



Botting in Video Games

By Liam Gomez, Zach Mekaelian, Nicholas Malamud, Young
Taek William Oh

Overview

- Botting has become a major issue in modern video games.
- Not only is it used to gain an unfair advantage, but it is also used to exploit games for financial gain.
- Botting uses coding scripts to emulate clicks and motions that players would actually use in real time to gain rewards.
- This creates not only a problem for game developers, but also detracts the playing experience for real players.

The Experiment

- We developed an experiment that takes on an adversarial approach to the problem space.
- Our group was divided into two teams with different objectives in mind:
 - A team specializing in game development and security features.
 - A team specializing in developing bot scripts to bypass the security features of the game.
- Our overall goal with this experiment was to find potential solutions to combat botting without diminishing the player experience.

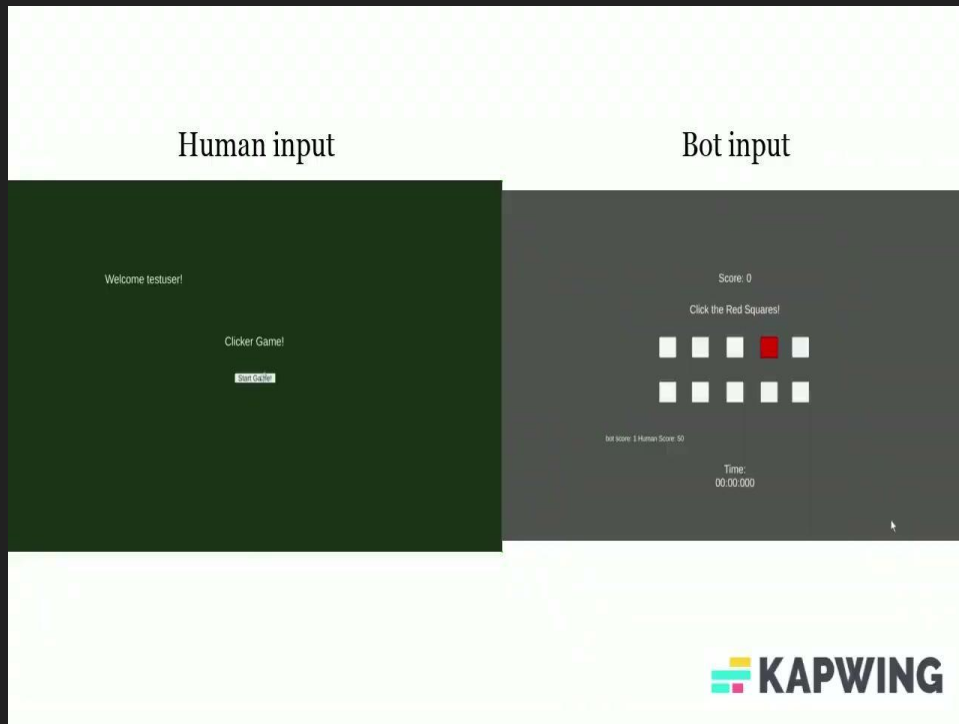
Completed Tasks: Game Team

- Implementation of an online only game featuring randomized clickable colored boxes using the Unity engine
- Implementation of a game launcher using .NET Maui and online authentication features from PlayFab
- Implementation of account creation features and a CAPTCHA using BlazorCaptcha
- Game botting detection system which tracks and validates player movement data against their physical device
- Basic memory encapsulation and protection was implemented as well utilizing a Unity Asset store plugin called Anti Cheat Free by GuardingPearSoftware.

Completed Tasks: Bot Team

- Bot script to click the specified randomized boxes through color recognition using OpenCV
- Functions to randomize player movement in a constant and curved motion rather than teleporting to simulate human like mouse movements
- Able to generate a large dataset of captchas for analysis
- Code injection was attempted

Demonstration



Bot Movement Code

```
def randMove(x,y):  
    #random placement of click on square button  
    rX = random.randint(int(x) - 25, int(x) + 25)  
    rY = random.randint(int(y) - 20, int(y) + 5)  
    #variables holds current position of mouse  
    curX, curY = pag.position()  
    #number of intervals of random mouse movement to point  
    rStep = 1/random.randint(5,10)  
    #variable holds percent of distance to the point (0-1) = (0%-100%) <- 1 = destination  
    pDist = rStep  
    #amount of randomness between the points of movement (higher = more random)  
    rAmount = 100 - 10 * int(1000/math.dist([curX, curY], [rX, rY]))  
  
    #while mouse before destination  
    while (pDist <= 1):
```

Bot Movement Code Continued

```
#while mouse before destination
while (pDist <= 1):
    #get the next position for the interval
    curX, curY = pag.getPointOnLine(curX, curY, rX, rY, pDist)
    #randomize the next point
    curY = curY + random.randint(-rAmount, rAmount)
    curX = curX + random.randint(-rAmount, rAmount)
    #move to point it with random speed
    pag.moveTo(curX, curY, random.uniform(0.05, 0.1))
    #go to next interval
    pDist = pDist + rStep
    #decrease random for each interval, as person gets close to button they slow down to not miss
    rAmount = int(rAmount * (1-pDist))
```


Botting Detection System and Anti-Cheat

```
public void MakeJudgement()
{
    if (botScore * 9 > humanScore)
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
    else
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 2);
    }
}
```

```
private ProtectedInt32 botScore = 0;
private ProtectedInt32 humanScore = 0;
```

```
public class AntiCheatListener : MonoBehaviour
{
    // Attach to the FieldCheatDetector OnFieldCheatDetected event.
    private void Start()
    {
        OPS.AntiCheat.Detector.FieldCheatDetector.OnFieldCheatDetected += Custom_OnFieldCheatDetected;
    }

    // Your custom event, what to do, if a cheat got detected.
    private void Custom_OnFieldCheatDetected()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
    }
}
```

```
void onMouseMove(InputAction.CallbackContext context)
{
    moved = true;
}

void onEscapePressed(InputAction.CallbackContext context)
{
    Application.Quit();
}

void onPosChange(InputAction.CallbackContext context)
{
    Vector2 mousePosition = context.ReadValue<Vector2>();

    if (moved)
    {
        moved = false;
        humanScore += 1;
        output.text = "bot score: " + botScore + " Human Score: " + humanScore;
    }
    else
    {
        botScore += 1;
        output.text = "bot score: " + botScore + " Human Score: " + humanScore;
    }
}
```

Results and Interesting Finds:

- Despite including human like movement, the game is able to filter out the bot like movement as “teleporting” over positions.
- Physical device validation is a much simpler and more accurate approach to bot detection than machine learning.
- User Input is able to be uniquely identified by which device it is coming from and filtered accordingly.
- It is possible for false positives to occur no matter how good the botting detection system is. Therefore, any bans made should be human reviewed.
- Most pre-developed anti-bot software is designed for online applications and websites rather than games.

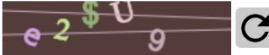
Attempt to identify Captchas

[How to break a CAPTCHA system in 15 minutes with Machine Learning | by Adam Geitgey | Medium](#)

Email:

Username:

Password:



Enter Captcha:

e2\$U9

Collect Dataset of Captchas

```
def recCaptcha(capCount):  
    #generate 100 images  
    if(capCount <= 100):  
        reload = (706, 516)  
        textStart = (412, 793)  
        #textEnd = (556, 791)  
        #Copy Text  
        pag.moveTo(textStart)  
        pag.doubleClick()  
        pag.doubleClick()  
        pag.keyDown('ctrl')  
        pag.keyDown('c')  
        pag.keyUp('c')  
        pag.keyUp('ctrl')  
        #Take screenshot of captcha  
        s = pag.screenshot(region=(424, 491, 254, 60))  
        capText = replaceInvalid(tkinter.Tk().clipboard_get())  
        s.save("Captchas/" + capText + '.png')  
        #Next One  
        pag.moveTo(reload)  
        pag.click()  
        capCount = capCount + 1  
    recCaptcha(capCount)
```

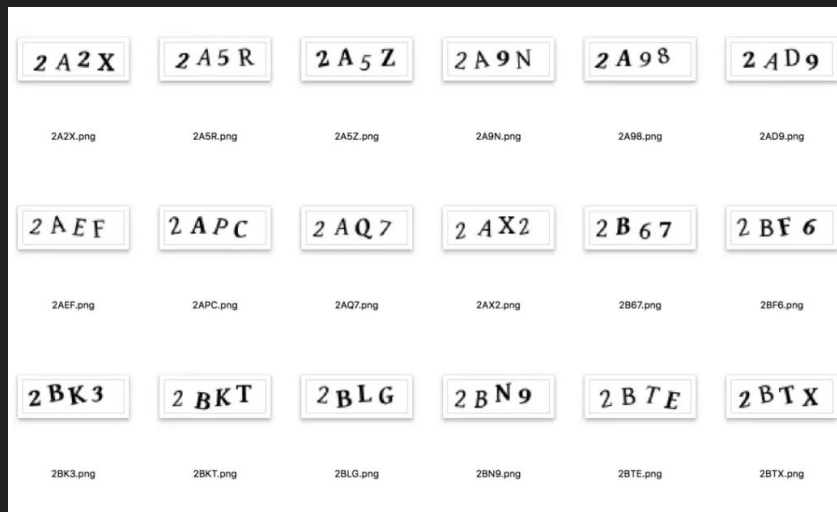
Collect Dataset of Captchas



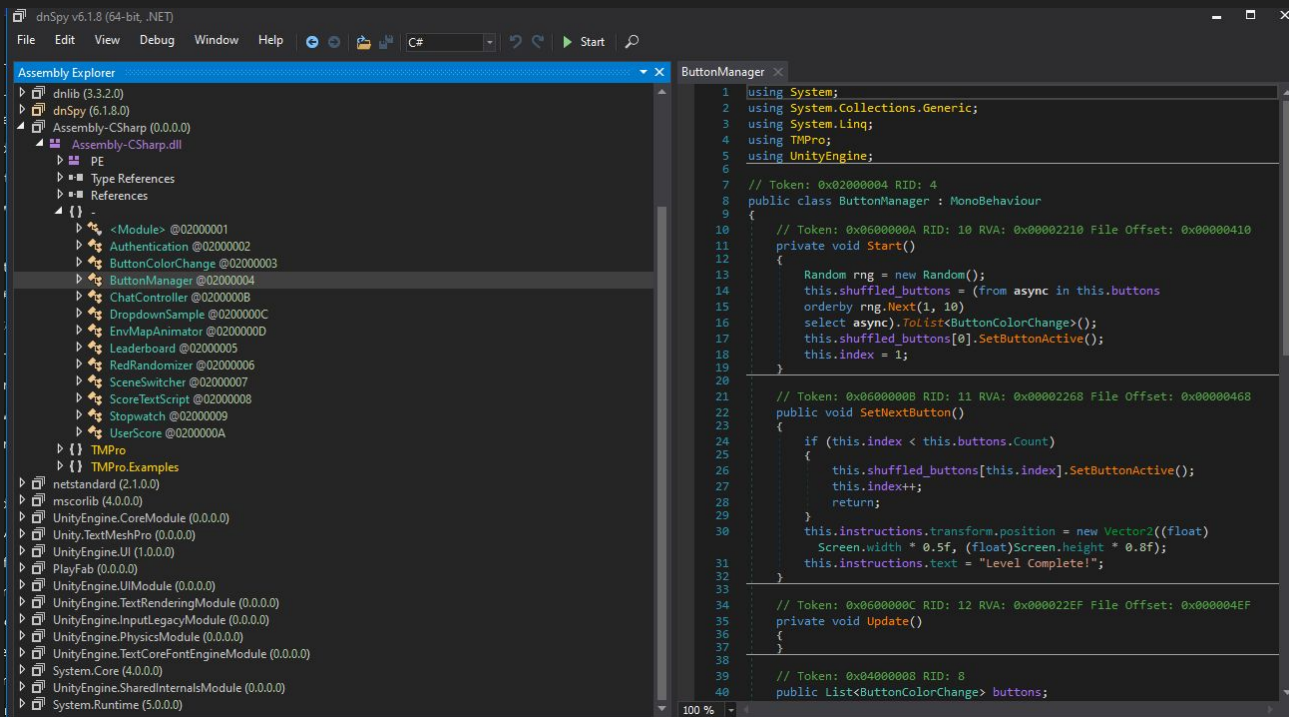
Result

- Failed to identify
- Different captcha
- Lines make it difficult to separate letters
- Non-alphabetical/Digit letters
- Different colors
- Partial Letters

```
def replaceInvalid(str):  
    str1 = str.replace('\\', '◆')  
    str2 = str1.replace('/', '◆')  
    str3 = str2.replace(':', '◆')  
    str4 = str3.replace('*', '◆')  
    str5 = str4.replace('?', '◆')  
    str6 = str5.replace('"', '◆')  
    str7 = str6.replace('<', '◆')  
    str8 = str7.replace('>', '◆')  
    str9 = str8.replace('|', '◆')  
    return str9
```



Code injection



First break apart the game's assembly dll file...

```

namespace ClickInject
{
    1 reference
    public class Cheat : UnityEngine.MonoBehaviour
    {
        0 references
        public void Start()
        {
            buttonManager = FindObjectOfType<ButtonManager>();
        }
        0 references
        public void Update()
        {
            buttonManager.buttons.Clear();
        }
        private ButtonManager buttonManager;
    }
}

```

```

1
2 namespace ClickInject
3 {
4     public class Loader
5     {
6         static UnityEngine.GameObject gameObject;
7
8         public static void Load()
9         {
10             gameObject = new UnityEngine.GameObject();
11             gameObject.AddComponent<Cheat>();
12             UnityEngine.Object.DontDestroyOnLoad(gameObject);
13         }
14
15         public static void Unload()
16         {
17             UnityEngine.Object.Destroy(gameObject);
18         }
19     }
20 }
21

```

Write code to modify game values...

then load the code into a dll

Code Injection

- Code injection was attempted using a tool called SharpMonoInjector
 - Idea was to create a dll which would then be inserted into the program while it was running, overriding the dll file which would then overwrite data values or run methods within the program
- Ultimately unsuccessful



Conclusion

- Novel methods exist that can be used to combat botting in online video games without diminishing the player experience.
- Botting detection can potentially be accomplished in-house with very little resources.
- Despite our success, our experiment is limited in scope and more research is needed to determine if our approach is a viable strategy for more complicated games.

Future Direction of Research

- Finding ways to validate user input against physical devices outside of the Unity game engine.
- Researching more into how inputs are registered virtually through scripts and if they can be spoofed to seem as if they are coming from valid physical devices.
- Performing similar experiments with keyboards and other controllers.
- Finding better method to solve captchas
- Find more effective ways to inject code into the game