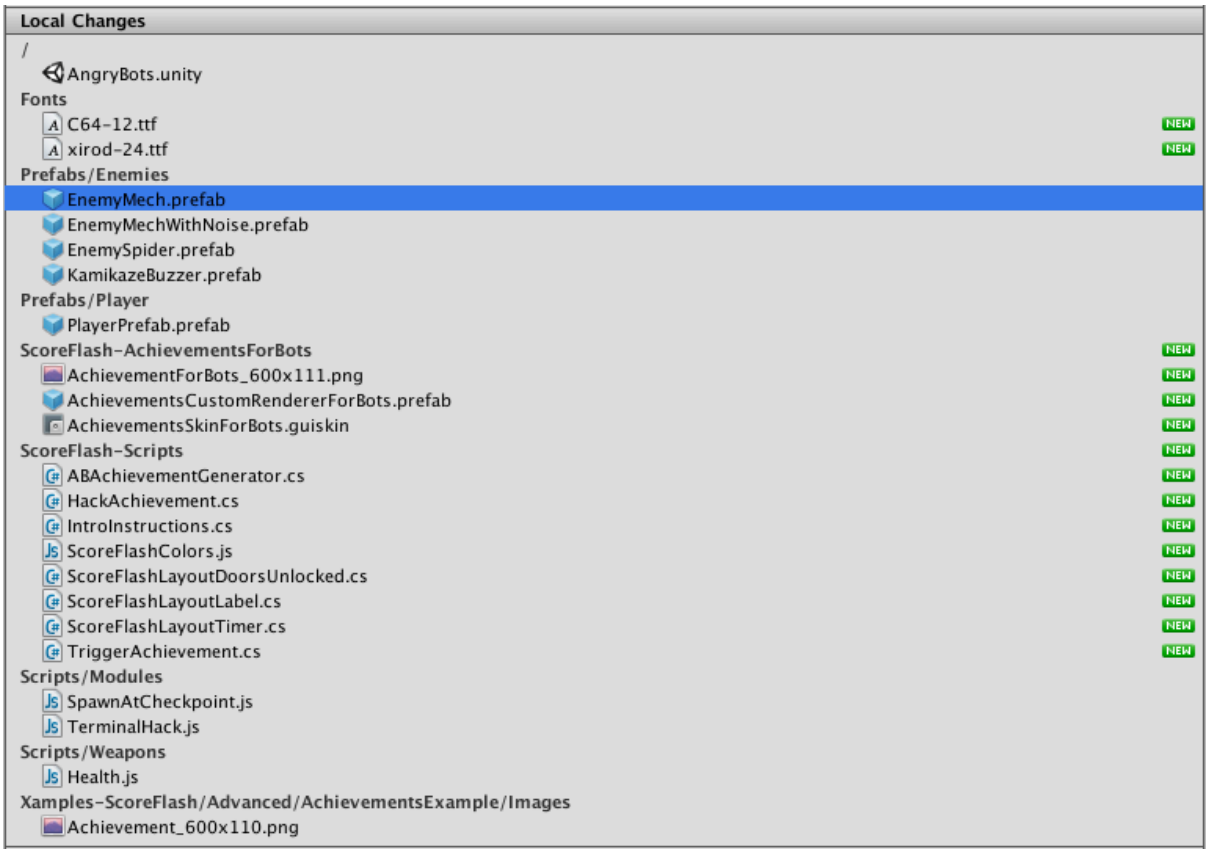
Changes in AngryBots to use ScoreFlash

This document lists the changes that were applied to the original AngryBots demo to have it fully support ScoreFlash. This may look like it's a lot - but it actually didn't take long to apply all of these one after the other, play-testing, tweaking, having fun.

After adding the ScoreFlash package, the following files were changed / added:



So here's the story - the text is rather brief, so you need to be a little familiar with Unity and the AngryBots demo to follow those instructions. The full project is available on the asset store so you can review it.

1. Added ScoreFlash prefab to Scene AngryBots

2. Changed Rendering to "UnityGUI_Font", pick Fonts / xirod as font

3. Added ScoreFlashFollow3D to Player as GameObject below Player, called PlayerScoreFlashFollow3DDamage (create an empty game object, drag it below player, add the ScoreFlashFollow3D component)

4. Optimized position of GameObject PlayerScoreFlashFollow3DDamage using the designer mode (to change the position of the object, **disable** the designer mode, with designer mode activated, you can move the 2D and 3D offsets around)

5. Set Leave Behind to 0.7, Lose Momentum to 0.25

6. Activated "Autogenerate Messages" on ScoreFlashFollow3D, deactivate it on ScoreFlash, first playtesting session

7. Assigned target renderer for PlayerScoreFlashFollow3DDamage (using player mesh, main_player_lorez)

8. Assigned camera to PlayerScoreFlashFollow3DDamage

9. Changed “Health” script to allow assigning ScoreFlashFollow3D

10. Assigned PlayerScoreFlashFollow3DDamage to Player (Health)

11. Disabled “Autogenerate Messages” on PlayerScoreFlashFollow3DDamage

12. Applied Prefab of Player to “save changes”

13. Saved scene, just to be sure ;-)

14. Added health messages to script Health.js

In OnDamage() after lastDamageTime:

```
if (scoreFlashFollow3DDamage != null) {
        scoreFlashFollow3DDamage.Push(-amount, Color.red);
}
```

In Regenerate() after health += regenerateSpeed;

```
if (scoreFlashFollow3DDamage != null) {
        scoreFlashFollow3DDamage.Push(regenerateSpeed, Color.green);
}
```

Here’s a diff (after a few more changes were already applied - but this should give you an idea):

First part:

```
  health -= amount;
  damageSignals.SendSignals (this);
  lastDamageTime = Time.time;

  // START Added for ScoreFlash
  if (scoreFlashFollow3DDamage != null) {
          scoreFlashFollow3DDamage.Push(-amount, ScoreFlashColors.instance.damageColor);
  }
  UpdateHealth();
  // END Added for ScoreFlash

  // Enable so the Update function will be called
```

Second part:

```
if (Time.time > lastDamageTime + 3) {
        health += regenerateSpeed;

        // START Added for ScoreFlash
        if (scoreFlashFollow3DDamage != null) {
                scoreFlashFollow3DDamage.Push(regenerateSpeed, ScoreFlashColors.instance.healthyColor);
        }
        UpdateHealth();
        // END Added for ScoreFlash

        yield;

        if (health >= maxHealth) {
                health = maxHealth;
                enabled = false;
        }
}
```

15. This is already looking quite nice - but we can do better: Duplicate PlayerScoreFlashFollow3DDamage

16. Name the new ScoreFlashFollow3D game object PlayerScoreFlashFollow3DHealth

17. Set Leave Behind to 0 (“lock” on object), Lose Momentum to 0.7, activate “Freeze on Read” ... actually, setting Leave Behind to a value above 0 might look better, just don’t give it too much “leaving behind”

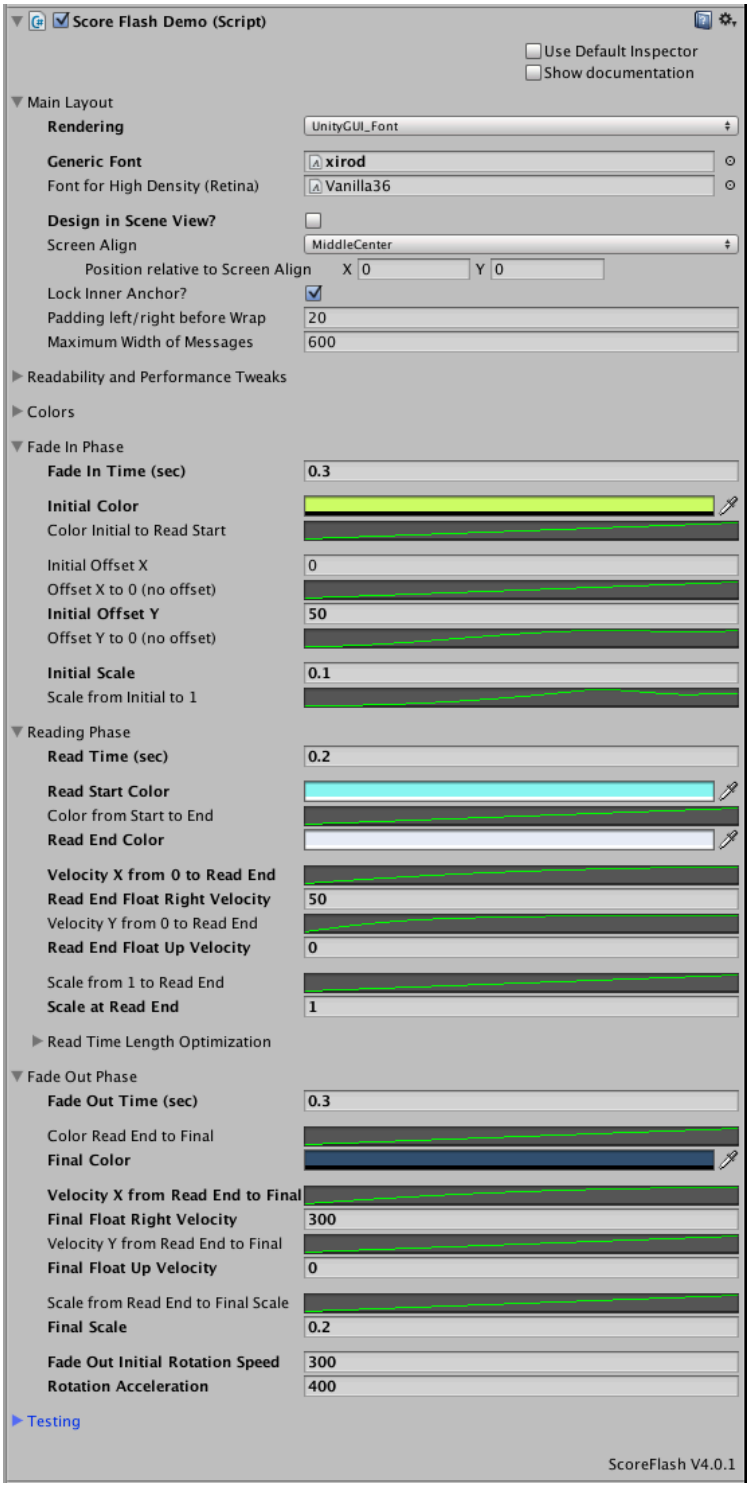
18. Screen Position Offset X=40

19. Set Screen Position Offset of PlayerScoreFlashFollow3DDamage to X=-40

20. Let’s add another ScoreFlash instance to push different kinds of messages, add a ScoreFlashManager

21. Call one instance of ScoreFlash *ScoreFlashDamage*, the other *ScoreFlashHealth*

22. ScoreFlashHealth: Make messages come in from below and move out to right, here’s a screen shot of the inspector of ScoreFlashHealth after all changes were applied:



23. Assign ScoreFlashHealth as Default Score Flash for PlayerScoreFlashFollow3DHealth

24. Assign ScoreFlashDamage as Default Score Flash for PlayerScoreFlashFollow3DDamage

25. Add scoreFlashFollow3DHealth to Health.js (add variable, and assign in editor)

26. Add this code to Health.js

```
function UpdateHealth() {
        if (scoreFlashFollow3DHealth != null) {
                var healthColor : Color = Color.green;
                if (health / maxHealth < 0.5) {
                        healthColor = Color.yellow;
                }
                if (health / maxHealth < 0.2) {
                        healthColor = Color.red;
                }
                scoreFlashFollow3DHealth.Push(health, healthColor);
        }
}
```

… add “UpdateHealth()” after both calls, outside the null-check (see the diff-screen shot above - there, those calls had already been included)

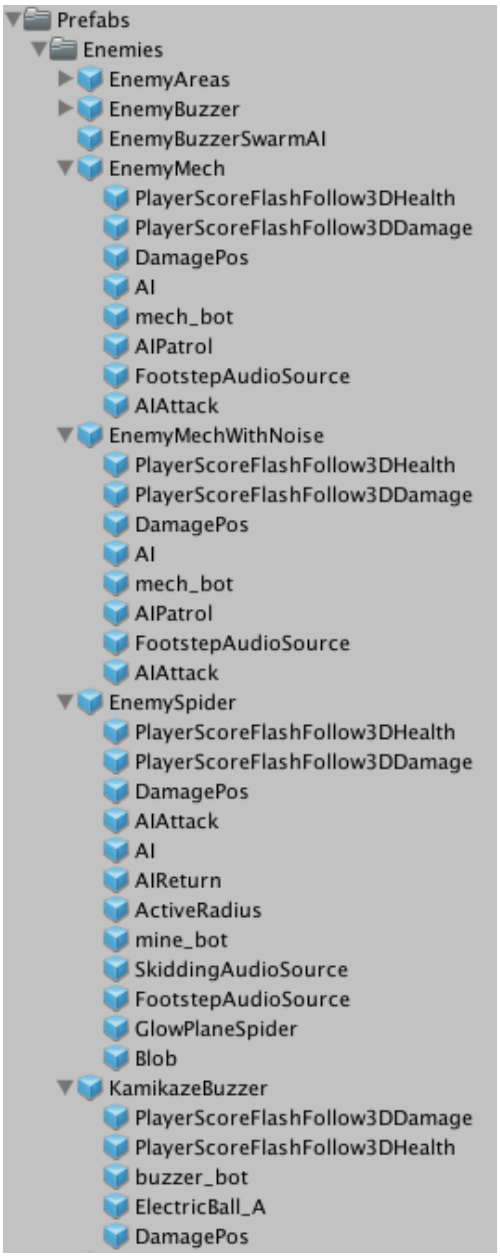
27. Increase “Regenerate Speed” to 2, change “yield WaitForSeconds(1.0f)” to 3f (this will look better … and let the player live longer ;-))

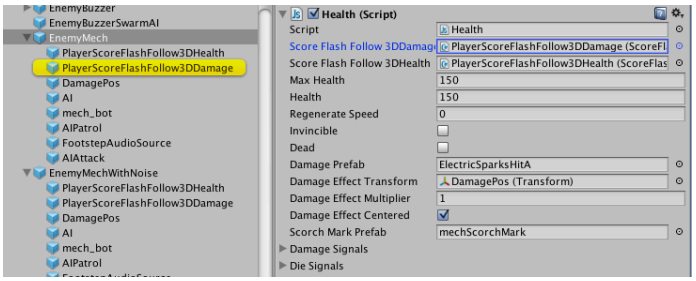
28. Make ScoreFlashDamage much faster, let it slip a little left and make this look like action ;-) ... this needs another screenshot ... look at the next page:

ScoreFlashDamage - after all changes were applied:

29. Add UpdateHealth() to Awake method of Health.js

30. Now add the same setup to the various enemy prefabs (EnemySpider, EnemyMech, KamikazeBuzzer, EnemyMechWithNoise): add the two ScoreFlashFollow3D instances (I simply made copies of those I already had), position them correctly, assign them to the Health component, apply the prefab) ... here's what those prefabs looked like after that operation:





31. As the ScoreFlash instances are in the scene, they won't be distributed by applying the prefab, so you have to do this manually - but that's surprisingly easy: search in

scene for “Health”, then apply ScoreFlashHealth as Default Score Flash to all “PlayerScoreFlashFollow3DHealth” by multi-selecting all PlayerScoreFlashFollow3DHealth objects in the scene and then dragging ScoreFlashHealth on the slot DefaultScoreFlash; do the same for “damage”

32. Notice that when enemies are gone, their info remains, so add in OnDamage, inside if (health <= 0), first we set freezeOnRead to false, then we push an “Aargh” message; there’s other ways to do this, but this one was most fun here:

```
// START Added for ScoreFlash
if (scoreFlashFollow3DHealth != null) {
        scoreFlashFollow3DHealth.freezeOnRead = false;
        scoreFlashFollow3DHealth.Push("Aaargh!!!", Color.red);
}
if (scoreFlashFollow3DDamage != null) {
        scoreFlashFollow3DDamage.freezeOnRead = false;
        scoreFlashFollow3DDamage.Push("Aaargh!!!", Color.red);
}
// END Added for ScoreFlash
```

33. KamikazeBuzzers need more health - up from 10 to 30 (in the prefab, of course ;-))

34. Play the game … this is already quite awesome :-)

35. Add a new ScoreFlash instance from prefab, call it “ScoreFlashInstructions”, use font C64 (this is fun ;-)

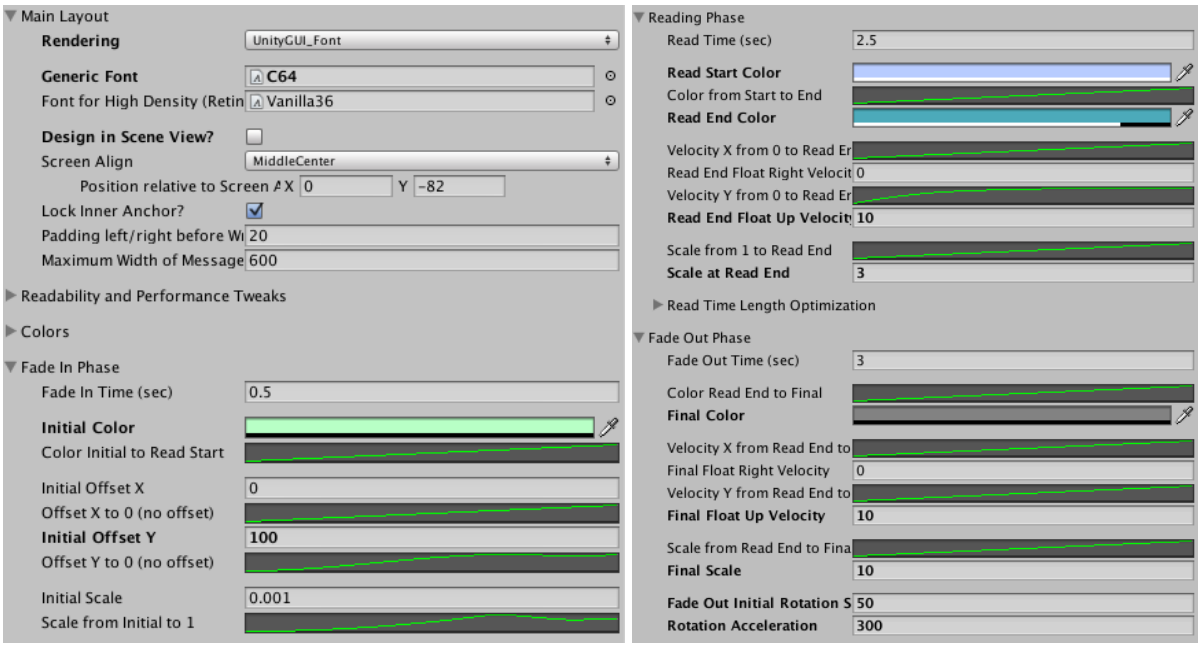
36. Create script IntroInstructions (see ScriptsForScoreFlash - whole script is in there), oh well, here it is for those who don’t have the project available:

```
using UnityEngine;
using System.Collections;

using NarayanaGames.ScoreFlashComponent;

public class IntroInstructions : MonoBehaviour {

        // Use this for initialization
        public void Start() {
                StartCoroutine(IntroMessages());
        }

        public IEnumerator IntroMessages() {
                IScoreFlash myScoreFlash =
                  ScoreFlashManager.Get("ScoreFlashInstructions");
                yield return new WaitForSeconds(1);
                myScoreFlash.PushLocal("ScoreFlash Bots");
                yield return new WaitForSeconds(1.5F);
                myScoreFlash.PushLocal("based on Unity’s demo Angry Bots");
                yield return new WaitForSeconds(3F);
                myScoreFlash.PushLocal("Scrolling Combat Text and Achievements");
                yield return new WaitForSeconds(2);
                myScoreFlash.PushLocal("powered by …");
                yield return new WaitForSeconds(1);
                myScoreFlash.PushLocal("Score Flash");
                yield return new WaitForSeconds(2.5F);
                myScoreFlash.PushLocal("(see buy button ;-) )");

                yield return new WaitForSeconds(7);
```

```
                myScoreFlash.PushLocal("Now … stay alive");
                yield return new WaitForSeconds(3);
                myScoreFlash.PushLocal("Find the terminals to hack the doors");
                yield return new WaitForSeconds(3);
                myScoreFlash.PushLocal("check the floor for a hint");
        }

}
```

37. Put that script on a game object IntroInstructions in scene

38. Make ScoreFlashInstructions nicer ;-) here's another ScoreFlashScreenShot:



39. Make a copy of AchievementGenerator (in the examples), call it ABAchievementGenerator

40. Replace the lengthy code (Start() and in particular GenerateAchievementsCo) with a simple:

```
public ScoreFlash myScoreFlash = null;

public void PushAchievement(string title, string description) {
     myScoreFlash.PushLocal(GenerateAchievement(title,
description));
}
```

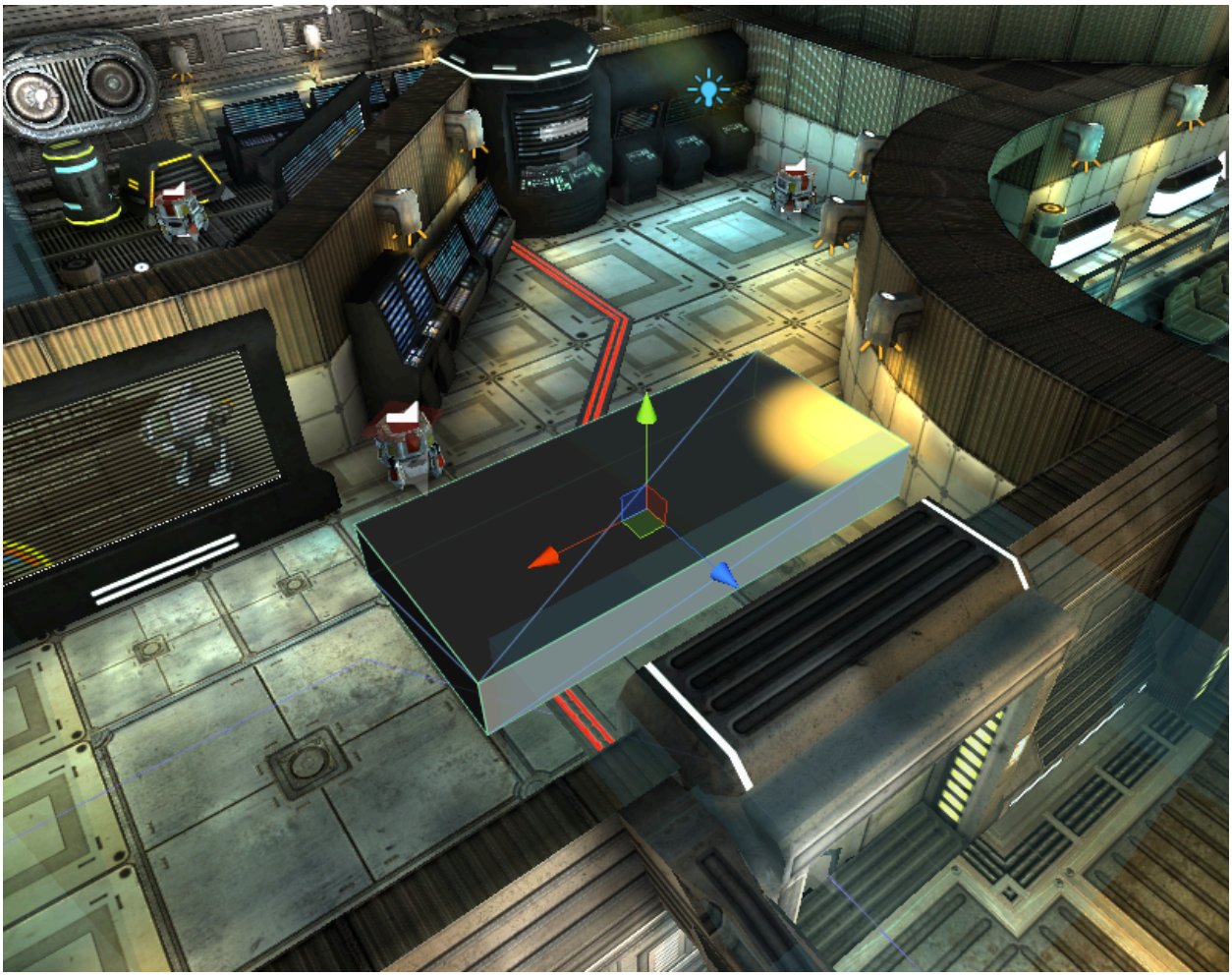
Well, okay, here's the whole thing for you once again (but using a very small font, hope you can read it anyways):

```
using UnityEngine;
using System.Collections;

using NarayanaGames.ScoreFlashComponent;

public class ABAchievementGenerator : MonoBehaviour {

    /// <summary>
    ///     This is the custom renderer that renders our awesome achievements.
    ///     It uses a custom implementation of ScoreFlashRendererBase. In most
    ///     cases, you'll want to create your own implementations of this,
    ///     specific to your individual needs. But you can use this and, if
    ///     you're using NGUI, the NGUI example as a starting point.
    /// </summary>
    public AchievementsCustomRenderer achievementPrefab;

          public ScoreFlash myScoreFlash = null;

          public void PushAchievement(string title, string description) {
                  myScoreFlash.PushLocal(GenerateAchievement(title, description));
          }

    private AchievementsCustomRenderer GenerateAchievement(string title, string description) {
        AchievementsCustomRenderer achievement = (AchievementsCustomRenderer) achievementPrefab.CreateInstance(this.transform);
        achievement.AchievementUnlocked(title, description);
        return achievement;
    }
}
```

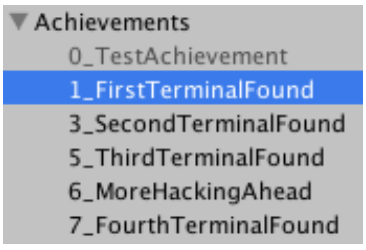
41. Create a new class TriggerAchievement

```
using UnityEngine;
using System.Collections;

public class TriggerAchievement : MonoBehaviour {

      public ABAchievementGenerator generator;

      public string title = "Achievement Title";
      public string description = "Some description, use the editor";

      private bool unlocked = false;

      public void OnTriggerEnter(Collider other) {
            if (other.gameObject.tag == "Player") {
                  if (!unlocked) {
                        generator.PushAchievement(title, description);
                        unlocked = true;
                  }
            }
      }
}
```

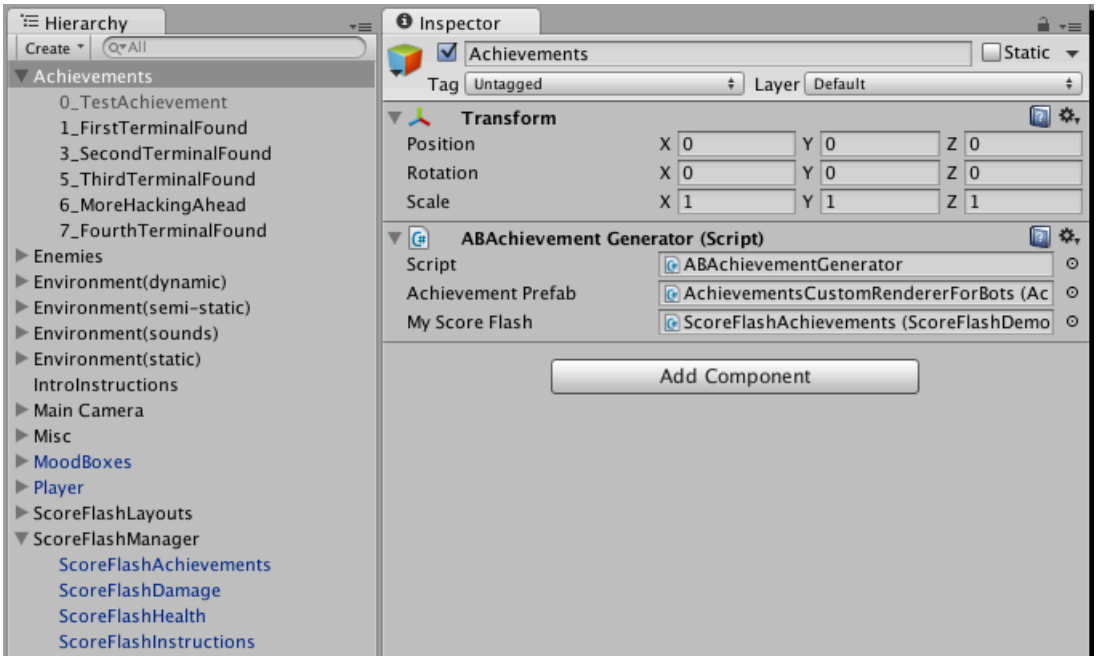
42. Create a box near the first terminal make its collider a trigger, attach script TriggerAchievement, disable mesh renderer. This is where it is in the scene, with the mesh renderer still activated (that makes the screenshot more descriptive):



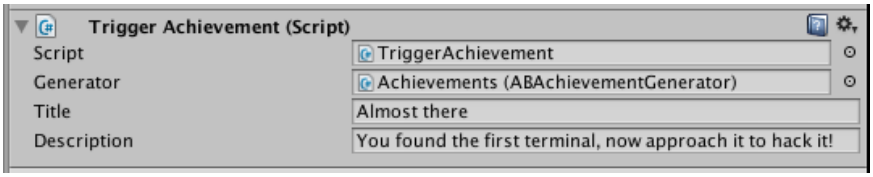
43. Create an Achievements game object, put the box below. Here's a screenshot with all those triggers already added:



44. Add ABAchievementsGenerator to Achievements, assign the prefab and the correct score flash instance (we need another one for this)



45. Type a useful title and description … we have our first achievement - YEAH!!!



46. Add the HackAchievement class (which could be the same as TriggerAchievement but we make it a little different)

```
using UnityEngine;
using System.Collections;

public class HackAchievement : MonoBehaviour {

	public ABAchievementGenerator generator;

	public string title = "Achievement Title";
	public string description = "Achievement Description that could be quite a bit longer";

	private bool unlocked = false;

	public void AchievementUnlocked() {
		if (!unlocked) {
			generator.PushAchievement(title, description);
			unlocked = true;
		}
	}
}
```

47. Add HackAchievement to TerminalForDoorFromHell_1, wire everything up and type title / description

48. Make class TerminalHack more awesome:

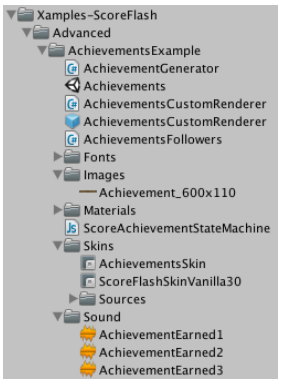
```
function OnHackingCompleted () {
	audio.Play ();
	animationComp.Stop ();
	enabled = false;

	// START Added for ScoreFlash
	if (GetComponent.<HackAchievement>() != null) {
		GetComponent.<HackAchievement>().AchievementUnlocked();
	}
	// END Added for ScoreFlash
}
```

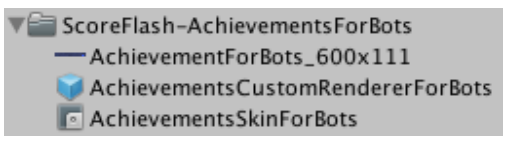
49. Get really angry that some weirdo though UnityScript was a good idea - IT WAS NOT!!! Do it like this instead (in other words - forget the previous suggestion from 48, this simply won't work, but this does):

```
function OnHackingCompleted () {
	audio.Play ();
	animationComp.Stop ();
	enabled = false;

	// START Added for ScoreFlash
	SendMessage("AchievementUnlocked");
	// END Added for ScoreFlash
}
```

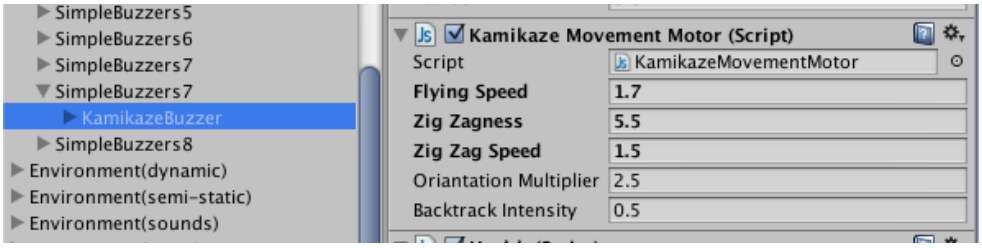
50. Add the other achievements for the other doors, one needs two hacks, make this a special one ;-)

51. Adjust the background graphics and fonts of the achievements to fit into the scene (I'm not an artist ;-)), I've created ScoreFlash-AchievementsForBots for that purpose (also using the fonts from the main game), and I used the stuff from the Achievements examples that comes with ScoreFlash to get started:





52. Make “SimpleBuzzers7 / KamikazeBuzzer”, the one that first hits the player much slower to give a little more time for the first messages to cycle through; try to make the first “hit” happens more or less with the text “now stay alive” ;-) also improve the timing of the health ScoreFlash of the player appearing: this now happens right after “ScoreFlash” appears in the instructions text :-)



... do you notice the typo ;-) ... oh well, I didn’t file a bug report with UT ;-)

53. Add two ScoreFlashLayouts for some additional info, make a copy of Xample_TestLayout for the label, call it ScoreFlashLayoutLabel, use ScoreFlashHealth for these (as Default Score Flash on the ScoreFlashLayout), add new class ScoreFlashLayoutTimer; position them like label and value for the label on top left of screen, activate “Freeze on Read” for both, assign the scripts - enjoy the timer ;-)

I’ve also added some cool magic for the locked doors ... so ... stay with me, here’s a few classes:

```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(ScoreFlashLayout))]
public class ScoreFlashLayoutLabel : MonoBehaviour {

	public string label = "Label:";

	private ScoreFlashLayout layout = null;

	void Awake() {
		layout = GetComponent<ScoreFlashLayout>();
		layout.freezeOnRead = true;
	}

	void Start() {
		StartCoroutine(ShowLabelWithDelay());
	}

	public IEnumerator ShowLabelWithDelay() {
		// we add these delays to make sure everything is
		// first set up properly; without it, sometimes
		// the layouts would get a little offset
		yield return new WaitForSeconds(0.5F);
		layout.Push(label);
	}
}
```

```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(ScoreFlashLayout))]
public class ScoreFlashLayoutTimer : MonoBehaviour {

	public float secondsToUpdate = 5F;

	private ScoreFlashLayout layout = null;

	void Awake() {
		layout = GetComponent<ScoreFlashLayout>();
		layout.freezeOnRead = true;
	}

	void Start() {
		StartCoroutine(TimerCounter());
	}

	public IEnumerator TimerCounter() {
		yield return new WaitForSeconds(0.5F);
		while (true) {
			int minutes = ((int)Time.realtimeSinceStartup) / 60;
			int seconds = ((int)Time.realtimeSinceStartup) - minutes * 60;
			layout.Push(string.Format("{0}:{1:00}", minutes, seconds));
			yield return new WaitForSeconds(secondsToUpdate);
		}
	}

}
```

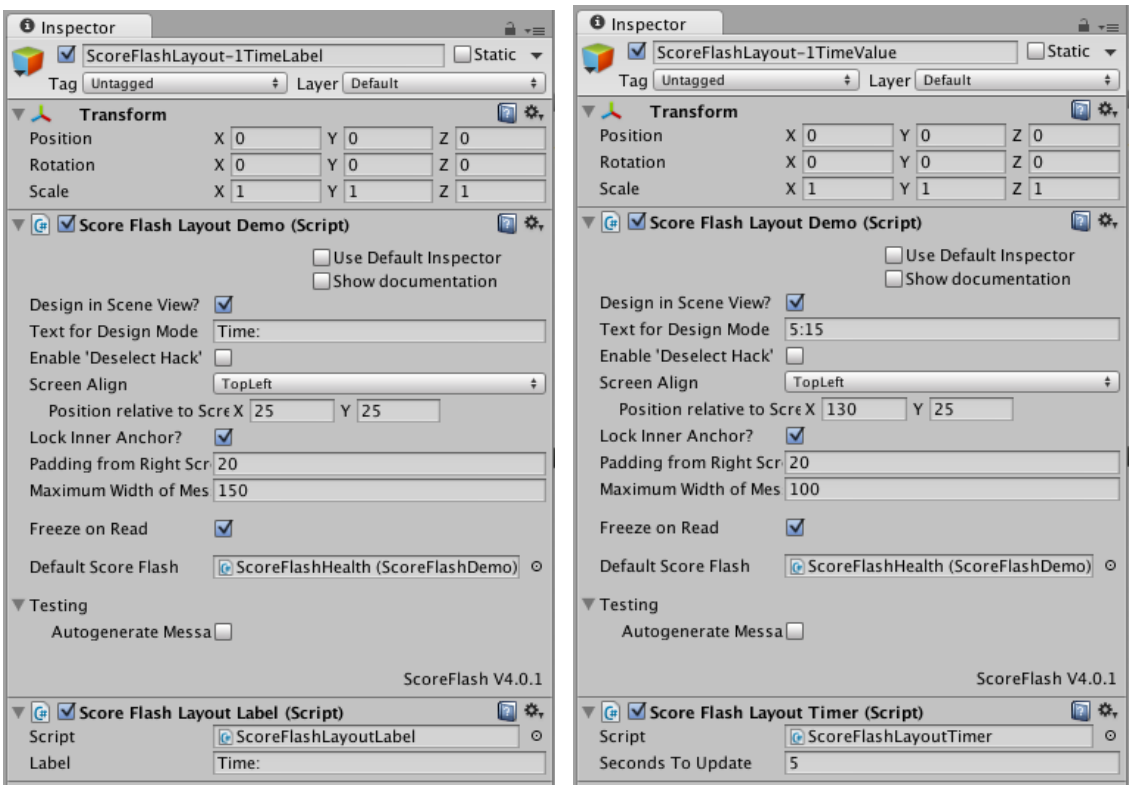
```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(ScoreFlashLayout))]
public class ScoreFlashLayoutDoorsUnlocked : MonoBehaviour {

	private int doorsCount = 0;
	private int doorsUnlockedCount = 0;

	private ScoreFlashLayout layout = null;

	void Awake() {
		layout = GetComponent<ScoreFlashLayout>();
		layout.freezeOnRead = true;
	}

	void Start() {
		UpdateCount ();
	}

	public void RegisterLockedDoor() {
		doorsCount++;
	}

	public void RegisterDoorUnlocked() {
		doorsUnlockedCount++;
		UpdateCount();
	}

	public void UpdateCount() {
		StartCoroutine(UpdateCountCo());
	}

	private IEnumerator UpdateCountCo() {
		yield return new WaitForSeconds(0.5F);
		layout.Push(string.Format("{0}/{1}", doorsUnlockedCount, doorsCount));
	}
}
```

This one we've already added - but we add some stuff (bold)

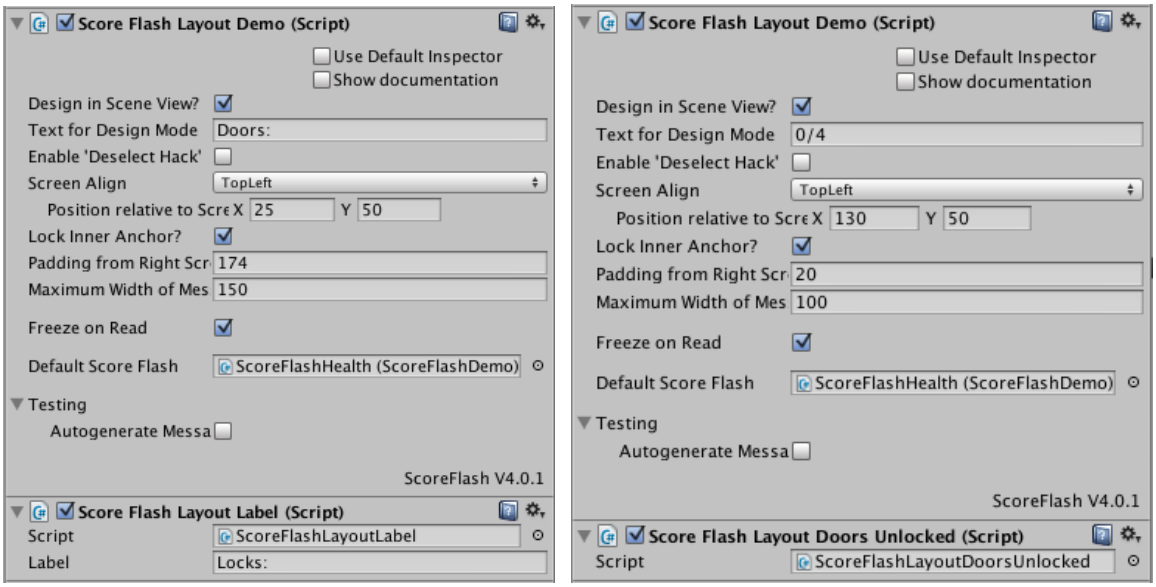
```
using UnityEngine;
using System.Collections;

public class HackAchievement : MonoBehaviour {

	public ABAchievementGenerator generator;

	public string title = "Achievement Title";
	public string description = "Achievement Description that could be quite a bit longer";

	private bool unlocked = false;

	private ScoreFlashLayoutDoorsUnlocked doorsCounter = null;

	public void Awake() {
		doorsCounter = FindObjectOfType<ScoreFlashLayoutDoorsUnlocked>();
		if (doorsCounter != null) {
			doorsCounter.RegisterLockedDoor();
		}
	}

	public void AchievementUnlocked() {
		if (!unlocked) {
			generator.PushAchievement(title, description);
			unlocked = true;
			if (doorsCounter != null) {
				doorsCounter.RegisterDoorUnlocked();
			}
		}
	}
}
```

Here's what it should look like with the designers active:

ScoreFlashLayout-1TimeLabel (left) and ScoreFlashLayout-1TimeValue (right)



ScoreFlashLayout-2DoorsUnlockedLabel (left) and ScoreFlashLayout-2DoorsUnlockedValue (right)



54. Instead of using Green, Red and Yellow, use colors that match the style of the game; we create a “Singleton-Skin” with those colors so we can easily look them up and call ScoreFlashColors, Health.js needs to be adjusted accordingly (as Health.js is

JavaScript, bad news, ScoreFlashColors needs to be JavaScript as well), we put this on ScoreFlashManager for simplicity's sake

ScoreFlashColors.js:

```
#pragma strict

public static var instance : ScoreFlashColors = null;

public var healthyColor : Color = Color.green;
public var damageColor : Color  = Color.red;
public var warnColor : Color  = Color.yellow;

function Awake() {
	instance = this;
}
```

So finally, here's what the full Health.js looks after all these changes (most of it is still the original file that comes with the Unity demo AngryBots):

```
#pragma strict

// START Added for ScoreFlash
public var scoreFlashFollow3DDamage : ScoreFlashFollow3D = null;
public var scoreFlashFollow3DHealth : ScoreFlashFollow3D = null;
// END Added for ScoreFlash

public var maxHealth : float = 100.0;
public var health : float = 100.0;
public var regenerateSpeed : float = 0.0;
public var invincible : boolean = false;
public var dead : boolean = false;

public var damagePrefab : GameObject;
public var damageEffectTransform : Transform;
public var damageEffectMultiplier : float = 1.0;
public var damageEffectCentered : boolean = true;

public var scorchMarkPrefab : GameObject = null;
private var scorchMark : GameObject = null;

public var damageSignals : SignalSender;
public var dieSignals : SignalSender;

private var lastDamageTime : float = 0;
private var damageEffect : ParticleEmitter;
private var damageEffectCenterYOffset : float;

private var colliderRadiusHeuristic : float = 1.0;
// START Added for ScoreFlash
private var initialWait : boolean = false;
// END Added for ScoreFlash


function Awake () {
	// START Added for ScoreFlash
	if (gameObject.tag != "Player") {
		enabled = false;
	}
	// END Added for ScoreFlash
	if (damagePrefab) {
		if (damageEffectTransform == null)
			damageEffectTransform = transform;
		var effect : GameObject = Spawner.Spawn (damagePrefab, Vector3.zero,
                                                 Quaternion.identity);
		effect.transform.parent = damageEffectTransform;
		effect.transform.localPosition = Vector3.zero;
```

```
                damageEffect = effect.particleEmitter;
                var tempSize : Vector2 =
                      Vector2(collider.bounds.extents.x,collider.bounds.extents.z);
                colliderRadiusHeuristic = tempSize.magnitude * 0.5;
                damageEffectCenterYOffset = collider.bounds.extents.y;

        }
        if (scorchMarkPrefab) {
                scorchMark = GameObject.Instantiate(scorchMarkPrefab, Vector3.zero,
                                                    Quaternion.identity);
                scorchMark.SetActive (false);
        }

        // START Added for ScoreFlash
        if (gameObject.tag == "Player") {
                initialWait = true;
                enabled = true;
                StartCoroutine(InitScoreFlash());
        }
        // END Added for ScoreFlash
}

// START Added for ScoreFlash
function InitScoreFlash() {
        yield WaitForSeconds(9.5F);
        UpdateHealth();
        initialWait = false;
        enabled = false;
}
// END Added for ScoreFlash

// START Added for ScoreFlash
function Reset() {
        dead = false;
        health = maxHealth;
        if (scoreFlashFollow3DHealth != null) {
                scoreFlashFollow3DHealth.freezeOnRead = true;
        }
        UpdateHealth();
}
// END Added for ScoreFlash

function OnDamage (amount : float, fromDirection : Vector3) {
        // Take no damage if invincible, dead, or if the damage is zero
        if(invincible)
                return;
        if (dead)
                return;
        if (amount <= 0)
                return;

        // Decrease health by damage and send damage signals

        // @HACK: this hack will be removed for the final game
        //  but makes playing and showing certain areas in the
        //  game a lot easier
        /*
        #if !UNITY_IPHONE && !UNITY_ANDROID && !UNITY_WP8
        if(gameObject.tag != "Player")
                amount *= 10.0;
        #endif
        */

        health -= amount;
        damageSignals.SendSignals (this);
        lastDamageTime = Time.time;

        // START Added for ScoreFlash
        if (scoreFlashFollow3DDamage != null) {
                scoreFlashFollow3DDamage.Push(-amount, ScoreFlashColors.instance.damageColor);
        }
        UpdateHealth();
```

```
        // END Added for ScoreFlash

        // Enable so the Update function will be called
        // if regeneration is enabled
        if (regenerateSpeed > 0)
                enabled = true;

        // Show damage effect if there is one
        if (damageEffect) {
                damageEffect.transform.rotation = Quaternion.LookRotation (fromDirection,
                        Vector3.up);
                if(!damageEffectCentered) {
                        var dir : Vector3 = fromDirection;
                        dir.y = 0.0;
                        damageEffect.transform.position = (transform.position + Vector3.up *
                            damageEffectCenterYOffset) + colliderRadiusHeuristic * dir;
                }
                // @NOTE: due to popular demand (ethan, storm) we decided
                // to make the amount damage independent ...
                //var particleAmount = Random.Range (damageEffect.minEmission,
                // damageEffect.maxEmission + 1);
                //particleAmount = particleAmount * amount * damageEffectMultiplier;
                damageEffect.Emit();// (particleAmount);
        }

        // Die if no health left
        if (health <= 0)
        {
                GameScore.RegisterDeath (gameObject);

                health = 0;
                dead = true;
                dieSignals.SendSignals (this);
                enabled = false;

                // START Added for ScoreFlash
                if (scoreFlashFollow3DHealth != null) {
                        scoreFlashFollow3DHealth.freezeOnRead = false;
                        // this only appears for a little moment,
                        // so we keep it plain read instead of using damageColor as usual
                        scoreFlashFollow3DHealth.Push("Aaargh!!!", Color.red);
                }
                if (scoreFlashFollow3DDamage != null) {
                        scoreFlashFollow3DDamage.freezeOnRead = false;
                        scoreFlashFollow3DDamage.Push("Aaargh!!!", Color.red);
                }
                // END Added for ScoreFlash


                // scorch marks
                if (scorchMark) {
                        scorchMark.SetActive (true);
                        // @NOTE: maybe we can justify a raycast here so we can place the mark
                        // on slopes with proper normal alignments
                        // @TODO: spawn a yield Sub() to handle placement, as we can
                        // spread calculations over several frames => cheap in total
                        var scorchPosition : Vector3 = collider.ClosestPointOnBounds
                               (transform.position - Vector3.up * 100);
                        scorchMark.transform.position = scorchPosition + Vector3.up * 0.1;
                        scorchMark.transform.eulerAngles.y = Random.Range (0.0, 90.0);
                }
        }
}

function OnEnable () {
        // START Added for ScoreFlash
        if (!initialWait) {
                Regenerate();
        }
        // END Added for ScoreFlash
}
```

```
// Regenerate health

function Regenerate () {
	if (regenerateSpeed > 0.0f) {
		while (enabled) {
			if (Time.time > lastDamageTime + 3) {
				health += regenerateSpeed;

				// START Added for ScoreFlash
				if (scoreFlashFollow3DDamage != null) {
					scoreFlashFollow3DDamage.Push(regenerateSpeed,
					   ScoreFlashColors.instance.healthyColor);
				}
				UpdateHealth();
				// END Added for ScoreFlash

				yield;

				if (health >= maxHealth) {
					health = maxHealth;
					enabled = false;
				}
			}
			yield WaitForSeconds (3.0f);
		}
	}
}

// START Added for ScoreFlash
function UpdateHealth() {
	if (scoreFlashFollow3DHealth != null) {
		var healthColor : Color = ScoreFlashColors.instance.healthyColor;
		if (health / maxHealth < 0.6) {
			healthColor = ScoreFlashColors.instance.warnColor;
		}
		if (health / maxHealth < 0.2) {
			// again, this is only for a short time, red is fine for this purpose
			healthColor = Color.red;
		}
		scoreFlashFollow3DHealth.Push(health, healthColor);
	}
}
// END Added for ScoreFlash
```

55. If you're happy with the results and haven't done so, yet, buy the full version of ScoreFlash and make a build (this document comes as part of a project that has all these changes applied but uses the free demo version of score flash) :-)

You can buy ScoreFlash here:

https://www.assetstore.unity3d.com/en/#!/content/4476

Check out the product page here:

http://scoreflash.narayana-games.net