ARCADE IDLE COMPONETS

Documentation

Table of Contents

Baranovsky Studio

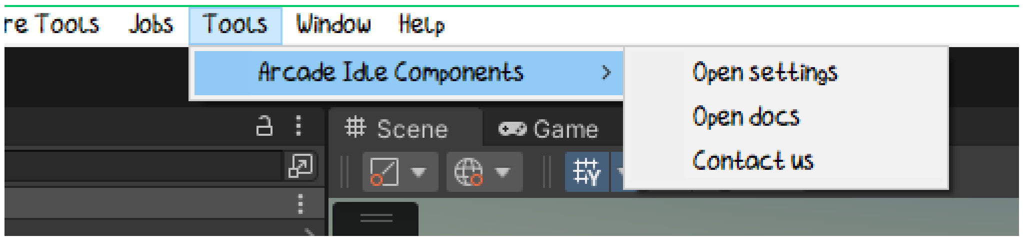
Introducing

Arcade Idle Components - a package containing a set of ready-to-use components: a controller for the player, backpacks, triggers, pointer arrow, and different components for the UI; instead of creating these components from scratch, you can use ready-made ones only configuring as you need.

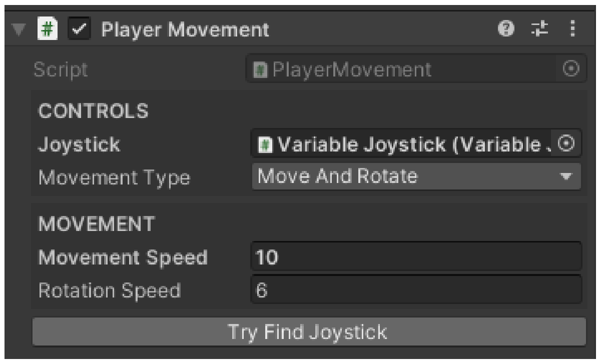
To help you understand all the components and how to use them, we have created a mini game and preview the scenes for all the components. You can launch a minigame by opening the **Example Game/Scenes** folder and running the **Main** scene. To preview the scenes, open the **Preview Scenes/Scenes** folder and run the scenes you are interested in.

Some tips:

- Please use Unity 2021.3.18 or higher for better expirience.
- Settings and online documentation can be easily opened through menu



- Some components, if not configured correctly, will send warnings to the console, you can disable them through settings.
- We advise you to view the preview scenes to understand how the components work. Use the resolution **1920x1080** for preview scenes and **1080x1920** for example game to display the UI correctly.
- There are several systems that are necessary for some components to work, such as **Save System** and **Resources System**. Without them **Unlockable Item**, **Upgradable Item** and **Buyable Item** will not work.
- Some scripts have buttons to find necessary links more quickly, such as this one.



Player Movement

Script: PlayerMovement

CONTROLS

Joystick: Variable Joystick (Variable

Movement Type: Move And Rotate

MOVEMENT

Movement Speed: 10

Rotation Speed: 6

Try Find Joystick

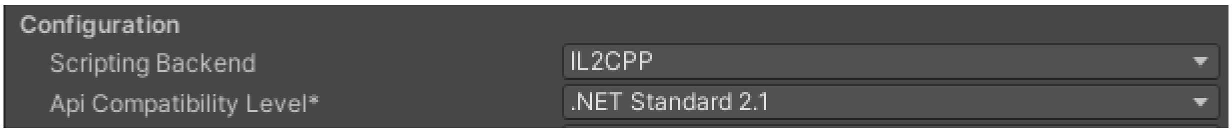
This documentation is meant for you to understand how to work with the asset. We have tried to make each component flexible to customization. However, if you don't understand something, please refer to the online documentation where each component is described. We made it online so you could ask questions, and we update it.

>>ONLINE DOCUMENTATION<<

Import of necessary assets

For correct performance of our asset it’s necessary to import two assets before starting work: Naughty Attributes and Joystick Pack.

To make Naughty Attributes work, open **Player Settings** and change the **Api Compatibility Level** to **.NET Standart 2.1**, as in the screenshot.


Configuration
Scripting Backend: IL2CPP
Api Compatibility Level*: .NET Standard 2.1

Do not forget to import them!

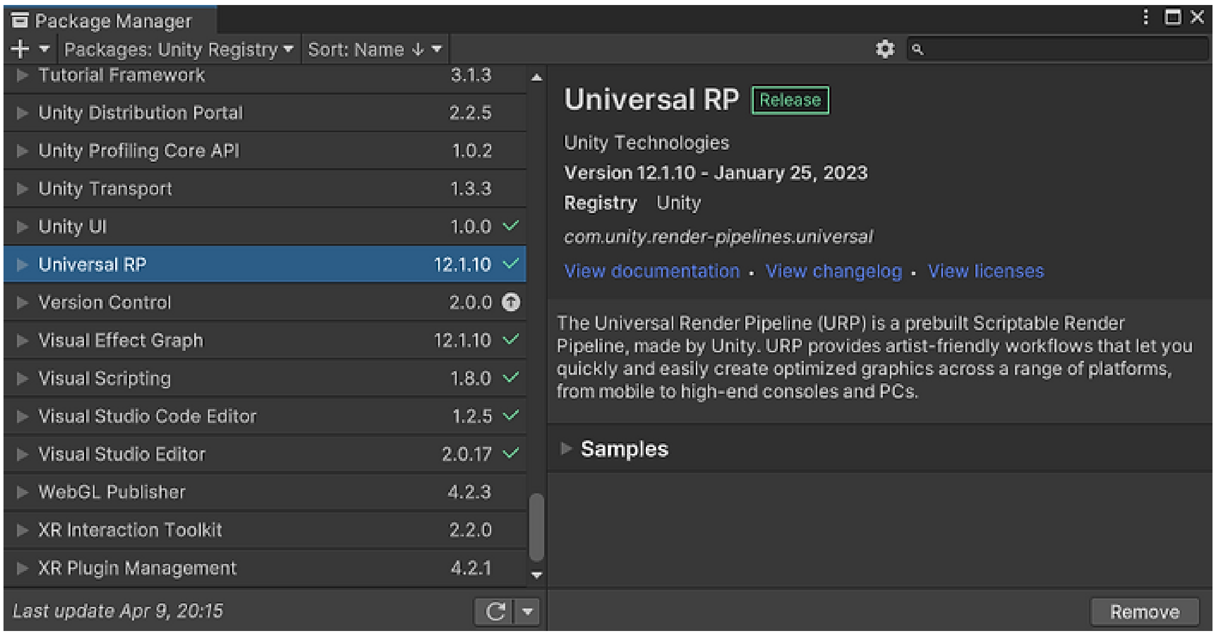
Folders

- **Example Game** – a small game created to demonstrate the use of components.
- **Prefabs** – prefabs of all components already ready for use.
- **Preview Scenes** – scenes showing how the components work.
- **Resources** – here are the settings file.
- **Sprites** – here are all UI sprites.

How to change renderer pipeline?

To setup Universal Renderer Pipeline:

1. Import **Universal RP** from **Package Manager**.



Package Manager

Packages: Unity Registry   Sort: Name

Tutorial Framework	3.1.3
Unity Distribution Portal	2.2.5
Unity Profiling Core API	1.0.2
Unity Transport	1.3.3
Unity UI	1.0.0 ✓
Universal RP	12.1.10 ✓
Version Control	2.0.0
Visual Effect Graph	12.1.10 ✓
Visual Scripting	1.8.0 ✓
Visual Studio Code Editor	1.2.5 ✓
Visual Studio Editor	2.0.17 ✓
WebGL Publisher	4.2.3
XR Interaction Toolkit	2.2.0
XR Plugin Management	4.2.1

Last update Apr 9, 20:15

Universal RP Release

Unity Technologies

Version 12.1.10 - January 25, 2023

Registry Unity

com.unity.render-pipelines.universal

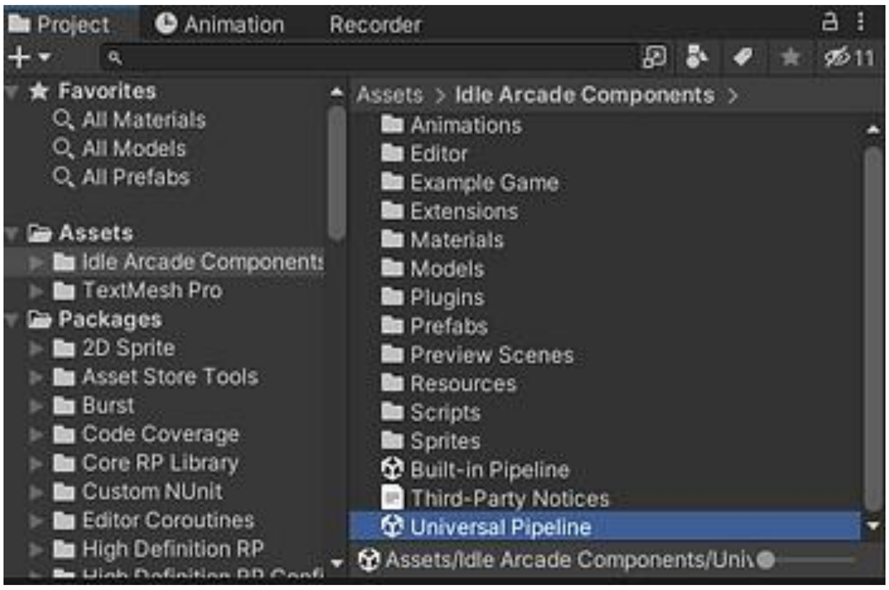
View documentation · View changelog · View licenses

The Universal Render Pipeline (URP) is a prebuilt Scriptable Render Pipeline, made by Unity. URP provides artist-friendly workflows that let you quickly and easily create optimized graphics across a range of platforms, from mobile to high-end consoles and PCs.

Samples

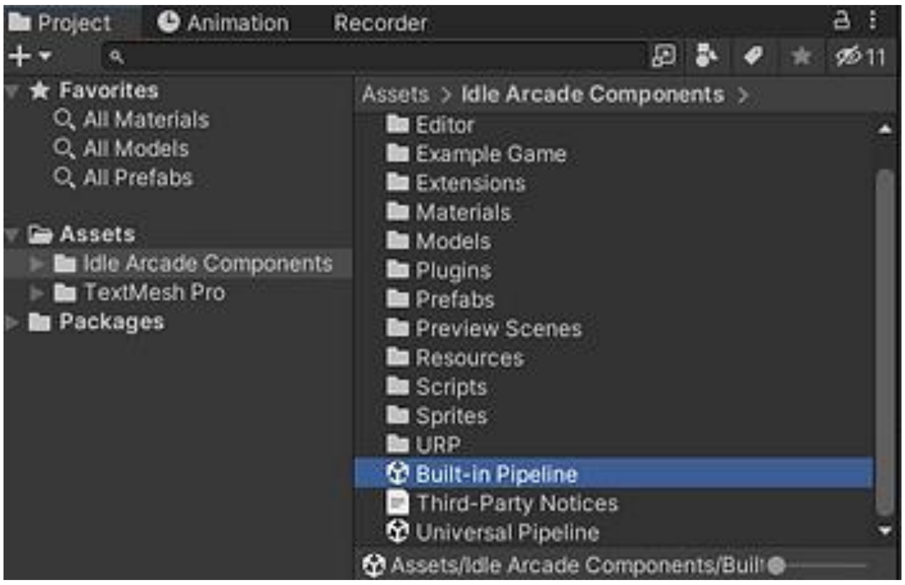
Remove

2. Import **Universal Pipeline**.**unitypackage**.

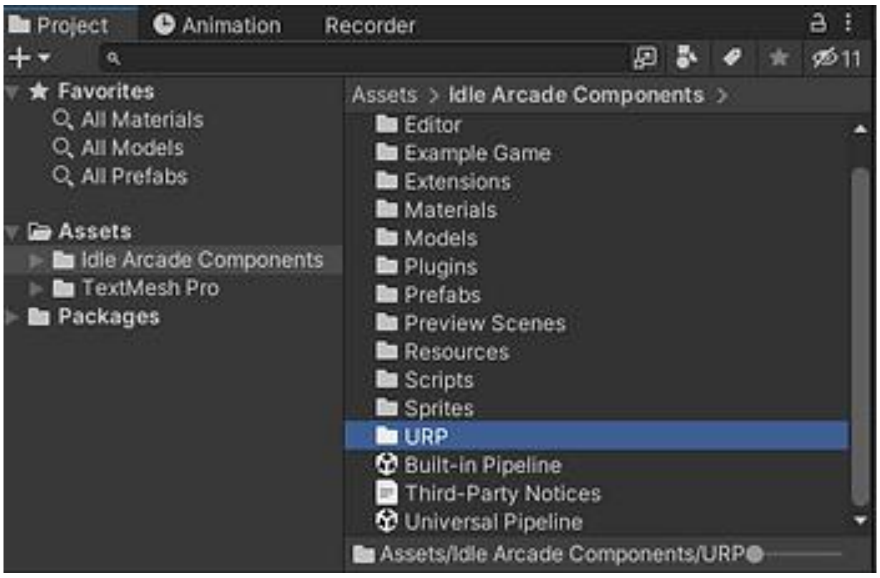


To revert to Built-in Renderer Pipeline:

1. Import **Built-in Pipeline**.**unitypackage**.

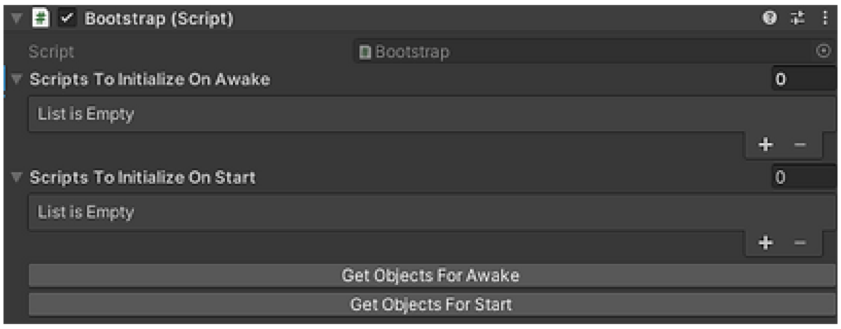


2. Delete **URP** folder.



What is Bootstrap?

Bootstrap - a components that simplifies the initialization of some components. Using it in conjunction with the abstract class **Initializable**, you can flexibly configure which sequence to initialize the systems and components.

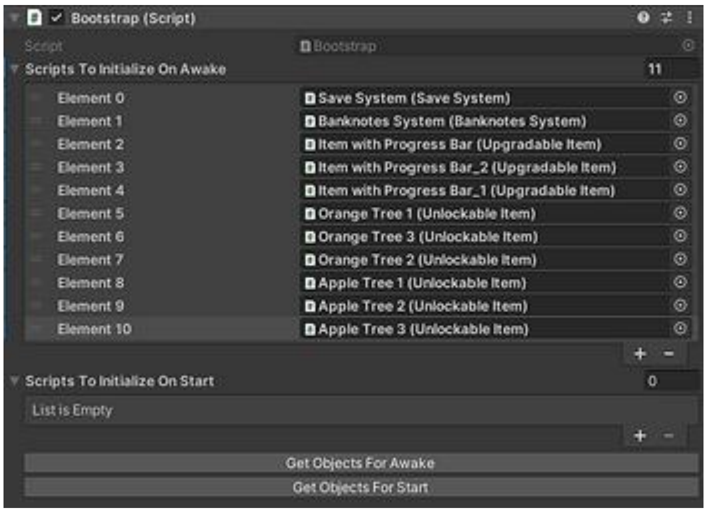


Bootstrap (Script)

Script	Bootstrap
Scripts To Initialize On Awake	0
List is Empty	
Scripts To Initialize On Start	0
List is Empty	

- **Scripts To Initialize On Awake** - list of scripts that will be initialized when calling method awake.
- **Scripts To Initialize On Start** - same, only initializes when calling the start method.
- **Get Objects For Awake\Get Objects For Start** - finds all objects inherited from the Initializable class.

EXAMPLE:



Bootstrap (Script)

Script	Bootstrap
Scripts To Initialize On Awake	11
Element 0	Save System (Save System)
Element 1	Banknotes System (Banknotes System)
Element 2	Item with Progress Bar (Upgradable Item)
Element 3	Item with Progress Bar_2 (Upgradable Item)
Element 4	Item with Progress Bar_1 (Upgradable Item)
Element 5	Orange Tree 1 (Unlockable Item)
Element 6	Orange Tree 3 (Unlockable Item)
Element 7	Orange Tree 2 (Unlockable Item)
Element 8	Apple Tree 1 (Unlockable Item)
Element 9	Apple Tree 2 (Unlockable Item)
Element 10	Apple Tree 3 (Unlockable Item)
Scripts To Initialize On Start	0
List is Empty	

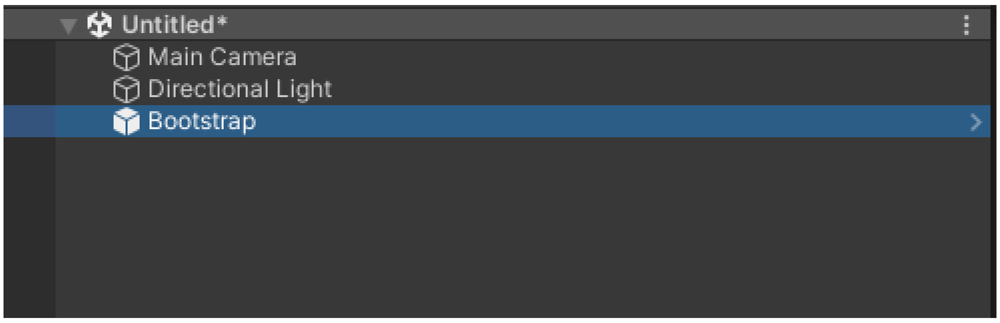
First the systems will initialize, then the components.

How to setup a scene?

There is nothing special about setting the scene, but depending on what components you will use, you may need to add Boostrap and several systems.

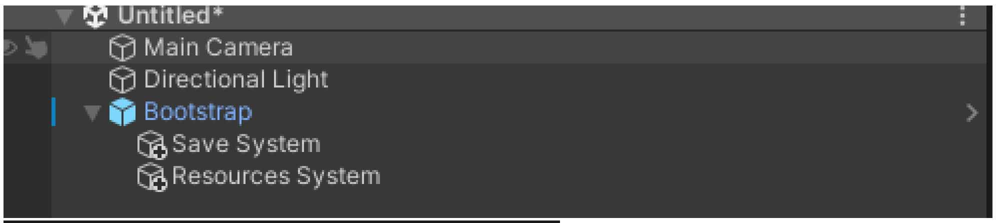
«There are several systems that are necessary for some components to work, such as **Save System** and **Resources System**. Without them **Unlockable Item**, **Upgradable Item** and **Buyable Item** will not work.»

1. Create a new scene and if you need to use any of the above components or systems, add a Bootstrap prefab from the Prefabs folder.

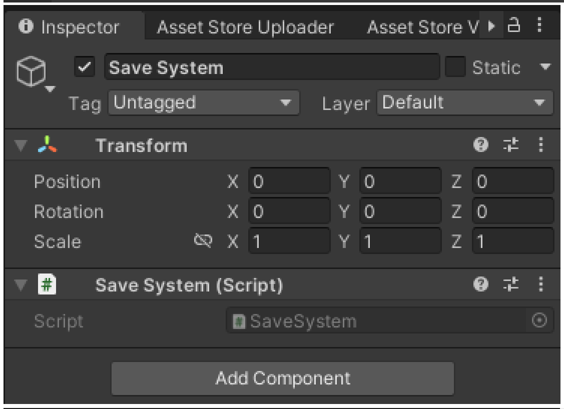


```
Untitled*
  Main Camera
  Directional Light
  Bootstrap
```

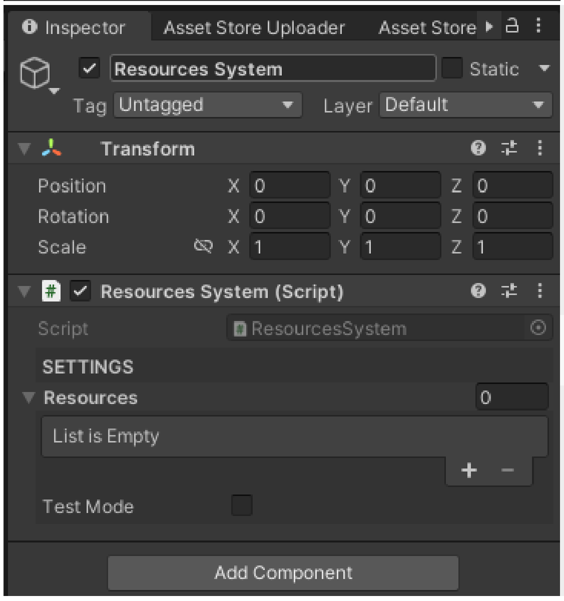
2. Now, create empty game objects and add scripts of the necessary systems.



```
Untitled*
  Main Camera
  Directional Light
  Bootstrap
    Save System
    Resources System
```

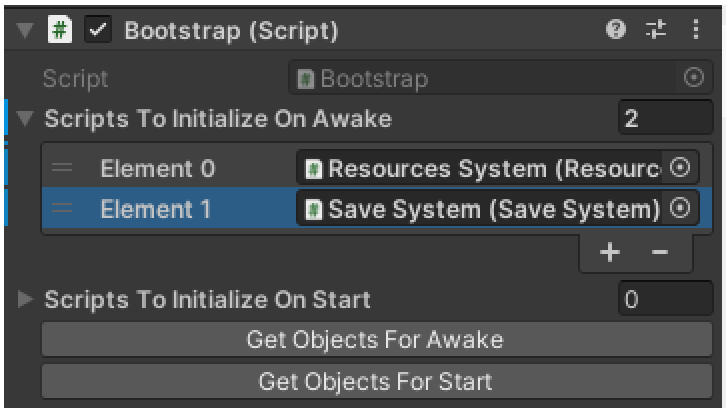


```
Inspector   Asset Store Uploader   Asset Store V
[✓] Save System                       Static
Tag Untagged           Layer Default
Transform
Position   X 0   Y 0   Z 0
Rotation   X 0   Y 0   Z 0
Scale      X 1   Y 1   Z 1
Save System (Script)
Script     SaveSystem
Add Component
```



```
Inspector   Asset Store Uploader   Asset Store
[✓] Resources System                  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
[✓] Resources System (Script)
Script     ResourcesSystem
SETTINGS
Resources                         0
  List is Empty
                              +  -
Test Mode  [ ]
Add Component
```

3. Then select **Bootstrap** and click on **Find Objects For Awake**.



Bootstrap (Script)

Script: Bootstrap

Scripts To Initialize On Awake: 2

Element 0: Resources System (Resourc

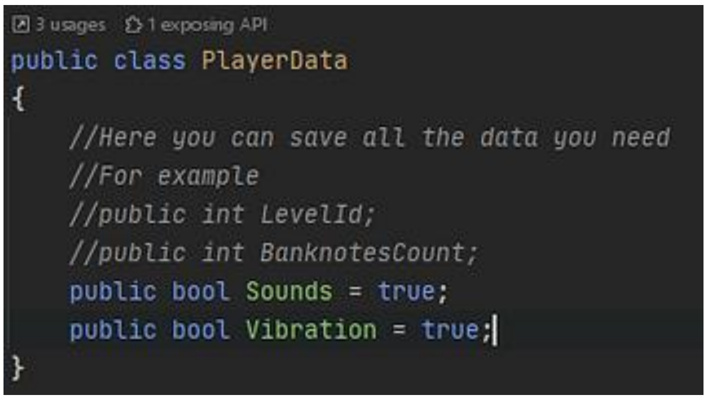
Element 1: Save System (Save System)

Scripts To Initialize On Start: 0

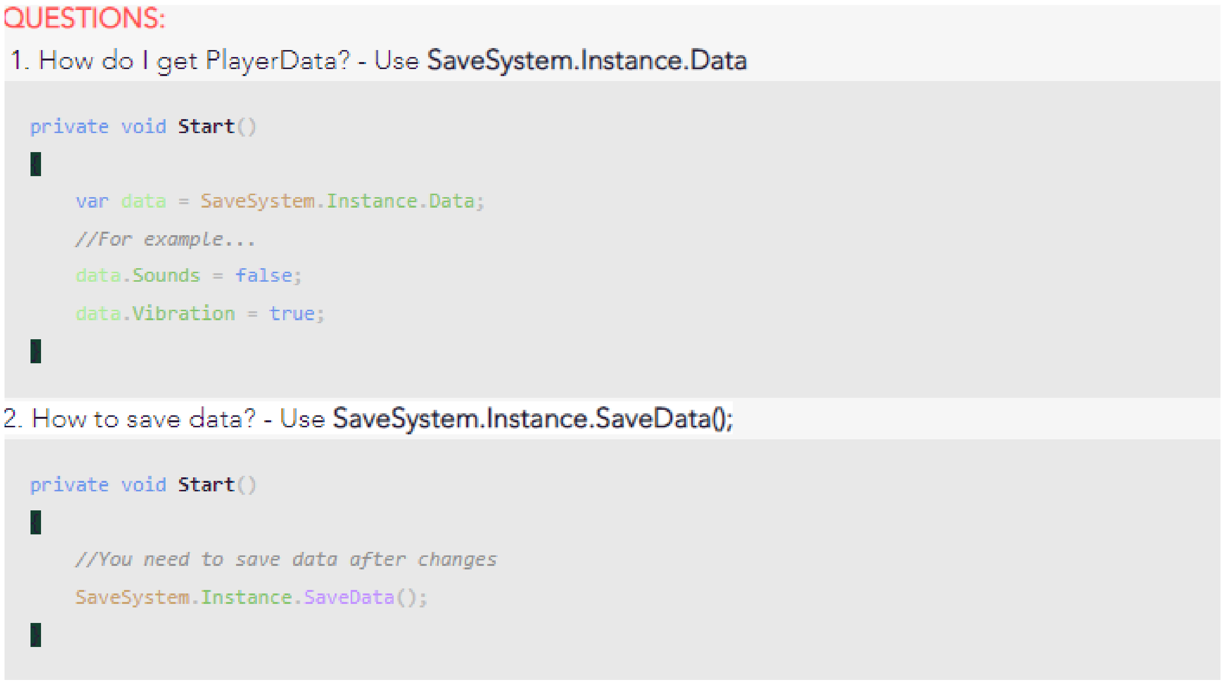
Move the **Save System** to be the first element, that is - load the save data before the other elements awake.

How to use Save System?

Save System - is a simple singleton class that can load and save PlayerData in PlayerPrefs. You can add your variables to this class and change their values during the game.



```
public class PlayerData
{
    //Here you can save all the data you need
    //For example
    //public int LevelId;
    //public int BanknotesCount;
    public bool Sounds = true;
    public bool Vibration = true;
}
```



QUESTIONS:

1. How do I get PlayerData? - Use **SaveSystem.Instance.Data**

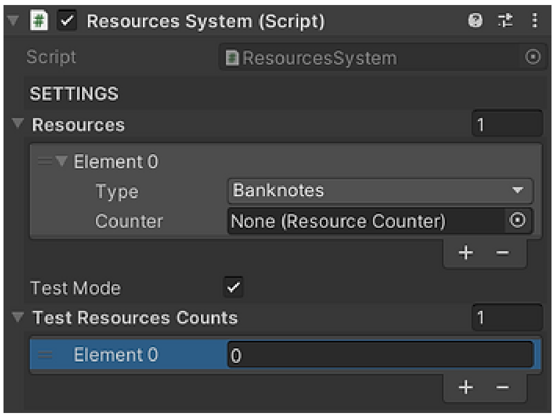
```
private void Start()
{
    var data = SaveSystem.Instance.Data;
    //For example...
    data.Sounds = false;
    data.Vibration = true;
}
```

2. How to save data? - Use **SaveSystem.Instance.SaveData();**

```
private void Start()
{
    //You need to save data after changes
    SaveSystem.Instance.SaveData();
}
```

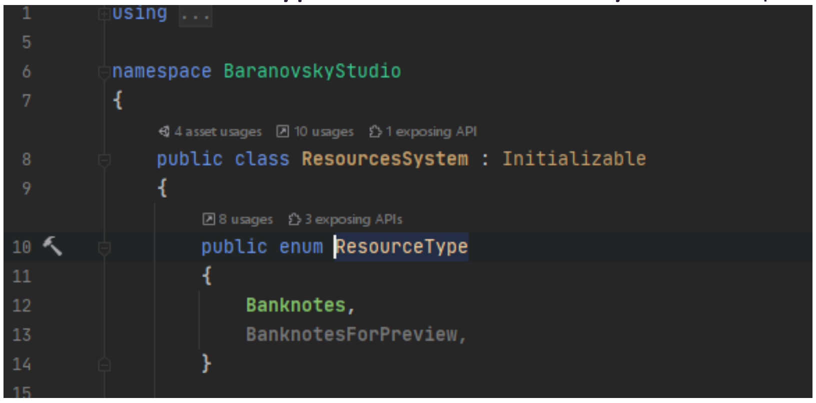
How to use Resources System?

Resources System - a system with which you can perform any operation with unlimited resources. Add, take, pay for purchases and upgrades.


Resources System (Script)
Script: ResourcesSystem
SETTINGS
Resources: 1
Element 0
Type: Banknotes
Counter: None (Resource Counter)
Test Mode: ✓
Test Resources Counts: 1
Element 0: 0

- **Resources** - list of all game resources.
- **Type** - resource type.
- **Counter** - resource counter link.
- **Test Mode** - enables a test mode where you set a certain number of resources at startup.
- **Test Resources Counts** - test resources values.

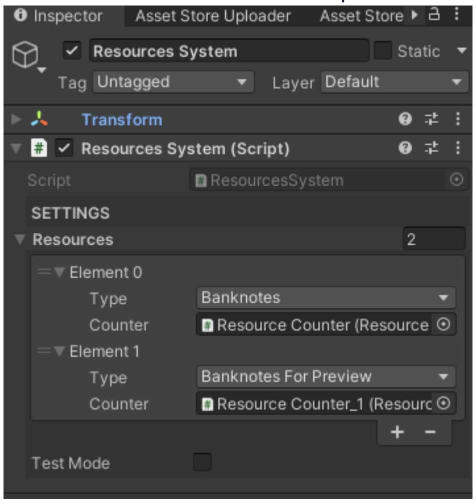
When creating a new game, you need to determine what resources you will have and add them to **Resources Type** in the **Resources System** script.



```
using ...

namespace BaranovskyStudio
{
    public class ResourcesSystem : Initializable
    {
        public enum ResourceType
        {
            Banknotes,
            BanknotesForPreview,
        }
```

Then add them to the resource list and move the links to **Resources Counters**. When you change the amount of resources, the value in **Resources Counter** will be automatically updated. **Resourcer Counter** prefabs is in folder **Prefabs/UI**.


Resources System
Tag: Untagged
Layer: Default
Resources System (Script)
Script: ResourcesSystem
SETTINGS
Resources: 2
Element 0
Type: Banknotes
Counter: Resource Counter (Resource
Element 1
Type: Banknotes For Preview
Counter: Resource Counter_1 (Resourc
Test Mode

QUESTIONS:

1. How to get resource count? - Use GetResourceCount(ResourceType type);

```
private void Start()
▮
    var resourceCount =
ResourcesSystem.Instance.GetResourceCount(ResourcesSystem.ResourceType.Banknotes);
    Debug.Log($"Resource count: {resourceCount}");
▮
```

2. How to change resource count? - Use AddResourceCount(ResourceType type, int value);

```
private void Start()
▮
    private void Start()
▮
    ResourcesSystem.Instance.AddResourceCount(ResourcesSystem.ResourceType.Banknotes, 100);
//increase value
    ResourcesSystem.Instance.AddResourceCount(ResourcesSystem.ResourceType.Banknotes, -80);
//decrease value
}
▮
```

3. How to use TryToBuy(ResourceType type, int price, Action onComplete)?

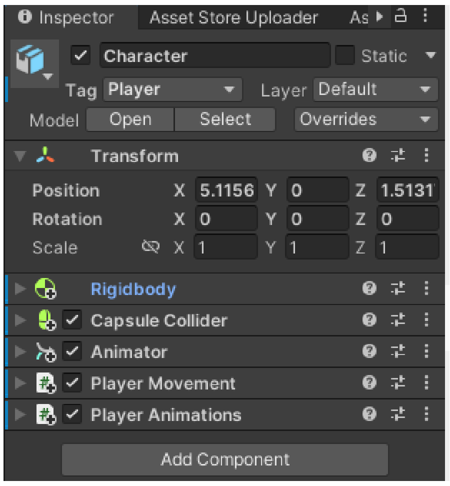
```
private void Start()
{
    ResourcesSystem.Instance.TryToBuy(ResourcesSystem.ResourceType.Banknotes,100, OnComplete);
}

private void OnComplete()
▮
    Debug.Log("Bought!");
▮
```

How to change the character?

Follow the following steps to replace the character or watch the video:

1. Find a character with a skeleton and animations for it; or make a skeleton on mixamo.com, in which case you will be able to use animations that already exist in the asset.
2. Add your character to the stage and add a **Player Movement** component to it. It will add all the other necessary components.



Inspector | Asset Store Uploader | As

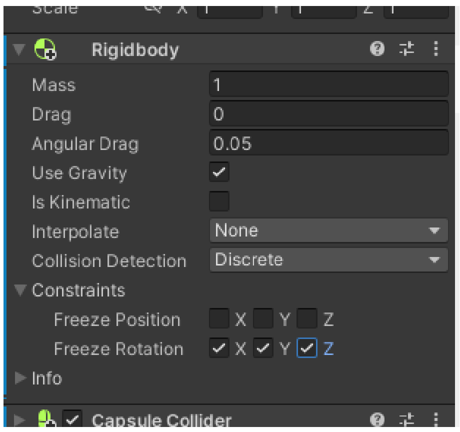
Character ✓ Static
Tag Player Layer Default
Model Open Select Overrides

Transform
Position X 5.1156 Y 0 Z 1.5131
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Rigidbody
✓ Capsule Collider
✓ Animator
✓ Player Movement
✓ Player Animations

Add Component

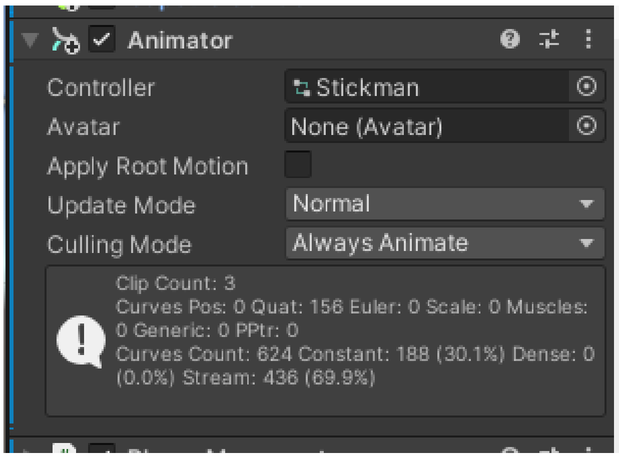
3. First configure **Rigidbody**: Freeze all rotation axes.



Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Use Gravity ✓
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Freeze Position X Y Z
Freeze Rotation ✓ X ✓ Y ✓ Z
Info

✓ Capsule Collider

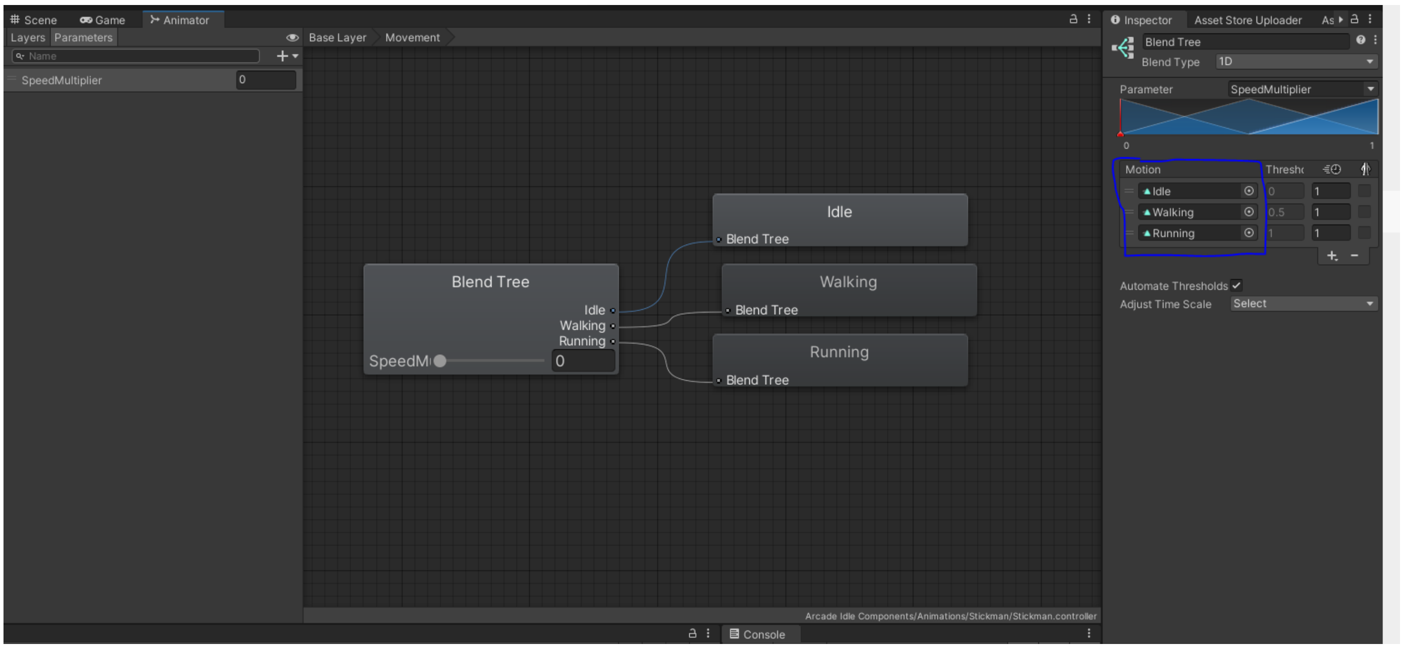
4. Second configure **Capsule Collider**.
5. Third setup the **Animator**: select the controller as in the screenshot.



✓ Animator
Controller Stickman
Avatar None (Avatar)
Apply Root Motion
Update Mode Normal
Culling Mode Always Animate

Clip Count: 3
Curves Pos: 0 Quat: 156 Euler: 0 Scale: 0 Muscles: 0 Generic: 0 PPtr: 0
Curves Count: 624 Constant: 188 (30.1%) Dense: 0 (0.0%) Stream: 436 (69.9%)

Open the animator settings and replace the animation data for your character. If your character is animated through mixamo.com you can't replace.



SpeedMultiplier 0

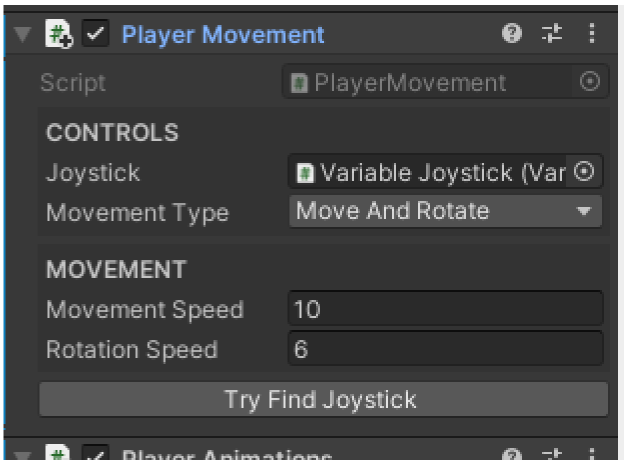
Blend Tree
Blend Type 1D
Parameter SpeedMultiplier

Motion	Threshold		
Idle	0	1	
Walking	0.5	1	
Running	1	1	

Automate Thresholds ✓
Adjust Time Scale Select

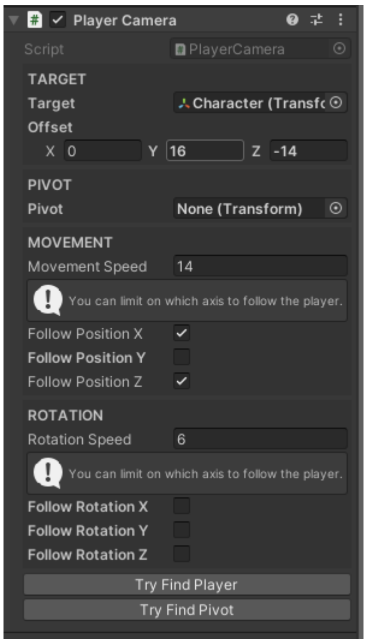
Arcade Idle Components/Animations/Stickman/Stickman.controller

6. Add a link to the joystick and change the speed settings if necessary.



Script PlayerMovement
CONTROLS
Joystick Variable Joystick (Var
Movement Type Move And Rotate
MOVEMENT
Movement Speed 10
Rotation Speed 6

7. Don't forget to replace **Player** link in the **Player Camera**.



Script PlayerCamera
TARGET
Target Character (Transfo
Offset
X 0 Y 16 Z -14
PIVOT
Pivot None (Transform)
MOVEMENT
Movement Speed 14
You can limit on which axis to follow the player.
Follow Position X ✓
Follow Position Y
Follow Position Z ✓
ROTATION
Rotation Speed 6
You can limit on which axis to follow the player.
Follow Rotation X
Follow Rotation Y
Follow Rotation Z

8. That's all you have to do. Now you can run and check.

Components brief

ENVIRONMENT

Player Movement - moves the player with a joystick.

Player Animations - controls the player's animations.

Player Camera - follows player.

Trigger - allows you to easily limit with tags who to work on.

Placement - same as trigger only with delay and progress bar.

Backpack - stacks any items into the number of columns you need.

Special Backpack - stacks several kinds of elements, and set it in a position you want.

Pointer Arrow - sets the position of the arrow relative to the player and rotates it to the target.

Buyable Item - a simple component to help organise an unlimited purchase of something.

Unlockable Item - a simple component that can help you to create object which can be unlocked one time.

Upgradable Item - a simple component that helps to create upgrades panel items. Have maximum upgrades limit.

UI

UI Element - an abstract script created to simplify the use of UI panels and windows.

Save System - a simple singleton class that can load and save PlayerData in PlayerPrefs.

Resources System - a simple singleton class that helps you to perform resource operations.

(Without description those whose name self-describes)

Panel Opener

Settings Panel

Shop Panel

Upgrades Panel

Follow Camera

Resource Counter

Progress Bar

Progress Bar with Items

Button Link

Button Animations

Button Buy

We hope that this documentation will help you understand and learn how to use our cassette, and most importantly, speed up the development of your games! In any case, you can ask us your questions by mail or on the online documentation forum, we will be happy to help!

>>MAIL<<   >>FORUM<<