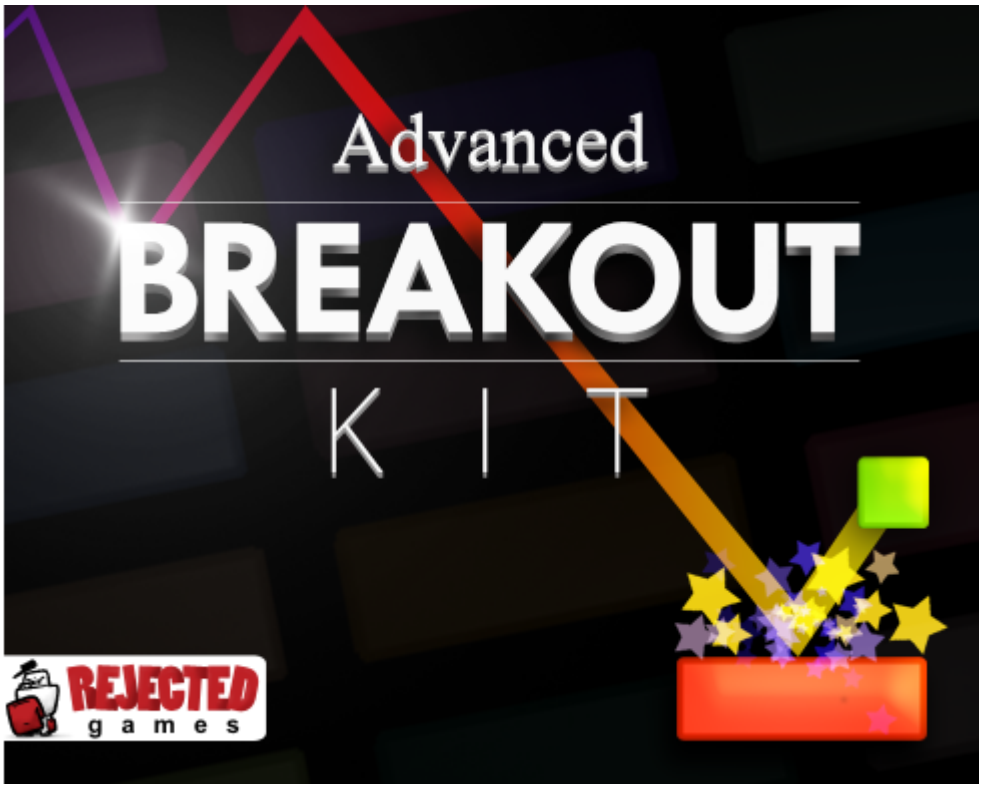

v1.0

Welcome

Hi there, thank you for purchasing Advanced Breakout Kit!
I've used this game in several Unity workshops to show students how Unity works. The workshop was based on a talk called "Juice-it-or-loose-it". There are several special effects in there to make it feel more exciting. The game setup itself is real easy. The effects, if you take them apart, are too. If you need help, do not hesitate to contact me.

If you like this package, please give it a rating in the AssetStore. If not, let me know what I need to improve.

Contents

This document mainly describes the basic setup of the scene but at the bottom you can find something I call “Scripting index”. If you are looking for an example of a specific Unity api usage, like coroutine, animation or raycast this table will show in which script you should look.
The scripts are kept small and focussed on one task. This makes them easy to reuse and understand. There are loads of comments placed in the scripts to help you understand what's going on.

Contact info

For bugs, feature requests and a friendly chat, you can send an email to: support@rejected-games.com

Basic Scene Setup

Ball

Like most of the gameobjects, the ball is pretty simple. Its core is based on a rigidbody with a sphere collider. The rigidbody gives us the (physics) reaction which makes it bouncing around.
There are a few thing to note:
Even though our ball is looking like a cube, it uses a sphere-collider to give an unpredictable bounce when a corner is hit.
The sphere-collider uses a physics material, the properties are set to fully bounce so its not losing energy when colliding.
The rigidbody has a “Freese position” constraint on the y axis as it a 2D game.
The Ball script makes sure the velocity (/speed) is limited, the limits are adjustable of course.
It also waits at the beginning of the game on user input before starting the game. If you do not want that, just call Launch() on Start().
Tip: For fun, duplicate the ball gameobject a few times, make sure they don’t touch each other and start the game. Madness! :)

Blocks

The blocks are real easy: its just a mesh with a box collider and a block script. On default they play an animation on hit with an option to make the block fall after the animation is finished.
There are a few components on the blocks to make them more interesting like RandomColor, PlaySoundOnHit and the ScreenShake and ScreenFlash Notifiers. Some blocks have a DropPowerUpOnHit script which tells the block which power up to drop.
For how they work, please check out the scripts themself as they are fully commented. It should be easy to understand.

Power ups

Each powerup should do the same thing: fall down until they are picked up. Thats why there is a base class named PowerUpBase which does exactly that.
What they do at pick up is implemented in the derived classes: ChangeSize and ExtraBall. This makes it real easy to add more power ups with just a few lines of code.

Walls

The walls are like a block only without most of the interesting components. Its just a mesh with a box collider, nothing special really. You might wonder why the walls are so big. Thats done so they cover the entire screen, whether you are on an iPad (4:3) or on an iPhone5 (16:9).

Paddle

The paddle is interesting but not difficult so hang on. I've used a mesh collider to take advantage of the rounded cube corners when it bounces on the one of the corners. There is a Paddle component which does a few things: when the used taps or clicks a ray is shoot from that point. If the ray hits the collider the user is allowed to move the paddle, but only in the x axis. It also restricts the paddle's x position so it cannot be dragged out of the screen.

Camera

The camera is set to orthographic to give the game that 2D look. Orthographic means there is no perspective (depth) at all. You might have heard of an isometric camera, well thats an orthographic camera (usually) angled at 45, 45 degrees.

GameManager

The gamemanager controls the current state of the game. Things like start game, level completed and level failed are controlled here. For now the graphics are very basic, just a line of text displaying the player some info.

Scripting Index

Tip: use ctrl+f or cmd+f to search for a keyword

rigidbody.velocity , Mathf.Clamp , Input.GetMouseButtonDown , Random.value	Ball.cs
RequireComponent , [HideInInspector] , IEnumerator , WaitForSeconds , animation[""] , Time.deltaTime , OnBecameInvisible()	Block.cs
[RequireComponent(typeof(BoxCollider))] , OnCollisionEnter(Collision c) , GameObject.Instantiate()	DropPowerUpOnHit.cs
RequireComponent , OnCollisionEnter(Collision c)	PlayParticlesOnHit.cs
enum , FindObjectsOfType(typeof)) , Input.GetMouseButtonDown , FindObjectOfType , StartCoroutine , IEnumerator , yield return new WaitForSeconds() , Application.LoadLevel	GameManager.cs
Input.GetMouseButton , Ray , camera.ScreenPointToRay , Plane , Raycast , Mathf.Abs	Paddle.cs
protected , override , base , FindObjectOfType , typeof	ChangeSize.cs
[RequireComponent(typeof())] , Time.deltaTime , IEnumerator OnTriggerEnter(Collider other) , yield return new WaitForSeconds() , protected virtual	PowerUpBase.cs
renderer , Random.value	RandomColor.cs
Camera.backgroundColor , StopAllCoroutines() , StartCoroutine() , IEnumerator , yield return new WaitForSeconds() , Camera.mainCamera	ScreenFlash.cs
OnCollisionEnter(Collision c) , FindObjectOfType , typeof	ScreenFlashNotifier.cs
Time.deltaTime , transform.position	ScreenShaker.cs
OnCollisionEnter(Collision c) , FindObjectOfType , typeof	ScreenShakerNotifier.cs
RequireComponent , AudioSource , AudioClip	PlayMusic.cs
RequireComponent , AudioSource , AudioClip , OnCollisionEnter(Collision c) , audio.pitch , audio.PlayOneShot , static	PlaySoundOnHit_Pitch.cs

RequireComponent , AudioSource , AudioClip , OnCollisionEnter(Collision c) , audio.PlayOneShot	PlaySoundOnHit.cs