Custom Slots Framework

Version 1.3 Documentation

1.Introduction

Feature

Custom Slots(CS) Framework is a lightweight, flexible modular slots framework that can be integrated to your project as a mini-game or as a standalone full-featured slot machine game.

Demo - Document - Forum

While offering full customizability, CS tries to maintain its core simple and adaptable to any scene a spinning reel could participate. It is also built purely on UGUI to keep up with Unity updates and to make integration less painful.

Feature

* Any number of reels, rows, symbols, lines, effects etc supported.

* Automatically adjusts layout according to your setup.

* Procedural symbol loadout generation and simulation.

* Pop multiple instances of CS with smooth transition any moment should your game need.

* Effects can be easily tweaked via inspector, or disabled to use your own.

* Wilds, scatters, free spins, betting and other slots functions supported.

* Change reel behaviours and swap symbols upon entering free spin, bonus etc.

* High Performance, optimized C# codes, minimum Updates, no GC Allocation during updates.

* Bonus pay table generator included.

* Callbacks and runtime capability, extendability.

* A full-featured complete slot game included as Demo.

* Included free-for-commercial-use arts by Nz.

* Detailed documentation.

CS is part of my other game-under-development and you can expect updates coming as the development progresses.

We are also available on the official CS thread on the Unity forum.
Please do not hesitate to ask a question or report a bug you found.

(CS uses Demigiant's powerful Tween library DOTween. It is not necessarily to learn DOTween but desirable to fully understand and customize CS.)

FAQs and Tips

* You can hover your mouse cursor over parameters on inspector to show tooltips.

* Press "Refresh Layout" button to apply changes after modifying layout, number of rows, reels or symbols per reel setting. Use Shortcut Tool to quickly access the button.

* Demo scene can be found under /CustomSlots/Demo folder.

* FPS is set to 70 in Demo scene.

(More Coming Soon)

2.Quick Start

Setup

There are several ways to set up your slot game.

Using prefab

1. Open or create a new scene in Unity.
2. Under CustomSlots/Prefab folder, drag and drop "Canvas CustomSlot" prefab into your scene.

or drag and drop "CustomSlot" prefab into an existing canvas in your scene.

Alternatively

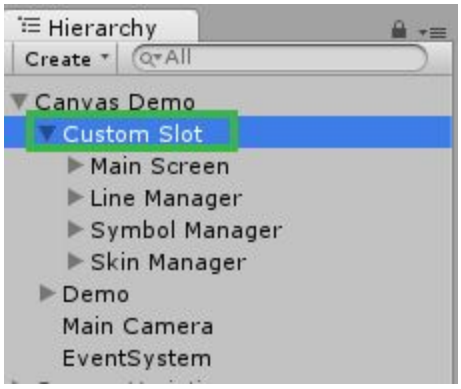
1. Load a demo scene and rename it to make a new working scene.

or load a demo scene into your active scene and copy/move stuff you need to the active scene.

Configuration

After setting up CS, you should be able to hit play and interact with default UI.

You can now start tweaking your CS via its inspector. The document does not cover all the configuration element CS has, but the most parameters will show you a tooltip when hovering your mouse over them, so please refer to them.



(Select Custom Slot in Hierarchy view to bring up CS inspector.)

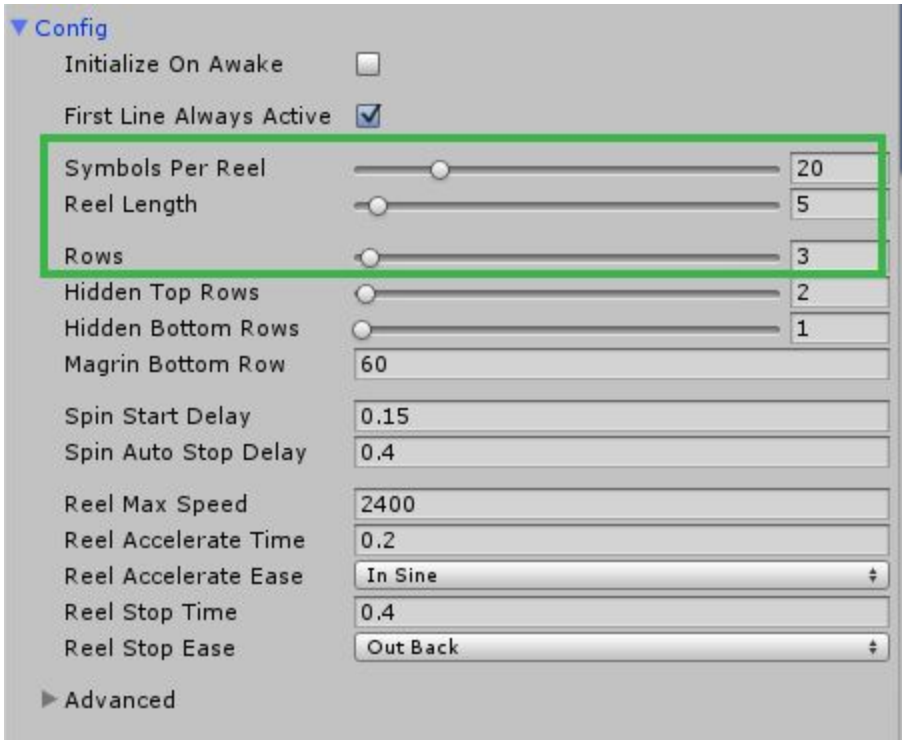
3.Basic Customization

Reels & Rows

To change the number of reels or rows, open CS's inspector and modify *Reel Length* or *Rows* in Config section. Then click "Refresh Layout" button.

Rows represents visible rows on reels. Hidden rows are extra, decorative rows that prevents graphical glitches from happening when reels spin.

Note that upon changing the number of *Rows*, *Reel Length* , or *Symbol Per Reel* parameter, the default symbol specified in SkinManager will be used when a reel needs new symbols.



Slot Modes

CS has 3 pre-defined slot modes CS has which are Default mode, Free Spin mode and Bonus mode. Each mode can be configured for different behaviour to vary the pace of gameplay and also allows you to swap symbols on the reels.

Modes can be configured via Modes section on CS inspector. Some important parameters are as follows.

Spin Mode

> Spin Mode determines how reels stop once they start spinning. "Auto Stop" makes all the reels automatically stop after certain time specified in *Auto Stop Time*. "Manual Stop All" makes all the reels stop when Play() is called. "Manual Stop One" stops a single reel when Play() is called.

Force Play

When set true, reels start spinning as soon as a round starts.

Symbol Swaps

This is a list of symbols you want to replace when the mode starts. All the symbols that matches the symbol specified in "*from"* will be replaced with the symbol specified in "*to"*. To specify a symbol, drag&drop a symbol from SymbolManager to either field.

For other parameters, please refer to tooltip hints on inspector.

Free Spin mode would normally be played at a faster pace with spins automatically starting until there's no free spins left. But you are not limited to use the mode as the name impliest. (We only named them in this way for simplicity)

For example, you could change the free spin mode a punishment mode by increasing cost per line or by swapping some symbols with ones that have negative payouts. Or you may choose to only use the default mode if feeling lazy.

Layout & Skins

In the layout section of CS inspector, you can modify the size of symbols, spacings, and the padding of main screen. After editing some values, click "Refresh Layout" button to apply changes and let CS adjust its layout.

Upon refreshing layout, each UI element CS has (such as gameobjects of rows, reels etc) will be also refreshed/replaced with the ones SkinManager has references to. So if you need to change the look of all the reels, you will only have to modify the reel SkinManager references.

Symbols

Symbols are managed by SymbolManager. Any gameobject with <Symbol> component within SymbolManager's transform hierarchy will act as a symbol.

To create a <Symbol> gameobject, click "Add a new Symbol" button on SymbolManager's inspector. Or simply press CTRL+D on an existing symbol to duplicate the symbol.

Here are the important parameters <Symbol> component has.

Sprite

The sprite to be displayed for the symbol.

Pays

Specified in a list of numbers separated with comma. (e.g 0,0,100,500).
The first number represents a pay when there's only 1 symbol in a line.
The 2nd number represents a pay when there're 2 matching symbols in a line(2-in-a-row, in other word). And so on.

While the number is "0", the symbol is not considered as a hit (or win).

In the above example, the symbol's pay would be 100 when 3-in-a-row and 500 when 4-in-a-row.

If there are more symbols in a chain than the numbers specified in *Pays*, CS currently returns the last pay(500 in the above example).

Pay Type

Represents how a symbol rewards a player.

"Normal" pays a player normally and "FreeSpin"(and "Bonus") gives a player free spins(or bonuses) both according to the numbers specified in *pays* parameter. "Custom" does nothing and it is there so you can use your own code.

Match Type

There are currently 3 match types predefined by CS. A "Normal" symbol can only match the same kind of the symbol. A "Wild" symbol can match any kind of symbol except scatters. A "Scatter" symbol can be anywhere on the slot's to make a hit(win).

You may add new types or choose "Custom" and evaluate each symbol manually but in most cases they should suffice.

Frequency

The parameter is used by SymbolGen and represents how often the symbol is randomly chosen to appear on a reel. The value is seen as relative weight in all the other symbols.

Min Count Per Reel

The parameter is also used by Symbol gen and represents a minimum number of symbols guaranteed to appear on a reel.

Lines

Lines(or pay lines) are managed by LineManager. Any gameobject with <Line> component within LineManager's transform hierarchy acts as a line(or payline).

To create a <Line> gameobject, click "Add a new Line" button on the LineManager's inspector. Or simply press CTRL+D on an existing line to duplicate the line.

Lines don't need to be directly under LineManager transform and can be moved and layouted freely as long as they belong to LineManager's hierarchy.

Here are the important parameters Line component has.

Order

Specify an order which CS will use for various operations like determining which line a hit check will be performed first. A line ordered 0 will be the first to be checked, following 1,2,3 and so on.

Row

Specify an index of a row the line's hit path starts. 0 always being the top-most-row.

If your slot has 3 rows, the row indexes would be as follows:

0 = top-row (1st)
1 = middle-row (2nd)
2 = bottom-row (3rd)
and so on.

Path

We'll cover how paths work in this section. However, it's a lot easier to understand if you take a look at the lines in Demo than reading the mechanism. Be sure to check Gizmo in the scene view which will visually show you the path.

You can skip the below sections if you took our advice.

- Inside Mechanism

Each line has a path(or route) that will be traced by CS when it performs hit checks after all the reels stop spinning.

A hit check will be always performed from the left-most reel to the right-most reel . It picks a row on a each reel then checks the symbol the row has. A hit check starts from the row the line specifies (See above).

A path is specified in a list of numbers separated with comma. (e.g.0,1,-1,0)

Each number in a list represents the number of row(rows) CS's hit check will shift when it checks the next reel.

So if the number is 0, CS will check a symbol on the same row when it moves to the next reel. And if the number is -1, the row CS will check will be one-row above the current row. And so on.

Again, make a good use of Gizmo to visually understand/confirm how the path changes.

- Loop Mode

Loop mode indicates how the path will continue when there're more reels than the numbers specified in the path. "Stay" means the index of the rows for subsequent reels will stay forever. "Continue" means the index of the rows will keep shifting by the last path's number for subsequent reels. "Loop" means the path will loop from the start and "PingPong" means the path will be reverted.

Effects

Effects like transition, intro animation, line switching effect and symbol highlighting can be modified via CS's inspector. In most cases, animations and effects are done by tweening transform and color.

The parameter names should be self-explanatory on how they influence the effect.

Symbol Hit Effects is a list of effects that will be played when a line is a hit(win).
Each effect on the list is fired when a certain condition is met.

For example, *If Chains At Least* parameter is set to 4, it will be only played when a hit has more than 4 chains(4-in-a-row, in other word).

When *If Symbol Matches* is specified, the effect will be only fired for the symbol specified.

Callbacks

CS has several <UnityEvent> events which you can subscribe <UnityAction>(delegate) methods. Depending on an event, subscribed <UnityAction> will be invoked when something happens in CS. For example, *onStopReel* event is fired each time when a reel stops and all the <UnityAction> that listen to will be invoked.

By default, BaseSlotGame that comes with CustomSlot prefab will show you in a debug text when those events are fired. You can also open up ElonaSlot.cs and refer to its usage. The list of events is as follows.

UnityEvent onActivated
Fired when CS is activated (After In-Transition is completed).

UnityEvent onDeactivated
Fired when CS is deactivated(After Out-Transition is completed).

UnityEvent onRoundStart
Fired when a round starts.

ReelInfo onReelStart
Fired each time a reel stops.

ReelInfo onReelStop
Fired each time a reel stops.

HitInfo onProcessHit
Fired each time a Hit Check(including scatter) on a line is successful.

UnityEvent onRoundComplete
Fired when a round ends.

SlotModeInfo onSlotModeChange
Fired when CS's mode changes(e.g going to free spin mode).

LineInfo onLineSwitch
Fired each time a line is enabled or disabled.

Some events are derived from<UnityEvent> and pass information as an argument to its listeners. To receive the argument, add a parameter of its <Type> in your method.

For example:

public void ProcessHit(HitInfo info) { Debug.Log(info.hitSymbol.name); }

When having this callback method subscribed to *onProcessHit* event, each time a player wins a line, the above method will be invoked and shows the hit symbol's name in debug console.

To subscribe to an event, you can open up CS inspector to use its editor GUI (just like you do with UIButton's onClick event) or add a line of code in your script like this: (*slot* being the reference to CS)

slot.callbacks.onProcessHit.AddListener(ProcessHit);

As for which information you can get from some events, please refer to SlotInfo.cs.

Basic Runtime Methods

Here're some basic methods you can call from your scripts.

CustomSlot class

public void Play()

Starts spinning reels or stops them(or stops each one manually) if they are already spinning. CS's current slot mode setting is used to determine how Play method behaves.

public void StopReel()
public void StopSpin()
public void AddFreeSpin(int amount)
public void AddBonus(int amount)
public void SetBet(int amount)

LineManager class

public bool EnableNextLine()
public bool DisableCurrentLine()
public void EnableAllLines()
public void DisableAllLines()

SlotEffect class

public Sequence IlluminateLines(float duration = 2f, int count = 10)

Illuminates line icons for the given duration. (as seen in the intro animation)

For more methods and coding help, please refer to the Advanced Customization section and commentations written in source codes.

Game Info

<CustomSlot> has a publicly accessible property named gameInfo where you can get basic game information like game rounds completed, total balance, number of free spins left etc etc.

You can use those information for your game or if you don't require those information you can also completely ignore them.

Please refer to SlotInfo.cs to see what information the game info stores.

Using SymbolGen

SymbolGen generates a random set of symbols and simulate a game to display an analysis.

To start Symbolgen, click "Open SymbolGen Window" on CS inspector or select "Window>Custom Slots>SymbolGen" On Unity's menu bar.

Generate new Loadout

> Clicking the button, SymbolGen will generate a new loadout for your slot and shows an analysis.

Try / Simulate a Game

> Clicking the button, SymbolGen will use the current symbol loadout and simulate a game by iterating certain numbers of spin(round) specified in the setting.

A game is simulated under an assumption that a player will bet all lines. The cost of each line is referenced from ones specified in slot modes. Each slot mode's configuration is reflected upon entering free spin mode, bonus mode etc.

Here we will explain each section of the summary shown in the screenshot.

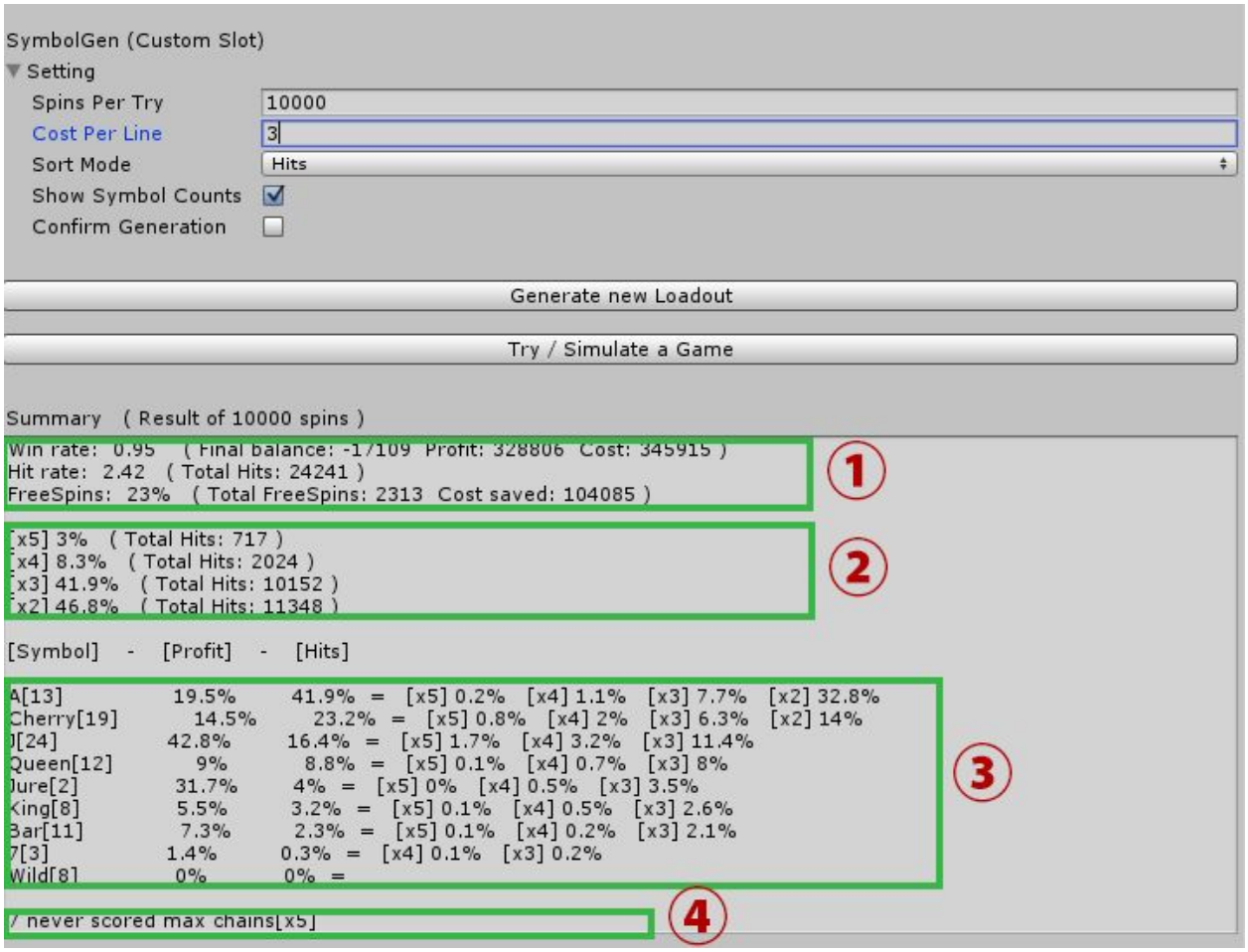
1. The section contains key informations for your slot setup.

> Win rate shows how much a player earned during the simulated game and is calculated by total balance / cost. If the number is less than 1, the player has debt.
>
> Hit rate shows hits per spin a player has scored.
>
> FreeSpins shows a percentage of total spins free spins account for.
> Bonuses shows a percentage of total spins bonus spins account for.

2. This section shows percentages and numbers of each chain(match) that was made during a simulated game. [x5] means 5-in-a-row and [x4] means 4-in-a-row and so on.

3. Here you will get statistics of each symbol,
Profit and Hits show a percentage of total profit/hits the symbol accounts for.

4. Warnings are shown here if there is any.



* The SymbolGen in the screenshot might be an old one as SymbolGen is constantly upgraded/fined tuned in our project.

Using PayTableGen

PayTableGen is a simple pay table generator that will instantiate and setup <PayTableItem> gameobjects according to your slot's symbol setup. The template <PayTableItem> gameobject to be used by the generator is located under SkinManager.

To generate a pay table, specify *TargetParent* parameter on the inspector and click "Generate Pay Table" button. The target parent transform should have a <LayoutGroup> (such as <GridLayoutGroup>) component as the generator does not layout the items. Note that old items the target parent has will be destroyed and replaced.

Please take a look at Elona Slot in Demo scene to see how it works.

Activation, Deactivation

By default, CS will be activated on Awake() but you may also turn off *Activate On Awake* option in Config section and manually call Activate() whenever you like. The first time CS is activated, CS will start an initialization process but if you are manually activating CS, you can call Initialize() in advance which might help controlling start-up lag if you are experiencing any.

Calling Deactivate() will start an Out-Transition effect (if not disabled) and after the effect, CS's gameobject will be deactivated. Optionally, use Deactivate(true) to destroy the gameobject instead of deactivating.

Debug Keys

Within the Debug foldout on CS inspector, you can find several debug options that would come handy when testing your setup. Debug settings include a boolean parameter called "Use Debug Keys" which when enabled provides you some advanced debug commands. The keys and actions are as follows.

* Please note that actions assigned to debug keys may change in future updates.

F1-F5	Shows a symbol hit effec(F2 = 2-in-a-row, F3 = 3-in-a-row and so on)
F6	Shows a line icon illuminating effect.
F11	Gives free spins.
F12	Give bonus spins.

Using Animation

(The feature is implemented but as of now we are waiting for the release of Unity Version 5.4 so there will be no unnecessary patching)

CS version 1.2 implemented a support for sprite animation using UnityEngine.Animation and Mecanim. As of now, you can only animate a symbol by swapping Image's sprite but feel free to ask us for more feature should your project need.

To use Animation for a symbol, you will need a GameObject with Animator component(which should have Controller reference set to the Animation Controller the symbol should be using).

Open up Inspector of the symbol you want to animate(under Symbol Manager's hierarchy) then simply drag&drop the Animator GameObject to the animator parameter the symbol has.

Now when CS starts, the symbol should play the Entry Animation specified in its Animation Controller. As this is more like an addon-feature, CS doesn't send any animator trigger by default but you can easily send triggers your symbols would need by using callbacks.

A demo scene is included which shows how to use Animation on symbols and how to send triggers on a line hit. As a request, It will also show how to swap a Symbol on a hit with Animation.

4.Advanced Customization

Making Derived Classes

If you are familiar with C# programming, you can further customize your slot by creating new classes that derive from CS classes.

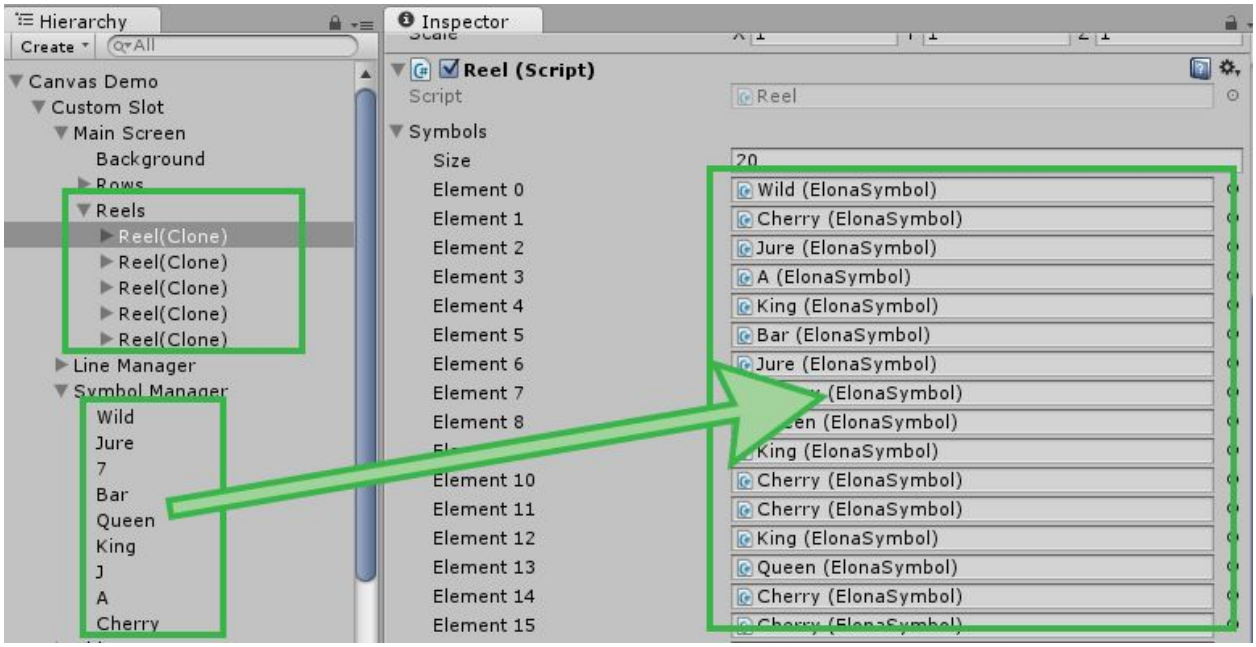
Elos(Elona Slot) in the Demo scene has an example ElosSymbol class that derived from Symbol which enables symbols to talk. ElosUI is derived from <BaseSlotGameUI> class which provides basic UI functions a slot game would need.

It is not necessary at all to derive your UI class from <BaseSlotGameUI> but it can be useful if you are creating a slot game..

Manually Placing Symbols

To manually setup symbol loadout for your slot, first select a reel in your CS's hierarchy to bring up inspector. Then drag a symbol from SymbolManager and drop it to the *Symbols* list.

It is not necessary to do this but to update sprite images in a scene view, click "Refresh Reel" button on the inspector after editing symbol loadout.



Modifying Sequences

Some <UnityEvent> like <EventInfo> and <HitInfo> has a reference to <DOTween.Sequence>.. A *sequence* contains a series of <DOTween.Tweener> objects to be played right after callback is completed. Sequences can be modified to extend/alter their effects(by calling methods such as Join(), Append(), Insert()). For more information on how to control sequences, please refer to DOTween documentation (http://dotween.demigiant.com/documentation.php).

Queueing Events

CS has a simple event queueing system to make it easy to manage events.

When events are queued, CS will stop its progress and plays the events one by one until the event queue becomes empty.

An event usually has a <DOTween.Sequence> which will be played at the same time the event starts. When the sequence is complete, the event also will be complete.

A good example would be the intro animation in Demo. While the animation(sequence) is played CS halts its progress ignoring all the player inputs and moves to next state when the event is complete.

An event can be added at any time by calling AddEvent() method <CustomSlot> has. AddEvent has several overloaded method to make it convenient. Some examples are as follows.

AddEvent(1.2f)

Add an event that makes CS halts for 1.2 seconds when dequeued.

AddEvent(<DOTween.Tween> or <DOTween.Sequence>)

Add an event that plays the tween or sequence when dequeued.

For more information, please refer to CustomSlot.cs 's commentation..

Slot State and Events Flow Chart

Here you can find out when and in what order events and callbacks are processed.

Slot State and events/callbacks

NotStarted	StartRound onSlotModeChange
Idle	onRoundStart StartSpin
SpinStarting	onReelStart
Spinning	
SpinStopping	onReelStop onRoundInterval

Result	Effect(Symbol Hit) // onProcessHit
	onRoundComplete

At any timing

When calling Activate()	Effect(In-Transition) Effect(Intro Animation) // onActivated
When calling Deactivate()	Effect(Out-Transition) onDeactivated
When a line is enabled/disabled	Effect(Switch Line) // onLineSwitch

* Event Callback

// in the tables means the event and the callback are processed at the same time. So if you want to execute SomeMethod() at the beginning of the event, just call SomeMethod() and if you want it to be executed after the event, use <CustomSlot>.AddEvent(SomeMethod) instead.

Making Bonus Game

You can use built-in bonus slot mode to represent a bonus game but if you rather want to implement an original bonus game, the source code of Elos(Demo) shows how We will use it as a tutorial example here.

(In Elos.cs)

First, we declare a variable named *bonusGame* which is a reference to <ElosBonusGame>. Also declaring a variable of CustomSlot type so we can interact with our slot.

```
public ElosBonusGame bonusGame;
public CustomSlot slot;
```

Then we make a callback method and subscribe to CS's onProcessHit event on inspector. Now every time a player scores a hit, the event will invoke the below method and passes a detailed information of the hit as an arugrument.

```
public void OnProcessHit(HitInfo info) {
        if (info.hitSymbol.payType == Symbol.PayType.Custom)
                slot.AddEvent(new SlotEvent(bonusGame.Activate));
 }
```

In the above code, the hit symbol's pay type is compared to determine if we should start a bonus game. On the next line, we use <CustomSlot>.AddEvent method explained in the last topic to add a new event to CS's event queue. This event is now queued and will be played after symbol's hit effect animation is complete.

The <SlotEvent> is a simple event class which has System.Action<SlotEvent> type parameter on its constructor. And as the parameter's name suggests, the given method will be invoked when the event is played(activated). The event is then played until Deactivate() method is called.

```
public SlotEvent(Action<SlotEvent> onActivate = null)
```

In this example, bonusGame.Activate() will be the one to be called since we passed the method as an argument.

Take a look at the code of the bonus game(ElosBonusGame.cs).

```
private SlotEvent slotEvent;

 public void Activate(SlotEvent slotEvent) {
        this.slotEvent = slotEvent;
```

The bonus game stores the reference to the event as slotEvent. Since your slot's progress halts until the event is complete, now it's up to the bonus game to show whatever it must show and then call slotEvent.Deactivate() when it's done to complete the event.

Saving and applying Symbol Map

SymbolManager class has a method to get a current symbol loadout from your reels as <SymbolMap> object and an another method to apply the mapping to your slot.

public SymbolMap GetSymbolMap()

public void ApplySymbolMap(SymbolMap map, List<SymbolSwapper> swaps = null)

When given a list of <SymbolSwapper> objects, CS will swap symbols on the newly applied mapping according to the list.

Extra Callbacks

There are a few callback events hidden on inspector by default. They are publicly accessible but to show them on inspector you have to enable Debug mode (see http://docs.unity3d.com/Manual/InspectorOptions.html). Those events are as follows.

UnityEvent onSlotStateChange
Fired when CS's state changes.

UnityEvent onNewSymbolAppear
Fired when a symbol holder's symbol changes while spinning.

UnityEvent onRoundInterval

Fired after all the reels stop and before CS's hit check begins.

UnityEvent onAddBalance

Fired when gameInfo updates balance information.

Manipulating Symbols/Reels

The newly introduced feature on CS 1.3 lets you manipulate symbols and reels on your slot. To use the feature, you would need to call SetManipulation method(CustomSlot class) any time after a round starts and before the reels stop.

SetManipulation method has a few overload methos which are:

public void SetManipulation(int rowOffset, string symbolId)
public void SetManipulation(int rowOffset, Symbol symbol)
public void SetManipulation(int rowOffset, params Symbol[] symbols)
public void SetManipulation(int rowOffset, params int[] symbolIndexes)

The first parameter(rowOffset) is an offset from the top visible row. When specified 0 the symbols will land on the top visible row, 1 for the 2nd row and etc.... Specify symbol(s) you want to land in the second parameter.

An example to manipulate reels to make "Cherry" land on the center row on 3x3 slot would be as follows.

```
public override void OnRoundStart() {
        base.OnRoundStart();
        slot.SetManipulation(1, "Cherry");
}
```

(More coming)

5.Tutorial - Activating CS as a mini-game

There are several ways of setting up and running your CS but in this brief tutorial we will make a new CS prefab and instantiate it by using Resources.Load() method.

Preparing Your Slot

1. First, make a new scene and a new canvas, put the default CS prefab(CustomSlots/Prefab/CustomSlot) on the canvas.

2. Configure and rename the CS as you like(In this tutorial we name it TestSlot). Drag&Drop the CS gameobject from the canvas to Resources folder in your Project Window to make a new prefab. Delete the CS gameobject in the canvas.

Instantiating Your Slot

Simply attach the below Monobehaviour to the Canvas you created. When hitting Play, it will load and activate your CS then deactivate it once a round is complete. There isn't much to explain in the code but if you like to reuse the CS instance rather than destroying it, you can call Deactivate() without passing true value and then call cs.Activate() later.

```
using CSFramework;
using UnityEngine;

public class MyGame : MonoBehaviour {
    private CustomSlot cs;
    private void Start() {
            cs = Instantiate<CustomSlot>(Resources.Load<CustomSlot>("TestSlot"));
            cs.transform.SetParent(GetComponent<Canvas>().transform, false);
            cs.callbacks.onProcessHit.AddListener(OnProcessHit);
            cs.callbacks.onRoundComplete.AddListener(OnRoundComplete);
    }
    public void OnProcessHit(HitInfo info) { Debug.Log(info.hitSymbol.name); }
```

```
    public void OnRoundComplete() { cs.Deactivate(true); }
}
```

6.Support

We are usually available on Unity forum and on the CS official thread there.
Please do not hesitate to ask questions or report bugs.

Credits

Lafrontier

- Nz (Artist)
- noa (Developer)
- MT (Artist)

Demigiant (DOTween)

Colorful Vacation(Music)
http://dova-s.jp/_contents/author/profile072.html

7Soul1 (Bonus Game Icons)
420 -Pixel Art- Icons for RPG

Little Robot Sound Factory

Licences

* Audio effects in the demo scene mostly have CC3.0 Attribution or less strict licences. You may use them in your commercial project but you must also give an appropriate credit.

* Arts included in the asset are free to use for commercial projects but redistribution of the arts is prohibited.