Advanced Game Programming

Week 10

Homework Review

Trees

Vote On Student Requests

Obfuscating Data

Why Obfuscate Data?

Data At Rest

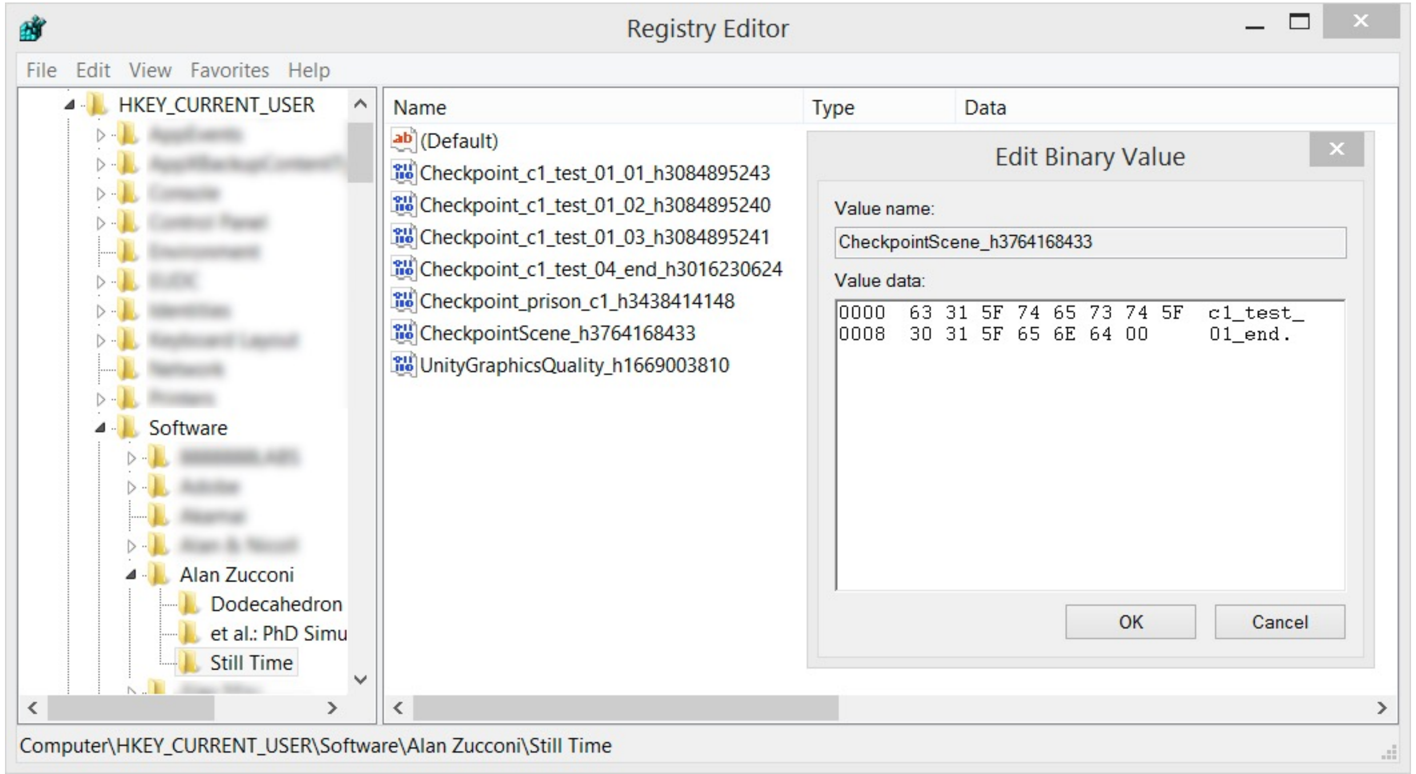
- Keep players from manipulating saves
- Keep players from ripping out assets
- Keep players from discovering spoilers

Data In Motion

- Keep players from submitting impossible/cheated scores
- Keep players from manipulating game data to have unfair advantage

Player Prefs

- On Windows Player Prefs are stored in the *system registry*.
- Access using *Regedit* (already installed)
  - HKEY_CURRENT_USER\Software\[company name]\[game name]:



(Default)
Checkpoint_c1_test_01_01_h3084895243
Checkpoint_c1_test_01_02_h3084895240
Checkpoint_c1_test_01_03_h3084895241
Checkpoint_c1_test_04_end_h3016230624
Checkpoint_prison_c1_h3438414148
CheckpointScene_h3764168433
UnityGraphicsQuality_h1669003810

Value name:
CheckpointScene_h3764168433

Value data:

```
0000  63 31 5F 74 65 73 74 5F   c1_test_
0008  30 31 5F 65 6E 64 00      01_end.
```

Computer\HKEY_CURRENT_USER\Software\Alan Zucconi\Still Time

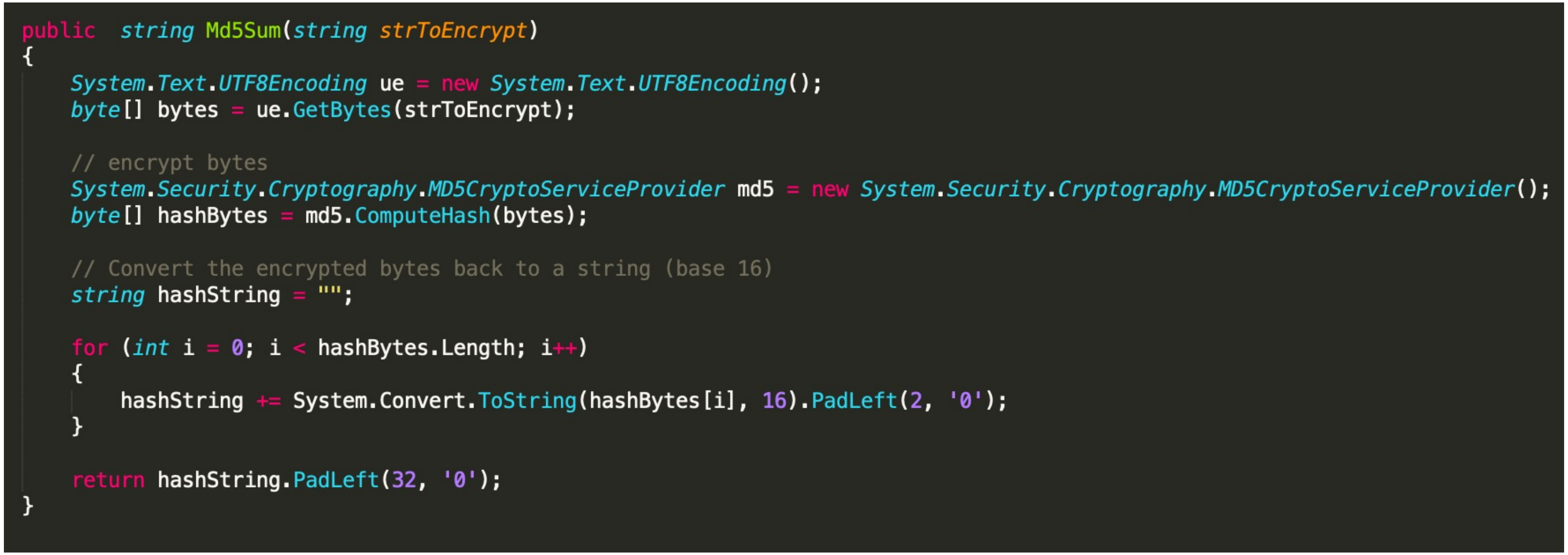
How to Protect Player Prefs?

- You cannot prevent them from being seen
- Can add a checksum that will identify an edit

What Is a Checksum?

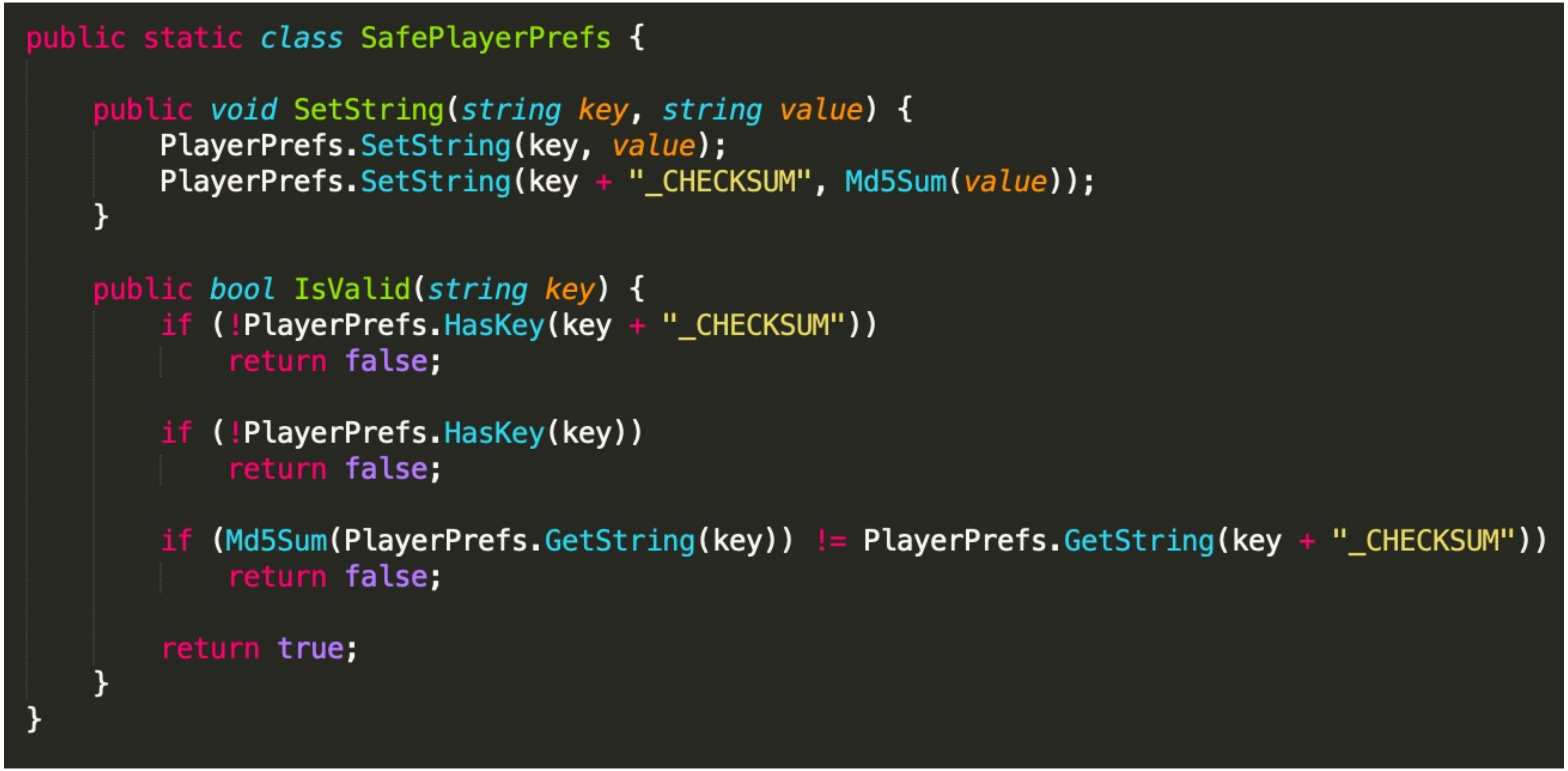
- A checksum is a hash of the values
- Common Hashes:
  - MD5 - 128-bit hash value
  - SHA-1 - 160-bit hash
- Make a wrapper class for PlayerPrefs to add a checksum

MD5 Sum (add to extension methods)



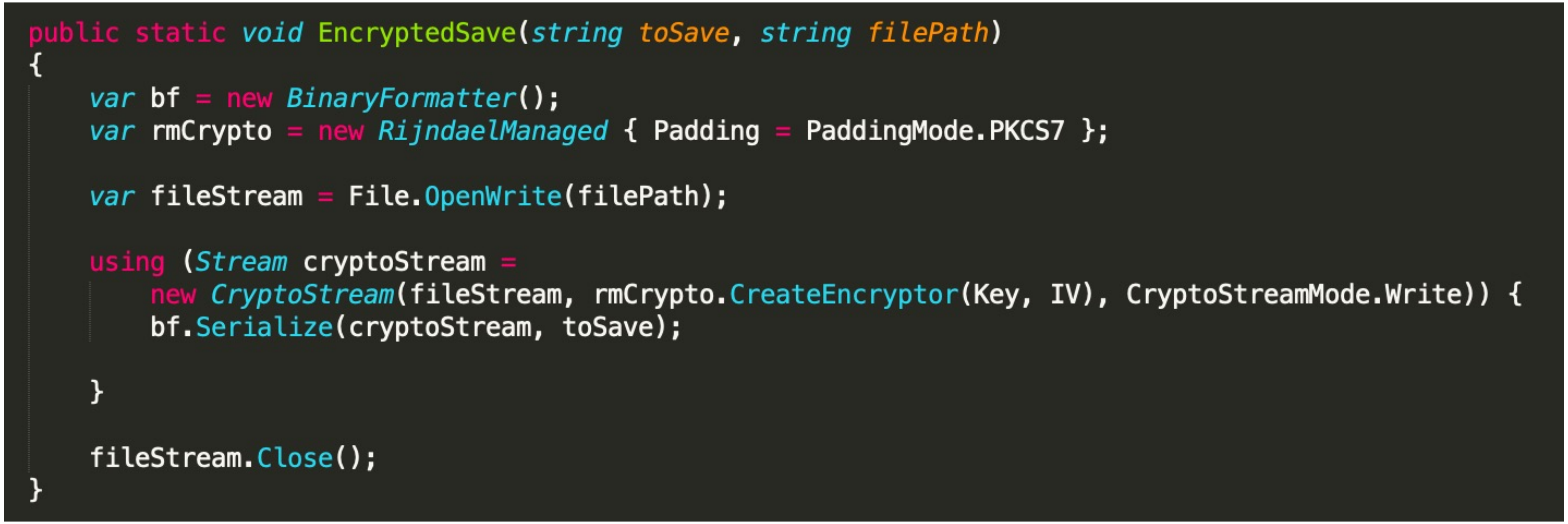
```
public  string Md5Sum(string strToEncrypt)
{
    System.Text.UTF8Encoding ue = new System.Text.UTF8Encoding();
    byte[] bytes = ue.GetBytes(strToEncrypt);

    // encrypt bytes
    System.Security.Cryptography.MD5CryptoServiceProvider md5 = new System.Security.Cryptography.MD5CryptoServiceProvider();
    byte[] hashBytes = md5.ComputeHash(bytes);

    // Convert the encrypted bytes back to a string (base 16)
    string hashString = "";

    for (int i = 0; i < hashBytes.Length; i++)
    {
        hashString += System.Convert.ToString(hashBytes[i], 16).PadLeft(2, '0');
    }

    return hashString.PadLeft(32, '0');
}
```

Safer Player Prefs



```
public static class SafePlayerPrefs {

    public void SetString(string key, string value) {
        PlayerPrefs.SetString(key, value);
        PlayerPrefs.SetString(key + "_CHECKSUM", Md5Sum(value));
    }

    public bool IsValid(string key) {
        if (!PlayerPrefs.HasKey(key + "_CHECKSUM"))
            return false;

        if (!PlayerPrefs.HasKey(key))
            return false;

        if (Md5Sum(PlayerPrefs.GetString(key)) != PlayerPrefs.GetString(key + "_CHECKSUM"))
            return false;

        return true;
    }
}
```

Serializing Data More Securely

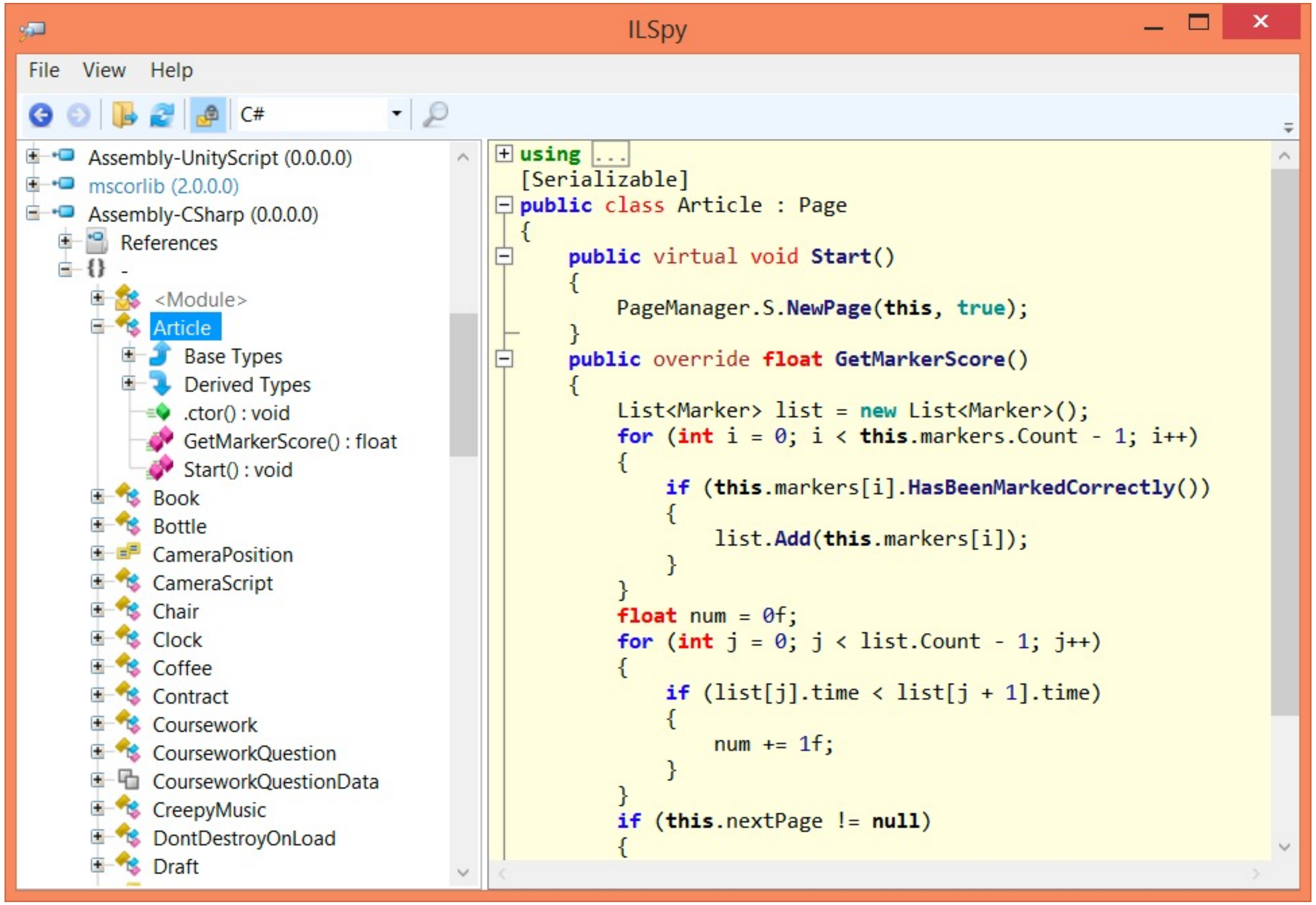


```
public static void EncryptedSave(string toSave, string filePath)
{
    var bf = new BinaryFormatter();
    var rmCrypto = new RijndaelManaged { Padding = PaddingMode.PKCS7 };

    var fileStream = File.OpenWrite(filePath);

    using (Stream cryptoStream =
        new CryptoStream(fileStream, rmCrypto.CreateEncryptor(Key, IV), CryptoStreamMode.Write)) {
        bf.Serialize(cryptoStream, toSave);

    }

    fileStream.Close();
}
```

Protecting Your Assets

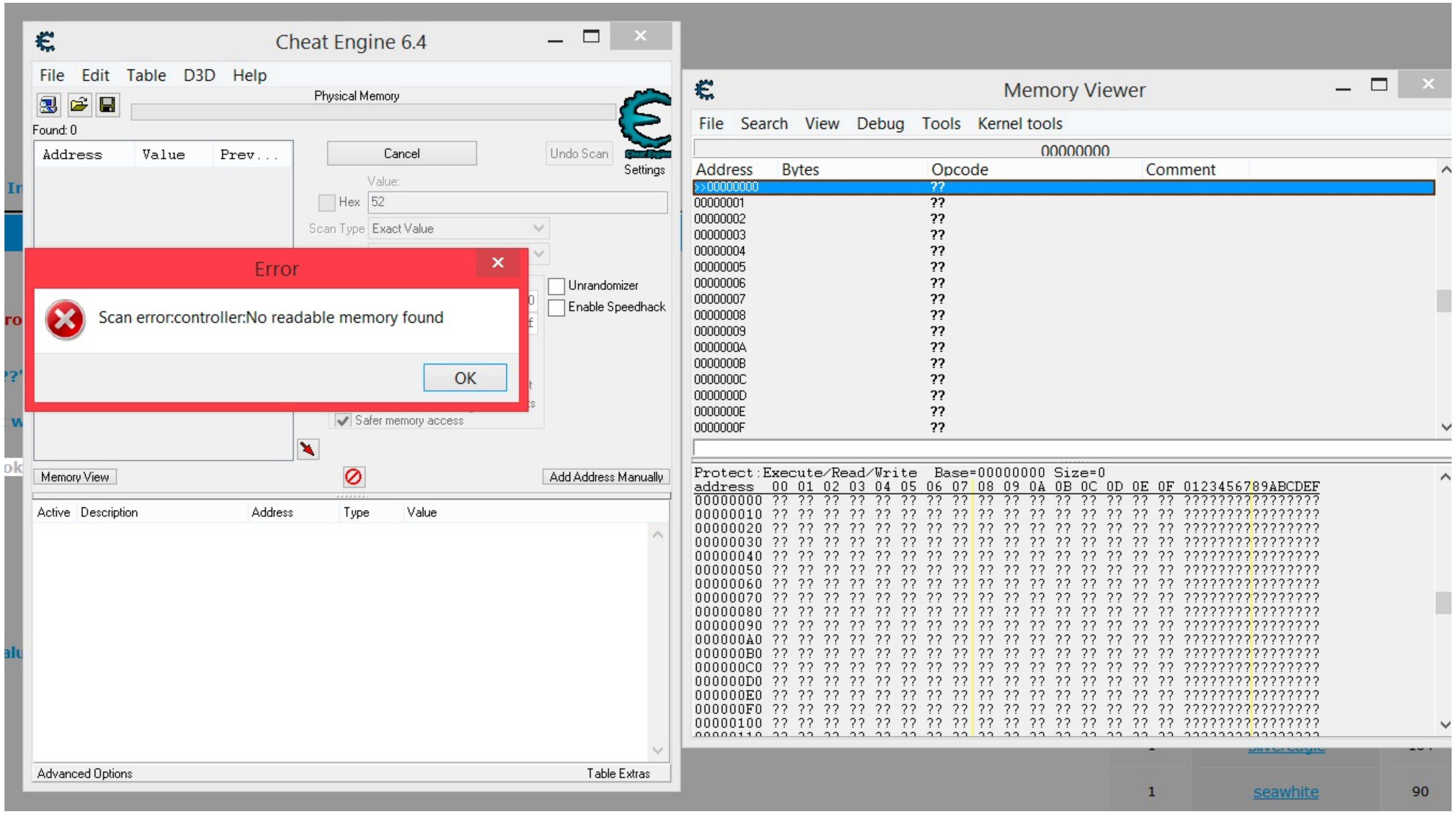
- While it is possible to use encryption to secure Assets
  - In transmission (SSL tunnel)
  - At rest (encrypt files)
- Once the data is on a computer, the client can always get the data
  - there are tools which record 3D data at the driver level, allowing users to extract models and textures as they are sent to the GPU.
- If users want your assets, they’ll get it.

Preventing Your Code From Being Stolen



```
using ...
[Serializable]
public class Article : Page
{
    public virtual void Start()
    {
        PageManager.S.NewPage(this, true);
    }
    public override float GetMarkerScore()
    {
        List<Marker> list = new List<Marker>();
        for (int i = 0; i < this.markers.Count - 1; i++)
        {
            if (this.markers[i].HasBeenMarkedCorrectly())
            {
                list.Add(this.markers[i]);
            }
        }
        float num = 0f;
        for (int j = 0; j < list.Count - 1; j++)
        {
            if (list[j].time < list[j + 1].time)
            {
                num += 1f;
            }
        }
        if (this.nextPage != null)
        {
```

Protecting Active Memory

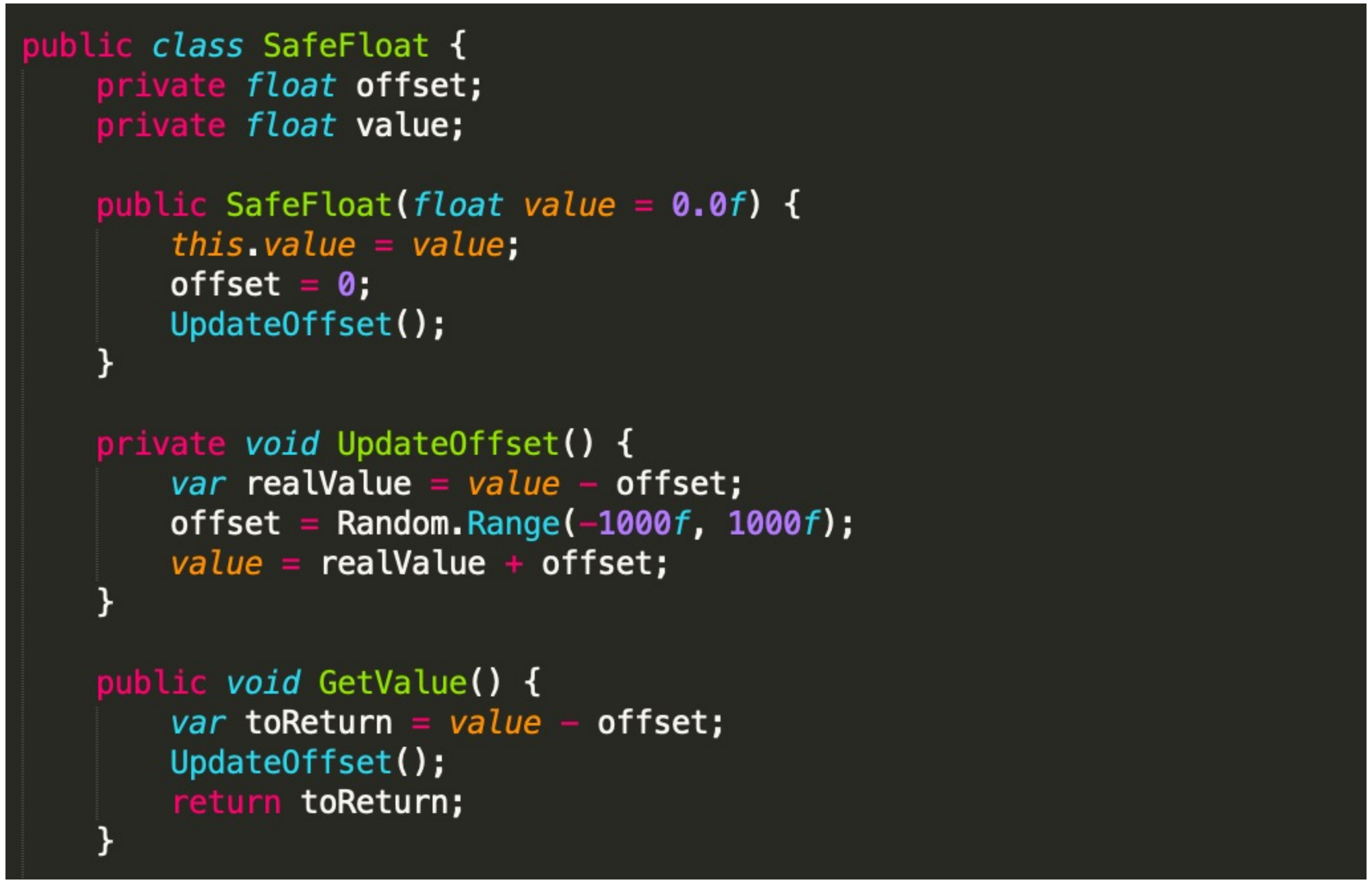


Scan error:controller:No readable memory found

```
00000000
>>00000000          ??
00000001            ??
00000002            ??
00000003            ??
00000004            ??
00000005            ??
00000006            ??
00000007            ??
00000008            ??
00000009            ??
0000000A            ??
0000000B            ??
0000000C            ??
0000000D            ??
0000000E            ??
0000000F            ??
```

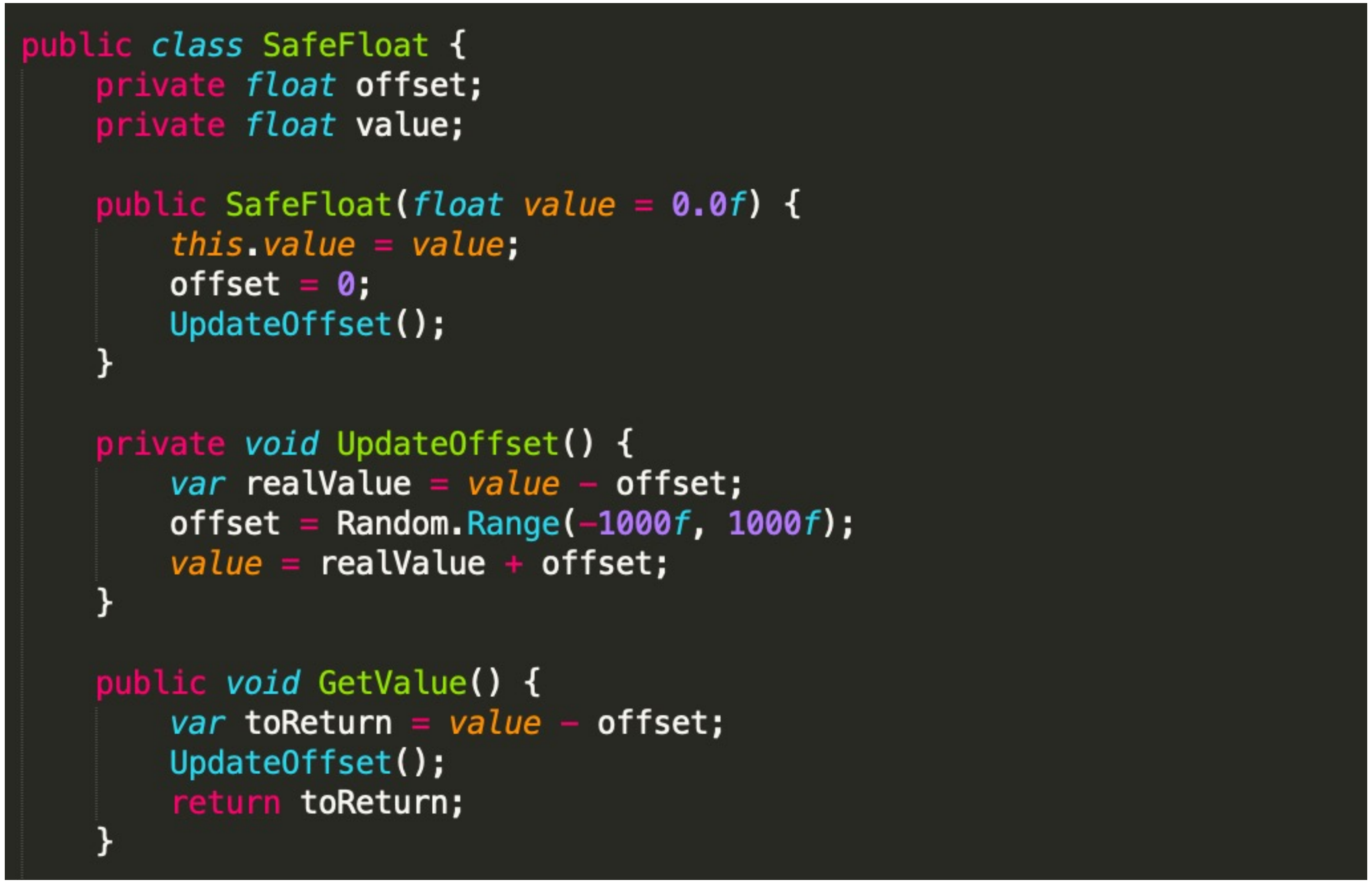
```
Protect :Execute/Read/Write  Base=00000000 Size=0
address   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF
00000000  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
00000010  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
00000020  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
00000030  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
00000040  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
00000050  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
00000060  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
00000070  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
00000080  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
00000090  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
000000A0  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
000000B0  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
000000C0  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
000000D0  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
000000E0  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
000000F0  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
00000100  ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ????????????????
```

Protecting Active Memory



```
public class SafeFloat {
    private float offset;
    private float value;

    public SafeFloat(float value = 0.0f) {
        this.value = value;
        offset = 0;
        UpdateOffset();
    }

    private void UpdateOffset() {
        var realValue = value - offset;
        offset = Random.Range(-1000f, 1000f);
        value = realValue + offset;
    }

    public void GetValue() {
        var toReturn = value - offset;
        UpdateOffset();
        return toReturn;
    }
```

Protecting Active Memory (cont.)



```
public class SafeFloat {
    private float offset;
    private float value;

    public SafeFloat(float value = 0.0f) {
        this.value = value;
        offset = 0;
        UpdateOffset();
    }

    private void UpdateOffset() {
        var realValue = value - offset;
        offset = Random.Range(-1000f, 1000f);
        value = realValue + offset;
    }

    public void GetValue() {
        var toReturn = value - offset;
        UpdateOffset();
        return toReturn;
    }
```

What Can We Take Away?

- If people want your code or assets, they can get them.
- If people want to mess with save files in a single-player game, it’s very difficult to completely prevent it
- In a multiplayer situation, you can monitor and prevent cheating on the server side
  - Assume everything the client sends you could be invalid

Input Systems

Goals

- Performance is important (no input lag)
- Allow for multiple types of input
  - Controller
  - Keyboard/mouse
  - Touches
- Allow changing between types of input
- Easy to maintain / add inputs
- Easy to debug
- Configurability

Event Based Control

- Centralize an input system based on Events
- Register for input events and interpret them in each class
- PROS:
  - No huge “control” codebase that’s difficult to manage
- CONS:
  - Difficult to do priority (everything registers for keystrokes / inputs)
  - Has no understanding of state
    - What if you’re in a menu? Paused? In design mode instead of movement mode?
  - Control is completely decentralized
    - Each class interprets what an input does

Input Context

- Defines what inputs are available at a given time
- Most games exist with different states
  - Moving around the world
  - Piloting a craft
  - Navigating a menu
- Contexts contain different types of input
  - Actions
  - States
  - Ranges

FSM Based Control

- Register for an input and a (set of) state(s)

```
Services.Input.Register(KeyPress.A, State.Paused,
        PauseSystem.Submit)
```

- PROS:
  - Understands state – doesn’t fire at the incorrect time
- CONS:
  - You have to manage a lot of state in code
  - End-systems become extremely coupled
  - Lots of complicated code in classes

Actions

- Single-time event in the game
    - Casting a spell
    - Opening a door
    - Talking to an NPC
    - Picking up an object
- Usually occurs when a button is pressed (or released)

States

- States are continuous activities
  - Running
  - Shooting
  - Charging
  - Scrolling through menus
- Can represent as a binary flag – on or off

Range

- Input that can have a numeric value associated with it
- Typically for analogue input
    - Joysticks
    - Thumbsticks
    - Mice
    - Spinners

Invert FSM/Event Model

- Define an input map for each context
- Store a Dictionary of Input events to actions
- Have the Input System dispatch the input
    - Callbacks
    - Polling

Chain of Responsibility Pattern

- Design pattern consisting of a source of **command objects** and a series of **processing objects.**
- Each processing object has logic defining what types of command objects it can handle, and what commands it passes the rest to the next processing object
- Other version involve dispatchers to create a *tree of responsibility*
- Tree of responsibility w/recursion on portions of the tree can be used to create parsers
- For input systems: the more specific the context, the higher priority it should carry.
  - Menus
  - Chat windows
  - Debug modes

New Input System

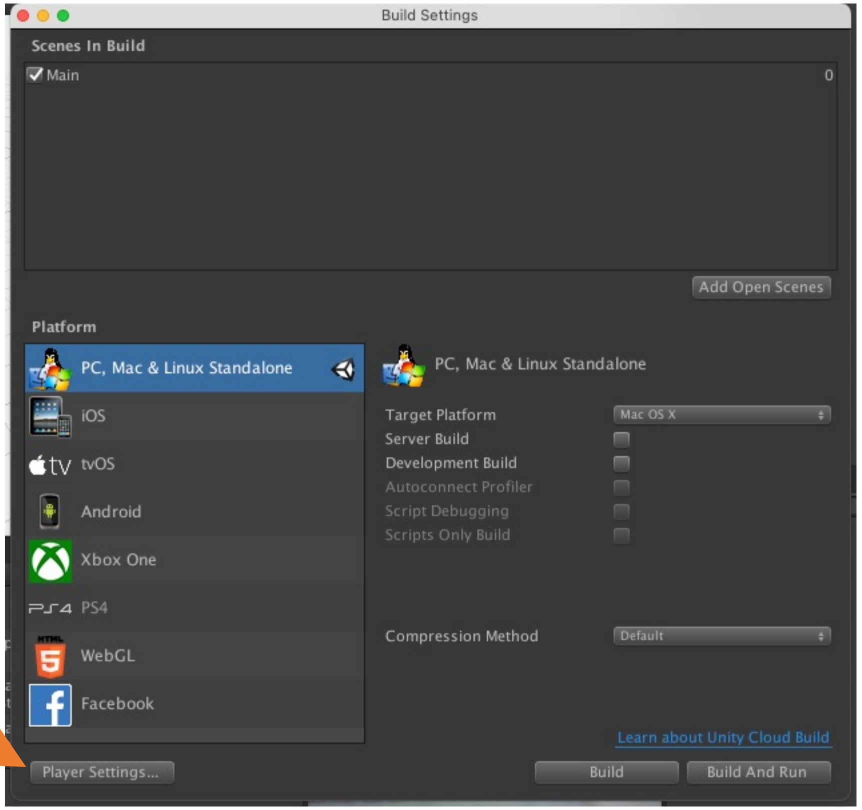
- Create a data structure:
  - List of valid actions
  - List of valid states
  - List of valid ranges and their current values
- Pass this data on to an ordered chain of callbacks.
- If a callback handles a piece of input, it should generally remove it from the data structure so that further callbacks don't issue duplicate commands.

Hardware to Game – Data Driven

- Process all raw inputs as buttons or axis's
  - Buttons are states or actions
  - Axis map to ranges
- Make a definition of each action, state, and range in game
- Create an input map data file, with a section for each context
- Read data file into game, and for each valid context, fire all callbacks that were defined in input map

How to Change Input Type On Fly

- Preprocessor Directives
  - In Unity player settings (Build -> Player Settings)
  - Added as scripting define symbols
- You can add preprocessor directives
  - (CONTROLLER_SUPPORTED, TOUCH_SUPPORTED)
- Wrap different platforms builds in different preprocessor directives.


Build Settings

Scenes In Build

✓ Main 0

Add Open Scenes

Platform

PC, Mac & Linux Standalone
iOS
tvOS
Android
Xbox One
PS4
WebGL
Facebook

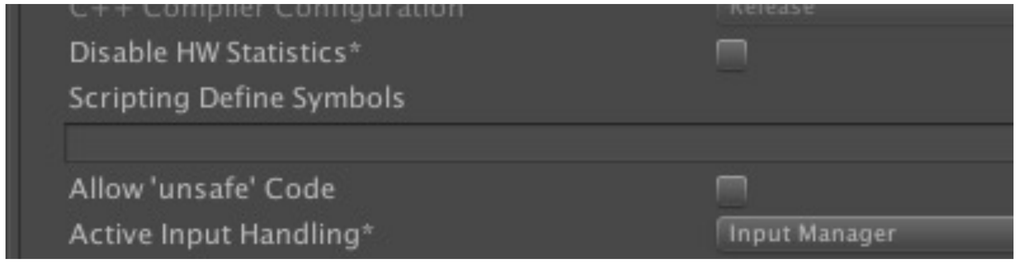
PC, Mac & Linux Standalone

Target Platform: Mac OS X
Server Build
Development Build
Autoconnect Profiler
Script Debugging
Scripts Only Build

Compression Method: Default

Learn about Unity Cloud Build

Player Settings... Build Build And Run


C++ Compiler Configuration: Release
Disable HW Statistics*
Scripting Define Symbols
Allow 'unsafe' Code
Active Input Handling*: Input Manager

Why Use Preprocessor Directives?

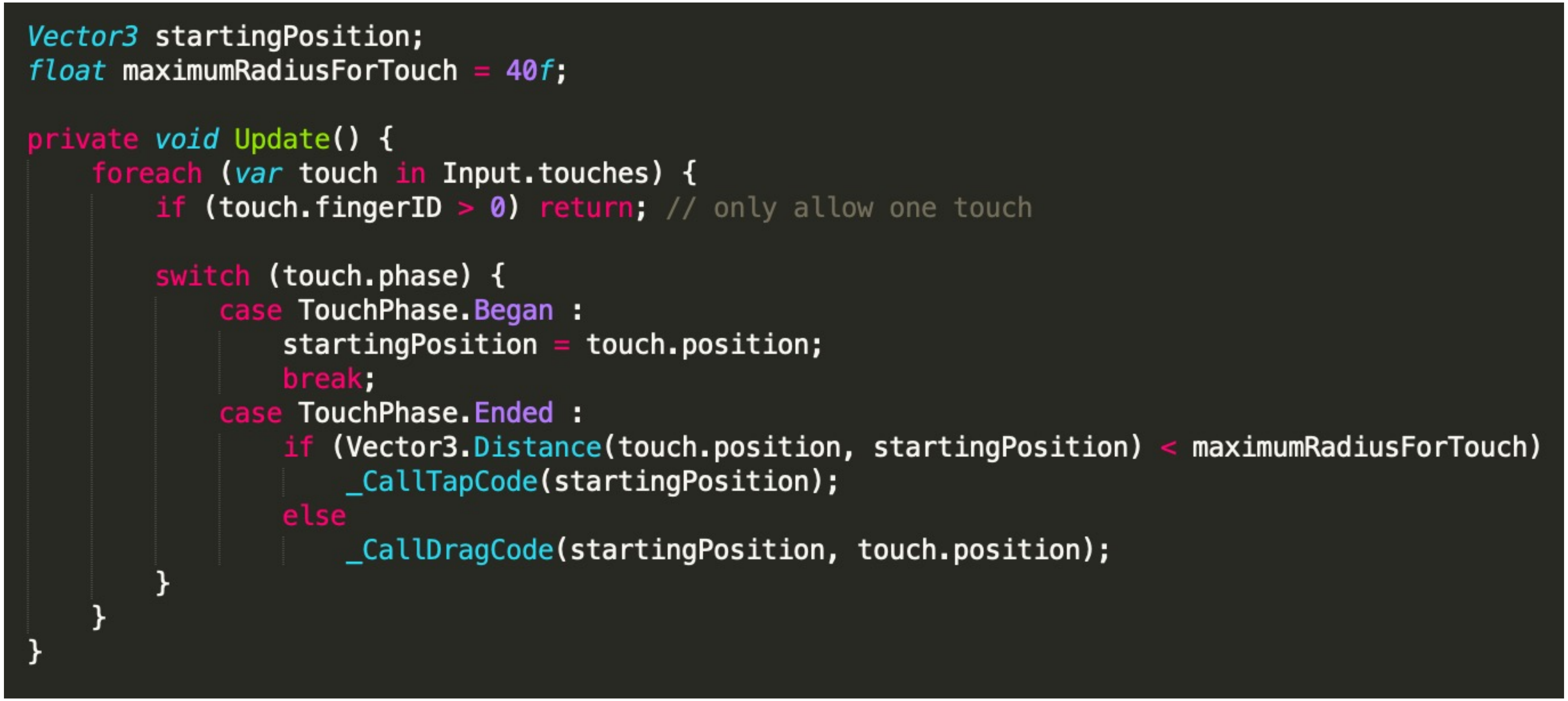
- You may want to allow fewer input types than the maximum allowed by the platform
  - Some Windows devices allow touch, some platforms allow keyboards, etc.
  - If you haven’t planned and QA-d for an input on a platform, you may end up in unintended states
  - Using unintended inputs can allow for access to debug states

Touches

Touches vs. Buttons

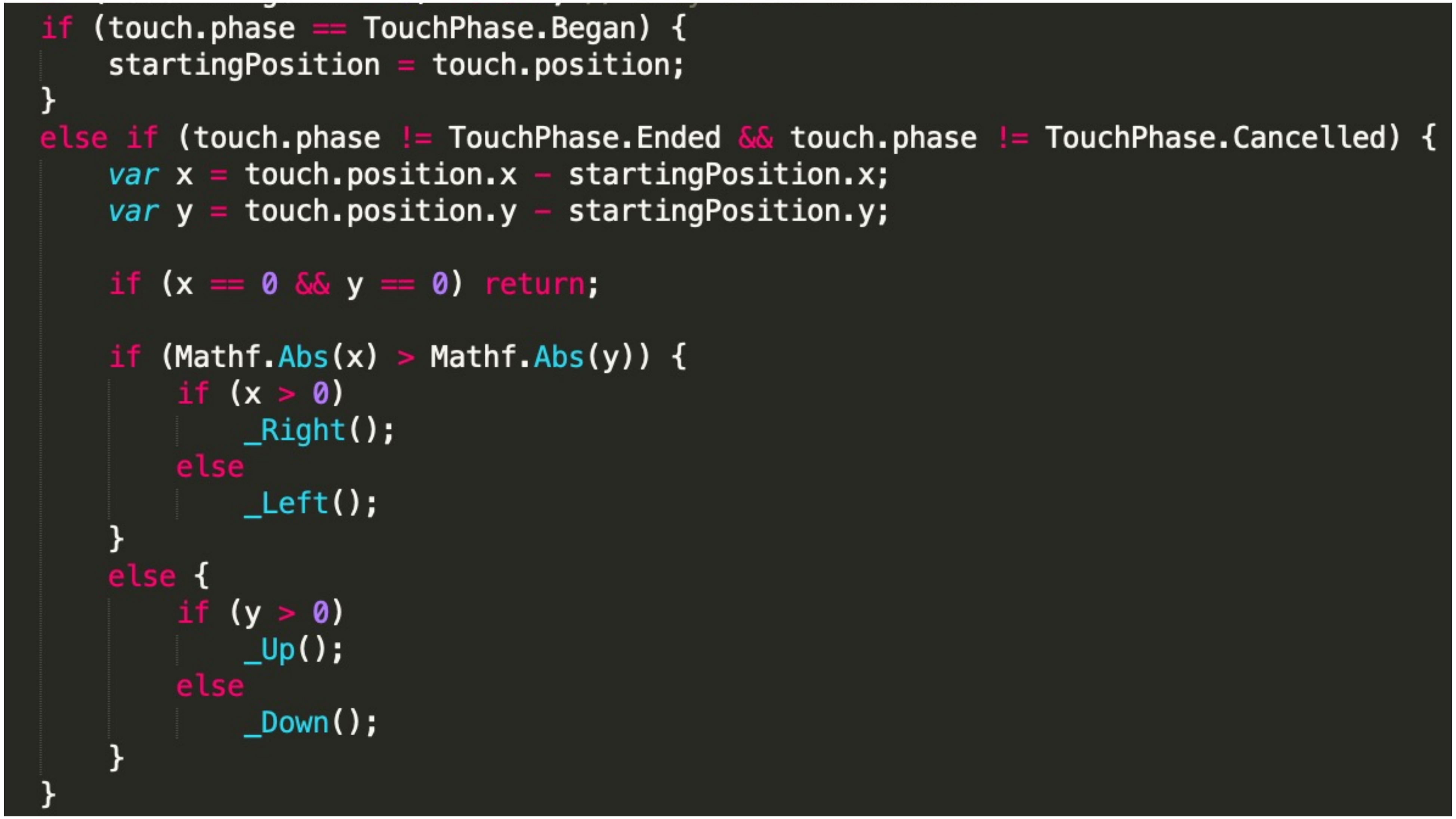
- Touches involve gesture / can have more complicated meaning than “pressing a button”.
- In unity, you get a touch object
  - “position” in Pixel coordinates (Screen space)
  - ”phase” which can be “Began”, “Moved”, “Stationary”, “Ended”, and “Cancelled”
  - “fingerID” for each touch (0 for first touch, 1 for second, etc.)
  - “pressure” and “maximumPossiblePressure” for force sensitive displays
  - “deltaPosition” for each frame

Simple Code for Tapping



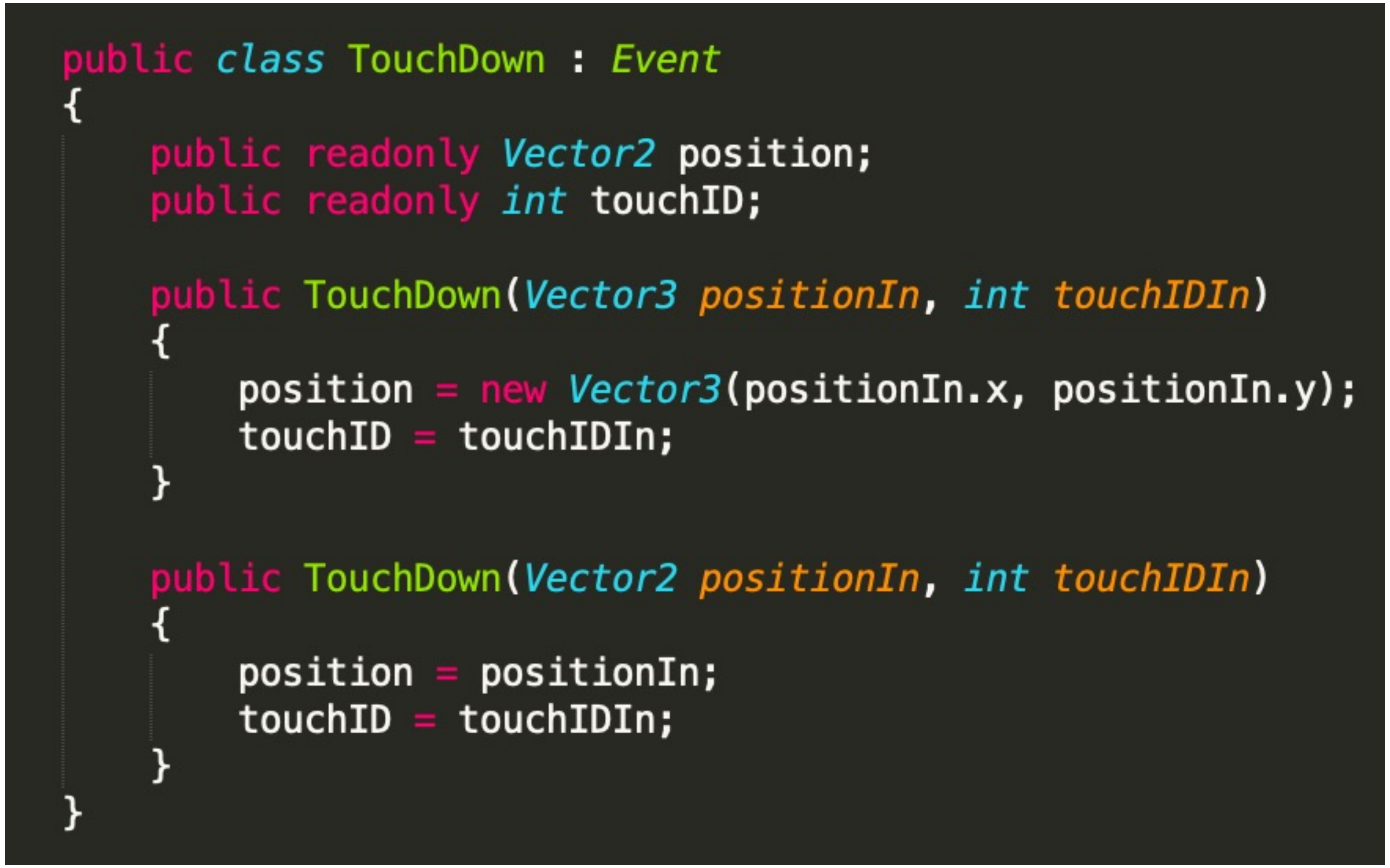
```
Vector3 startingPosition;
float maximumRadiusForTouch = 40f;

private void Update() {
    foreach (var touch in Input.touches) {
        if (touch.fingerID > 0) return; // only allow one touch

        switch (touch.phase) {
            case TouchPhase.Began :
                startingPosition = touch.position;
                break;
            case TouchPhase.Ended :
                if (Vector3.Distance(touch.position, startingPosition) < maximumRadiusForTouch)
                    _CallTapCode(startingPosition);
                else
                    _CallDragCode(startingPosition, touch.position);
        }
    }
}
```

Virtual Joystick



```
if (touch.phase == TouchPhase.Began) {
    startingPosition = touch.position;
}
else if (touch.phase != TouchPhase.Ended && touch.phase != TouchPhase.Cancelled) {
    var x = touch.position.x - startingPosition.x;
    var y = touch.position.y - startingPosition.y;

    if (x == 0 && y == 0) return;

    if (Mathf.Abs(x) > Mathf.Abs(y)) {
        if (x > 0)
            _Right();
        else
            _Left();
    }
    else {
        if (y > 0)
            _Up();
        else
            _Down();
    }
}
```

Event-based Architecture



```
public class TouchDown : Event
{
    public readonly Vector2 position;
    public readonly int touchID;

    public TouchDown(Vector3 positionIn, int touchIDIn)
    {
        position = new Vector3(positionIn.x, positionIn.y);
        touchID = touchIDIn;
    }

    public TouchDown(Vector2 positionIn, int touchIDIn)
    {
        position = positionIn;
        touchID = touchIDIn;
    }
}
```

Recommendations

- Don’t use mouse inputs for a touch device
- Only use as many touches as your game uses
- Repackage the individual touches into a data structure that tracks them over time if you need more information about gestures
- UnityEngine.GestureRecognizer may help if you need gesture recognition.
- Use “Input.touchSupported” to separate your mobile and desktop input systems
    - #if UNITY_IOS || UNITY_ANDROID
    - #elif UNITY_PC
    - endif