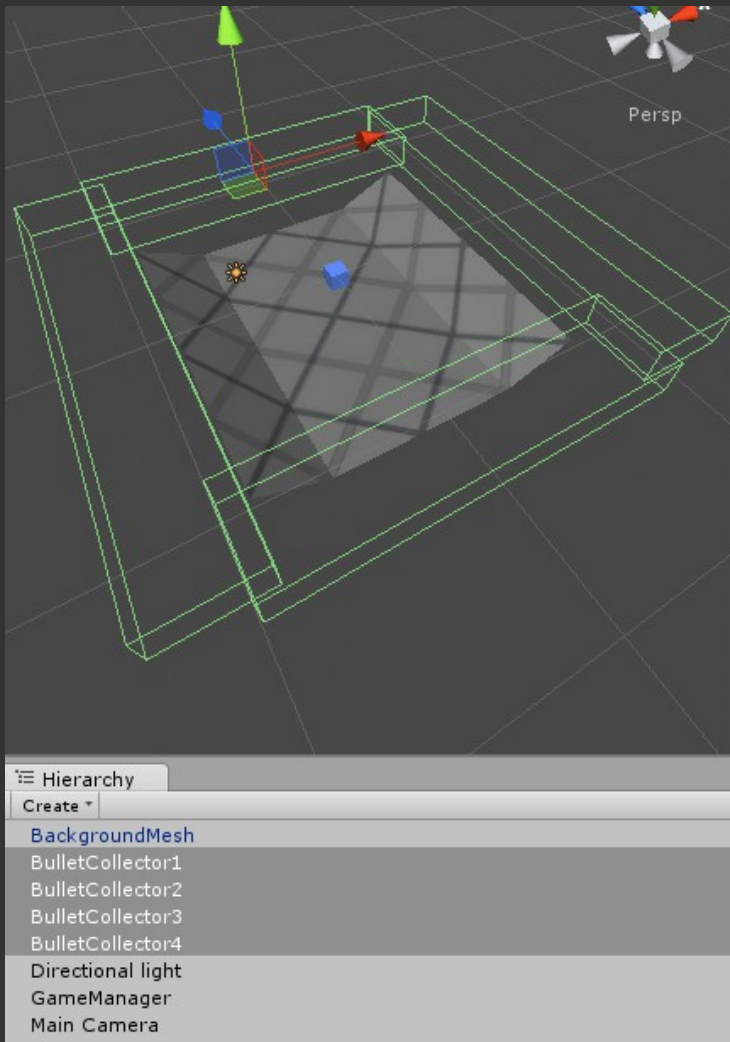
```
function OnTriggerEnter(other : Collider)
{
    if (other.CompareTag("PlayerBullet")) // hit by a bullet
    {
        TakeDamage(1); // take away 1 hit point

        // disable the bullet and put it back on its stack
        other.gameObject.active = false;
        gameManager.GetComponent(GameManager).playerBulletStack.Push(other.GetComponent(Bullet));
    }
}
```

When bullets hit an enemy they are disabled and put back into the pool for later use.

This is faster than instantiating bullets at run-time.

# Unity Shmup Template Optimizations



Triggers surrounding the scene detect bullets that have left the camera view, disable them and return them to their correct stack.

# Unity Shmup Template Optimizations

```
function Update ()
{
    // read movement inputs
    var horizontalMove = (playerSpeed * Input.GetAxis("Horizontal")) * Time.deltaTime;
    var verticalMove = (playerSpeed * Input.GetAxis("Vertical")) * Time.deltaTime;
    var moveVector = new Vector3(horizontalMove, 0, verticalMove);

    // prevent the player moving above its max speed on diagonals
    moveVector = Vector3.ClampMagnitude(moveVector, playerSpeed * Time.deltaTime);
}
```

Control inputs are clamped to prevent faster movement on diagonals. This is important when the game is played with digital inputs such as an arcade stick or keyboard.

# Unity Shmup Template Optimizations

```
var myTransform : Transform; // for caching
```

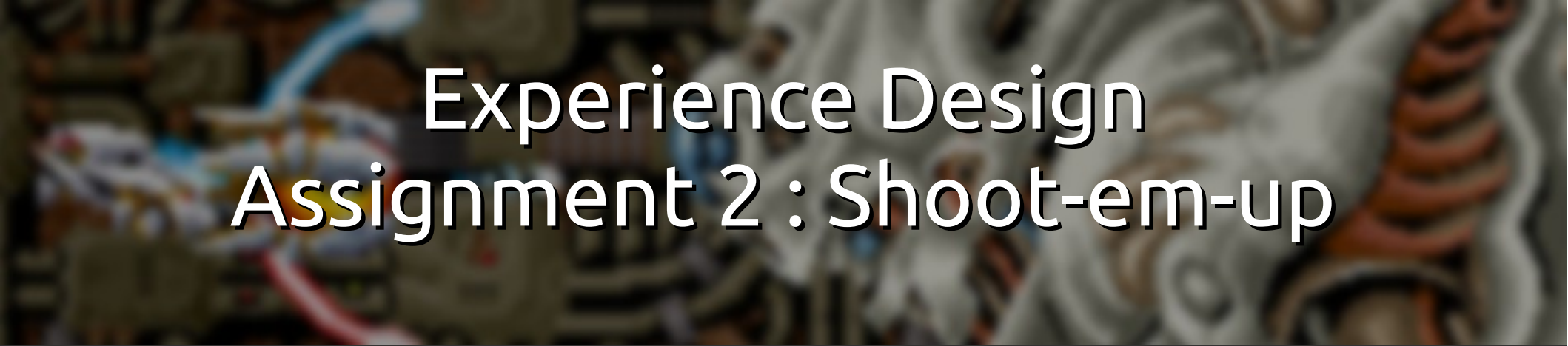
```
function Start ()  
{  
    myTransform = transform;  
}
```

```
// move the player  
myTransform.Translate(moveVector);
```

Saving the player's transform into a variable and then addressing the variable instead of 'transform' is faster.

This should be used on any objects where the transform is accessed frequently (such as in OnUpdate).





# Experience Design Assignment 2 : Shoot-em-up

Conor O'Kane, 2013

[conor.okane@rmit.edu.au](mailto:conor.okane@rmit.edu.au)