Curative Trees Game Dev

w/ UNITY!!!

CS Capstone

November +

December

Learning Notes

C# BASICS

C#	· files end in .cs · 5th most popular language 2019 · part of .NET platform - Microsoft cross platform developer platform · must import w/ using System;
OUTPUT/INPUT	Console.WriteLine("Hi"); string input = Console.ReadLine();
COMMENTS	//single line /* multi line */
VARIABLES	· int, double, char, bool, string datatypes · all variables must be assigned type at declaration
STRING FUNCTIONS	· .ToUpper() .ToLower() .Length · .Substring() $"string {varName}"

CONDITIONALS	if (condition)
	{
	//code
	}
	else if ( )
	{
	//code
	}
	else
	{
	}

← typical C# syntax puts braces on their own lines

switch	switch (varName)
	{
	case "one":
	//code
	break; ← mandatory break keyword
	case "two"
	// code
	break;
	default:
	break;
	}

Ternary operators	String varName = "one" ? (condition) : "two"; ↑ if ↑ else

METHODS	static (returnType) (name) (pType parameter) {
	//code
	}

expression bodied methods	static retType Name () => 1+1; ↑ for 1 line functions ↑ no return necessary

default parameters

```
static int MyFunc(int p1, p2 = 2, p3 = 4) { }
```

mandatory parameters come first

out parameters

- take variables as arguments, sets parameters as values
- actions performed on out parameters persist on actual variable outside method

```
static void MyFunc(out string name, out string color){
    name = "C";
    color = "purple";
}
string name;
string color;
MyFunc(out name, out color);
Console.WriteLine(name, color)   → // C, purple
```

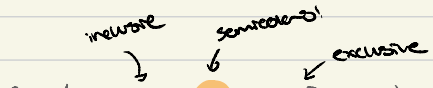
ARRAYS

```
type[] name = { };
type[] name = new type[size]
```

- Arrays are mutable : size can change

LOOPS

for loop



foreach loop

foreach (type iteratorName in arrayName) {}

while loop

while (var != "value") {}

do while loop

- always runs at least once

```
do {
    //code
} while ( condition );
```
← semicolon use

jump statements

- break; continue, return, as usual

Classes

```
class Cat
{
    public string name
    { get; set; }
    public int age
    {
        get { return age; }
        set { name = value; }
    }
    public Cat (string name, int age)
    {
        this.name = name;
        this.age = age;
    }
    static Cat()
    {
        Console.WriteLine("First Cat instance created");
    }
}
Cat cat0 = new Cat("Fido", 1);
```

define getters : setters w/ property

constructors can be overloaded!

static constructors run once, when first instance created

inheritance

```
class Animal
{
    public virtual void Noise()
    {
        Console.WriteLine("noise sound");
    }
}
class Cat : Animal
{
    public override void Noise()
    {
        Console.WriteLine("meow");
    }
}
```

must be abstract or virtual to be overridden

inherit from

override superclass method w/ same name

- public accessible anywhere
- private only class
- protected only class + classes that inherit from it

INTERFACES

- like mandatory blueprints for classes

```
interface IAnimal        ← names start with I
{
    string name { get; }
    int age { get; set; }
    void Noise();
}
public class Cat : IAnimal {}
```

all classes using interface MUST have these properties or methods

↑ inherit like superclass

LISTS

- need any type, size can change

```
using System.Collections.Generic;
List<T> stuff = new List<T>();
```

- .Count — property size of list
- .Contains("") — true/false
- .Remove(value)
- .Clear() — remove all values
- .Add(value) — added to end

UNITY w/ C#

IMPORTS

- at the top of every file

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

using UnityEngine.UI; <- only for working w/ UI elements
```

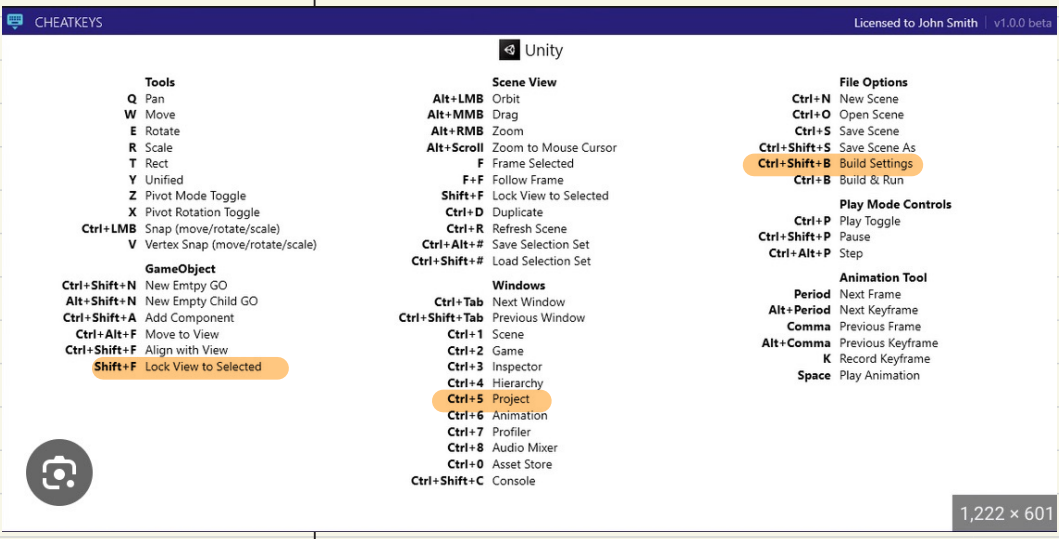
STRUCTURE

- scripts attach to individual objects

```
public class ScriptName : MonoBehaviour
{
    // define global vars here
    void Start()
    {
        // called before 1st frame update
    }
    void Update()
    {
        // called once per frame
    }
    // define other methods below
}
```

Scene Management	using UnityEngine.SceneManagement; ← required import SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex+1); advance to the next scene using build index.
Notewise methods	void OnTriggerEnter2D(Collider2D collision) {} void OnCollisionEnter2D(Collision2D collision) {} Input.GetButtonDown("Space") ← weapons Input.GetAxisRaw("Vertical"/"Horizontal") ← movement in direction
Notewise components	Animator AudioSource LayerMask Rigidbody2D BoxCollider2D SpriteRenderer myComponent = GetComponent<Type>();
serialize field	· make variable from easier while keeping it private [SerializeField] private string varName; · easier for quick adjustments/ testing

Hotkeys : Shortcuts



L rotate tile right

J rotate tile left