2.Basic programming with Unity

- This course aims at learning the basics of programming with Unity.
- Unity allows beginners to create own games easily.
- Have fun for programming!

LET'S START

Our Goals

What you can after this course...

1. Understand basic of programming on C#.
2. Understand the role of variables, functions and classes and how they can be used to create efficient code.
3. Able to design simple classes.
4. Able to edit simple scripts and test the impact of changes on the player experience.
5. Able to write your own code using C#.

Class Plan

1. **Introduction**
2. Hello world in Unity
3. Basic programming
4. Object oriented programming
5. Interface programming
6. Programming design 1
7. Programming design 2
8. Improvement and testing 1
9. Improvement and testing 2
10. Summary

1
Introduction

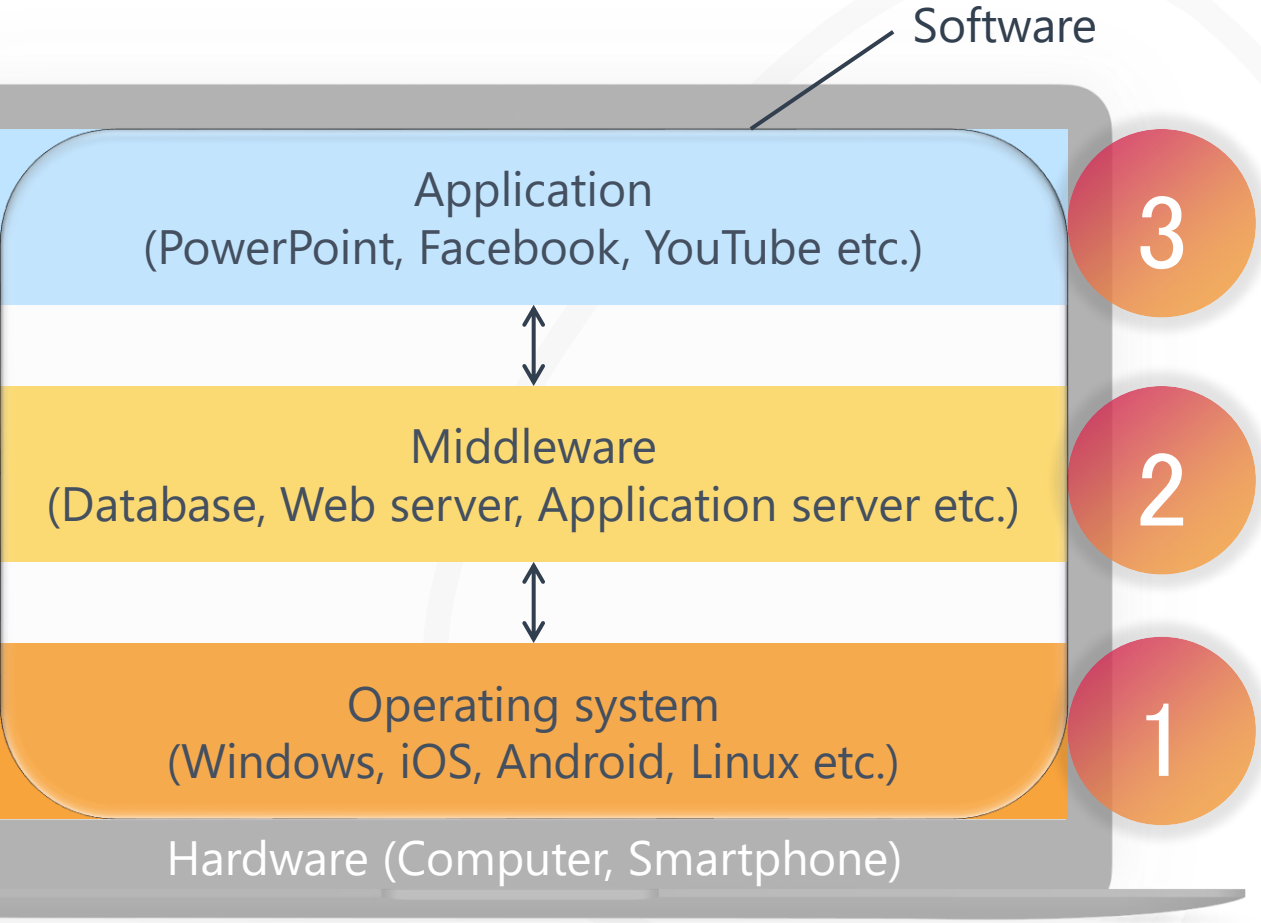
1. Introduction

Class summary

1. Explain the basic knowledge of programming.
2. Explain the programming languages.
3. Explain the process of game and software development.
4. Set up the software used in this course.
5. Explain the basics of Unity.
6. Explain the basic syntax on programming.

What is Programming?

Writing instructions to drive software



3 layer of the Software

Application

Generally, users only can use this layer's software directly to fill ther porpose. In this cource, we will develop it by programming.

3

Middleware

Middleware supports to handle data exchange between Application and Operating System.

2

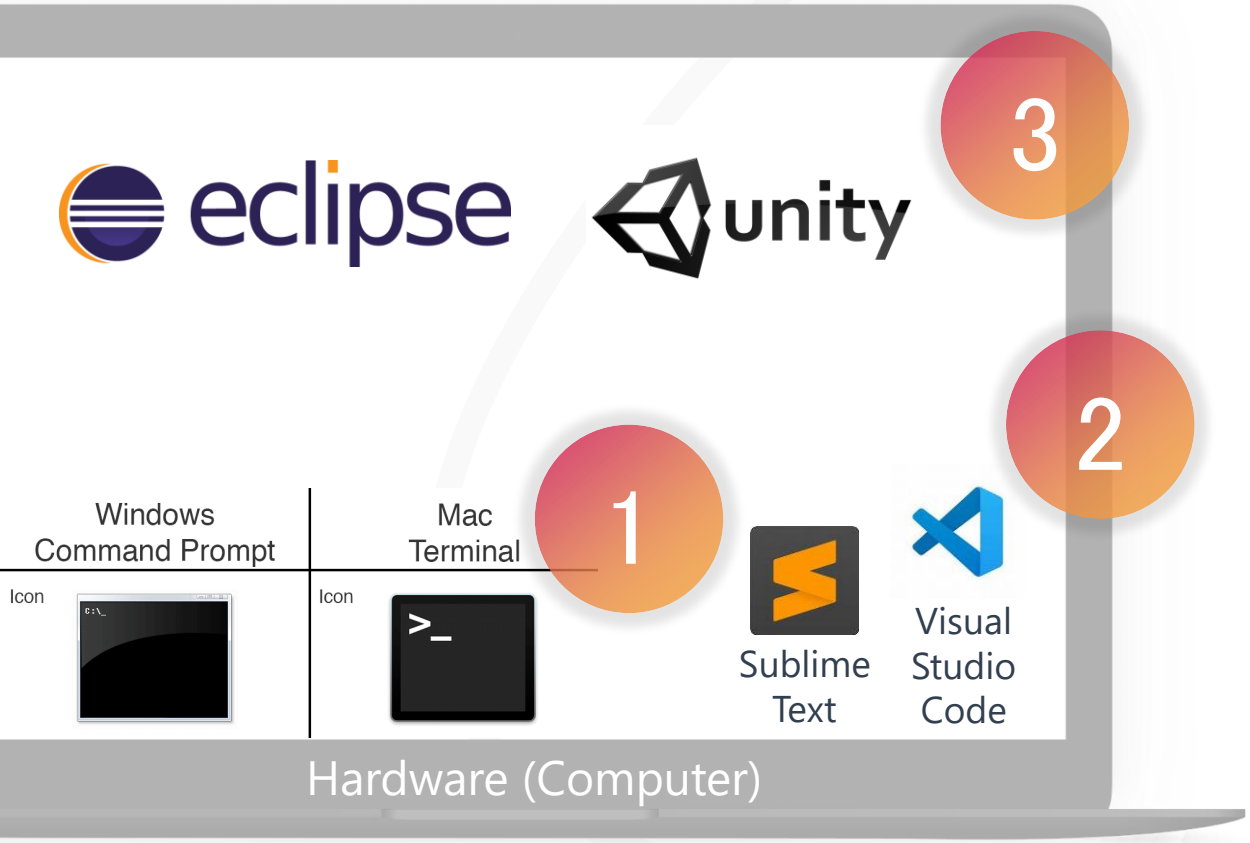
Operating System

OS aims to abstract hardware, manage resources, and improve the efficiency of computer usage.

1

What is Programming?

Writing instructions to drive software



3 type of tools for programming

Integrated Development Environment

IDE is consist of a set of various tools like Programming Editor, Compiler and debuger etc.

3

Programming Editor

Tools to write program efficiently. It supports GUI: Graphical User Interface, input completion and customize etc.

2

Command Prompt / Terminal

Text (command) based interface to manipulate files on computer. It is also called command line interface. Most primitive control method.

1

What is Programming?

Writing instructions to drive software

Program

- Program needs compile to translate machine language.
- The pre-translation provides faster execution.
- It is used to embedded system like OS, car mozule, smartphone, imaging solution, AI and big data etc.
- Languages: C++, C#, Go, Java, Swift etc.

```
#include <stdio.h>

int main(void) {
  printf("Hello World¥n");
  return 0;
}
```

compile

```
0011 0101 0111 1111 0110
0001 1000 0101 1010 1100
1110 1011 0111 0001 0011
1101 0111 1011 1110 0010
```

Source code

Interpriter

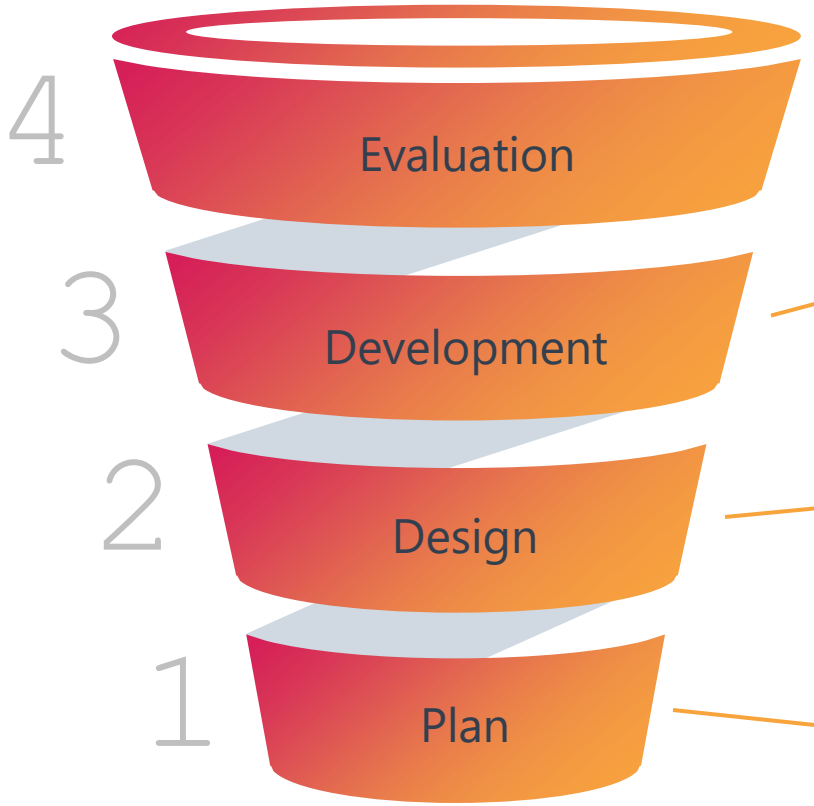
```
<!doctype html>
<html>
 <head>
   <meta charset="utf-8">
   <title>sample</title>
 </head>
<body>
```

Script

- Program needs interpriter to translate machine language.
- The sequential-translation provides faster testing because users do not need waiting compile.
- It is used to web service etc.
- Languages: Javascript, PHP, Phython, Ruby etc.

Development Process

Waterfall and spiral model



- Waterfall: Conduct 1 -> 4 at once.
- Spiral: Conduct 1 or 2 -> 4 repeatedly.

Evaluation / Assessment

Test your code according to documents. How do you find bugs effectively?

Development / Coding

Type code according to design. How do you create beautiful and efficient code?

Design

Express what function do you need to develop it. How do you realize it?

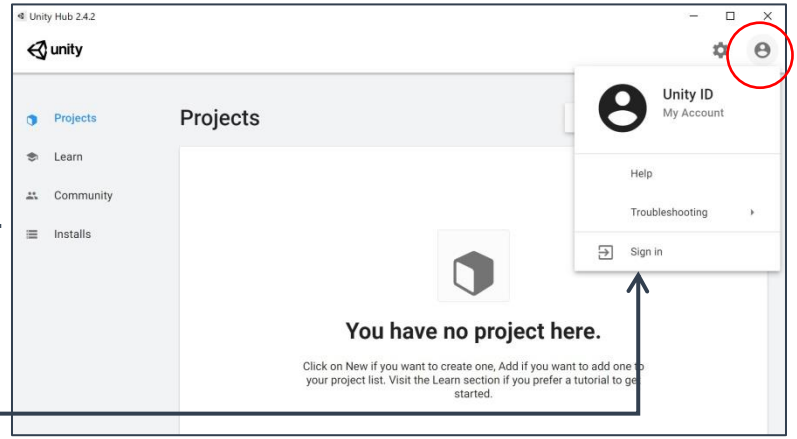
Plan

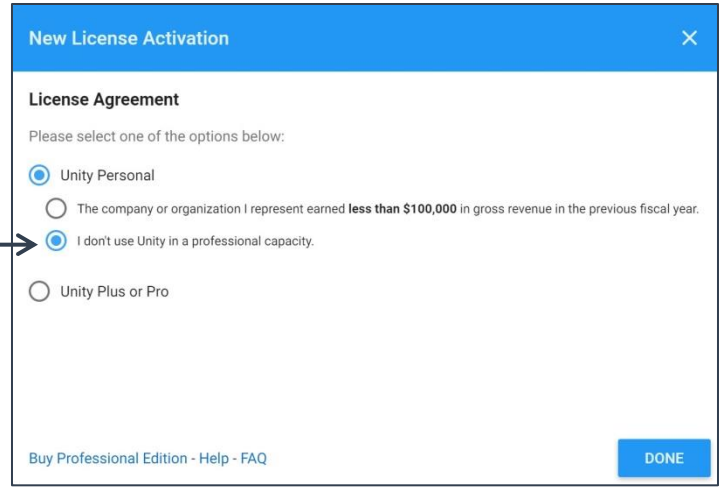
Determine what porpose do you want to meet. How do you satisfy users?

Set up Unity

To create your game

1. Download and Install Unity Hub: Unity's version management system.
    a. Access: https://unity3d.com/jp/get-unity/download
    b. Agree to Unity Terms of Service.
    c. (Windows only) Select install folder. No problem with the default.
2. Activate with free Unity license.
    a. Execute Unity hub.
    b. Create Unity account or sign in via top-right profile icon.
    c. Select ACTIVATE NEW LICENSE in Preferences -> License management.
    d. Select "Unity personal" and "I don't use Unity in a professional capacity."
    e. Select DONE .



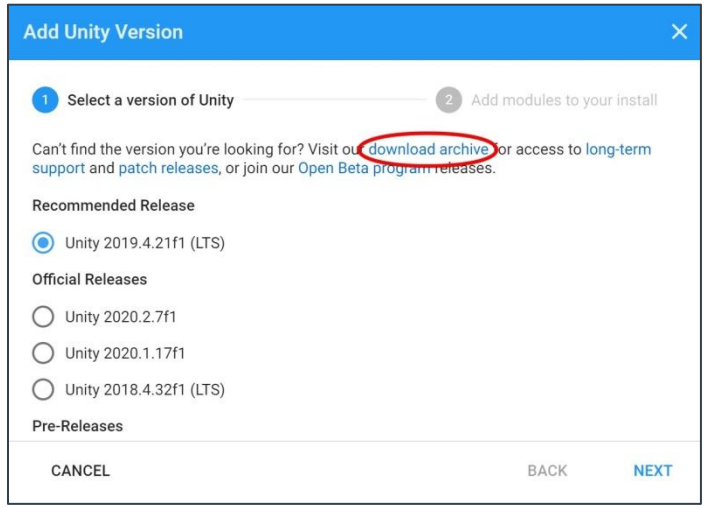


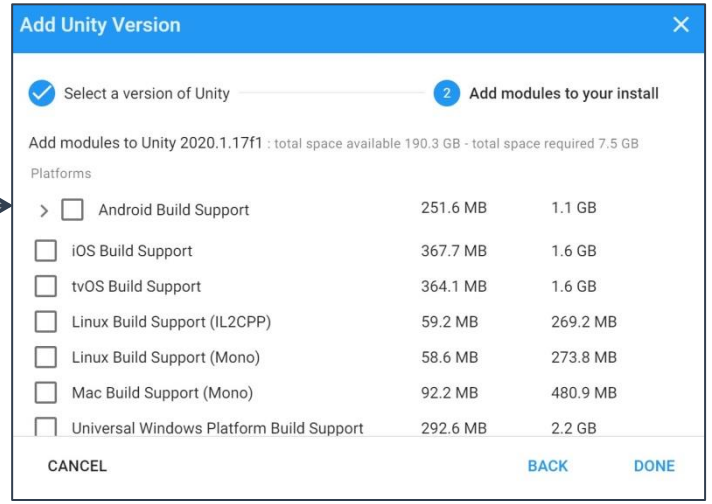
Set up Unity

To create your game

3. Install Unity (and Visual Studio 2019 community).
   a. Back to top from Preferences and select “Installs” -> ADD .
   b. Select “download archive”.
   c. Select Unity Hub on “Unity 2020.2.0”
   d. ⚠ (Mac Only) Check “Visual studio for mac”. Documentation and Language packs are optional.
   e. Select DONE .
   f. Check terms and select DONE .
   g. Wait finishing the download.

This mark means it depends on your environment. I follow up with it on this class.

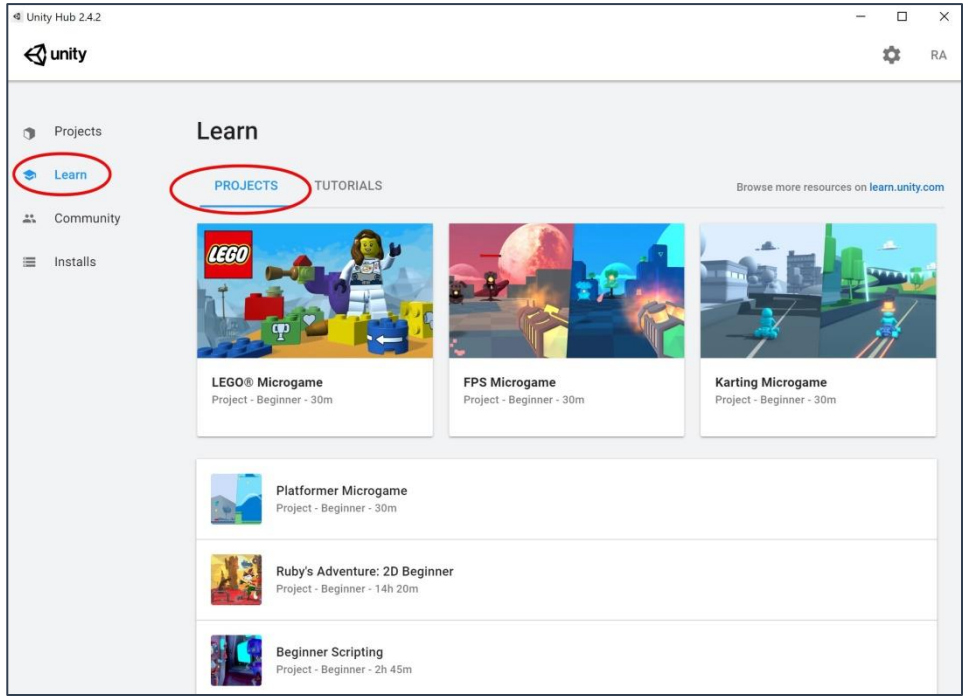


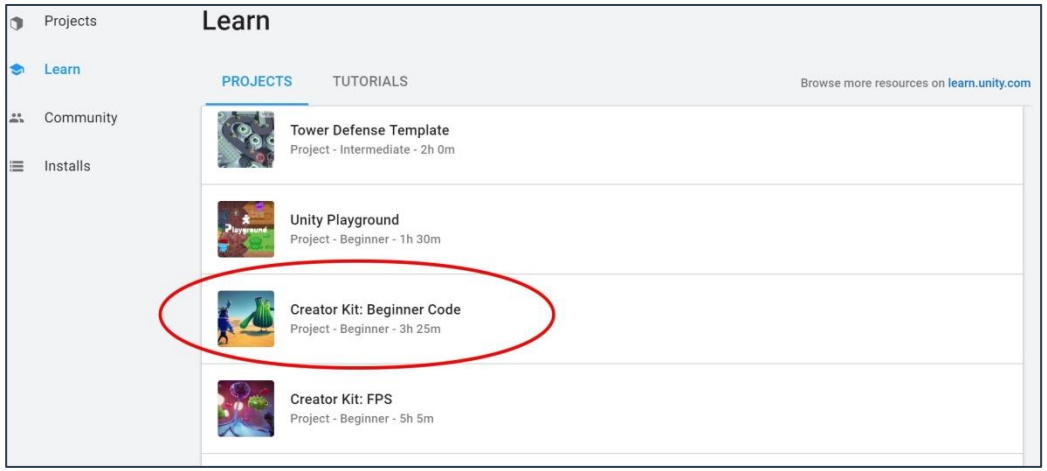


Set up Unity

To create your game

5. After finishing Install, set up tutorial project.
    a. Select “Learn” -> “PROJECTS” tab.
    b. Select “Creator Kit: Beginner Coding”.
    c. ⚠ Select “DOWNLOAD PROJECT”.
    d. ⚠ Wait finishing the download.
    e. ⚠ Select “OPEN PROJECT”.
    Wait a few minutes.
    f. This instruction is written in official page:
    https://learn.unity.com/project/creator-kit-beginner-code
6. Name and save the project.
    - File -> Save as -> save it as “BeginnerCode”.
    - When you save the project, please use ctrl+s (Wndows) or cmd+s (Mac). Useful.
    - If you have problem to save, it may be error from Unity. Please check here and try it.
    - It is recommended to create new folder like “unity/projects” to manage it.

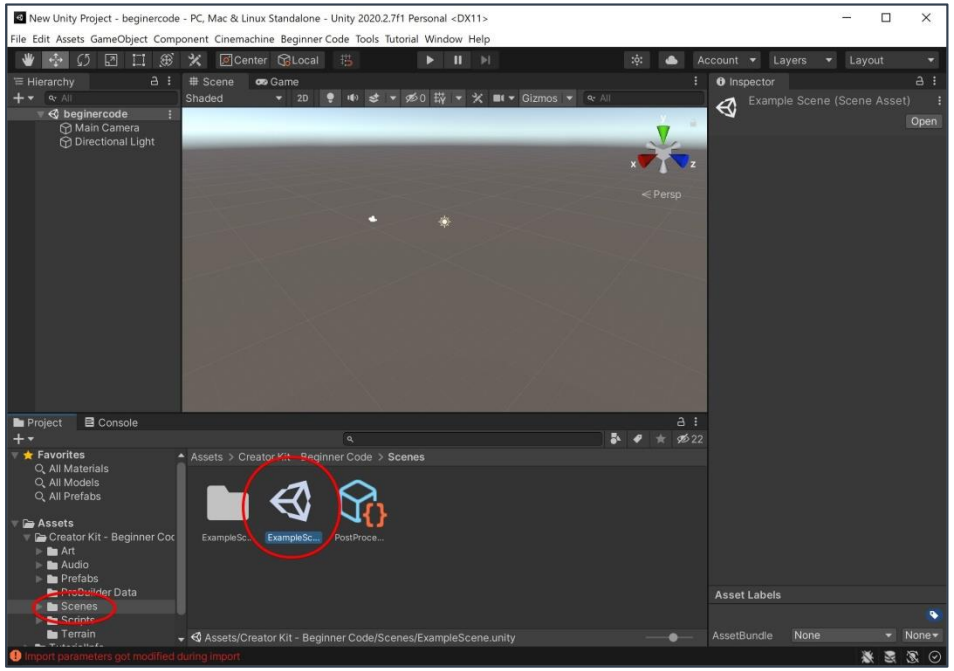


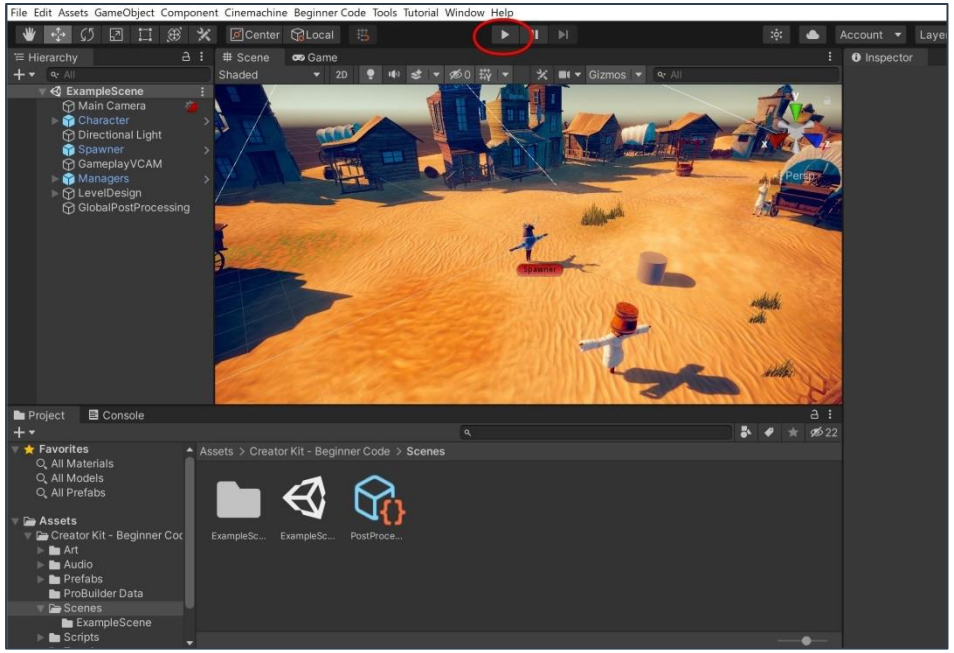


Set up Unity

To create your game

7. Play sample game.
    a. Select below folder.
    “Assets/Creator Kit – Beginner Code/Scenes”
    b. Double click “ExampleScene”.
    c. Select ▶ “Play” button in top-middle.
    d. Start the game.
        - Move character by click.
        - Get Potions on the ground.
        - Open inventory by “i” key or bag icon.
        - Explore and Beat enemies!
        - Use portions by double click on inventory.
        - Find new weapon.
        - End game with Play button again.



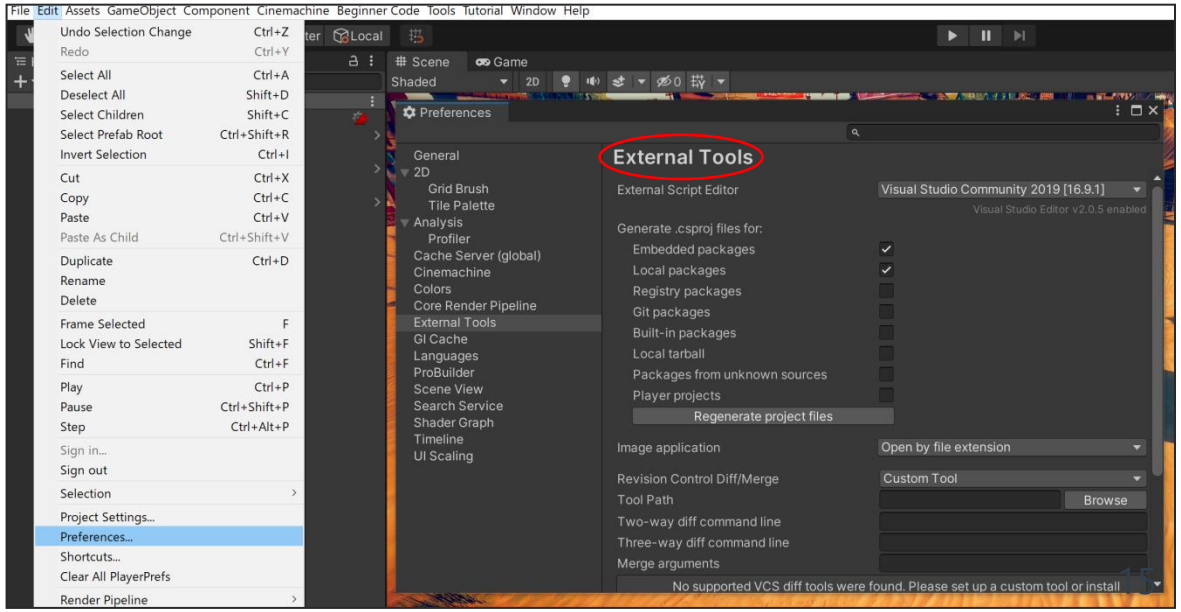


Set up Visual Studio

To create your game

1. (Windows only) Install Visual Studio community.
    a. Exit Unity.
    b. Download and execute installer.
    https://visualstudio.microsoft.com/ja/downloads/
    c. Select Game Development option.
    d. Deselect Unity hub checkbox and install.
    e. Skip or sign in with Microsoft account.
    f. Proceed as default settings.

2. (Windows only) Set up Visual Studio in Unity.
    a. Execute Unity Hub.
    b. Open “beginnercode” project.
    c. Select “Edit” -> “Preferences…” -> “External tools”.
    d. Select “Visual Studio Community 2019” on “External Script Editor”.



What is Unity?

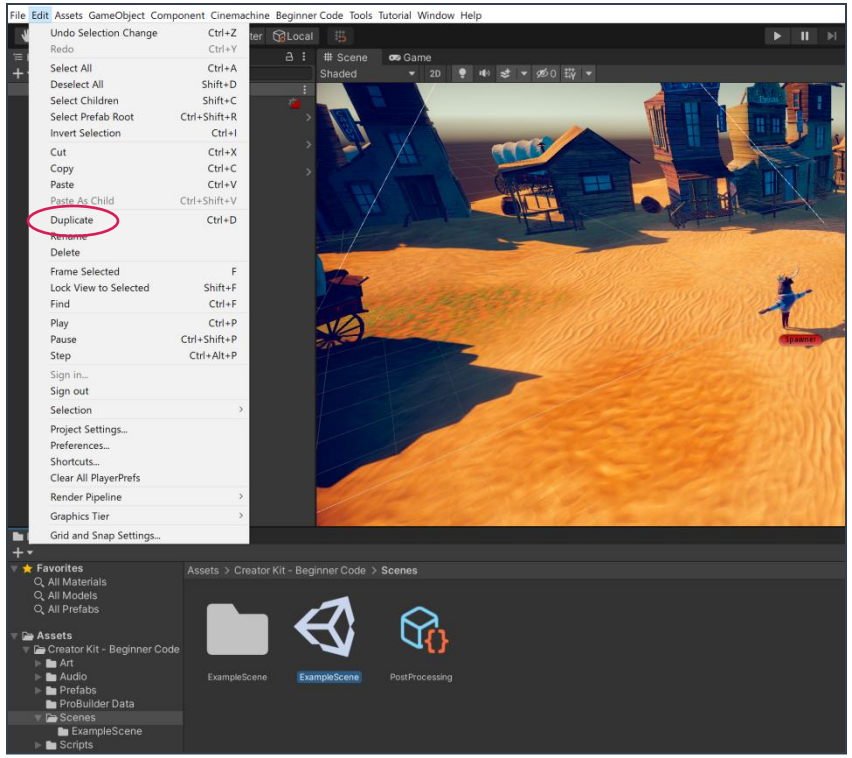
To create your game

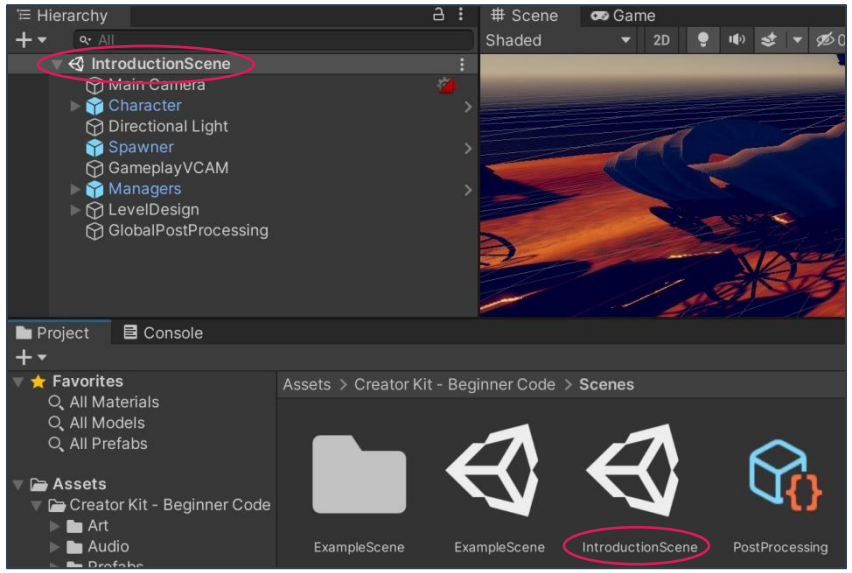
- Most popular game engine IDE in the world.
- Over 50% of smartphone games created with Unity. Maybe, PC games also that.
- It supports 3D/2D programming, physics, lightning, motion capture (for Vtubers) and so on.
- Many 3d models, materials, audio, animations and effects (=assets) can be download.
- C# is supported.
- You can use Unity free!

Understand basics of Unity

To create your game

1. Prepare to edit this game what you want.
    a. Select below folder again.
    “Assets/Creator Kit – Beginner Code/Scenes”
    b. Select “ExampleScene”.
    c. Select Edit -> Duplicate. →
    d. Select “Example scene 1”
    and right click -> “rename” as “IntroductionScene”.
    e. Double click “IntroductionScene”.
    f. Check the below 2 points. →
        - Hierarchy shows “IntroductionScene”.
        - Project shows “IntrodctionScene”.
2. Congratulations! You have became god on this game!
    - Try to use  tools freely.
    - If you want to cancel your last operation, use ctrl+z (windows) or cmd+z (mac).
    - If you want to discard all of your changes, double click “ExampleScene” -> select “Don’t save” and open again.





Understand basics of Unity

To create your game

Components: Gives feature to GameObjects.

e.g. Audio, Render, Physics, Textures, Control Scripts… etc.
Each Component has Properties.

GameObjects: All objects in the game.

e.g. Camera, Light, Buildings, Player character, Enemies, Portions, Ground, Weapons… etc.
Each GameObject has Components.

Scenes: Game project is composed by some of scenes.

e.g. Title Scene, Stage 1 Scene, Game Over Scene… etc.
Each scene has GameObjects.

Understand basics of Unity

To create your game

1. Please translate 5. Review the Unity Editor interface.
2. Please translate 6. Review the Unity Editor toolbar.
3. Attachments to explain keywords on Unity:

Scenes: Game project is composed by some of scenes.
e.g. Title Scene, Stage 1 Scene, Game Over Scene… etc. Each scene has GameObjects.

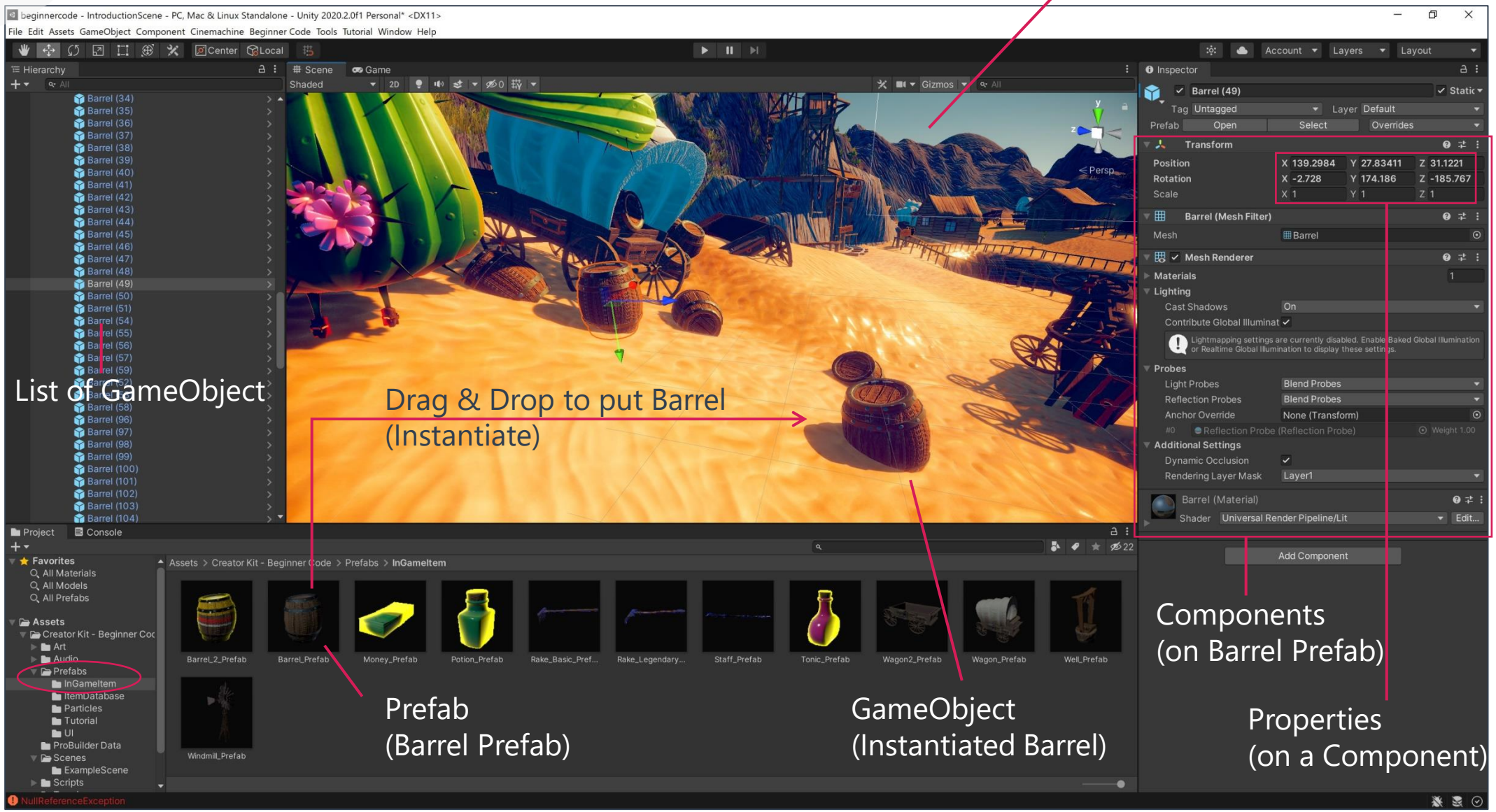
GameObjects: All objects that appear in the game called GameObject.
e.g. Player character, Enemies, Ground, Portions, Weapons… etc. Each GameObject has Components.

Components: It gives feature to GameObjects. A plate GameObject become wall attached by components.
e.g. Physics, Audio, Render, Textures, Control, Scripts… etc. Each Component has Properties.

Prefabs: It is template to create copies of a GameObject with Components. By using wall Prefab, you can put same condition of wall into the Scene. The created copies are called Instance.

Understand basics of Unity

To create your game

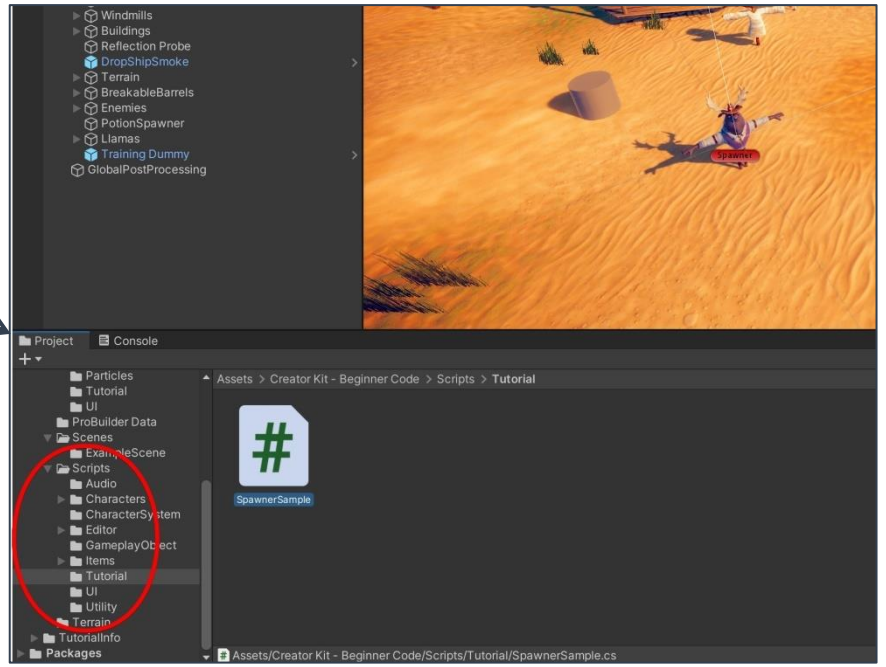


Understand basics of Programming

Architecture of scripts

1. Find “SpawnerSample” script on Project.
2. Click the script.
3. Inspector shows the contents of this script.

   This is program to create 3 portions on the scene.



Spawner Sample (Mono Script)

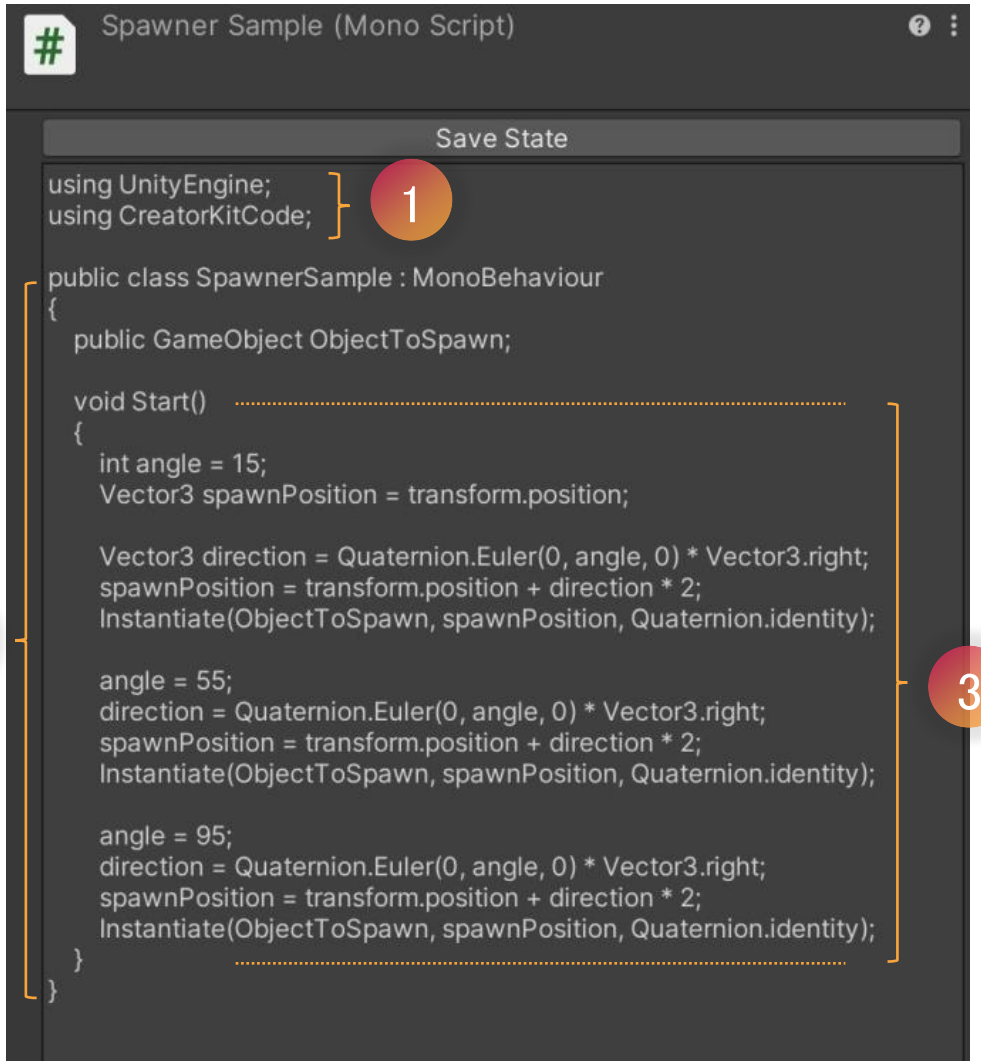
Save State

```
using UnityEngine;
using CreatorKitCode;

public class SpawnerSample : MonoBehaviour
{
    public GameObject ObjectToSpawn;

    void Start()
    {
        int angle = 15;
        Vector3 spawnPosition = transform.position;

        Vector3 direction = Quaternion.Euler(0, angle, 0) * Vector3.right;
        spawnPosition = transform.position + direction * 2;
        Instantiate(ObjectToSpawn, spawnPosition, Quaternion.identity);

        angle = 55;
        direction = Quaternion.Euler(0, angle, 0) * Vector3.right;
        spawnPosition = transform.position + direction * 2;
        Instantiate(ObjectToSpawn, spawnPosition, Quaternion.identity);

        angle = 95;
        direction = Quaternion.Euler(0, angle, 0) * Vector3.right;
        spawnPosition = transform.position + direction * 2;
        Instantiate(ObjectToSpawn, spawnPosition, Quaternion.identity);
    }
}
```

You don’t have to understand this script now, It’s too difficult!
I’ll explain it in order.

Understand basics of Programming

Architecture of scripts



1. Namespace Definitions

- using statements is used to express relationship between this script and the other. These 2 scripts are imported.
- It's called namespace.

2. Class Definitions

- public class statements is used to express the script has folowing class. This script has "SpawnerSample" class.
- "{" and "}" show the class range (scope).

3. Function (Method) Definitions

- void Start() statements is used to express the class has Start function. void and () are described later.
- Functions are also scoped by "{" and "}".

Understand basics of Programming

Architecture of scripts

1. Class is blueprint of scripts. Unity provides a lot of classes.
2. Class can inherit variables and functions of its parent.
3. Namespace provides shortcut to call other classes' factor.

C# C# C# C#

Component class

```
namespace UnityEngine
{
    public class  Component : Object
    {
        public GameObject gameObject;
        public String tag;
        public Transform transform;
        ...

        public Component GetComponent
(string type)
        {
            ...
        }
        ...
    }
}
```

Behaviour class

```
namespace UnityEngine
{
    public class  Behaviour : Component
    {
        public bool enabled;
        public bool isActiveAndEnabled;
        ...
    }
}
```

MonoBehaviour class

```
namespace UnityEngine
{
    public class  MonoBehaviour : Behaviour
    {
        public bool runInEditMode;
        public bool useGUILayout;

        public void Start()
        {
            // In fact, it is defined elsewhere.
        }

        public void Update()
        {
            // In fact, it is defined elsewhere.
        }
        ...
    }
}
```

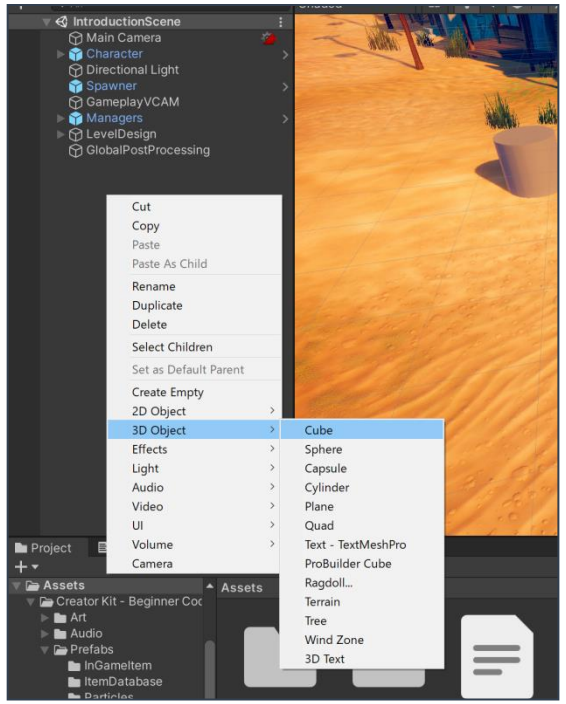
SpawnerSample class

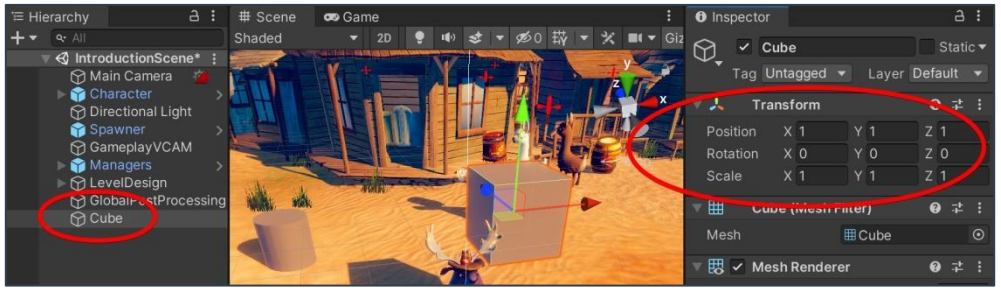
```
using UnityEngine;
using CreatorKitCode;

public class  SpawnerSample : MonoBehaviour
{
    public GameObject ObjectToSpawn;

    public void Start()
    {
        int angle = 15;
        Vector3 spawnPosition = transform.position;
        ...
    }
}
```

Try to write own Script

The first step of programming.

1. Create GameObject to attach script.
    a. Right click blank space at Hierarchy.
    b. Select 3D Object -> Cube.
    c. Select Cube and set Transform.position to (1, 1, 1).
    d. Play the game and check the cube appeared.
2. Add Physics to the cube.
    a. Select Add Component on inspector of Cube.
    b. Select Physics -> Rigidbody.
    c. Play the game and check the changes.
3. Create script file for cube.
    a. Open Scripts/Tutorial folder on project.
    b. Right click blank space at tutorial folder and select Create -> C# Script.
    c. Right click the new one and select Rename to “CubeController”.
    d. Open CubeController (Double click).





Try to write own Script

The first step of programming.

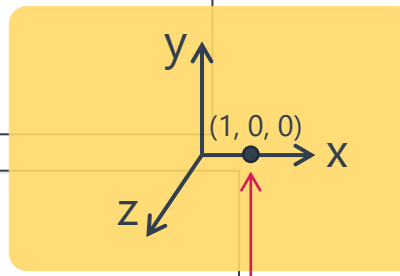
1. Check architecture of the script.
   a. // means 1 line comments.
   b. using means importing class and functions.
   c. /* ~ */ also means multiple line of comments.
   d. public class statement is class definition.
   e. void Start() and void Update() are functions.

2. Write the script for cube.
   a. Edit Update() method according to this image.
      You don't have to write my comments.
   b. Save (ctrl+S / cmd+s) the file and back to the Unity.

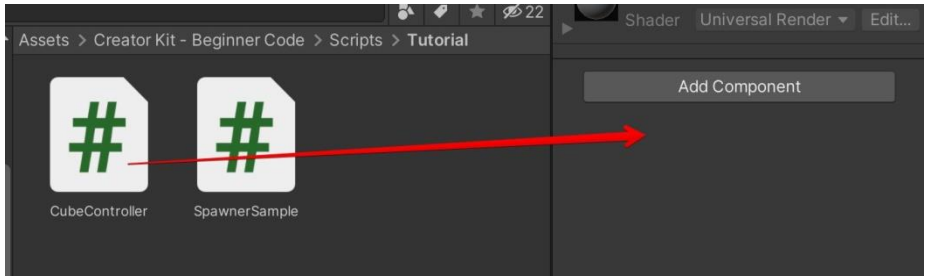
3. Attach the script as Cube component.
   a. Select Cube.
   b. Drag & Drop "CubeController" into Inspector.
      (Near the "Add Component")
   c. Play the game on Unity.

CubeController.cs

```
1   // Import C# and Unity functions
2   using System.Collections;
3   using System.Collections.Generic;
4   using UnityEngine;
5
6   /*
7    * Check or edit the class name according to file namme
8    *  -> "CubeController"
9    */
    Unity スクリプト|0 個の参照
10  public class CubeController : MonoBehaviour
11  {
12      // Start is called before the first frame update
        Unity メッセージ|0 個の参照
13      void Start()
14      {
15
16      }
17
18      // Update is called once per frame
        Unity メッセージ|0 個の参照
19      void Update()
20      {
21
22      }
23  }
24
```




```
// Update is called once per frame
Unity メッセージ|0 個の参照
void Update()
{
    /*
     * To move cube, change the transform.position
     * by adding Vector3.right. It means (1,0,0).
     */
    transform.position = transform.position + Vector3.right;
}
```



Try to write own Script

The first step of programming.

1. Fix the cube speed (It's too fast!).
   a. Edit CubeController according to this image. → You don't have to write my comments.
   b. Save the file and back to the Unity.
   c. Play the game again.

CubeController.cs

```
public class CubeController : MonoBehaviour
{
    /*
     * Definition of speed variable
     * Variable type: float (decimal number)
     */
    public float speed = 0.5f;

    // Start is called before the first frame update
    Unity メッセージ | 0 個の参照
    void Start()
    {

    }

    // Update is called once per frame
    Unity メッセージ | 0 個の参照
    void Update()
    {
        /*
         * To move cube, change the transform.position
         * by adding Vector3.right. It means (1,0,0).
         */
        transform.position = transform.position + Vector3.right * speed;
    }
```

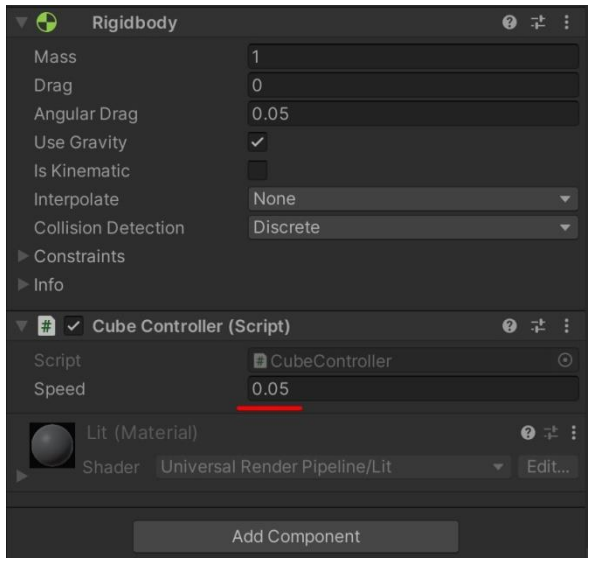
2. Still too fast? OK, let's fix it on Inspector.
   a. Select Cube and find CubeController on Inspector.
   b. Edit the "Speed" to 0.05. →
   c. Play the game again.

0.05

Variable

(float) speed

Types	Means	Examples
int	Integer	1
float	Decimal	0.5f
string	Characters	"Unity"
bool	Boolean	true or false
Vector3	x,y,z	new Vector3(1,1,1)
GameObject	Object	Object name



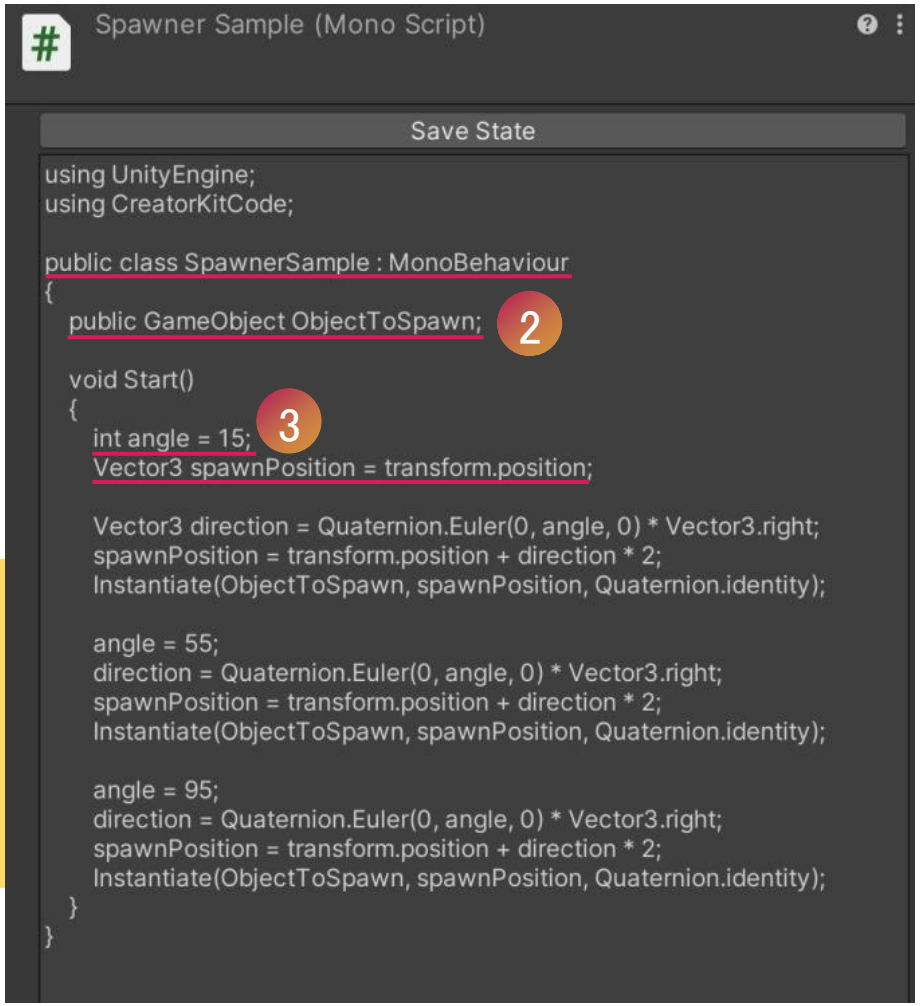
Try to write own Script

The first step of programming.

1. Please translate 2. Find an instruction in the script.
2. Please translate 3. Introduction to variables.
3. Please translate 4. Identify the type and name of the variable.
4. Please translate 5. Identify the value assigned to the variable.
5. You can understand parts of red line on this image.

3 case of variables

1. Global (private) variable: Only used in the class scope.
2. Global public variable: Used in classes scope + Unity inspector.
3. Local variable: Used in the function scope.



Attachment

This is important!

Coding Rules: Examples

Target	Major Rules	Examples
Class name	a. UpperCamel	a. SampleClass
Class variables	a. LowerCamel b. Snake_case c. Underscore	a. sampleVariable b. sample_variable c. _sampleVariable (Only for private variables)
Bool variables	a. Is_bool	a. is_run
Methods	a. UpperCamel	a. SampleMethod()
Constants	a. All-Caps b. UpperCamel	a. MAX_NUMBER b. MaxNum
Scene name	a. UpperCamel	a. TitleScene
Object name	a. UpperCamel	a. CactusNails
If Statement	a. if (target < condition)	a. if (num < 10)
Comments	You don't' need to write all of contents. Good code & Good name explains the role or function itself.	
Scope	Try to minimize the range.	
Hardcoding	Don't use it. You should use constants.	
Core mind	Simple and Dry. Functions should be under 30〜50 lines.	

Target	Pronunciation	Examples
()	Parenthesis	void Method()
[]	Bracket	int array[3]
{}	Brace	if (num < 10) { }

Attachment

This is important!

When you get Errors!

Check Points / ToDo	Details
Spell miss	Check your script carefully! We need to be case-sensitive in programming.
Error messages	Error messages may indicate the cause of the error or the corresponding line.
Search on Google	Type the error message into Google search. You may find the solutions. Sometimes, the errors / cautions can be ignored.
Narrow down the cause	Use comment out or Debug.Log("") wisely to find the problem.
Attached components	Check inspector on the GameObjects. You may forgot attach the components.
Careless miss	• Forgotten to save the scripts • Target GameObject was disabled • Log message was disabled
When you ask it…	When you ask someone the problem, make sure you clarify the followings. • **What problems happened?** • **When the problems happened?** • **Were you able to identify where is problem?** • **How did you try to solve it so far?** • **What is your goal?**

2.Basic programming with Unity

Class Plan

1. Introduction
2. **Hello world in Unity**
3. Basic programming
4. Object oriented programming
5. Interface programming
6. Programming design 1
7. Programming design 2
8. Improvement and testing 1
9. Improvement and testing 2
10. Summary

2
Hello world in Unity

2. Hello world in Unity

Class summary

1. Explain the variables and data type.
2. Explain basic arithmetic operations.
3. Explain the functions and scope.
4. Explain the conditional statements.
5. Explain debug method.

Review of Previous Class

The first step of programming.

Types	Means	Examples
int	Integer	1
float	Decimal	0.5f
string	Characters	“Unity”
bool	Boolean	true or false
Vector3	x,y,z	new Vector3(1,1,1)
GameObject	Object	Object name

Create CubeController script.

```
1     // Import C# and Unity functions
2     using System.Collections;
3     using System.Collections.Generic;
4     using UnityEngine;
5
6     /*
7      * Check or edit the class name according to file namme
8      *  -> "CubeController"
9      */
      Unity スクリプト|0 個の参照
10    public class CubeController : MonoBehaviour
11    {
12        /*
13         * Definition of speed variable
14         * Variable type: float (decimal number)
15         */
16        public float speed = 0.5f;   2
17
18        // Start is called before the first frame update
          Unity メッセージ|0 個の参照
19        void Start()
20        {
21
22        }
23
24        // Update is called once per frame
          Unity メッセージ|0 個の参照
25        void Update()
26        {
27            /*
28             * To move cube, change the transform.position
29             * by adding Vector3.right. It means (1,0,0).
30             */
31            transform.position = transform.position + Vector3.right * speed;
32        }
33    }
```

3

3 case of variables

1. Global (private) variable: Only used in the class scope.
2. Global public variable: Used in classes scope + Unity inspector.
3. Local variable: Used in the function scope.

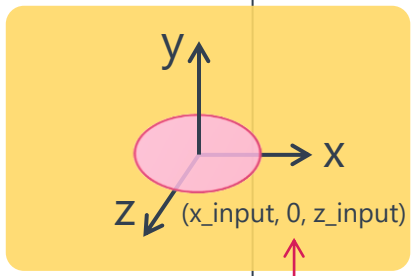
Try to write own Script

The first step of programming.

1. Fix the CubeController.
    a. Edit CubeController according to this image. You don't have to write my comments.
    b. Save the file and back to the Unity.
    c. Play the game again.
        - You can move Cube with arrow keys.

CubeController.cs

```
Unity スクリプト | 0 個の参照
public class CubeController : MonoBehaviour
{
    public float speed = 0.5f;

    // Start is called before the first frame update
    Unity メッセージ | 0 個の参照
    void Start()
    {

    }

    // Update is called once per frame
    Unity メッセージ | 0 個の参照
    void Update()
    {
        // Variable definition for Input ( ← and → )
        float x_input = Input.GetAxis("Horizontal");

        // Variable definition for Input ( ↑ and ↓)
        float z_input = Input.GetAxis("Vertical");

        // Substitute transform.position(Vector3) + inputed value as Vector3.
        transform.position += new Vector3(x_input,0,z_input) * speed;
    }
}
```



Description.

- Input.GetAxis(string axisName) method by UnityEngine
    - Returns float value between -1 to +1
    - Can be able "Horizontal", "Vertical" and more. User can define it yourself.
- new Vector3(x, y, z) statement
    - Create Vector3 struct to substitute inputted float values for transform.position (Vector3).
    - Vector3 has float x, float y, float z variables.
    - new statement is used to create class/struct.

Arithmetic operators	Means	Examples
=	Substitution	Speed = 0.5f
==	Equal	A == B
+	Addition, Connection	10 + 20 == 2 "ABC" + "DEF" == "ABCDEF"
-	Subtraction	10 – 5 == 5
*	Multiplication	5 * 8 == 40
/	Division	40 / 8 == 5
%	Remainder	5 % 2 == 1
(operator)=	Operate itself	**(A += 10)** == **(A = A + 10)**

Exercise

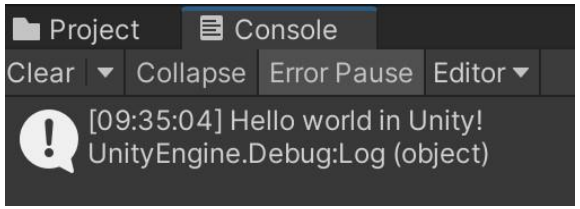
Try to basic arithmetic

1. Edit CubeController.Start() according to this image.

CubeController.cs

```
// Start is called before the first frame update
Unity メッセージ|0 個の参照
void Start()
{
    Debug.Log("Hello world in Unity!");
}
```

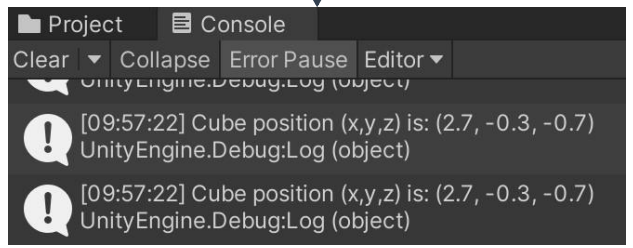
2. Select Console and play the game.
   - Unity shows this message.
   - Debug.log() method shows the message when called it.



3. Edit CubeController.update() according to this image.

```
// Update is called once per frame
Unity メッセージ|0 個の参照
void Update()
{
    // Variable definition for Input ( ← and → )
    float x_input = Input.GetAxis("Horizontal");

    // Variable definition for Input ( ↑ and ↓)
    float z_input = Input.GetAxis("Vertical");

    // Substitute transform.position(Vector3) + inputed value as Vector3
    transform.position += new Vector3(x_input,0,z_input) * speed;
    Debug.Log("Cube position (x,y,z) is: " + transform.position);
}
```

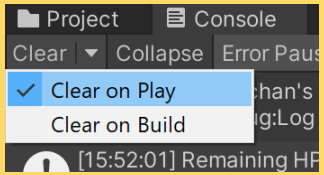
4. Select Console and play the game.
   - Unity shows these messages.
   - Update() is called once per frame, so a lot of messages appeared.
   - Debug.log() method is useful to check parameters.



5. Delete or comment out Debug.Log() on Update().
   - It fills up console by log messages…

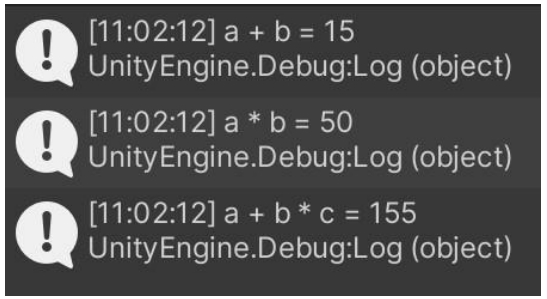
Exercise

Try to basic arithmetic

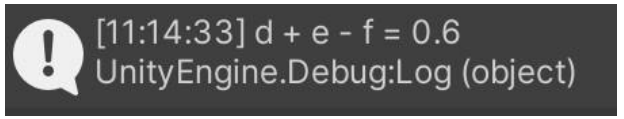


Please use clear option in console.

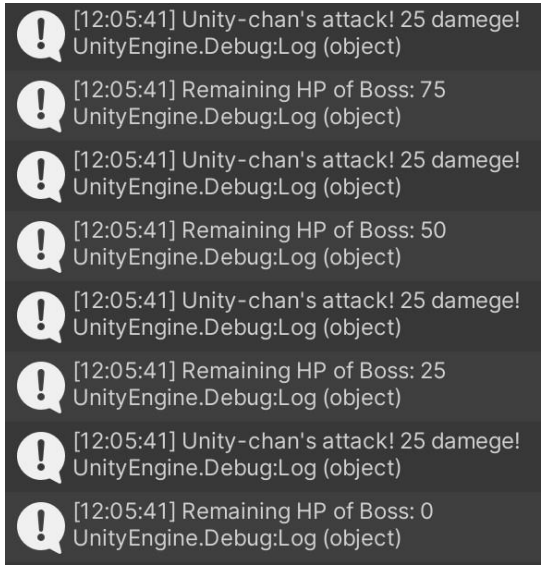
1. Edit Start() to meet below arithmetic.
    a. Define int a = 5, int b = 10, int c = 15.
    b. Calculate and output according to this log on console. →



2. Edit Start() to meet below arithmetic.
    a. Comment out Exercise 1.
    b. Define float d = 1.0, float e = 0.1f, float f = 0.5f, float result;
    c. Calculate {d} + {e} and substitute it for result.
    d. Calculate {result} -= {f} and output it on console. →



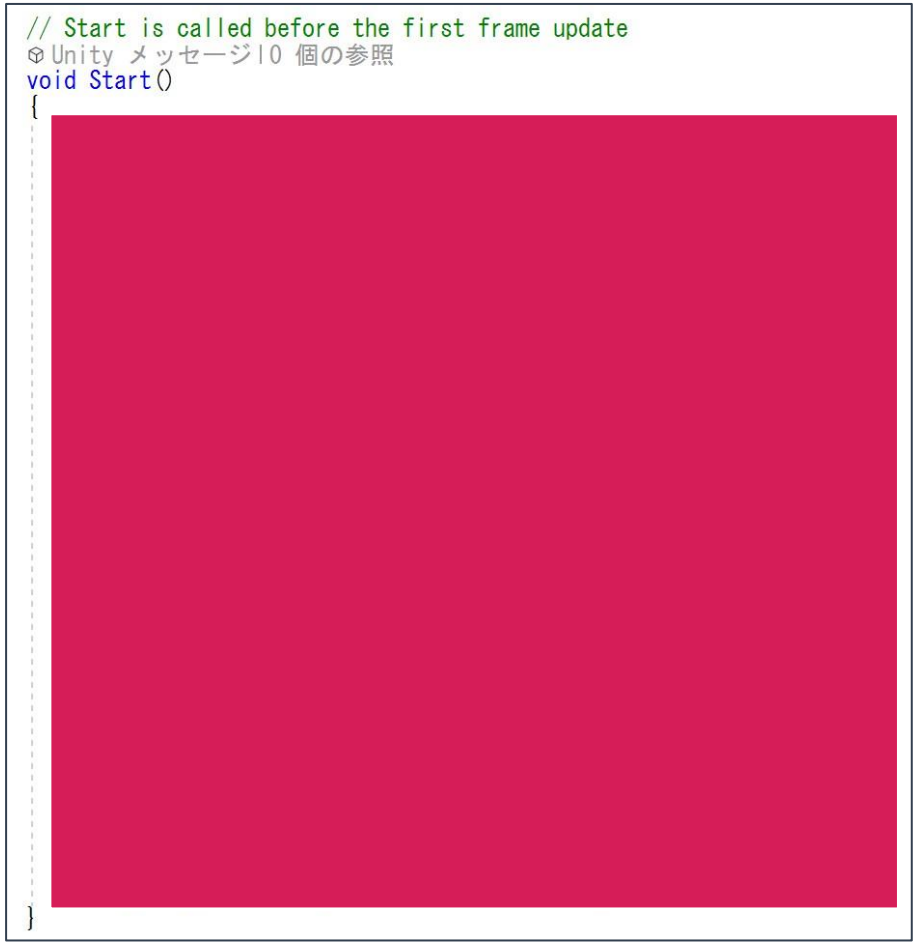
3. Edit Start() to meet below arithmetic.
    a. Comment out Exercise 2.
    b. Define int boss_HP = 100, int player_ATK = 25,
    string player_name = “Unity-chan”.
    c. Calculate {boss_HP} by taking player’s attack with “-=“ operator.
    d. Out put the remaining HP of boss. →
    e. Calculate and output according to this log on console.
    (To beat boss: repeat it until the HP become 0)



Exercise

Try to basic arithmetic

1. Edit Start() to meet below arithmetic.
    a. Define int a = 5, int b = 10, int c = 15.
    b. Calculate and output according to this log on console.
2. Edit Start() to meet below arithmetic.
    a. Comment out Exercise 1.
    b. Define float d = 1.0, float e = 0.1f, float f = 0.5f, float result;
    c. Calculate {d} + {e} and substitute it for result.
    d. Calculate {result} -= {f} and output it on console.
3. Edit Start() to meet below arithmetic.
    a. Comment out Exercise 2.
    b. Define int boss_HP = 100, int player_ATK = 25,
    string player_name = “Unity-chan”.
    c. Calculate {boss_HP} by taking player’s attack with “-=“ operator.
    d. Out put the remaining HP of boss.
    e. Calculate and output according to this log on console.
    (To beat boss: repeat it until the HP become 0)



It's not Cool...

Try to basic arithmetic

```
// Excercise 3
int boss_HP = 100, player_ATK = 25;
string player_name = "Unity-chan";
boss_HP -= player_ATK;
Debug.Log(player_name + "'s attack! " + player_ATK + " damege!");
Debug.Log("Remaining HP of Boss: " + boss_HP);
boss_HP -= player_ATK;
Debug.Log(player_name + "'s attack! " + player_ATK + " damege!");
Debug.Log("Remaining HP of Boss: " + boss_HP);
boss_HP -= player_ATK;
Debug.Log(player_name + "'s attack! " + player_ATK + " damege!");
Debug.Log("Remaining HP of Boss: " + boss_HP);
boss_HP -= player_ATK;
Debug.Log(player_name + "'s attack! " + player_ATK + " damege!");
Debug.Log("Remaining HP of Boss: " + boss_HP);
}
```

1

2

1. Hard-coded "Boss" string

- Boss also want to attack to player. However, this code always show "Boss".
- This code don't have versatile. It shoud be replaced to {target_name} variable.

2. Redundant code

- This code has a lot of same statements. It's low-readability and editibility.
- It should be replaced to function.

Speck of Attack() function:

1. It uses following variables:
   int attacker_name, int attacker_ATK, string target_name, int target_HP
2. It calculates {target_HP} -= {attacker_ATK}
3. It outputs attack message and remaining HP.

Try to write Functions

The second step of programming.

1. Optimize previous code by using functions.
    a. Edit CubeController according to this image. You don't have to write my comments.
    b. Save the file and back to the Unity.
    c. Play the game again.
        - Check output in console.

Description.

- void FunctionName([Type] variable, ...)
    - variable is called "arguments".
    - variable only can be used in the function.
    - variable has data-copy or data-address of the caller's arguments.

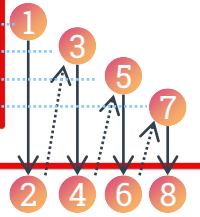
```
Attack(player_name, player_ATK, boss_name, boss_HP);
```

```
void Attack(string attacker_name, int attacker_ATK, string target_name, int target_HP)
```

CubeController.cs

```
Unity スクリプト | 0 個の参照
public class CubeController : MonoBehaviour
{
    public float speed = 0.5f;

    // Define names as global scope because
    // names will be used in many functions.
    public string player_name = "Unity-chan";
    public string boss_name = "Cuctus-king";

    // Start is called before the first frame update
    Unity メッセージ | 0 個の参照
    void Start()
    {
        // Define variables
        int boss_HP = 100, boss_ATK = 20;
        int player_HP = 85, player_ATK = 25;

        // Call Attack function with variables
        Attack(player_name, player_ATK, boss_name, boss_HP);
        Attack(boss_name, boss_ATK, player_name, player_HP);
        Attack(player_name, player_ATK, boss_name, boss_HP);
        Attack(boss_name, boss_ATK, player_name, player_HP);
    }

    // Define Attack() function with variables
    4 個の参照
    void Attack(string attacker_name, int attacker_ATK, string target_name, int target_HP)
    {
        // Caluculate
        target_HP -= attacker_ATK;

        // Output
        Debug.Log(attacker_name + "'s attack! " + attacker_ATK + " damege!");
        Debug.Log("Remaining HP of " + target_name + ": " + target_HP);
    }
```

Flow of program



Try to write Functions

The second step of programming.

```
[15:52:01] Unity-chan's attack! 25 damege!
UnityEngine.Debug:Log (object)
[15:52:01] Remaining HP of Cuctus-king: 75
UnityEngine.Debug:Log (object)
[15:52:01] Cuctus-king's attack! 20 damege!
UnityEngine.Debug:Log (object)
[15:52:01] Remaining HP of Unity-chan: 65
UnityEngine.Debug:Log (object)
[15:52:01] Unity-chan's attack! 25 damege!
UnityEngine.Debug:Log (object)
[15:52:01] Remaining HP of Cuctus-king: 75
UnityEngine.Debug:Log (object)
[15:52:01] Cuctus-king's attack! 20 damege!
UnityEngine.Debug:Log (object)
[15:52:01] Remaining HP of Unity-chan: 65
UnityEngine.Debug:Log (object)
```

100-25=75

100-25=75

Description.

- void FunctionName([Type] variable, ...)
  - variable is called “arguments”.
  - variable only can be used in the function.
  - variable has data-copy or data-address of the caller’s arguments.

```
Attack(player_name, player_ATK, boss_name, boss_HP);
```

```
void Attack(string attacker_name, int attacker_ATK, string target_name, int target_HP)
```

CubeController.cs

```
Unity スクリプト | 0 個の参照
public class CubeController : MonoBehaviour
{
    public float speed = 0.5f;

    // Define names as global scope because
    // names will be used in many functions.
    public string player_name = "Unity-chan";
    public string boss_name = "Cuctus-king";

    // Start is called before the first frame update
    Unity メッセージ | 0 個の参照
    void Start()
    {
        // Define variables
        int boss_HP = 100, boss_ATK = 20;
        int player_HP = 85, player_ATK = 25;

        // Call Attack function with variables
        Attack(player_name, player_ATK, boss_name, boss_HP);
        Attack(boss_name, boss_ATK, player_name, player_HP);
        Attack(player_name, player_ATK, boss_name, boss_HP);
        Attack(boss_name, boss_ATK, player_name, player_HP);
    }

    // Define Attack() function with variables
    4 個の参照
    void Attack(string attacker_name, int attacker_ATK, string target_name, int target_HP)
    {
        // Caluculate
        target_HP -= attacker_ATK;

        // Output
        Debug.Log(attacker_name + "'s attack! " + attacker_ATK + " damege!");
        Debug.Log("Remaining HP of " + target_name + ": " + target_HP);
    }
```

Try to write Functions

The second step of programming.

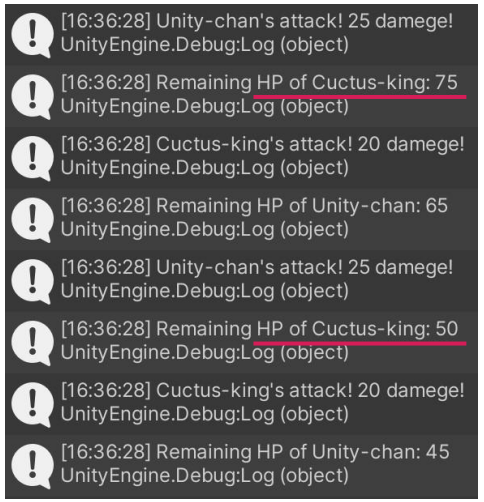
1. Refactor the code.
    a. Edit CubeController according to this image. You don't have to write my comments.
    b. Save the file and back to the Unity.
    c. Play the game again.
        - Check output in console.

Description.

- [Type] FunctionName([Type] variable, …)
    - The function returns [Type] value / object.
    - It called "Return value".
    - It needs return statement.

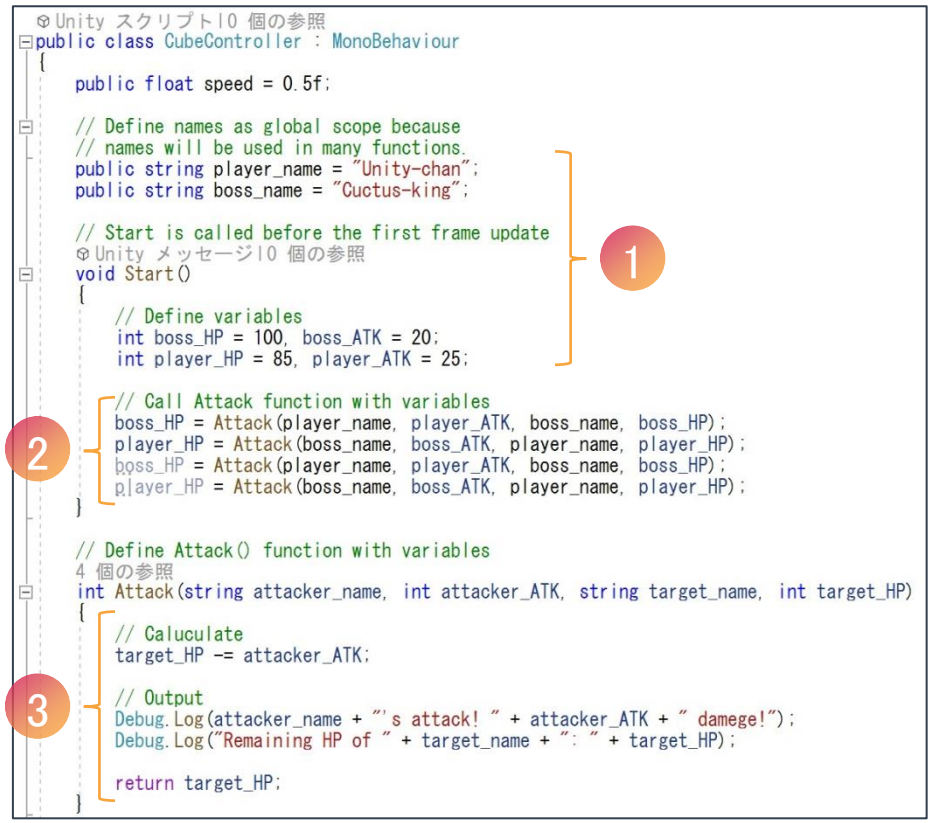
CubeController.cs

```
public class CubeController : MonoBehaviour
{
    public float speed = 0.5f;

    // Define names as global scope because
    // names will be used in many functions.
    public string player_name = "Unity-chan";
    public string boss_name = "Cuctus-king";

    // Start is called before the first frame update
    Unity メッセージ | 0 個の参照
    void Start()
    {
        // Define variables
        int boss_HP = 100, boss_ATK = 20;
        int player_HP = 85, player_ATK = 25;

        // Call Attack function with variables
        boss_HP = Attack(player_name, player_ATK, boss_name, boss_HP);
        player_HP = Attack(boss_name, boss_ATK, player_name, player_HP);
        boss_HP = Attack(player_name, player_ATK, boss_name, boss_HP);
        player_HP = Attack(boss_name, boss_ATK, player_name, player_HP);
    }

    // Define Attack() function with variables
    4 個の参照
    int Attack(string attacker_name, int attacker_ATK, string target_name, int target_HP)
    {
        // Caluculate
        target_HP -= attacker_ATK;

        // Output
        Debug.Log(attacker_name + "'s attack! " + attacker_ATK + " damege!");
        Debug.Log("Remaining HP of " + target_name + ": " + target_HP);

        return target_HP;
    }
```



However, still not Cool...

The third step of programming.



```
Unity スクリプト | 0 個の参照
public class CubeController : MonoBehaviour
{
    public float speed = 0.5f;

    // Define names as global scope because
    // names will be used in many functions.
    public string player_name = "Unity-chan";
    public string boss_name = "Cuctus-king";

    // Start is called before the first frame update
    Unity メッセージ | 0 個の参照
    void Start()
    {
        // Define variables
        int boss_HP = 100, boss_ATK = 20;
        int player_HP = 85, player_ATK = 25;

        // Call Attack function with variables
        boss_HP = Attack(player_name, player_ATK, boss_name, boss_HP);
        player_HP = Attack(boss_name, boss_ATK, player_name, player_HP);
        boss_HP = Attack(player_name, player_ATK, boss_name, boss_HP);
        player_HP = Attack(boss_name, boss_ATK, player_name, player_HP);
    }

    // Define Attack() function with variables
    4 個の参照
    int Attack(string attacker_name, int attacker_ATK, string target_name, int target_HP)
    {
        // Caluculate
        target_HP -= attacker_ATK;

        // Output
        Debug.Log(attacker_name + "'s attack! " + attacker_ATK + " damege!");
        Debug.Log("Remaining HP of " + target_name + ": " + target_HP);

        return target_HP;
    }
```

1 2 3

1. Rolls are not divided

- This script have both rolls of player and enemy.
  It's not good because complicated by their parameters.
- To clarify the rolls, we can create Player class and Enemy class. It's called Object-oriented thinking.

2. Manual calling

- Even now, we have to call Attack() repeatedly.
- It's seems good to use loop statement.
  Loop statement is described in next part.

3. Bug can be occurred

- Generally, HP should not become minus number.
- The function have to check target_HP before return statement. It's realized by conditional statements.

Speck change of Attack() function:

1. It check {target_HP} before return it.
2. If {target_HP} < 0, it will be set to 0.
3. It outputs "{target_name} died...".

Try to write If/Else statement

The third step of programming.

1. Fix the cube control.
    a. Edit CubeController according to this image. You don't have to write my comments.
    b. Save the file and back to the Unity.
    c. Play the game and check console.

Description.

- if (condition) { //programs }
    - Condition becomes true: programs are executed.
    - Condition becomes false: jump to else {}.
- else if (condition2) { //programs }
    - When you need more conditions after if statement, you can use else if statement.
- Conditional operators
    - (target_HP <= 0) : it becomes true when HP is 0
    - (!target_HP >= 0) : it's same as (target_HP < 0) "!" means reverse condition.
    - (target_HP != 0) means not 0. Equal is "==".

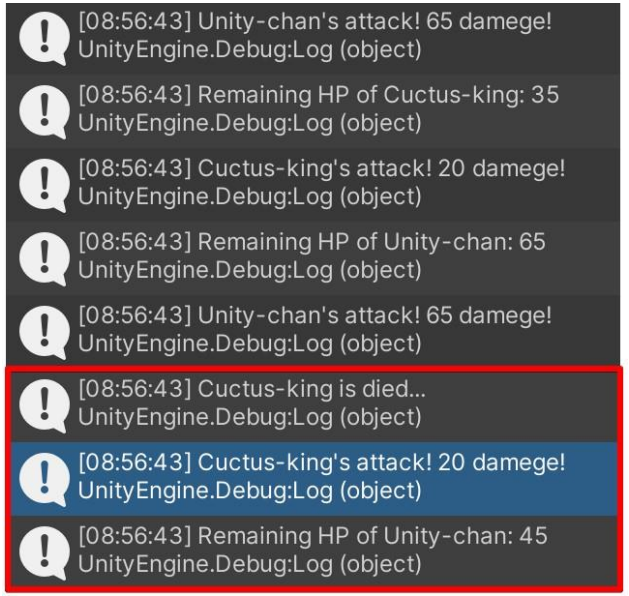
CubeController.cs

```
void Start()
{
    // Define variables
    int boss_HP = 100, boss_ATK = 20;
    int player_HP = 85, player_ATK = 65;

    // Call Attack function with variables
    boss_HP = Attack(player_name, player_ATK, boss_name, boss_HP);
    player_HP = Attack(boss_name, boss_ATK, player_name, player_HP);
    boss_HP = Attack(player_name, player_ATK, boss_name, boss_HP);
    player_HP = Attack(boss_name, boss_ATK, player_name, player_HP);
}

// Define Attack() function with variables
4 個の参照
int Attack(string attacker_name, int attacker_ATK, string target_name, int target_HP)
{
    // Caluculate
    target_HP -= attacker_ATK;

    // Output
    Debug.Log(attacker_name + "'s attack! " + attacker_ATK + " damege!");

    // Check HP
    if (target_HP < 0)
    {
        target_HP = 0;
        Debug.Log(target_name + " is died...");
        return 0;
    }
    else
    {
        Debug.Log("Remaining HP of " + target_name + ": " + target_HP);
        return target_HP;
    }
}
```

Try to write If/Else statement

The third step of programming.

Zombie attack!




```
// Define names as global scope because
// names will be used in many functions.
public string player_name = "Unity-chan";
public string boss_name = "Cuctus-king";

// Start is called before the first frame update
Unity メッセージ|0 個の参照
void Start()
{
    // Define variables
    int boss_HP = 100, boss_ATK = 20;
    int player_HP = 85, player_ATK = 65;

    // Call Attack function with variables
    boss_HP = Attack(player_name, player_ATK, boss_name, boss_HP);
    player_HP = Attack(boss_name, boss_ATK, player_name, player_HP);
    boss_HP = Attack(player_name, player_ATK, boss_name, boss_HP);
    player_HP = Attack(boss_name, boss_ATK, player_name, player_HP);
}

// Define Attack() function with variables
4 個の参照
int Attack(string attacker_name, int attacker_ATK, string target_name, int target_HP)
{
    // Caluculate
    target_HP -= attacker_ATK;

    // Output
    Debug.Log(attacker_name + "'s attack! " + attacker_ATK + " damege!");

    // Check HP
    if (target_HP < 0)
    {
        target_HP = 0;
        Debug.Log(target_name + " is died...");
        return 0;
    }
    else
    {
        Debug.Log("Remaining HP of " + target_name + ": " + target_HP);
        return target_HP;
    }
}
```

Exercise

1. How do you fix the bug with if (??? != ???) in Attack() ?
2. How do you fix the bug by defining boolean flag and if statement?

Try to write If/Else statement

The third step of programming.

```
public class CubeController : MonoBehaviour
```

Exercise

1. How do you fix the bug with if (??? != ???) in Attack() ?
2. How do you fix the bug by defining boolean flag and if statement?

Try to write If/Else statement

The third step of programming.

1. Add rotation key for cube.
    a. Edit CubeController according to this image. You don't have to write my comments.
    b. Save the file and back to the Unity.
    c. Play the game again.
        - Press 'Q' or 'E' key.

CubeController.cs

```
// Update is called once per frame
Unity メッセージ | 0 個の参照
void Update()
{
    // Variable definition for Input ( ← and → )
    float x_input = Input.GetAxis("Horizontal");

    // Variable definition for Input ( ↑ and ↓)
    float z_input = Input.GetAxis("Vertical");

    // Substitute transform.position(Vector3) + inputed value as Vector3
    transform.position += new Vector3(x_input, 0, z_input) * speed;

    if (Input.GetKey(KeyCode.Q))
    {
        Quaternion rotation = Quaternion.Euler(0, -2, 0);
        this.transform.rotation *= rotation;
    }

    if (Input.GetKey(KeyCode.E))
    {
        Quaternion rotation = Quaternion.Euler(0, 2, 0);
        this.transform.rotation *= rotation;
    }
}
```

Description.

- Input.Getkey()
    - It returns bool (true / false) by user's input.
    - The argument is string or KeyCode enum.
- KeyCode enum
    - It called enumerator to manage constants.
    - KeyCode enum has input related constant. e.g. "Space", "UpArrow" etc.
    - You can access the constants easily.

- Quaternion.Euler(x, y, z)
    - Quaternion is used to rotate objects.
    - Quaternion.Euler(0, -2, 0) means rotate -2 degree on Y axis.
    - "this" means the cube object.

Try to write Functions

The second step of programming.

1. Please translate chapter: Write your own instruction

   1. Find the instruction for potion spawn distance.

   to

   4. Check your script changes.

2. Please translate chapter: Introduction to functions

   1. Challenge: Create another potion at game start.

   to

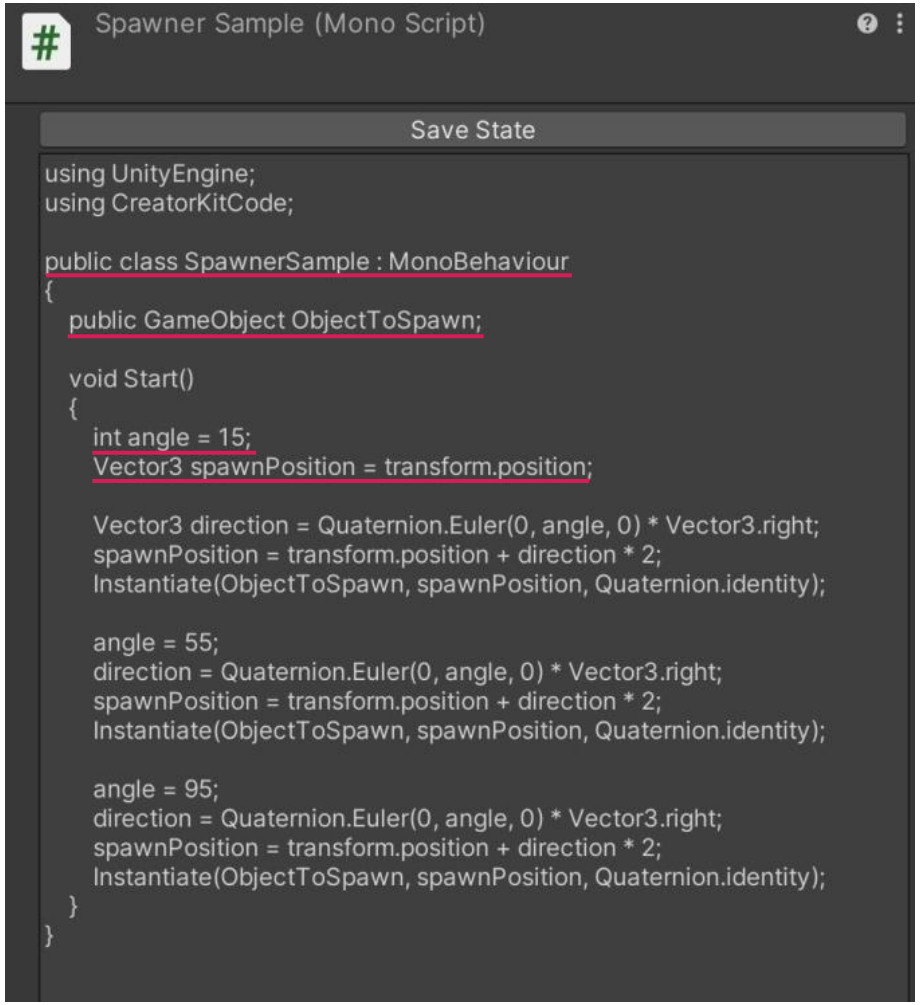
   8. Identify the function that already exists in your script.

3. Please translate chapter: Write your own spawn function

   1. Write a function declaration.

   to

   8. Check your function calls.



Try to write Functions

The second step of programming.

SpawnerSample.cs

```
Unity スクリプト | 0 個の参照
public class SpawnerSample : MonoBehaviour
{
    public GameObject ObjectToSpawn;

    Unity メッセージ | 0 個の参照
    void Start()
    {
        int angle = 15;
        int radius = 5;
        Vector3 spawnPosition = transform.position;

        Vector3 direction = Quaternion.Euler(0, angle, 0) * Vector3.right;
        spawnPosition = transform.position + direction * radius;
        Instantiate(ObjectToSpawn, spawnPosition, Quaternion.identity);

        angle = 55;
        direction = Quaternion.Euler(0, angle, 0) * Vector3.right;
        spawnPosition = transform.position + direction * radius;
        Instantiate(ObjectToSpawn, spawnPosition, Quaternion.identity);

        angle = 95;
        direction = Quaternion.Euler(0, angle, 0) * Vector3.right;
        spawnPosition = transform.position + direction * radius;
        Instantiate(ObjectToSpawn, spawnPosition, Quaternion.identity);
    }
}
```

```
public class SpawnerSample : MonoBehaviour

    // Attach portion object on inspector.
    public GameObject ObjectToSpawn;

    Unity メッセージ | 0 個の参照
    void Start()
    {
        SpawnPotion(0);
        SpawnPotion(45);
        SpawnPotion(90);
        SpawnPotion(135);
        SpawnPotion(180);
    }

    5 個の参照
    void SpawnPotion(int angle)
    {
        int radius = 5;

        Vector3 direction = Quaternion.Euler(0, angle, 0) * Vector3.right;
        Vector3 spawnPosition = transform.position + direction * radius;

        // Instantiate (Gameobject object, Vector3 position, Quatenion rotation)
        Instantiate(ObjectToSpawn, spawnPosition, Quaternion.identity);
    }
```

Attachment

This is important!

Coding Rules: Examples

Target	Major Rules	Examples
Class name	a. UpperCamel	a. SampleClass
Class variables	a. LowerCamel b. Snake_case c. Underscore	a. sampleVariable b. sample_variable c. _sampleVariable (Only for private variables)
Bool variables	a. Is_bool	a. is_run
Methods	a. UpperCamel	a. SampleMethod()
Constants	a. All-Caps b. UpperCamel	a. MAX_NUMBER b. MaxNum
Scene name	a. UpperCamel	a. TitleScene
Object name	a. UpperCamel	a. CactusNails
If Statement	a. if (target < condition)	a. if (num < 10)
Comments	You don't' need to write all of contents. Good code & Good name explains the role or function itself.	
Scope	Try to minimize the range.	
Hardcoding	Don't use it. You should use constants.	
Core mind	Simple and Dry. Functions should be under 30〜50 lines.	

Target	Pronunciation	Examples
()	Parenthesis	void Method()
[]	Bracket	int array[3]
{}	Brace	if (num < 10) { }

Attachment

This is important!

When you get Errors!

Check Points / ToDo	Details
Spell miss	Check your script carefully! We need to be case-sensitive in programming.
Error messages	Error messages may indicate the cause of the error or the corresponding line.
Search on Google	Type the error message into Google search. You may find the solutions. Sometimes, the errors / cautions can be ignored.
Narrow down the cause	Use comment out or Debug.Log("") wisely to find the problem.
Attached components	Check inspector on the GameObjects. You may forgot attach the components.
Careless miss	• Forgotten to save the scripts • Target GameObject was disabled • Log message was disabled
When you ask it…	When you ask someone the problem, make sure you clarify the followings. • **What problems happened?** • **When the problems happened?** • **Were you able to identify where is problem?** • **How did you try to solve it so far?** • **What is your goal?**

2.Basic programming with Unity

Class Plan

1. Introduction
2. Hello world in Unity
3. **Basic programming**
4. Object oriented programming
5. Interface programming
6. Programming design 1
7. Programming design 2
8. Improvement and testing 1
9. Improvement and testing 2
10. Summary

3
Basic Programming

3. Basic programming

Class summary

1. Explain Loop statements.
2. Explain Arrays.
3. Explain Enumerators.
4. Improve Usability.
5. Attach Animations.
6. Explain Classes.

Try to write Loop statement

The fourth step of programming.

SpawnerSample.cs

```
ublic class SpawnerSample : MonoBehaviour

    // Attach portion object on inspector.
    public GameObject ObjectToSpawn;

    Unity メッセージ | 0 個の参照
    void Start()
    {
        SpawnPotion(0);
        SpawnPotion(45);
        SpawnPotion(90);
        SpawnPotion(135);
        SpawnPotion(180);
    }

    5 個の参照
    void SpawnPotion(int angle)
    {
        int radius = 5;

        Vector3 direction = Quaternion.Euler(0, angle, 0) * Vector3.right;
        Vector3 spawnPosition = transform.position + direction * radius;

        // Instantiate (Gameobject object, Vector3 position, Quatenion rotation)
        Instantiate(ObjectToSpawn, spawnPosition, Quaternion.identity);
    }
```

Try to write Loop statement

The fourth step of programming.

1. Optimize the spawn code.
    a. Edit SpawnerSample according to this image. You don't have to write my comments.
    b. Save the file and play the game.

SpawnerSample.cs

```
Unity メッセージ | 0 個の参照
void Start()
{
    // Repeat procotol from 0 to 4
    for(int i = 0; i < 5; i++)
    {
        // Pass (index * 45) as angle
        SpawnPotion(i * 45);

        // i++ will be executed
        // It means: i = i + 1
    }
}
```

Description: for(A; B; C) statement

- A: Define a variable for loop condition
    - Often used 'i', 'j' and 'k' when we need nested loop.
    - Often set 0 because it's useful when handle arrays.
- B: Loop condition
    - It checked before execute instructions.
    - When true, inside one will be executed.
- C: Prepare for next loop.
    - It executed when this loop reach to end.
    - Often used "i++" (increment) or "i--" (decrement).
    - Be careful to avoid infinite loop. It cause freeze.

```
for(int i = 0; i < 36; i++)
{
    SpawnPotion(i * 10);
}
```

Try to write Loop statement

The fourth step of programming.

1. Expand the spawn code.
    a. Edit SpawnerSample according to this image. You don't have to write my comments.
    b. Save the file and play the game.

Description: Array and foreach

- type[] {array_name} = new type[num]
    - Array is container to manage variables.
    - You can access array like array_name[num].
    - It can be nested like if or loop statements. It called multidimensional array.
- type[] {array_name} = new type[] {num1, num2…}
    - Declare array with the data.
- foreach(type {variable_name} in {array_name})
    - Access each factor in array to the end.
    - It's useful because we don't have to know the index.
    - You can use break; (exit the loop) and continue; (jump to next loop) same as for-loop statement.

SpawnerSample.cs

```
Unity メッセージ | 0 個の参照
void Start()
{
    // Declare int type Array can contain 5 factors
    int[] angleArray = new int[5];

    // Substitute int values to the Array
    angleArray[0] = 0;
    angleArray[1] = 10;
    angleArray[2] = 50;
    angleArray[3] = 125;
    angleArray[4] = 220;

    // Use all factors of angleArray
    // {array}.Length returns the total number of the factors
    for (int i = 0; i < angleArray.Length; i++)
    {
        // Use i as index of angleArray
        SpawnPotion(angleArray[i]);
    }
}
```

```
Unity メッセージ | 0 個の参照
void Start()
{
    // Define int type Array can contain 5 factors
    int[] angleArray = new int[] {0, 10, 50, 125, 220};

    // Use all factors of angleArray
    foreach(int angle in angleArray)
    {
        SpawnPotion(angle);
    }
}
```

Implement Jump key

To improve your programming skills.

1. Expand the cube function.
    a. Edit CubeController according to this image. You don't have to write my comments.
    b. Save the file and play the game.

CubeController.cs

```
// Update is called once per frame
Unity メッセージ | 0 個の参照
void Update()
{
    // Variable definition for Input ( ← and → )
    float x_input = Input.GetAxis("Horizontal");

    // Variable definition for Input ( ↑ and ↓)
    float z_input = Input.GetAxis("Vertical");

    // Substitute transform.position(Vector3) + inputed value as Vector3
    transform.position += new Vector3(x_input, 0, z_input) * speed;

    if (Input.GetKey(KeyCode.Q))
    {
        Quaternion rotation = Quaternion.Euler(0, -2, 0);
        this.transform.rotation *= rotation;
    }

    if (Input.GetKey(KeyCode.E))
    {
        Quaternion rotation = Quaternion.Euler(0, 2, 0);
        this.transform.rotation *= rotation;
    }

    if (Input.GetKeyUp(KeyCode.Space))
    {
        this.transform.position += new Vector3(0, 1, 0);
    }
}
```

```
namespace UnityEngine
{
    public enum KeyCode
    {
        None = 0,
        Backspace = 8,
        Tab = 9,
        Clear = 12,
        Return = 13,
        Pause = 19,
        Escape = 27,
        Space = 32,
        Exclaim = 33,
        DoubleQuote = 34,
        Hash = 35,
        Dollar = 36,
        Percent = 37,
        Ampersand = 38,
        Quote = 39,
        LeftParen = 40,
        RightParen = 41,
        Asterisk = 42,
        Plus = 43,
        Comma = 44,
        Minus = 45,
```

Description: enum and GetKeyUp

- enum {enum_name} { }
    - It is container to handle const variables.
    - Often used to create {Days} and {Colors} etc.
    - {Days} enum should have "Monday", "Tuesday"...
    - {Colors} enum should have "Red", "Green", "Blue"...
    - Each element will be set the data as from int 0 by dafault;
    - You can set the data manually.
- Intput.GetKeyUP()
    - This function will be called only once.
    - If you use GetKey(), the cube can be fly while space key has been pressed.

Improve Usability

To improve your programming skills.

3 type of movement

2. Set rigidbody.velocity

- Simple movement with physics. It is located in midlle of real-phisics and game-phisics like double-jump.
- rigidbody.velocity = new Vector3(1, 0, 0) * speed

1. Set transform.position

- Most simple “teleport” movement without physics. It cannot express acceleration.
- transform.position += new Vector3(1, 0, 0)

When use phisics, the instructions sould be written in FixedUpdate() method instead of Update() exept Input.

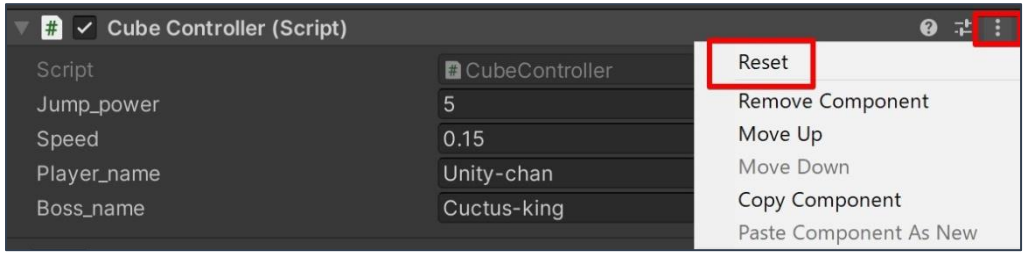
3. Use rigidbody.Addforce()

- Most realistic movement with physics. However, to handle it well, we need knowledge of phisics.
- rigidbody.Addforce(new Vector3(1, 0, 0))

Improve Usability

To improve your programming skills.

1. Improve the jump function.
   a. Edit CubeController according to this image. ⟶
      You don't have to write my comments.
   b. Save the file.
   c. Select Cube object on Hierarchy.
   d. Select "Reset" to adapt variable settings.



   e. Play the game.

FixedUpdate() method

- It called before caluclate phisics.
- It called constantly regardless of the frame rate so it's not suitable for Input related instructions.

CubeController.cs

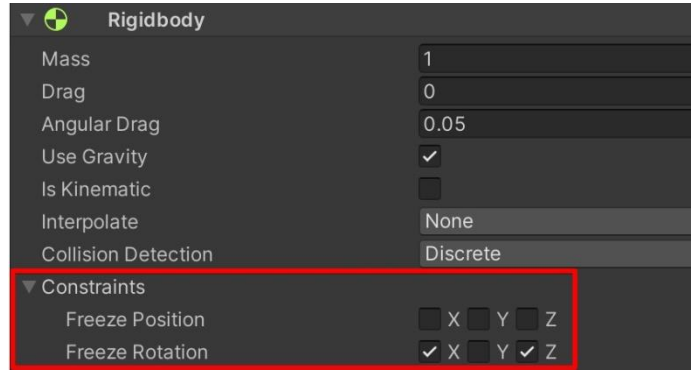
```
public class CubeController : MonoBehaviour
{
    // Declear Rigidbody variable
    Rigidbody rb = null;

    public float jump_power = 5.0f;
    public float speed = 0.15f;

    // Define names as global scope because
    // names will be used in many functions.
    public string player_name = "Unity-chan";
    public string boss_name = "Cuctus-king";
```

```
    void Start()
    {
        // Set Rigidbody to rb
        rb = this.GetComponent<Rigidbody>();

        // Define variables
        int boss_HP = 100, boss_ATK = 20;
        int player_HP = 85, player_ATK = 65;
```

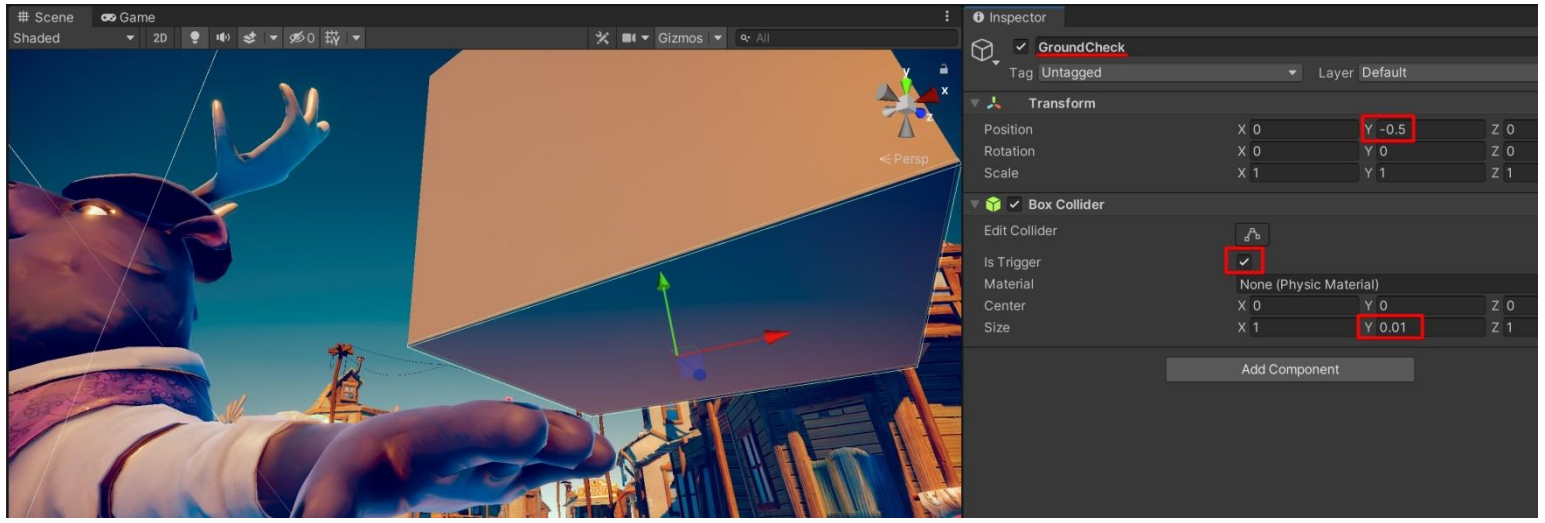
```
void Update()
{
    // Variable definition for Input ( ← and → )
    float x_input = Input.GetAxis("Horizontal");

    // Variable definition for Input ( ↑ and ↓)
    float z_input = Input.GetAxis("Vertical");

    // Substitute transform.position(Vector3) + inputed value as Vector3
    transform.position += new Vector3(x_input, 0, z_input) * speed;

    if (Input.GetKey(KeyCode.Q))
    {
        Quaternion rotation = Quaternion.Euler(0, -2, 0);
        this.transform.rotation *= rotation;
    }

    if (Input.GetKey(KeyCode.E))
    {
        Quaternion rotation = Quaternion.Euler(0, 2, 0);
        this.transform.rotation *= rotation;
    }

    if (Input.GetKeyUp(KeyCode.Space))
    {
        rb.velocity = new Vector3(0, jump_power, 0);
    }
}
```

Improve Usability

To improve your programming skills.



1. Prevent jump related glitches (Infinite jump).
    a. Prevent tumble by adding constraints on X and Z axis. ⟶
    Select Cube object and Check “Freeze Rotation” on Rigidbody.
    b. Create Empty object as “GroundCheck” below Cube object.
    (Right click Cube -> “Create Empty” -> Rename it)
    c. Attach Box Collider to GroundCheck.
    (Select GroundCheck -> Add Component -> Box Collider)
    d. Edit GroundCheck’s position and collider size.
    Move it to bottom of the Cube like thin plane and check “Is Trigger”.



Collider

- Component for hit detection.
- When “Is Trigger” is checked, it’s not collide with other objects.

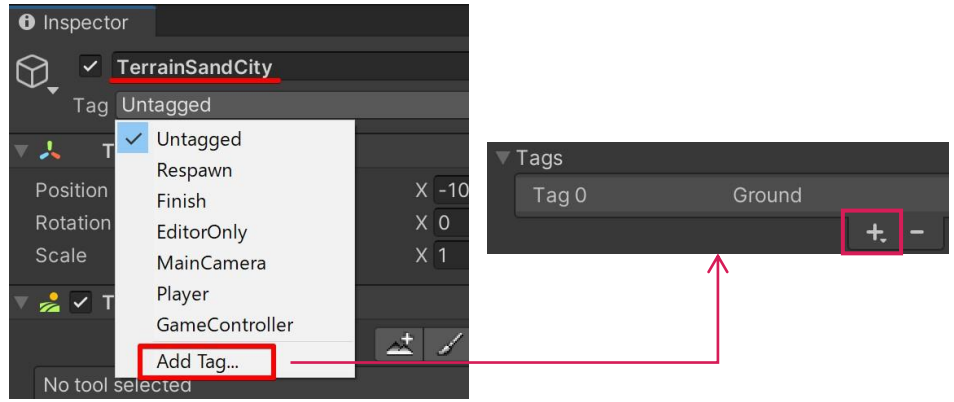
Improve Usability

To improve your programming skills.

1. Prevent jump related glitches (Infinite jump).
    e. Edit CubeController according to this image.
    You don't have to write my comments.
    f. Save the file and play the game to check console logs.
    g. Select Ground object named "TerrainSandCity".
    h. Select "Add Tag" -> + button
    -> Input Tag name as "Ground" on Inspector.
    i. Select Ground object named "TerrainSandCity" again.
    j. Attach "Ground" tag to "TerrainSandCity".
    k. Edit CubeController according to this image.
    You should delete or comment out
    OnTriggerStay() because it dump a lot of logs.
    l. Play the game and check console logs.

CubeController.cs

```
// When the Cube collide with something
Unity メッセージ | 0 個の参照
private void OnTriggerEnter(Collider other)
{
    Debug.Log("Enter!");
}

Unity メッセージ | 0 個の参照
private void OnTriggerStay(Collider other)
{
    Debug.Log("Stay!");
}

Unity メッセージ | 0 個の参照
private void OnTriggerExit(Collider other)
{
    Debug.Log("Exit!");
}
```




```
Unity メッセージ | 0 個の参照
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Ground"))
    {
        Debug.Log("Enter the Ground!");
    }
}
```

OnTriggerEnter() method

- It called when the object collide with other colliders.
- Variable: other has collided object's collider.
- You can check the collider tag with CompareTag().

Improve Usability

To improve your programming skills.

1. Prevent jump related glitches (Infinite jump).
    m. Edit CubeController according to these images.
    n. Save the file and play the game to check console logs.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CubeController : MonoBehaviour
{
    Rigidbody rb = null;
    public float jump_power = 5.0f;
    public float speed = 0.15f;
    bool is_ground = false;

    void Start()
    {
        rb = this.GetComponent<Rigidbody>();
    }

    void Update()
    {
        float x_input = Input.GetAxis("Horizontal");
        float z_input = Input.GetAxis("Vertical");
        transform.position += new Vector3(x_input, 0, z_input) * speed;

        if (Input.GetKey(KeyCode.Q))
        {
            Quaternion rotation = Quaternion.Euler(0, -2, 0);
            this.transform.rotation *= rotation;
        }

        if (Input.GetKey(KeyCode.E))
        {
            Quaternion rotation = Quaternion.Euler(0, 2, 0);
            this.transform.rotation *= rotation;
        }

        if (Input.GetKeyUp(KeyCode.Space) && is_ground)
        {
            rb.velocity = new Vector3(0, jump_power, 0);
        }
    }
```

```
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Ground"))
        {
            Debug.Log("Enter the Ground!");
            is_ground = true;
        }
    }

    private void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Ground"))
        {
            Debug.Log("Exit the Ground!");
            is_ground = false;
        }
    }

}
```

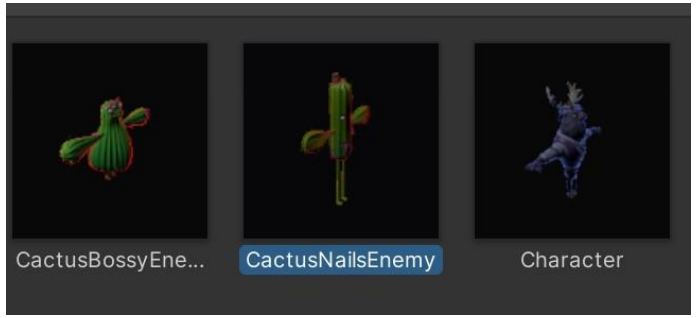
Current source code
without Attack() related things.

```
Unity スクリプト|0 個の参照
public class CubeController : MonoBehaviour
{
    Rigidbody rb = null;

    public float jump_power = 5.0f;
    public float speed = 0.15f;

    public string player_name = "Unity-chan";
    public string boss_name = "Cuctus-king";

    bool dead_flag = false;
    bool is_ground = false;
```

```
void Update()
{
    // Variable definition for Input ( ← and → )
    float x_input = Input.GetAxis("Horizontal");

    // Variable definition for Input ( ↑ and ↓)
    float z_input = Input.GetAxis("Vertical");

    transform.position += new Vector3(x_input, 0, z_input) * speed;

    if (Input.GetKey(KeyCode.Q))
    {
        Quaternion rotation = Quaternion.Euler(0, -2, 0);
        this.transform.rotation *= rotation;
    }

    if (Input.GetKey(KeyCode.E))
    {
        Quaternion rotation = Quaternion.Euler(0, 2, 0);
        this.transform.rotation *= rotation;
    }

    if (Input.GetKeyUp(KeyCode.Space) && is_ground)
    {
        rb.velocity = new Vector3(0, jump_power, 0);
    }
}

Unity メッセージ|0 個の参照
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Ground"))
    {
        Debug.Log("Enter the Ground!");
        is_ground = true;
    }
}

Unity メッセージ|0 個の参照
private void OnTriggerExit(Collider other)
{
    if (other.CompareTag("Ground"))
    {
        Debug.Log("Exit the Ground!");
        is_ground = false;
    }
}
```

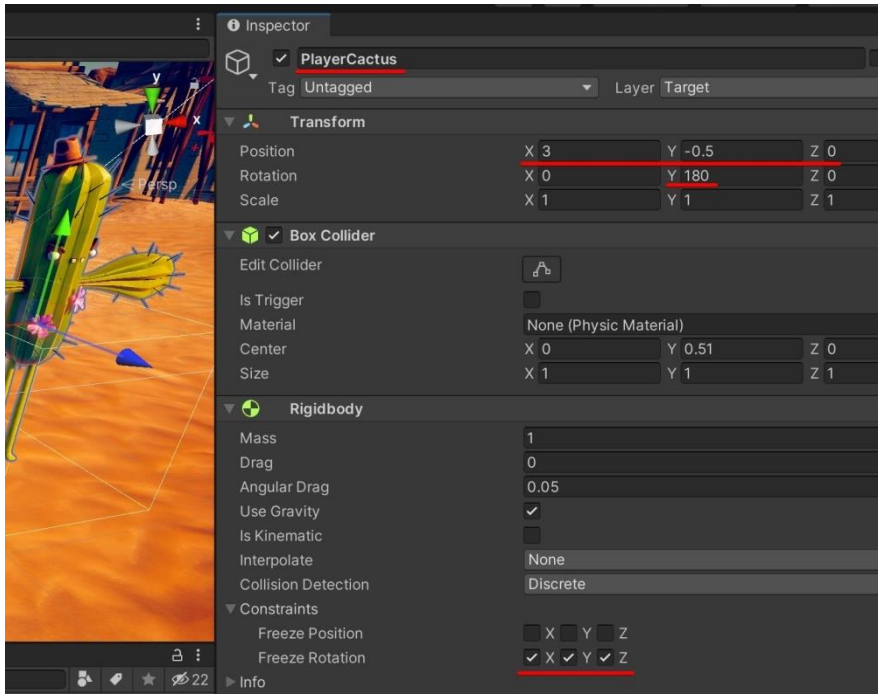
Update Cube into Cactus

Exercise



1. Drag & Drop a prefab of “CactusNailsEnemy” into Hierarchy. ⟶
2. Right click “CactusNailsEnemy” -> Prefab -> Unpack Completely.
3. Rename it to “PlayerCactus”.
4. Remove below components with top-right ⋮ button on PlayerCactus.
    - Nav Mesh Agent
    - Simple Enemy Controller
    - Character Data
    - Character Audio
5. Add Rigidbody and set parameters according to this image. ⟶
6. Create “CactusController” script on script/tutorial folder.
7. Edit CactusController script according to CubeController.

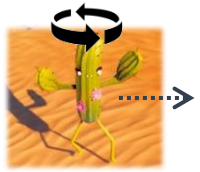
   You don’t need to write Attack() related code. Use copy & paste.
8. Drag & Drop CactusController into PlayerCactus as component.
9. Play the game and check movement of Cactus.



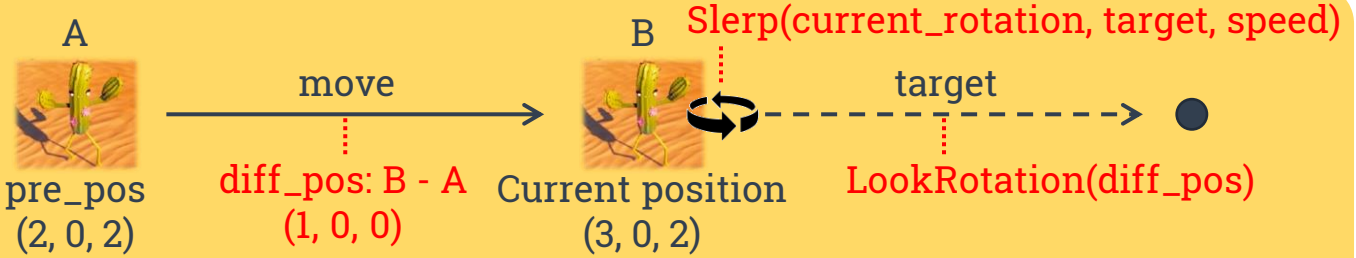
Update Cube into Cactus

Exercise

10. Attach GroundCheck as Empty Object on Hierarchy under CactusNails.
11. Attach Box collider to GroundCheck and move & resize it to the bottom.
12. Check “Is Trigger” on GroundCheck’s Box collider.
13. Play the game and check movement & jump of Cactus.
    - It needs rotation to look forward when you walk.



14. Edit CactusController according to this image and play the game. →



Quaternion.LookRotation(Vector3 forward) :

- Returns Quaternion (≒ rotation) to forward position.

Quaternion.Slerp(Quaternion a, Quaternion b, float t) :

- Returns Quat. as result of spherically interpolates between a and b by t.

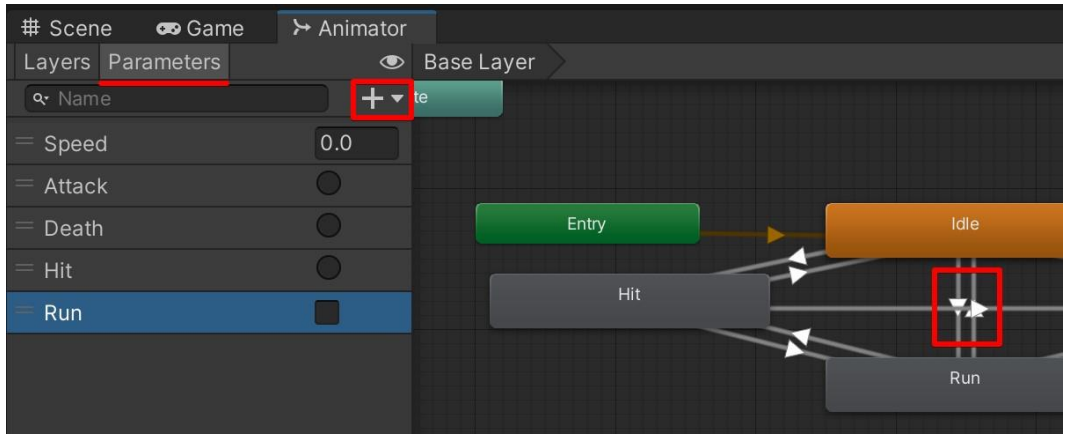
CactusController.cs

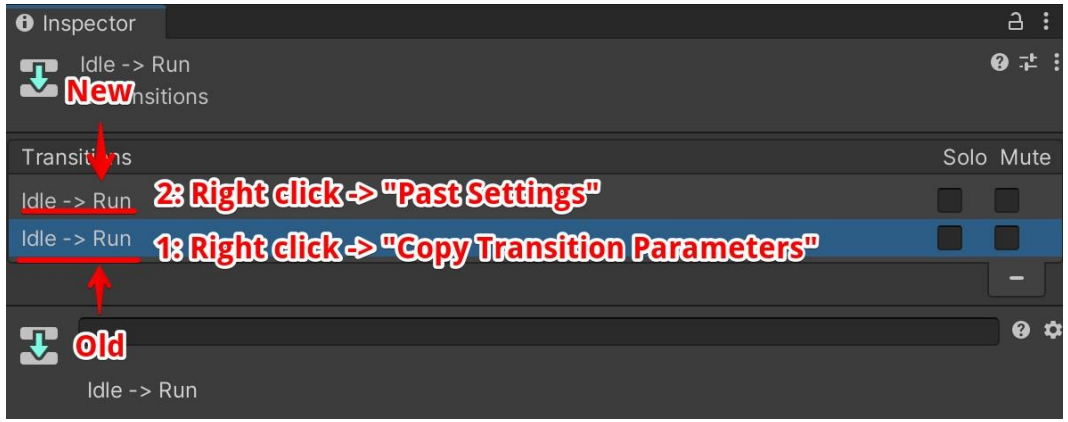
```
Unity スクリプト|0 個の参照
public class CactusController : MonoBehaviour
{
    Rigidbody rb = null;

    public float jump_power = 5.0f;
    public float speed = 0.15f;

    bool is_ground = false;
    Vector3 pre_pos;

    // Start is called before the first frame update
    Unity メッセージ|0 個の参照
    void Start()
    {
        // Set Rigidbody to rb
        rb = this.GetComponent<Rigidbody>();

        // Set Start position
        pre_pos = this.transform.position;
    }

    // Update is called once per frame
    Unity メッセージ|0 個の参照
    void Update()
    {
        // Set current position
        pre_pos = this.transform.position;

        // Variable definition for Input ( ← and → )
        float x_input = Input.GetAxis("Horizontal");

        // Variable definition for Input ( ↑ and ↓)
        float z_input = Input.GetAxis("Vertical");

        transform.position += new Vector3(x_input, 0, z_input) * speed;

        // Calculate difference of position from previous it.
        Vector3 diff_pos = this.transform.position - pre_pos;

        if (diff_pos.magnitude > 0.01f)
        {
            Quaternion target = Quaternion.LookRotation(diff_pos);
            this.transform.rotation =
                Quaternion.Slerp(this.transform.rotation, target, speed);
        }
```

Attach Animations

Exercise

1. Open Animator view by double click “Art/Animations/CactusNailsController” on Project.
2. Select “Parameters” and + button.
   Create “Run” parameter as bool type.
3. Select ↓ to set transition trigger on Idle -> Run.
   However, It already has Speed trigger… Set more way.
   Right click “Idle” statement -> “Make Transition”.
   Then, select “Run” statement to add new transition.
4. Select updated ↓ and copy & paste the setting from old to new one (Upper one) according this image.
5. Select + button on Conditions.
   Add Run condition as true.





Attach Animations

Exercise

7. Set a transition from Run to Idle too.
   Right click “Run” statement -> “Make Transition”.
   Then, select “Idle” statement to add new transition.
8. Select updated ▲▲ and copy & paste the setting from old to new one (Upper one) according this image.
9. Select ＋ button on Conditions.
   Add Run condition as false.
10. Edit CactusController according to this image. ⟶
11. Play the game. (You can delete Cude)

transform.Find(string name) :

- Returns the transform of child’s objects by the name.

Vector3.magnitude:

- Returns the Vector3’s length: root of ($x^2 + y^2 + z^2$).

CactusController.cs

```
Unity スクリプト|0 個の参照
public class CactusController : MonoBehaviour
{
    Rigidbody rb = null;
    Animator anim = null;

    public float jump_power = 5.0f;
    public float speed = 0.15f;

    bool is_ground = false;
    Vector3 pre_pos;

    // Start is called before the first frame update
    Unity メッセージ|0 個の参照
    void Start()
    {
        // Set Rigidbody to rb
        rb = this.GetComponent<Rigidbody>();

        // Set Start position
        pre_pos = this.transform.position;

        anim = this.transform.Find("CactusNails").gameObject.GetComponent<Animator>();
    }

    // Update is called once per frame
    Unity メッセージ|0 個の参照
    void Update()
    {
        // Set current position
        pre_pos = this.transform.position;

        // Variable definition for Input ( ← and → )
        float x_input = Input.GetAxis("Horizontal");

        // Variable definition for Input ( ↑ and ↓)
        float z_input = Input.GetAxis("Vertical");

        transform.position += new Vector3(x_input, 0, z_input) * speed;

        // Calculate difference of position from previous it.
        Vector3 diff_pos = this.transform.position - pre_pos;

        if (diff_pos.magnitude > 0.01f)
        {
            Quaternion target = Quaternion.LookRotation(diff_pos);
            this.transform.rotation =
                Quaternion.Slerp(this.transform.rotation, target, speed);
            anim.SetBool("Run", true);
        } else
        {
            anim.SetBool("Run", false);
        }
```

Understand basic of Class

Class as object

- Class is blueprint of scripts.
  -> Declare and implement the factors.
  If declare only, use abstract syntax.
- Class can inherit variables and functions from the parent.
  -> When inherited factors are implemented on the parent class, it can be used without implement in child class. e.g. Move()

Class Variables / Member variables / Fields:

- string name;
- int HP, ATK, SPD;

Class Functions / Methods:

- void Attack()
- void Die()
- void Move() { … // Instructions … }

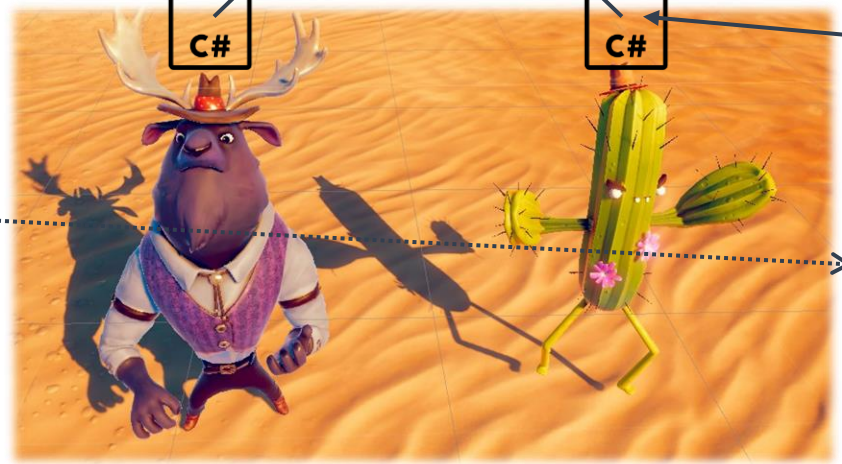
Character class

Constructor

- Method with same name as the class name.
- It's used to set default parameters.

name = MikeMoose;
HP = 20;
ATK = 5;
SPD = 5;

Player(){}
void Attack() {}
void Die() {}
~~void Move(){}~~
void GetItem(){}



Player class

EnemyA class

name = CactusNails;
HP = 11;
ATK = 2;
SPD = 3;
EXP = 2;

EnemyA(){}
void Attack() {}
void Die() {}
~~void Move(){}~~
void DropItem(){}

EnemyABoss class

….
HP = 50;
ATK = 8;
SPD = 1;
EXP = 25;

…

Try to write Class

Architecture of script.

1. Please translate chapter: Introduction to classes

   2. Identify the level above functions

   to

   6. Using class as a type

2. Please translate chapter: Write your own spawner class

   1. Using classes to make your code more efficient

   to

   12. Test your changes

3. Please translate chapter: Introduction to the public keyword and inheritance

   1. Understand the difference between public and private

   to

   11. What does the MonoBehaviour class contain?

Coding Rules: Examples

Target	Major Rules	Examples
Class name	a. UpperCamel	a. SampleClass
Class variables	a. LowerCamel b. Snake_case c. Underscore	a. sampleVariable b. sample_variable c. _sampleVariable (Only for private variables)
Bool variables	a. Is_bool	a. is_run
Methods	a. UpperCamel	a. SampleMethod()
Constants	a. All-Caps b. UpperCamel	a. MAX_NUMBER b. MaxNum
Scene name	a. UpperCamel	a. TitleScene
Object name	a. UpperCamel	a. CactusNails
If Statement	a. if (target < condition)	a. if (num < 10)
Comments	You don't' need to write all of contents. Good code & Good name explains the role or function itself.	
Scope	Try to minimize the range.	
Hardcoding	Don't use it. You should use constants.	
Core mind	Simple and Dry. Functions should be under 30～50 lines.	

Attachment

This is important!

When you get Errors!

Check Points / ToDo	Details
Spell miss	Check your script carefully! We need to be case-sensitive in programming.
Error messages	Error messages may indicate the cause of the error or the corresponding line.
Search on Google	Type the error message into Google search. You may find the solutions. Sometimes, the errors / cautions can be ignored.
Narrow down the cause	Use comment out or Debug.Log("") wisely to find the problem.
Attached components	Check inspector on the GameObjects. You may forgot attach the components.
Careless miss	• Forgotten to save the scripts • Target GameObject was disabled • Log message was disabled
When you ask it…	When you ask someone the problem, make sure you clarify the followings. • **What problems happened?** • **When the problems happened?** • **Were you able to identify where is problem?** • **How did you try to solve it so far?** • **What is your goal?**

2.Basic programming with Unity

Class Plan

1. Introduction
2. Hello world in Unity
3. Basic programming
4. **Object oriented programming**
5. Interface programming
6. Programming design 1
7. Programming design 2
8. Improvement and testing 1
9. Improvement and testing 2
10. Summary

4
Object oriented programming

4. Object oriented programming

Class summary

1. Try to Refactoring.
2. Implement Attack() with Raycast.
3. Complete Unity Tutorials.
4. Explain Encapsulation.
5. Explain Polymorphism.
6. Explain Object oriented programming.
7. Set up Tools for class diagram.

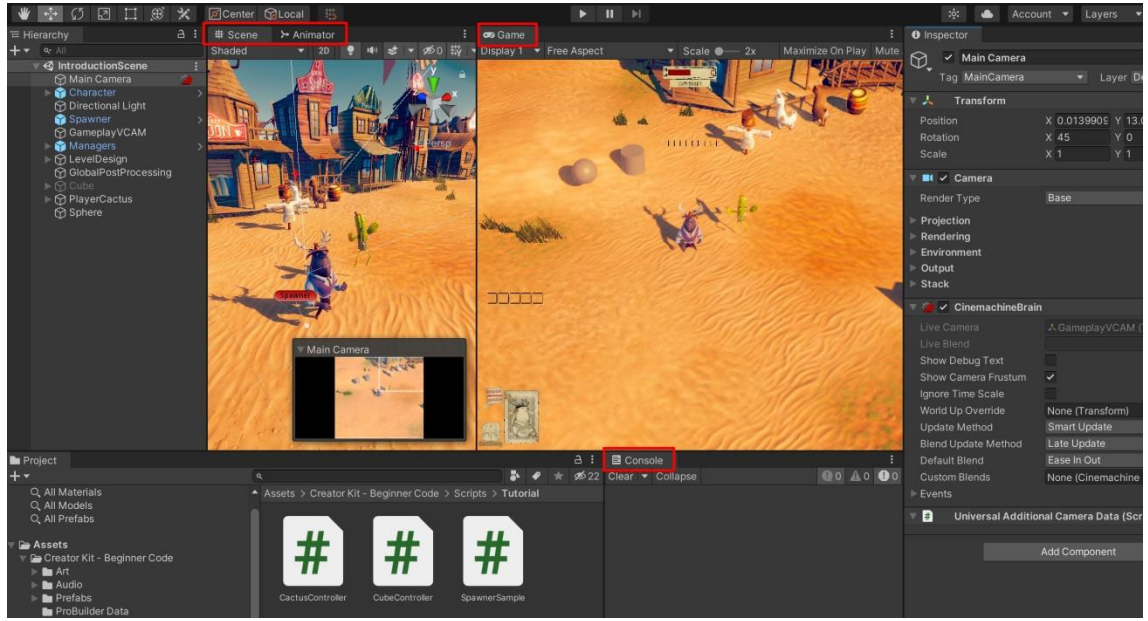
Coding Rules: Examples

Target	Major Rules	Examples
Class name	a. UpperCamel	a. SampleClass
Class variables	a. LowerCamel b. Snake_case c. Underscore	a. sampleVariable b. sample_variable c. _sampleVariable (Only for private variables)
Bool variables	a. Is_bool	a. is_run
Methods	a. UpperCamel	a. SampleMethod()
Constants	a. All-Caps b. UpperCamel	a. MAX_NUMBER b. MaxNum
Scene name	a. UpperCamel	a. TitleScene
Object name	a. UpperCamel	a. CactusNails
If Statement	a. if (target < condition)	a. if (num < 10)
Comments	You don't' need to write all of contents. Good code & Good name explains the role or function itself.	
Scope	Try to minimize the range.	
Hardcoding	Don't use it. You should use constants.	
Core mind	Simple and Dry. Functions should be under 30〜50 lines.	

Try to Refactoring

To simple and dry.

1. Edit CactusController according to this image. ⟶
   a. Declare methods in Update()
   b. Create methods
   c. Copy and Paste according to the method name (No need to write TryToAttack() contents now)
   d. Play the game and check it works correctly.
2. (Try to adjust Unity Editor)



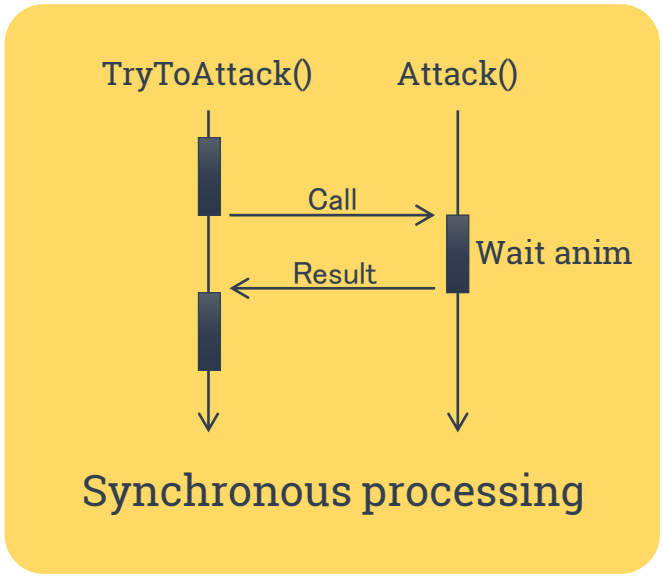

```
// Update is called once per frame
Unity メッセージ | 0 個の参照
void Update()
{
    Move();
    Rotate();
    TryToAttack();
}

1 個の参照
private void Move()
{
    pre_pos = this.transform.position;
    float x_input = Input.GetAxis("Horizontal");
    float z_input = Input.GetAxis("Vertical");
    transform.position += new Vector3(x_input, 0, z_input) * speed;
    Vector3 diff_pos = this.transform.position - pre_pos;

    if (diff_pos.magnitude > 0.01f)
    {
        Quaternion target = Quaternion.LookRotation(diff_pos);
        this.transform.rotation =
            Quaternion.Slerp(this.transform.rotation, target, speed);
        anim.SetBool("Run", true);
    }
    else
    {
        anim.SetBool("Run", false);
    }

    if (Input.GetKeyUp(KeyCode.Space) && is_ground)
    {
        rb.velocity = new Vector3(0, jump_power, 0);
    }
}

1 個の参照
private void Rotate()
{
    if (Input.GetKey(KeyCode.Q))
    {
        Quaternion rotation = Quaternion.Euler(0, -2, 0);
        this.transform.rotation *= rotation;
    }

    if (Input.GetKey(KeyCode.E))
    {
        Quaternion rotation = Quaternion.Euler(0, 2, 0);
        this.transform.rotation *= rotation;
    }
}
```

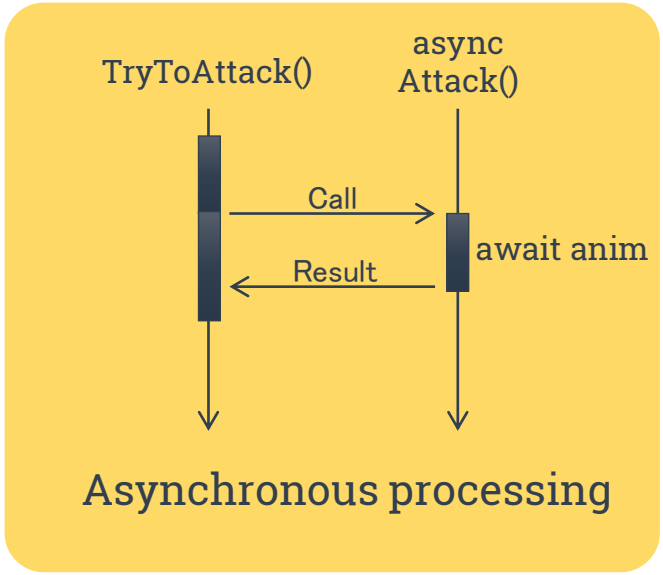
Implement Attack! (0/5)

With Ray

0. Define the flow of cactus' attack.

1. Check the cactus can attack or not.
   - Use Collider
   - ✔ Use Ray
2. Play attack animation.
   - Use Animator
3. Wait finishing attack animation.
   - ✔ Use async / await
   - Use Coroutine
4. Check the attack success or not.
   - Calculate distance
5. Deal damage to MikeMoose.
   - Use CharacterData.Stats
6. Set attack cooldown.





Implement Attack! (1/5)

With Ray

```
public int atk = 1;
public int atk_coldown = 1;
public float atk_range = 2.0f;
public bool is_attackable = true;
public AnimationClip attack_clip;
```

1. Add global variables for attack on CactusController .
2. Add TryToAttack() method according to this image.

```
private void TryToAttack()
{
    // Create Ray from current position + (0, 1, 0) to the forward direction
    Ray ray = new Ray(this.transform.position + Vector3.up, this.transform.forward);

    // Declare to get hit-target
    RaycastHit hit;

    // Set 10000000000000 as binary because LayerMask.NameToLayer("Player") == 13
    // It is same 8192 as decimal number(int)
    int targetLayerMask = 1 << LayerMask.NameToLayer("Player");

    if (Physics.Raycast(ray, out hit, atk_range, targetLayerMask) && is_attackable)
    {
        Debug.DrawRay(ray.origin, ray.direction * atk_range, Color.green, 5);
        Debug.Log("Attack!");
        // Next step↓
        // Attack(hit.collider.GetComponent<CharacterData>());
    }
}
```

1

Ray(Vector3 origin, Vector3 direction) :

- Creates a ray starting at origin along direction.
- Add Vector3.up: (0, 1, 0) to adjust ray's position to the eye.
- transform.forward equals (0, 0, 1).

RaycastHit:

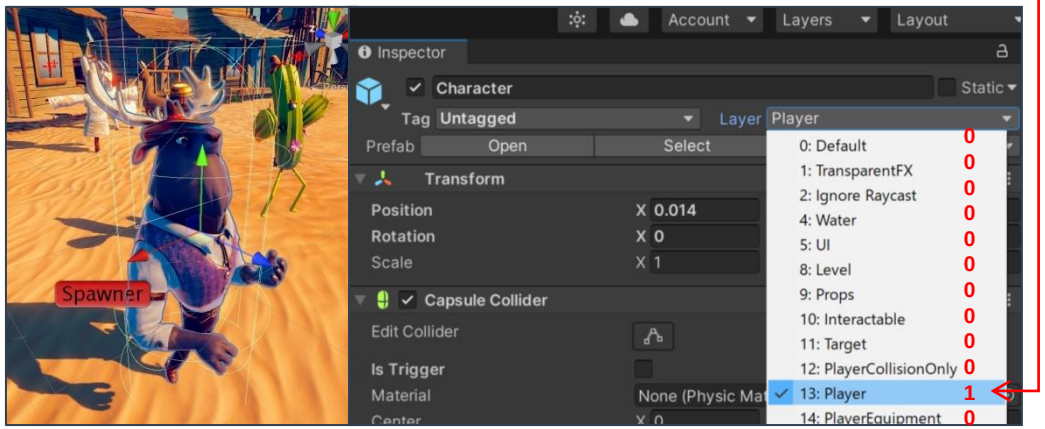
- Class to manage hit-object by the ray.
- It's often used with "out" operator to pass by refference.

Raycast(Ray ray, RaycastHit hit, float distance, int layerMask):

- Cast the ray with the passed parameters.
- You can access hit-object with the hit variable.

Debug.DrawRay(Vector3 start, Vector3 dir, Color color, float duration) :

- Visualize the ray with the passed parameters.



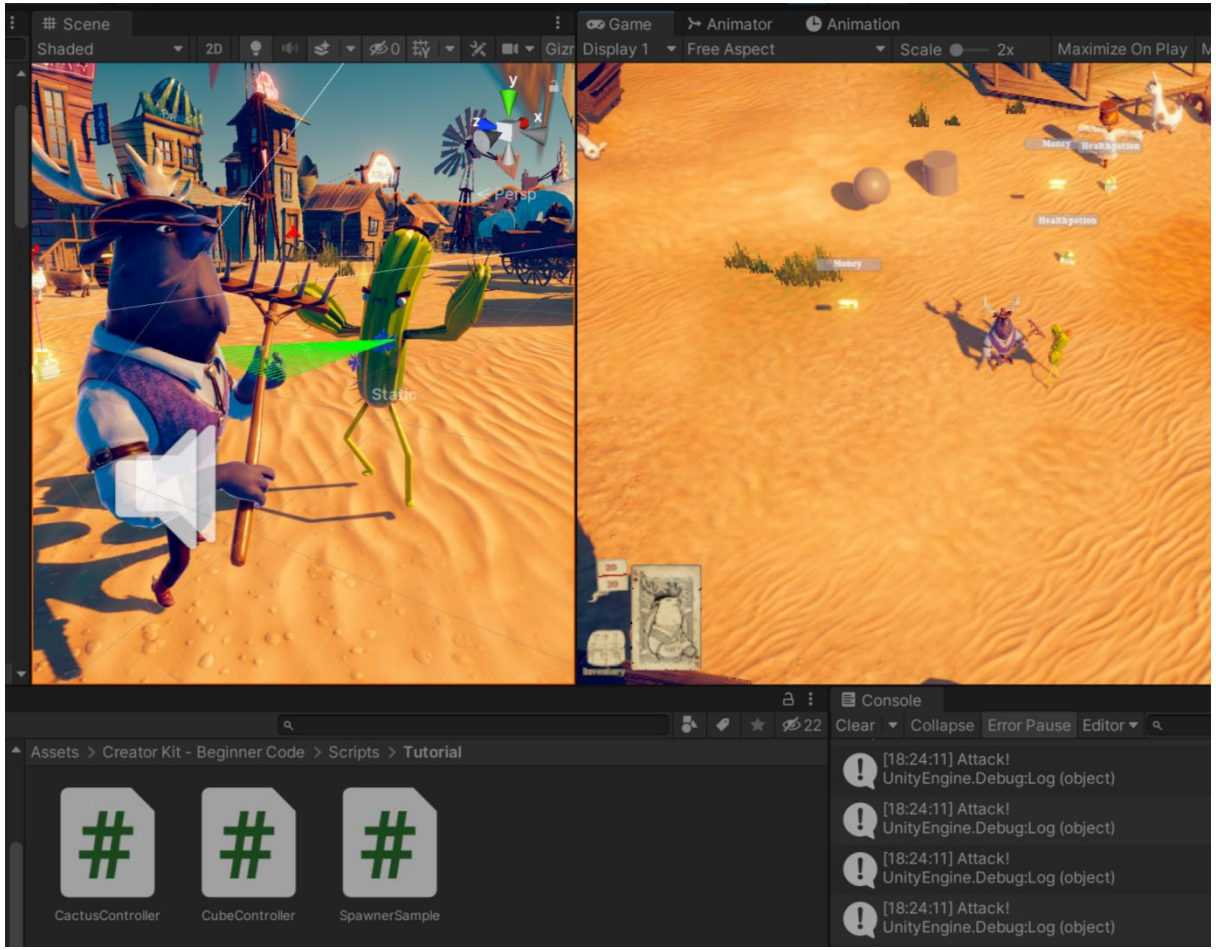
Shift Operator: {value} << {shift_number}

- Shift {value} with {shift_number} as binary.
- It's often used to manage enum contains.

Implement Attack! (2/5)

With Ray

3. Play the game and check the ray and console.



Ref: Pass by reference / value

Gif image is quoted from this web site.

Pass by reference:

- Creates copy but the changes will be adapt to the original.

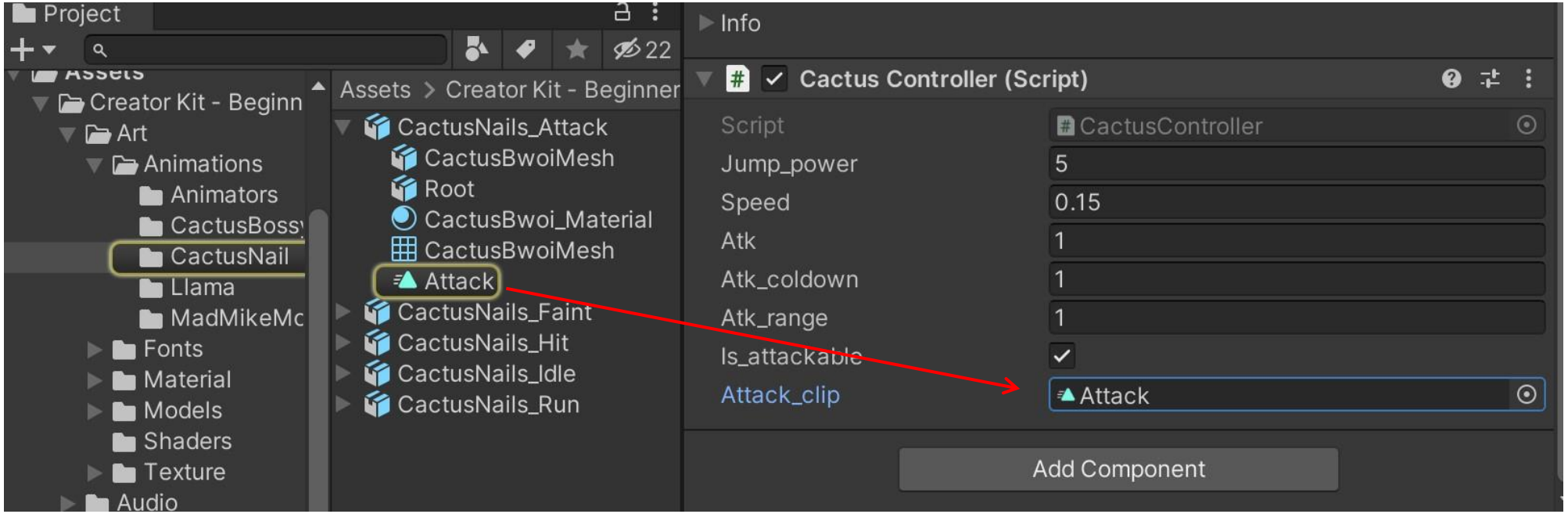
Pass by value:

- Creates copy. The changes will not be adapt to the original.

Implement Attack! (3/5)

With Ray

4. Attach animation clip of Attack to CactusController.
    a. Find Attack animation clip on Project.
    -> Art/Animations/CactusNail/CactusNails_Attack/Attack
    b. Select PlayerCactus on Hierarchy.
    c. Drag & Drop Attack animation clip to Attack_clip variable.



Implement Attack! (4/5)

With Ray

```
using System.Collections;
using System.Collections.Generic;
using System.Threading.Tasks;
using UnityEngine;
using CreatorKitCode;
```

5. Add using System.Threading.Tasks to top of CactusController.
6. Enable Attack(hit.collider.GetComponent<CharacterData>());
7. Edit CactusController according to this images.

```
Debug.DrawRay(ray.origin, ray.direction * atk_range, Color.green, 5);
Debug.Log("Attack!");
// Next step↓
Attack(hit.collider.GetComponent<CharacterData>());
```

```
private async void Attack(CharacterData target)
{
    // Play attack animation
    transform.Find("CactusNails").GetComponent<Animator>().Play("Attack");
    is_attackable = false;

    // Wait finishing attack motion
    await Task.Delay((int)(attack_clip.length * 1000));

    // Check target position before damage
    Vector3 diff_pos = target.transform.position - this.transform.position;
    float collider_offset = this.GetComponent<BoxCollider>().size.z / 2f;
    if (diff_pos.sqrMagnitude < atk_range * atk_range + collider_offset)
    {
        target.Stats.ChangeHealth(this.atk * -1);
    }

    // Set attack cooldown
    await Task.Delay(atk_coldown * 1000);
    is_attackable = true;
}
```

async void Attack() :

- async declares this function use await.
- await waits finishing following function.

Task.Delay(int milliseconds):

- Sleep the designated milliseconds.
- Attack_clip.length returns the duration as float so we need convert it to int type. (int) is doing that. The value will be rounded. e.g. (int)(0.35f * 1000) == 350, (int)0.7f == 0.

float collider_offset / Vector3.sqrMagnitude :

- collider_offset is just adjustment with collider's size.
- sqrMagnitude returns $x^2 + y^2 + z^2$: faster than using Magnitude.

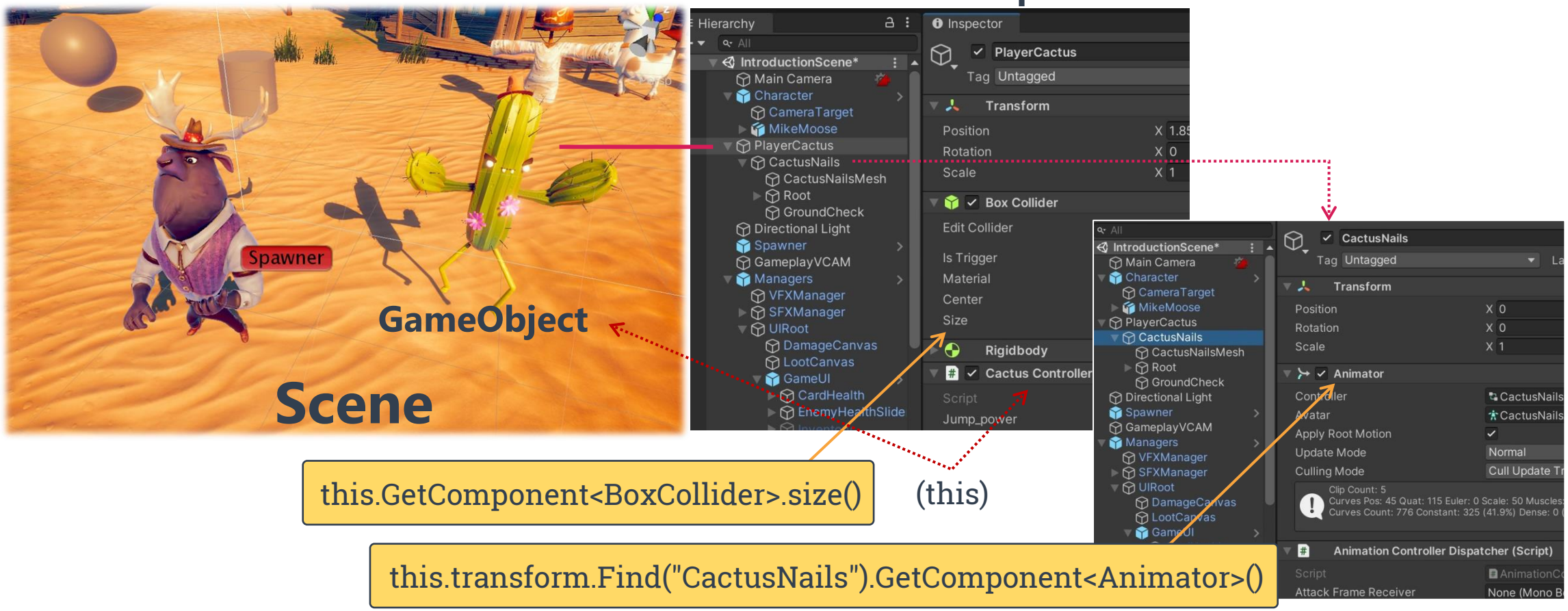
CharacterData.Stats.ChangeHealth(int amount):

- Implemented by BeginnerCode creator.
- Plus amount == heal, minus amount == damage.

Implement Attack! (5/5)

With Ray

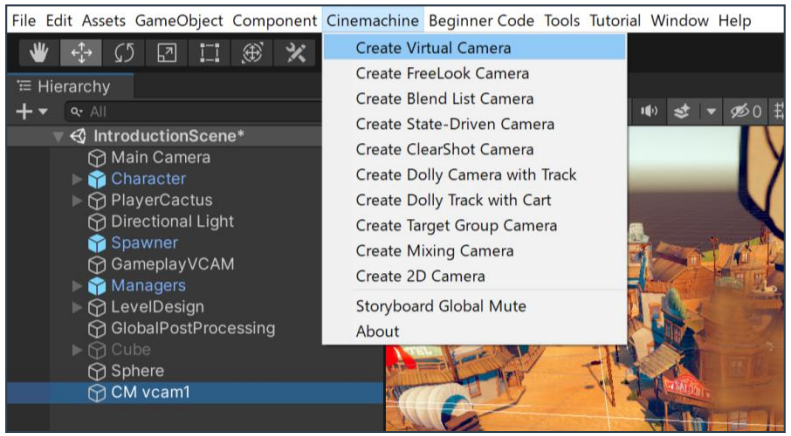
8. Play the game and check the attack.

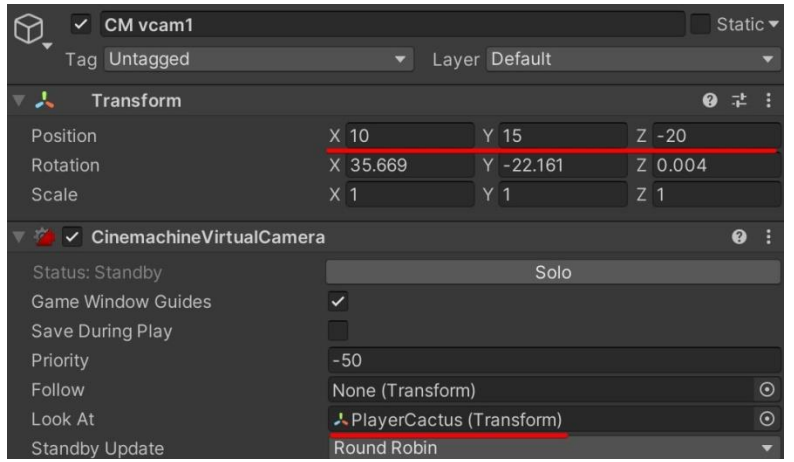


Adjustment Camera

With Cinemachine

1. Create Cinemachine camera according to this image. →
2. Select “CM vcam1” and set the parameters and drag & drop PlayerCactus object according to this image. →
3. Create CameraController script in scripts/tutorial folder.
4. Edit the script according to below image.






```
Unity スクリプト|0 個の参照
public class CameraController : MonoBehaviour
{
    public Cinemachine.CinemachineVirtualCamera cam;

    Unity メッセージ|0 個の参照
    void Start()
    {
        cam.Priority = -50;
        /*
         * You can access with GetComponent like below but it's not smart.
         * this.GetComponent<Cinemachine.CinemachineVirtualCamera>().Priority = -50;
         * Attaching GameObject or Component with public statement: serialized field
         * make your code simple and fast.
         */
    }

    Unity メッセージ|0 個の参照
    void Update()
    {
        // Abbreviated style: Only handle 1 line.
        if (Input.GetKeyDown(KeyCode.Return)) cam.Priority *= -1;
    }
}
```

Ternary operator: Abbreviated style for if & else

Syntax: {Condition} ? {When true} : {When false}

```
if (is_true)
    message = "true";
else
    message = "false";

Debug.Log(message);
```

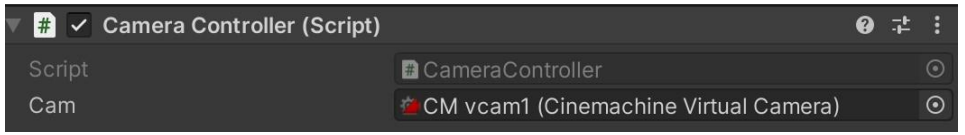
→

```
message = is_true ? "true" : "false";
Debug.Log(message);
```

Adjustment Camera

With Cinemachine

5. Attach the script to “CM vcam1”. →
6. Play the game and press “Enter” / “Return” key.



Chinemachine

- Official camera asset by Unity.
- You can easily set camera parameters as FPS, TPS, free camera and more.
- You need to import from asset store when you create new project. It’s free.

Understand basic of Class

Class as object

- Class is blueprint of scripts.
  -> Declare and implement the factors.
  If declare only, use abstract syntax.
- Class can inherit variables and functions from the parent.
  -> When inherited factors are implemented on the parent class, it can be used without implement in child class. e.g. Move()

Class Variables / Member variables / Fields:

- string name;
- int HP, ATK, SPD;

Class Functions / Methods:

- void Attack()
- void Die()
- void Move() { … // Instructions … }

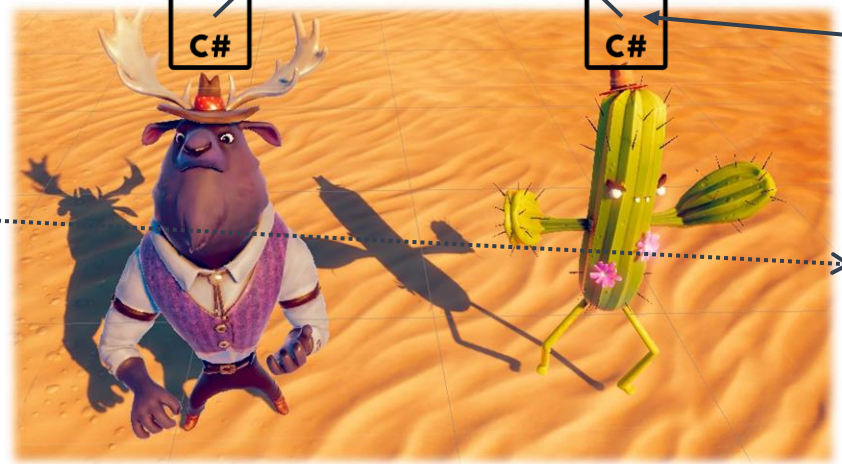
Character class

Constructor

- Method with same name as the class name.
- It's used to set default parameters.

name = MikeMoose;
HP = 20;
ATK = 5;
SPD = 5;

Player(){}
void Attack() {}
void Die() {}
~~void Move(){}~~
void GetItem(){}



name = CactusNails;
HP = 11;
ATK = 2;
SPD = 3;
EXP = 2;

EnemyA(){}
void Attack() {}
void Die() {}
~~void Move(){}~~
void DropItem(){}

….
HP = 50;
ATK = 8;
SPD = 1;
EXP = 25;

…

Player class EnemyA class EnemyABoss class

Try to customize Class

Architecture of script.

1. Please translate chapter: Customize the health potions

   1. Customize the player experience

   to

   9. Tutorial summary

2. Congratulations! Tutorial is finished.

   See last chapter if you want customize it more.

Abstruct class / Abstruct method:

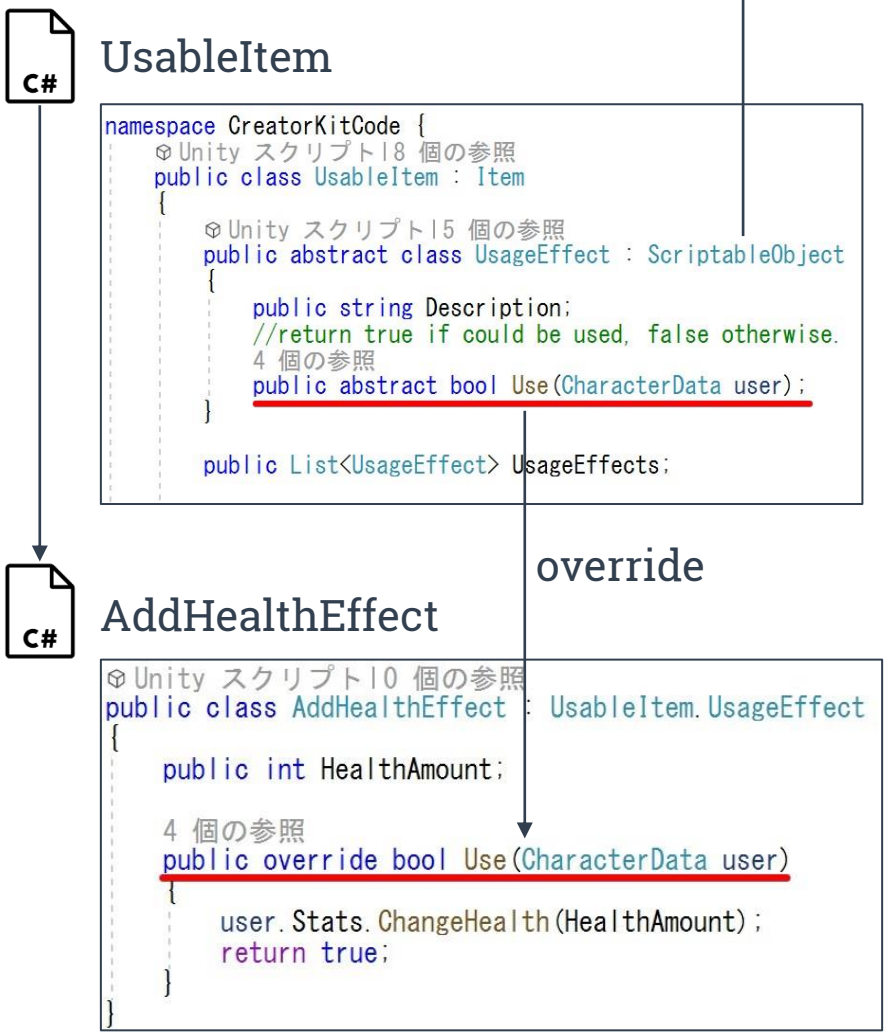
- Abstruct class has one or more abstruct methods.
- Abstruct class must be inherited by other class.
- Abstruct methods must be implemented in the child class.

Override / Overload:

- Override means rewriting the parent's method in child class. The arguments must be same.
- Overload means redeclaring with different arguments againt to the same name function.
  e.g. public bool Use(string target_name)

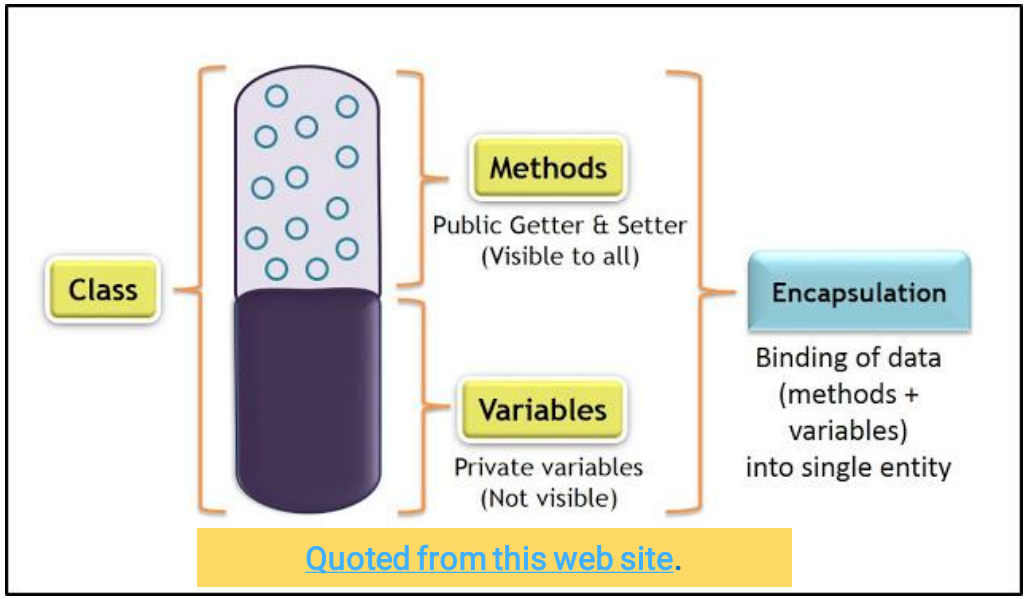
ScriptableObject

- Base class provided by Unity to share same parameters within many same-type objects.
- It's often used to manage status like atk and def by the enemy type.



Encapsulation

Architecture of script.




```
Unity スクリプト | 0 個の参照
public class AddHealthEffect : UsableItem.UsageEffect
{
    public int HealthAmount;

    4 個の参照
    public override bool Use(CharacterData user)
    {
        user.Stats.ChangeHealth(HealthAmount);
        return true;
    }
}
```

```
If (CurrentHealth < 0)
{
    CurrentHealth = 0;
}
```

```
public void ChangeHealth(int amount)
{
    CurrentHealth =
        Mathf.Clamp(CurrentHealth + amount, 0, stats.health);
}
```

HP Damage!

Encapsulation:

Can access from itself or inherited class but it may be not cool.

- Distinguish method and variables in class with public / protected / private modifier .
- Generally, class variables will be set private except for serialized field to prevent bugs from changing the value directly.
- public methods to set / get the private variables are called as “Getter / Setter”. e.g. GetHP(), SetHP(int num) etc.
- Methods only used in the public methods should be set private except for utility. e.g. CalculateDamege(atk) etc.
- Appropriate encapsulation improves the maintenance performance and readability.

More realistic Class

Class as object

Variables :

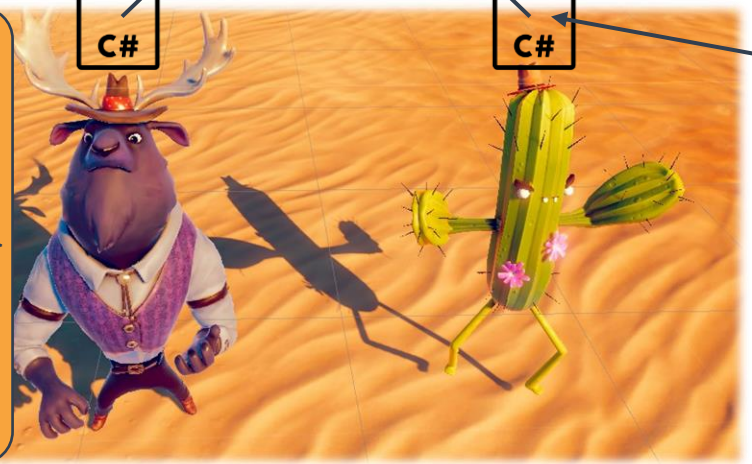
- protected string name;
- protected int HP, ATK, SPD;

Methods:

- Getter & Setter (SetName(), GetName etc...)
- public abstract void Attack()
- public abstract void Die()
- public void Move() { MoveAction() }
- private void MoveAction()

abstract Character class

```
name = MikeMoose;
HP = 20;
ATK = 5;
SPD = 5;

// Constructor
Player(string name, int HP, int ATK, int SPD){}

public override void Attack() {}
public override void Die() {}
~~public void Move(){}~~
public void GetItem(){}
```



Player class

EnemyA class

EnemyABoss class

Interface

- Multiple inheritance is prohibited on abstract class. If it's required, interface modifier should be used.
- public interface ICharacterA.
  public interface ICharacterB.
  ↓
  public class SampleClass : IcharacterA, ICharacterB { }
- Abstruct and Interface will makes your software design good (descried later).

Example of Setter / Getter (1/2)

Base class (Character class)

abstract Character class

```
Unity スクリプト | 1 個の参照
public abstract class Character : MonoBehaviour
{
    // Warning happns but it can be ignored.
    // If you want to avoid it, add "new" before string
    protected string name;

    // Full type of getter/setter will be used on name and HP.
    protected int HP;

    // Abbreviated style of getter/setter will be used on ATK and SPD.
    2 個の参照
    protected int ATK { get; set; }
    2 個の参照
    protected int SPD { get; set; }

    // Declare constants
    private const int MAX_NAME_LENGTH = 10;
    private const int MIN_NAME_LENGTH = 2;
    private const int MAX_HP = 20;
    private const int MIN_HP = 0;
    private string ERR_MSG_NAME =
        "Name length must be between " + MIN_NAME_LENGTH + " and " + MAX_NAME_LENGTH;

    // Getter for name
    1 個の参照
    public string GetName()
    {
        return this.name;
    }

    // Setter for name
    1 個の参照
    public void SetName(string name)
    {
        if (name.Length > MAX_NAME_LENGTH || name.Length < MIN_NAME_LENGTH)
            throw new System.Exception(ERR_MSG_NAME);
        else
            this.name = name;
    }
```

Same as:

```
private string _ATK;
public string ATK
{
  get { return this._ATK;}
  set { this._ATK = value;}
}
```

From outside of the class, it behaves like variable.
From inside of the class, it behaves like method.

Abbriviated style of this line:

$"Name length must be between {MIN_NAME_LENGTH} and {MAX_NAME_LENGTH}";

Output the error message on console.

```
    // Getter for HP
    1 個の参照
    public int GetHP()
    {
        return this.HP;
    }

    // Setter for HP
    1 個の参照
    public void SetHP(int HP)
    {
        if (HP > MAX_HP)
            this.HP = MAX_HP;
        else if (HP < MIN_HP)
            this.HP = MIN_HP;
        else
            this.HP = HP;
    }

    // Declare abstract methods
    1 個の参照
    public abstract void Attack();
    1 個の参照
    public abstract void Die();

    // Implement normal public method
    1 個の参照
    public void Move()
    {
        MoveAction();
    }

    // Implement private method for Move()
    1 個の参照
    private void MoveAction()
    {
        this.GetComponent<Rigidbody>().velocity = new Vector3(1, 0, 0);
    }
}
```

Use const variables to avoid hard-coded numbers.

Just declare only in this abstract class.

Encapsulate MoveAction() as private because it's only used in this class.

Example of Setter / Getter (2/2)

Inherited class (Player class)

Player class

```
Unity スクリプト | 3 個の参照
public class Player : Character
{
    // Constructor
    1 個の参照
    Player(string name, int HP, int ATK, int SPD)
    {
        // Using normal setter
        SetName(name);
        SetHP(HP);

        // Using abbreviated style of setter
        this.ATK = ATK;
        this.SPD = SPD;
    }

    // Abstract methods will be wrote automatically by Visual Studio.
    1 個の参照
    public override void Attack()
    {
        throw new System.NotImplementedException();
    }

    0 個の参照
    public void Attack(int ATK)
    {
        throw new System.NotImplementedException();
    }

    // Override Die method.
    1 個の参照
    public override void Die()
    {
        // Delete this object.
        Destroy(this.gameObject);
    }
```

Delete the GameObject from the Scene

```
    Unity メッセージ | 0 個の参照
    void Start()
    {
        // Warning happns when use this constractor but it can be ignored.
        // MonoBehaviour inherited class should not use "new" but this is sample.
        Player p = new Player("MikeMoose", 999, 5, 5);

        // Use getters
        Debug.Log(p.GetName());
        Debug.Log(p.GetHP());
        Debug.Log(p.ATK);
        Debug.Log(p.SPD);
    }

    Unity メッセージ | 0 個の参照
    void Update()
    {
        // Use method of parent calss.
        this.Move();
    }
}
```

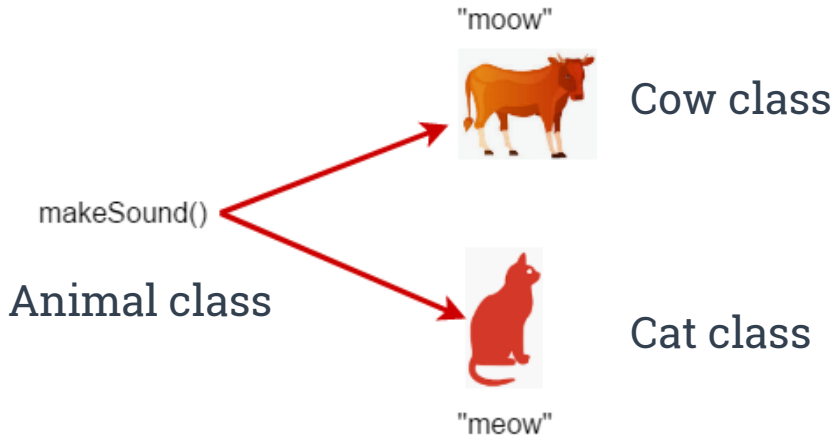
Set 999 with SetHP()

The output shows “20”

Create Sphere and attache the Player script to it.
Play the game and check console.

Polymorphism

Implemented same-name methods provide different effects.



Quoted from this web site.

```
Player.Move()
{
  // Mouse based move
}
```

Player class

EnemyA class

```
EnemyA.Move()
{
  // Keyboard based move
}
```

Polymorphism

Implemented same-name methods provide different effects.

Case1: Simple inherited classes

1

```
public class ShapeNonP : MonoBehaviour
{
    void Start()
    {
        TriangleNonP tri = new TriangleNonP { Base = 5, Height = 4 };
        RectangleNonP rec = new RectangleNonP { Width = 4, Height = 4 };
        CircleNonP cir = new CircleNonP { Radius = 3 };

        Debug.Log($"Area of {tri.Name()}: {tri.Area()}");
        Debug.Log($"Area of {rec.Name()}: {rec.Area()}");
        Debug.Log($"Area of {cir.Name()}: {cir.Area()}");
    }
}

public class TriangleNonP : ShapeNonP
{
    public double Base { get; set; }
    public double Height { get; set; }
    public string Name() { return "Triangle"; }
    public double Area() { return Base * Height / 2; }
}

public class RectangleNonP : ShapeNonP
{
    public double Width { get; set; }
    public double Height { get; set; }
    public string Name() { return "Rectangle"; }
    public double Area() { return Width * Height; }
}

public class CircleNonP : ShapeNonP
{
    public double Radius { get; set; }
    public string Name() { return "Circle"; }
    public double Area() { return Radius * Radius + System.Math.PI; }
}
```

Use with { } like Constractor.

double: More precice type than float.

Same as:

```
private double _Width;
public double Width
{
  get { return this._Width;}
  set { this._Width = value;}
}
```

Case2: Abstract inherited classes

2

```
public abstract class ShapeP2
{
    public abstract string Name();
    public abstract double Area();

    void Start() // This method is not work in Unity without MonoBehaviour
    {
        ShapeP2[] shapeArray =
        {
            new TriangleP2 { Base = 5, Height = 4 },
            new RectangleP2 { Width = 4, Height = 4 },
            new CircleP2 { Radius = 3 }
        };

        foreach (ShapeP2 shape in shapeArray)
        {
            Debug.Log($"Area of {shape.Name()}: {shape.Area()}");
        }
    }
}

public class TriangleP2 : ShapeP2
{
    public double Base { get; set; }
    public double Height { get; set; }
    public override string Name() { return "Triangle"; }
    public override double Area() { return Base * Height / 2; }
}

public class RectangleP2 : ShapeP2
{
    public double Width { get; set; }
    public double Height { get; set; }
    public override string Name() { return "Rectangle"; }
    public override double Area() { return Width * Height; }
}

public class CircleP2 : ShapeP2
{
    public double Radius { get; set; }
    public override string Name() { return "Circle"; }
    public override double Area() { return Radius * Radius + System.Math.PI; }
}
```

Parent class can handle inherited class.

Polymorphism

Implemented same-name methods provide different effects.

Case2: Abstract inherited classes

2

```
public abstract class ShapeP2
{
    public abstract string Name();
    public abstract double Area();

    void Start() // This method is not work in Unity without MonoBehaviour
    {
        ShapeP2[] shapeArray =
        {
            new TriangleP2 { Base = 5, Height = 4 },
            new RectangleP2 { Width = 4, Height = 4 },
            new CircleP2 { Radius = 3 }
        };

        foreach (ShapeP2 shape in shapeArray)
        {
            Debug.Log($"Area of {shape.Name()}: {shape.Area()}");
        }
    }
}

public class TriangleP2 : ShapeP2
{
    public double Base { get; set; }
    public double Height { get; set; }
    public override string Name() { return "Triangle"; }
    public override double Area() { return Base * Height / 2; }
}

public class RectangleP2 : ShapeP2
{
    public double Width { get; set; }
    public double Height { get; set; }
    public override string Name() { return "Rectangle"; }
    public override double Area() { return Width * Height; }
}

public class CircleP2 : ShapeP2
{
    public double Radius { get; set; }
    public override string Name() { return "Circle"; }
    public override double Area() { return Radius * Radius + System.Math.PI; }
}
```

=

```
public interface IShape
{
    // Just declare on interface.
    public string Name();
    public double Area();
}
```

Case3: Interface + classes

3

```
public class ShapeP : MonoBehaviour
{
    void Start()
    {
        IShape[] shapeArray =
        {
            new Triangle { Base = 5, Height = 4 },
            new Rectangle { Width = 4, Height = 4 },
            new Circle { Radius = 3 }
        };

        foreach (IShape shape in shapeArray)
        {
            Debug.Log($"Area of {shape.Name()}: {shape.Area()}");
        }
    }
}

public class Triangle : IShape
{
    public double Base { get; set; }
    public double Height { get; set; }
    public string Name() { return "Triangle"; }
    public double Area() { return Base * Height / 2; }
}

public class Rectangle : IShape
{
    public double Width { get; set; }
    public double Height { get; set; }
    public string Name() { return "Rectangle"; }
    public double Area() { return Width * Height; }
}

public class Circle : IShape
{
    // You can declare Name as variable with initialize like below.
    // public string Name { get; set; } = "Circle";
    public double Radius { get; set; }
    public string Name() { return "Circle"; }
    public double Area() { return Radius * Radius + System.Math.PI; }
}
```

Interface can handle implemented class.

Try to write 1 ~ 3 case for your practice.

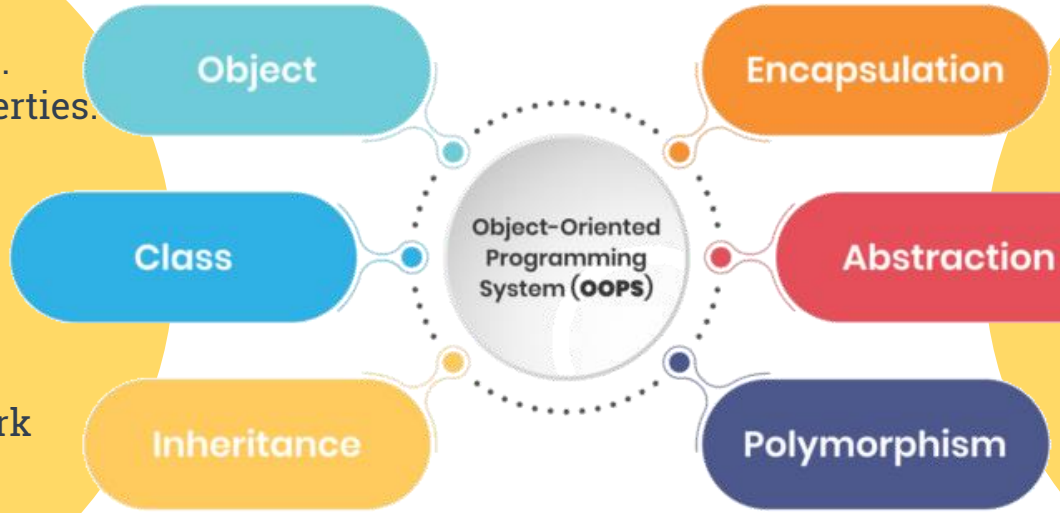
Object Oriented Programming

Already you know it!

Object is instantiated class.
Every object has own properties.

Class is designed model of real stuff. It has variables and methods.

Inheritance is effective work to avoid repeated code.



Encapsulation prevent unexpected data update.

Abstraction / Interface makes your code clean and easy-updatable.

Polymorphism can handle various classes by same way. Your code will be simple.

Quoted from this web site.

Set up for Class diagram

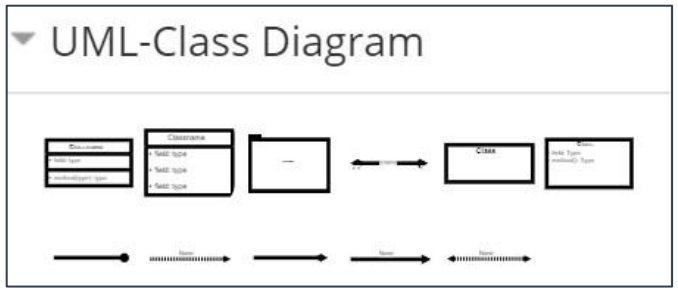
The first step of design: UML

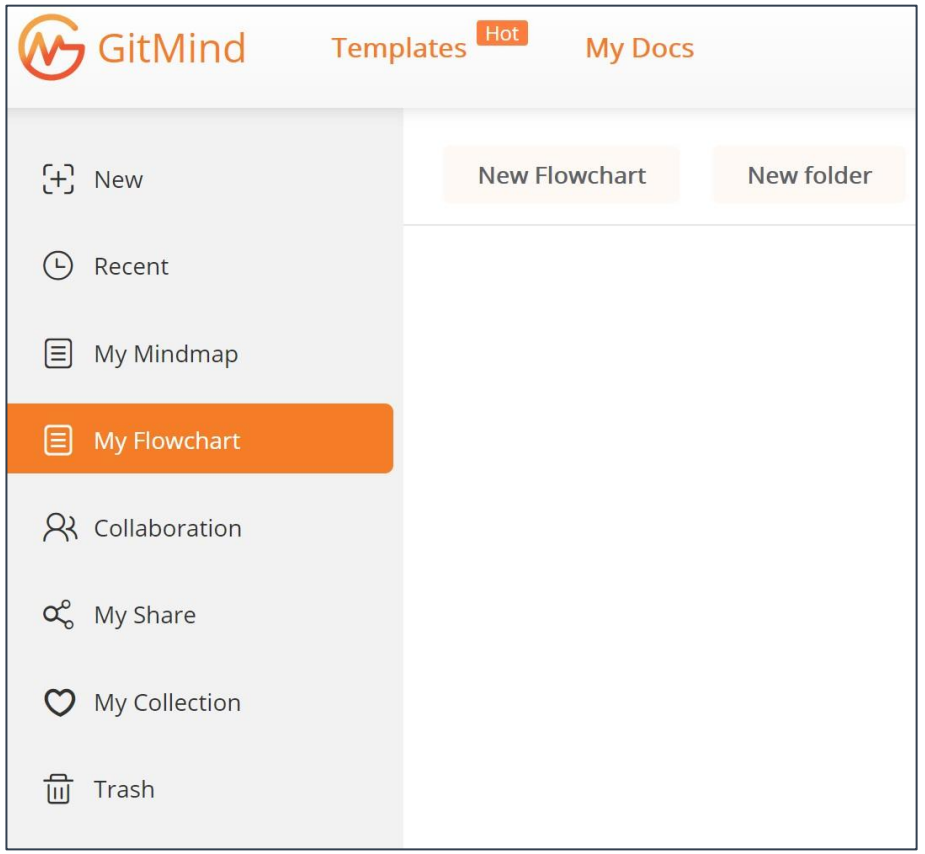
1. Access GitMind.
2. Login or register your account.
3. Access My Flowchart.

   (Mindmap is also useful to draw your ideas!)
4. Select New Flowchart.
5. You can draw diagrams!

   In design class, we mainly use UML-Class Diagram.

   (UML: Unified Modeling Language)

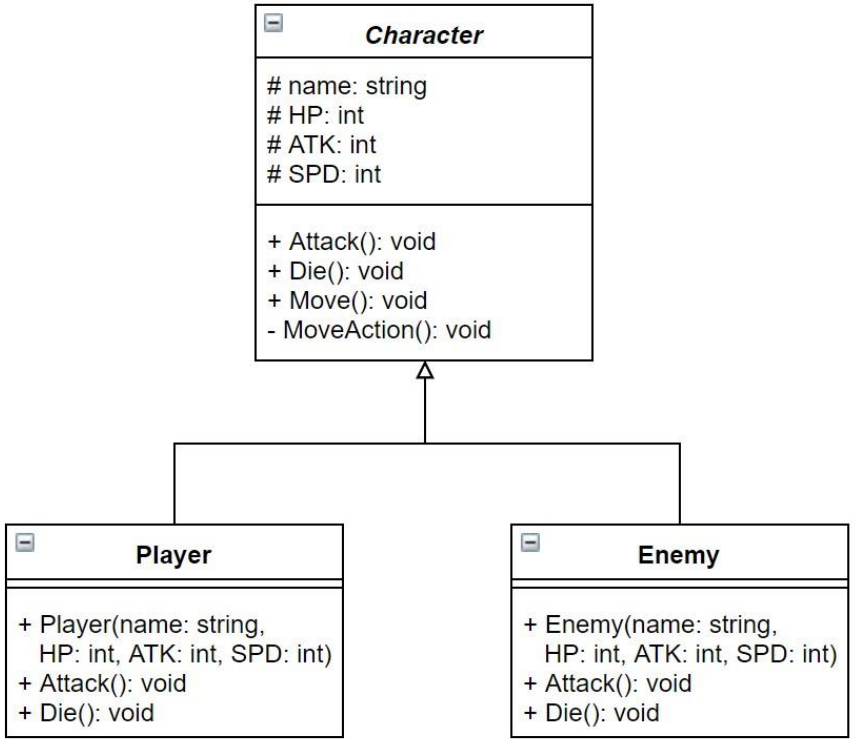




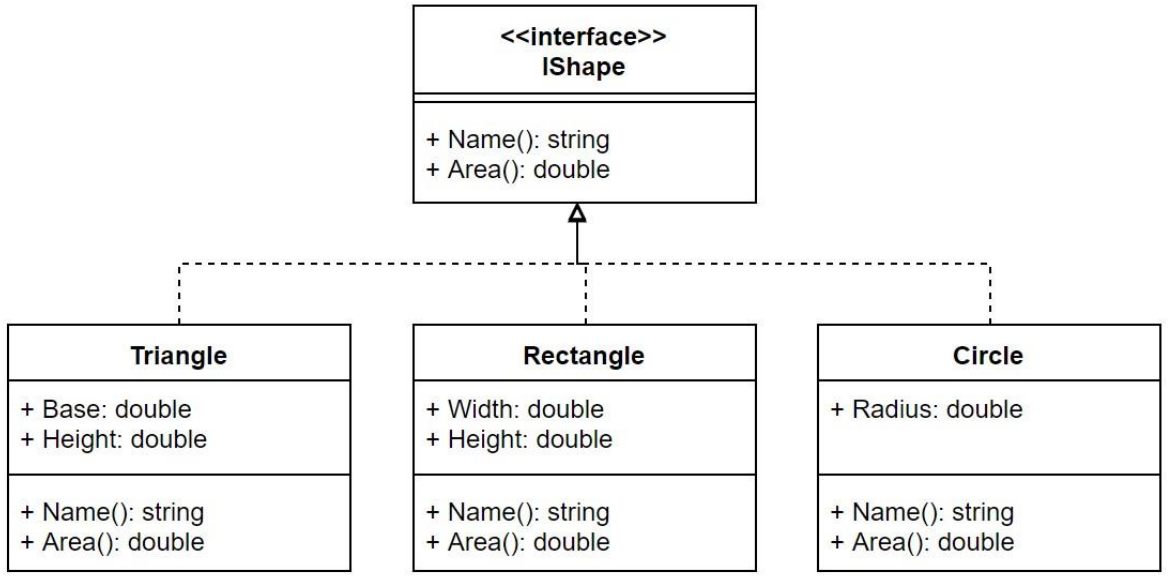
Set up for Class diagram

The first step of design: UML

Abstract class + Inherited classes

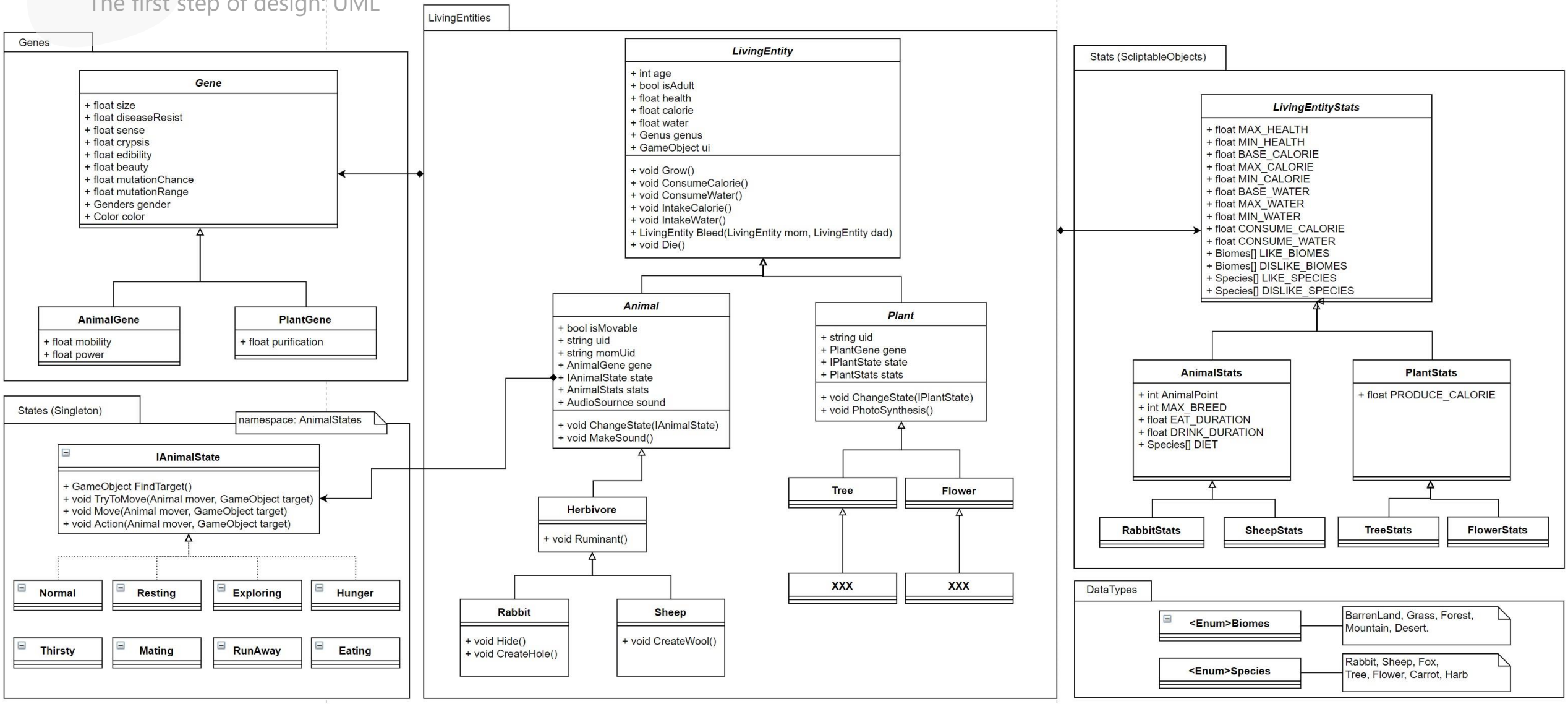


<<Interface>> + Implemented classes



Set up for Class diagram

The first step of design: UML



2.Basic programming with Unity

Class Plan

1. Introduction
2. Hello world in Unity
3. Basic programming
4. Object oriented programming
5. **Interface programming**
6. Programming design 1
7. Programming design 2
8. Improvement and testing 1
9. Improvement and testing 2
10. Summary

5
Interface programming

5. Interface programming

Class summary

1. Explain Interface.
2. Explain Unity GUI.
3. Try to create UI.
4. Try to change material / shader.
5. Explain Assets.
6. Explain Terrain.

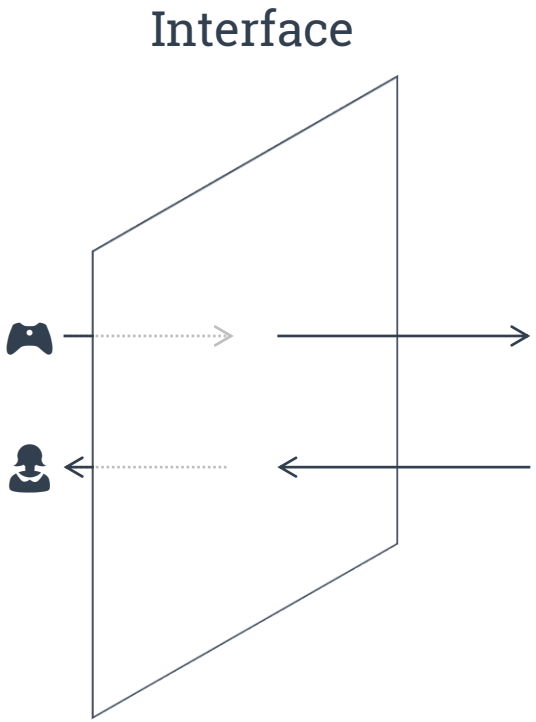
Coding Rules: Examples

Target	Major Rules	Examples
Class name	a. UpperCamel	a. SampleClass
Class variables	a. LowerCamel b. Snake_case c. Underscore	a. sampleVariable b. sample_variable c. _sampleVariable (Only for private variables)
Bool variables	a. Is_bool	a. is_run
Methods	a. UpperCamel	a. SampleMethod()
Constants	a. All-Caps b. UpperCamel	a. MAX_NUMBER b. MaxNum
Scene name	a. UpperCamel	a. TitleScene
Object name	a. UpperCamel	a. CactusNails
If Statement	a. if (target < condition)	a. if (num < 10)
Comments	You don't' need to write all of contents. Good code & Good name explains the role or function itself.	
Scope	Try to minimize the range.	
Hardcoding	Don't use it. You should use constants.	
Core mind	Simple and Dry. Functions should be under 30〜50 lines.	

What is UI?

"Interface" again.

© Unity Technologies Japan/UCL



© Unity Technologies

"In general, an interface is a device or a system that unrelated entities use to interact. According to this definition, a remote control is an interface between you and a television set, the English language is an interface between two people."

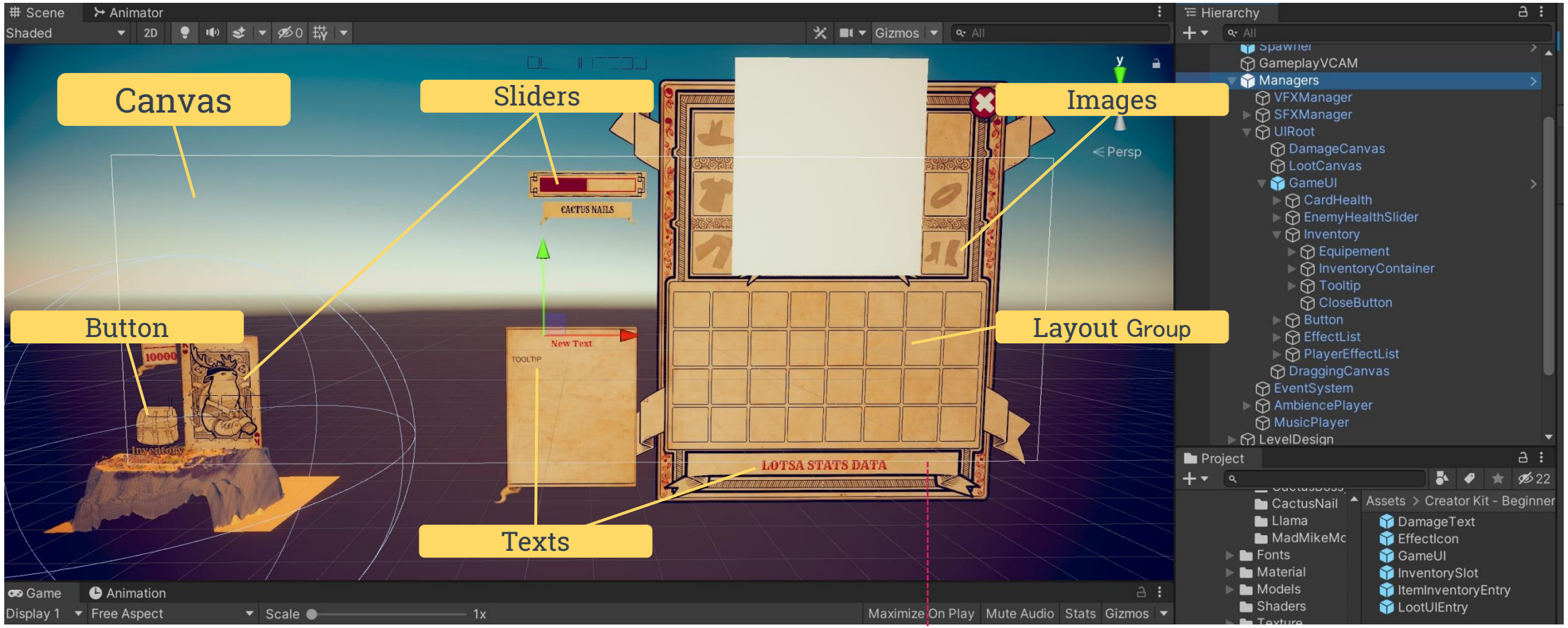
https://www.iitk.ac.in/esc101/05Aug/tutorial/java/concepts/interface.html

What is Unity GUI?

"Interface" of overlay information.

What is Unity GUI?

"Interface" composed by UI object on the Canvas.



Canvas size is depending on screen resolution.

Today's Goal

"Interface" composed by UI object on the Canvas.

Video

Flow of the design

1. Determine screen resolution.
    - Full HD (1920*1080)
2. Create TitleScene.
    - Change project settings
    - Set AudioSource
3. Create UI on TitleScene.
    - Canvas
    - Image-bg
    - Text_title
    - Slider_progress
    - Button_start
4. Create scripts.
    - Load IntroductionScene
5. Build and Test.

Create Interface

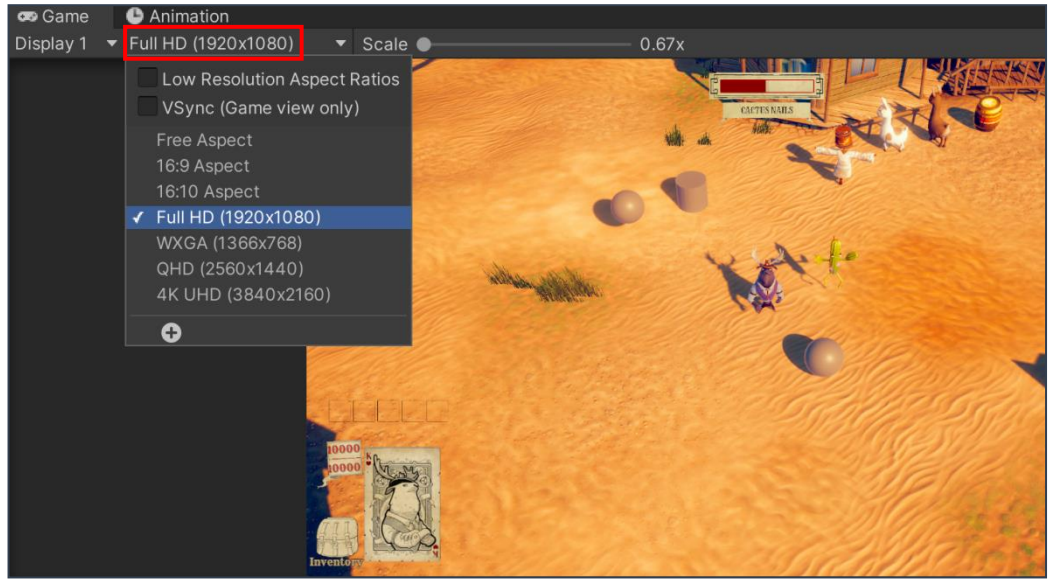
1. Determine screen resolution.

1. Select Full HD option on Game view. →

   (It just emulates screen by the resolution.)

2. Set Full HD on script.
   a. Create “StartButton” script on Scripts/Tutorial
   (It will be attached to Button_start object later)
   b. Edit the script according to below image.

```
public class StartButton : MonoBehaviour
{
    public void Start()
    {
        Screen.SetResolution(1920, 1080, false);
    }

    public void Update()
    {

    }
}
```



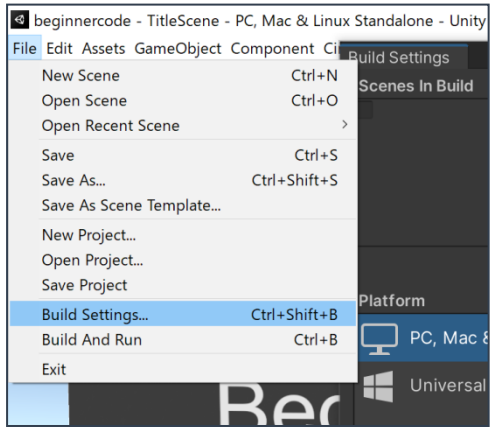
Screen.SetResolution(int width, int height, bool mode)

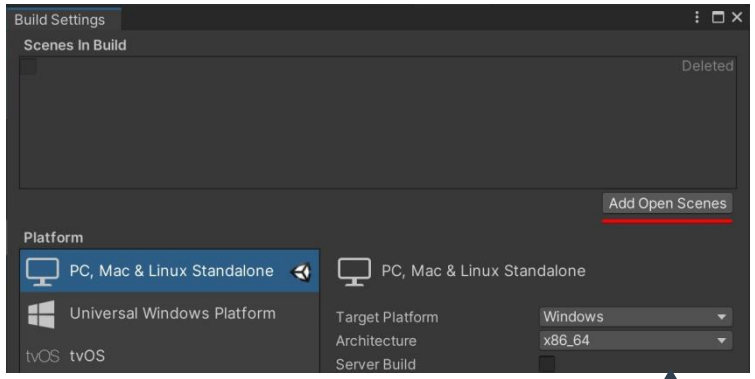
- Set the resolution with the arguments.
- If mode is true, it will become full-screen.
  (false == window mode)
- There are overloaded version to set refresh-rate.

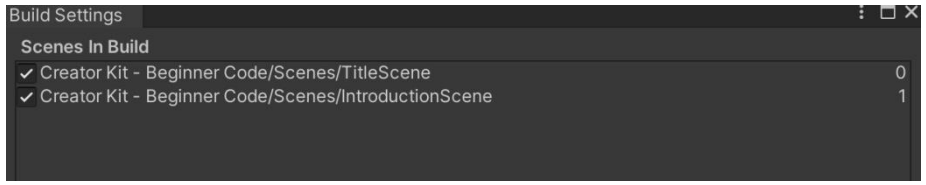
Create Interface

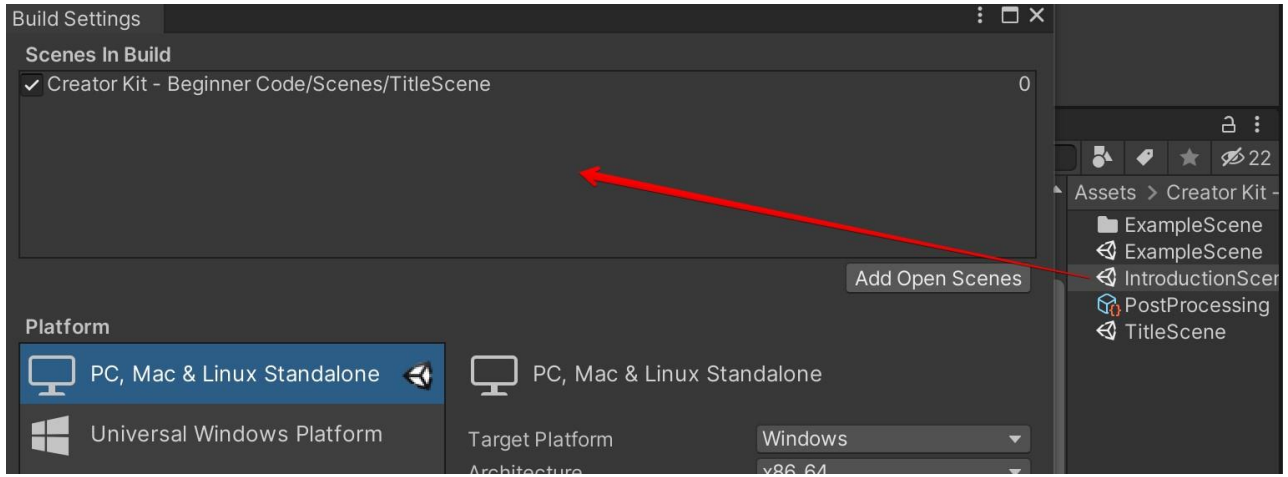
2. Create TitleScene.

1. Right click in Scenes folder.
   Select “Create” -> “Scene”.
2. Rename it to “TitleScene”.
   Double click the “TitleScene” to open it.
3. Select “File” -> “Build Settings…”.
4. Select “Add Open Scenes”.
5. Drag “IntroductionScene” and drop it.
6. Check the order according to this image.







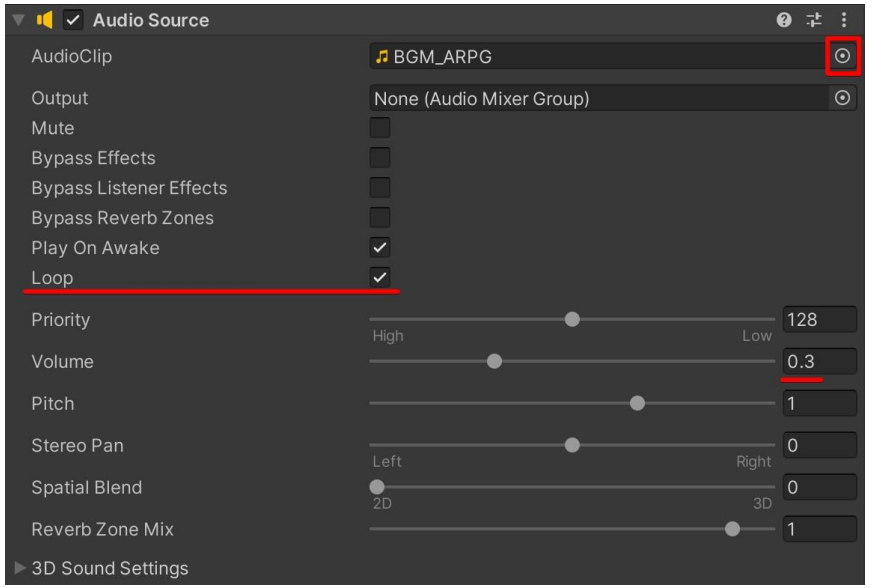


Your game starts from 0 numbered scene.

Create Interface

2. Create TitleScene.

7. Add AudioSource component to Main Camera.
8. Set the Audio Parameters.
9. Play the game and check sound.



Create Interface

3. Create UI on TitleScene.

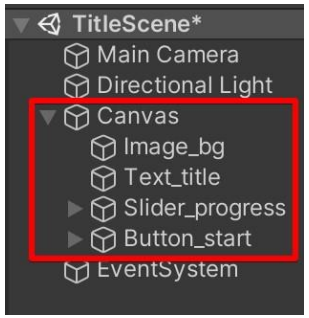
1. Create Canvas on Hierarchy.

   Right click -> UI -> Canvas.

2. Create below UI in the Canvas.

   Right click the Canvas -> UI -> :

   - Image
   - Text
   - Slider
   - Button

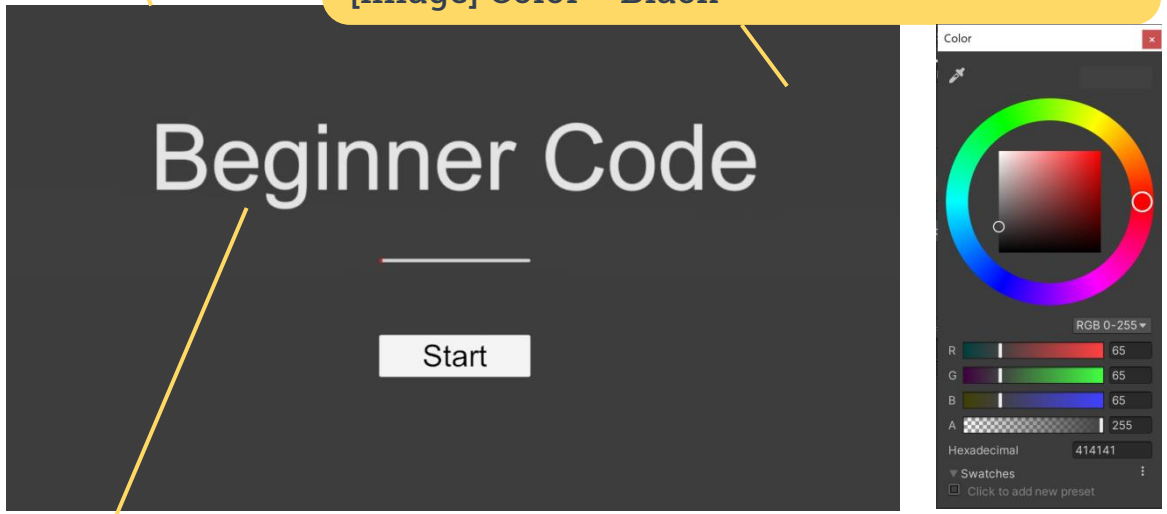
   and rename it like above image.

3. Adjust the layout according to this image.

   (Continue to next page for Slider & Button.)

Canvas:
[Rect Transform] Width = 1920, Height = 1080
[Canvas] Render Mode = Screen Space - Camera

Image_bg:
[Rect Transform] Width = 1920, Height = 1080
[Image] Color = Black

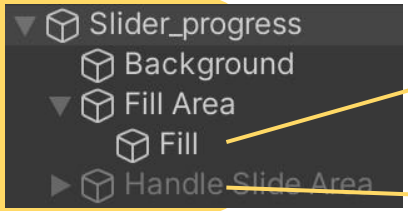


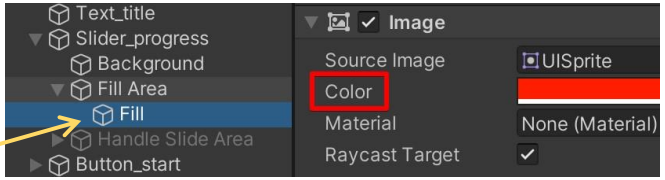
Text_title:
[Rect Transform] Pos Y = 300, Scale X = 1.5, Scale Y = 1.5
[Text] Text = Beginner Code, Font Size = 128,
Alignment = middle  ,
Horizontal Overflow = Overflow, Vertical Overflow = Overflow

Create Interface

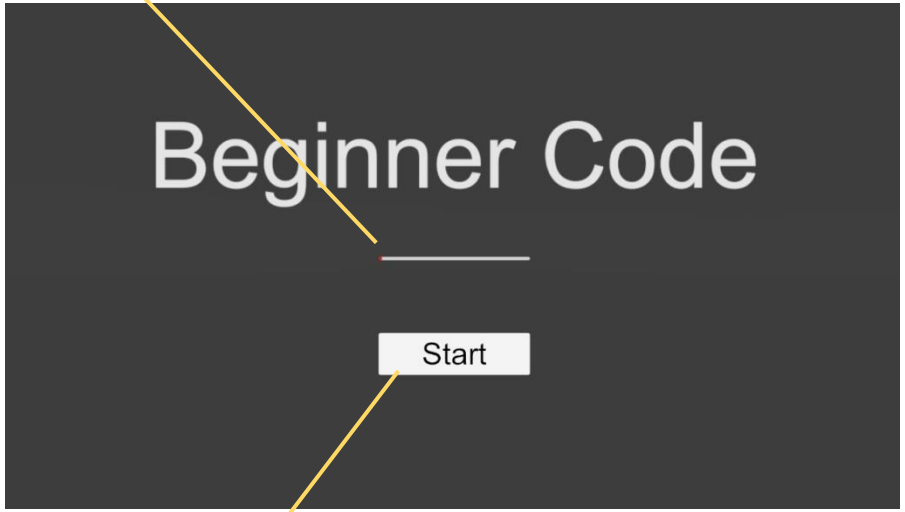
3. Create UI on TitleScene.

Slider_progress:
[Rect Transform] Width = 320
[Fill Area/Fill/Image] Color = Red
[Handle Slide Area] SetActive(false)





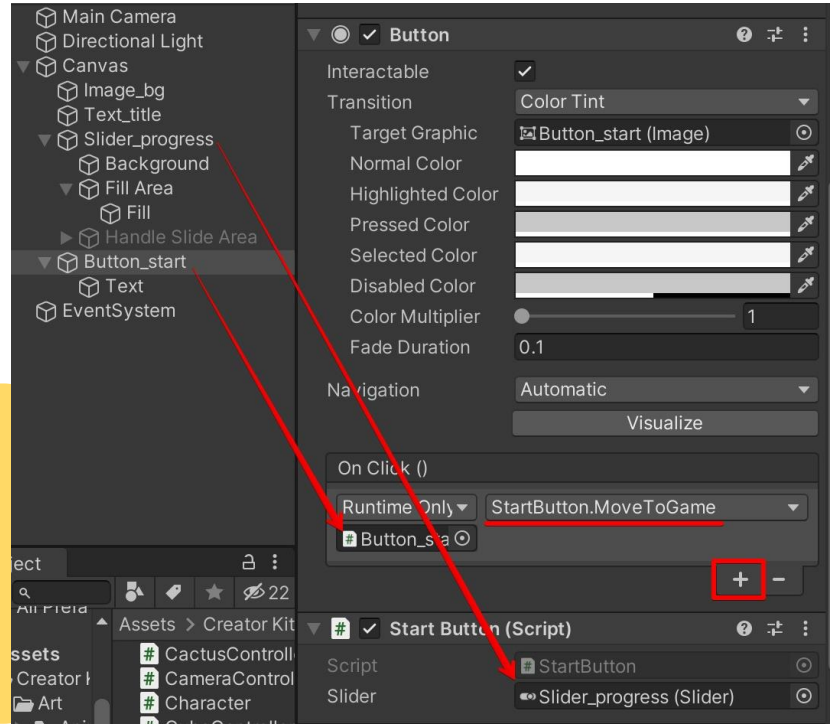

```using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class StartButton : MonoBehaviour
{
    public Slider slider;

    public void MoveToGame()
    {
        // Loading
    }

    public void Start()
    {

    }

    public void Update()
    {

    }
}
```



Button_start:
[Rect Transform] Pos Y = -200, Width = 320, Height = 90

- Add edited StartButton script as component.
  Drag Slider_progress and Drop it to serialized field.
- Press + button on On Click(). Drag Button_start and Drop it. Then, select StartButton.MoveToGame.

[Text/Text] Text = Start, Font Size = 64



Create Interface

4. Create scripts.

1. Edit the StartButton script according to this image. →
2. Play the game and check it.

   (When your device cannot load correctly,

   it may freeze… Please reboot Unity or your device.)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class StartButton : MonoBehaviour
{
    public Slider slider;
    private bool isLoading = false;
    private AsyncOperation ao;

    public void MoveToGame()
    {
        // Prevent multiple click
        if (!isLoading)
        {
            isLoading = true;
            slider.gameObject.SetActive(true);
            ao = SceneManager.LoadSceneAsync("IntroductionScene");
        }
    }

    public void Start()
    {
        Screen.SetResolution(1920, 1080, false);
        slider.gameObject.SetActive(false);
    }

    public void Update()
    {
        if (ao != null)
        {
            Debug.Log(ao.progress);
            slider.value = ao.progress;
        }
    }
}
```

gemeObject.SetActive(bool value):

- Activate or Deactivate target.
- Deactivated target become invisible.
  The attached scripts will be deativated too.

AsyncOperation:

- Class to manage async related parameters.

SceneManager.LoadSceneAsync(string sceneName) :

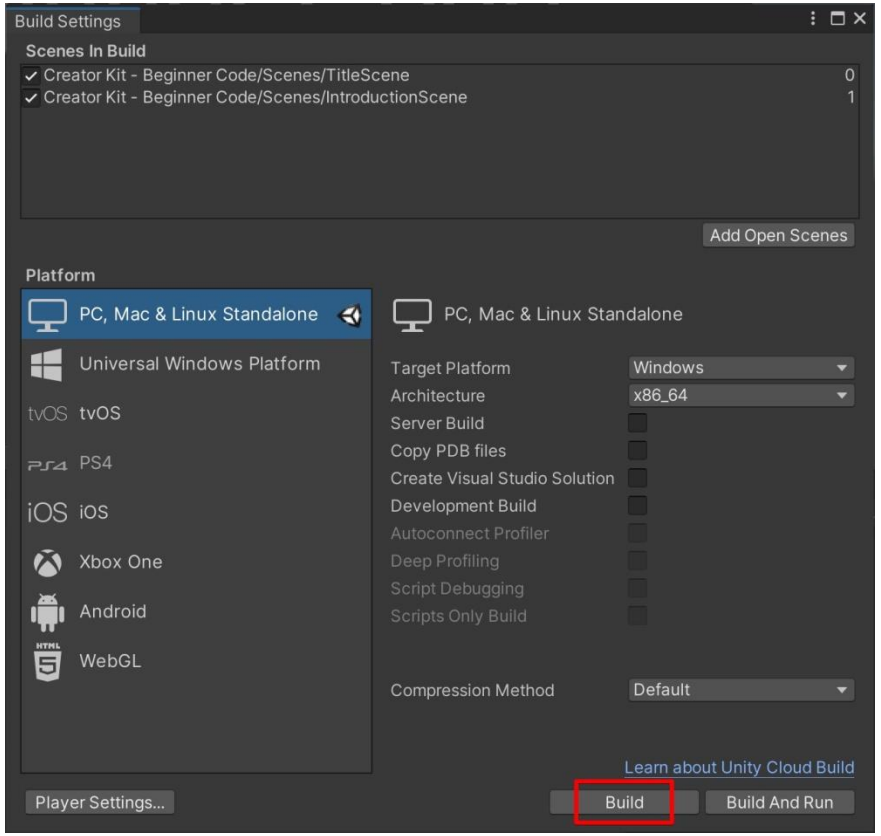
- Load target scene with async.
- Returns AsyncOperation.
- You can check the load progress with ao.progress as float [0 ~ 1.0]. In this case, use the value for update Slider_progress.

Create Interface

5. Build and Test.

1. Open build settings via File -> Build Settings…
2. Click Build.
3. Select save folder.

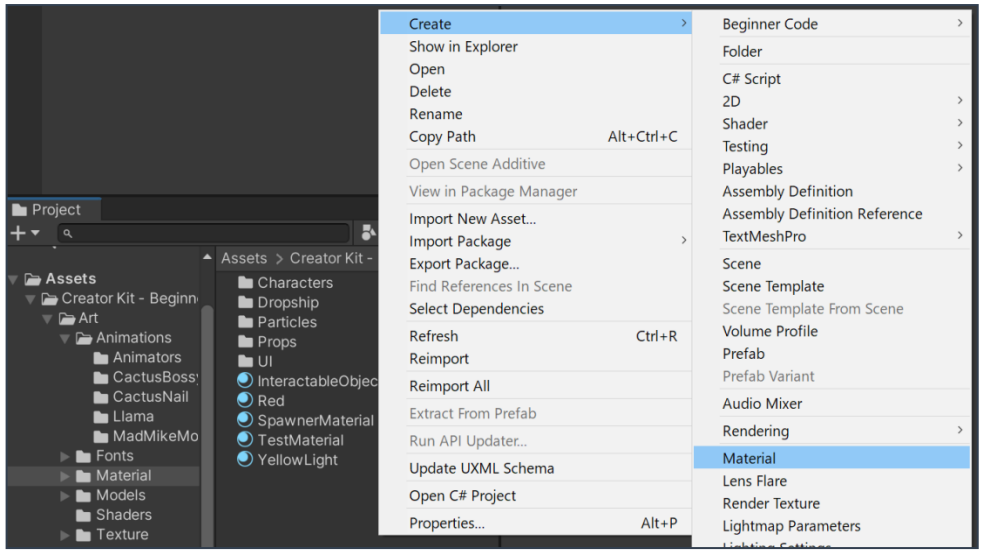
   e.g. /projects/beginnercode/builds
4. Wait few minutes.
5. Double click “New Unity Project.exe” in saved folder.
6. Play the game.

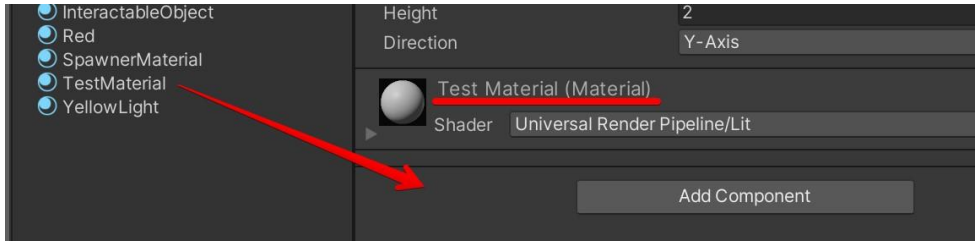


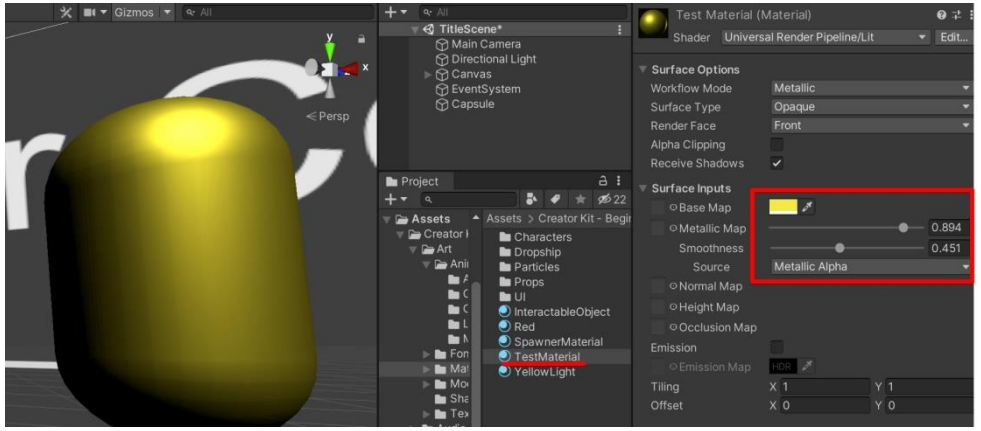
Try to change Material

Material = Shader(script) + Parameters

1. Create 3D Object -> Capsule in Hierarchy.
2. Create Material in Project (Material Folder)
   Name it as “TestMaterial”.
3. Select Capsule and Drag & Drop the “TestMaterial”
   into near of “Add Component”.
4. You can change appearance of the Capsule
   from “TestMaterial” .

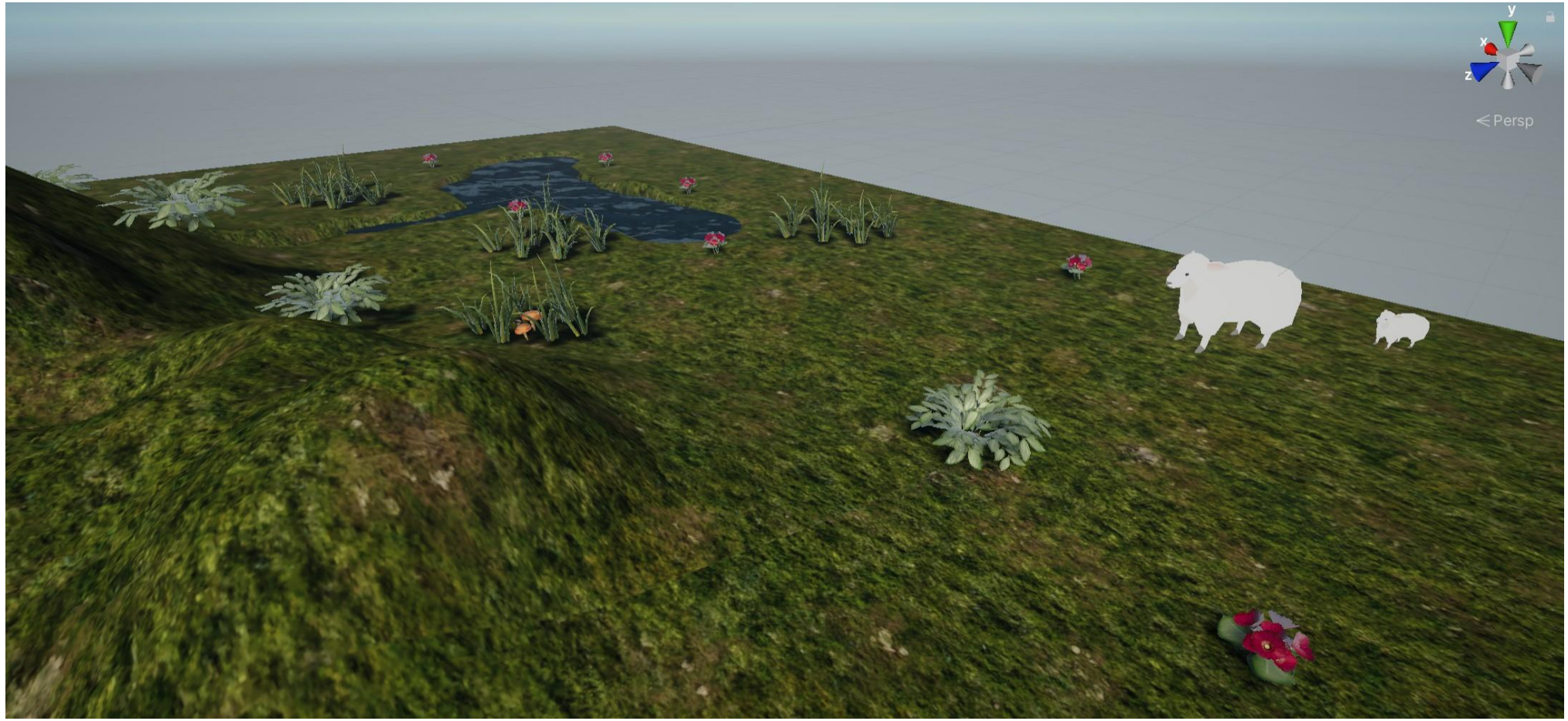




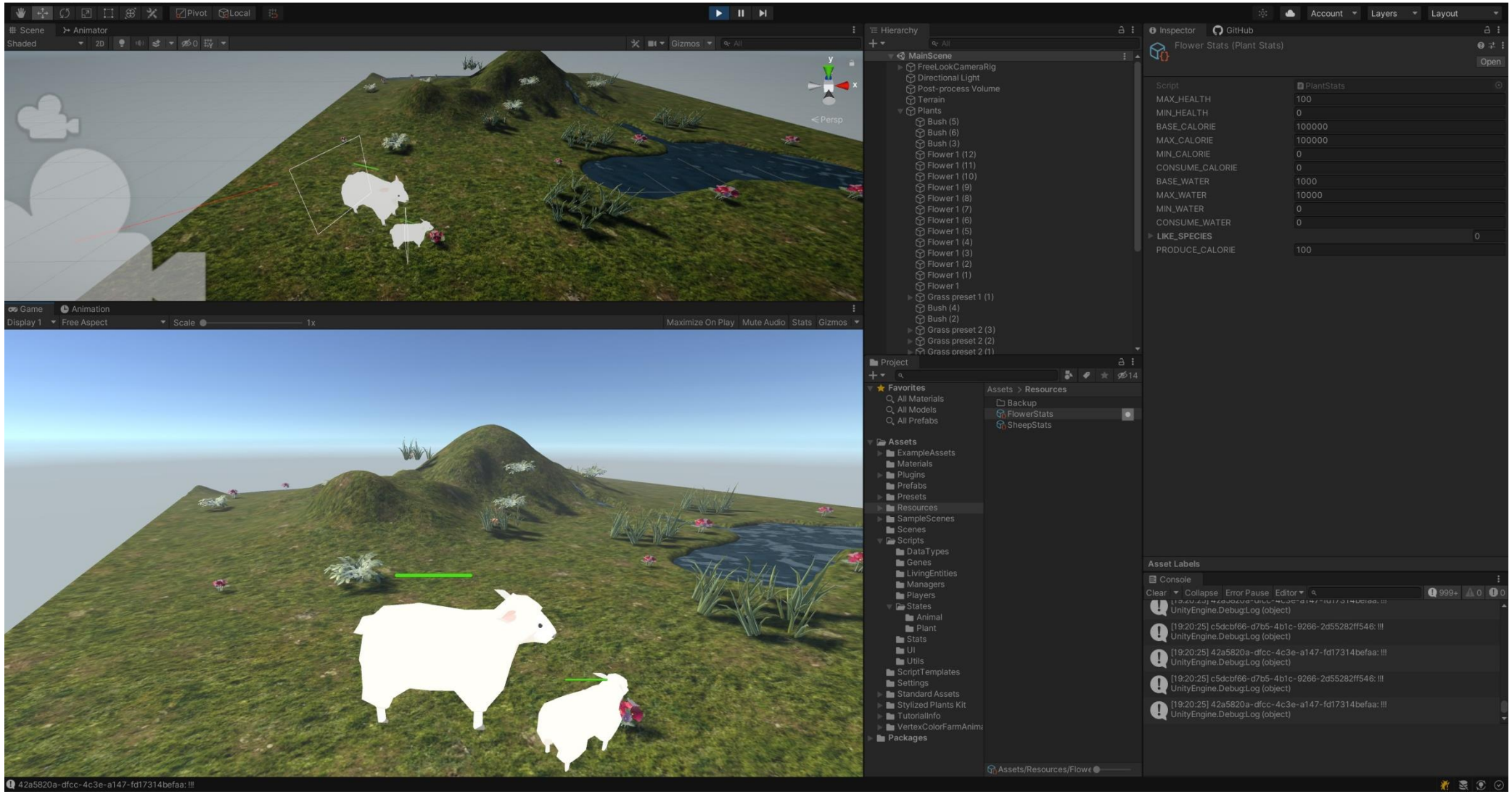


Congratulations!
Already you’ve got skills to develop your Game!

Challenge the New Game!



Challenge the New Game!

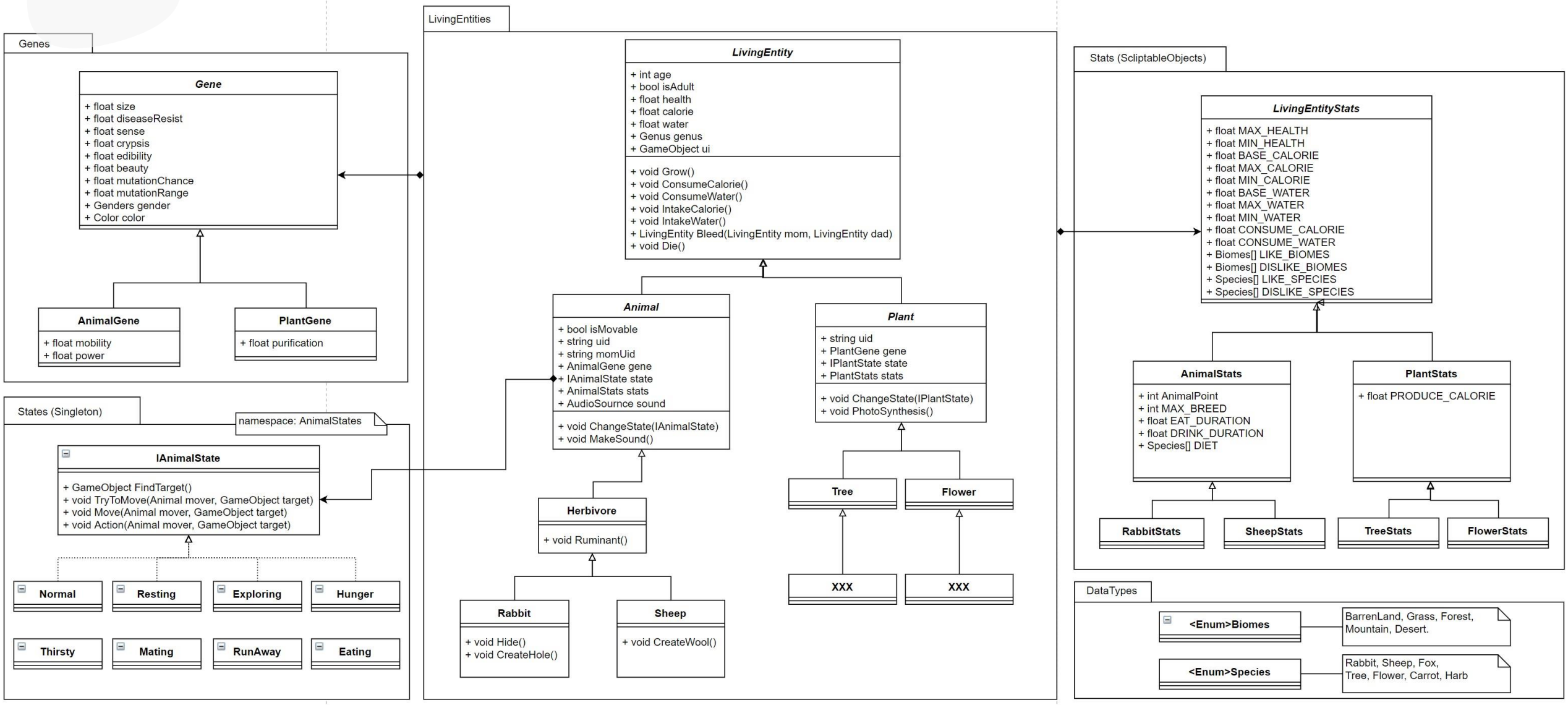


Challenge the New Game!

Create Ecosystem simulation game.

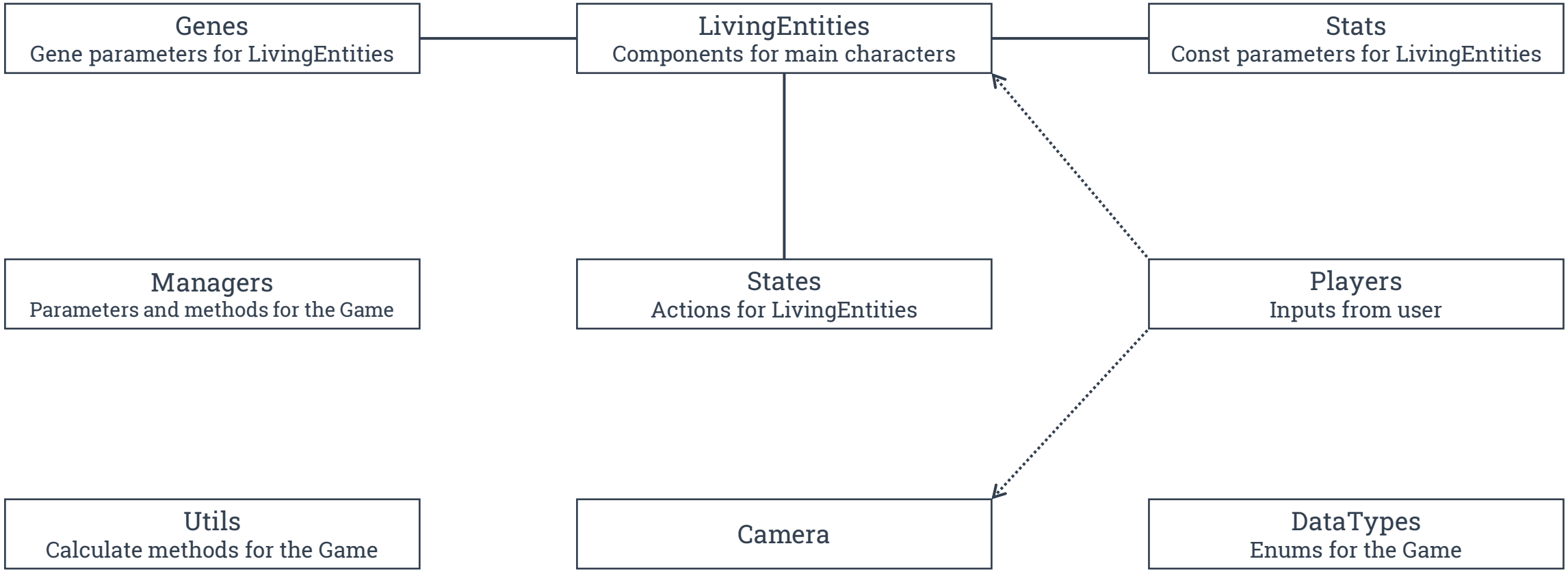
No	Type	Task	Contents
1	Plan	Game planning	Decide what type of game you want to create. Write down it 1 or 2 line phrase(s).
2	Plan	UX planning	Think about what the fun (good experience) of the game.
3	Design	Class identify	Identify what class you need.
4	Design	Method identify	Identify what method you need.
5	Design	Draw relationship	Draw the relationship on the classes.
6	Implement	Implement one by one	Implement according to your design. Check your task to clarify your progress.
7	Implement	Redesign	Redesign and redraw your design belong your implement.
8	Test	Method based test	Check each method works correctly.
9	Test	Class based test	Check each class works correctly.
10	Review	Review	Clarify what points are good or nod good.
X	Visual design	Visual design	Create or introduce terrains, prefabs, assets, sounds etc.

Challenge the New Game!



Game Design

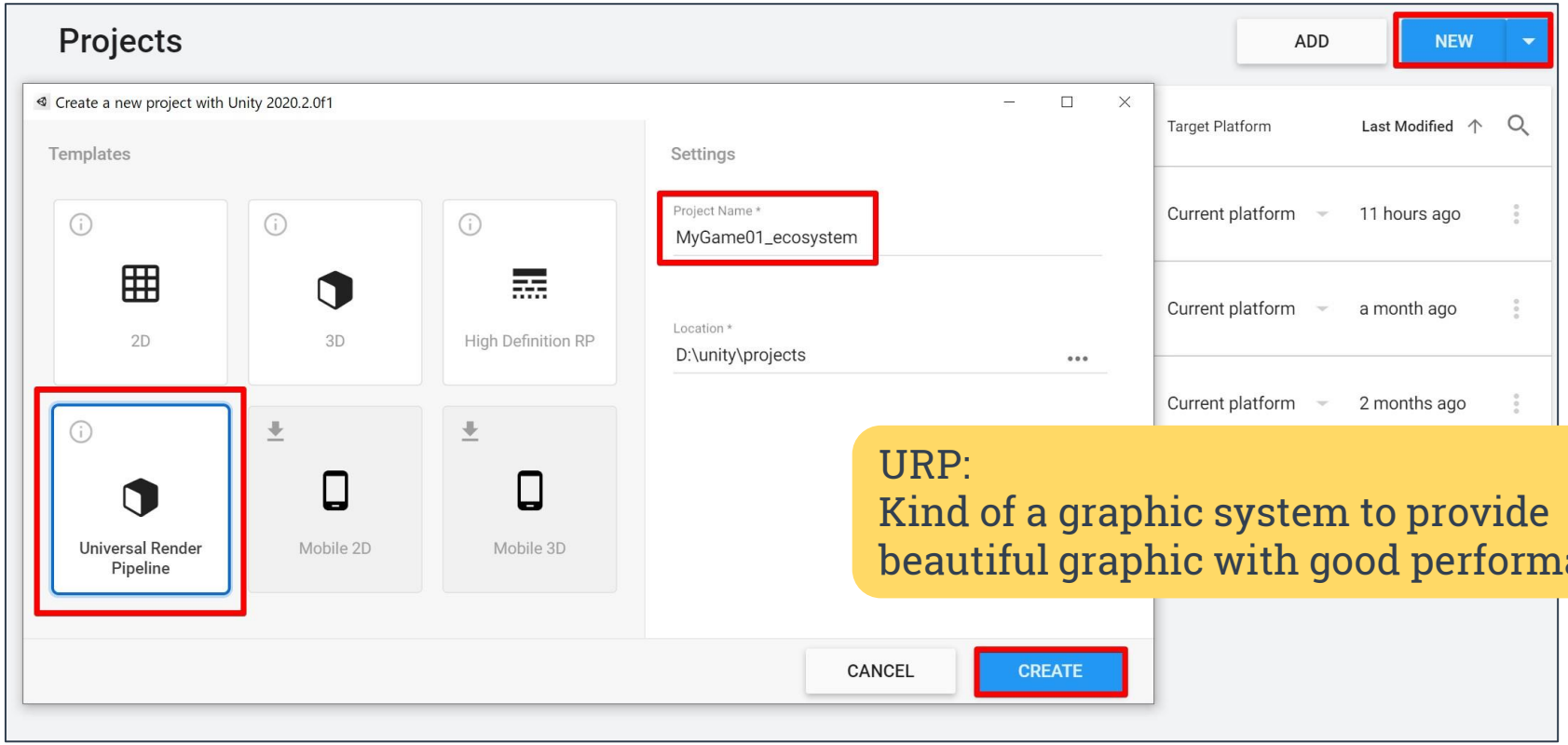
Main components of your game.



Challenge the New Game!

Create new project.

1. Quit BeginnerCode project.
2. Open Unity Hub.
3. Create new project as “MyGame01_ecosystem” as Universal Render Pipeline(URP).



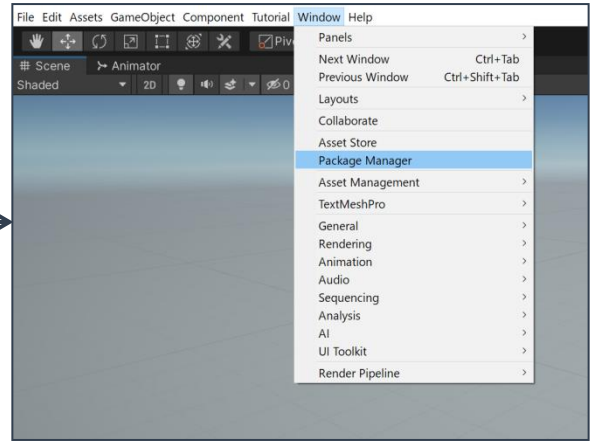
URP:
Kind of a graphic system to provide beautiful graphic with good performance.

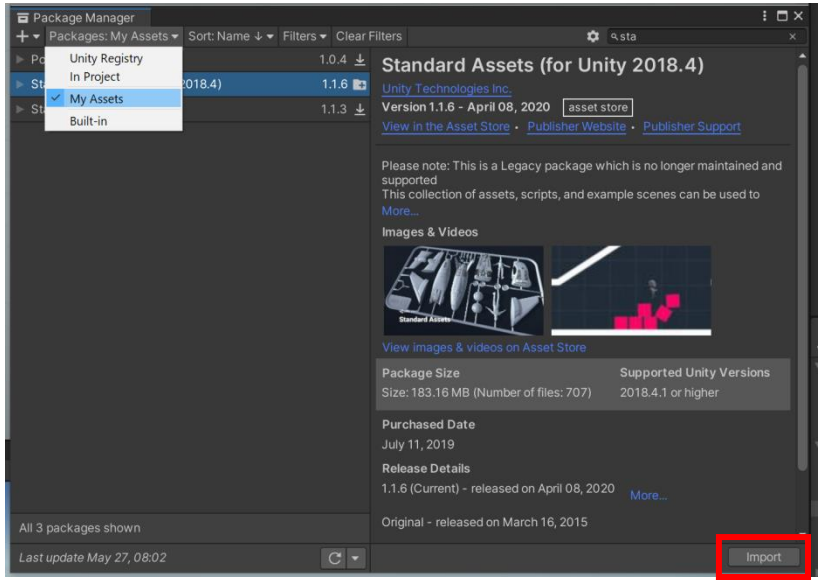
Setup for New Game!

Import assets.

1. Open “Assets/Scenes/SampleScene”.
2. Delete “Example Assets” from Hierarchy.
3. Get bellow assets via Web browser with **Add to My Assets** button.
    - Standard Assets: https://assetstore.unity.com/packages/essentials/asset-packs/standard-assets-for-unity-2018-4-32351
    - Stylized Plants Kit: https://assetstore.unity.com/packages/3d/environments/stylized-plants-kit-188109
    - Farm Animals Set: https://assetstore.unity.com/packages/3d/farm-animals-set-97945
4. Open Package Manager in Unity.
5. Select “My assets” option and find “Standard Assets”.
6. Click “Download” button and wait finishing it.

   The button will be changed into “Import”. Then, click it again.



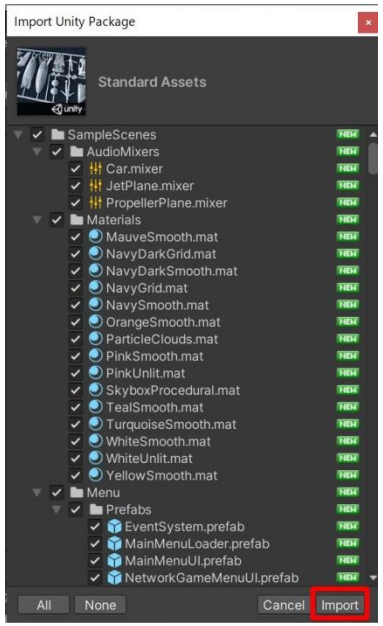


Setup for New Game!

Import assets.

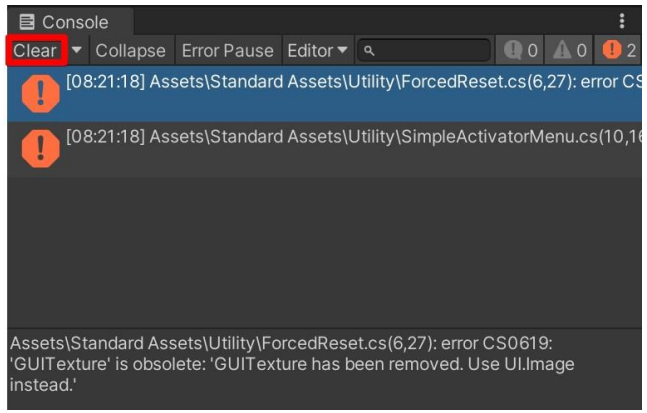
1. Import menu will be shown. Click “Import” and wait.
2. Warnings and errors will appear in your Console.

   Click “Clear” button to find core problems.
3. Double click the errors and fix the problems according to bellow images.




```
[RequireComponent(typeof (UnityEngine.UI.Image))]
public class ForcedReset : MonoBehaviour
{
    private void Update()
    {
        // if we have forced a reset ...
        if (CrossPlatformInputManager.GetButtonDown("ResetObject"))
        {
            //... reload the scene
            SceneManager.LoadScene(SceneManager.GetSceneAt(0).name);
        }
    }
}
```

```
namespace UnityStandardAssets.Utility
{
    public class SimpleActivatorMenu : MonoBehaviour
    {
        // An incredibly simple menu which, when given references
        // to gameobjects in the scene
        public UnityEngine.UI.Text camSwitchButton;
        public GameObject[] objects;


        private int m_CurrentActiveObject;
```



↑ This message tell you how to fix it.

When no errors appeared after click “Clear” button, go ahead to next page.

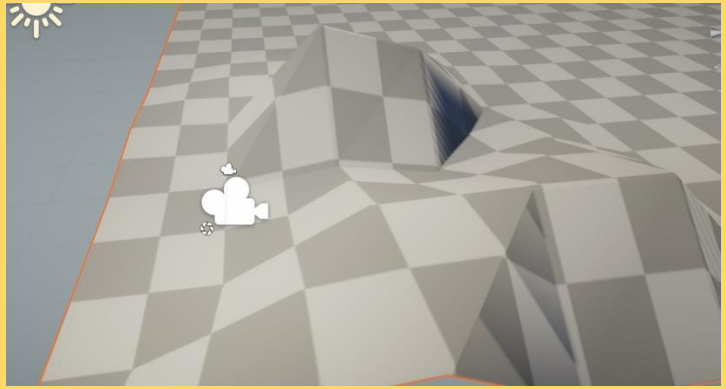
Setup for New Game!

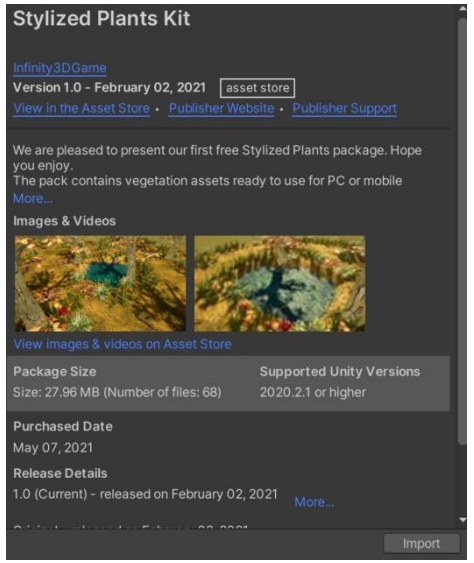
Import assets.

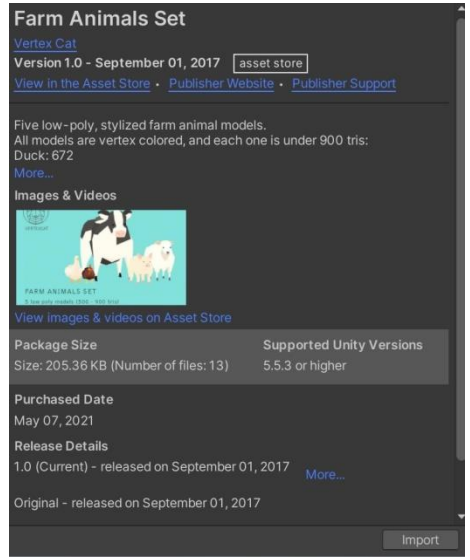
1. Import “Stylized Plants Kit” and “Farm Animals Set” too.
   It also shows warnings but you can clear all of it.
2. Check any warnings and errors remained.
3. Create -> 3D Object -> Terrain on Hierarchy.

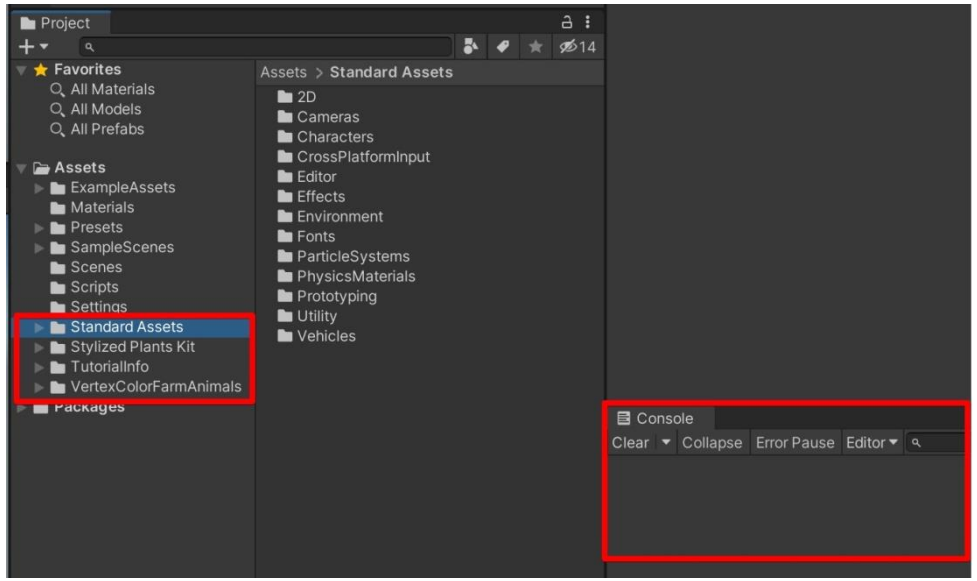
Terrain:

- Geographic system of Unity.
- It can create large 3D map like open-world game.
- The map can be formed by pushing, pulling, and coloring.





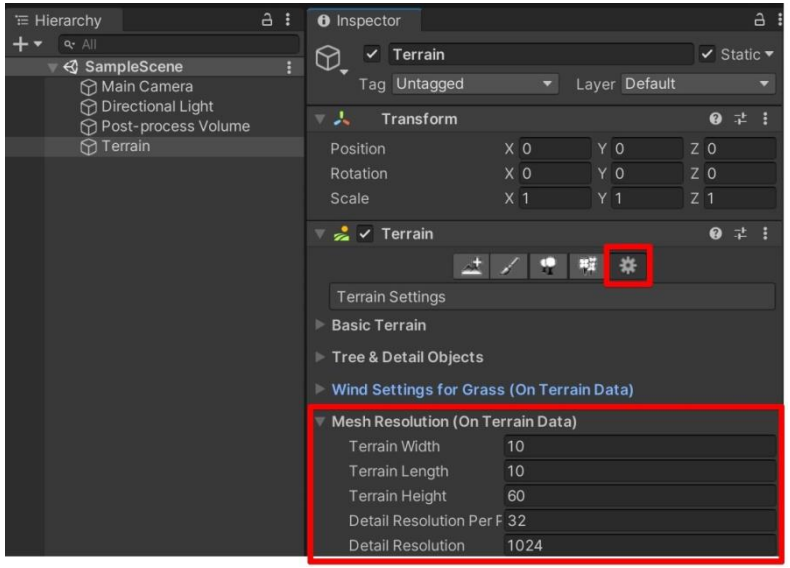


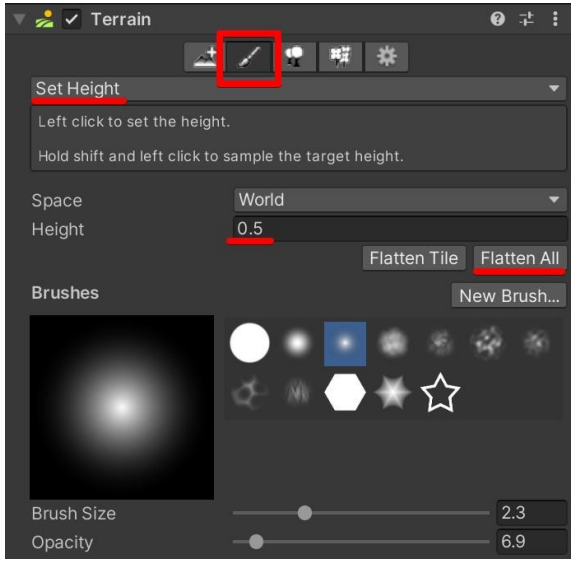


Create your Map!

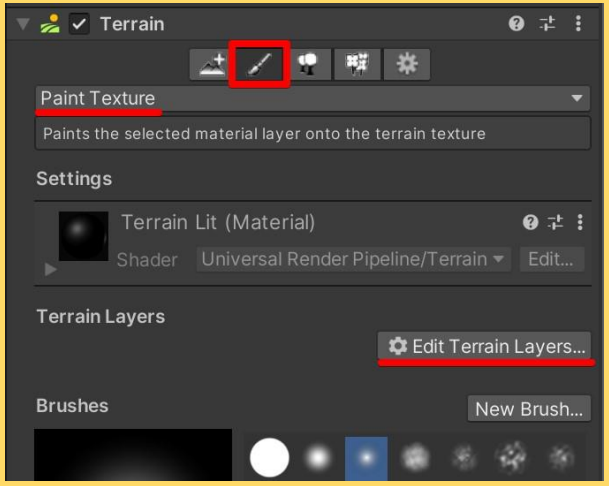
With Terrain.

1. Edit Terrain settings by Inspector.
   - Map size settings
   - Base height settings

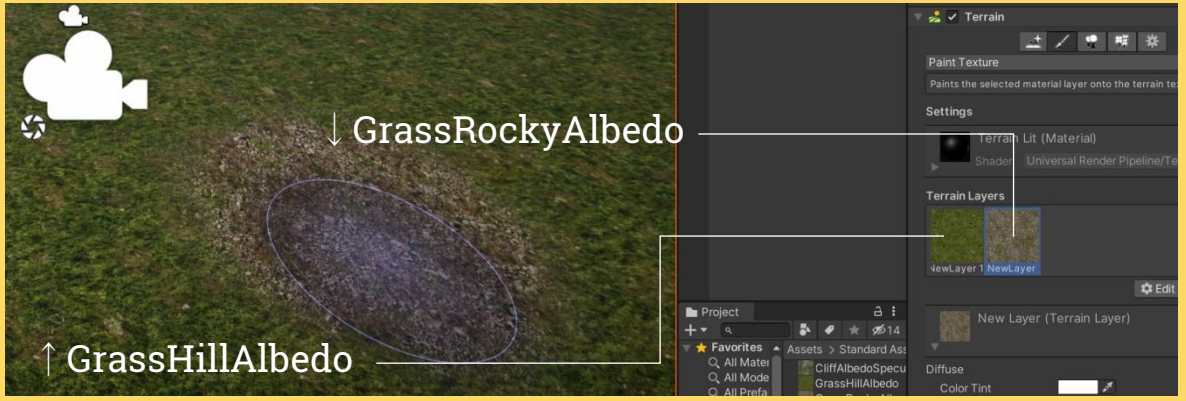




2. Set ground texture by Inspector.
   - Select paint tool and “Paint Texture”.
   - Select “Edit Terrain Layers” -> “Create Layer…”.
   - Search and select “GrassHillAlbedo”.

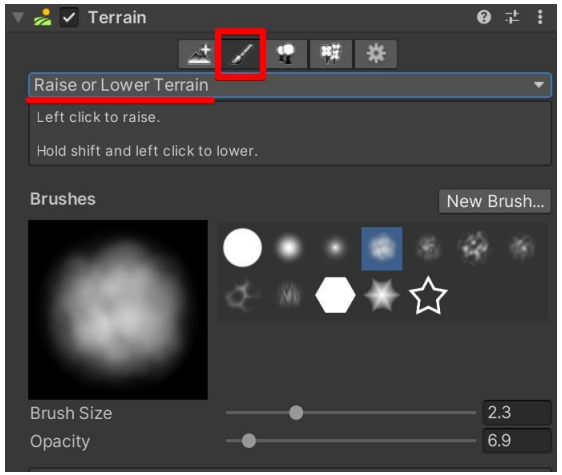


You can add more layers to paint different biomes like sand and desert with selected brush.

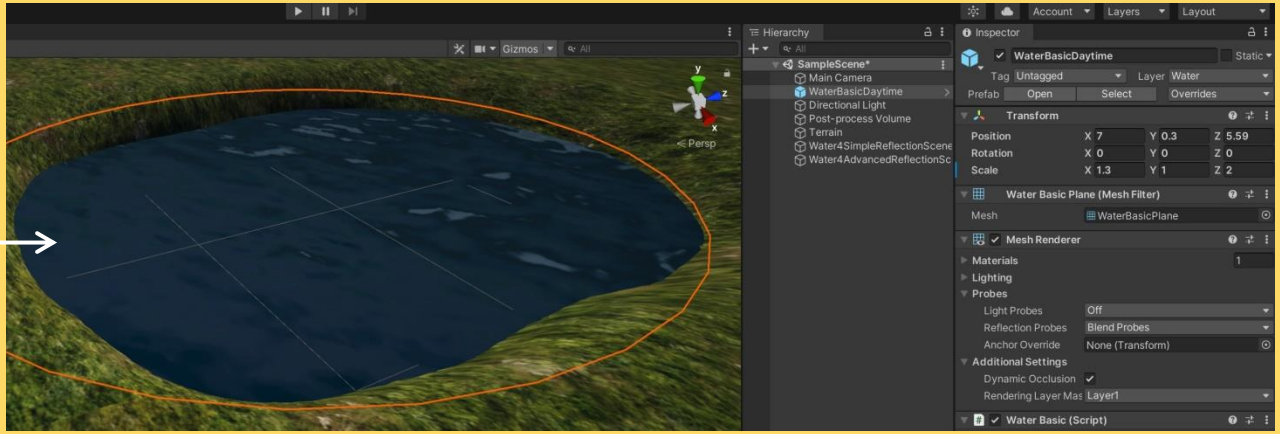


Create your Map!

With Terrain.



1. Form your Map.
   - Select “Paint Terrain” tool by Inspector.
   - Select “Raise or Lower Terrain”.
   - Select brush and the size/opacity as you like.
   - Draw your Terrain with Left click!
   - You can lower the Terrain with Shift + Left click.
2. Place water.
   - Select “Assets/Standard Assets/Environment/Water (Basic)/Prefabs/WaterBasicDaytime”.
   - Drag and Drop the Prefab into dent you made.
   - Adjust the water prefabs position, rotation and scale.



Create your Map!

With Terrain.

1. Create your map as you like!
   - Plants: “Assets/Stylized Plants Kit/Prefabs”.
   - Animals: “Assets/VertexColorFarmAnimals/Prefabs”. (You need at least 1 sheep)
   - Prefabs may be too big so you should adjust the size.
     You can select many objects at once by using shift + click. It is useful to apply same parameters to objects.
   - You can also use “Empty objects” as a parent to organize your objects.
   - This is Homework.

Tips: On SimpleCameraController, you can move camera with right click + WASD key during play the game.

2.Basic programming with Unity

Class Plan

1. Introduction
2. Hello world in Unity
3. Basic programming
4. Object oriented programming
5. Interface programming
6. **Programming design 1**
7. **Programming design 2**
8. **Improvement and testing 1**
9. **Improvement and testing 2**
10. Summary

6
Programming design 1

6. Programming design 1

Class summary

1. Explain game design.
2. Explain original game development.
3. Explain class design (class diagram).

Coding Rules: Examples

Target	Major Rules	Examples
Class name	a. UpperCamel	a. SampleClass
Class variables	a. LowerCamel b. Snake_case c. Underscore	a. sampleVariable b. sample_variable c. _sampleVariable (Only for private variables)
Bool variables	a. Is_bool	a. is_run
Methods	a. UpperCamel	a. SampleMethod()
Constants	a. All-Caps b. UpperCamel	a. MAX_NUMBER b. MaxNum
Scene name	a. UpperCamel	a. TitleScene
Object name	a. UpperCamel	a. CactusNails
If Statement	a. if (target < condition)	a. if (num < 10)
Comments	You don't' need to write all of contents. Good code & Good name explains the role or function itself.	
Scope	Try to minimize the range.	
Hardcoding	Don't use it. You should use constants.	
Core mind	Simple and Dry. Functions should be under 30〜50 lines.	

Last class Homework

We develop this game between class 6 ~ 9.
Tips: On SimpleCameraController, you can move camera with right click + WASD key.

Game Design

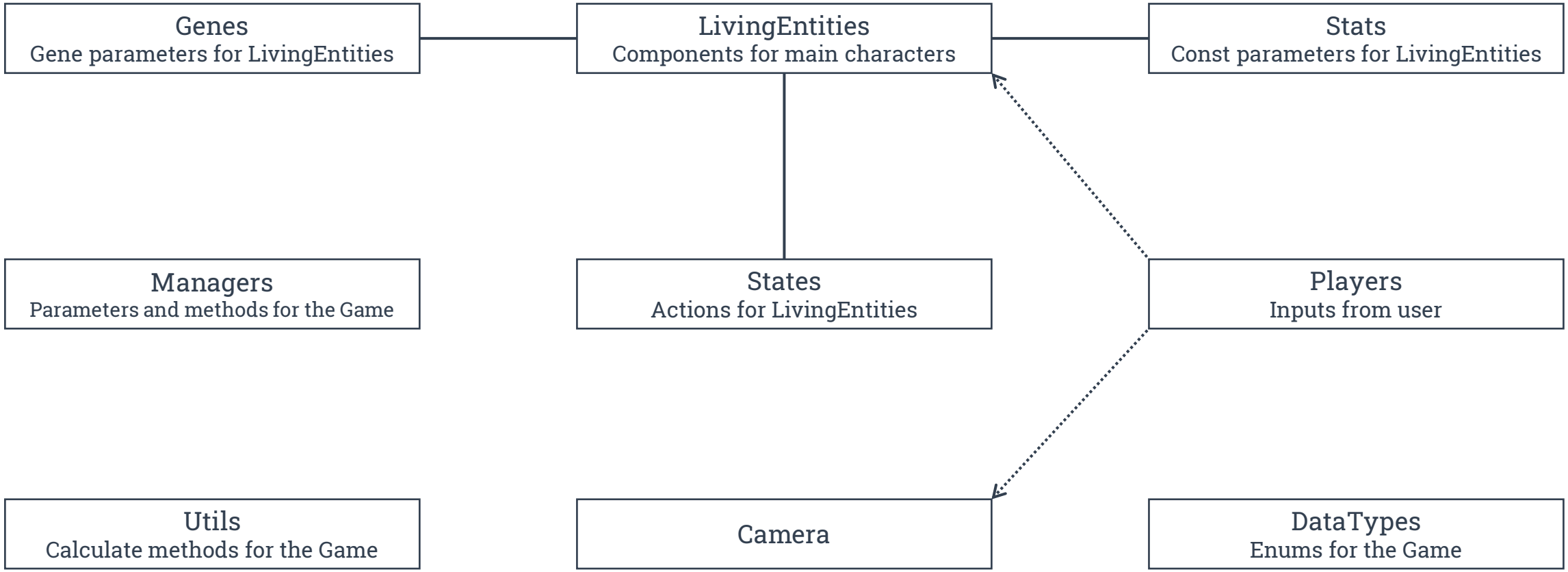
Thanking about "Experience"

1. Good game experiences are architected from psychology, anthropology, design, and emotion.
   Thinking about it is "Game Design".
2. The main element that determines the game design is called the theme or concept.
   This game's theme is "Real Ecosystem".
3. We have to explain "Why the functions / parameters are needed".

- Game type: Simulation
- Visual: 3D
- Real points(Concepts):
  1. Diets of animals
  2. Gene of animals and plants
  3. Calorie management of world
- Deformation points:
  1. Time related things
  2. Animation of animals

Game Design

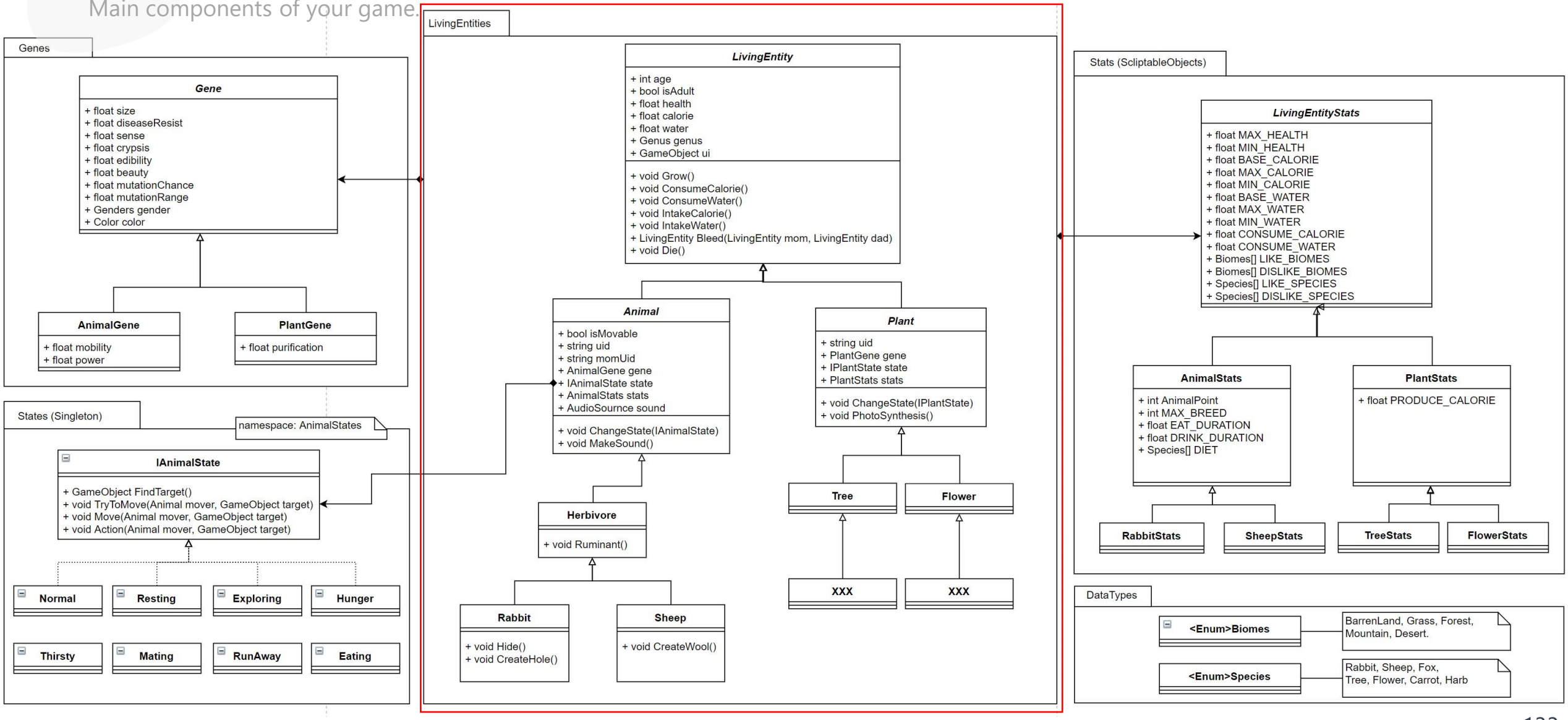
Main components of your game.



Game Design

Main components of your game.

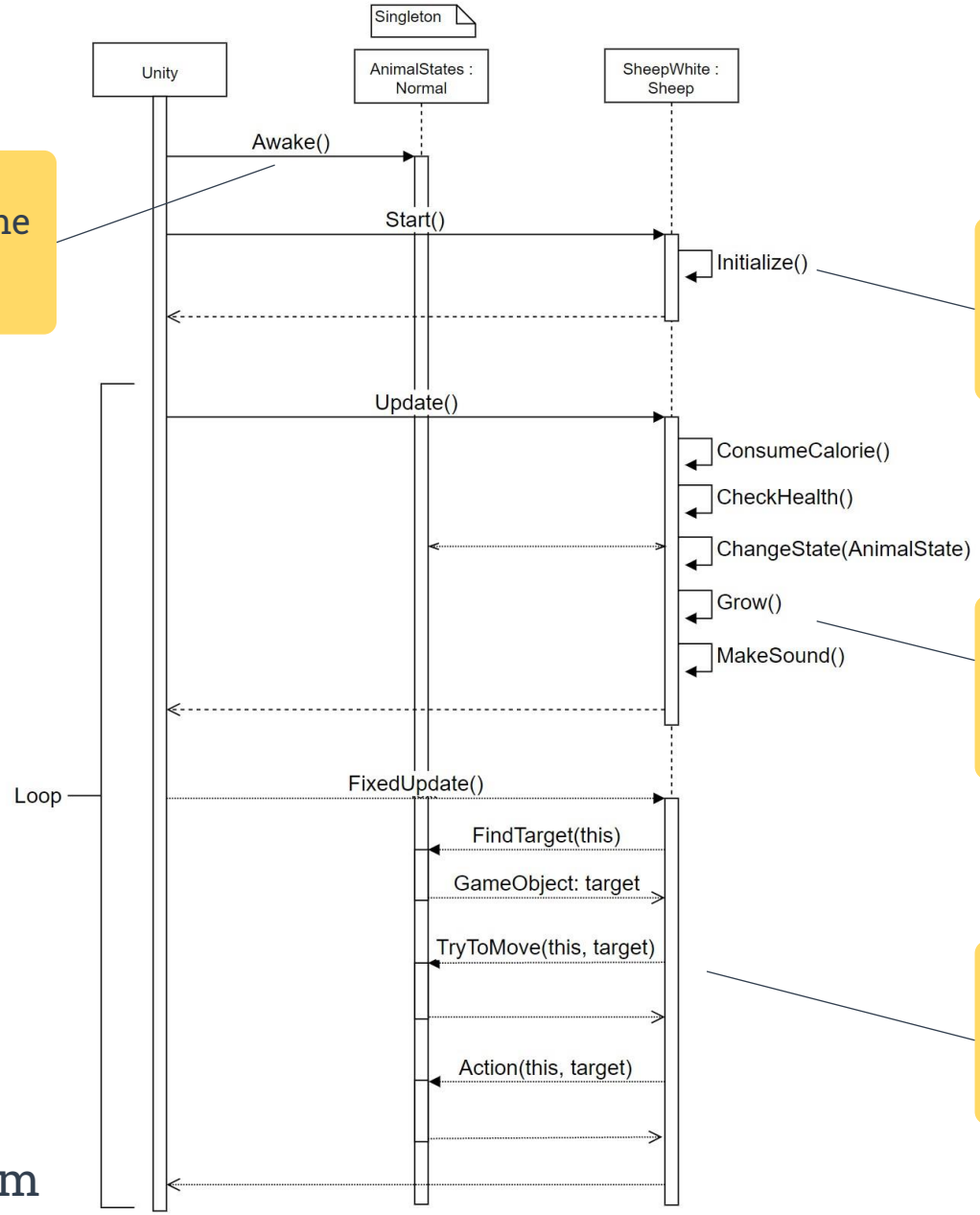
First step



Game Design

Main sequence of your game.

Instantiate Normal (or, Hunger) class on Awake() by attaching the script component to AnimalStates GameObject.



Initialize Sheep's health, calorie, age and size etc.

Update the sheep's parameters and state per frame.

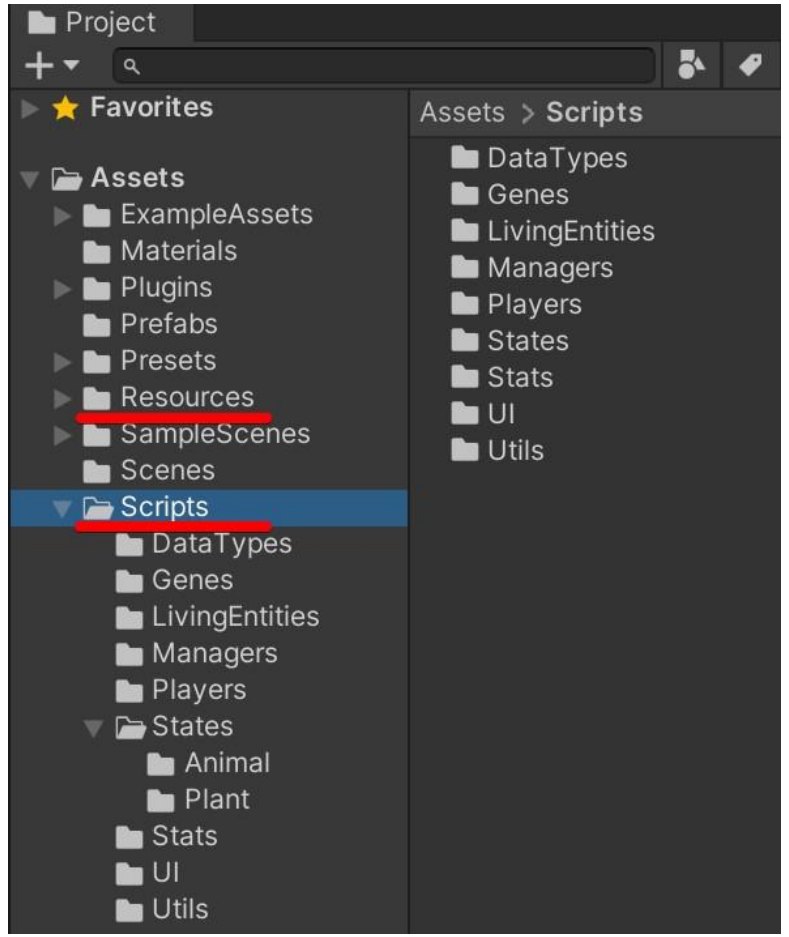
Conduct actions with the current state.
Methods: FindTarget(), TryToMove and Action().

Sequence Diagram

Game Development

Setup

1. Open Mygame01_ecosystem project.
2. Create bellow folders under Scripts folder. ⟶
   - LivingEntities
   - Genes
   - Stats
   - Managers
   - DataTypes
   - States (This folder has more 2 folders: Animal and Plant)
   - Players
   - Utils
   - UI
3. Create Resources Folder on same layer of Scripts. ⟶



Game Development

LivingEntities

1. Select LivingEntities folder.
2. Create LivingEntity script according to this image.

   Some lines are commentouted to avoid compile errors or to memo TODO (Please write the line) .

```
public abstract class LivingEntity : MonoBehaviour
{
    public int age;
    public bool isAdult;
    public float health;
    public float calorie;
    public float water;
    public GameObject ui;
    // public Genus genus;

    public abstract void Grow();
    public abstract void ConsumeCalorie();
    public abstract void ConsumeWater();
    public abstract void InTakeCalorie();
    public abstract void InTakeWater();
    // public abstract LivingEntity Bleed(Gene momGene, Gene dadGene);

    public void Die()
    {
        // TODO: Play dying animation and wait few seconds
        Destroy(this.gameObject);
    }
}
```

This Class is Base (Super) class of Animals.

In this reason, this class is declared as Abstract.

Grow() ~ InTakeWater() methods are also declared as Abstract because each animals may perform different functions.
It will be implemented in each Animal class.

On the other hand, Die() method may be same function of all animals so it is implemented here.

Game Development

LivingEntities

1. Create Animal script according to this image (Don't forget: using AnimalStates).

   Some errors will occur but its ok now.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using AnimalStates;

public abstract class Animal : LivingEntity
{
    public bool isMovable = true;
    public string uid = System.Guid.NewGuid().ToString();
    public string momUid = null;
    public AnimalGene gene;
    public IAnimalState state;
    public AnimalStats stats;
    public AudioSource sound;

    public void ChangeState(IAnimalState state)
    {
        this.state = state;
    }

    public void MakeSound()
    {
        if (sound != null)
        {
            this.sound.Play();
        }
    }
}
```

uid is Unique id as string.
It can be created with Syste.Guid.NewGuid().

uid is useful to find target object in Debug.log().

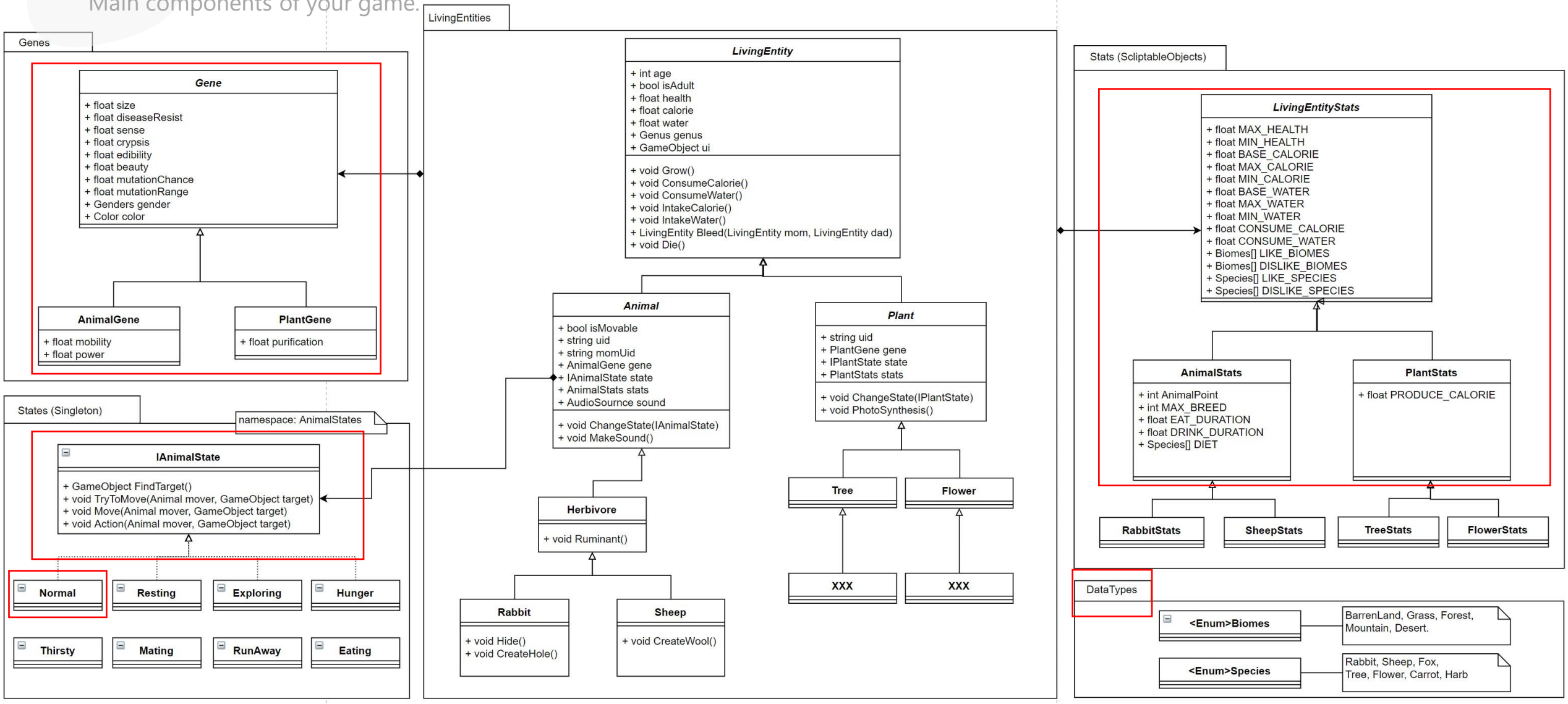
In this game, uid is used to find mother animal as momUid. Child animals will follow the mother.

sound will be used to play their voice like "Baaaa"(Sheep).

Game Design

Main components of your game.

Create some of classes to avoid compile errors.



Game Development

Genes.

1. Create Gene related scripts in Genes folder. You can copy and paste.
2. Create Gender scripts in DataTypes folder.

Gene.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public abstract class Gene
{
    public float size;
    public float diseaseResist;
    public float sense;
    public float crypsis;
    public float edibility;
    public float beauty;
    public float mutationChanve;
    public float mutationRange;
    public Genders.Type gender;
    public Color color;
}
```

AnimalGene.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AnimalGene : Gene
{
    public float mobility;
    public float power;

    public AnimalGene(Genders.Type gender)
    {
        size = 1.0f;
        diseaseResist = 1.0f;
        sense = 1.0f;
        crypsis = 1.0f;
        edibility = 1.0f;
        beauty = 1.0f;
        mutationChanve = 1.0f;
        mutationRange = 1.0f;
        this.gender = gender;
        color = Color.black;
        mobility = 1.0f;
        power = 1.0f;
    }
}
```

Constructor to initialize the gene parameters.

PlantGene.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlantGene : Gene
{

}
```

Gender.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Genders : MonoBehaviour
{
    public enum Type
    {
        Female = 0,
        Male = 1,
        None = 2,
    }
}
```

Game Development

States.

1. Create State related scripts in States/Animal folder.

   You can copy and paste.

IAnimalState.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace AnimalStates
{
    public interface IAnimalState
    {
        public GameObject FindTarget(Animal mover);
        public void TryToMove(Animal mover, GameObject target);
        public void Move(Animal mover, GameObject target);
        public void Action(Animal mover, GameObject target);
    }
}
```

Define namespace to avoid compile error by exists same class name: "Normal". The class will be created for Plants.

We will add functions to this class to express "Normal" state actions.

Auto implement by IAnimalState interface.

Normal.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Threading.Tasks;
using UnityEngine;
using AnimalStates;

namespace AnimalStates
{
    public class Normal : MonoBehaviour, IAnimalState
    {
        public void Action(Animal mover, GameObject target)
        {
            throw new System.NotImplementedException();
        }

        public GameObject FindTarget(Animal mover)
        {
            throw new System.NotImplementedException();
        }

        public void Move(Animal mover, GameObject target)
        {
            throw new System.NotImplementedException();
        }

        public void TryToMove(Animal mover, GameObject target)
        {
            throw new System.NotImplementedException();
        }
    }
}
```

Game Development

Stats.

1. Create Stats related scripts in Stats folder. You can copy and paste.
2. Create Species scripts in DataTypes folder. Then, Check no errors shown in your Unity project.

LivingEntityStats.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LivingEntityStats : ScriptableObject
{
    public float MAX_HEALTH = 100;
    public float MIN_HEALTH = 0;
    public float BASE_CALORIE;
    public float MAX_CALORIE;
    public float MIN_CALORIE;
    public float CONSUME_CALORIE;
    public float BASE_WATER;
    public float MAX_WATER;
    public float MIN_WATER;
    public float CONSUME_WATER;
    //public Biomes.Type[] LIKE_BIOMES;
    //public Biomes.Type[] DISLIKE_BIOMES;
    public Species.Type[] LIKE_SPECIES;
    public Species.Type[] DISLIKE_SPECIES;
}
```

AnimalStats.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[CreateAssetMenu(menuName = "MyScriptable/Create
AnimalStats")]
public class AnimalStats : LivingEntityStats
{
    public int AnimalPoint; // Score or Money in the game
    public int MAX_BREED;
    public float EAT_DURATION;
    public float DRINK_DURATION;
    public Species.Type[] DIET;
}
```

[CreateAssetMenu()] is format to create ScripdableObject (Kind of an asset) on Unity editor.

PlantStats.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[CreateAssetMenu(menuName = "MyScriptable/Create
PlantStats")]
public class PlantStats : LivingEntityStats
{
    public float PRODUCE_CALORIE;
}
```

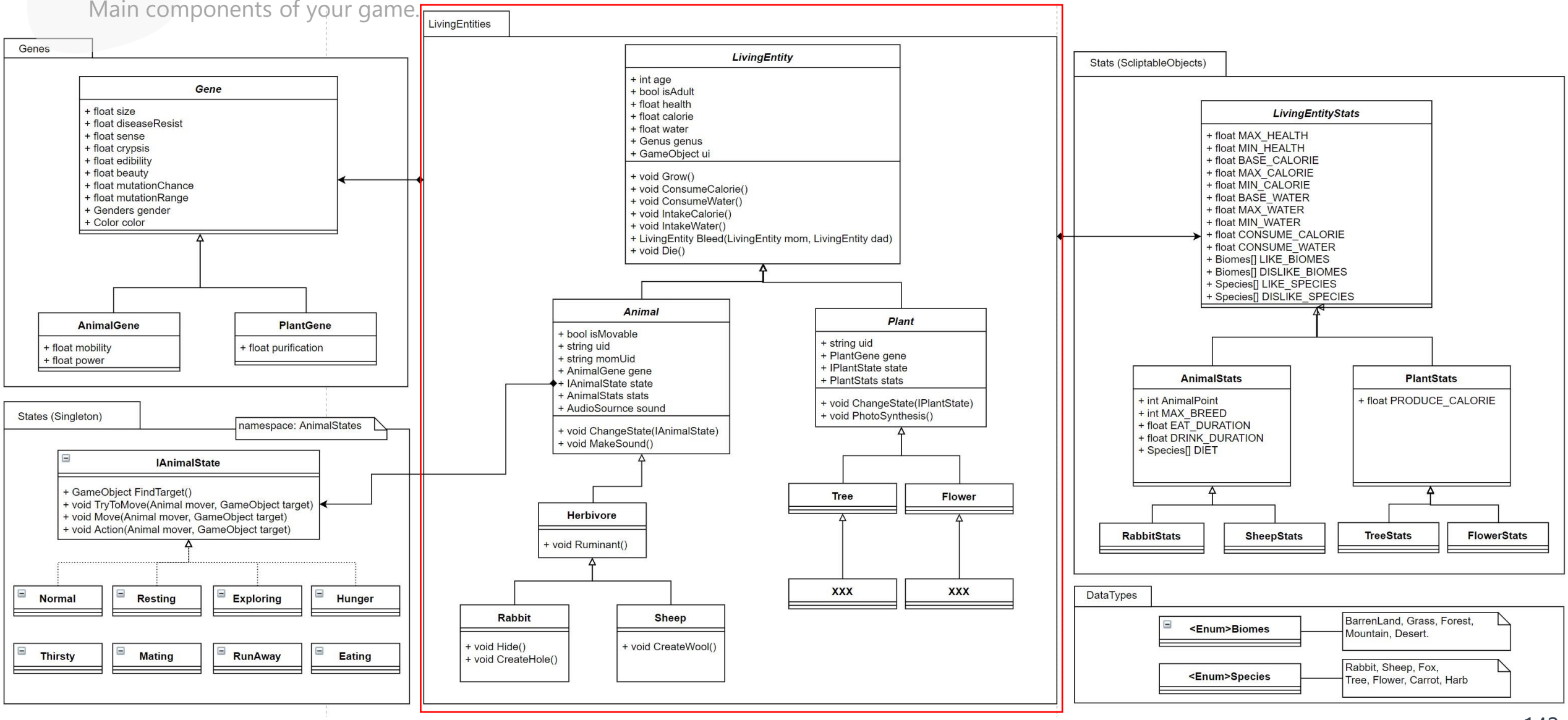
Species.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Species : MonoBehaviour
{
    public enum Type
    {
        Flower,
        Bush,
        Grass,
        Sheep,
    }
}
```

Game Design

Main components of your game.

Proceed to implement Animals.



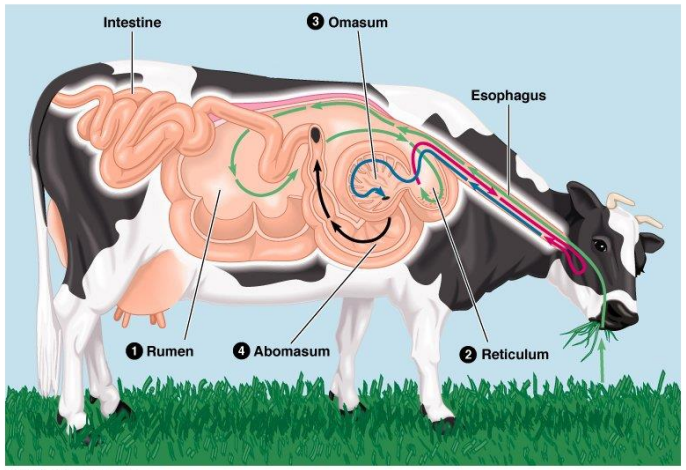
Game Development

LivingEntities.

1. Create Herbivore script in LivingEntities folder. You can copy and paste.
2. Create Sheep script in LivingEntities folder. You can copy and paste.

Herbivore.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public abstract class Herbivore : Animal
{
    public abstract void Ruminant();
}
```



Ruminant digestion system of Herbivore.
They can get calories from grass by this system.

At Start(), the sheep objects conduct Initialize(). It is basic knowledge for programming.

At Update(), the sheep consume their calories. When their HP is damaged, it may heal it. If they were hungry, their states will be changed to “Hungry”.

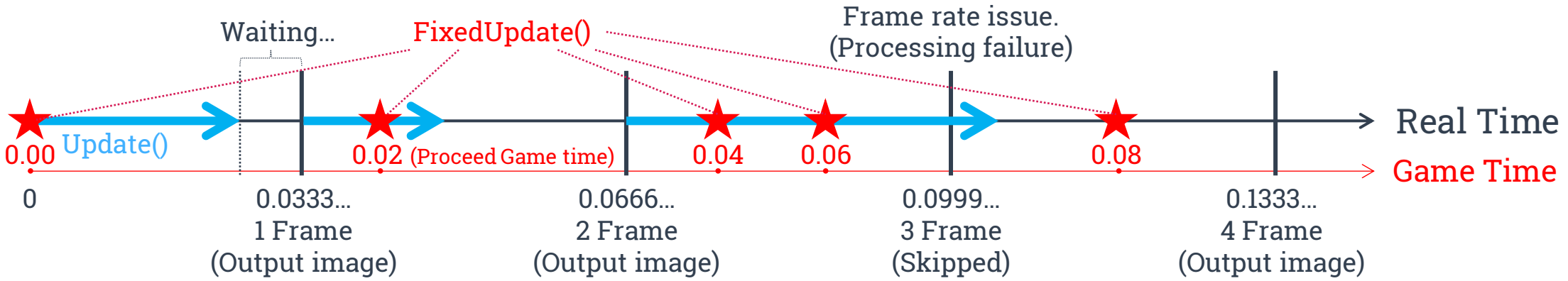
At FixedUpdate(), the sheep try to take actions like find something and move.

Sheep.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using AnimalStates;

public class Sheep : Herbivore
{
    public Rigidbody rb = null;
    public GameObject target = null;

    void Start()
    {
        Initialize();
    }

    void Update()
    {
        ConsumeCalorie();
    }

    private void FixedUpdate()
    {

    }

    private void Initialize()
    {

    }

    public override void Ruminant()
    {
        throw new System.NotImplementedException();
    }

    public override void Grow()
    {
        throw new System.NotImplementedException();
    }

    public override void ConsumeCalorie()
    {
        throw new System.NotImplementedException();
    }

    public override void ConsumeWater()
    {
        throw new System.NotImplementedException();
    }

    public override void InTakeCalorie()
    {
        throw new System.NotImplementedException();
    }

    public override void InTakeWater()
    {
        throw new System.NotImplementedException();
    }
}
```

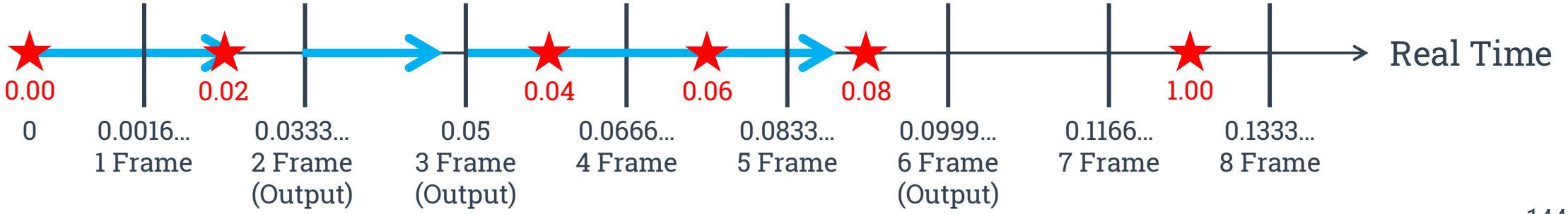
Update() and FixedUpdate()

- Case1 30FPS : 1 / 30 = 0.03333….



Default FixedUpdate() Time step = 0.02.
Each FixedUpdate() call proceeds Game time by 0.02 to get closer to elapse of real time.
After FixedUpdate(), Unity calculates physics so physics related functions should be write in FixedUpdate().

- Case2 60FPS : 1 / 60 = 0.01666….



Game Development

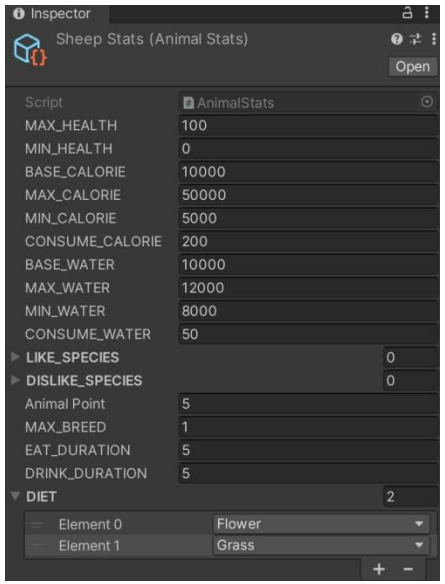
LivingEntities: Sheep - Initialize().

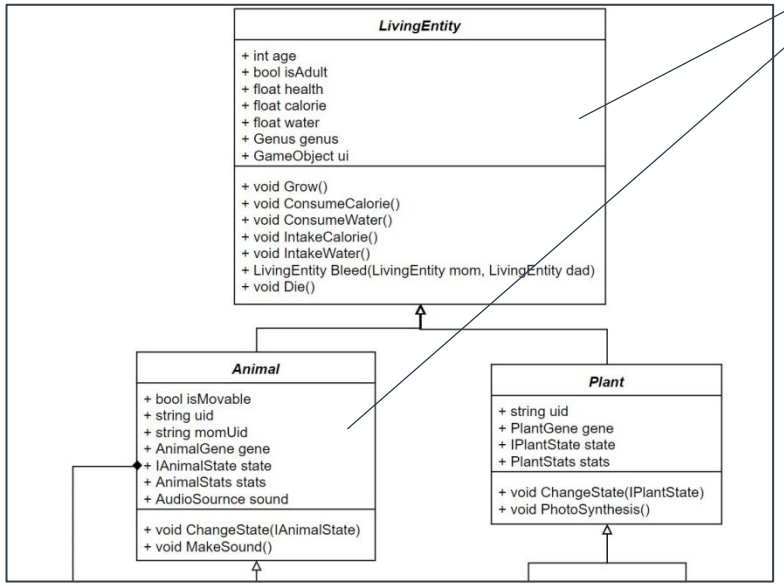
1. Edit Initialize() of Sheep script according to this image. →
2. Create SheepStats asset in Resources folder.
   - Right click on Resouces folder.
   - Select Create -> MyScriptable -> Create AnimalStats.
3. Input Stats info into SheepStats as you like.

   Below image is just sample.

Sheep.cs

```
public class Sheep : Herbivore
{
    public Rigidbody rb = null;
    public GameObject target = null;

    void Start()
    {
        Initialize();
    }

    void Update()
    {
        ConsumeCalorie();
    }

    private void FixedUpdate()
    {

    }

    private void Initialize()
    {
        // You have to place stats.asset files into "Resources" folder
        this.stats = Resources.Load<AnimalStats>("SheepStats");
        this.age = 0;
        this.isAdult = false;
        this.health = stats.MAX_HEALTH;
        this.calorie = stats.BASE_CALORIE;
        this.water = stats.BASE_WATER;
        this.isMovable = true;
        this.state = null; // Should be set to "Normal"
        this.sound = null; // Should be set sound file
        this.rb = this.GetComponent<Rigidbody>();
        this.stats.DIET = new Species.Type[] { Species.Type.Flower, Species.Type.Grass };
    }
```





When you initialize the class, you should set value to all field to avoid errors.

However, in this case, we skip it since some classes or assets are not ready yet.

[Tips] xxxStats (ScriptableObjects) have shared or same value parameters between all of the species. It is efficient for memory.

Game Development

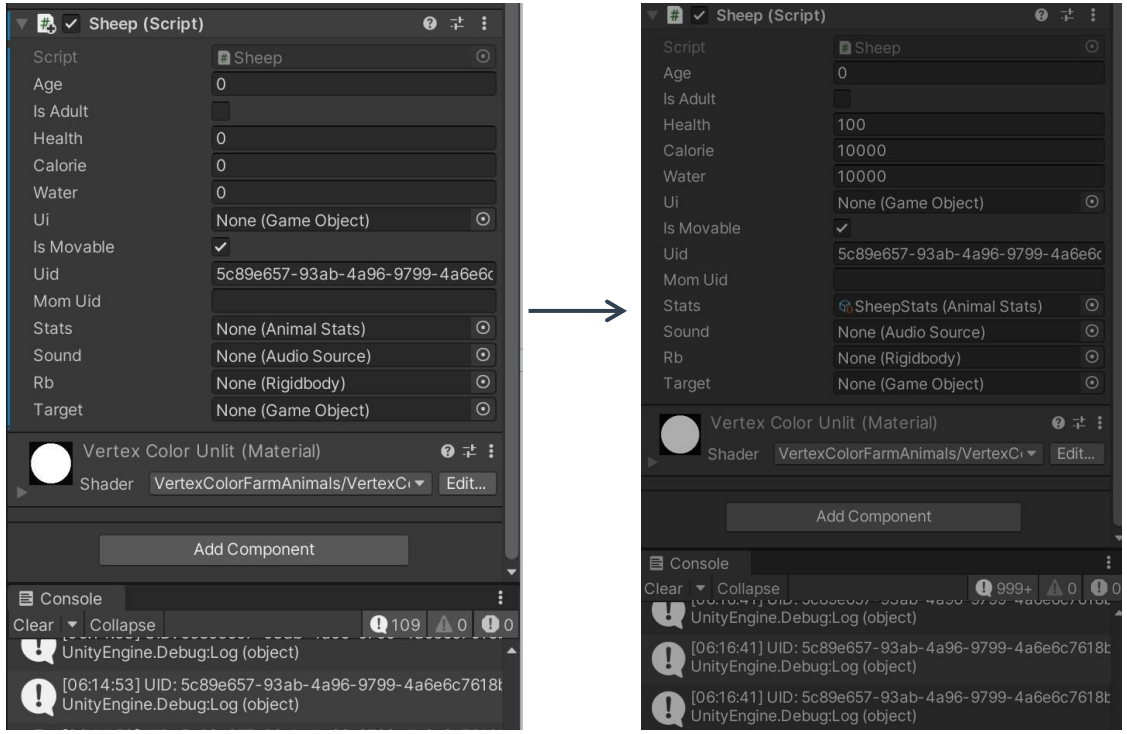
LivingEntities: Sheep - Initialize().

1. Edit ConsumeCalorie() of Sheep script according to this image. →

Sheep.cs

```
public override void ConsumeCalorie()
{
    Debug.Log("UID: " + this.uid + " calorie is " + this.calorie);
}
```

2. Check current implement.
   - Attach the Sheep script into your Sheep game object (Drag & Drop).
   - Play your game.
   - Make sure that the Initialization() is successful.
   - Debug.log() should be checked too.



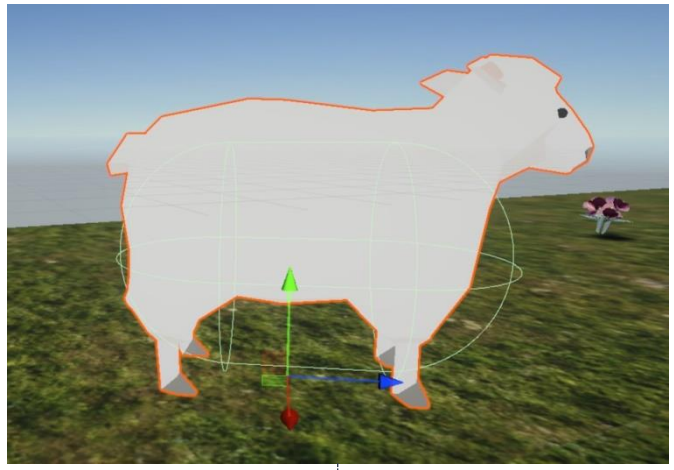
Game Development

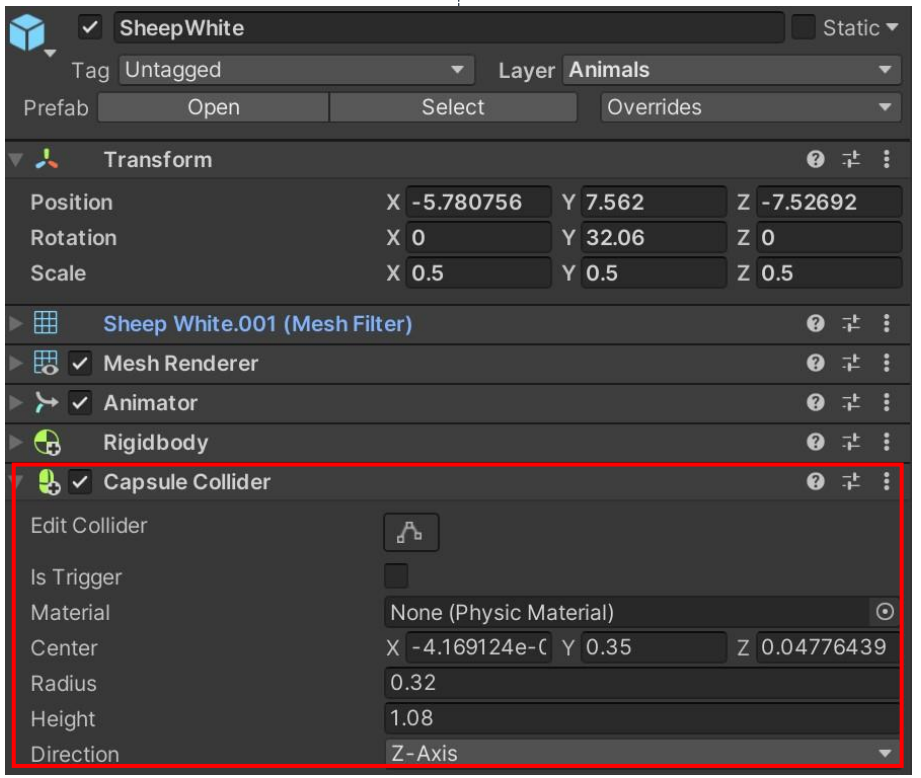
LivingEntities: Sheep - Initialize().

Game Development

LivingEntities: Sheep - Initialize().

1. Add Collider component to your Sheep.
   - Attach capsule collider.
   - Adjust the parameters according to the sheep's size.
2. Check current implement again.
   - Play your game.
   - Make sure that the Initialization() is successful.
   - Debug.log() should be checked too.





Game Development

LivingEntities: Sheep - Calorie.

1. Edit Sheep script according to this image. →
   - Edit Update().
   - Edit ConsumeCalorie().
2. Check current implement.
   - Play your game.
   - Make sure that the calorie will be consumed.
   - Make sure that the health will be damaged.

Mathf.Max(num1, num2) returns bigger one of the values. Min() does the opposite.
E.g. Mathf.Max(100, 500) returns 500.

Time.deltaTime returns elapsed real time from last frame as second.
E.g. {this.calorie = this.calorie - 200 * Time.deltaTime} in Update() means decrease the calorie by 200 per second.

[Tips] Time.deltaTime (s) * FPS (frame/s) = 1 (frame)

Animal.cs

```
public abstract class Animal : LivingEntity
{
    public bool isMovable = true;
    public string uid = System.Guid.NewGuid().ToString();
    public string momUid = null;
    public AnimalGene gene;
    public IAnimalState state;
    public AnimalStats stats;
    public AudioSource sound;

    public void ChangeState(IAnimalState state)
    {
        this.state = state;
    }

    public void MakeSound()
    {
        if (sound != null)
        {
            this.sound.Play();
        }
    }
}
```

Sheep.cs

```
void Update()
{
    ConsumeCalorie();

    if (this.health <= this.stats.MIN_HEALTH)
    {
        Die();
    }

    if (this.calorie < this.stats.BASE_CALORIE)
    {
        // Hunder class doesn't exsist now
        // ChangeState(new Hunger());
    }
    else
    {
        ChangeState(new Normal());
    }
}

public override void ConsumeCalorie()
{
    Debug.Log("UID: " + this.uid + " calorie is " + this.calorie);
    Debug.Log("UID: " + this.uid + " health is " + this.health);
    this.calorie = Mathf.Max(this.calorie - this.stats.CONSUME_CALORIE * Time.deltaTime, this.stats.MIN_CALORIE);
    if (this.calorie == this.stats.MIN_CALORIE)
    {
        // Damage
        this.health = Mathf.Max(this.health - Time.deltaTime, this.stats.MIN_HEALTH);
    }
    else
    {
        // Heal
        this.health = Mathf.Min(this.health + Time.deltaTime, this.stats.MAX_HEALTH);

    }
}
```

Equals

```
if (this.calorie <= this.stats.MIN_CALORIE)
    this.calorie = this.stats.MIN_CALORIE;
else
    this.calorie -= this.stats.CONSUME_CALORIE * Time.deltaTime;
```

Game Development

LivingEntities: Sheep - Calorie.

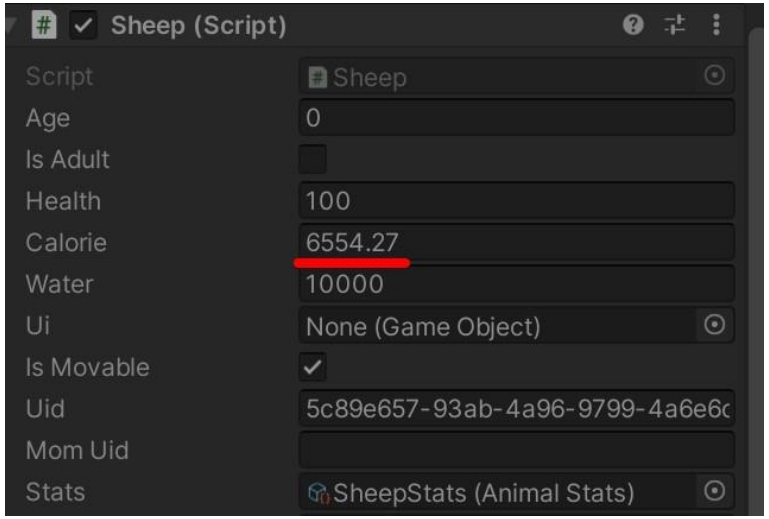
1. Edit Sheep script according to this image.
   - Edit Update().
   - Edit ConsumeCalorie().
2. Check current implement.
   - Play your game.
   - Make sure that the calorie will be consumed.
   - Make sure that the health will be damaged.
   - When the health become 0, the sheep will be dead.

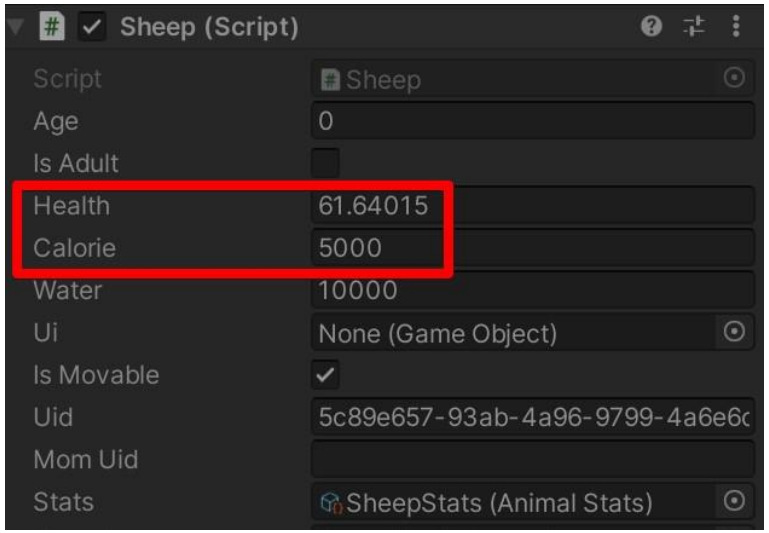
[Tips]

You can modify parameters of your sheep during Play mode.

Drag to left on Health parameter on Inspector.

It is fast method for debug.





Game Development

LivingEntities: Sheep – Grow().

1. Edit Sheep script according to this image. ⟶
   - Add public fields.
   - Edit Grow().
   - Edit Update()
2. Check current implement by playing your game.
   - Make sure that the sheep will grow (Scaled to MaxScale).
   - Make sure that the age will be incremented.

Vector3.one returns Vector3 (1, 1, 1).
When {nextScale == 0.9}, localScale of your sheep will be set to Vector3 (0.9, 0.9, 0.9) because 1 * 0.9 = 0.9.

Mathf.Min() is used to prevent the sheep's scale become too large.

If statement of {(int) currentTime % (int) soundSpan == 0} set interval. In this case, soundSpan = 7f so MakeSound() will be conduct every 7 seconds. The % operator returns the remainder of the division, so if the remainder is zero, it means divisible.

Sheep.cs

```
public class Sheep : Herbivore
{
    public Rigidbody rb = null;
    public GameObject target = null;

    public float minScale = 0.1f;
    public float maxScale = 1.0f;
    public float growSpan = 30f;
    public float soundSpan = 7f;
    public float currentTime = 0f;
    public float growRate = 0.1f;

    void Start()
    {
        Initialize();
    }

    public override void Grow()
    {
        float nextScale = Mathf.Min(this.transform.localScale.x + growRate, maxScale);
        this.transform.localScale = Vector3.one * nextScale;
        this.age++;
    }

    void Update()
    {
        ConsumeCalorie();

        // Should be contain as CheckHealth()
        if (this.health <= this.stats.MIN_HEALTH)
        {
            Die();
        }

        if (this.calorie < this.stats.BASE_CALORIE)
        {
            // Hunger class doesn't exsist now
            // ChangeState(new Hunger());
        }
        else
        {
            ChangeState(Normal.instance);
        }

        currentTime += Time.deltaTime;
        if (currentTime > growSpan)
        {
            currentTime = 0f;
            Grow();
        }
        if ((int)currentTime % (int)soundSpan == 0)
        {
            MakeSound();
        }
    }
```

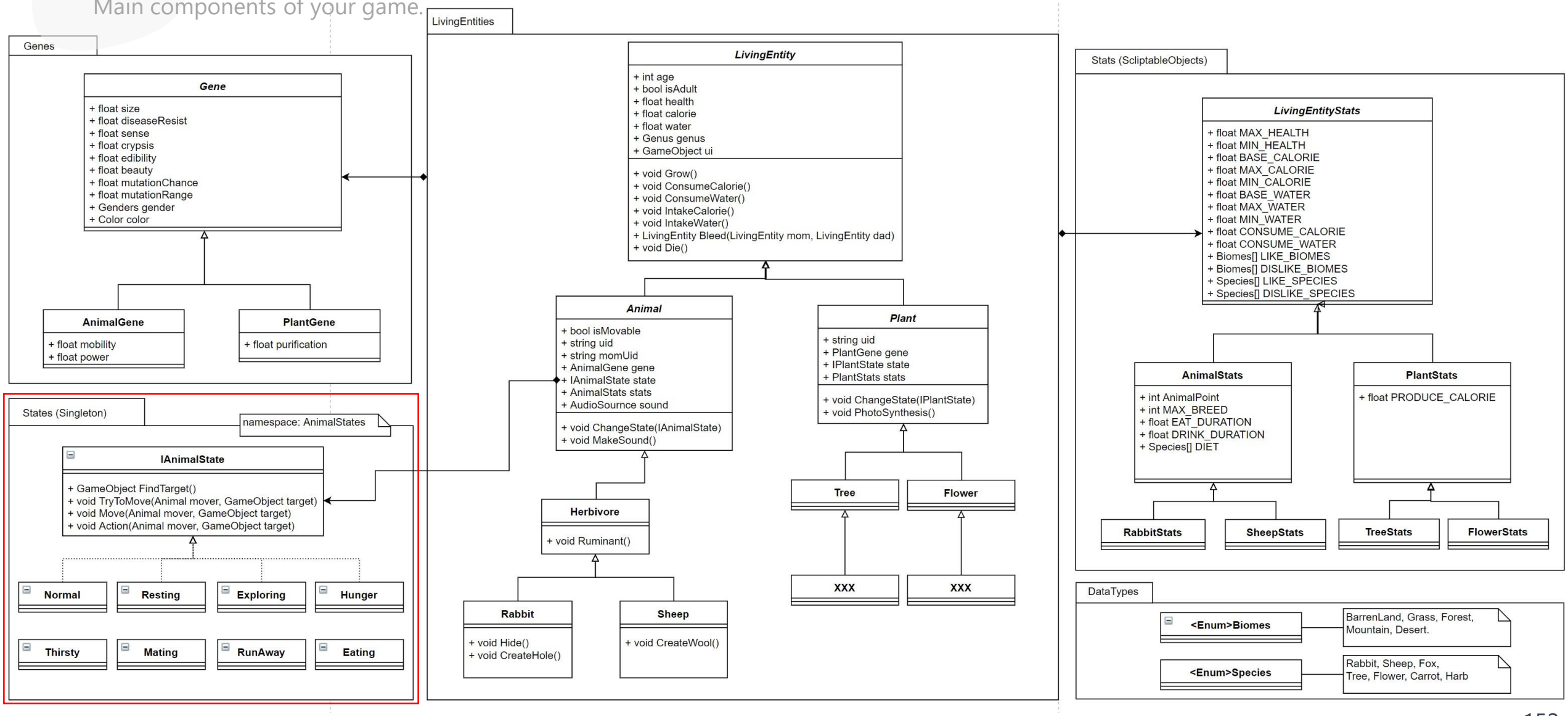
You can change these parameters as you want.

Count time and reset it to implement timer function.

Game Design

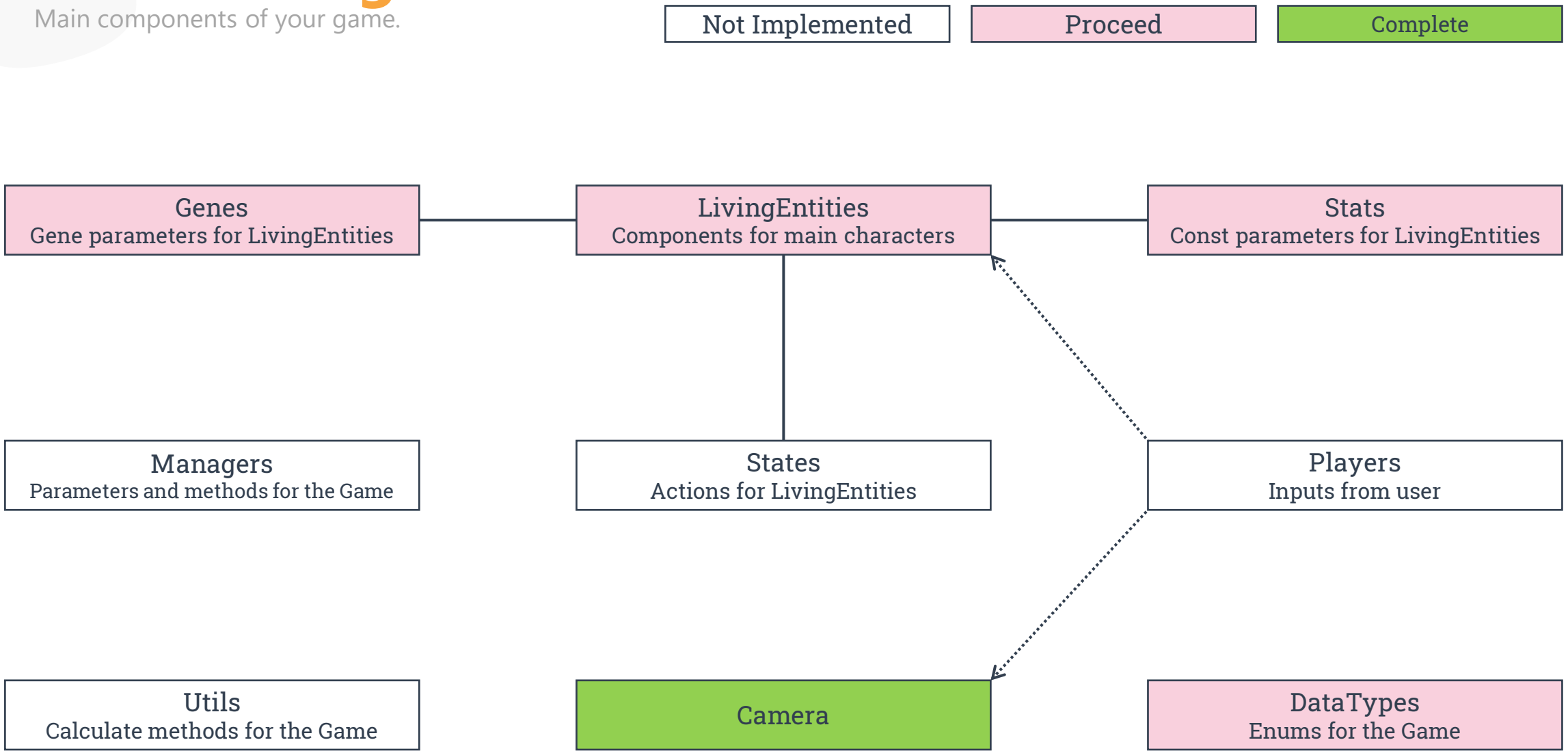
Main components of your game.

Next step: States (Normal, Hunger)



Game Design

Main components of your game.



2.Basic programming with Unity

Class Plan

1. Introduction
2. Hello world in Unity
3. Basic programming
4. Object oriented programming
5. Interface programming
6. **Programming design 1**
7. **Programming design 2**
8. **Improvement and testing 1**
9. **Improvement and testing 2**
10. Summary

7
Programming design 2

7. Programming design 2

Class summary

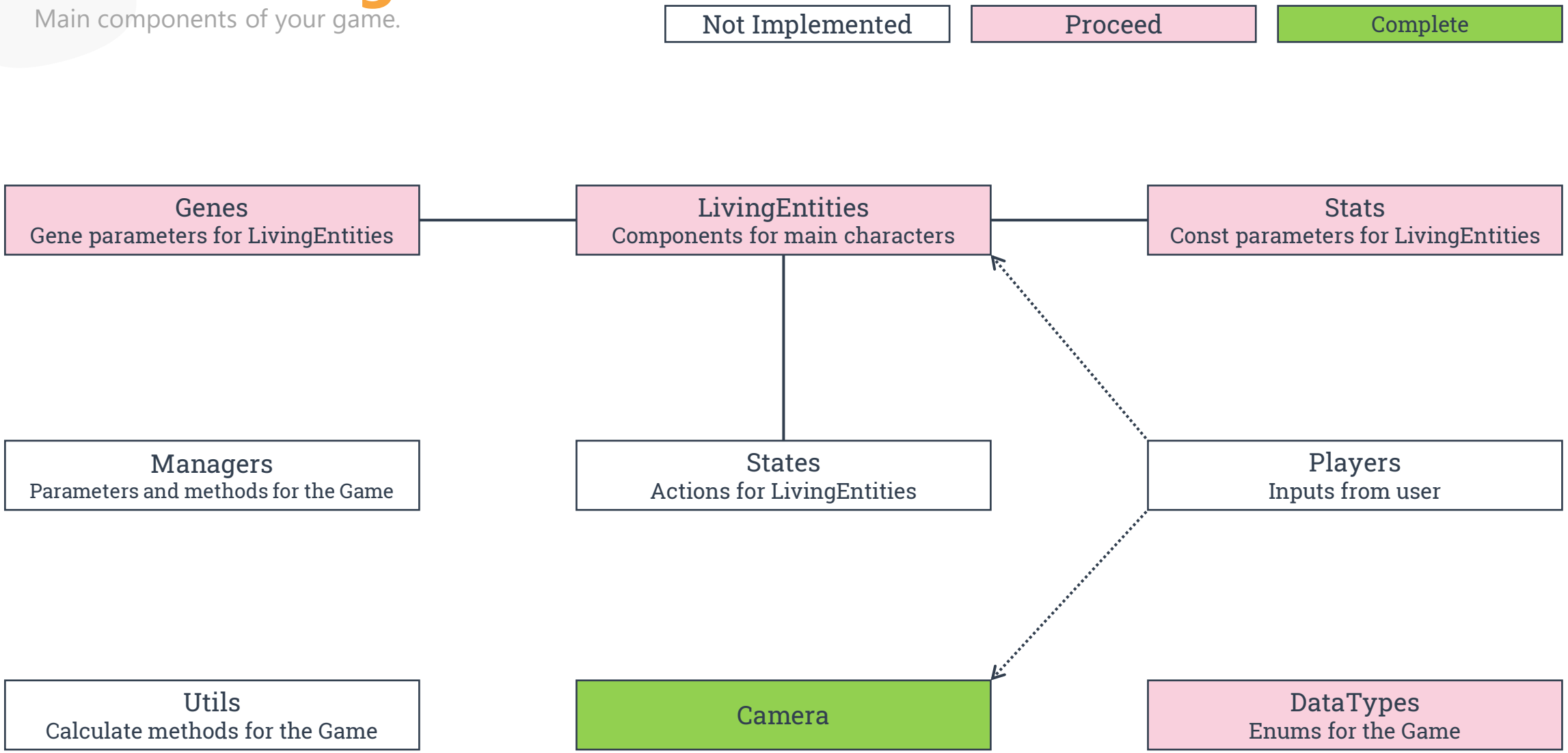
1. Explain original game development.
2. Explain class design (class diagram).
3. Explain the SOLID principle and design patterns.

Coding Rules: Examples

Target	Major Rules	Examples
Class name	a. UpperCamel	a. SampleClass
Class variables	a. LowerCamel b. Snake_case c. Underscore	a. sampleVariable b. sample_variable c. _sampleVariable (Only for private variables)
Bool variables	a. Is_bool	a. is_run
Methods	a. UpperCamel	a. SampleMethod()
Constants	a. All-Caps b. UpperCamel	a. MAX_NUMBER b. MaxNum
Scene name	a. UpperCamel	a. TitleScene
Object name	a. UpperCamel	a. CactusNails
If Statement	a. if (target < condition)	a. if (num < 10)
Comments	You don't' need to write all of contents. Good code & Good name explains the role or function itself.	
Scope	Try to minimize the range.	
Hardcoding	Don't use it. You should use constants.	
Core mind	Simple and Dry. Functions should be under 30〜50 lines.	

Game Design

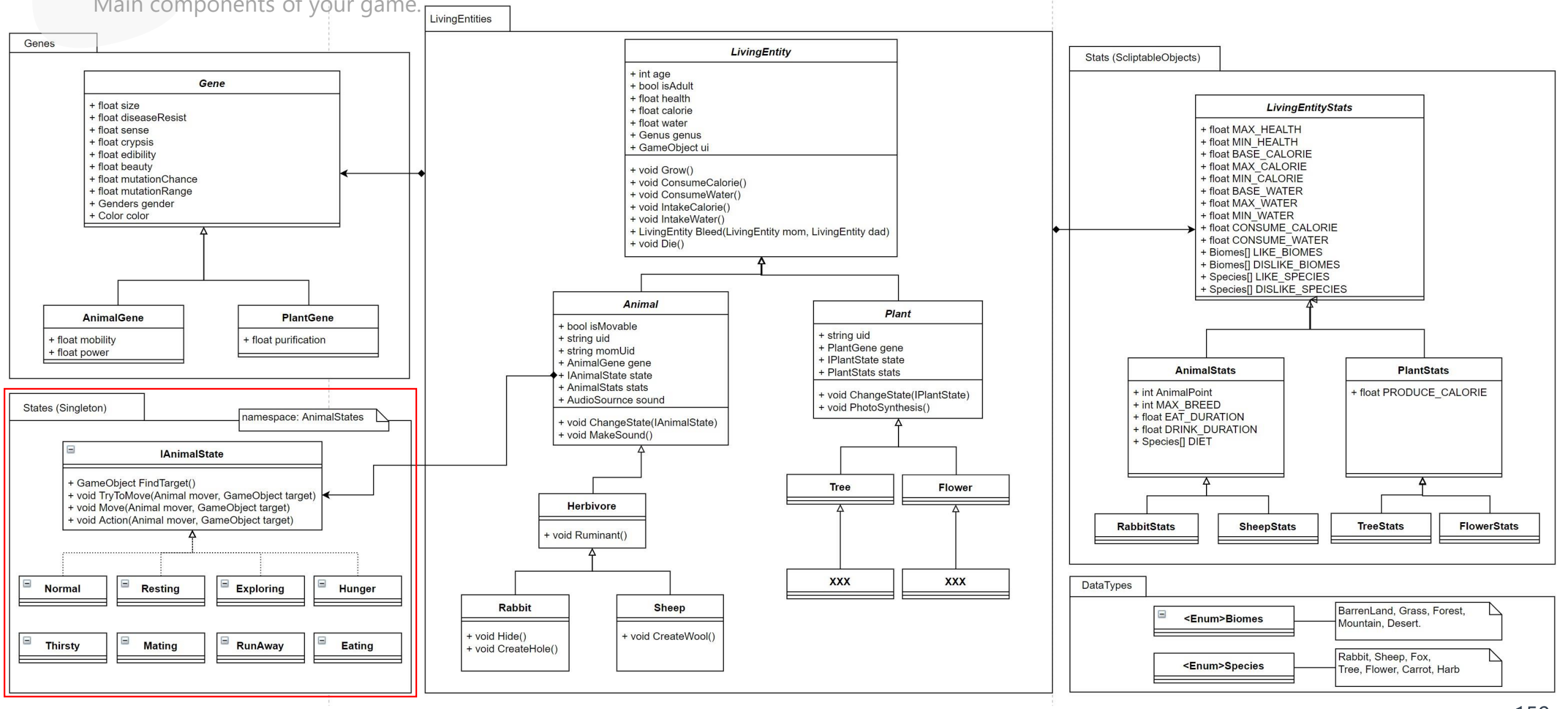
Main components of your game.



Game Design

Main components of your game.

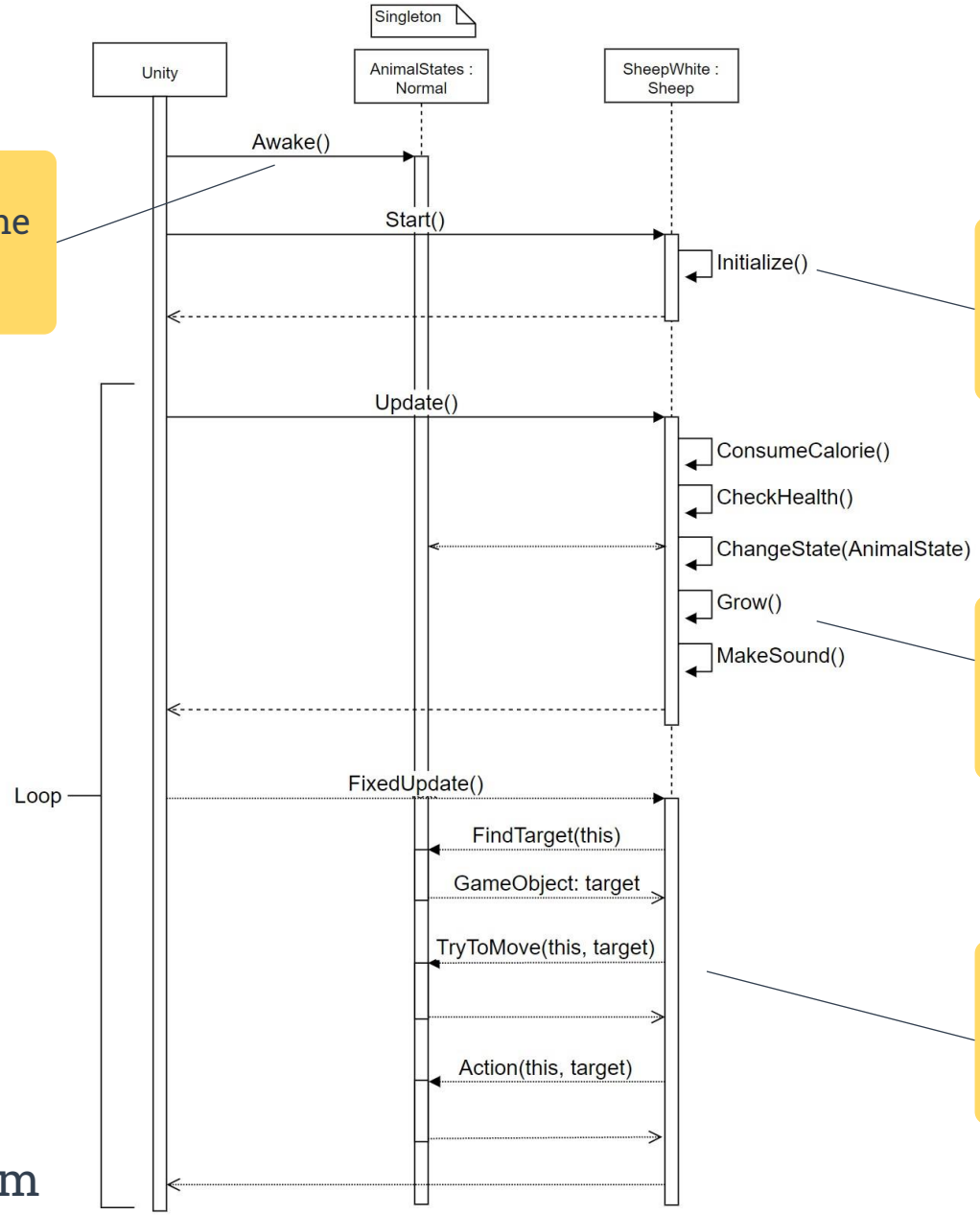
Next step: States (Normal, Hunger)



Game Design

Main sequence of your game.

Instantiate Normal (or, Hunger) class on Awake() by attaching the script component to AnimalStates GameObject.



Initialize Sheep's health, calorie, age and size etc.

Update the sheep's parameters and state per frame.

Conduct actions with the current state.
Methods: FindTarget(), TryToMove and Action().

Sequence Diagram

Game Development

LivingEntities: States - Normal.

1. Edit Normal script according to this image. →
   - Add static field for Normal.
   - Add Awake() function.

The static modifier is used when you want to limit the number of instances to only one.

Variables or methods with the static modifier are allocated fixed memory in the application. It is good to save memory.

In this case, more than two states of “Normal” instance is not needed, so the static modifier is added.
This pattern is called as “Singleton”.

Awake() method is called before Start().
In this case, only one instance will be created as initialize.

DontDestroyOnLoad() makes the target gameobject unbreakable during play the game.

2. Edit Sheep script to use this “instance” field.

```
this.state = Normal.instance;
```

```
ChangeState(Normal.instance);
```

Normal.cs

```
namespace AnimalStates
{
    public class Normal : MonoBehaviour, IAnimalState
    {
        // Singleton
        public static Normal instance { get; private set; }

        private void Awake()
        {
            if (instance == null)
            {
                instance = this;
                DontDestroyOnLoad(this.gameObject);
            }
            else
            {
                Destroy(this.gameObject);
            }
        }

        public void Act
        {
            throw new
        }

        public GameObj
        {
            throw new
        }

        public void Mo
        {
            throw new
        }

        public void TryToMove(Animal mover, GameObject target)
        {
            throw new System.NotImplementedException();
        }
    }
}
```

You can access the static variable or method without instantiate.

e.g.
- Normal.instance
- Vector3.one
- Time.deltatime
- Mathf.Max(num1, num2)

Game Development

LivingEntities: States - Normal.

1. Edit Normal script according to this image. ⟶
   - Edit FindTarget(), TryToMove(), Move() and Action() .
   - Add namespace: using System.Threading.Tasks;

TryToMove() is similar to the TryToAttack() of CactusController script in class 4.

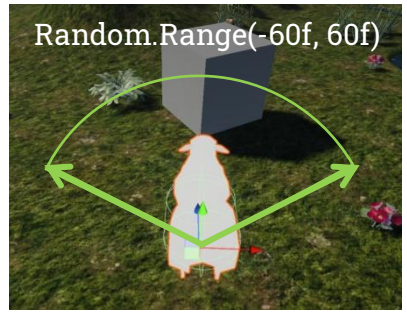
In this case, the sheep checks straight and diagonally downward with Ray. The check function will be implemented later.

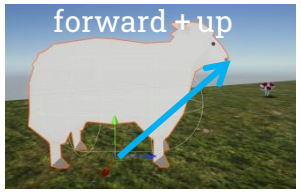
When the Ray touch something, the sheep changes its direction. When nothing there are, they call async Move() function.

- Random.Range(val1, val2)
  returns random value between val1 and val2.
- Rotate(axis, degree)
  rotates the gameobject by the value degree on the axis.
- Rotate(x_degree, y_degree, z_degree)
  rotates the gameobject by the degree on each of the axis.

Normal.cs

```
public GameObject FindTarget(Animal mover)
{
    // Normal state doesn't look for anything.
    return null;
}

public void TryToMove(Animal mover, GameObject target)
{
    if (mover.isMovable)
    {
        bool isStraightRayTouchSomething = false; //CheckStraight(mover);
        bool isDiagnallyDownwardRayTouchWater = false; //CheckDiagnallyDownward(mover);

        if (isStraightRayTouchSomething || isDiagnallyDownwardRayTouchWater)
        {
            mover.transform.Rotate(Vector3.up, Random.Range(-60.0f, 60.0f));
        }
        else if (!isStraightRayTouchSomething && !isDiagnallyDownwardRayTouchWater)
        {
            mover.isMovable = false;
            if (mover.momUid == null)
            {
                Move(mover, target);
            }
            else
            {
                // TODO: Follow the mother
                Move(mover, target);
            }
        }
    }
}

public async void Move(Animal mover, GameObject target)
{
    Rigidbody rb = mover.GetComponent<Rigidbody>();
    rb.transform.Rotate(-10.0f, 0.0f, 0.0f);
    rb.velocity = rb.transform.forward + rb.transform.up;
    await Task.Delay(1500);
    mover.isMovable = true;
}

public void Action(Animal mover, GameObject target)
{
    Debug.Log(mover.uid + ": Normal action");
}
```



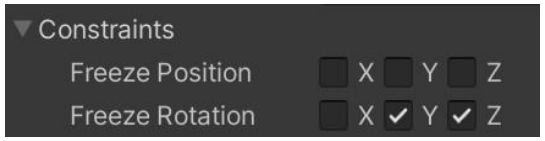


Game Development

LivingEntities: States - Normal.

1. Edit Sheep script according to this image. ⟶
   - Call FindTarget(), TryToMove() and Action() .
2. Edit Rigitbody of your Sheep.
   - Select your Sheep and find the Rigidbody component
   - Check constrains -> Freeze Rotation : Y and Z

   (To avoid tumbling down)

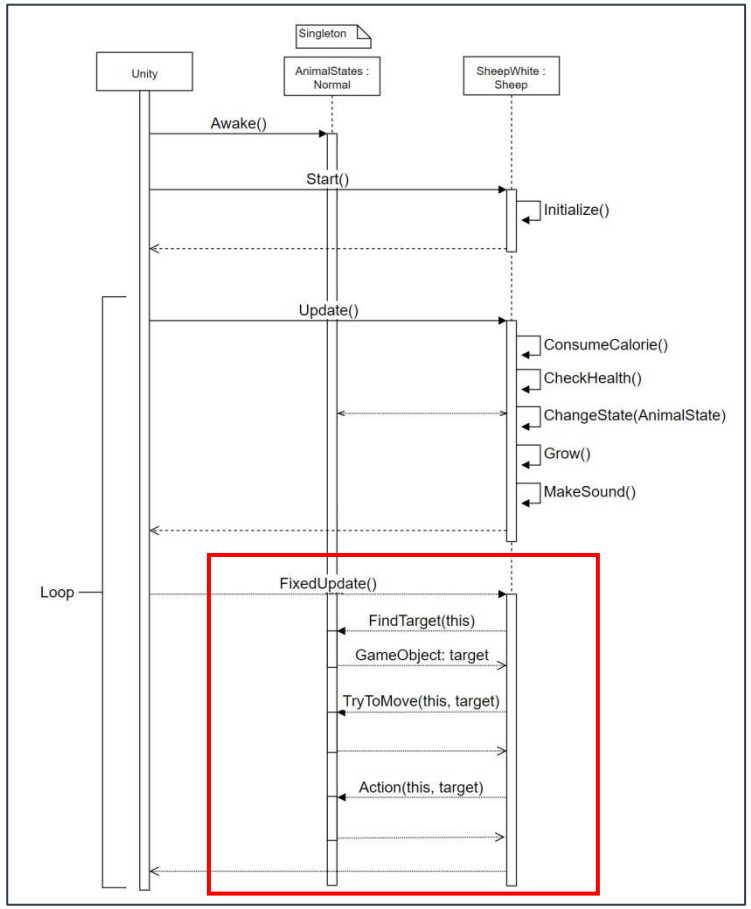


3. Create Empty object to attach State related scripts.
   - Create -> CreateEmpty as “AnimalStates”
   - Attach the Normal script into the AnimalStates.
4. Check current implement by playing your game.
   - The Sheep moves every 1.5 seconds.

(Hard-coded “1500” value is not smart so it will be fixed later.)

Sheep.cs

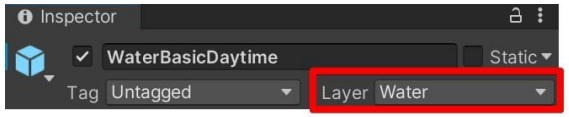
```
private void FixedUpdate()
{
    target = state.FindTarget(this);
    state.TryToMove(this, target);
    state.Action(this, target);
}
```



Game Development

LivingEntities: States - Normal.

1. Edit Normal script according to this image.
   - Add public variables.
   - Use CheckStraight() and CheckDiagnallyDownward().
   - Add above two methods.
2. Check water GameObjects belongs “Water” layer.



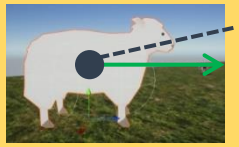
Normal.cs

```
public class Normal : MonoBehaviour, IAnimalState
{
    // Singleton
    public static Normal instance { get; private set; }

    public float rayScale = 1.0f;
    public RaycastHit hit;

    private void Awake()
    {
        if (instance == null)
        {
```

```
public void TryToMove(Animal mover, GameObject target)
{
    if (mover.isMovable)
    {
        bool isStraightRayTouchSomething = CheckStraight(mover);
        bool isDiagnallyDownwardRayTouchWater = CheckDiagnallyDownward(mover);

        if (isStraightRayTouchSomething || isDiagnallyDownwardRayTouchWater)
        {
            mover.transform.Rotate(Vector3.up, Random.Range(-60.0f, 60.0f));
        }
```

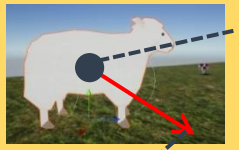
```
private bool CheckStraight(Animal mover)
{
    Ray straightRay = new Ray(mover.GetComponent<Renderer>().bounds.center, mover.transform.forward);
    Debug.DrawRay(straightRay.origin, straightRay.direction * mover.transform.localScale.z * rayScale, Color.green, 2);
    return Physics.Raycast(straightRay, out hit, mover.transform.localScale.z * rayScale);
}

private bool CheckDiagnallyDownward(Animal mover)
{
    Ray diagnallyDownwardRay = new Ray(mover.GetComponent<Renderer>().bounds.center, mover.transform.forward - mover.transform.up / 2);
    Debug.DrawRay(diagnallyDownwardRay.origin, diagnallyDownwardRay.direction * mover.transform.localScale.z * rayScale, Color.red, 2);
    int waterLayerMask = LayerMask.GetMask("Water"/*Layers.Name.Water.ToString()*/);
    return Physics.Raycast(diagnallyDownwardRay, out hit, mover.transform.localScale.z * rayScale, waterLayerMask);
}
```

- CheckStraight():



Renderer().bounds.center

- CheckDiagnallyDownward():



Renderer().bounds.center

mover.transform.forward - mover.tranform.up / 2

```
public class Layers : MonoBehaviour
{
    public enum Name
    {
        Water = 4,
        UI = 5,
        Animals = 6,
        Plants = 7,
        Ground = 8,
    }
}
```

* You should create Layers class to avoid hard-coding: “Water”. Try do this when you have free time.

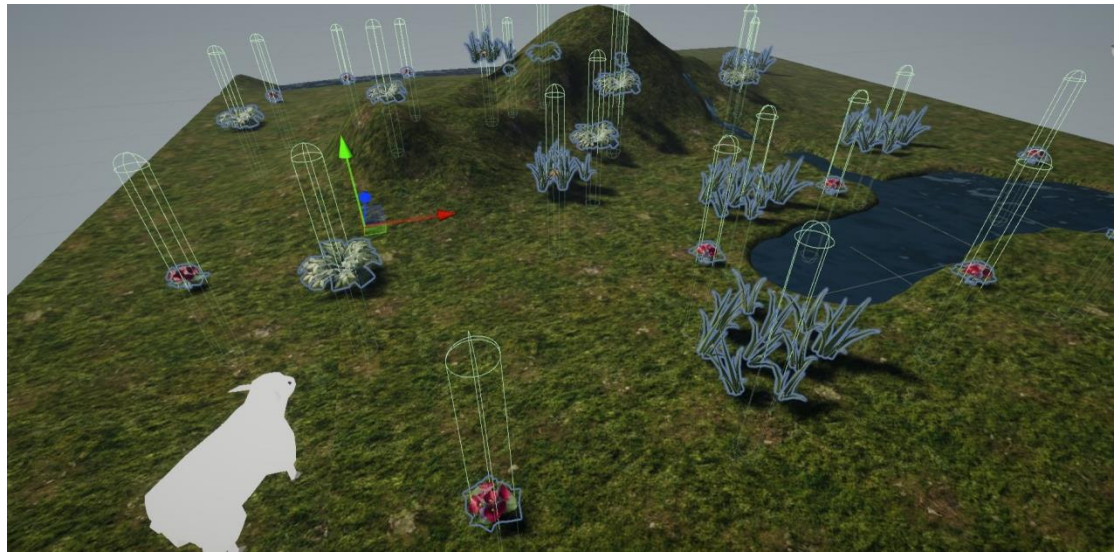
Game Development

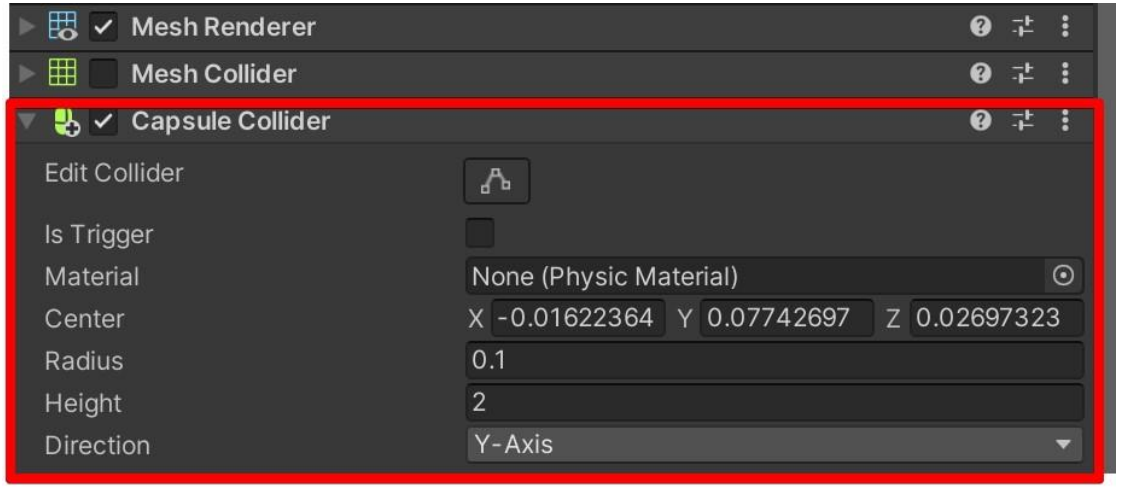
LivingEntities: States - Normal.

1. Edit colliders of your plants.
   - Select all of your plants from Hierarchy.
   - Add Capsule Collider and adjust it.
   - Uncheck “Mesh Collider”.

2. Check current implement by playing your game.
   - The sheep moves every 1.5 seconds.
   - When the ray hit something,
     the sheep will turn to avoid it and move.

3. In some case, the sheep may stumble or stack.
   It’s OK, it will be fixed later.



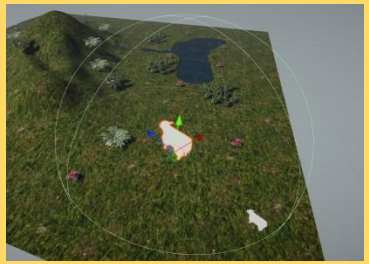


Game Development

LivingEntities: States - Hunger.

1. Copy all of Normal script.
2. Create Hunger script in States/Animal folder.
3. Open Hunger script and paste.
4. Edit the Hunger script according to this image.
5. Create and add “Plants” layers to your plants.
6. Create CalcUtils script in Utils folder.

- Physics. OverlapSphere(Vector3 position, float radius, int layerMask):



Search target layer’s object in designated range.
In the future, it use the sheep’s sense value by gene.

- forasch (Collider hit in colliders) … :
Check distance of each found objects.
If more closer one found, update “nearestObj”.
Then return it.

Hunger.cs (1/2 page. Continued on next page.)

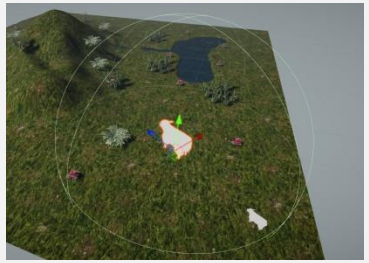
```
using System.Collections;
using System.Collections.Generic;
using System.Threading.Tasks;
using UnityEngine;
using AnimalStates;

namespace AnimalStates
{
    public class Hunger : MonoBehaviour, IAnimalState
    {
        // Singleton
        public static Hunger instance { get; private set; }

        public float rayScale = 1.0f;
        public RaycastHit hit;
        public Vector3 targetPos;

        private void Awake()
        {
            if (instance == null)
            {
                instance = this;
                DontDestroyOnLoad(this.gameObject);
            }
            else
            {
                Destroy(this.gameObject);
            }
        }

        public GameObject FindTarget(Animal mover)
        {
            // You can also use onCllision() method
            int plantLayer = LayerMask.GetMask("Plants"/*Layers.Name.Plants.ToString()*/);
            Collider[] colliders = Physics.OverlapSphere(mover.transform.position, 5f/* * mover.gene.sense*/, plantLayer);
            GameObject nearestObj = null;
            Vector3 nearestObjPos = Vector3.one * 999;
            foreach (Collider hit in colliders)
            {
                if ((hit.gameObject.transform.position - mover.transform.position).sqrMagnitude < nearestObjPos.sqrMagnitude)
                {
                    nearestObjPos = hit.transform.position;
                    nearestObj = hit.gameObject;
                }
            }
            return nearestObj;
        }
```

Game Development

LivingEntities: States - Hunger.

1. Copy all of Normal script.
2. Create Hunger script in States/Animal folder.
3. Open Hunger script and paste.
4. Edit the Hunger script according to this image.
5. Create and add “Plants” layers to your plants.
6. Create CalcUtils script in Utils folder.

- Physics. OverlapSphere(Vector3 position, float radius, int layerMask):



Search target layer’s object in designated range.
In the future, it use the sheep’s sense value by gene.

- forasch (Collider hit in colliders) … :
Check distance of each found objects.
If more closer one found, update “nearestObj”.
Then return it.

Hunger.cs (2/2 page.)

```
public async void Move(Animal mover, GameObject target)
{
    Rigidbody rb = mover.GetComponent<Rigidbody>();
    if (target == null)
    {
        rb.transform.Rotate(-10.0f, 0.0f, 0.0f);
        rb.velocity = rb.transform.forward + rb.transform.up;
        await Task.Delay(1500);
        mover.isMovable = true;
    }
    else if ((target.transform.position - mover.transform.position).sqrMagnitude < mover.transform.localScale.sqrMagnitude)
    {
        CalcUtils.LookTarget(mover, target);
        mover.ChangeState(Eating.instance);
        // You have to change isMovable = true after Eating
    }
    else
    {
        CalcUtils.LookTarget(mover, target);
        rb.velocity = rb.transform.forward + rb.transform.up;
        await Task.Delay(1500);
        mover.isMovable = true;
    }
}

public void Action(Animal mover, GameObject target)
{
    Debug.Log(mover.uid + ": Hunger action");
}
```

No editing required
- TryToMove()
- CheckStraight()
- CheckDiagnallyDownward()

Utils/CalcUtils.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public static class CalcUtils
{
    public static void LookTarget(Animal mover, GameObject target)
    {
        Vector3 direction = target.transform.position - mover.transform.position;
        direction.y = 0f;
        mover.transform.rotation = Quaternion.LookRotation(direction, Vector3.up);
    }

}
```

Game Development

LivingEntities: States - Hunger.

1. Edit Sheep script to test.
   - Remove comment out on ChangeState()
2. Edit Hunger script to test.
   - Comment out on ChangeState()
   - Add Debug.log()
3. Attach Hunger script as component on AnimalState GameObject.
4. Check current implement by playing your game.
   - The sheep moves every 1.5 seconds.
   - When the sheep become hungry, search plants.
   - When find plants, the sheep looks nearest one and move toward to it.

(Your sheep may stumble. It's OK, it will be fixed later.)

Sheep.cs

```
void Update()
{
    ConsumeCalorie();

    // Should be contain as CheckHealth()
    if (this.health <= this.stats.MIN_HEALTH)
    {
        Die();
    }

    if (this.calorie < this.stats.BASE_CALORIE)
    {
        ChangeState(Hunger.instance);
    }
    else
    {
        ChangeState(Normal.instance);
```

Hunger.cs

```
public async void Move(Animal mover, GameObject target)
{
    Rigidbody rb = mover.GetComponent<Rigidbody>();
    if (target == null)
    {
        rb.transform.Rotate(-10.0f, 0.0f, 0.0f);
        rb.velocity = rb.transform.forward + rb.transform.up;
        await Task.Delay(1500);
        mover.isMovable = true;
    }
    else if ((target.transform.position - mover.transform.position).sqrMagnitude <
    {
        CalcUtils.LookTarget(mover, target);
        //mover.ChangeState(Eating.instance);
        Debug.Log("Eating state");
        // You have to change isMovable = true after Eating
    }
    else
    {
```

After Checking, remove comment out.

```
CalcUtils.LookTarget(mover, target);
mover.ChangeState(Eating.instance);
```

Game Development

LivingEntities: States - Eating.

1. Create and edit Eating script in States/Animal folder. Try to reading and coding it.

Eating.cs (1/2)

```
using System.Collections;
using System.Collections.Generic;
using System.Threading.Tasks;
using UnityEngine;
using AnimalStates;

namespace AnimalStates
{
    public class Eating : MonoBehaviour, IAnimalState
    {
        // Singleton
        public static Eating instance { get; private set; }

        public float eatAmount = 3000;

        private void Awake()
        {
            if (instance == null)
            {
                instance = this;
                DontDestroyOnLoad(this.gameObject);
            }
            else
            {
                Destroy(this.gameObject);
            }
        }

        public GameObject FindTarget(Animal mover)
        {
            return Hunger.instance.FindTarget(mover);
        }

        public void TryToMove(Animal mover, GameObject target)
        {
            if (mover.isMovable)
            {
                mover.isMovable = false;
                Move(mover, target);
            }
        }

        public async void Move(Animal mover, GameObject target)
        {
            Rigidbody rb = mover.GetComponent<Rigidbody>();

            if (target == null)
            {
                mover.isMovable = true;
            }
            else if ((target.transform.position - mover.transform.position).sqrMagnitude < mover.transform.localScale.sqrMagnitude)
            {
                rb.transform.Rotate(-10.0f, 0.0f, 0.0f);
                await Task.Delay(1500);
                mover.isMovable = true;
            }
            else
            {
                mover.isMovable = true;
            }
        }

        public void Action(Animal mover, GameObject target)
        {
            foreach (Species.Type diet in mover.stats.DIET)
            {
                if (target.CompareTag(diet.ToString()))
                {
                    Eat(mover, target, target.tag);
                }
            }
        }

        private void Eat(Animal mover, GameObject target, string tag)
        {
            if (target.GetComponent<Flower>().calorie <= target.GetComponent<Flower>().stats.MIN_CALORIE)
            {
                Destroy(target);
                mover.ChangeState(Normal.instance);
                mover.isMovable = true;
            }
            else
            {
                // It is not good because "Flower" is hard-coding
                if (tag == Species.Type.Flower.ToString())
                {
                    target.GetComponent<Flower>().calorie =
                        Mathf.Max(target.GetComponent<Flower>().calorie - eatAmount * Time.deltaTime, target.GetComponent<Flower>().stats.MIN_CALORIE);
                    mover.calorie = Mathf.Min(mover.calorie + eatAmount * Time.deltaTime, mover.stats.MAX_CALORIE);
                }
            }
        }

    }
}
```

Eating.cs (2/2)

Game Development

LivingEntities: States - Eating.

1. Create and edit Eating script in States/Animal folder. Try to reading and coding it.

Eating.cs (1/2)

```
using System.Collections;
using System.Collections.Generic;
using System.Threading.Tasks;
using UnityEngine;
using AnimalStates;

namespace AnimalStates
{
    public class Eating : MonoBehaviour, IAnimalState
    {
        // Singleton
        public static Eating instance { get; private set; }

        public float eatAmount = 3000;

        private void Awake()
        {
            if (instance == null)
            {
                instance = this;
                DontDestroyOnLoad(this.gameObject);
            }
            else
            {
                Destroy(this.gameObject);
            }
        }

        public GameObject FindTarget(Animal mover)
        {
            return Hunger.instance.FindTarget(mover);
        }

        public void TryToMove(Animal mover, GameObject target)
        {
            if (mover.isMovable)
            {
                mover.isMovable = false;
                Move(mover, target);
            }
        }
```

Template of Singleton.

Use same function of Hunger class.

Don't forget attach Eating script as component to AnimalStates GameObject.

Eating.cs (2/2)

```
        public async void Move(Animal mover, GameObject target)
        {
            Rigidbody rb = mover.GetComponent<Rigidbody>();

            if (target == null)
            {
                mover.isMovable = true;
            }
            else if ((target.transform.position - mover.transform.position).sqrMagnitude < mover.transform.localScale.sqrMagnitude)
            {
                rb.transform.Rotate(-10.0f, 0.0f, 0.0f);
                await Task.Delay(1500);
                mover.isMovable = true;
            }
            else
            {
                mover.isMovable = true;
            }
        }

        public void Action(Animal mover, GameObject target)
        {
            foreach (Species.Type diet in mover.stats.DIET)
            {
                if (target.CompareTag(diet.ToString()))
                {
                    Eat(mover, target, target.tag);
                }
            }
        }

        private void Eat(Animal mover, GameObject target, string tag)
        {
            if (target.GetComponent<Flower>().calorie <= target.GetComponent<Flower>().stats.MIN_CALORIE)
            {
                Destroy(target);
                mover.ChangeState(Normal.instance);
                mover.isMovable = true;
            }
            else
            {
                // It is not good because "Flower" is hard-coding
                if (tag == Species.Type.Flower.ToString())
                {
                    target.GetComponent<Flower>().calorie =
                        Mathf.Max(target.GetComponent<Flower>().calorie - eatAmount * Time.deltaTime, target.GetComponent<Flower>().stats.MIN_CALORIE);
                    mover.calorie = Mathf.Min(mover.calorie + eatAmount * Time.deltaTime, mover.stats.MAX_CALORIE);
                }
            }
        }
    }
}
```

Make eating motion by Rotate().

Check diet type from stats file.

You need to create and attach appropriate tags on your plants.

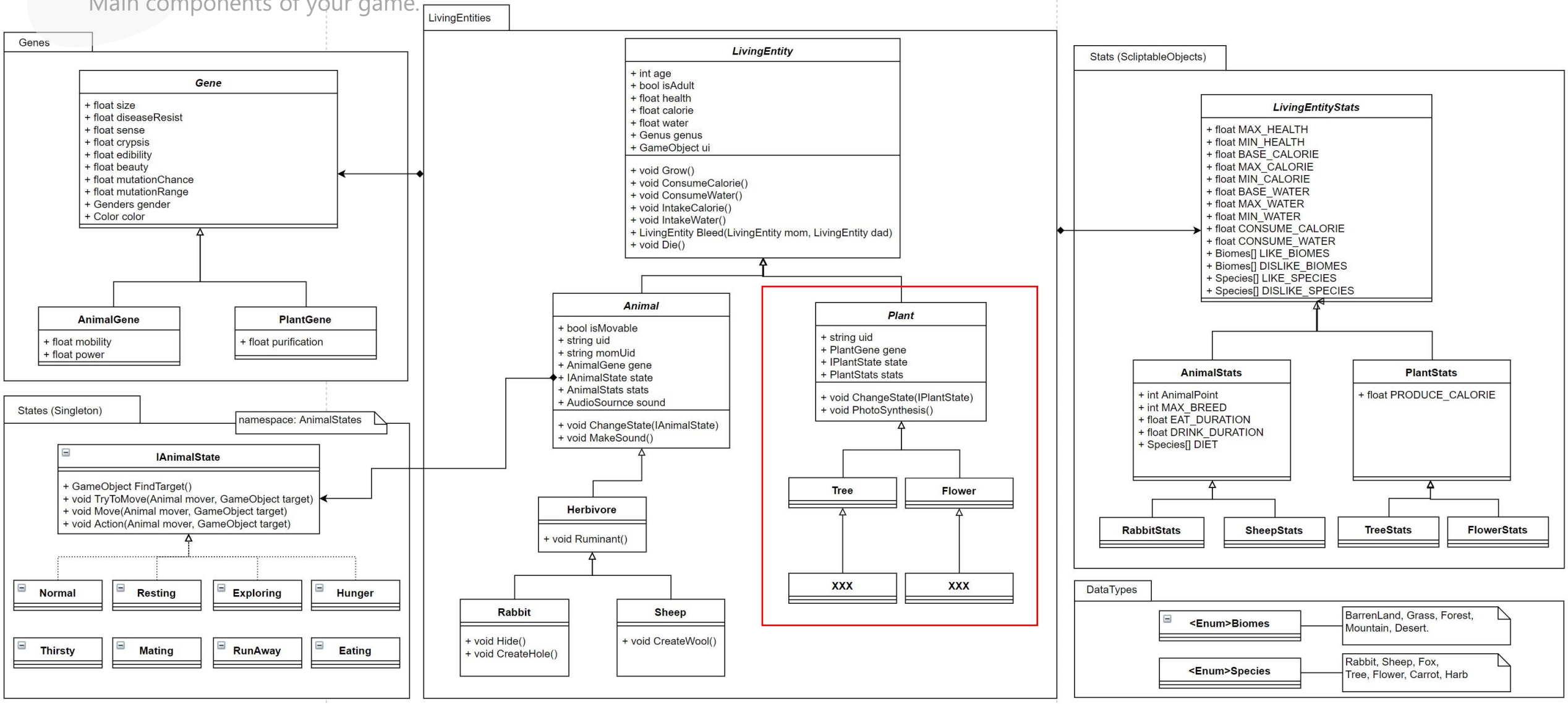
When finish eating.

The calorie move from Flower to the Animal.

Game Design

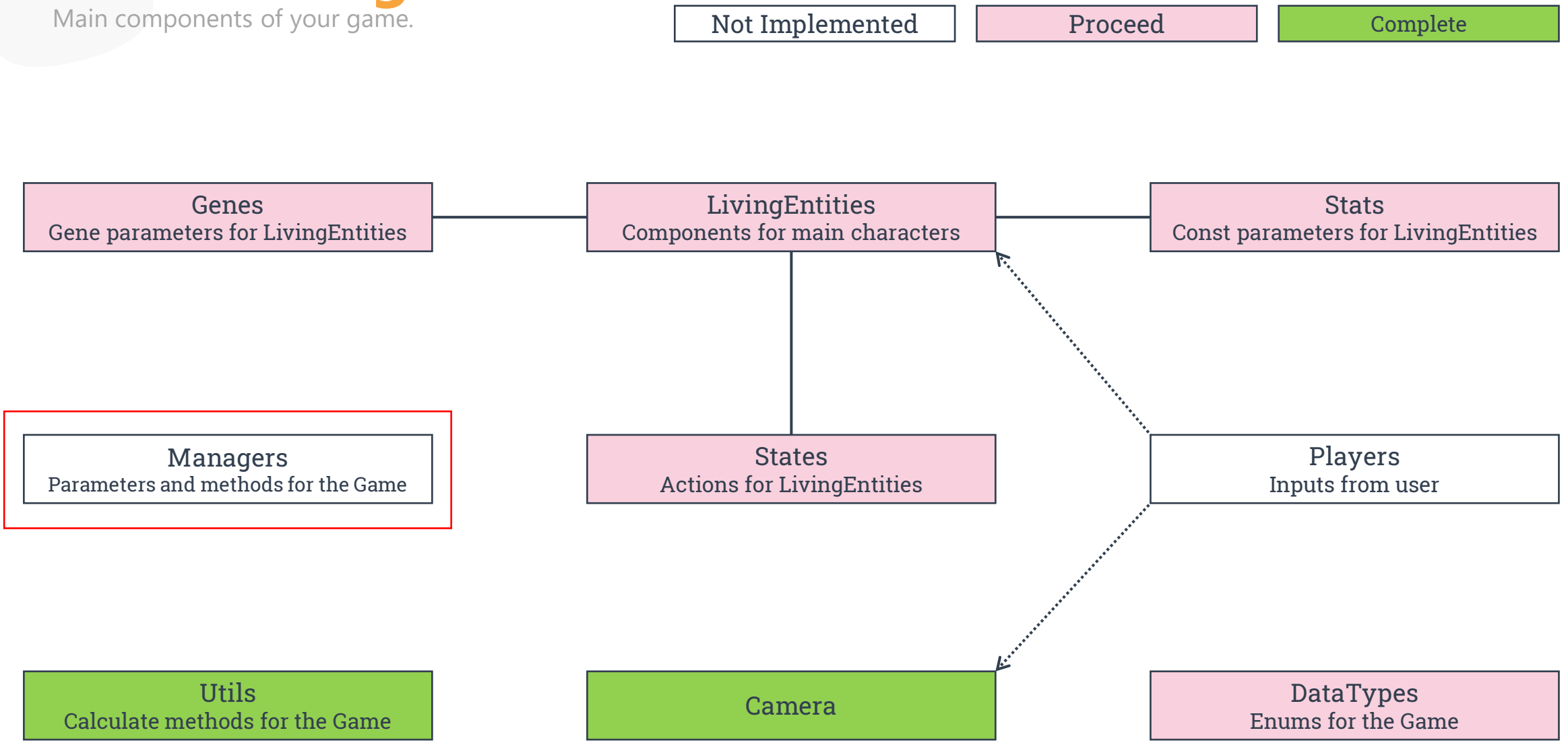
Main components of your game.

Next step: Plants (Flower) and Managers



Game Design

Main components of your game.



Note: SOLID Principles

Knowledge for sophisticated programming.

Your design of programming will be good by following these principles.
However, as it is difficult for beginner, I explain it briefly.

Principle	Simple Explain
Single responsibility principle	A class should have only one role.
Open / Closed principle	You should use abstract or interface for easy maintenance.
Liskov substitution principle	Inherited classes must obey the rules of their parent classes.
Interface segregation principle	The function of interfaces must be kept to a minimum.
Dependency inversion principle	Dependencies should be organized using interfaces.

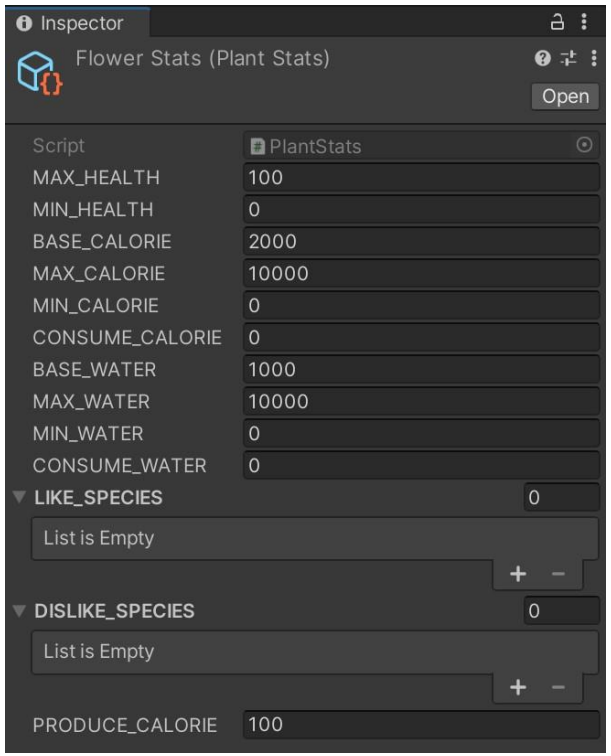
Homework(1/3)

Implement Base of Plants.

1. Create IPlantState, Plant and Flower scripts.
2. Attach Flower script as component to your Flower GameObjects.
3. Create FlowerStats in Resources folder

   (Create -> MyScriptable -> Create PlantStats)

   and set the parameters like bellow.



States/Plant/IPlantState.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace PlantStates
{
    public interface IPlantState
    {
        public GameObject FindTarget();
        public void Action();
    }
}
```

LivingEntities/Plant.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using PlantStates;

public abstract class Plant : LivingEntity
{
    public string uid = System.Guid.NewGuid().ToString();
    public PlantGene gene;
    public IPlantState state;
    public PlantStats stats;
}
```

LivingEntities/Flower.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using PlantStates;

public class Flower : Plant
{
    public override void ConsumeCalorie()
    {
        throw new System.NotImplementedException();
    }

    public override void ConsumeWater()
    {
        throw new System.NotImplementedException();
    }

    public override void Grow()
    {
        throw new System.NotImplementedException();
    }

    public override void InTakeCalorie()
    {
        throw new System.NotImplementedException();
    }

    public override void InTakeWater()
    {
        throw new System.NotImplementedException();
    }

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

Homework(2/3)

Implement Base of Plants.

1. Create States/Plant/Normal script.
2. Create PlantStates GameObject.
3. Attach the Normal scrips as component to PlantStates.
4. Edit Flower script.

States/Plant/Normal.cs

```cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using PlantStates;

namespace PlantStates
{
    public class Normal : MonoBehaviour, IPlantState
    {
        // Singleton
        public static Normal instance { get; private set; }

        public void Action()
        {
            throw new System.NotImplementedException();
        }

        public GameObject FindTarget()
        {
            throw new System.NotImplementedException();
        }

        private void Awake()
        {
            if (instance == null)
            {
                instance = this;
                DontDestroyOnLoad(this.gameObject);
            }
            else
            {
                Destroy(this.gameObject);
            }
        }
    }
}
```

LivingEntities/Flower.cs

```cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using PlantStates;

public class Flower : Plant
{
    public Rigidbody rb = null;
    public GameObject target = null;
    public float minScale = 0.1f;
    public float maxScale = 1.0f;
    public float repeatSpan = 1f;
    public float currentTime = 0f;
    public float growRate = 0.1f;

    public override void ConsumeCalorie() // Nothing Change
    public override void ConsumeWater() // Nothing Change

    public override void Grow()
    {
        float nextScale = Mathf.Min(this.transform.localScale.x + growRate, maxScale);
        this.transform.localScale = Vector3.one * nextScale;
    }

    public override void InTakeCalorie()
    {
        this.calorie = Mathf.Min(this.calorie + this.stats.PRODUCE_CALORIE, this.stats.MAX_CALORIE);
    }

    public override void InTakeWater() // Nothing Change

    void Start()
    {
        Initialize();
    }

    void Update()
    {
        currentTime += Time.deltaTime/* * TimeManager.instance.getCurrentGameSpeedValue()*/;
        if (currentTime > repeatSpan)
        {
            currentTime = 0f;
            Grow();
            InTakeCalorie();
        }
    }

    private void Initialize()
    {
        this.stats = Resources.Load<PlantStats>("FlowerStats");
        this.age = 0;
        this.isAdult = false;
        this.health = stats.MAX_HEALTH;
        this.calorie = stats.BASE_CALORIE;
        this.water = stats.BASE_WATER;
        this.state = Normal.instance;
        this.rb = this.GetComponent<Rigidbody>();
        float initialScale = Random.Range(minScale, maxScale);
        this.transform.localScale = Vector3.one * initialScale;
    }
}
```

Homework(3/3)

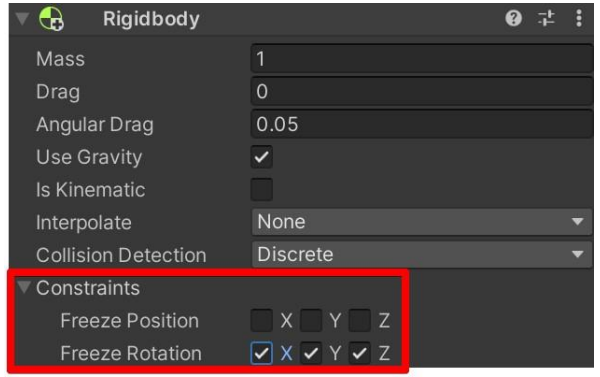
Implement Base of Plants.

1. Edit Sheep script.
2. Create and attach Flower / Grass tags to your GameObjects.
3. Check current implement by playing your game.
   - The sheep moves every 1.5 seconds.
   - When the sheep become hungry, search plants.
   - When find plants, the sheep looks nearest one and move toward to it.
   - Plants and Sheep will grow every certain interval.
   - When the sheep reaches to the target, the sheep eat it.
   - During eating, the plant's calorie will be decreased.
     On the other hand, the sheep takes the calorie.
     (It can be checked in Inspector view)
   - When finish eating, the plant will be die. Then, the sheep starts moving again.

※ If you are worried about sheep tumbling,
you can temporarily change the rigidbody settings as follows.
(Please uncheck again when you finished the testing)

Sheep.cs

```
void Update()
{
    ConsumeCalorie();

    // Should be contain as CheckHealth()
    if (this.health <= this.stats.MIN_HEALTH)
    {
        Die();
    }

    if (this.state.GetType() != Eating.instance.GetType())
    {
        if (this.calorie < this.stats.BASE_CALORIE)
        {
            ChangeState(Hunger.instance);
        }
        else
        {
            ChangeState(Normal.instance);
        }
    }

    currentTime += Time.deltaTime;
    if (currentTime > growSpan)
    {
```



2.Basic programming with Unity

Class Plan

1. Introduction
2. Hello world in Unity
3. Basic programming
4. Object oriented programming
5. Interface programming
6. **Programming design 1**
7. **Programming design 2**
8. **Improvement and testing 1**
9. **Improvement and testing 2**
10. Summary

8
Improvement and testing 1

8. Improvement and testing 1

Class summary

1. Explain improvement.
2. Explain testing.
3. Explain exception handling.
4. Explain regular expressions.

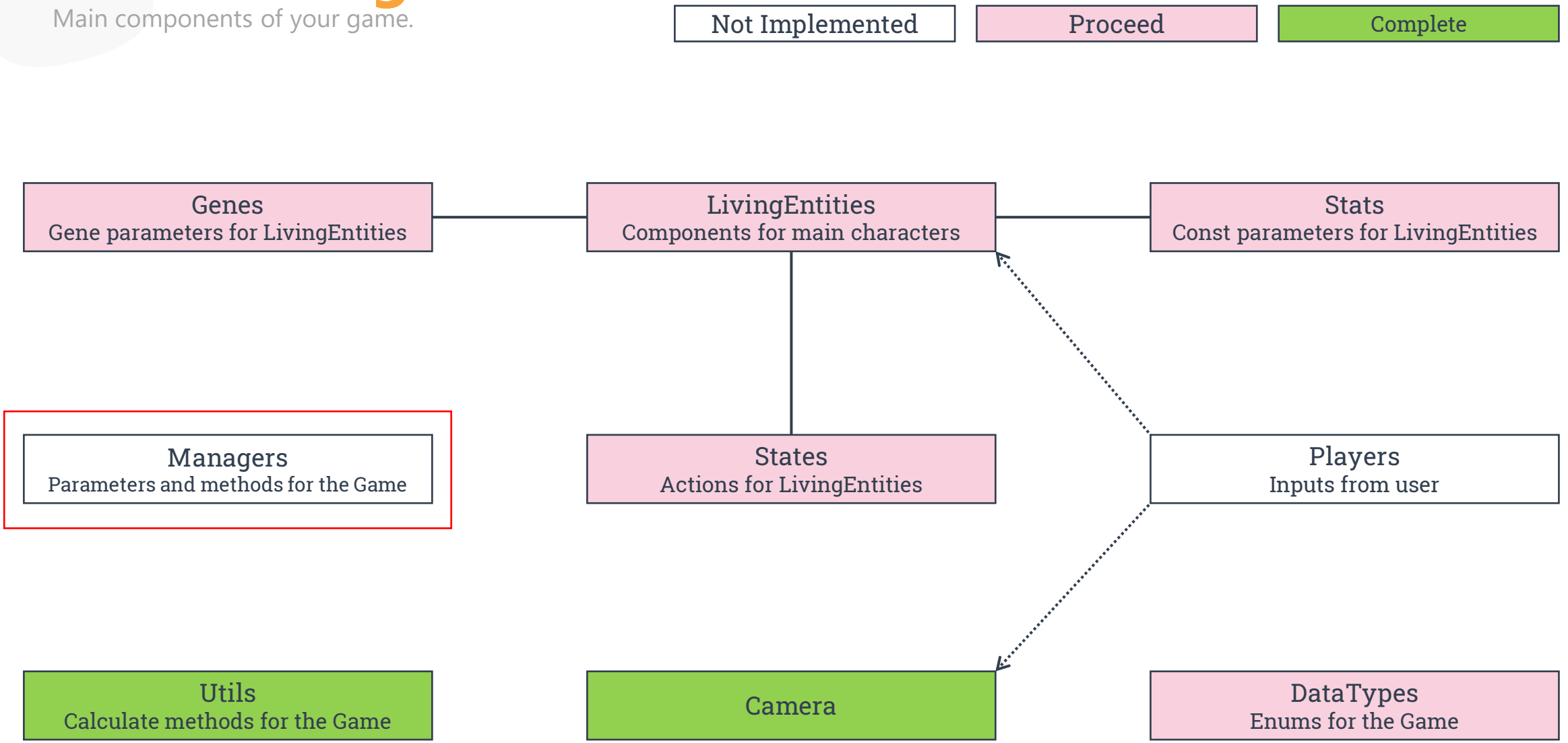
These are just tips so I'll explain them next week and maybe this week I'll support those who are behind.

Coding Rules: Examples

Target	Major Rules	Examples
Class name	a. UpperCamel	a. SampleClass
Class variables	a. LowerCamel b. Snake_case c. Underscore	a. sampleVariable b. sample_variable c. _sampleVariable (Only for private variables)
Bool variables	a. Is_bool	a. is_run
Methods	a. UpperCamel	a. SampleMethod()
Constants	a. All-Caps b. UpperCamel	a. MAX_NUMBER b. MaxNum
Scene name	a. UpperCamel	a. TitleScene
Object name	a. UpperCamel	a. CactusNails
If Statement	a. if (target < condition)	a. if (num < 10)
Comments	You don't' need to write all of contents. Good code & Good name explains the role or function itself.	
Scope	Try to minimize the range.	
Hardcoding	Don't use it. You should use constants.	
Core mind	Simple and Dry. Functions should be under 30〜50 lines.	

Game Design

Main components of your game.



Managers

It is used to manage parameters throughout your game.
So they often use singleton pattern.

Class Name	Simple Explain
TimeManager	Manage time and game speed.
InputManager	Manage player's inputs.
GeneManager	Manage gene related factors.
UIManager	Manage UI.

Template of Managers.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Manager : MonoBehaviour
{
    // Singleton
    public static Manager instance { get; private set; }

    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(this.gameObject);
        }
        else
        {
            Destroy(this.gameObject);
        }
    }
}
```

Game Development

Managers 1: TimeManager.

1. Create TimeManager script in Managers folder.
2. Edit TimeManager script according to this image.
3. Search your code with “1500” and fix all of it.
   - Use ctrl+F (Win) or ⌘+F (Mac)

Entire solution.

```
public async void Move(Animal mover, GameObject target)
{
    Rigidbody rb = mover.GetComponent<Rigidbody>();
    rb.transform.Rotate(-10.0f, 0.0f, 0.0f);
    rb.velocity = rb.transform.forward + rb.transform.up;
    await Task.Delay(1500);
    mover.isMovable = true;
}
```

await Task.Delay(TimeManager.instance.getCurrentGameSpeedIntervalA());

4. Create empty game object as Managers in Hierarchy.
5. Attach TimeManager script as component to the Managers.
6. Check current implement by playing your game.

TimeManager.cs

gameSpeeds[0] = 0.5 = Slow,
gameSpeeds[1] = 1.0 = Normal,
gameSpeeds[2] = 2.0 = Fast speed

gameSpeeds[GameSpeedType.SLOW] = 0.5,
gameSpeeds[GameSpeedType.NORMAL] = 1.0,
gameSpeeds[GameSpeedType.FAST] = 2.0

Array for move intervals.
(To avoid hard-coded “1500”).
* SLOW = 3000, NORMAL = 1500, FAST = 750.

Utility function for debug or co-developers.

The value of Time.timeScale controls your game speed.

Initialize as Normal game speed.

```
public class TimeManager : MonoBehaviour
{
    // Singleton
    public static TimeManager instance { get; private set; }

    public float[] gameSpeeds = { 0.5f, 1.0f, 2.0f };
    public int currentGameSpeed = 1;
    public enum GameSpeedType
    {
        SLOW = 0,
        NORMAL = 1,
        FAST = 2,
    }
    public int[] intervalA = { 3000, 1500, 750 };

    public float getCurrentGameSpeedValue()
    {
        return gameSpeeds[currentGameSpeed];
    }

    public string getCurrentGameSpeedName()
    {
        switch (currentGameSpeed)
        {
            case (int)GameSpeedType.SLOW:
                return GameSpeedType.SLOW.ToString();
            case (int)GameSpeedType.NORMAL:
                return GameSpeedType.NORMAL.ToString();
            case (int)GameSpeedType.FAST:
                return GameSpeedType.FAST.ToString();
        }
        return "[Warning] Something Wrong.";
    }

    public void changeGameSpeed(GameSpeedType type)
    {
        this.currentGameSpeed = (int)type;
        Time.timeScale = gameSpeeds[currentGameSpeed];
    }

    public int getCurrentGameSpeedIntervalA()
    {
        return this.intervalA[currentGameSpeed];
    }

    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(this.gameObject);
        }
        else
        {
            Destroy(this.gameObject);
        }
    }

    void Start()
    {
        changeGameSpeed(GameSpeedType.NORMAL);
    }
}
```

Game Development

Managers 2: InputManager.

1. Create InputManager script in Managers folder.
2. Edit InputManager script according to this image.
3. Attach InputManager script as component to the Managers.
4. Check current implement by playing your game.
   - Press 1 , 2 and 3 key to change gamespeed.

InputManager.cs


```
public class InputManager : MonoBehaviour
{
    // Singleton
    public static InputManager instance { get; private set; }

    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(this.gameObject);
        }
        else
        {
            Destroy(this.gameObject);
        }
    }

    void Update()
    {
        if (Input.GetKey(KeyCode.Alpha1))
        {
            TimeManager.instance.changeGameSpeed(TimeManager.GameSpeedType.SLOW);
        }

        if (Input.GetKey(KeyCode.Alpha2))
        {
            TimeManager.instance.changeGameSpeed(TimeManager.GameSpeedType.NORMAL);
        }

        if (Input.GetKey(KeyCode.Alpha3))
        {
            TimeManager.instance.changeGameSpeed(TimeManager.GameSpeedType.FAST);
        }
    }
}
```

Game Development

Managers 3: GeneManager.

1. Create GeneManager script in Managers folder.
2. Edit GeneManager script according to this image.
3. Edit Sheep script according to this image.
4. Attach GeneManager script as component to the Managers.
5. Check current implement by playing your game.
6. (Optional) You can use gene factors now!
   Try to edit Hunger script to use it.

↓

Hunger.cs: FindTarget()

```
public GameObject FindTarget(Animal mover)
{
    // You can also use onCllision() method
    int plantLayer = LayerMask.GetMask("Plants"/*Layers.Name.Plants.ToString()*/);
    Collider[] colliders = Physics.OverlapSphere(mover.transform.position, 5f * mover.gene.sense, plantLayer);
```

GeneManager.cs

```
public class GeneManager : MonoBehaviour
{
    // Singleton
    public static GeneManager instance { get; private set; }

    public AnimalGene AnimalGeneInit()
    {
        return new AnimalGene(RundomGender());
    }

    public AnimalGene AnimalGeneInit(Genders.Type type)
    {
        return new AnimalGene(type);
    }

    public Gene Inherite(Gene momGene, Gene dadGene)
    {
        // TODO
        return null;
    }

    public Genders.Type RundomGender()
    {
        Genders.Type gender;
        float val = Random.Range(0f, 1f);
        if (val <= 0.50)
        {
            gender = Genders.Type.Female;
        }
        else
        {
            gender = Genders.Type.Male;
        }
        return gender;
    }

    private void Awake() ... // Same as template
}
```

[Random gender ver]
Create and return AnimalGene class with the constructor.

[Designated gender ver]

AnimalGene.cs

```
public class AnimalGene : Gene
{
    public float mobility;
    public float power;

    public AnimalGene(Genders.Type gender)
    {
        size = 1.0f;
        diseaseResist = 1.0f;
        sense = 1.0f;
        crypsis = 1.0f;
        edibility = 1.0f;
        beauty = 1.0f;
        mutationChanve = 1.0f;
        mutationRange = 1.0f;
        this.gender = gender;
        color = Color.black;
        mobility = 1.0f;
        power = 1.0f;
    }
}
```

Sheep.cs: Initialize()

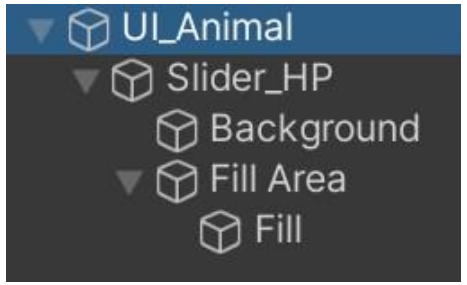
```
this.rb = this.GetComponent<Rigidbody>();
this.stats.DIET = new Species.Type[] { Species.Type.Flower, Species.Type.Grass };
this.gene = GeneManager.instance.AnimalGeneInit();
```

Game Development

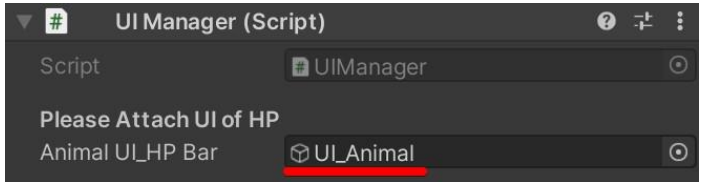
Managers 4: UIManager and UI.

1. Create UIManager script in Managers folder.
2. Edit UIManager script according to this image.
3. Attach InputManager script as component to the Managers.
4. Create base of UI (HP bar).
   - Create Canvas as UI_Animal on your Hierarchy.
   - Create Slider as Slider_HP in UI_Animal.
   - Open Slider_HP and delete Handle Slide Area.
5. Attach the UI_Animal to AnimalUI_HPBar of UIManager.

Your current Hierarchy of the UI.



5. Attached UI_Animal on UIManager

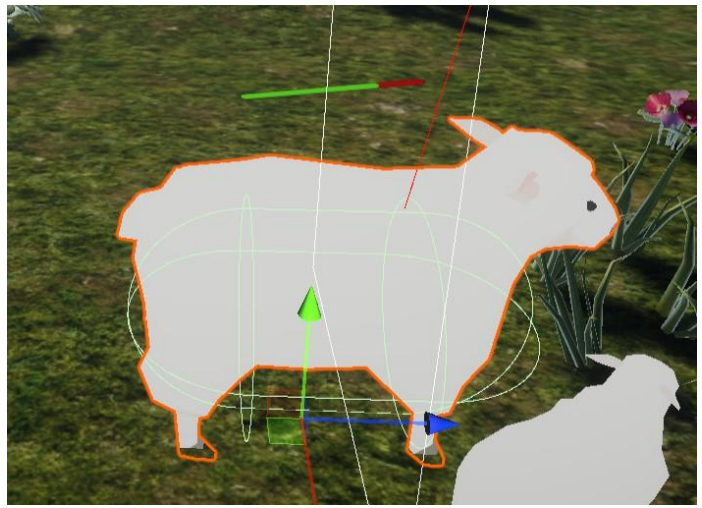


UIManager.cs

```
public class UIManager : MonoBehaviour
{
    // Singleton
    public static UIManager instance { get; private set; }

    [Header("Please Attach UI of HP")]
    public GameObject animalUI_HPBar;

    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(this.gameObject);
        }
        else
        {
            Destroy(this.gameObject);
        }
    }
}
```

Header for inspector.

UI image: Slider_HP

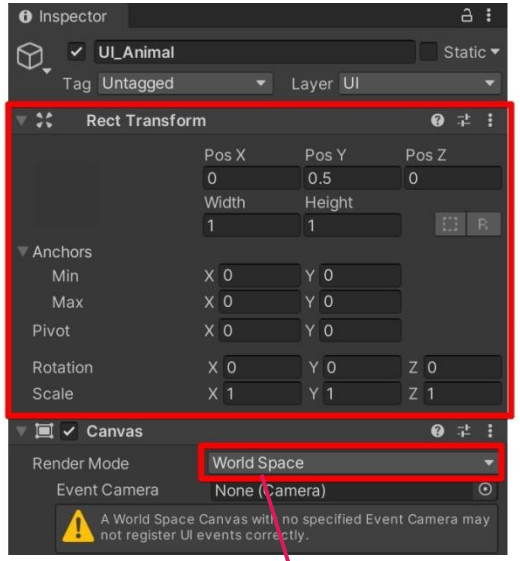


Game Development

Managers 4: UIManager and UI.

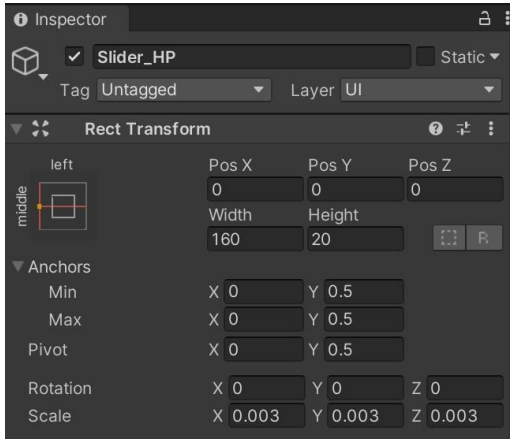
1. Edit the UI_Animal parameters.
   - Select World Space from pull down.
   - Set the parameters according to this image.
2. Edit the Slider_HP parameters.
   - Set the parameters according to this image.
3. Edit the Background parameters.
   - Set Red to Color.
4. Edit the Fill Area parameters.
   - Set the parameters according to this image.
5. Edit the Fill parameters.
   - Set the parameters according to this image.
   - Set Green to Color.

1. UI_Animal

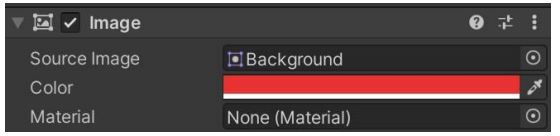


The UI will be rendered as game object: not overlay.

2. Slider_HP

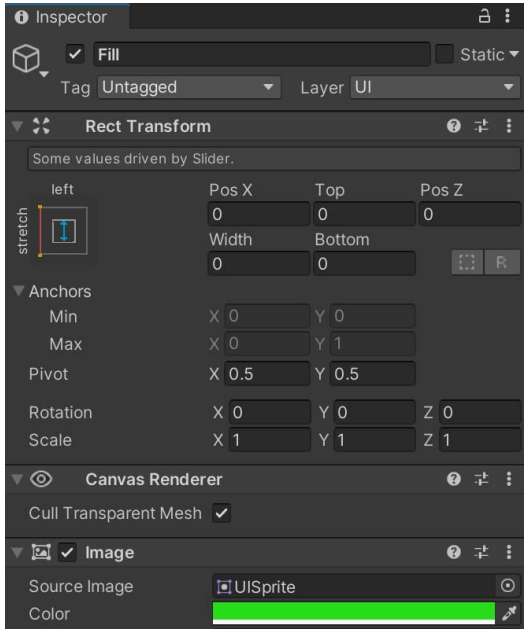


3. UI_Animal



4. Fill Area

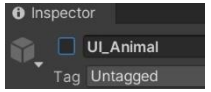
5. Fill



Game Development

Managers 4: UIManager and UI.

1. Create Prefabs folder in Assets.
   - Right click Assets folder -> Create -> Folder.
   - Rename the new folder.
2. Create UI_Animal Prefab.
   - Drag UI_Animal and drop it to Prefabs folder.
3. Deactivate the original UI_Animal.
   - Uncheck the object in the inspector. ⟶



4. Edit LivingEntity script according to this image.
5. Edit Sheep script to add the UI according to this image.
   - Don't forget adding using statement of UnityEngine.UI.

   - Instantiate the UI with UIManager to the sheep's position.
   - Set the Slider info to this.slider.

6. Play your game and check the UI. ⟶

4. LivingEntity.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public abstract class LivingEntity : MonoBehaviour
{
    public int age;
    public bool isAdult;
    public float health;
    public float calorie;
    public float water;
    public GameObject ui;
    public Slider slider;
    // public Genus genus;
```

5. Sheep.cs

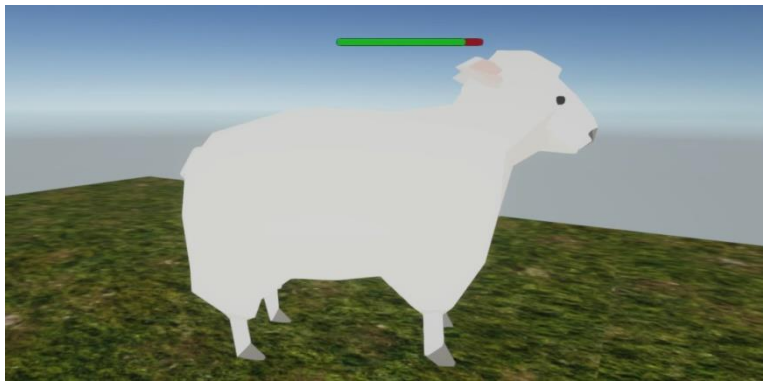
+ using UnityEngine.UI;

```
private void Initialize()
{
    // You have to place stats.asset files into "Resources" folder
    this.stats = Resources.Load<AnimalStats>("SheepStats");
    this.age = 0;
    this.isAdult = false;
    this.health = stats.MAX_HEALTH;
    this.calorie = stats.BASE_CALORIE;
    this.water = stats.BASE_WATER;
    this.isMovable = true;
    this.state = Normal.instance;
    this.sound = null; // Should be set sound file
    this.rb = this.GetComponent<Rigidbody>();
    this.stats.DIET = new Species.Type[] { Species.Type.Flower, Species.Type.Grass };
    this.gene = GeneManager.instance.AnimalGeneInit();
    this.ui = Instantiate(UIManager.instance.animalUI_HPBar, this.transform);
    this.ui.SetActive(true);
    this.slider = this.ui.transform.Find("Slider_HP").GetComponent<Slider>();
    this.slider.minValue = this.stats.MIN_HEALTH;
    this.slider.maxValue = this.stats.MAX_HEALTH;
}
```

Game Development

Managers 4: UIManager and UI.

1. Edit Sheep script to add the UI according to this image.
2. Create RotateToCamera script in UI folder.
3. Edit RotateToCamera script according to this image.
4. Attach the RotateToCamera to UI_Animal as component.
5. Play your game and check the UI.
   - It is shown in front of main camera.
   - It starts as green but it turns red when the sheep hunger.



1. Sheep.cs

```
void Update()
{
    ConsumeCalorie();

    // Should be contain as CheckHealth()
    if (this.health <= this.stats.MIN_HEALTH)
    {
        Die();
    }
    else
    {
        slider.value = this.health;
    }

    if (this.state.GetType() != Eating.instance.GetType())
    {
        if (this.calorie < this.stats.BASE_CALORIE)
        {
            ChangeState(Hunger.instance);
        }
        else
        {
```

3. RotateToCamera.cs

```
public class RotateToCamera : MonoBehaviour
{
    // LateUpdate is called after Update()
    void LateUpdate()
    {
        transform.rotation = Camera.main.transform.rotation;
    }
}
```

Congratulations!
Your World is Started!

Game Development

And Then...?

Time to Improvement

You should implement as your ideas!

1. Improve Visual.
   - Use your own textures.
   - Add new Animals or Plants.
2. Improve World System.
   - Add Day & Night system.
   - Add Clock.
   - Add new Biomes.
3. Improve AI.
   - Add new states.
   - Add ability of children to follow their parents.
   - Prevent stumble / fall of animals.
   - Implement Smarter AI.
4. Improve Functions.
   - Add Bleed & Gene system.
   - Add Predators.
5. Improve Game System.
   - Add Player inputs.
   - Add In-Game Money to purchase Animals / Plants.
6. Improve UI.
   - Improve Camera control system.
   - Add new UI to manage Animals / Plants.
7. Refactoring
   - Improve your code.
   - Improve class design.

...And more!

Time to Improvement

You should implement as your ideas!

Assignment: Improve your game as you like.

1. Document(s) to be submitted should be contains ...
   - Screenshots of your works.
   - Descriptions of your works.
   - Source codes of your works.
   - Images and descriptions you think you need.
   - What you thought when you implemented it.
   - Comments on this class.
2. Where to submit.
   - Announce it on this class.
   - The document should preferably be in PDF format.
   - The document title format is 02_[YourName].pdf
3. Deadline.
   - Announce it on this class.

The assignment is subject to assessment.

The excellent document will be presented in class 10.

Good luck!

Support Time

Self-Study time.

- This is catch-up time for students who got behind.

  Please feel free to ask any questions.

1. Explain improvement.
2. Explain testing.
3. Explain exception handling.
4. Explain regular expressions.

These tips will be explained in next week.

2.Basic programming with Unity

Class Plan

1. Introduction
2. Hello world in Unity
3. Basic programming
4. Object oriented programming
5. Interface programming
6. **Programming design 1**
7. **Programming design 2**
8. **Improvement and testing 1**
9. **Improvement and testing 2**
10. Summary

9
Improvement and testing 2

9. Improvement and testing 2

Class summary

1. Explain improvement.
2. Explain testing.
3. Explain exception handling.
4. Explain regular expressions.

Coding Rules: Examples

Target	Major Rules	Examples
Class name	a. UpperCamel	a. SampleClass
Class variables	a. LowerCamel b. Snake_case c. Underscore	a. sampleVariable b. sample_variable c. _sampleVariable (Only for private variables)
Bool variables	a. Is_bool	a. is_run
Methods	a. UpperCamel	a. SampleMethod()
Constants	a. All-Caps b. UpperCamel	a. MAX_NUMBER b. MaxNum
Scene name	a. UpperCamel	a. TitleScene
Object name	a. UpperCamel	a. CactusNails
If Statement	a. if (target < condition)	a. if (num < 10)
Comments	You don't' need to write all of contents. Good code & Good name explains the role or function itself.	
Scope	Try to minimize the range.	
Hardcoding	Don't use it. You should use constants.	
Core mind	Simple and Dry. Functions should be under 30〜50 lines.	

Testing

To prevent Degradation: bugs occurred by improvement.

Please remember, the check list of testing…

- The sheep moves every 1.5 seconds.
- When the sheep become hungry, search plants.
- When find plants, the sheep looks nearest one
  and move toward to it.
- Plants and Sheep will grow every certain interval.
- When the sheep reaches to the target, the sheep eat it.
- During eating, the plant's calorie will be decreased.
  On the other hand, the sheep takes the calorie.
  (It can be checked in Inspector view)
- When finish eating, the plant will be die.
  Then, the sheep starts moving again.

These Testing memo is not cool.

So you should create testing table like this.

Some developer make this table first because testing table shows specification of your game.

It is called "Test Driven Development": TDD.

Testing Table v1. Testing Date: xx/xx/xxxx

Type	Title	How to Test	Check
Game System	Default game speed: NORMAL	1. Start Game. 2. Check console and find debug message of "Normal: Timescale = 1.0".	✓
	Change game speed: SLOW	1. Start Game. 2. Press "1" key. 3. Check console and find debug message of "Normal: Timescale = 0.5".	✓
	Change game speed: NORMAL	1. Start Game. 2. Press "1" key. 3. Press "2" key. 4. Check console and find debug message of "Normal: Timescale = 1.0".	✓
	Change game speed: FAST	1. Start Game. 2. Press "3" key. 3. Check console and find debug message of "Normal: Timescale = 2.0".	✓
Animals: Sheep	Moves every 1.5 seconds(NORMAL).	1. Start Game. 2. Count 1.5 seconds and check the sheep's move.	✓
	Moves every 3.0 seconds(SLOW).	1. Start Game. 2. Press "1" key. 3. Count 3.0 seconds and check the sheep's move.	✓
	Moves every 0.75 seconds(FAST).	1. Start Game. 2. Press "3" key. 3. Count 0.75 seconds and check the sheep's move.	✓
…	…	…	✓
…	…	…	✓

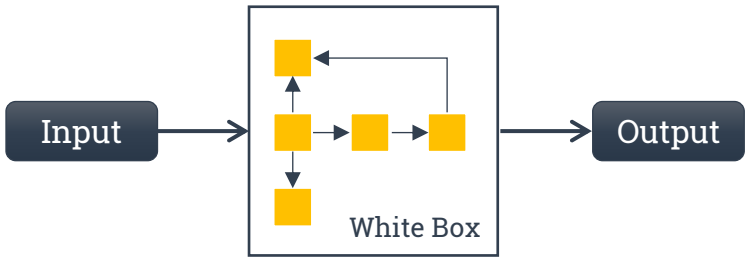
Testing methods

You should check boundary values.

Major testing methods.

Type	Description	When you use this method
Black Box Test	Test target as black box: unknown functions.	Checking output, UI, and Screen info etc. based on spec.
White Box Test	Test target as white box: known functions.	Checking function, values and conditions etc. based on code.
Testing Code	Test target with coding: True or False.	Checking class functions based on code. (Unit Test)

Check as User



Check as Developer

xUnit

nunit

Check by Machine

Exception handling

Clarify what type of error occurred.

Major exception classes.

Type	Description	Example
Exception	Base class of all exceptions.	-
DivideByZeroException	Performed division by zero.	int num = 100 / 0;

IndexOutOfRangeException	Out-of-range indexing.	int[] array = new int[3]; // You can access with [0], [1], and [2]
		int num = array[3];

OverflowException	Overflow of digits occurred.	int num = 2147483647; // int range: -2,147,483,648 ~ 2,147,483,647
		num++;

NullReferenceException NullPointerException	Accessing an empty object. Forgetting attach game object.	Sheep sheep = null; Sheep.MakeSound();

The exceptions can be handled with try ~ catch statement for system-level errors.

In product level, developers often define original exception class to clarify what error happened.

Sample: Sheep.cs

```
void Start()
{
    try
    {
        Initialize();
    }
    catch (Exception e)
    {
        Debug.LogError("Error occurred on Initialize(). Message: " + e.Message + ",StackTrace: " + e.StackTrace);
        // The stack trace stores information about which script caused the error and on which line.
    }
}
```

When exception is thrower in Initialize(), variable “e” will catch it.
* You need “using System;” statement in Unity.

Regular Expressions

To prevent unexpected input: when you want to use input fields.

Major expressions.

Patterns	Description	Example
.	Matches any single letter.	“1234” can be expressed as “....”.
*	0 or more repetitions.	“1234” can be expressed as “.*”.
{n}	n repetitions.	“1234” can be expressed as “.{4}”.
[0-9]	Matches any number between 0 and 9.	“11223344” can be expressed as “1{2}2{2}3{2}4{2}”, “[0-9]{8}”.
[a-z] [A-Z]	Matches any letter between a and z. Matches any letter between A and Z.	“Richard” can be expressed as “[A-Z][a-z]{6}” “Richard1234” can be expressed as “[A-Z][a-z]*[1-4]*”

Regex.IsMatch(string target, @”expressions”) returns true or false depend on match or not.

If you want to extract only the part of the target that matches your expressions, you can use Regex.Match() method. Please Google it if you need it.

Sample

```
// + using System.Text.RegularExpressions;
void Start()
{
    string targetName = "Sheep";
    if (Regex.IsMatch(targetName, @"[A-Z][a-z]*"))
    {
        Debug.Log("Matched!");
    }
    else
    {
        Debug.Log("Not matched!");
    }
}
```

Time to Improvement

You should implement as your ideas!

1. Improve Visual.
   - Use your own textures.
   - Add new Animals or Plants.
2. Improve World System.
   - Add Day & Night system.
   - Add Clock.
   - Add new Biomes.
3. Improve AI.
   - Add new states.
   - Add ability of children to follow their parents.
   - Prevent stumble / fall of animals.
   - Implement Smarter AI.
4. Improve Functions.
   - Add Bleed & Gene system.
   - Add Predators.
5. Improve Game System.
   - Add Player inputs.
   - Add In-Game Money to purchase Animals / Plants.
6. Improve UI.
   - Improve Camera control system.
   - Add new UI to manage Animals / Plants.
7. Refactoring
   - Improve your code.
   - Improve class design.

...And more!

Time to Improvement

You should implement as your ideas!

Assignment: Improve your game as you like.

1. Document(s) to be submitted should be contains ...
   - Screenshots of your works.
   - Descriptions of your works.
   - Source codes of your works.
   - Images and descriptions you think you need.
   - What you thought when you implemented it.
   - Comments on this class.
2. Where to submit.
   - Announce it on this class.
   - The document should preferably be in PDF format.
   - The document title format is 02_[YourName].pdf
3. Deadline.
   - Announce it on this class.

The assignment is subject to assessment.

The excellent document will be presented in class 10.

Good luck!

Support Time

Self-Study time.

- This is catch-up time for students who got behind.

  Please feel free to ask any questions.

2.Basic programming with Unity

Class Plan

1. Introduction
2. Hello world in Unity
3. Basic programming
4. Object oriented programming
5. Interface programming
6. Programming design 1
7. Programming design 2
8. Improvement and testing 1
9. Improvement and testing 2
10. **Summary**

10
Summary

10. Summary

Class summary

1. Review assignments.
2. Tips.
3. Summary.

Coding Rules: Examples

Target	Major Rules	Examples
Class name	a. UpperCamel	a. SampleClass
Class variables	a. LowerCamel b. Snake_case c. Underscore	a. sampleVariable b. sample_variable c. _sampleVariable (Only for private variables)
Bool variables	a. Is_bool	a. is_run
Methods	a. UpperCamel	a. SampleMethod()
Constants	a. All-Caps b. UpperCamel	a. MAX_NUMBER b. MaxNum
Scene name	a. UpperCamel	a. TitleScene
Object name	a. UpperCamel	a. CactusNails
If Statement	a. if (target < condition)	a. if (num < 10)
Comments	You don't' need to write all of contents. Good code & Good name explains the role or function itself.	
Scope	Try to minimize the range.	
Hardcoding	Don't use it. You should use constants.	
Core mind	Simple and Dry. Functions should be under 30~50 lines.	

Review assignments

For your sustainable development.

Tips

For your sustainable development.

- Git supports manage your codes as version management system.
  - Easy to revert your code to an older version.
  - Easy to clone (copy) the code to your device.
  - Share your editing history with multiple people.
- GitHub provides online Git services.
  - You can create repository to manage your codes.
  - Support code review and marge it.
- Sourcetree provides graphical user interface of Git.
  - Easy to use Git without commands.
  - Visualize adding / editing / deleting files.
  - Conduct commit / push / clone etc. by buttons.