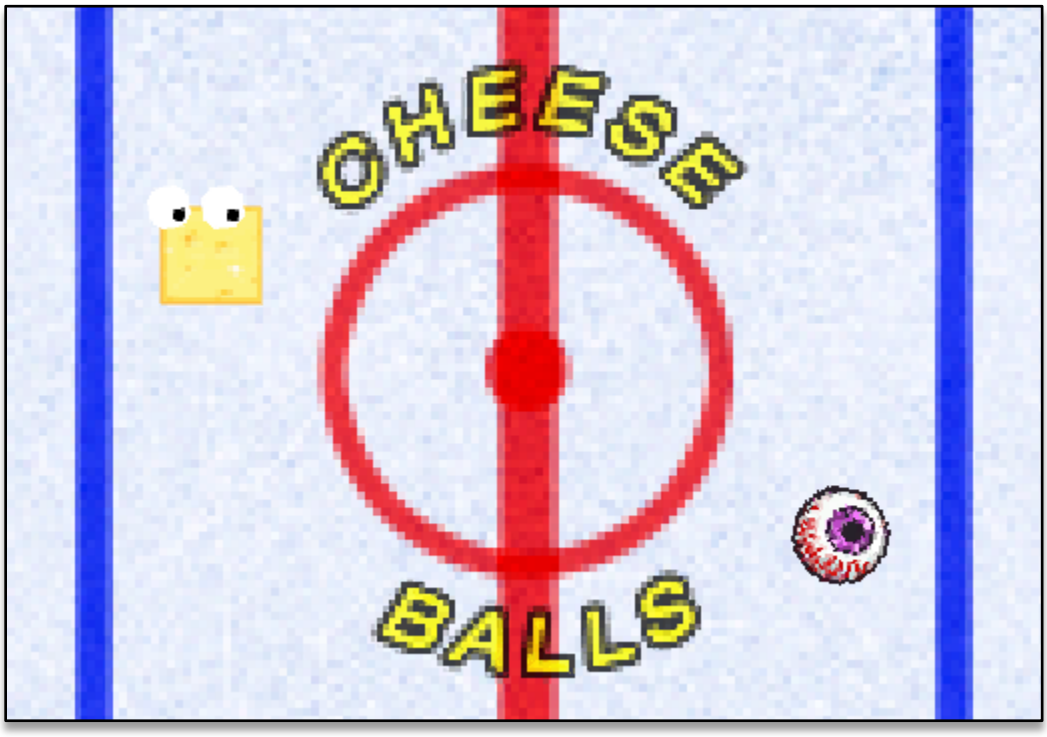
DevU
DEVELOPER UNIVERSITY

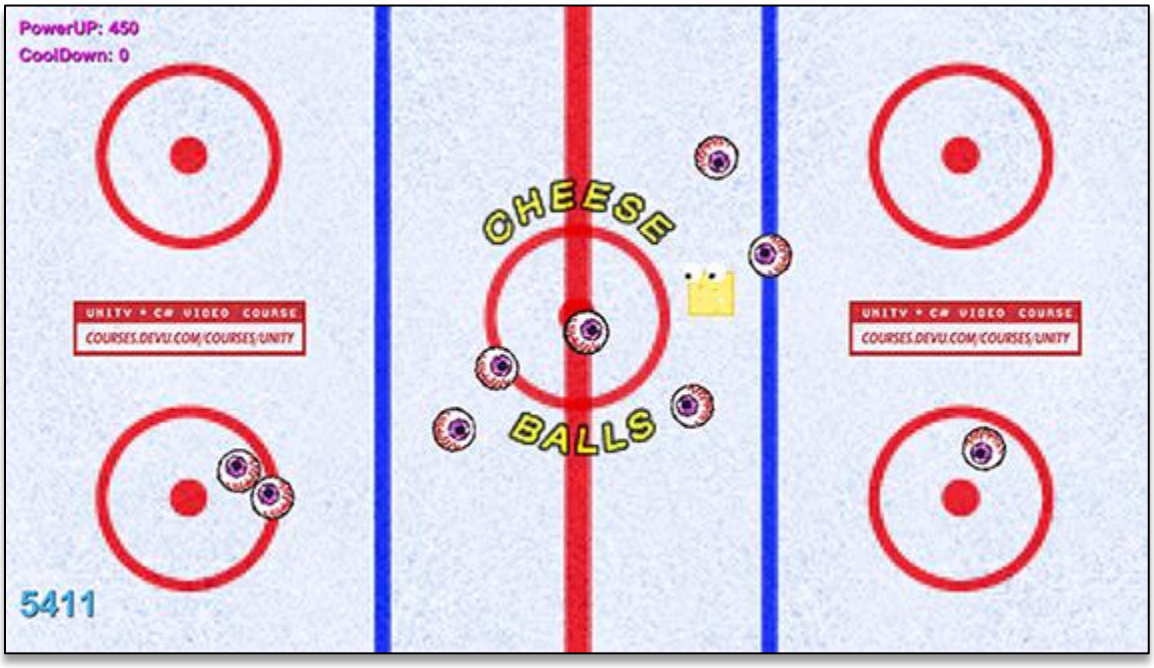
Presents



Unity2D Demonstration Project
- FREE Asset Store Package -

For More Info: http://www.DevU.com/Unity

Take a Peek Behind the Curtain and See How to Make a Fun and Simple 2D Action Game with Unity and C#!



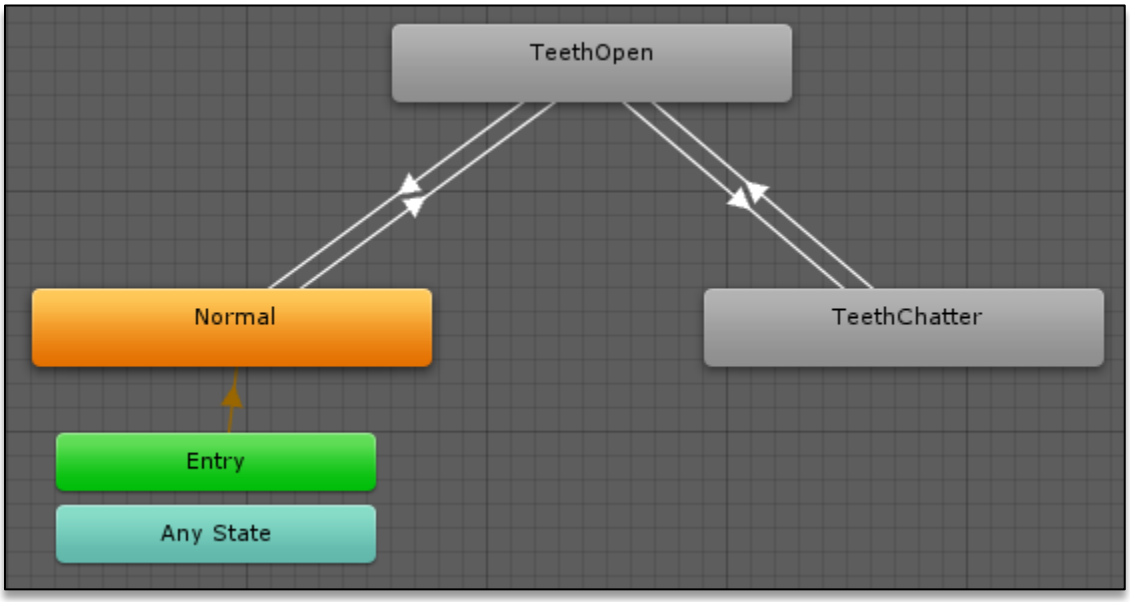
This Project Demonstrates How To...

- **Load a level (or, “Scene”) From another Level/Scene:**

```
private void LoadGame()
{
    if (BlinkTimer.Counter == 0)
    {
        SpriteRenderer pressEnterRenderer = gameObject.GetComponent<SpriteRenderer>();
        pressEnterRenderer.enabled = !pressEnterRenderer.enabled;
    }

    if (Input.GetKeyDown(KeyCode.Return) || Input.GetButtonDown("Start"))
    {
        Application.LoadLevel("MainGame");
        CubeController.Speed = 0.07f;
        SphereController.Speed = 0.06f;
    }
}
```

- **Create Animations with Spritesheets:**

- **Control Animations Using the Animator Component:**



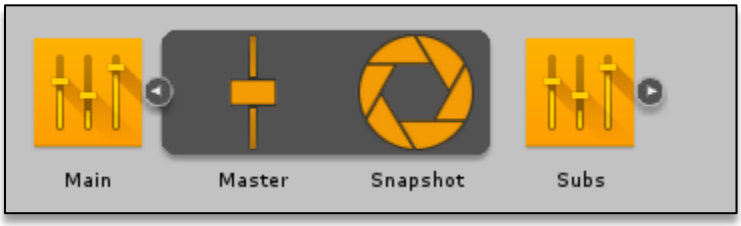
- **Control Player Movement and Behavior without Requiring Physics:**

```
public class CubeController : MonoBehaviour
{
    Vector3 Move;
    public static float Speed = 0.07f;
    const float CamWidthX = 6.2f;
    const float CamHeightY = 3.4f;

    float LeftEdge;
    float RightEdge;
    float BottomEdge;
    float TopEdge;

    float Teleport;
    public bool IsDisappear;
    public static Timer TeleportCool { get; private set; }
```

- **Loop Through Components in Parent/Child GameObjects and Apply a Process to Each Component:**

```
private void TeleportFadeForEach()
{
    SpriteRenderer[] cubeRenderers = GetComponentsInChildren<SpriteRenderer>();

    CubeController controller = GetComponent<CubeController>();

    foreach (SpriteRenderer renderer in cubeRenderers)
    {
        Color color = renderer.color;

        if (controller.IsDisappear)
            color.a = 0;
        else
            color.a = Mathf.Lerp(color.a, 1f, 0.029f * Timer.DeltaTimeMod);

        renderer.color = color;
    }
}
```

- **Work with Audio, Mixers and Effects:**



- **Display Text that Scales to the Size of the Screen:**

- **Spawn Enemies and Control their Behavior:**

```
private void SpawnEnemy()
{
    //Spawns a new enemy copied from this script every 500 ticks
    if (SpawnTimer.Counter > SpawnInterval)
    {
        float x = Random.Range(1f, 6.4f) * (Random.Range(0, 2) * 2 - 1);
        float y = Random.Range(1f, 3.5f) * (Random.Range(0, 2) * 2 - 1);
        float z = transform.position.z;
        GameObject.Instantiate(this.gameObject, new Vector3(x, y, z), transform.

        WaveCount++;
    }
    SpawnTimer.RunForwardTo(SpawnInterval);
}
```

- **Manage Game State with a GameOverManager Script:**

```
public class GameOverManager : MonoBehaviour
{
    SpriteRenderer GameOverRenderer;
    public static bool IsGameOver;
```

- **Use Various “Lerp” Techniques to Produce Change Across Frames:**

```
private void AnimatePowerUp()
{
    transform.localScale = Vector3.Lerp(transform.localScale,
}
```

- **Create Collectible Item Prefabs and Manage a “PowerUp” State:**

```
public class PowerUpManager : MonoBehaviour
{
    public GameObject PowerUpPrefab;
    private Timer SpawnTimer;
    public static Timer PowerUpMeter;

    public bool IsPowered;
```

- **Create a Custom Timer Class and Implement Time.deltaTime to Make the Game Framerate Independent:**

```
public class Timer
{
    public static float DeltaTimeMod
    {
        get { return Time.deltaTime * 60; }
    }

    public float Counter;

    public Timer(float startingPoint)...

    public void RunReverse()...

    public void RunForwardTo(float limit)...

    public void RunReverseFrom(float resetTo)...
}
```

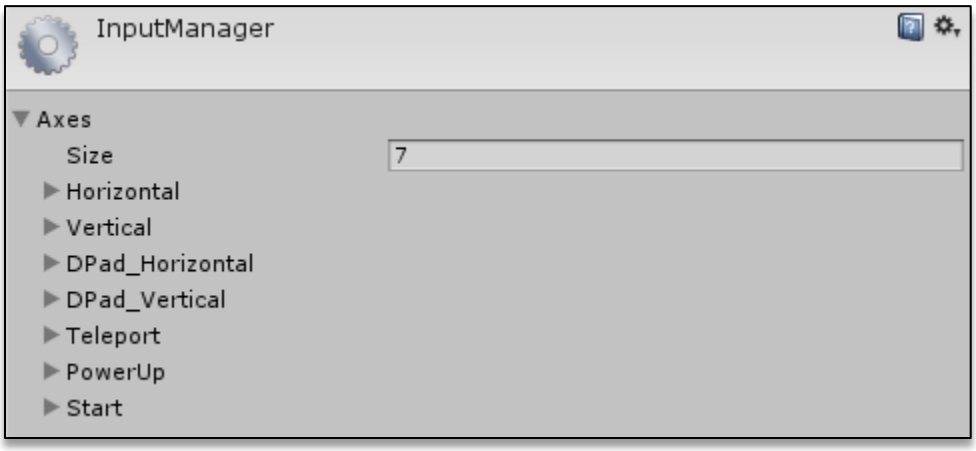
- **Change the Game's Timescale (for example: by creating a slowed-down "bullet-time" effect or to pause the game):**

```
if (Input.GetButtonDown("PowerUp") && !IsPowered && PowerUpMeter.Counter > 0)
    TimeChange(0.5f);
else if (Input.GetButtonDown("PowerUp") && IsPowered)
    TimeChange(2f);
```

- **Handle Camera Movement, Including Zooming In and Out of the Action:**

```
public class Timer
{
    public static float DeltaTimeMod
    {
        get { return Time.deltaTime * 60; }
    }

    public float Counter;

    public Timer(float startingPoint)...

    public void RunReverse()...

    public void RunForwardTo(float limit)...

    public void RunReverseFrom(float resetTo)...
}
```

- **Implement Xbox Controller input:**



Also Learn How To…

- **Create Scoring Mechanics,**
- **Reference Outside GameObjects in Code,**
- **Load Assets Dynamically at Runtime,**
- **Change Framerates for Testing Purposes,**
- **Create Animations Entirely with Code,**
- **Implement a Variety of Game Design Basics using Unity and C#!**

For More Information…

See the "**Script Synopsis**" at the top of every script to better understand the purpose of each script used in this project.

For detailed **step-by-step video lesson tutorials** showing how this game was constructed - along with additional beginner-focused C# and Unity instruction - please see the course available at:

http://courses.devu.com/courses/unity

For more information on this course check out:

http://www.devu.com/unity

All Lessons for Introduction to Unity with C#

01. Who This Course is For, Message For Beginners

02. Course Outline

03. Installation and Getting Started

04. Starting the First Project

05. Prototype Workflow

06. Basic Code Review

07. Understanding the Game Loop

08. Prototyping Continued

09. C# Fundamentals and Hello World

10. Variables and Operations

11. Variables and Operations Continued, Math Operations

12. Floats, Bools and Casting

13. If() Conditional Statements

14. If() Conditional Statements Continued

15. Complex Evaluations and States

16. Code Syntax vs Style, Errors

17. Variable Scope

18. Object Oriented Programming Intro

19. OOP, Access Modifiers, Instantiation

20. Object Containment and Method Returns

21. "Has-A" Object Containment

22. "Is-A" Inheritance Containment

23. Static Fields and Methods

24. Method Inputs and Returns

25. Reference vs Value Types

26. Intro to Polymorphism

27. Navigating the Unity API

28. Applying What You Learned and Refactoring

29. Constructors, Local Variables in the Update Method

30. Creating Collectible Items and PowerUps

31. Spawning and Managing Prefab PowerUps

32. Implementing PowerUp State Logic

33. Displaying Text, OnGUI, Method Overloading

34. Referencing Instantiated GameObjects, Parenting

35. Understanding the Lerp Method

36. Creating Pseudo Animations in Code

37. Understanding Generic Classes and Methods

38. Animations Using SpriteSheets and Animator

39. Working with Arrays and Loops

40. Debugging Unity Projects with Visual Studio

41. Camera Movement and LateUpdate

42. Playing Audio Clips

43. Routing Audio, Mixers and Effects

44. Add Scoring Mechanics and Enhancements

45. Scene Loading and Game Over Manager

46. Understanding Properties

47. Controller Mapping and Input Manager

48. Understanding Enums

49. Dealing with Null References

50. Handling Variable Framerates with Time. deltaTime

51. Preparing the Project for Final Build

52. Final Build and Project Settings

53. Introduction to the Unity Physics Engine

54. Understanding FixedUpdate vs Update

55. Movement using Physics, Singletons and Enums for Managing States

56. Attack Script and Collision Events with OnCollisionEnter2D

57. Projectiles and Stomping Attack

58. Parallax Background and Scrolling Camera

59. Infinitely Tiling Background Sprites

60. OOP Enemy Classes

61. OOP Enemy Classes Continued

62. Trigger Colliders and Dealing Damage

63. Multi-Dimensional Arrays, Procedural Platforms, Materials and Effectors

64. Finishing Touches

65. Series Wrap