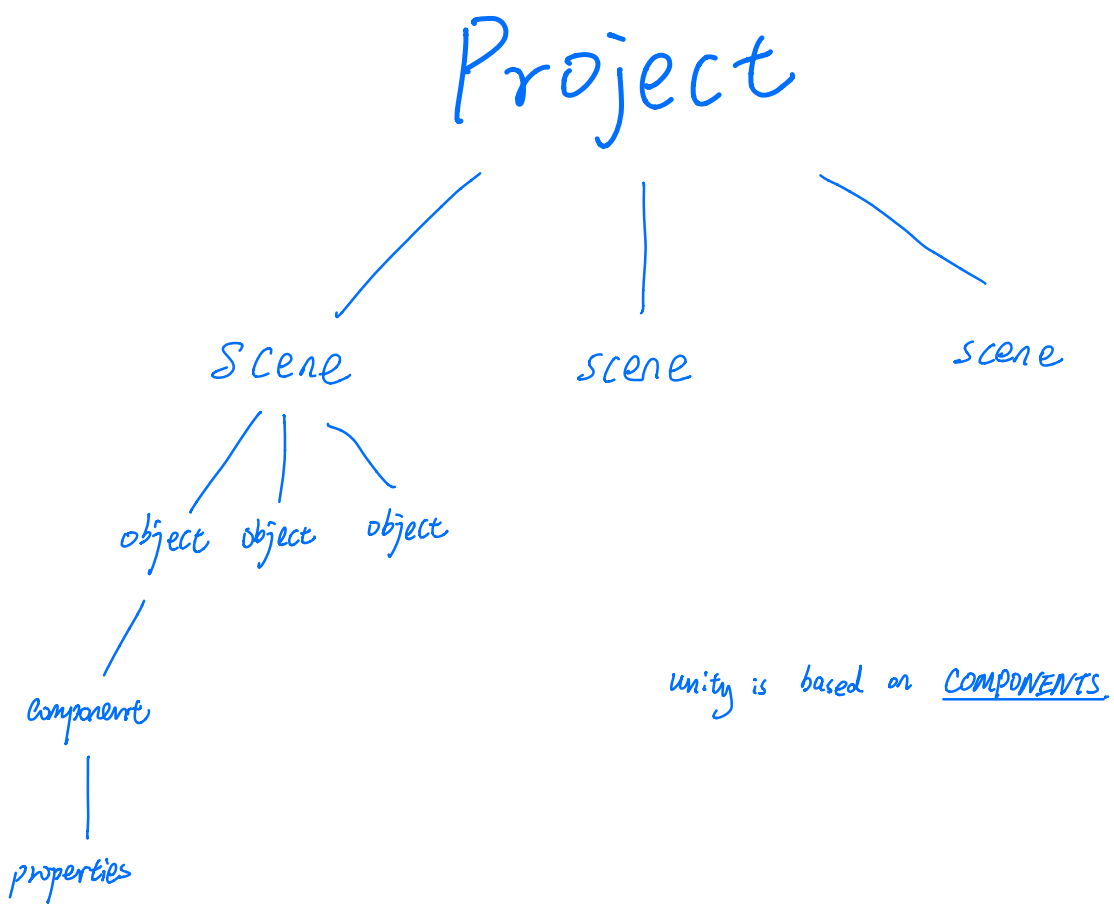


unity is based on COMPONENTS

---

project window: manage all project files

inspector window: show object's components and properties

Short-cuts:

scene:

move angle:

- Right Click
- Alt + Left Click

move:

- Middle button
- "Hand" on tool bar

Game object:

every game object has different grids.

Less grid = good performance

Center vs Pivot: (switch middle point)

center: all objects middle point =>

pivot: select object =>

short-cut: 'z'

Global vs Local: (switch coordinate system)

global: global coord.

local: object's coord.

short-cut: 'x'

parent & child:

1. one game object can ONLY have 1 parent.
2. one game object can have INFINITE children.
3. child DO NOT MOVE with parent.
4. parent MOVES with children.

Components:

- Mesh Filter: determine object's shape
- Mesh Renderer: render object's shape (material: texture)
- If you can't find component, write a script (c#)

Change object's color:

1. create new material
2. change its color
3. drag material to object

Use picture to change object texture:

1. create new material
2. drag picture next to 'Albedo'

   or

   click circle next to 'Albedo'
3. drag material to object

Prefab: save a game object as a file. (预设体)

- Move game object from Hierarchy to Project window. (file type: .prefab)
- The game object will turn to blue.
- Manage multiple game object. (批量管理)
- change prefab will also change game object.
- change individual game object will lose connection to prefab.

Prefab function:

- Select: highlight prefab inside project → assets folder.
- revert: revert game object's component & properties back to prefab.
- apply: apply game object's component & properties to prefab.

> 相反的

Package: copy/paste several files from one project to another.

Exporting package:

1. Right click on project → assets window
2. export package
3. select what you want to export (file type: .unitypackage)

Importing package:

Drag package into project window

or

1. Right click on project → assets window
2. import package

Unity Terrain:

Terrain: a game object

Opacity: the speed of rise / lower.

setheight: hold down shift can set height.

flatten: brush all terrain same height.

paint texture: when first select texture, the texture will cover whole terrain.
but not the second one.

paint tree:
1. edit tree → add tree (prefab)

terrain height: highest point in a terrain

*terrain files CAN MOVE around, but NOT DELETE!!!

Rigid body: It provides physics.

*add in component

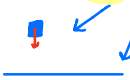
is kinematic: if it is checked, it would not move in mid-air.

Collider: when two objects collide with each other, it is actually 'the collider'.

Edit box collider: edit 'collider' on object

box collider material: only apply to physics material (create new in project window)

physics material friction combine: combine both surface objects (cube and plane)



C#

- Console.WriteLine(); Print new Line
- Console.Write(); Print Line
- const value must have a default value and can not be changed.
- change type:

1. From low to high
   byte → int

```
sbyte num = 1;
int num1 = num;
```

2. From high to low
   int → byte

```
int num = 1000;
sbyte num1 = (sbyte)num;
```

*May lose accuracy / data

- char type: each char has different numeric values.
  'A' = 65, 'a' = 98. int ⟷ char
- increment: a++; ++a;
  先值,后加. 先加,后值.

```
int a = 10;
Console.WriteLine(a++);   10
Console.WriteLine(++a);   12
Console.WriteLine(--a);   11
```

- region: can collapse code.

```
#region
  :
#endregion
```

- VS short-cut:
  comment: ctrl + k, C
  uncomment: ctrl + k, u
- +=, -= work in C#. a+=10; <=> a=a+10;
- switch is better than if when more items are presented. go to case
- $: 解析

```
int meter = 1;
string a = $"run{meter}";
print(a);
```

(run 1)

- @：原义：You can write " " in a string.

String a = @"C:\Program Files(x86)\";

*Caution: can not use " " in a string.

- 替换 Console.WriteLine( );

meter = 1;

second = 2;

Console.WriteLine("he runs {0} meters. {1} seconds.", meter, second);

he runs 1 meters, 2 seconds.

C# input / output:

Console.Read(): ask user for first char input.

- Only take the first char.
- Only can assign to "int".

int a = Console.Read();

Console.Readkey(): ask user for anything.

- ConsolekeyInfo info = Console.Readkey();

Console.ReadLine(): ask user for all input. string a = Console.ReadLine();

- Convert String to other types:

Integer:

string input = Console.ReadLine();
Convert.ToInt32(input);

number represents bit.

16 → 2 byte → short
32 → 4 byte → int
64 → 8 byte → long

String:

Convert.ToString(input); OR str = 3.14 + " ";

- 对齐：{column * line, -2}
  ↑ 强制结果2位，负号是左对齐。
  没有号是右对齐。

- foreach vs for：

  foreach：good for displaying items, CAN NOT modify them.

  ```
  foreach (int item in array) { Console.Write(item + ", "); }
  ```

  for：CAN modify them, but no restriction.

- params：在调用方法的时候，可以将数组中的元素直接写到参数列表中。

  ```
  Test(1, 2, 3, 4, 5);

  static void Test(params int[] array){...}
  ```

- ref：传递地址，参数都需要 ref。

  ```
  Swap(ref x, ref y);
  static void Swap(ref int x, ref int y){...}
  ```

- out：传递地址，参数都需要 out。

  ```
  Swap(out x, out y);
  static void Swap(out int x, out int y){...}
  ```

- differences：

  - out 需要赋值参数值，在方法结束前
  - ref 参数默认是有值的，指向实参的值。

  VS

  out 参数默认是没值的，不能直接使用。

- 2D Array in C#:

```
int[,] array = new int[3,4];
array.Length = 12.
```

Assign integer in array: `array[1,2] = 1;`

- Constructor: class will provide you with a default constructor
  - How to run detailed constructor with less-detailed constructor

```
class person {
    public person(){}
    public person(int a): this(){ }
    public person(int a, int b): this(int a){ }
}
```

```
public person(int age, string name){
    this.name = name;
    this.age = age;
}
```

- Static constructor:
  - CAN NOT modify with public or private
  - CAN NOT have 参数
  - No relationship with regular constructor
  - It will run when you first use this class, only run ONCE

- Object 属性访问器

```
public int age {
    set {
        if (value <=150) age = value;
    }
    get {
        return age;
    }
}
```

- readonly: read only.

readonly VS const

- can change value in constructor
- 不需要赋值

- can not change value
- 需要赋值

- Inheritance:
  - class 子类 : 父类 { }
  - CAN NOT Inherit Constructor
  - 子类实体化之前，要先实体化父类
    ① 父类要有 default constructor
    ② 子类的 constructor extend 父类的参数 constructor

父类: Animal
子类: Dog

父类 → public Animal (int a) { }
子类 → public Dog : base(1) { }

- permission reserved key words
  - private : ONLY access in class
  - protected : access in class AND child class inside or outside the project
  - internal : WHOLE PROJECT PERMISSION
  - protected internal : access in class AND child class outside the project
  - public : EVERYTHING (public > internal)
  - * Default class is internal type.

- Polymorphism : 多态
  - Upcast : convert sub-class to super-class
    - 100% success rate `Animal a = new Dog();`
    - After upcasting, it CAN NOT use original method.
  - Downcast : convert super-class to sub-class * only use in reference data type
    - It may fail... `Dog d = new Animal() as Dog;`
      - ↳ if it fails, you will get Null.
    - Before performing downcast, use 'is' to determine type.

- 方法隐藏 : super and sub class have the same method, we can hide "super".

```
class Animal {
    public void eat() { }
}
```
⟺
```
class Dog : Animal {
    public new void eat() { }
}
```
* 'new' keyword

- Override : 重写 : if sub-class wants to over-write super-class method

  虚函数 : virtual

  - super class must have virtual keywords `public virtual void Animal`
    and sub class must have override keywords `public override void Dog`

- 枚举 enum: list all values in terms of numbers

```
public enum Season {
    spring
    summer
    autumn
    winter
}
```

<=> `Season s = Season.spring;`

- You can change data type by using ":"

```
public enum Season : long { }
```

- Struct: 结构体: similar to class

struct vs class

- object in struct memory is in stack.
- struct is value type, NOT reference type
- CAN NOT write default constructor
- ALWAYS HAVE a hidden default constructor
- CAN NOT have destructor '~'
- MORE efficient than class, because it builds on stack.
- Struct only has 1 super-class called object.

- Static: 静态类: only has 1 super-class called object.
  - 设计工具类, 用 static.

- Sealed: 密封: 不能被继续重写
  - Modify class + method
  - CAN NOT USE inheritance

- ONLY USE when 'Override'

- Name Space: virtual combination

```
namespace Space0{}
namespace Space1{}
```

- 防止类名重复
- 管理类
- using system → 用system的命名空间. 只能在最上方写

System.Console.WriteLine(); 省去System.

- 命名空间名称可重复
- 如果多个命名空间的名称是重复的，那么他们是一个命名空间
- 无限嵌套

- Operator Override: 运算符重载

```
public static Point operator + (Point a, Point b){ }
```

override operator

override 'add' operator

- MUST BE PUBLIC AND STATIC
- 一元运算符：运算符左右两边只需1个数字. -1, -2, etc...
  二元运算符：运算符左右两边需要2个数字. 1+2 / 1-2 / 1*2 / 1÷2

- Abstract: 抽象类

```
Abstract Hello{}
```

- 不能实例化对象
- 可以被其他类继承

- Abstract method: 抽象方法

- ONLY DECLARATION, NO CODING
  只有声明，没有实现.
- Abstract method CAN ONLY WRITE in Abstract Class.
- If a non-abstract class inherit an abstract class,
  sub-class must override abstract methods in super-class.
- When to use abstract???
  - 用来约束所有子类的行为 (Make rules that all sub-classes have to follow)

- Interface: 接口：一系列规范 (A set of rules)
  - 定义接口： interface USB { }
  - 实现接口： class Mouse : USB { }

- Interface method: 接口命名用大写'I'开头 eg. IUSB, ICPU, IGPU
- 接口的方法，不是abstract方法
  接口的方法不能用访问权限

- 实现接口method：
  - 必须是public method
  - 不能用override
  - 如果abstract class 想实现接口，可以把接口方法实现为abstract method.
  - 接口方法可以实现为虚方法，让子类去重写 (override)
  - 如果一个class即有接口，又有父类，

    class Mouse : USB, Hardware
    父类在前，接口在后

  - 一个类可以有很多接口，多个接口用逗号分隔.

- 多态进阶：
  - 接口的引用可以指向实现类的对象
  - Casting — Upcast：由实现类类型转为接口类型
    Downcast：由接口类型转为实现类型.
    *特点与上面一样.
  - 接口会变成参数
- 接口内可以写 Get/Set

```
int Age { set; get; }
```

有声明，无实现.

  - 实现接口需要定义 set/get.
- 接口可以 inheritance

- Delegate：委托：方法的类型

声明：

```
delegate void TestDelegate();
```

```
TestDelegate a = new TestDelegate(TestMethod);
```

  - 实例化一个委托对象需要一个方法来实例化
  - 此方法的返回值类型和参数列表需要和委托保持一致

```
delegate void TestDelegate();
public static void TestMethod(){}
```

  - 如何使用？

```
TestDelegate a = new TestDelegate(TestMethod);
a();
```

'a' 就是一个方法

* - 什么时候用？

当我们想把方法当作参数传来传去时.

- 泛型：Generic

```
class Person<T> { }
```

在一个类中，可以有不止一种泛型

- 在同一个命名空间中，可以存在2个类名相同的类，但泛型类需要不一样
- 泛类只能作用在当前方法中，不能作用在其他方法中
- T 不知道什么类型，但可以在方法中使用
- 方法中的泛型在调用方法时使用
- 如果泛型被当做参数来使用，可以不给他进行赋值

* · 什么时候用？
  - 当你不知道一个方法的参数时
  - 当你不知道一个方法的 return type 时　　`T GetComponent<T>();`
- 在接口（Interface）中，
  - T 不知道什么类型，但可以在方法中使用
  - 泛型不能被接口继承

- ArrayList（集合）　　`using System.Collections;`
  - Array 定长，ArrayList 可变长
  - ArrayList 快捷的方法操作元素

```
ArrayList list = new ArrayList();
```

  - ArrayList 可添加任何元素
  - ArrayList.Add(); 增加一个新元素
  - ArrayList.AddRange(); 将一个 arraylist 加入到另一个 arraylist 中
  - ArrayList.Insert(X,Y); 在 index X 插入 Y 元素.
  - ArrayList.InsertRange(X,Y); 在 index X 插入 arrayList Y.
  - ArrayList.Count(); 得出这个 arraylist 有多少元素

- ArrayList.Remove(); 移除该数组中第一次找到的元素
- ArrayList.RemoveAt(); 移除该数组中index的位置
- ArrayList.RemoveRange(X, Y); X是从index开始, Y是移除多少位.
- ArrayList[]: 用index改arraylist
- ArrayList.SetRange(X,Y); 从index X开始, 替换arrayList Y.
- 在arraylist里, 不要用foreach来操作arraylist元素!!!
- ArrayList.IndexOf(): 查询第一次出现的index
- ArrayList.LastIndexOf(): 查询最后一次出现的index
- ArrayList.BinarySearch(); 二分查找法查询index
- ArrayList.ToArray(); arrayList转array.
- ArrayList.Sort(); arrayList从小到大排序
- ArrayList.Reverse(); arrayList从大到小排序
- ArrayList.contains(); 判断集合里是否包含此元素
- ArrayList.copyTo(); 将一个集合的元素指定到另一个集合中

一 List<>

- 指定类型的元素
- using System.Collections.Generic;

```
List<string> list = new List<string>();
```

- list.Add(); 添加一个元素
- list.Remove(); 移除第一个元素
- list.RemoveAll(); 移除所有对应元素
- list.Exists(); 判断是否存在的对应元素 *注意, 括号里只能写委托
- list.Find(); 查找第一个对应的元素
- list.FindAll(); 查找所有对应元素

- Stack

- using System.collections;
- First in, Last out

```
Stack s = new Stack();
```

- s.Push(); push element into a stack
- s.Peek(); find the element on TOP of the stack
- s.Pop(); delete the element on TOP of the stack

- Queue

- using System.collections;
- First in, First out

```
Queue q = new Queue();
```

- q.Enqueue(); add element into queue
- q.Peek(); find the element the first of the queue
- q.Dequeue(); delete the element the first of the queue

- Hash Table

- using System.collections;
- key-value-pairs (key, value)

```
Hashtable table = new Hashtable();
```

- table.Add("name", "kenh"); key="name", value="kenh".
- key and value MUST pair
- key must be unique
- table.key (Retrieve key); table.value (Retrieve value)

- table[key] = value; update key-value pair using index
- ICollection keys = table.keys; get all keys
- Use foreach to get all values

```
foreach (object key in keys){
    object value = table[key];
```

- table.Remove(key); delete one key-value-pair

- Dictionary
  - using System.collections.generic;
  - need to specify key & value types

```
Dictionary<string, string> dic = new Dictionary<string, string>();
```

  - Add, Remove are all the same to Hash Table
  - use for each to get all values

```
foreach (KeyValuePair<string, string> pair in dic){
    string key = pair.key;
    string value = pair.value;
```

  - get all keys, similar to Hash Table.

- Regular Expression
  - check if the string matches the format
  - using System.Text.RegularExpressions;

```
Regex r = new Regex("^hello world$");
```

change accordingly

  - r.IsMatch("hello world"); compare current string with Regex 'r' rule.

- Exception handling

```
try {
}
catch (exception type){
}
finally {
}
```

codes that may contain errors

execute when error occurs

always will run, no matter what

- When encounter multiple exception, Parents exception MUST at the end
- Customize exception (write our own)

```
class ScoreException : Exception {}
```
create an exception class

```
ScoreException exception = new ScoreException();
```
initialize it

```
throw exception;
```
throw it!

* if you want to add custom message in custom exception, create a constructor with parameter.
* still need to catch custom error!

- Reflection
  - using System.Reflection;

```
Type t = Type.GetType("Person");
```
(a classname) access the type of a class

  - * if this type exists in certain namespace, then you need to add namespace in it
  - Activator.CreateInstance(t); initial an object of type 't' (return an object)
  - * MUST use public default constructor in that object

- Activator.CreateInstance(t, "parameter"); initial an object with public parameter constructor

- Activator.CreateInstance(t, true); if the second parameter is true, it can use any methods. (public, private, etc.)
- BindingFlags: 访问的方法及权限的描述, Must contain the following:
  1. 访问成员的权限描述 — BindingFlags.Public / BindingFlags.NonPublic
  2. Static or non-static? — BindingFlags.Static / BindingFlags.Instance

- Access variables in class through reflection (variable: int a)
  1. Access public, and private, non-static variables:

```
FieldInfo a = t.GetField("variable name", BindingFlags.NonPublic | BindingFlags.Instance);
a.SetValue(obj, 1);   set value of object 'obj' variable 'a' to 1.
object aa = a.GetValue(obj);   get value of object 'obj' variable 'a'.
```

  2. Access public, and private, static variables:

```
FieldInfo a = t.GetField("V.N", BindingFlags.Static | BindingFlags.Public/NonPublic);
a.SetValue(null, 3);   set value
a.GetValue(null);   get value
```

- Access methods in class through reflection (method: a)
  1. Regular methods

```
MethodInfo method0 = t.GetMethod("a", .....);   *change BindingFlags.
method0.Invoke(null, null);
```

who is invoking method? / parameter List

  2. Overload methods

Specify type of parameter.

```
MethodInfo method1 = t.GetMethod("a", null, new Type[] { typeof(int) }...);
obj result = method1.Invoke(obj, new object[] { .... });   *return value
```

String manipulation

```
string str = new string();   Initialize
```

- `"A".CompareTo("A");` Compare string 'A' to 'A', one by one.
  If the letter is bigger,
- `"AB".Contains("A");` Check if string AB contains A.