Cut The Wood

Mamadou Cisse

Thank you for your purchase!

Table Of Contents

IMPORTANT

This package works with the plugin Infinity Engine written with C# 7 syntax, this means that you must active this option in Unity Editor. If you don't do this, Unity will not compile your project. Go to this link to enable C# 7 syntax.

How to use Infinity Engine?

Infinity Engine is a set of plugins that extends Unity. Each plugin is located at :

"Assets/Infinity Engine/Frameworks/Name of the feature"

You can find a documentation for each of these plugins here

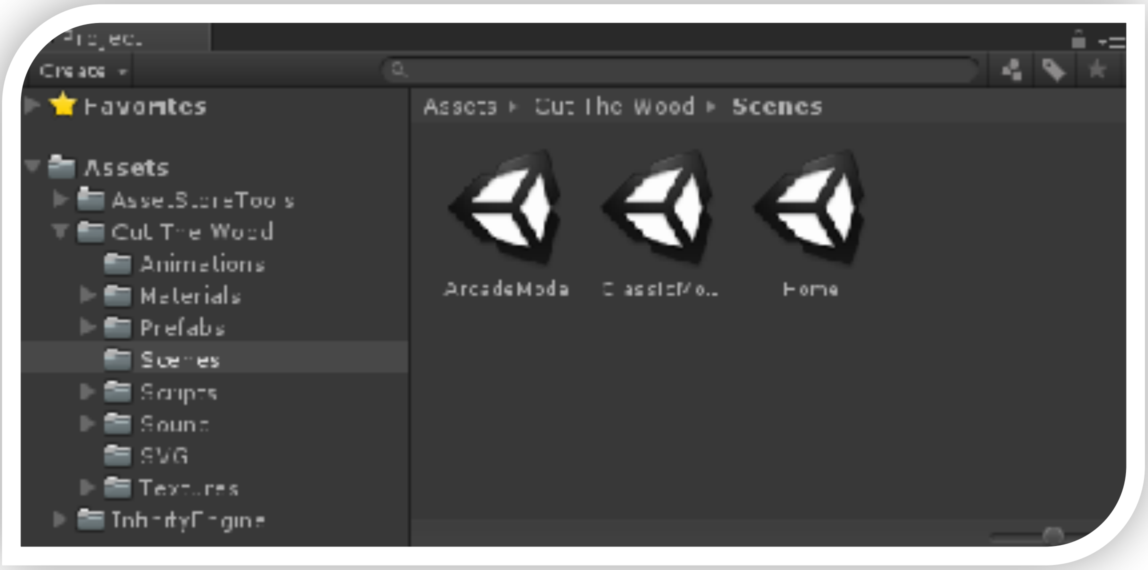
–

How to test this package?

- **On Desktop:**

Open Unity editor and import the package that you downloaded on Unity Asset Store.

Open the scene “Home” placed at “Assets/Cut The Wood/Scenes/”.

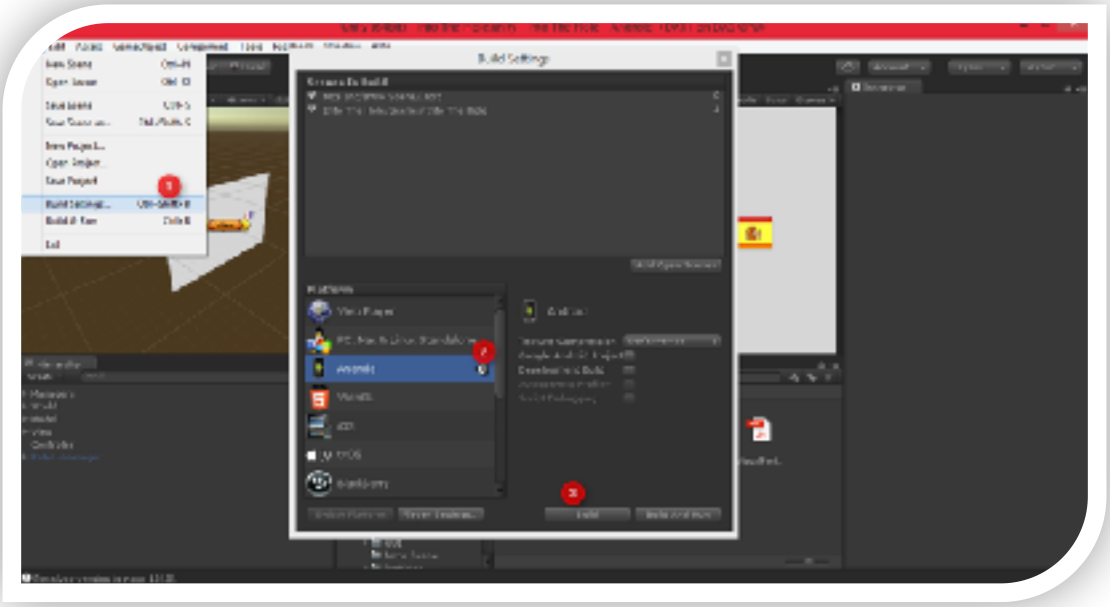


Click on Play button to start the game.

- **On Android Device:**

Go to the tab “File/Build Settings…” and switch the current build platform to Android.

Click on the button build to generate the apk file of the game.



What's in the Package?

Infinity Engine.dll: The dll library which contains the main code of Infiniy Engine API

Infinity Engine Folder: Placed at the root of the project, this folder contains all plugins of Infinity Engine API.

Cut The Wood Folder: Placed at the root of the project, this folder contains the main code of Cut The Wood game.

Model.cs : Model component of MVC Design pattern.
This class represents the data of a game mode. It inherits from IGameModel. The class is inherited for each game mode

View.cs: View component of MVC Design pattern. Represents the GUI of a game mode.

Controller.cs: Controller component of MVC Design pattern.
This script manages the game events (game starts, game ends) and control the model and the view. When the player starts a

game mode, the event onGameStartCallback of this script is invoked and when the player loses, the event onGameStartCallback is invoked.

Blade.cs: Base class of all blades.

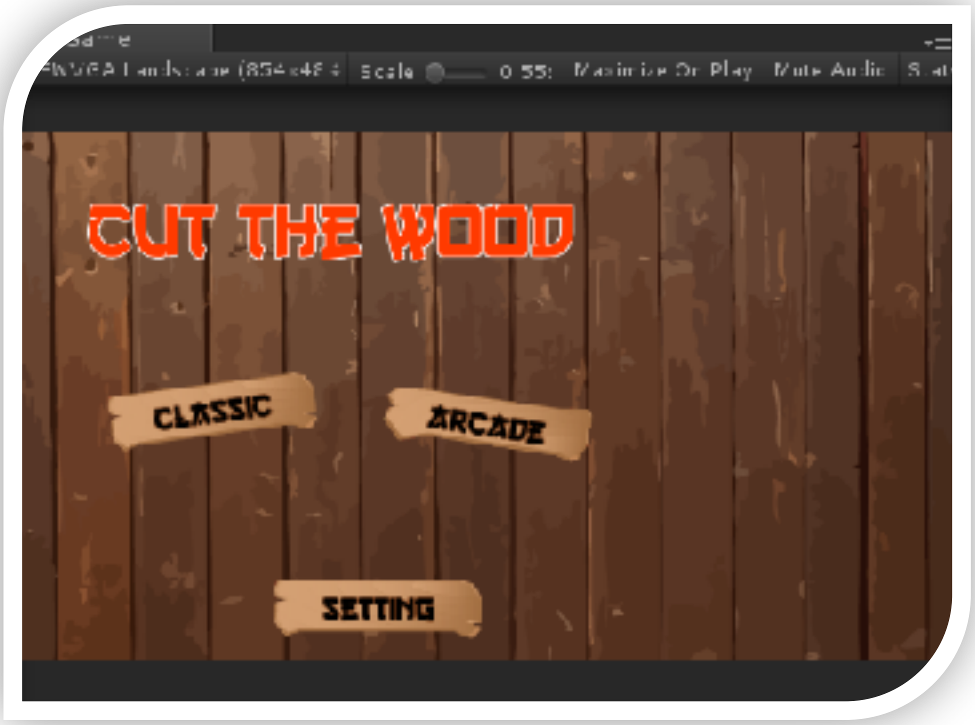
GameItem.cs: Base class of all game items.

Getting Started

A - Gameplay

Game Modes

Cut The Wood is a slicing game like fruit ninja app. The game contains 2 modes:



In Classic mode, the goal is to cut the maximum wood that you can, but you must avoid the bombs. You lose when you hit a bomb or miss 3 woods. Each time you reach 100 points, you gain one life.

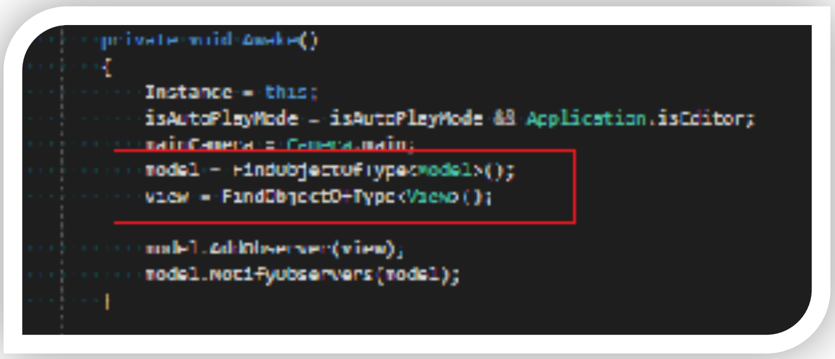
In Arcade mode, the goal is to cut the maximum wood in a given time.

MVC Design Pattern

All these game modes work in the same way thanks to MVC design pattern. MVC is an architectural pattern used to separate an application into tree components, the **model**, the **view** and the **controller**.
The model is responsible to **manages the data** of the application.
The view is responsible to **display a graphical** representation of the data of the model.
The controller is responsible to **manages the actions** of the user.

For each game mode of the project there is an instance of a component in the scene which inherits from View and for Model.

The controller is the same for all game modes. When you start a scene, the controller search in the current scene the instance of the classes Model and View in the scene inside of the function 'Awake',



```
private void Awake()
{
    Instance = this;
    isAutoPlayMode = isAutoPlayMode && Application.isEditor;
    mainCamera = Camera.main;
    model = FindObjectOfType<Model>();
    view = FindObjectOfType<View>();

    model.AddObserver(view);
    model.NotifyObservers(model);
}
```

As you see in the picture above, the controller does not know the type of the instance of the model and view classes, it uses an abstract reference to these classes. This the reason why the controller is the same for all game mode.
This system allows to:

- Reduce dependency between classes
- Reuses the same core gameplay for each game modes
- Facilitate the updating of the project

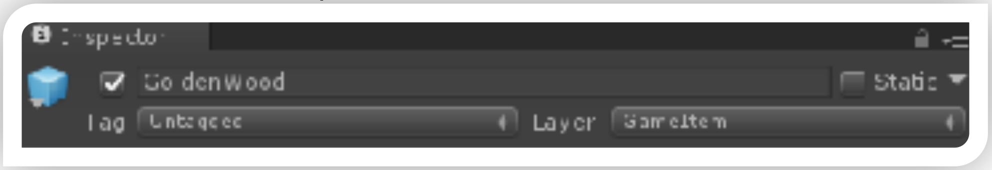
- Add the possibility to change at runtime the model by coupling Decorator and Strategy pattern (You can inherit the class ModelDecorator to add extra behaviors to the model, for example you can create a special item or a special blade that multiply by two the point of the player each time he cut an item or anything else).

For information, I created the game mode ‘Arcade’ in 10 minutes with this system.

Core Gameplay

After getting the view and the model, the controller invokes the methods linked to the delegate onGameStartCallback like (the method OnStart of the view class) and start throwing items after a delay specific to each instance of the model class (1 second for the model ‘ClassicModeModel’ and 3 seconds for the model ‘ArcadeModelModel’).

The controller checks the inputs of the user inside of the function Update and gets all the Game Objects with the layer ‘GameItem’ which intersects the position of the mouse (or finger in mobile device).


Inspector

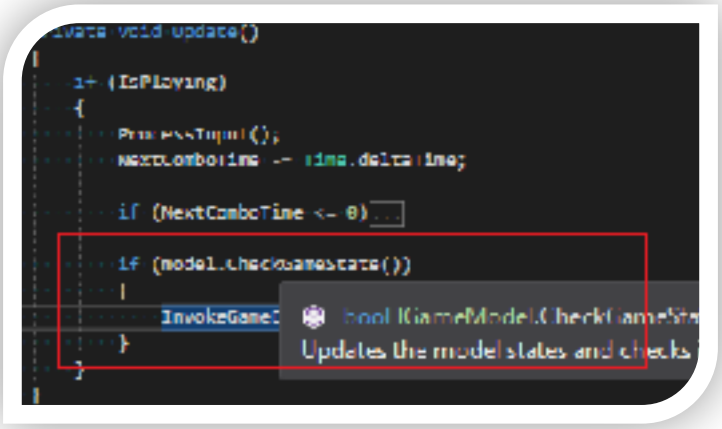
☑ GoldenWood ☐ Static

Tag Untagged Layer GameItem

When the player hit an item, the controller does the following job:

- Call the function DOHitAction of the item.
- Call the function OnHitItem of the blade which it the item (this allow you to create blade with special powers).
- Generates temporary wood fragments at the position of the item.
- Shakes the main camera.

The controller ends the game when the function CheckGameState of the model return true.



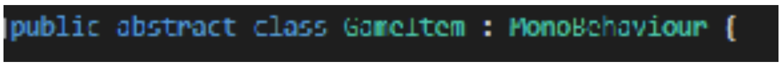
```
ivate void update()
{
    if (IsPlaying)
    {
        ProcessInput();
        NextComboTime -= Time.deltaTime;

        if (NextComboTime <= 0) ...

        if (model.CheckGameState())
        {
            InvokeGame
        }
    }
}
```

bool IGameModel.CheckGameSta
Updates the model states and checks

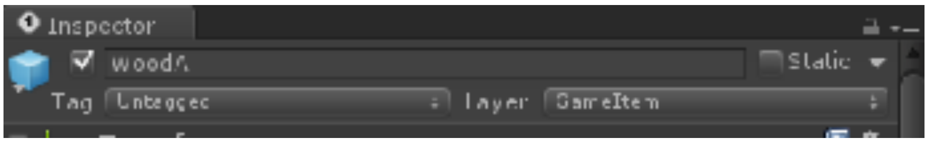
Note: You can play the game in automatic mode by setting the Boolean IsAutoPlayMode of the class Controller to true.

B – How items are instantiated?

All game items have a component which inherits from the class “GameItem” and they have the layer “GameItem”.



```
public abstract class GameItem : MonoBehaviour {
```



Inspector

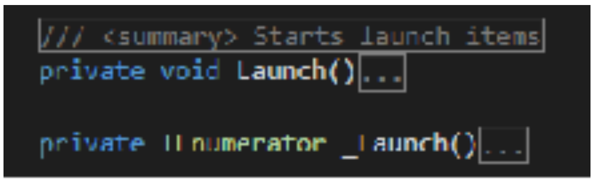
wood

Static

Tag Untagged

Layer GameItem

The items are instantiated by the component Controller thanks to the function “Launch”. The function is call a coroutine that is repeating in loop with a rate specific to the instance of the model.

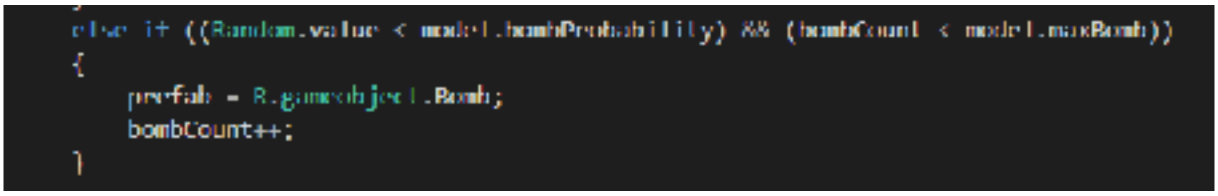


```
/// <summary> Starts launch items
private void Launch()...

private IEnumerator _Launch()...
```

Each time the coroutine restarts, the controller instantiates a number defined of items depending to the value of the properties “MinItemToLaunch” and “MaxItemToLaunch” of the model component.

These values are dynamics, there are modified during the game.

During the generation of the items, the function “Launch” compares the value of a random number in the range [0, 1] to the value of the property “bombProbability” of the model which depends to the score also. If the value of the random number is less than the probability, and the number of generated bomb in the current call of the function is less than the value of “model.maxBomb“, the function generates a bomb. This means that you can increase, decrease or disable dynamically the probability to instantiate a bomb.



```
else if ((Random.value < model.bombProbability) && (bombCount < model.maxBomb))
{
    prefab = R.gameobject.Bomb;
    bombCount++;
}
```

At the instantiation, each item has its own speed and vertical velocity. When the player hit an item, the function “OnTouch” of the item is invoked, this function calls the abstract function “DOTouchAction” which is overridden by each sub-class.

For the optimization of the game, the items are not instantiated normally, they are instantiated thanks to the pool management system of the component “PoolManager“ and when they are touched by the player or that they fall, they are not destroyed, they are deactivated and pushed in the “PoolManager “.

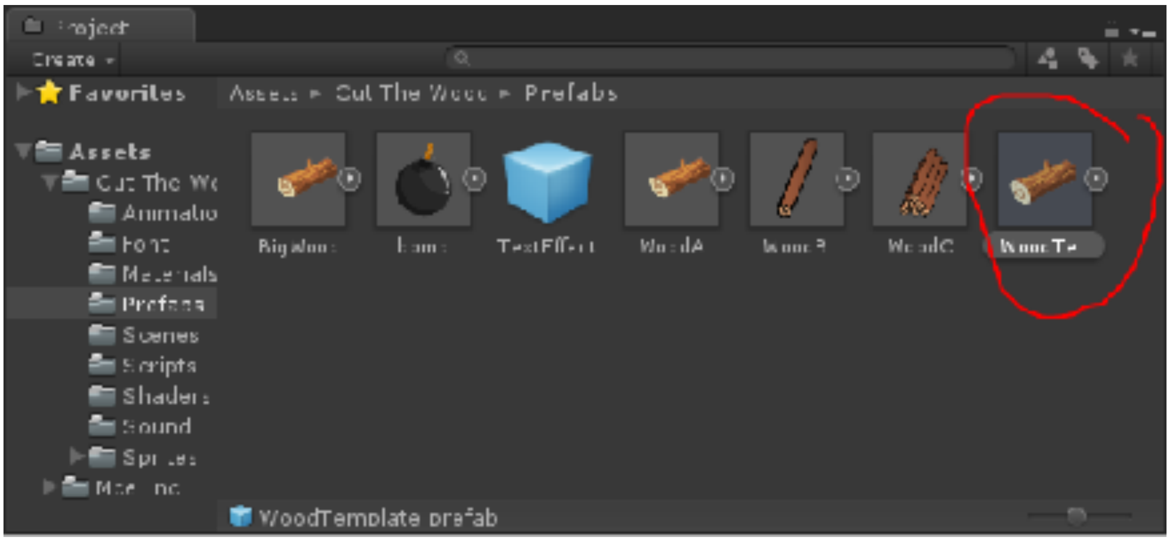
```
var item = PoolManager.Pop(prefab).GetComponent<GameItem>();
```

C – How to customize this package?

Add New Game Item:

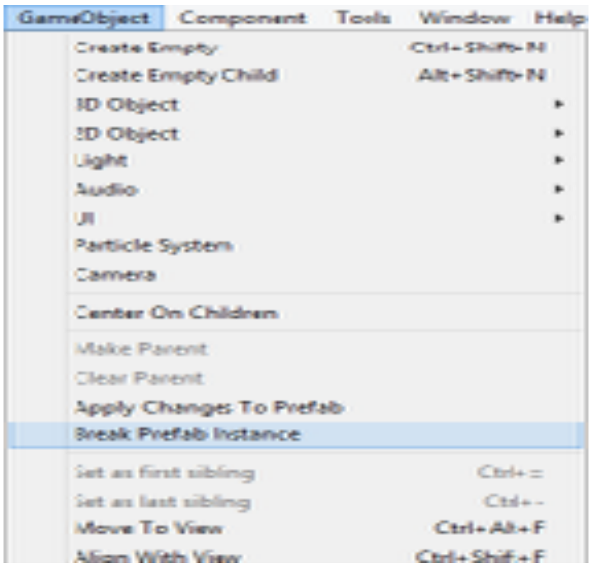
If you want to add new game item by keeping the same “cut animation” that is used by all woods,

- Place the prefab **WoodTemplate** on the scene.

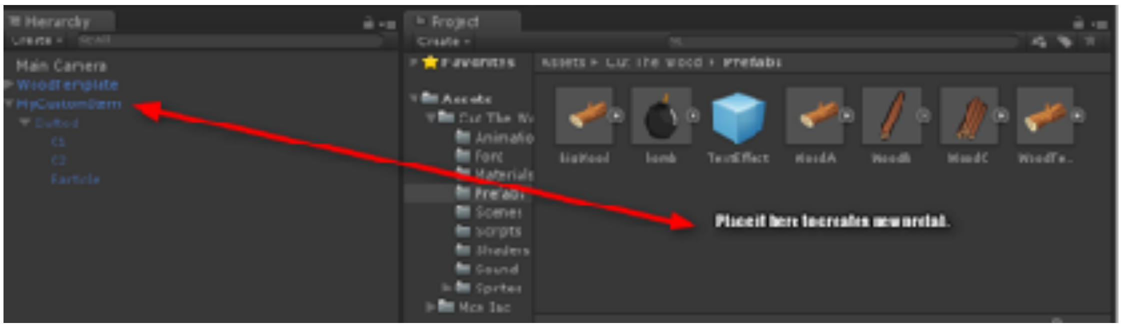


- Break the instance of the prefab on the tab “**GameObject**”

–



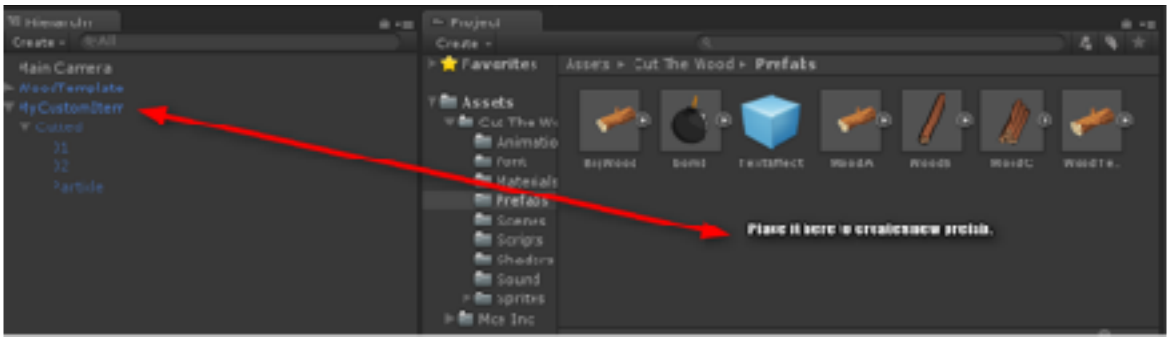
– Duplicate the GameObject in the scene, rename it and create new prefab by dragging the GameObject on Assets folder.



– Delete the GameObject **WoodTemplate**.

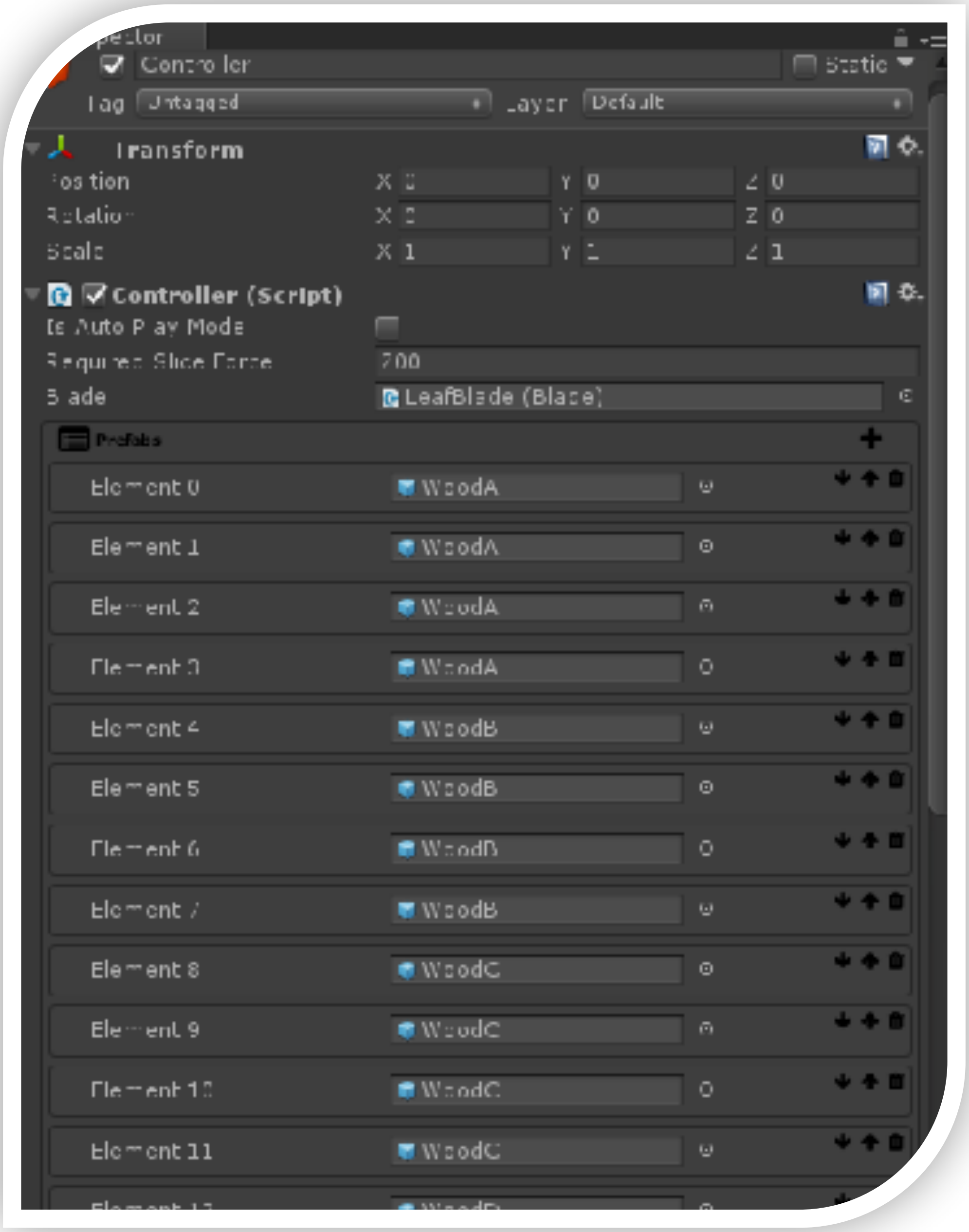
– Change the sprite of the GameObject “**MyCustomItem**” which is the normal state of the item, then the sprite of the GameObjects “**01**” and “**02**” which are respectively the left and right part of the cutted version of the normal sprite. You can also change the particle system of the GameObject “**Particle**”, but you **don't have to delete any GameObject** (except if you know what you are doing) because they are used by the component “GameItem”.

- Add a component which inherits from GameItem to the GameObject.



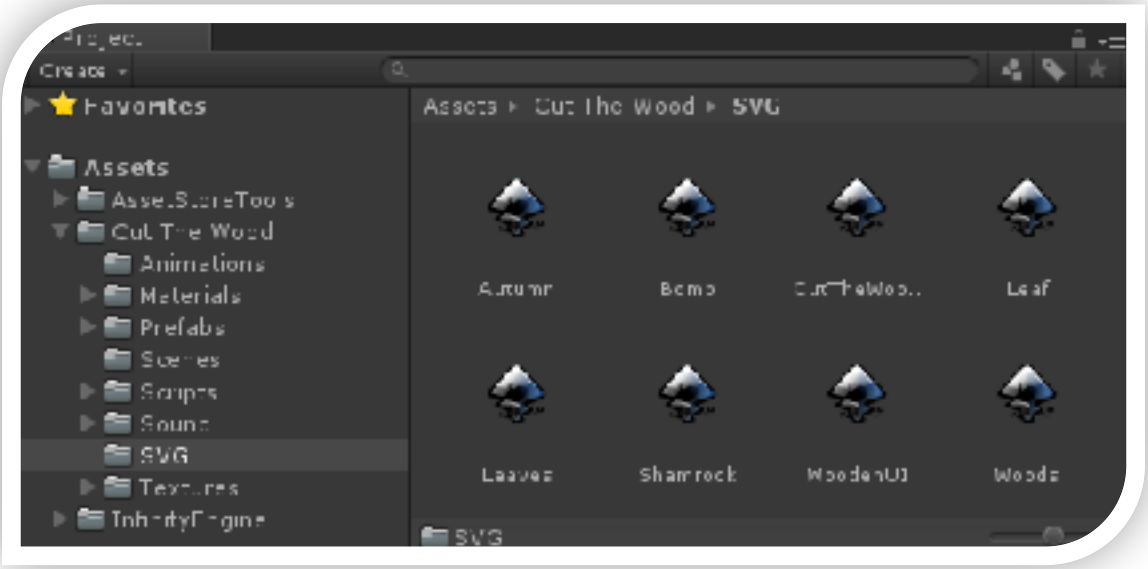
- Save the prefab and updates ISI Resource database by clicking on the keys “**Alt+U**” or by opening the resources editor on the tab “**Tools/Infinity Engine/Resources/Editor**”.

- The last step is to reference the prefab in controller class by changing the array “prefabs”.



```
Inspector
☑ Controller                                   ☐ Static
Tag  Untagged                  Layer  Default

Transform
Position    X 0    Y 0    Z 0
Rotation    X 0    Y 0    Z 0
Scale       X 1    Y 1    Z 1

☑ Controller (Script)
Is Auto Play Mode       ☐
Required Slice Force    700
Blade                   LeafBlade (Blade)

Prefabs
Element 0     WoodA
Element 1     WoodA
Element 2     WoodA
Element 3     WoodA
Element 4     WoodB
Element 5     WoodB
Element 6     WoodB
Element 7     WoodB
Element 8     WoodC
Element 9     WoodC
Element 10    WoodC
Element 11    WoodC
Element 12    WoodD
```

For information, the package is shared with all the original Vector artworks. There are placed at “**Assets/Cut The Wood/SVG**”, you can modify the artworks with the free software **Inkscape** to creates new woods and more..



Add New Blade

- Place the prefab '**OriginalBlade'** located at '**Assets/Cut The Wood/Prefabs/Blades'** in the scene.
- Do the same steps as for item creation and add your custom Blade script to the blade. You can override the virtual function OnHitItem as for the script GoldenBlade to add extra power ups to the blade.

Localize this package

The plugin I[SI Localization](#) is included in this package, you can use it to localize your games easily.

Complements

How The Menu System Works?

The menu system of the project is controlled by the plugin ISI Interpolation. This plugin provides a powerful tool to creates an inspector driven user interfaces without programming any line of code. The entire user interface of the game is created thanks to this plugin.

How The Screen Is Faded Between The Scenes?

There is the script ScreenFader in all the scenes of the game. The script contains two functions FadeIn() called when the scene is closed and FadeOut() called when the scene is opened.

How To Use The Power Of Infinity Engine?

Infinity Engine is very useful tool for unity there is a lot of tools shared with the API, the API is fully documented and easy to use. Infinity Engine is shared with this project as a dll library because there are lots of scripts, using a dll makes it easier to use the API with other projects. Never delete any files in the Infinity Engine folder (unless you know what you are doing). If you want to get the source code of the API, you can send an email at mciissee@gmail.com.

Final Words

Thanks for your purchase, please rate this asset if you like it and consult my asset store page to see more assets.

If you have any question, feel free contact me at mciissee@gmail.com

Good luck for your projects.