
Object Slicing Game



“-An Object Slicing Example Game”

I would like to start by saying thank you for downloading this Game Example. Please rate it if you get the time... its mean a lot to me. If any of you had previously downloaded the Fruit Slicing Game, that was originally on the asset store for free, then I would like to first say, that I am sorry that it is not available anymore. It too closely mirrored another title, and so therefore was removed. I have had to spend a great deal of time on this title, and for that reason, I opted to go ahead and sell it for .99 cents. I’ve spent weeks on it twice now... This is a smaller and lighter version that contains balls that can be sliced. I have included 2 game modes. One with some obstacles/penalties, and one that is just for fun (no obstacles/penalties). This Doc is fairly long, but it does described any nuances, or setup issues that you may run into. If you have any additional questions you can reach out to me via email... It may take me a little while to respond as I am only one person, but I do my best to get back to everyone as soon as I can.

Best Regards

-Brian  

- 1.) *Setup*
- 2.) *Things to Know*
- 3.) *Scripts*
- 4.) *UI, Menu, and Canvas Scaling*

1.) Setup

Setting Up Gravity, Tags, Scene Build Order, and Layers

- **Set Gravity to -100 m/s² on the Y Axis.** See Figure 1.

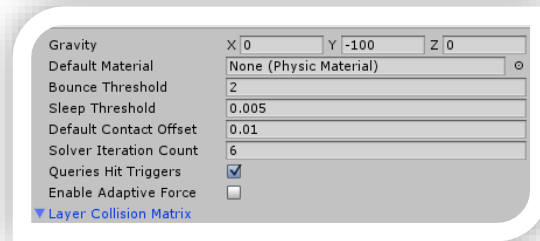


Figure 1- No other changes are necessary in the Physics Manager besides the -100..

- **There are Several Tags used.** See Figure 2.

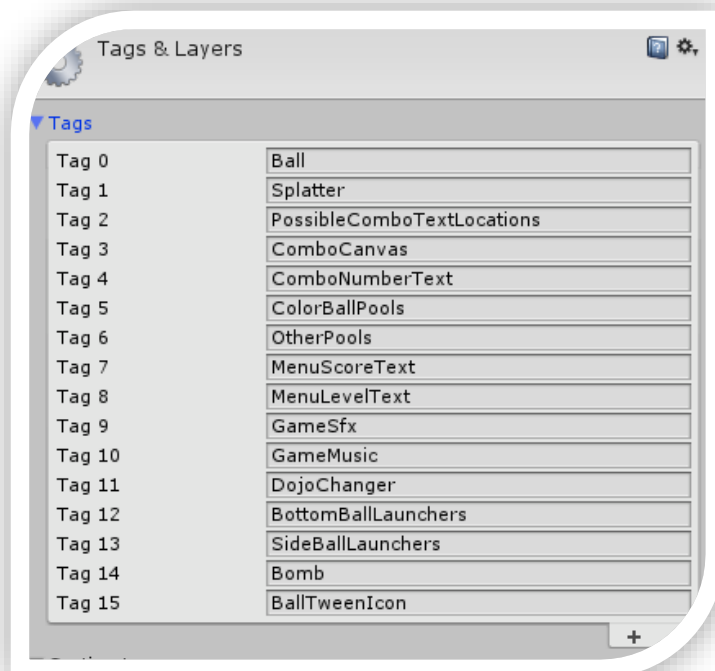


Figure 2 - Several Tags

- **Make sure these Scenes are added to the Build Settings. In this order.** *See Figure 3a.*

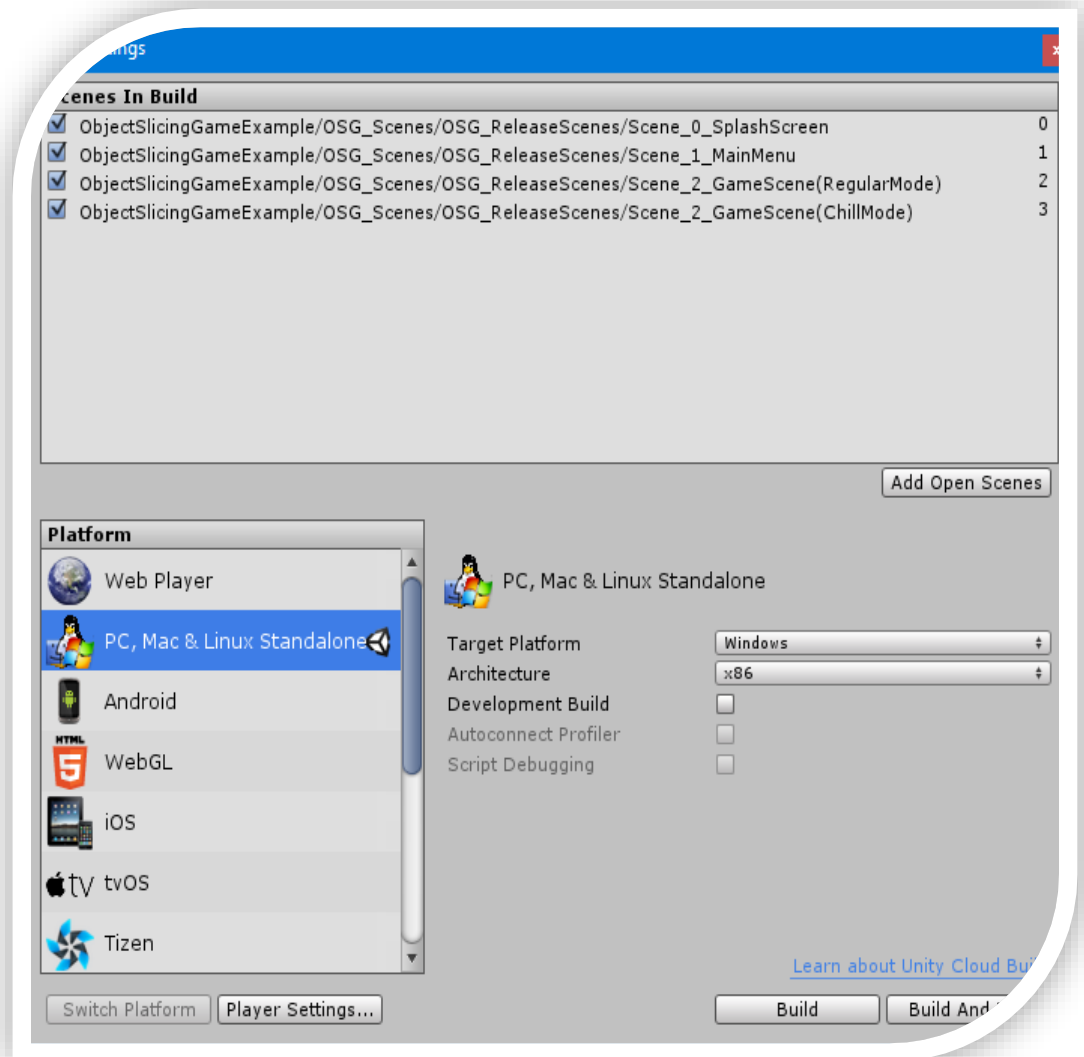


Figure 3a - Splash, Menu, Regular Mode, and Chill Mode

- **New Layers added for OSG.** See *Figure 4a/b*.

Layer Collision Matrix

	Default	TransparentFX	Ignore Raycast	Water	UI	SliceCollider	Ball	BallDebris	DojoBoundary	Bomb
Default	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
TransparentFX	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Ignore Raycast	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Water	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
UI	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
SliceCollider	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Ball	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
BallDebris	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
DojoBoundary	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Bomb	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Figure 4b - Layer Matrix

Layers

Builtin Layer 0	Default
Builtin Layer 1	TransparentFX
Builtin Layer 2	Ignore Raycast
Builtin Layer 3	
Builtin Layer 4	Water
Builtin Layer 5	UI
Builtin Layer 6	
Builtin Layer 7	
User Layer 8	SliceCollider
User Layer 9	Ball
User Layer 10	BallDebris
User Layer 11	DojoBoundary
User Layer 12	Bomb
User Layer 13	
User Layer 14	
User Layer 15	
User Layer 16	
User Layer 17	
User Layer 18	
User Layer 19	
User Layer 20	
User Layer 21	
User Layer 22	
User Layer 23	
User Layer 24	
User Layer 25	
User Layer 26	
User Layer 27	
User Layer 28	
User Layer 29	
User Layer 30	
User Layer 31	

Figure 4a - Listed Layers

2.) Things to Know

The meat, potatoes, and nuances...

This section we will cover “Things to Know”. I will give a brief rundown of some of the main scripts. This section will be long... my apologies ;).

The General Stuff...

If you are attempting to play in the editor make sure the game is started at Scene 0 (the “Splash Scene”). When the project is built that point becomes kind of a “Non-Issue”. As long as the “Build Settings” have the scenes in the proper order everything should work properly.

While the game “should” be started in scene 0, there should not be any errors when started elsewhere. The cameras in the scenes all have a FaderCaller.cs attached to them, so testing from a specific scene should be error free. The “FaderCaller” class instantiates the fader canvas (if one does not exist). If you run into any issues or exceptions try starting the game from scene 0 and see if the problem(s) persist.

You may notice that the gravity is set to an extremely high value. Initially I was having some performance issues. The “Cutting” part of the project has been refactored 3 times. There was a version with actual mesh deformation (turned out too heavy for my mobile), a physics driven version (still... ended up being about 45 rigidbodies on screen at one time), and then I finally decided to make a very simple “cut” method, and use the animation system to fulfill the “falling gibs” portion. So now, we simple make “Slice”, record the angle, and whatever angle it is close to, we instantiate “halves” that look appropriate, and the animation is played of the “halves” falling. Early in the project I noticed that cutting those tiny balls with a raycast or collider didn’t work all that well at high speeds. Even with some of the interpolation settings turned on. The distance in between frames did not always catch a ball. My answer was to dial up the scale

quite considerably. I made sure that other things were somewhat proportionate to those changes... I.e. camera size, and gravity. Now that the system is in a pretty optimal state (at least from a performance standpoint), the scale could probably be returned.

The current implementation leaves a lot to be desired, but I am hoping to revisit the Mesh Deformation version to see about increasing performance. That will be later down the road though. As it stands, this project is a very basic, and simple way to achieve the effect of “cutting objects”. Since I am trying to make these simple, and to be used as a learning resource, that is a good place to start. As long as the setup bullet points are hit then you are ready to go.

*****NOTE*****

There are DoxyGen Doc pages with a search and navigation pane in the form of a webpage, and of course the scripts themselves are heavily commented. There are probably too many comments... so strip the class/method comments down if you need! My only intention was to be thorough for beginners. Between this Document, Doc Pages, and Comments in the Scripts themselves you shouldn't have too many questions. If you refer to the various scenes and the written material everything should be pretty clear. With CSC(Color Switch Clone) the Free Example Games are basically released “As-Is”... I should have a FAQ on the website soon, and you can e-mail me, but it may take me a while to get back to you. I stay pretty busy with work and stuff like this. If you are patient, and my spam folder did not grab your email, I will get back to you. Example Projects that I sell will get priority when it comes to e-mail support.

(DoxyGen Doc pages grabs “///” comments, but not the line by line “//” comments)... so that means that there is more detail in the actually script files.

The Difference between Menu Balls and Game Scene Balls...

There are basically a few different categories when it comes to the Scripts/Classes in the Object Slicing Game. We load the game via the splash screen, and then it loads the Main Menu. The “Menu Balls” that load the different scenes are almost identical to the “Game Balls”, they just have some values modified in the inspector.

See Figure 5. The “In Game Scene” Boolean value at the top is checked for balls in the game scenes. If you are at the main menu, then the value is unchecked. If it is unchecked the variable below it (“Scene To Load If Not”) needs to be given a value between 0 and the max number of scenes there are. This integer is used like... *SceneManager.LoadScene*

(*sceneToLoadIfNot*); so, we pass that value to the *LoadLevel* function. There are some other small differences at the end of the “CutBall” method, but those are detailed in the script... BUT as an example 1.) We don’t Increment “Balls Missed” if a ball is chopped in the Menu Scene. 2.) We make sure all colliders are immediately deactivated in the Menu Scene so we don’t have double level loads... Things like that. These are the types of differences, and something to keep in mind when going through the script.

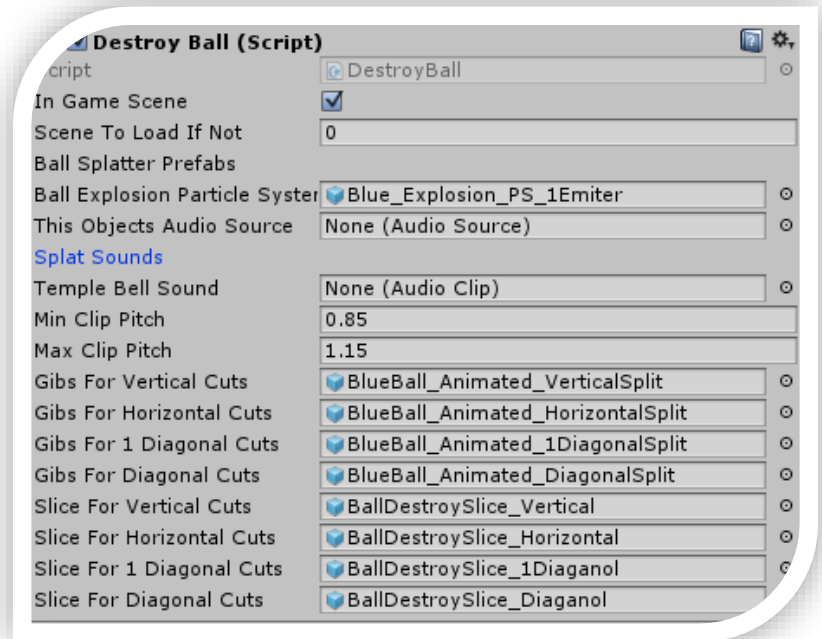


Figure 5 - Destroy Ball Class. Make sure every destructible ball has one.

Re-Using the ScreenFaderSingleton from the Color Switch Clone...

This game uses the ScreenFaderSingleton just like the ColorSwitchClone does. There have been a couple of additions to it. The “BackGround” Music/Settings object is a child of the Fader Canvas (so that it persists through the scenes with the Fader Canvas). Unlike in CSC, where the “Background Settings/Music” Gameobject was a separate persistent object. Since CSC had just 3 scenes the setup was a little simpler. We needed a method that would accept a parameter and load the parameter value, so that we could request specific scenes to load. Not just move in a linear scene loading fashion. There are a few Classes that are returning from the CSC Example Game though.... FaderCaller, FaderReferenceSetup, Singleton, ScreenFaderSingleton, RotateObject (little more verbose), ChromaticAberration, SimpleCameraShake, GameVariables, AnimationCurveMover, ObjectPoolScript, ObjectFollow, and Destroy/DisableGameObject have all been reused. They are very close (if not identical) to their versions in CSC. We also added another Static Class named “Tags” because there are definitely more tags in OSG. So, if you got somewhat familiar with those Classes from CSC they have remained mostly the same, and that should be helpful. I get tired a re-writing the same scripts over and over, so I have a starter template that saves me some time, and those scripts are part of it. They will be in a majority of the kits I publish.

The GameController gameObject...

The game scenes contain a gameObject named “__GameController (BackUp)”. The GameController consists of a few separate Classes...

- ***GameController.cs***
- ***LaunchController.cs***

- ***CountdownTimer.cs***
- ***DojoBoundaryController.cs***

➤ **GameController.cs**

The Game Controller controls the active game mode. See *Figure 6*. It calls the Game-Over Panel to activate when the round ends. Depending on the game mode it does a few different things. In Regular Mode it Activates 3 blue X's in the top right hand corner. If a ball is “missed” then it removes the blue X, and activates a red X at the position. The X's have an animation that plays when they are activated, this causes them to wiggle.

If we are in Chill Mode the Game Controller just calls for the Game-Over Panel when the timer has run out. It also keeps the current / best scores updated on the top left hand side of the screen. Note: best scores display will be updated the following round, for that game mode.

At the end of a round the game controller will disable the OSG Touch Slicer (the “sword/swipe object that cuts balls, and is followed by a trail renderer). The last two visible variables on the Game Controller inspector are “Game Is Running” and “Wait For

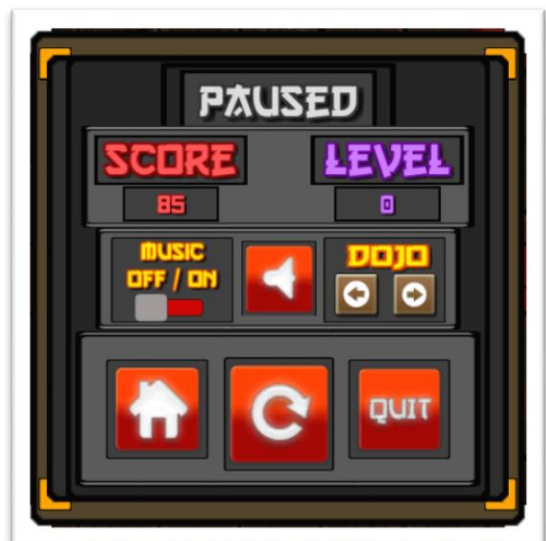
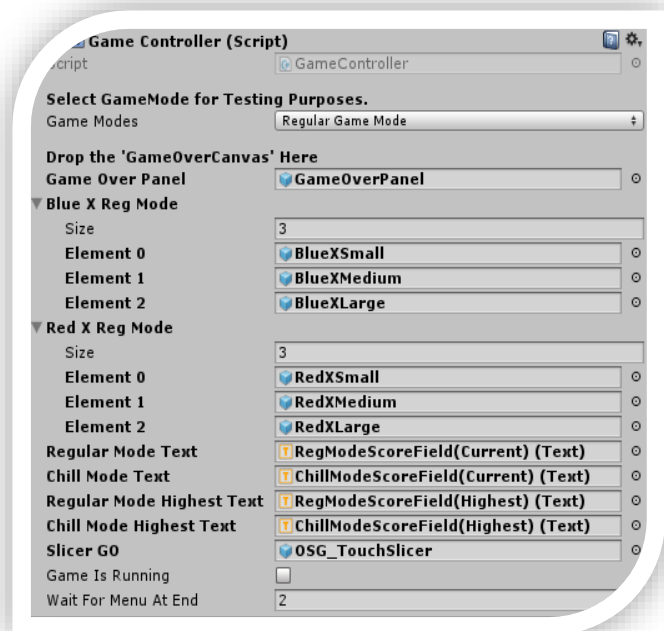


Figure 7 - Settings And Pause Menu that slides up from the bottom of the screen. The “Score” on this screen is the TOTAL score of all completed rounds.

Menu At End”. The Boolean stays true if the round is still active, and false if it is over... it is only public for visual reference. The Float “WaitForMenuAtEnd” is the amount of time after the Game-Over Panel is called that the “Settings/Pause Menu” will slide on to the screen. See Figure 7. 1.5f seemed like a reasonable setting to me. The Game Controller obviously does more than just that, but this is a pretty good explanation of the inspector side. The Game Controller repeatedly asks the “Launcher Controller” to launch balls, or bombs, depending on the game mode, while the “gamelsRunning”.

➤ LauncherController

LauncherController.cs is responsible for calling “Launch” on all of the “BallLauncher.cs” scripts in the scene. See Figure 8. Launch Controller handles all of the amounts, and timing of the Launches. So let’s go through each of the public fields. The two Lists labeled “Bottom Ball Launchers”, and “Side Ball Launchers” can be ignored for now. Those are populated via the script at round start. The “Bottom Launcher Salvo Amt” is incremented after each salvo. It should fall in



Figure 8 – Launch Controller Inspector

between the “Max Simultaneous Ball Launches”, and the “Reset Ball Launcher

Amt”. So that once it is launching 10, 15 or however many the “Max Simultaneous Ball Launches” is set to, then it will reset itself to the “Reset Ball Launcher Amt” and then start incrementing “Bottom Launcher Salvo Amt” again. This is so that the level can get progressively more difficult but only to a point. You could set the reset amount to 1 and the Max amount to 20, and it would increase by one each launch. There are some random launches mixed in to break up that monotony, but otherwise it would increment by one. The “min Wait Time”, and “max Wait Time” is how long it will take the ball to launch when a salvo is requested. Set these values so that the ball do not all launch at the same time. Set the “Time Between _____ Launches” variables to the frequency in which Balls, or Bombs should spawn. That way you can control the spawn rates of different items. The “Chill Mode Extra Ball” var adds extra balls when launched since Chill Mode does not have any bombs, or consequences we can increase the number of balls launched. Especially because there is no penalty for missing balls in Chill Mode. Then at the bottom of the inspector you enter the number of the bomb objects you want to start with on the first launch. They will increment themselves after that. The “max Bomb salvo amount”.

➤ CountdownTimer

CountdownTimer.cs is the round timer. *See Figure 9.* You can set at what time the clock (UI Text Component) should start flashing. You can also set what color it should change to once it starts flashing. The “timeBetweenTimerTextFlashes” is how long the time blink is. How many seconds the display is OFF. “Time Left” (controlled via script, it is just public for visual reference). The “uiText” is the text component of the canvas or panel that contains the timer. Then of course... the start and flashing color. Set them as desired.

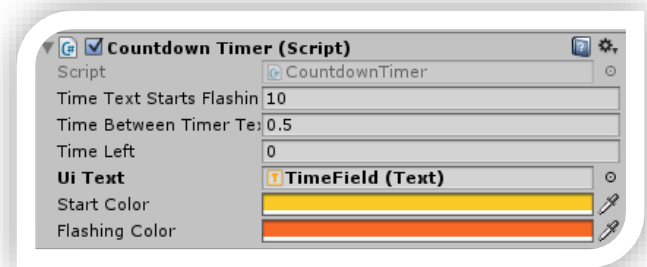


Figure 9 – Round Countdown Timer

➤ DojoBoundaryController

The DojoBoundaryController.cs is responsible for making ball, bombs, and powe-ups that pass into its trigger area inactive (returning them to their respective pools). See Figure 10. Whenever a ball passes into the trigger it increases the “ballMissed” var. The static variable that all of the classes access. There are 3 Red X gameobjects in the scene that are inactive. When we are in classic mode and a ball is lost, it will move one of the inactive red X’s to the lost ball’s position and then to the height of -22 on the Y-Axis (so that it’s just above the bottom of the screen). Then after a few seconds it goes inactive again. When 3 ball have been missed, all three of the red X’s have been used.

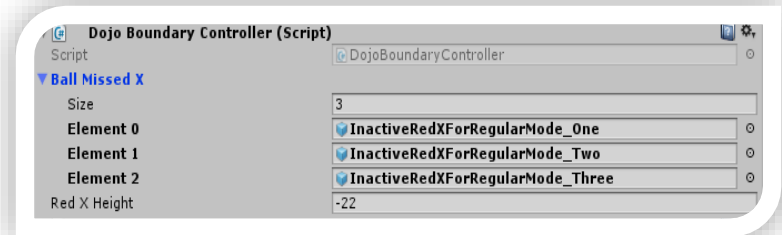


Figure 10 - Dojo Boundary Controller

**** Game Controller Section Ends****

The OSGTouchSlicer

The OSGTouchSlicer is the class that gets input from mouse position or touch position. See Figure 11. While a finger is on the screen, or the mousebutton(0) is held down the script fires a raycast into the scene at that position. If “emulateTouchesWithMouse” is selected then the mouse will power all of the Inputs. Input.GetMouseButtonDown(0) & Input.mousePosition. If it is **Not** checked then we use Input.touches. Checking for “touch.phase == TouchPhase.Began, Moved, and Ended” while recording the “touch.position”.

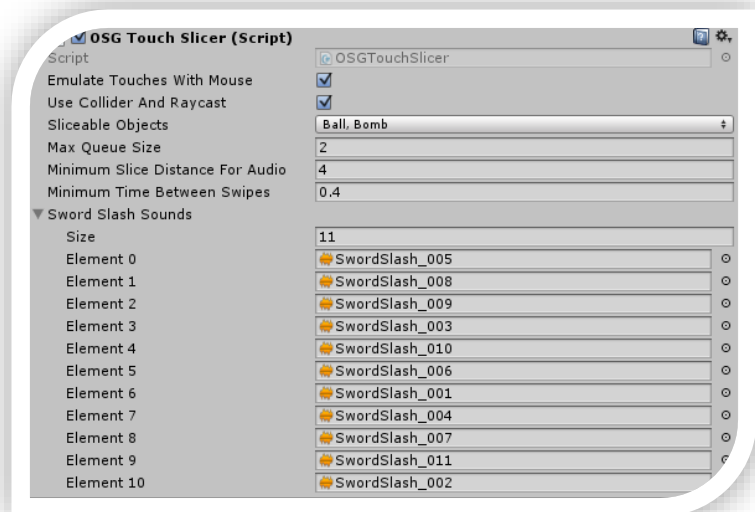


Figure 11 – OSG Touch Slicer

Since the kit will work on standalone, mobile, or web... I usually leave the “Emulate Touches with Mouse” checked.

There is another check that you can decide to do simultaneously (catches some fringe cases). You can select “useColliderAndRaycast” towards the top of the Touch Slicer Inspector. That will activate an elongated box collider that protrudes into the scene, allowing for raycast hits, AND collider hits. This collider is obviously larger (thicker) than the raycast line. This means that there are more “close” hits than with the raycast alone.

UPDATE:

I did end up adding a small radius `Physics.OverlapSphere()` check at the end of the Raycast. This caught most of the fringe cases that the collider was previously responsible for. After this addition was made, it really isn’t necessary to check “useColliderAndRaycast”, but if you run into any situations where some are being missed (during fast swipes, or other times), then this can still be “Checked(set to true)”.

The majority of the time, when you are moving quickly the raycast will be responsible for the cuts either way. To test/see this. Add a “Debug.Log” to the Raycast Destroy, and to the “OnTriggerEnter” destroy, set the console to collapse, and count or watch for the difference. For this reason I set the “Bombs” colliders to be slightly smaller than the balls (proportionately). Make sure that you select the layers that contain your ball, and bombs for the layers that the raycast/collision will interact with. The “maxQueueSize” is a queue that stores the most recent “transform.positions” of the Touch Slicer. Depending on how high/low the “maxQueueSize” is, will determine the swipe angle accuracy. Setting this to 2 is probably ideal. When we are trying to get the direction vector we will subtract our current position from the position 2 frames earlier. Note: a single frame was not providing enough delta, and so I implemented this queue to help. “minimumSliceDistanceForAudio” & “minimumTimeBetweenSwipes” both are in regard to the “swordSlashSound”. The MinimumSliceDistanceForAudio is the length a swipe has to be to trigger the slash sound. The minimumTimeBetweenSwipes” is how often swordSlashSound can play (we don’t

want them spamming constantly). The array of sound clips at the bottom of the inspector are the sounds that will randomly play when we swipe.

Below is a unit square that we will refer to for understanding the angle calculation (what decides which cut prefab to instantiate). In this game there are several balls. Each ball has a halved equivalent with cuts going from top to bottom, left to right, diagonally up-right to down-left, and diagonally up-left to down-right. When we hit a ball we compare our current position to where we were coming from (2 frames earlier). Now we use `Mathf.Atan2` and we feed it our `xPos` – our `storedXPos`, and our `yPos` – our `storedYPos`. Since we get radians back from `Mathf.Atan2` we will multiply our radians by `57.2957795f` (instead of using `Mathf.Rad2Deg`... it's the same thing anyway....

```
Float angle = Mathf.Atan2(currentPos.x – storedPos.x, currentPos.y – storedPos.y);
```

Then we multiply the returned radian value by `57.2957795f` and we get our angle in degrees. Now... with that angle recorded, we create several conditionals. See the UnitCircle below and note the

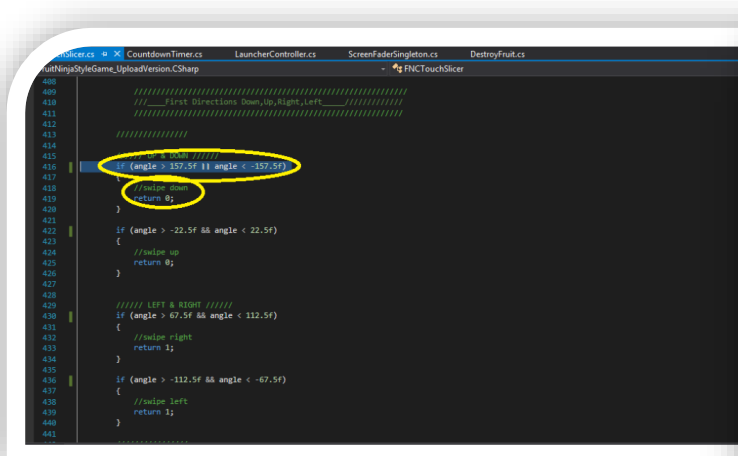


Figure 12 – Visual Studio
OSGTouchSlicer

degrees I added to it, and how they enclose an area that will help you visualize our “swipeDirection”. I have also pasted some code of the conditionals. Note the “angle”... So, the first conditional states that

```
IF ( “angle” is greater than 157.5f OR “angle” is less than -157.5f )
```

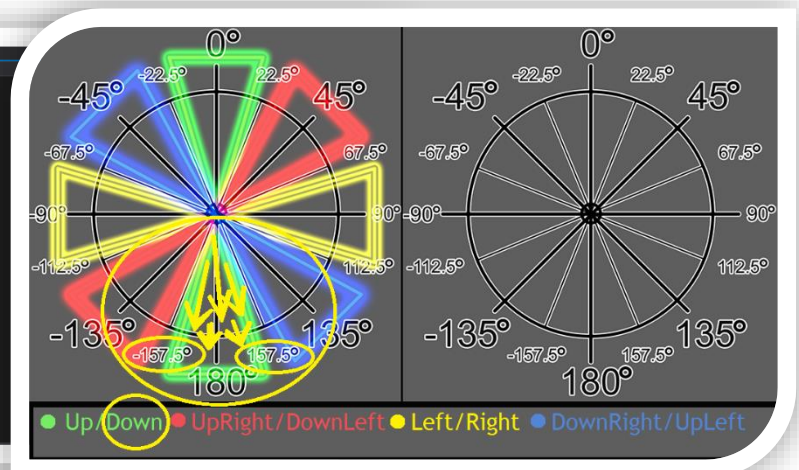


Figure 13 – Unit Circle showing swipe angles

I circled the degrees on the unitCircle, and the Conditional in Visual Studio with yellow ellipses. Notice the smaller font I added to the unitCircle pic are the evenly spaced degrees that are used in the Conditionals. We could make a 4 direction version, or an 8 direction (what we used here). You could also do a 16, or a 32... it's up to you. Just ask the right question. As an added exercise... what would the angles be for a 4 direction setup? Here is a hint... those degrees are listed on the unitCircle as well. The Answer is...

We would use Less than 45 AND Greater than -45 for UP

We would use Greater than 135 OR less than -135 for DOWN

We would use Greater than 45 AND less than 135 for RIGHT

We would use Less than -45 AND Greater than -135 for LEFT

****Note...** You could also decide that you didn't want to make the angles looser... you could use the smaller angles from the 8way, BUT if you wanted to make good use of the space you'd make forgiving "casually held phone" swipes work. For that you would use the full range in between -45 and 45 for up... instead of requiring the tighter -22.5 and 22.5 for up. Which would make it a lot looser.

This OSGTouchSlicer returns an integer when we ask it to calculate the angle of our most recent swipe.

It returns **0** for an up or down swipe.

1 for a left or right swipe,

2 for a diagonal up-Right or down-Left swipe, or finally...

3 for an up-Left or down-Right.

We pass this integer to the DestroyBall.cs's "CutBall" Method that way it knows which halves to instantiate. This is kind of a tricky/hacky way to do this. We could get even crazier and make it a 12 or 16 directional swipe and then create even more mesh variations, but this isn't the way we program these sorts of things. Fewer lines of code, and minimal number of repeated art assets are ideal... not the opposite. I didn't go that direction with this though.. so the more

complex it gets, the work to setup It up gets exponentially greater... bad juju. With the 8 direction version the ball mesh file is getting pretty big, and the whole thing requires a fair amount of inspector setup. It is not a maintainable, or elegant solution. I will however state that low powered systems can benefit from this sort of thinking/system creation, especially with “Destruction” and using “non-physics gibs” to make things look very breakable/real, but save the complex mesh deformation, and the rigidbody count/physics calculations... You can make pretty believable explosions by doing mesh swaps then animating gibs. Low end systems like some mobiles will thank you.

I personally loved the Mesh Deformation Version that cut the mesh, added faces to cover the whole, and then had a predetermined set of coordinates on a texture sheet that had the “Ball Core” where it unwrapped the UV’s to. It was just too heavy for my mobile, and had basically a 2sec Delay before finishing the job, and it made use of rigidbodies for the “new” pieces. One Rb for the original deformed piece, and another for the discarded “half”. I am sure it can be optimized and I will be looking to do that to make this kit more “Proper”, but for now, and for an easy to understand implementation. This technique will do.

The OSGTouchSlicer mainly triggers our “CutBall” method, and provides the current direction vector to a few other components... I.e. the direction of the ball splatter that is plastered on the dojo walls. It gets the direction from our OSGTouchSlicer, so it knows how far out to instantiate the splatter sprite renderer, and what rotation to make the splatter face. Etc. etc....

There are plenty of other Notes/Comments in the OSGTouchSlicer.cs / DoxyGen. If you need any further explanation have a look at them, and if you still don’t understand it then reach out, and I’ll make some more material.

The Animated Ball Cut Types

There are 4 Unity Animations that were made on Empty GameObjects, which have 2 Empty Children. This was done so that we had a generic base to use for the Ball Halves.

When making these animation I added the ball halves as children of the Empty Children. That way I could visualize the necessary motion of the different cut types, and so you could change out the meshes. After they were close enough to “done”, I removed the ball halves I had used temporarily, lined up the necessary ball half combinations, and started duplicating the different Empty Cut Type GameObjects for the number of balls I had. One by one I brought the Animated parent I.e.

“VerticalSliceAnimatedParent01” to the world cords 0,0,0 and then I brought “BlueBall_VerticialCut1”, and “BlueBall_VerticialCut 2” to world position 0,0,0 and I parented each half to the empty children. Went into the animation tab, and hit play. Then I made sure that the animation threw one gib to the left, and the other to the right, and called it “done”. Then I labelled that instance of “VerticalSliceAnimatedParent01” as “BlueBall_Animated_VerticialSplit” and dragged it into the project side to create a prefab. Then I brought over a horizontal cut animated empty, and repeated the whole process. That was kind of tedious, but it only took an hour (not counting the initial 30mins to create the 4 animations with the empties).

Later I made a particle system with stretched bill-boarded directional “tear drop” shaped sprites and a “round drop” shaped (non-stretched) particle system... but as usual all the transparency and particle systems started to kill my

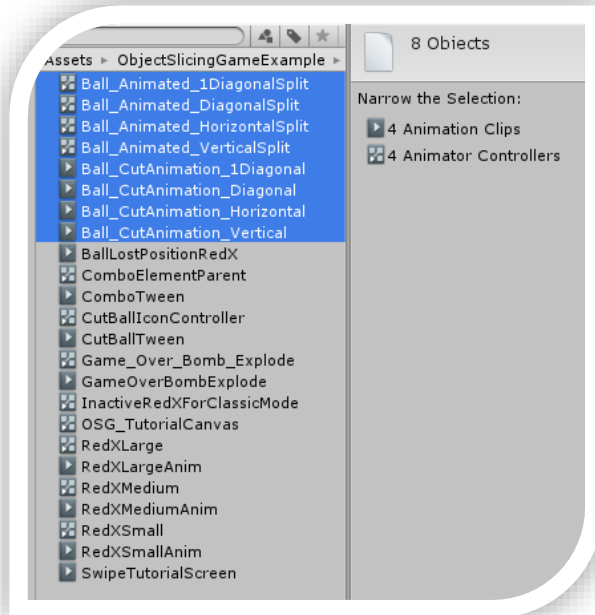


Figure 14 – Unity Animations and Animator Controllers

mobile device... Draw calls were up above 50, and the transparency was the bulk of the rendering hit when profiling on mobile (this was after decimating the meshes in 4 passes to bring down the total number of batches). The ball destroy “particle blast” was reduced from a pretty “3 system setup”, to a “1 system setup”, and the “2 system slice Particles” went down to a “1 particle system” setup. Lots of things were reduced until the materials and drawcalls were under control. That effectively describes the “Ball Halves”. The same 4 animations are used for all balls. There are ball juice splash particle systems, a slice particle system (elongated yellow diamond that is instantiated the direction of swipe), and then the splatter meshes which fade in and out (as described at the end of the TouchSlicer section above. They orient to the direction of our swipe as well).

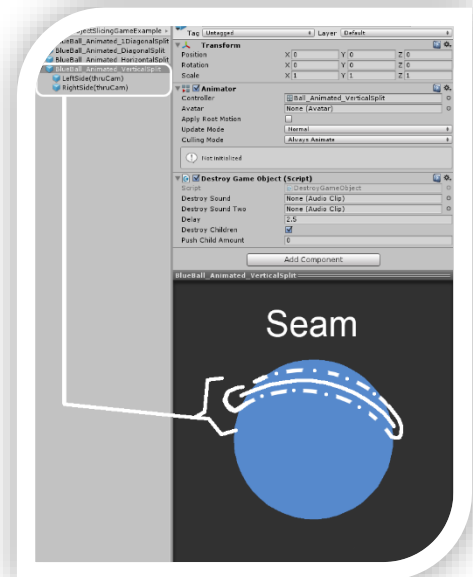


Figure 15 – An Animated Ball Half Selected (Vertical Split)

The only other thing these “DestroyedBallPrefabs” have is a Destroy GameObject Script, which will destroy it 2.5 seconds after we spawn it. That can be seen in As an example... to get the vertical sliced blue ball shown above, we would need to slice a ball (the whole untouched ball), and be swiping at an upward, or downward angle... that would call:

```
CutBall(0); //(Zero for a Vertical Split)
```

The Particles Used for the Ball “Cut/Destroy”

There are 3 particle systems used when a ball is sliced. As mentioned in the above text. The first particle system is the “BallExplosionParticleSystem”. It is the “juice” from inside the ball. See Figure 17. The next is the “Slice” particle. See Figure 18. It plays very quickly and is not that noticeable, but it adds a fair amount. It used to have a nice sparkle, as well, but there were just too many particle systems for low end hardware. Now it is just the diamond shape quickly growing to full size, and then zips back to nothing. Based on the slice direction the matching “slice diamond” is spawned.

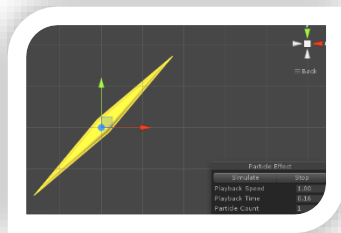


Figure 18 - Slice Particle (2)
“Diagonal Up-Right”

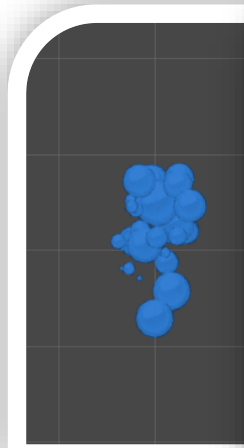


Figure 17 – A watermelon explosion particle

There are four versions of it as well. 0 = Vertical, 1 = Horizontal, 2 = 1Diagonal, and 3 = Diagonal, just like with the “Ball Cut Type”. Note that in the “BallDestroy.cs” Inspector above that the slices are in the

exact same order as the gibs. This is important. They have to be in that order.

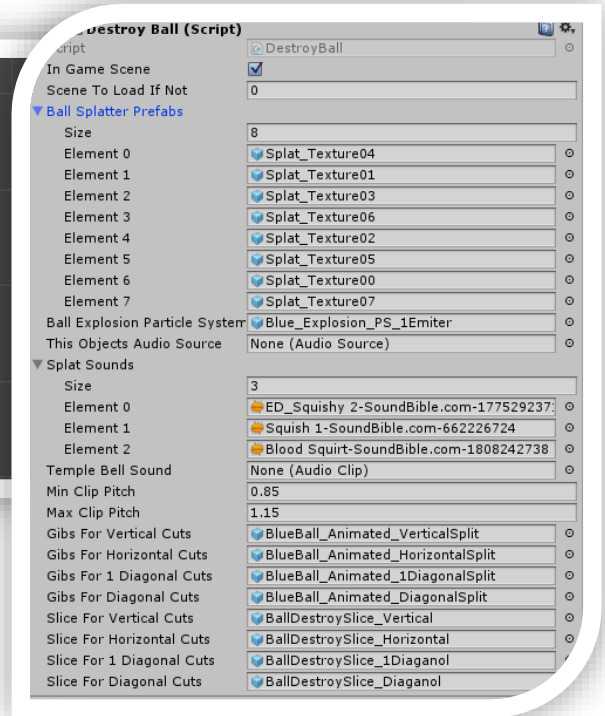


Figure 16 – The Expanded View of Destroy Ball. (Figure 5 is the collapsed view)

The Ball Launchers... not the Launcher Controller

The BallLauncher is on several scene objects. See the Red and Blue arrows in the SceneView. See Figure 19. The red



Figure 19 - The Arrows in the Scene are Ball Launchers

arrows are the bottom launchers in the scene, there are 5 of them in each game scene. The LauncherController gets a reference to each of these. The BallLauncher is the class that actually fires the ball or bombs, but the LauncherController is what requests they fire. That way the LauncherController can loop thru the “salvoAmt” and call “Launch” for every loop iteration. You will see that there are not many options on the

BallLauncher.cs. The “force” is the force that the object is fired at. We use “ForceMin” and “ForceMax” to generate a random value and we assign it to “force”. “Force” is always between “forceMin”, and “forceMax”. The last variable is a “CannonThud” sound clip. This is the sound that plays anytime a bottom launcher fires. Finally, just above the “CannonThud” There are two Arrays... “BallPoolScripts”, and “BombPoolScript”.

That Array(and single single slot) will actually contain all of the object pools in the scene, once the game is running. There are two private gameObject variables named “ballPoolsGameObject” and “bombPoolGameObject”. The “ballPoolsGameObject” is tagged with “BallPools”, and “bombPoolGameObject” is tagged with “OtherPools”. Each of these gameObjects have children. There is a child for every type of ball we launch in the game. The same can be said for the “bombPoolGameObject” but it has only one child for the bomb type instead. So we will assign the “BallPoolScripts” array the returned values of “ballPoolsGameObject.GetComponentInChildren<ObjectPoolScript>()”. That way “BallPoolScripts” has a reference to all of the ball pools. We will do the same thing for the “BombPoolScript” GameObject but with the singular

GetComponentInChildren. So we do

“bombPoolGameObject.GetComponentInChildren<ObjectPoolScript>() that way
“BombPoolScript” is a reference to the bomb pool. See Figure 20a.



So, when the LauncherController needs to launch 4 balls, It will for loop through the number of ball requested(4), And each time it will start a coroutine (still in the LauncherController) that will call the method “LoadAndFireRandomBall()” on the BallLauncher.cs.

Figure 20a - The 2 GameObjects that are Tagged “Color Ball Pools” and “Other Pools”... and their children.

3.) Scripts See Figure 20b.

Some scripts we didn't cover... The Inspector, and a brief description.

I will try to keep this part short. The info in the Setup and Things to Know sections really covers some of the most important points. Again, there is a lot of information in the scripts themselves, and if you happen to not be in the IDE, you can always access the DoxyGen Docs Page... for the Class/Method Descriptions and the members list.



Figure 20b - List of all Scripts in OSG

Select Dojo Background See Figure 21.

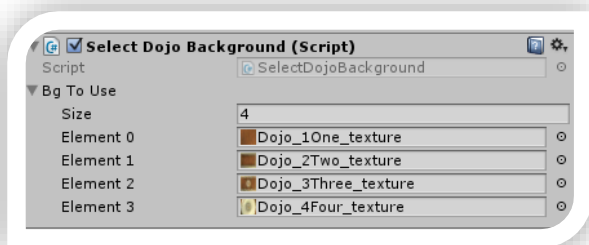


Figure 21 - DojoBackgroundSelector.cs - Inspector

There is a gameObject in all of the scenes that have a Dojo Background named “DojoChooserParent”, and it has a child gameObject named “OSG_ChooseDojo”. This script has an array, a few simple variables, and 2 methods that cycle forward, and backwards through the array of

“Backgrounds”, and then changes the materials texture to one of the background textures. The value that holds where we are at in the cycle is a Static Int in the GameVariables class. When its value is updated we save its value to PlayerPrefs. That way all of the scenes, and even when the game is closed you can come back to the same dojo. I was originally going to make them unlock based on the Players Experience/Level, but never got to it. The project took me a little longer than I intended to spend on it, with all of the performance issues and refactoring.

Show Cut Ball UI

ShowCutBallUI.cs has a very simple job. The only thing it does is update WHICH current score/best score is displayed at the top left corner of the game. It disables all of the text elements in the 2 arrays on start, and then based on the current game mode it enables only those text elements. The game controller controls the values, and this class controls whether or not they are enabled, or disabled. In hindsight, I should have just put this functionality on the game controller itself. I don't think I really know that I did that until writing this... lol ;) How silly... I will restructure and have the game controller take over that responsibility at a later time.

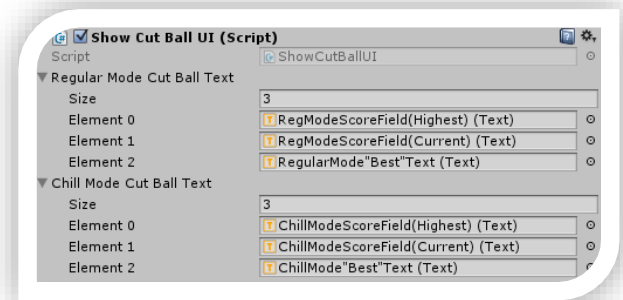


Figure 25 - ShowCutBallUI.cs - Inspector

Ball Destroy Combo

BallDestroyCombo.cs takes care of the ball cut combo pop-ups. The system has an approximate 0.5 second count in which it will record how many ball are destroyed. As soon as that time runs out it starts it again. So basically if you swipe through 6 ball in a half a second, it will say... “ok, the player reached the

minimum required “ballDestroyedInTime” within a half-second countdown. Enable the combo text at the anchor position nearest to where the last ball was cut, and select the appropriate sprite for the number of balls they cut. By unchecking the Boolean “useImagesForComboNum” you can have a text component fed the combo amount. Which works fine... I just so happened to have numbers 3 – 9 on the UI atlas in the same style as the combo text so I use them. It looks okay with the regular text, but mine had a subtle gradient and I wanted it to match. The

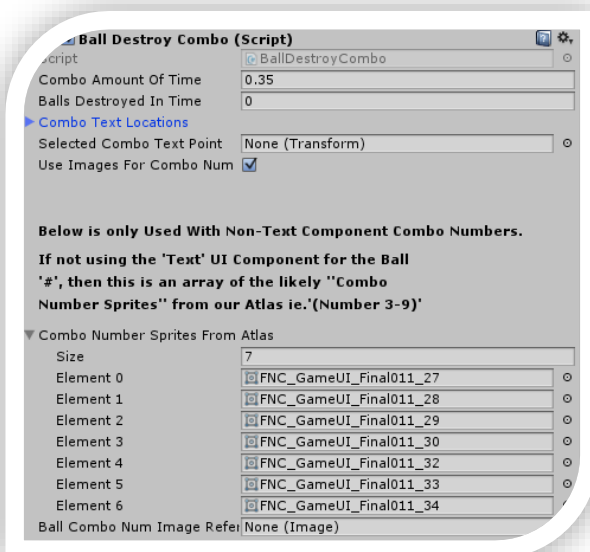


Figure 26 - BallDestroyCombo.cs - Inspector

“Text” component is in place (just disabled), so you can try that out too. This class has some detailed comments in it, I would recommend checking them out, and then inspecting the canvas in the scene that is labelled “OSG_InGameDisplaysCanvas” (a child of OSG_Canvases) if you have any additional questions.

ChromaticAberration & SimpleCameraShake

Both of these scripts are very similar to the versions in CSC. I did more complete commenting in them, and I made several fields that were formerly public, private. Keeps the clutter down in the inspector. If you have seen these before you will be able to tell that the inspector is much cleaner... I am just going to recommend you go look at how these are setup.

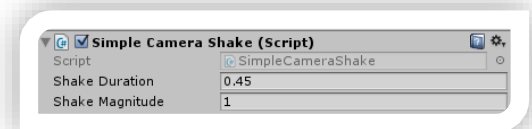


Figure 27 - SimpleCameraShake.cs - Inspector

As you can see from the inspector of the “SimpleCameraShake” the variables are pretty self-explanatory... once those variables are set, and that script is attached to your main camera... you can call StartShake(); and If you look at the Chromatic

Aberration.cs’s Inspector... All you have to do is drag the

ChromaticAberration Shader from the project directory onto the field, set the duration, and speed (both pretty self-explanatory), and then call StartAberration(); if you want to really see what the chromatic aberration does then set the duration high, and the speed low, that way you can watch the color channels expand and contract.

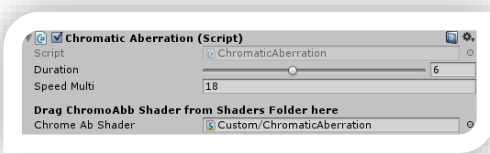


Figure 28 - ChromaticAberration.cs - Inspector

Settings and Pause Menu

SettingsAndPauseMenu.cs is responsible for displaying the menu in the game. There is only one menu and it is not a traditional “Pause Menu”, or a traditional “Settings Screen”. This settings and pause menu can be accessed from

any scene (it is a child of the fader canvas). When the user hits “Esc”, or “Back” on android the menu slides up onto the screen.

From it you can exit the game, reload the current level, go to the main menu, toggle music, see the player’s experience/level, and if you are in a scene with a dojo background you can cycle forward, and backward through the available dojos. For this class you will need to drag the “fullSizeBG” onto the “tintColorBG” variable, and then drag the “menuWindow” panel onto the “pauseMenu” variable. Check “ignoreGameTime” (so that it will still slide up onto the screen if called, and the Time.timeScale is set to 0f). Check “useScreenSizeCalculations” to make the class measure the screen size on game start, and then find the center. It will make it so that no matter the screen size, the menu will always slide from off screen below the viewable area, and it will always slide to the center of the screen.

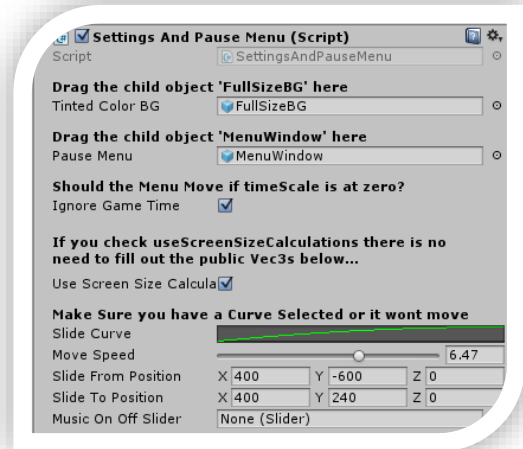


Figure 29 - SettingsAndPauseMenu.cs

Note: if you change the screen size after the game has started, or later you make the game work in multiple orientations, then will have to set a way to call the “calculate screen size methods” again, otherwise the menu will use coordinates that are not accurate anymore.

For now as long as the game is locked to landscape, in production it will be fine, and in the editor as long as you don’t resize the game windows “mid-play” it will be fine.

The rest of the variables are fairly straight forward. The curve needs to be created (this is how it will slide (the “easing”)). Set the move speed to your liking. The “slideFromPosition” and “slideToPosition” only have to be filled out IF you did **NOT** check useScreenSizeCalculations. Finally the slider is a reference the the slider underneath the Music On/Off text, that shows us which is selected. It

cannot be moved by tapping or sliding, it is just for visual reference. To Mute or UnMute the music you hit the red button with the speaker on it in the middle of the settings and pause menu... See Figure 7. Refer to the SettingsAndPauseMenu.cs Comments or DoxyGen Doc Pages for any more information about the class.

FaderCaller & UIDojoSelector

FaderCaller.cs and UIDojoSelector.cs are both very short and simple. The FaderCaller has returned from the CSC Example Project... It only does one thing. It instantiates our ScreenFaderSingleton. Way back when, it did a little more, but it only has one responsibility now. I will probably start handing this off to some other manager class, but up until now... it has worked well. “If it isn’t broke, don’t fix it”. So FaderCaller is on all of the cameras in the game, and that is one of a handful of reasons that the game can be started from scenes other than Scene0 without there being several errors/reference exceptions. You should be able to fire up the game from any scene while in the editor, BUT the intended use is start from Scene0 (I know I

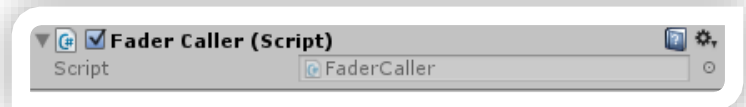


Figure 30 - FaderCaller.cs has returned from CSC. It calls the ScreenFaderSingleton into existence from the splash scene. That is it.

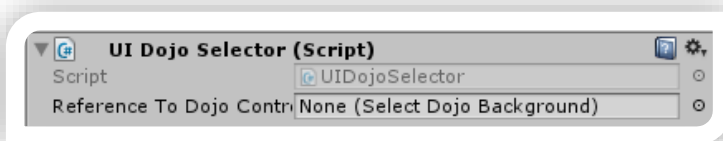


Figure 31 - The UIDojoSelector.cs is a short script that we attach to the SettingAndPauseMenu gameObject and it makes the “Forward” / “Backward” arrows that change the “Current Dojo” call Methods in the class, and those methods increment/write to disk the Static var that determines what DojoBG we use.

have said that a few times, just keep an eye out for any issues if you start the game elsewhere).

The UIDojoSelector.cs is a pretty small class as well. It uses a static “instance” reference so we can access the SelectDojoBackground

class that switches the backgrounds. The only methods this class contains are RunChangeNext(), and RunChangePrevious(). The SettingsAndPauseMenu UI holds two buttons (a back arrow, and a forward arrow) under the “Dojo”. Those arrows call these two methods. See Figure 7, again. That is pretty much

everything... I know some things have been missed, but most the time how these things work can be deduced by jumping in, and looking around. I am sorry this ended up being so long. I try really hard to make sure I include lots of comments, and documentation, but often I feel it is too much. Let me know what you guys (and gals) are interested in, in the way of documentation.

4.) UI, Menu, and the Canvas

UI... stuffs!

The UI...

Okay, I think that the UI/Canvas Scaling portion of this complete project template will go much better than with version 1.0 of CSC. This project was built for landscape mode only, and I did add canvas scalers to the canvas objects. The canvas scalers are all set to scale with “width”, and the reference resolution has been set to my device resolution, which is 800x480 in landscape. Most android devices will be 15:9, 16:9, or 16:10, in my experiences, and this game should not have any problems with those. So, if for some reason you run into any inappropriate scaling, stretching, or misalignment of UI resources, then you have some work to do!! Lol. Try inputting your own devices reference resolution if you have problems, and make sure that the Canvas Scalers are on. If you are not familiar with the Unity UI system I would suggest watching some of the Live Trainings, and going through the manual... it’s, but some parts can be more difficult.

This kit is not necessarily a “ready to release” template, but rather a fairly complete “vertical slice”, or “Example Game”. It still needs upgrades, some new assets, game-modes, and some good social/score integration work, but it should get you started if you are looking to create a simple Object Slicing Game. These kits are normally intended to serve as a starting point, and then the user expands, adapts, and dresses it to their liking. You can do whatever you would like with it, besides the audio files that I have cited sources for, everything else is mine and I am telling you to use it however you wish. Whether you use it for learning, or as a mini-game in a larger experience, or if you strip it apart, and only use a small chunk... you can do anything you want, with one exception. You cannot repack it and sell it as a asset/template.

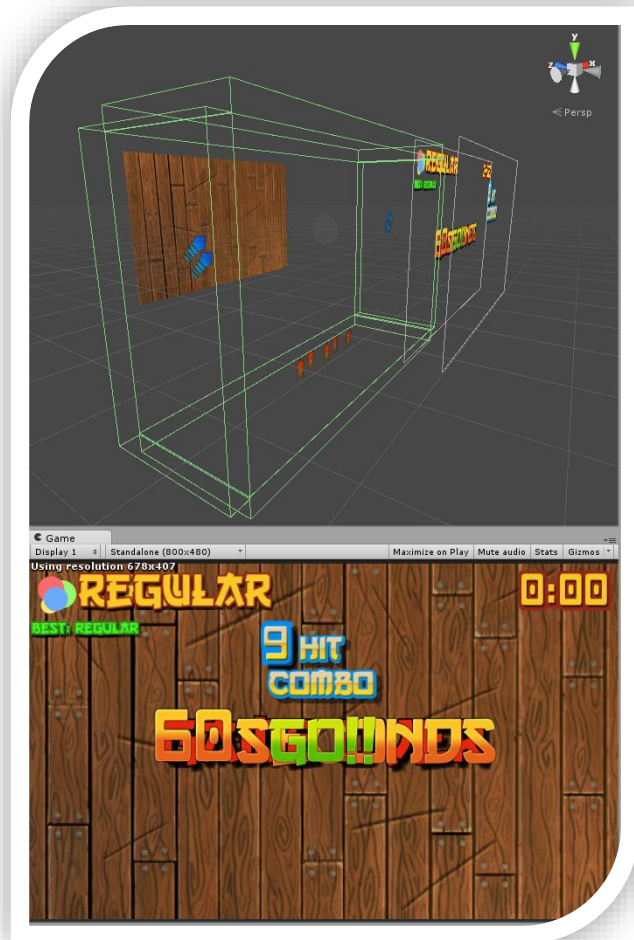


Figure 32 - Scene and Game View

I want to say thank you again... Everyone has been very kind, and supportive on the forums, store, and via e-mail... THANK YOU!

Best Regards

-Brian 🚀👤

If you haven't done so already check out Color Switch Clone on the Asset Store...

<https://www.assetstore.unity3d.com/en/#!/content/59187> As always, I appreciate reviews, everyone has been great about that. It really helps when working through a project. When you can see people that appreciate having some stuff to look through/learn from.

I felt like there could be a few extra free complete project templates on the asset store. Unity keeps pushing out new features and releases, and more and more people are diving in. There are some really good kits, and lots of good resources on the asset store.

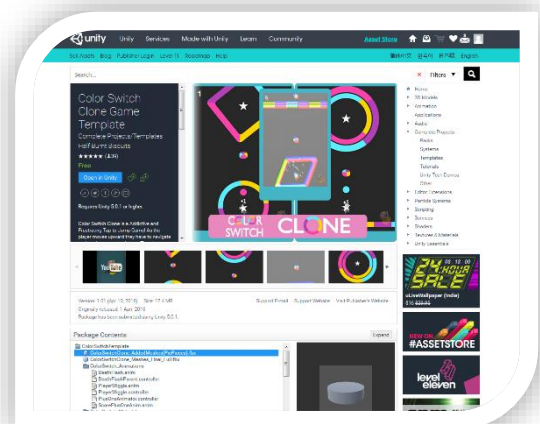
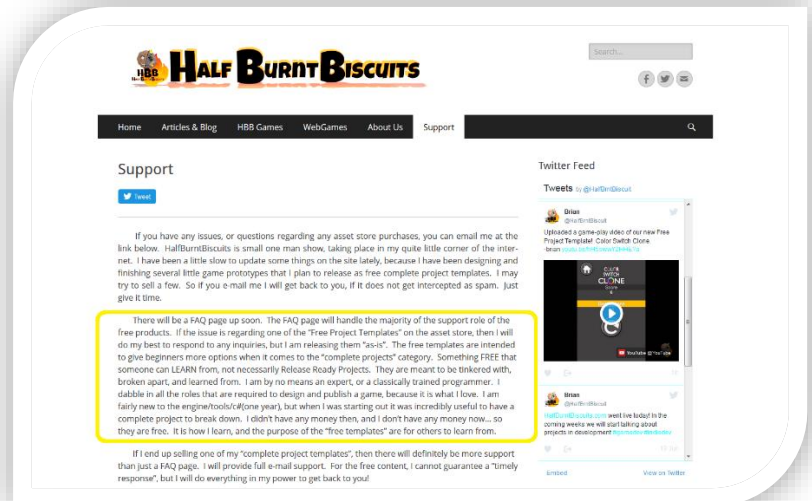


Figure 33 - ColorSwitchClone store page screenshot

If anyone has any comments, questions, or suggestions you can e-mail me at HalfBurntBiscuits@gmail.com. I will respond as soon as I can. I will eventually get a FAQ page setup, but as a solo developer it takes me time to take care of game design, making the game, and updating the website. The “free” Complete

Free Project Templates / Example games are released “as-is”, and the majority of the “support” will be provided by the “FAQ”. Paid Projects/Examples/Asset support does take priority.

I will also be selling some of the Complete Project Templates / Example Games I publish, but they will be super cheap, and e-mail support will be provided. I guess what I am saying is if you send me an e-mail, be patient. As long as the spam filter does not grab the e-mail, I will get back to you. If you haven't heard anything in 5 days, then definitely email me again, maybe even try a different address, though I do try to go through the spam as well, I may have missed something. So I will get back to you, it may however take a few days.



Best Regards

-Bria  

Contact E-mail:

Brian

HalfBurntBiscuits@gmail.com: Comments and Suggestions are Welcome. This is the best way to contact me.

YouTube Video of the Object Slicing Game in action:

<https://youtu.be/zWnsYlsrDLY>

APK: (is signed, but this is on my google drive). Plan to record a video of adding social integrations, and in the process I will publish to Play Store.

<https://drive.google.com/open?id=0B2k4ZlzaIPfRdmhRdzVIWTFITHc>

WebGL build of the Object Slicing Game – Example Game:

<http://www.halfburntbiscuits.com/object-slicing-game/>

DoxyGen Docs Download Link (there is also a copy in a zip file within the project):

<https://drive.google.com/open?id=0B2k4ZlzaIPfRbFVpWk5sdWliWWc>

HBB Homepage:

<http://www.halfburntbiscuits.com/>

**Website: Working on getting some content uploaded. Site has a WebGL Demo of Color Switch Clone, and the Ball Slicing Game. ** But it takes me a while to get other things updated on the site.

Twitter: New Account. Once some more content is uploaded to the site, I will use this more.

[@HalfBrntBiscuit](#)

*The Change-logs are now in the same folder that the rest of the Documentation is in.

- None yet for OSG "Object Slicing Game" ...

****AudioClips for Ball Slicing Game - Cites and Sources:****

- Game Background Music is "SHAMISEN Slow Mood In D-Minor" by Jens Egbert. You can check out the loop at : <http://www.looperman.com/loops/detail/96855/shamisen-slow-mood-in-d-minor-by-jensmuse-free-80bpm-ethnic-harp-loop> OR check out his YouTube Page at <https://www.youtube.com/user/jensdab>. He has lots of nice work. The sound used is a loop from Looperman.com, and was created by Jens Egbert.
- "Sword_cutting_through_air" was used for the "swipe/slash" sounds. The original File contained about 20 seconds of slashes. I cut 11 individual sounds out of it. Check their website at : <http://www.freesfx.co.uk> About The track is half way down the page at listed as "sword_cutting_through_air" at : <http://www.freesfx.co.uk/soundeffects/swords-knives/>
- Tracks [BloodSquirt, ED-Squishy 2, Squish-1, Metal Gong, Japanese temple bell] are from soundbible.com at: <http://soundbible.com/> The original author / Recorded by Mike Koenig. All of these tracks from SoundBible.com is Mike Koenig, and all sounds are under Attribution 3.0 Unported (CC BY 3.0). Full License file is in this files "parent directory". I would like to give special thanks to Mike Koenig... he has a lot of great sounds! Thanks <http://soundbible.com/royalty-free-sounds-1.html>
- "FreeSfx Pyro – Fuse Burning", by VoxHouse Studio: [Film * Music * Design]]. Source file found at <https://www.youtube.com/watch?v=hhl8TE0xfc>. All credit goes to the original author, and/or recorder... There is a graphic inside this font's folder that declares this sound available to use for free. Check out the YouTube

link, and their website at <http://www.VoxHouse.net> Thanks, Great fuse
burning sound!!