JAN 27, 2019 - 7 MINUTE READ - COMMENTS - GAME HACKING

Cheating at Moonlighter - Part 2 - Changing Game Logic with dnSpy

- Why dnSpy?
- Increasing Will's Damage
  - DealDamageToEnemy
  - `CalcHitDamage`
  - Increasing Will's Damage
    - Returning float.PositiveInfinity
    - Return hitStrength + num
- Modifying Will's Stats
  - HeroMerchantProjectile.DealDamage

Who am I?

I am Parsia, a security engineer at Electronic Arts.

I write about application security, reverse engineering, Go, cryptography, and (obviously) videogames.

Click on About Me! to know more.

Collections

    - The Case of the Missing Intelligence
  - Adding Extra Stats
  - A Closer Look at Base Stats
- Lesssons Learned

In part 1 we messed a bit with Moonlighter but modifying the save file. In this part, we will modify game logic using dnSpy.

We will modify our damage, player stats and discover a hidden stat.

Why dnSpy?

Moonlighter is built with the Unity game engine (C#). Game logic is usually in `Assembly-CSharp.dll`. In my VM, it's at:

- `C:\Program Files (x86)\Steam\steamapps\common\Moonlighter\Moonlighter_Data\Managed\Assembly-CSharp.dll`

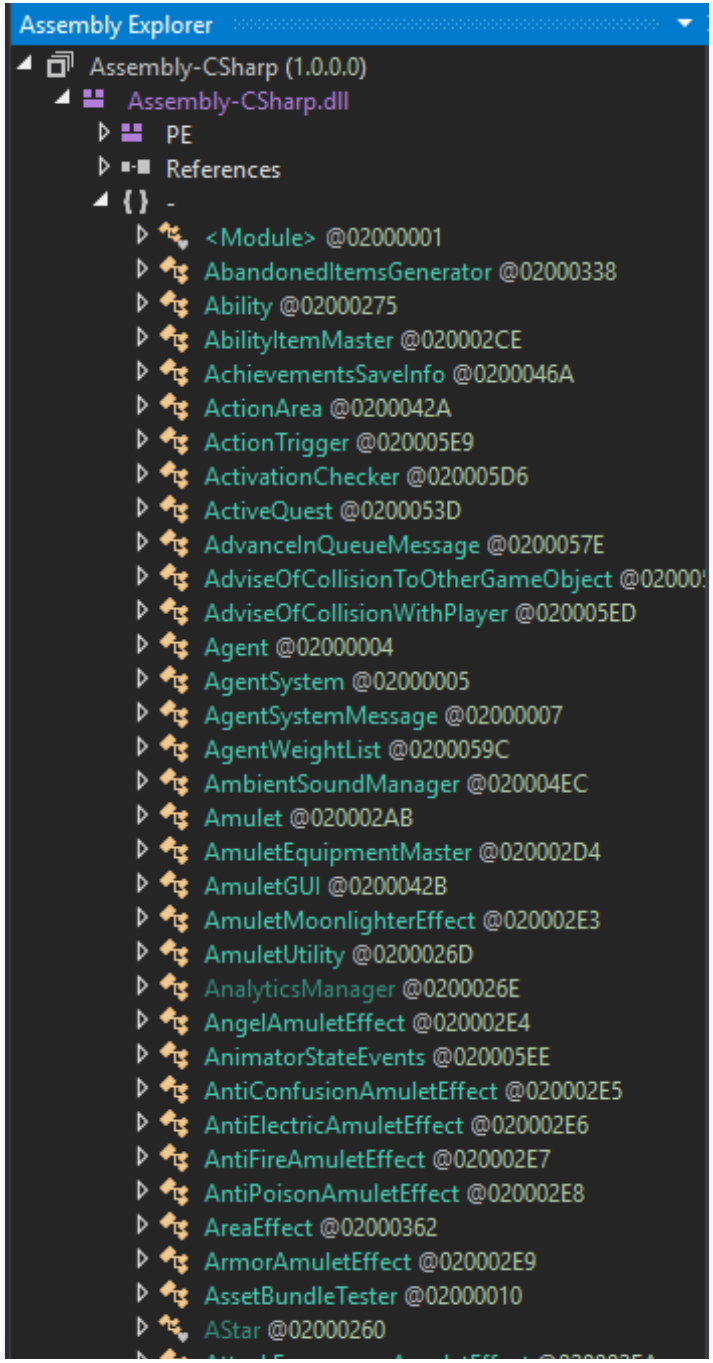
I was not successful in debugging the game with dnSpy. But the instructions are here:

- https://github.com/0xd4d/dnSpy/wiki/Debugging-Unity-Games

This is my first unity game so I might be doing something wrong or it does not work with Steam versions.

Increasing Will's Damage

Game logic is inside `{}`:



Moonlighter's classes

Going around the list, I saw the `Bow` class and clicked on it.

```
BossCaveRoom @0200028D
BossRoomMiner @0200028F
BoundingBox2D @020005F1
Bow @0200041E
    Base Type and Interfaces
    Derived Types
    .ctor() : void @06001CBE
    CreateAndLaunchMainAttackProjectile() : G
    CreateAndLaunchProjectile(GameObject, fl
    CreateAndLaunchSecondaryAttackProjectil
    DealSecondaryAttackDamage(GameObject)
    Init(BowWeaponMaster) : void @06001CC0
    OnEnchanted() : void @06001CC1
    OnSecondaryAttackHit(GameObject) : void
    ShootMainAttackArrow() : void @06001CC2
    ShootSecondarydAttackArrow() : void @060
    StoreCurrentHeroDirectionAndPositionToSh
    equipment : Equipment @17000282
    bowMaster : BowWeaponMaster @040015A
    originalHeroDirection : HeroMerchantContr
    originalHeroPosition : Vector3 @040015AE
    _attackArrowPrefab : GameObject @040015
    _equipment : Equipment @040015AF
    _isEnchanted : bool @040015B0
BowArrow @0200041B
BowWeaponMaster @020002D7
BreakableBoxBehaviour @02000339
BreakableDetail @0200033A
BreakableObject @0200033B
BreakableObjectBehaviour @0200033C
BreakEffectAction @02000297
BreakItemAction @02000298
BrokenGolemButlerBehaviour @020003A3
BrokenGolemTurretBehaviour @020003A4
BrokenGolemTurretProjectile @020003A5
BubbleAnimation @0200057A
BubbleAnimationEndedMessage @0200057D
Buggies @020003DC
ButtonEvents @020004A7
ButtonIconSwitcher @020005F2
ButtonPlatformIconManager @02000432
ButtonPressedChecker @020005D7
ButtonPrizeHandler @020004A8
ButtonSpriteConfig @02000430
ButtonSpriteSwitcher @020005F3
ButtonSwitchConfig @0200042E
BuyDetailsPanel @02000433
BuyTransactionCompleteMessage @0200057F
```

```
1  using System;
2  using UnityEngine;
3
4  // Token: 0x0200041E RID: 1054
5  public class Bow : MonoBehaviour
6  {
7      // Token: 0x17000282 RID: 642
8      // (get) Token: 0x06001CBF RID: 7359 RVA: 0x000D4A89 File Offset: 0x000D2E8
9      public Equipment equipment
10     {
11         get
12         {
13             if (!this._equipment)
14             {
15                 this._equipment = base.GetComponent<Equipment>();
16             }
17             return this._equipment;
18         }
19     }
20
21     // Token: 0x06001CC0 RID: 7360 RVA: 0x000D4AB0 File Offset: 0x000D2EB0
22     public void Init(BowWeaponMaster master)
23     {
24         this.bowMaster = master;
25         this._attackArrowPrefab = ItemDatabase.GetPrefab(master.arrowPrefabName
26         Equipment equipment = this.equipment;
27         equipment.onEquipmentEnchanted = (Action)Delegate.Remove(equipment.onEq
28         Equipment equipment2 = this.equipment;
29         equipment2.onEquipmentEnchanted = (Action)Delegate.Combine(equipment2.o
30     }
31
32     // Token: 0x06001CC1 RID: 7361 RVA: 0x000D4B23 File Offset: 0x000D2F23
33     public void OnEnchanted()
34     {
35         this._isEnchanted = true;
36     }
37
38     // Token: 0x06001CC2 RID: 7362 RVA: 0x000D4B2C File Offset: 0x000D2F2C
39     public void ShootMainAttackArrow()
40     {
41         this.StoreCurrentHeroDirectionAndPositionToShoot();
42         this.CreateAndLaunchMainAttackProjectile().GetComponent<WeaponCollision
43     }
44
45     // Token: 0x06001CC3 RID: 7363 RVA: 0x000D4B4C File Offset: 0x000D2F4C
46     public void ShootSecondarydAttackArrow()
47     {
48         this.StoreCurrentHeroDirectionAndPositionToShoot();
49         WeaponCollisionDetection component = this.CreateAndLaunchSecondaryAttac
50         component.SetWeapon(base.GetComponent<Weapon>());
51         component.weaponAttackType = WeaponCollisionDetection.WeaponAttackType.
52     }
53
54     // Token: 0x06001CC4 RID: 7364 RVA: 0x000D4B7E File Offset: 0x000D2F7E
55     public void StoreCurrentHeroDirectionAndPositionToShoot()
56     {
```

The Bow class

Inside, I searched for the string `damage` and I got lucky.

```
Bow
118     {
119         GameObject gameObject = this.CreateAndLaun
              false);
120         gameObject.GetComponent<BowArrow>().SetArr
              (WeaponCollisionDetection.WeaponAttackType.SECONDARYATTACK);
121         return gameObject;
122     }
123
124     // Token: 0x06001CC8 RID: 7368 RVA: 0x000D4DE0 File Offset: 0x000D31E0
125     public void OnSecondaryAttackHit(GameObject other)
126     {
127         if (other.tag == "HardEnemy" || other.tag == "SoftEnemy")
128         {
129             this.DealSecondaryAttackDamage(other);
130         }
131         else if (other.tag == "EnemyDamageReceiver")
132         {
133             EnemyDamageReceiver component = other.GetComponent<EnemyDamageReceiver>();
134             if (component)
135             {
136                 this.DealSecondaryAttackDamage(component.gameObjectToAdvise.gameObject);
137             }
138         }
139     }
140
141     // Token: 0x06001CC9 RID: 7369 RVA: 0x000D4E5C File Offset: 0x000D325C
142     protected void DealSecondaryAttackDamage(GameObject other)
143     {
144         float num = this.bowMaster.secondaryAttackDamageMultiplier;
145         if (MoonlighterEffectRegistry.GetEffect<EnergyFluxAmuletEffect>() != null)
146         {
147             num = 1f;
148         }
149         other.GetComponent<Enemy>().DealDamageToEnemy
              (HeroMerchant.Instance.heroMerchantStats.strength.Value * num, null,
              Enemy.EnemyDamageReceivedType.Normal);
150     }
151
```

Searching for "Damage" in the class

DealDamageToEnemy

`DealDamageToEnemy` sounds interesting. Let's double-click on it. We end up in the `Enemy` class.

```
Enemy ×
149                 }
150             }
151         }
152
153         // Token: 0x0600150E RID: 5390 RVA: 0x000825B0 File Offset: 0x000809B0
154         public void DealDamageToEnemy(float attackStrength, EnemyDamageReceiver damageReceiver =
              null, Enemy.EnemyDamageReceivedType enemyDamageReceivedType =
              Enemy.EnemyDamageReceivedType.Normal)
155         {
156             this.hitStrength = attackStrength;
157             this.otherDefense = this.enemyStats.defense;
158             this.totalDamage = this.CalcHitDamage(this.hitStrength, this.otherDefense);
159             this.beforeAttackHealth = this.enemyStats.CurrentHealth;
160             if (this.beforeAttackHealth > 0f)
161             {
162                 bool flag = this.enemyBehaviour.IsEnemyPlayingDamagedEffect();
163                 bool invencible = this.enemyStats.Invencible;
164                 this.enemyStats.CurrentHealth -= this.totalDamage;
165                 if (Enemy.OnEnemyRecivesDamage != null)
166                 {
167                     Enemy.OnEnemyRecivesDamage(this, this.totalDamage, enemyDamageReceivedType);
168                 }
169                 if (enemyDamageReceivedType == Enemy.EnemyDamageReceivedType.Effect && !
                      invencible)
170                 {
171                     this.enemyStats.Invencible = false;
172                 }
173                 this.dealedDamage = Mathf.Abs(this.enemyStats.CurrentHealth -
                      this.beforeAttackHealth);
174                 if (enemyDamageReceivedType != Enemy.EnemyDamageReceivedType.Effect && !
                      string.IsNullOrEmpty(this.enemyBehaviour.soundWhenHit))
175                 {
176                     if (this.res != null)
177                     {
178                         this.res.ActingVariation.Stop(false, false);
179                     }
180                     this.res = MasterAudio.PlaySound(this.enemyBehaviour.soundWhenHit, 1f, null,
                          0f, null, false, false);
```

```
181                 if (this.res != null)
182                 {
183                     SoundGroupVariation actingVariation = this.res.ActingVariation;
184                     if (actingVariation.IsPlaying)
185                     {
186                         Enemy.enemyPlayingSound = this;
187                         actingVariation.SoundFinished += delegate()
188                         {
189                             Enemy.enemyPlayingSound = null;
190                         };
191                     }
192                 }
193             }
194             if (!flag)
195             {
196                 EnemyHealthBar componentInChildren =
                      base.GetComponentInChildren<EnemyHealthBar>();
197                 Vector3 vector = (!componentInChildren) ? base.transform.position :
```

100 %

DealDamageToEnemy

We can analyze this a bit. `attackStrength` and enemy defense are used to calculate the damage using `CalcHitDamage`:

- `this.totalDamage = this.CalcHitDamage(this.hitStrength, this.otherDefense);`

Then the damage is applied:

- `this.enemyStats.CurrentHealth -= this.totalDamage;`

Note: It doesn't matter if the enemy is invincible (`invencible` in the code) or not, the damage is still applied.

`CalcHitDamage`

Double-click on `CalcHitDamage`:

```
// Token: 0x0600150F RID: 5391 RVA: 0x00082838 File Offset: 0x00080C38
public virtual float CalcHitDamage(float hitStrength, float targetDefense)
{
    float num = (float)Mathf.RoundToInt(hitStrength * (targetDefense / 100f));
    return Mathf.Clamp(hitStrength - num, 0f, float.PositiveInfinity);
}
```

This code calculates the target's resistance and deducts it from `hitStrength`.

Increasing Will's Damage

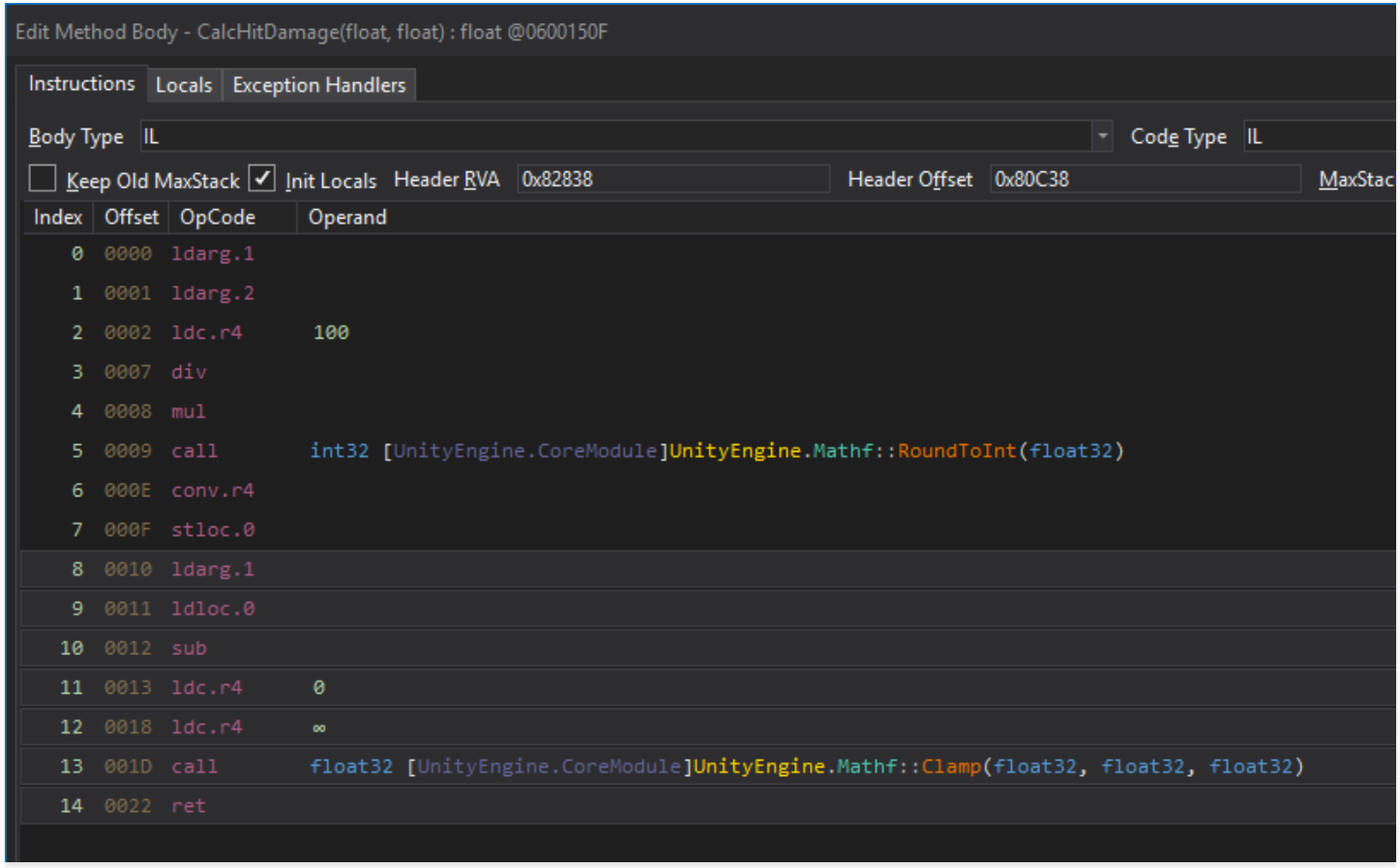
We don't know the value of damage numbers and the hitpoints of enemies yet. Let's brainstorm a bit:

1. Return `float.PositiveInfinity`. This might result in an integer underflow. I do not know to be honest but we will definitely try.
2. Return `hitStrength + num` instead. This will definitely increase our damage but will it be enough to kill enemies in one hit?
3. Multiply the output by a constant.
4. Change the lower band of Mathf.Clamp to a large number (e.g. 10000f).

Returning float.PositiveInfinity

Let's try this one and see what happens.

Right-click on the `return` line and select `Edit IL Instructions...`.



CalcHitDamage's IL instructions

IL is a stack-based language. Values are pushed to the stack before functions or operators are called.

Look at lines 13 and 14. Line 13 calls `Math.Clamp` and the next line returns it. In order to return infinity, we need to add another instruction before the `return` and copy line 12 to it (pushes infinity to the stack).

1. Click on `12` to select that line.
2. `Ctrl+C` to copy
3. Click on `13` and `Ctrl+V` to paste.
4. Press `Ok`.

```
// Token: 0x0600150F RID: 5391 RVA: 0x00082838 File Offset: 0x00080C38
public virtual float CalcHitDamage(float hitStrength, float targetDefense)
{
    float num = (float)Mathf.RoundToInt(hitStrength * (targetDefense / 100f));
    Mathf.Clamp(hitStrength - num, 0f, float.PositiveInfinity);
    return float.PositiveInfinity;
}
```

Modified CalcHitDamage

Save the module, overwrite the original DLL with the modified one and start the game.

No damage

Our evil plan was foiled.

Return hitStrength + num

Grab a fresh copy and edit IL instructions again. This time we need to change the `sub` instruction in line 10 to `add`. Click on `sub` and dnSpy shows a helpful drop-down menu of all valid instructions. Choose `add`.

Edit Method Body - CalcHitDamage(float, float) : float @060016B3

Instructions | Locals | Exception Handlers

Body Type: IL Code Type: IL

☐ Keep Old MaxStack ☑ Init Locals Header RVA: 0xA9CB0 Header Offset: 0xA7EB0 Max

Index	Offset	OpCode	Operand
0	0000	ldarg.1	
1	0001	ldarg.2	
2	0002	ldc.r4	100
3	0007	div	
4	0008	mul	
5	0009	call	int32 [UnityEngine.CoreModule]UnityEngine.Mathf::RoundToInt(float32)
6	000E	conv.r4	
7	000F	stloc.0	
8	0010	ldarg.1	
9	0011	ldloc.0	
10	0012	sub	
11	0013		
12	0018		
13	001D		at32 [UnityEngine.CoreModule]UnityEngine.Mathf::Clamp(float32, float32, float32)
14	0022		

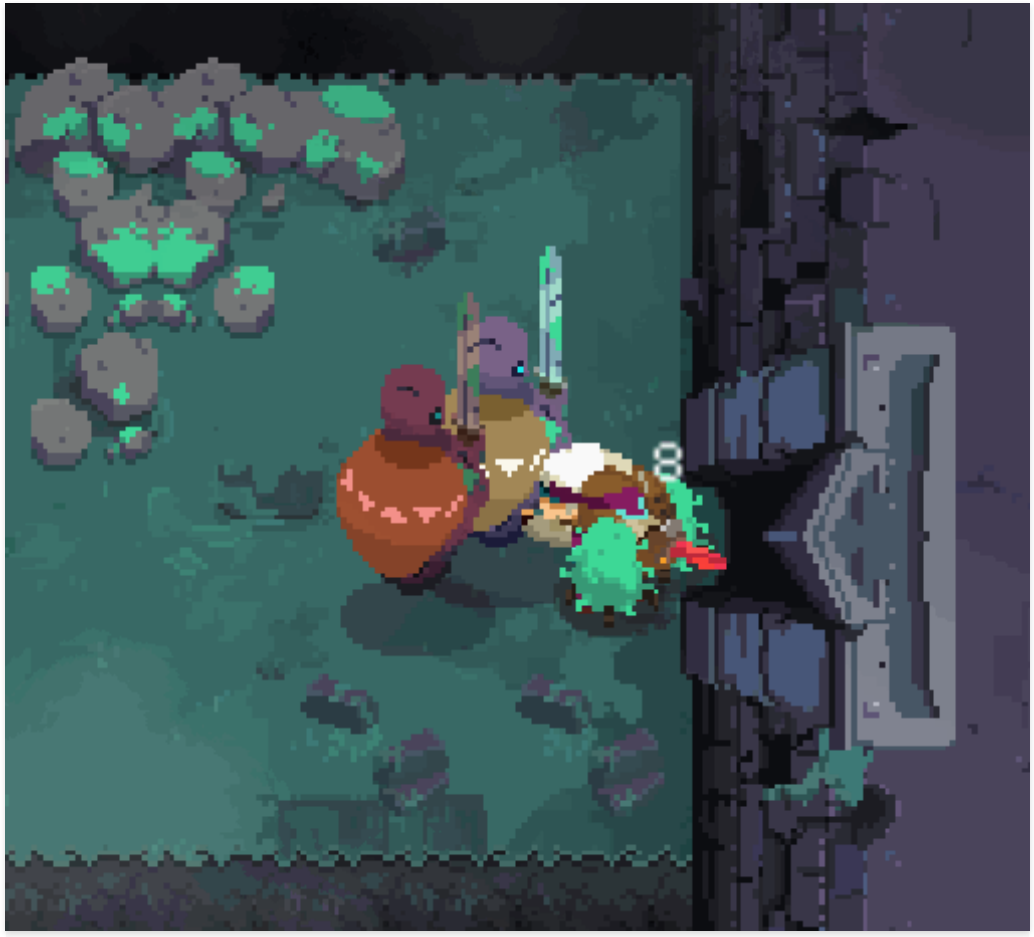
add
add.ovf
add.ovf.un
and
arglist
beq
beq.s
bge
bge.s
bge.un
bge.un.s
bgt
bgt.s
bgt.un
bgt.un.s
ble
ble.s
ble.un

Changing sub to add

```
// Token: 0x060016B3 RID: 5811 RVA: 0x000A9CB0 File Offset: 0x000A7EB0
public virtual float CalcHitDamage(float hitStrength, float targetDefense)
{
    float num = (float)Mathf.RoundToInt(hitStrength * (targetDefense / 100f));
    return Mathf.Clamp(hitStrength + num, 0f, float.PositiveInfinity);
}
```

Sub changed to add

This is better. We are one-shotting enemies. Our damage is a constant `436` with `King Sword` from part 1 regardless of enemy type.



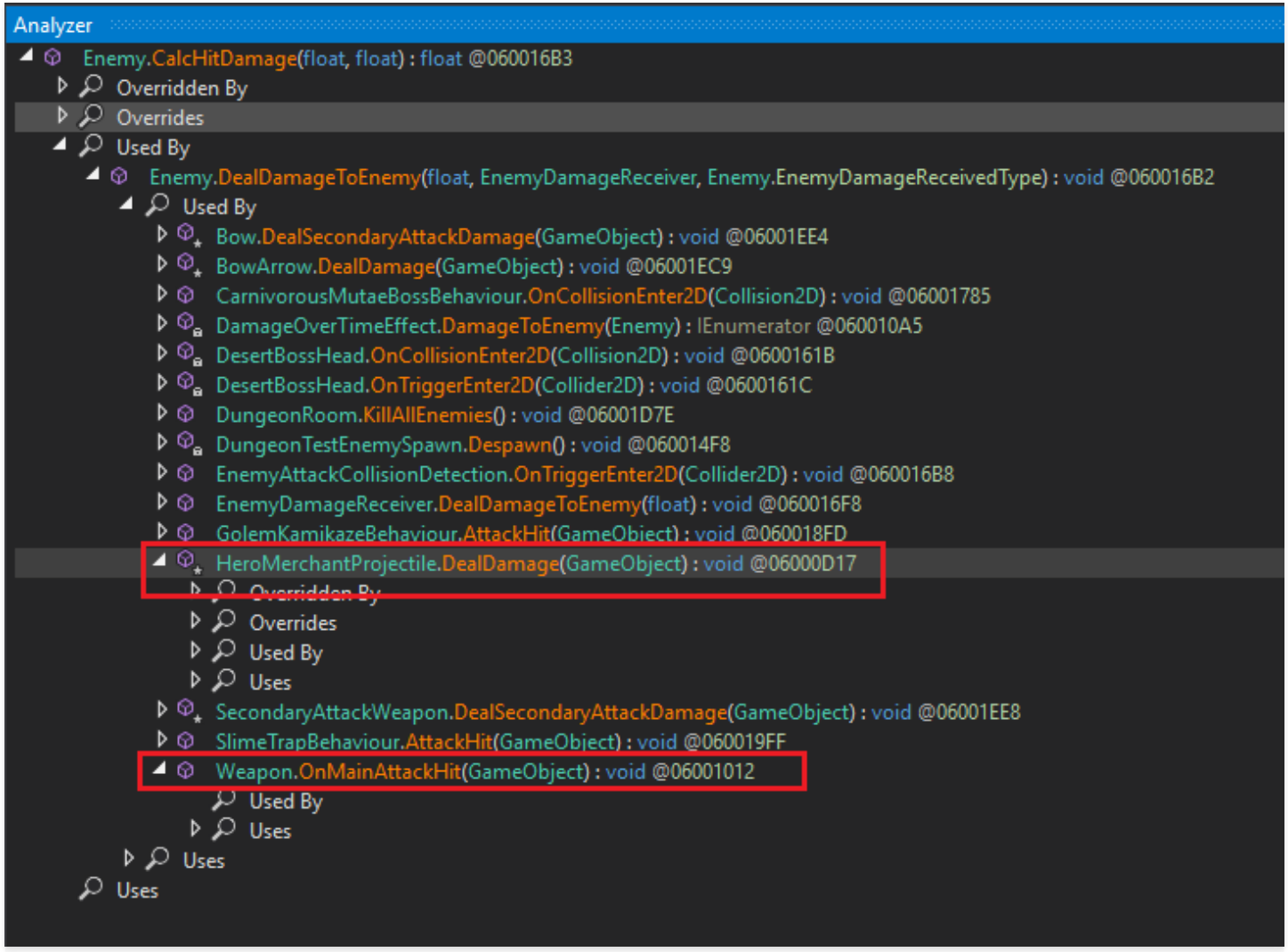
Doing constant damage

We have accomplished our goal of increasing Will's damage. But you can try the other methods or fiddle with the method in any way you want. Experiment!

Modifying Will's Stats

Player stats are important. They are used to calculate damage. Remember `attackStrength` or `hitStrength` in the previous section? They should come from somewhere based on our weapon. Let's track them.

Right-click on `CalcHitDamage` and select `Analyze`. A new window opens up. It shows who calls the target method (`Used By` which is similar to x-ref in IDA) and what the target method calls and other information.



Analyzing CalcHitDamage

Two functions look promising:

- `HeroMerchantProjectile.DealDamage(GameObject)`

- `Weapon.OnMainAttackHit(GameObject)`

HeroMerchantProjectile.DealDamage

Let's start with `HeroMerchantProjectile.DealDamage`.

```
// Token: 0x06000D17 RID: 3351 RVA: 0x0007AE24 File Offset: 0x00079024
protected virtual void DealDamage(GameObject other)
{
    if (other.gameObject.tag == "SoftEnemy" || other.gameObject.tag == "HardEnemy" ||
      other.tag == "EnemyDamageReceiver")
    {
        if (other.tag == "EnemyDamageReceiver")
        {
            other.GetComponent<EnemyDamageReceiver>().DealDamageToEnemy
              (this.projectileOwner.heroMerchantStats.intelligence.Value);
        }
        else
        {
            other.GetComponent<Enemy>().DealDamageToEnemy
              (this.projectileOwner.heroMerchantStats.intelligence.Value, null,
              Enemy.EnemyDamageReceivedType.Normal);
        }
    }
}
```

HeroMerchantProjectile.DeadlDamage

We can see that the `intelligence` stat is used to calculate bow damage.

On a side note, clicking on `Value` opens an object called `ObscuredFloat` in the `Stat` class. I vaguely remember reading about this obscured values in Unity on some Cheat Engine forum threads. It's something we might return and look at again when we are dealing with Cheat Engine. Apparently, they are hard to track in memory.

```
Stat ×
19      // Token: 0x17000233 RID: 563
20      // (get) Token: 0x06001148 RID: 4424 RVA: 0x0000BC15 File Offset:
21      // (set) Token: 0x06001149 RID: 4425 RVA: 0x000887F4 File Offset:
22      public ObscuredFloat Value
23      {
24          get
25          {
26              return this._value;
27          }
28          set
29          {
30              if (this.variableStat)
31              {
32                  if (value > this.maxValue)
33                  {
34                      if (this.OnStateUpdated != null)
35                      {
36                          this.OnStateUpdated(this.maxValue);
37                      }
38                      this._value = this.maxValue;
39                  }
40                  else if (value < this.minValue)
41                  {
42                      if (this.OnStateUpdated != null)
43                      {
44                          this.OnStateUpdated(this.minValue);
45                      }
46                      this._value = this.minValue;
47                  }
48                  else
49                  {
50                      if (this.OnStateUpdated != null)
51                      {
```

ObscuredFloat

The Case of the Missing Intelligence

There is no intelligence stat in the game. This is a picture from part 1 that show's Will's inventory. There's no intelligence stat. It shows `Vitality`, `Strength`, `Defence` and `Speed`. Is the empty green space supposed to be the intelligence?

Will's stats - no intelligence here

At first, I thought it's missing in the PC version. I looked at screenshots of the Nintendo Switch version and they looked the same.

Items do not grant intelligence either. This picture shows an item's stats in the blacksmith's UI.

Item stats - no intellience here either

In dnSpy, right-click on `intelligence` and select `Analyze`.

```
HeroMerchantStats.intelligence : Stat @0400158D
  Assigned By
    HeroMerchantStats.Init() : void @06001CC9
  Read By
    EquipmentStats.AddToHeroMerchant(HeroMerchantStats) : void @06000F59
    EquipmentStats.SubtractToHeroMerchant(HeroMerchantStats) : void @06000F5A
    HeroInventoryPanel.UpdateLabels() : void @06002710
    HeroMerchantProjectile.DealDamage(GameObject) : void @06000D17
```

Intelligence analysis

We can see it's set in `HeroInventoryPanel.UpdateLabels()`:

```
// Token: 0x06002710 RID: 10000 RVA: 0x0010CE3C File Offset: 0x0010B03C
private void UpdateLabels()
{
    string text = "#.";
    this.labelVitality.text = HeroMerchant.Instance.heroMerchantStats.currentHealth.maxValue.ToString(text);
    this.labelStrength.text = HeroMerchant.Instance.heroMerchantStats.strength.Value.ToString(text);
    this.labelDefense.text = HeroMerchant.Instance.heroMerchantStats.defense.Value.ToString(text);
    this.labelSpeed.text = HeroMerchant.Instance.heroMerchantStats.speed.Value.ToString(text);
    this.labelIntelligence.text = HeroMerchant.Instance.heroMerchantStats.intelligence.Value.ToString(text);
    HeroMerchantInventory heroMerchantInventory = HeroMerchant.Instance.heroMerchantInventory;
    Weapon weaponSetWeapon = heroMerchantInventory.GetWeaponSetWeapon(heroMerchantInventory.GetEquippedWeaponSet());
    this.imageCombatEffect.enabled = weaponSetWeapon;
    if (this.imageCombatEffect.enabled)
    {
        ItemMaster weaponMaster = weaponSetWeapon.weaponMaster;
        this.imageCombatEffect.enabled = (weaponMaster is WeaponEquipmentMaster);
        if (this.imageCombatEffect.enabled)
        {
            string combatEffect = (weaponMaster as WeaponEquipmentMaster).combatEffect;
            this.imageCombatEffect.enabled = !string.IsNullOrEmpty(combatEffect);
            if (this.imageCombatEffect.enabled)
            {
                this.imageCombatEffect.sprite = PrefabRegister.Instance.GetPrefabByName(combatEffect).GetComponent<Moo
```

HeroInventoryPanel.UpdateLabels

It's updated along with other stats but does not appear in the UI. This is not good because it's an important stat.

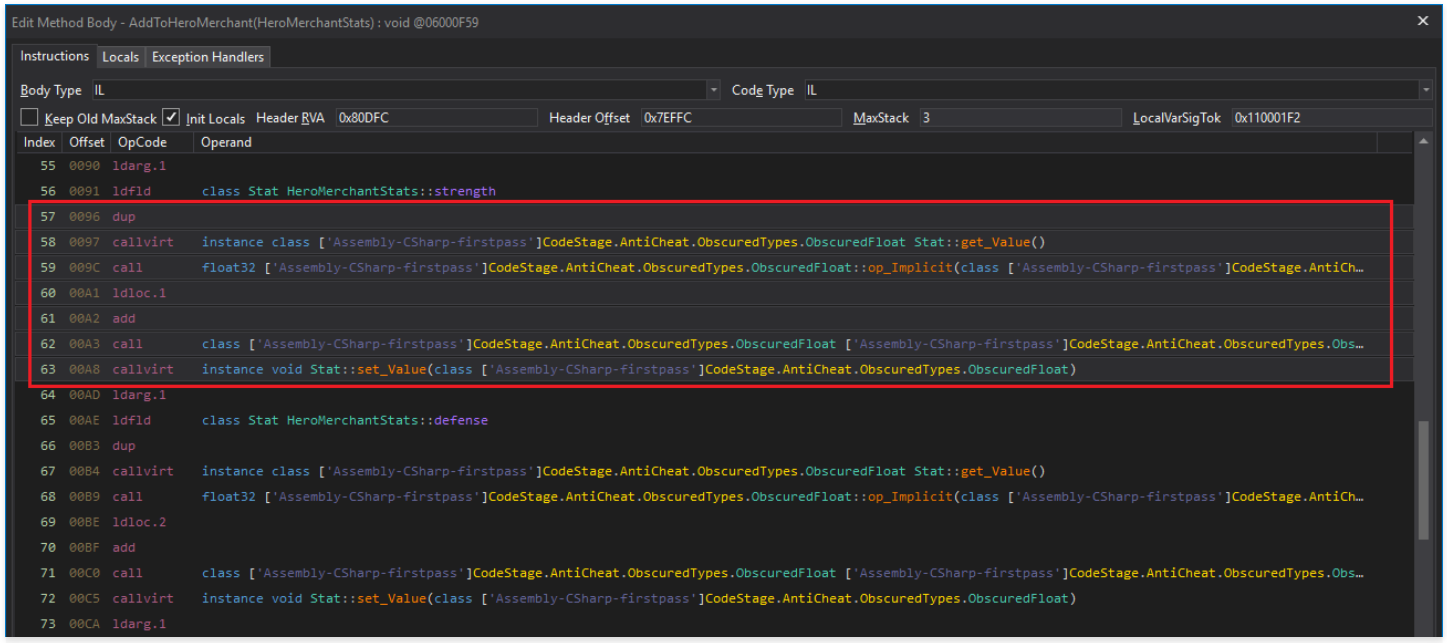
Adding Extra Stats

Look inside `EquipmentStats.AddToHeroMerchant(HeroMerchantStats)`.

```
43
44      // Token: 0x06000F59 RID: 3929 RVA: 0x00080DFC File Offset: 0x0007EFFC
45      public void AddToHeroMerchant(HeroMerchantStats heroMerchantStats)
46      {
47          this._Added = true;
48          this._HeroMerchantStats = heroMerchantStats;
49          float num = (float)this.modificator.health - this._HealthAdded;
50          float num2 = (float)this.modificator.strength - this._StrengthAdded;
51          float num3 = (float)this.modificator.armor - this._DefenseAdded;
52          float num4 = (float)this.modificator.speed - this._SpeedAdded;
53          float num5 = (float)this.modificator.intelligence - this._IntelligenceAdded;
54          Stat currentHealth = heroMerchantStats.currentHealth;
55          currentHealth.maxValue += num;
56          Stat strength = heroMerchantStats.strength;
57          strength.Value += num2;
58          Stat defense = heroMerchantStats.defense;
59          defense.Value += num3;
60          Stat speed = heroMerchantStats.speed;
61          speed.Value += num4;
62          Stat intelligence = heroMerchantStats.intelligence;
63          intelligence.Value += num5;
64          this._HealthAdded = (float)this.modificator.health;
65          this._StrengthAdded = (float)this.modificator.strength;
66          this._DefenseAdded = (float)this.modificator.armor;
67          this._SpeedAdded = (float)this.modificator.speed;
68          this._IntelligenceAdded = (float)this.modificator.intelligence;
69      }
70
```

EquipmentStats.AddToHeroMerchant

Stats are added to the base stats. We can modify each stat and add any amount. For example, to add `10000` to strength we need to modify line 57: `strength.Value += num2;`. Right-click line 57 and select `Edit IL Instructions ...`.



Line 57 IL instructions

See those highlighted lines? Those are IL instructions for line 57 in the source code (coincidentally it also starts from line 57). dnSpy has helpfully highlighted them for us. We must add two instructions before the final `add` on line 61. One to load `10000f` and another to `add` it to the previous value.

Edit Method Body - AddToHeroMerchant(HeroMerchantStats) : void @06000F59

Instructions | Locals | Exception Handlers

Body Type IL

☐ Keep Old MaxStack ☑ Init Locals Header RVA 0x80DFC Header Offset 0x7EFFC

```
Index Offset OpCode    Operand
   51 0084   ldloc.0
   52 0085   add
   53 0086   call      class ['Assembly-CSharp-firstpass']CodeStage.AntiCheat.ObscuredType
   54 008B   callvirt  instance void Stat::set_maxValue(class ['Assembly-CSharp-firstpass
   55 0090   ldarg.1
   56 0091   ldfld     class Stat HeroMerchantStats::strength
   57 0096   dup
   58 0097   callvirt  instance class ['Assembly-CSharp-firstpass']CodeStage.AntiCheat.Obs
   59 009C   call      float32 ['Assembly-CSharp-firstpass']CodeStage.AntiCheat.ObscuredTy
   60 00A1   ldloc.1
   61 00A2   ldc.r4    10000
   62 00A7   add
   63 00A8   add
   64 00A9   call      class ['Assembly-CSharp-firstpass']CodeStage.AntiCheat.ObscuredType
   65 00AE   callvirt  instance void Stat::set_Value(class ['Assembly-CSharp-firstpass']Co
   66 00B3   ldarg.1
```

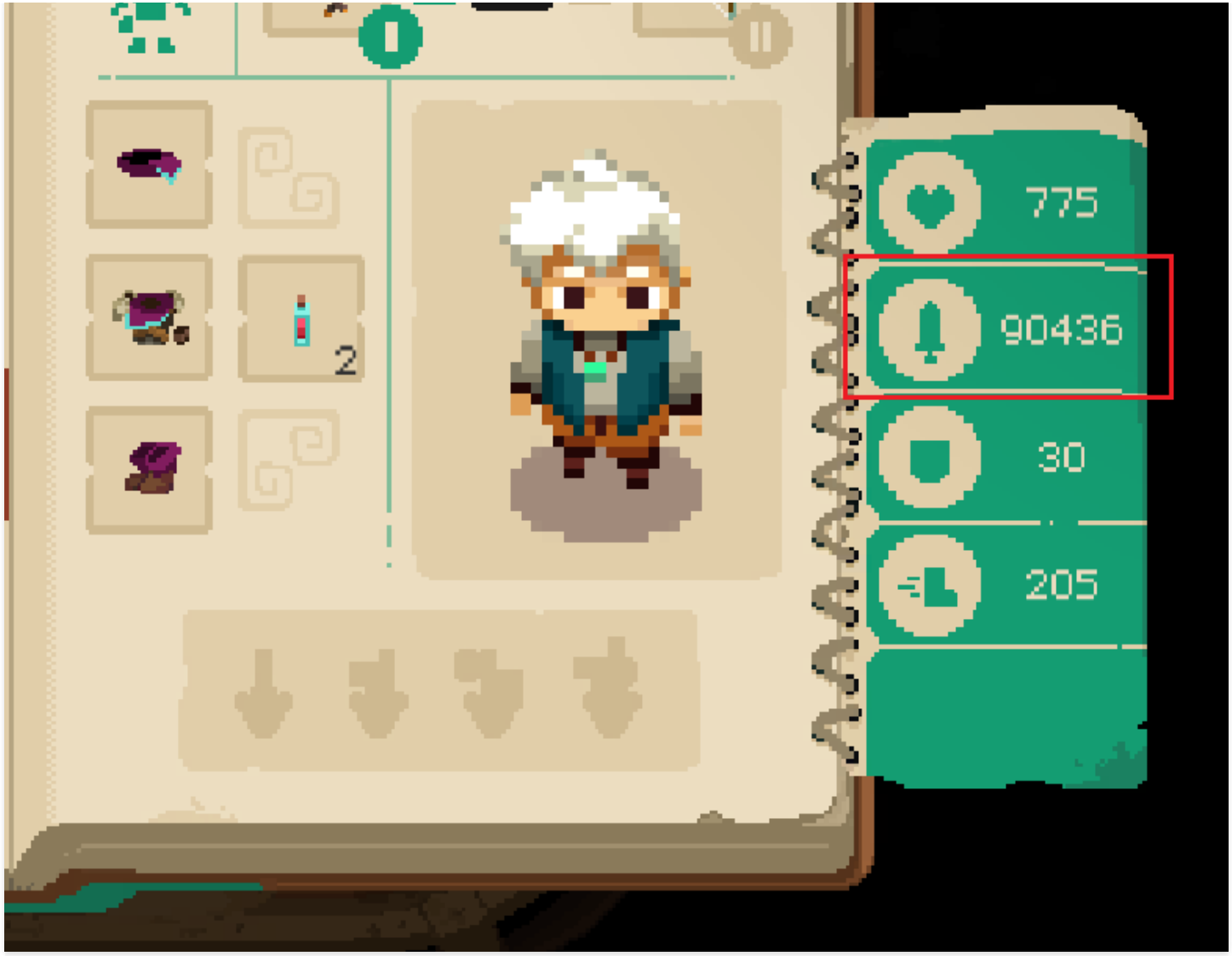
Line 57 modified

And the result in decompiled C# is:

```
43
44      // Token: 0x06000F59 RID: 3929 RVA: 0x00080DFC File Offset: 0x0007EFFC
45      public void AddToHeroMerchant(HeroMerchantStats heroMerchantStats)
46      {
47          this._Added = true;
48          this._HeroMerchantStats = heroMerchantStats;
49          float num = (float)this.modificator.health - this._HealthAdded;
50          float num2 = (float)this.modificator.strength - this._StrengthAdded;
51          float num3 = (float)this.modificator.armor - this._DefenseAdded;
52          float num4 = (float)this.modificator.speed - this._SpeedAdded;
53          float num5 = (float)this.modificator.intelligence - this._IntelligenceAdded;
54          Stat currentHealth = heroMerchantStats.currentHealth;
55          currentHealth.maxValue += num;
56          Stat strength = heroMerchantStats.strength;
57          strength.Value += num2 + 10000f;
58          Stat defense = heroMerchantStats.defense;
59          defense.Value += num3;
60          Stat speed = heroMerchantStats.speed;
61          speed.Value += num4;
62          Stat intelligence = heroMerchantStats.intelligence;
63          intelligence.Value += num5;
64          this._HealthAdded = (float)this.modificator.health;
65          this._StrengthAdded = (float)this.modificator.strength;
66          this._DefenseAdded = (float)this.modificator.armor;
67          this._SpeedAdded = (float)this.modificator.speed;
68          this._IntelligenceAdded = (float)this.modificator.intelligence;
69      }
```

Line 57 modified in C#

Now Will has `90436` strength:



Strong Will

Why did Will's strength increase by `90000`? My guess is that each equipped item calls `AddToHeroMerchant` individually. We have nine items (remember there were nine items in the

`willEquippedItems` array in the save file in part 1?). Will does `90436` damage now.

Much strong, very damage, wow

We could easily do the same and modify any other stat.

A Closer Look at Base Stats

Back in the analysis result for `HeroMerchantStats.Intelligence` we can see it's modified inside `HeroMerchantStats.Init()`:

```
// Token: 0x06001CC9 RID: 7369 RVA: 0x000CE4CC File Offset: 0x000CC6CC
public void Init()
{
    this.gold = new Stat(Constants.GetFloat("kMaxGold"), Constants.GetFloat("kMinGold"), Constants.GetFloat("kInitGold"));
    this.fullHealth = new Stat(Constants.GetFloat("kMaxHealth"), Constants.GetFloat("kMinHealth"), Constants.GetFloat("kBaseHealth"));
    this.currentHealth = new Stat(Constants.GetFloat("kMaxHealth"), Constants.GetFloat("kMinHealth"), Constants.GetFloat("kBaseHealth"));
    this.speed = new Stat(Constants.GetFloat("kMaxSpeed"), Constants.GetFloat("kMinSpeed"), Constants.GetFloat("kBaseSpeed"));
    this.intelligence = new Stat(Constants.GetFloat("kMaxIntelligence"), Constants.GetFloat("kMinIntelligence"), Constants.GetFloat("kBaseIntelligence"));
    this.strength = new Stat(Constants.GetFloat("kMaxStrength"), Constants.GetFloat("kMinStrength"), Constants.GetFloat("kBaseStrength"));
    this.defense = new Stat(Constants.GetFloat("kMaxDefense"), Constants.GetFloat("kMinDefense"), Constants.GetFloat("kBaseDefense"));
    this.absorption = new Stat(Constants.GetFloat("kMaxAbsorption"), Constants.GetFloat("kMinAbsorption"), Constants.GetFloat("kBaseAbsorption"));
    this.shield = new Stat(Constants.GetFloat("kMaxShield"), Constants.GetFloat("kMinShield"), Constants.GetFloat("kBaseShield"));
    this.currentHealth.OnStateUpdated += this.HealthValueChanged;
    this.isDead = false;
    this.gold.Value = (float)GameManager.Instance.currentGameSlot.willGold;
    GUIManager.Instance.HUDSPanel.GetComponent<HUDManager>().UpdateGoldQuantity(0, this.gold.Value, true);
    this.gold.OnStateUpdated += this.OnGoldStateUpdated;
}
```

HeroMerchantStats.Init

```
this.intelligence = new Stat(
    Constants.GetFloat("kMaxIntelligence"),
    Constants.GetFloat("kMinIntelligence"),
    Constants.GetFloat("kBaseIntelligence")
);
```

This line creates a new character stat named `intelligence`. Then sets the maximum, minimum and base values. Let's see where these default values are set. Double-Click on `Constants.GetFloat` to go there:

```
// Token: 0x060031A5 RID: 12709 RVA: 0x00143780 File Offset: 0x00141980
public static float GetFloat(string key)
{
    if (!Constants.CheckKey(key))
    {
        return 0f;
    }
    fsData fsData = Constants.GetValue(key);
    if (fsData.IsInt64)
    {
        fsData = new fsData((double)fsData.AsInt64);
    }
    else if (!fsData.IsDouble)
    {
        return 0f;
    }
    return (float)fsData.AsDouble;
}
```

Constants.GetFloat

A little bit further up in the same file, we can see how these constants are obtained.

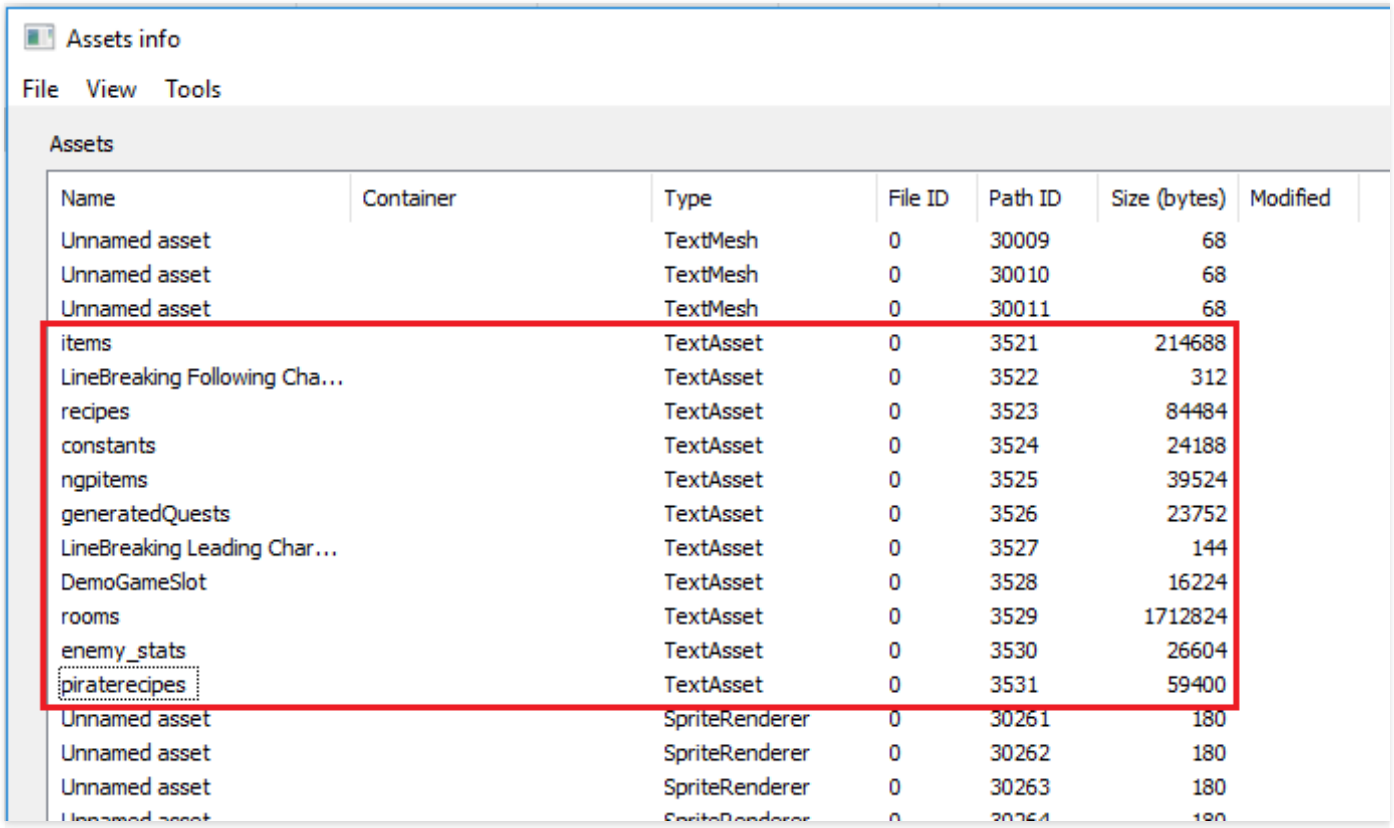
```
32      // Token: 0x060031A0 RID: 12704 RVA: 0x000024E7 File Offset: 0x000006E7
33      private void Init()
34      {
35      }
36
37      // Token: 0x060031A1 RID: 12705 RVA: 0x000200DD File Offset: 0x0001E2DD
38      public static void ReadDefaultFile()
39      {
40          Constants.ReadFile("constants");
41      }
42
43      // Token: 0x060031A2 RID: 12706 RVA: 0x001436E0 File Offset: 0x001418E0
44      public static void ReadFile(string file)
45      {
46          FsJSONManager.RawDeserialize(Resources.Load<TextAsset>(file).text, out Constants.Instance._values);
47          Debug.Log("CONSTANTS - ReadFile - Values.Count = " + Constants.Instance._values.AsDictionary.Count);
48      }
49
50      // Token: 0x060031A3 RID: 12707 RVA: 0x00143730 File Offset: 0x00141930
51      private static bool CheckKey(string key)
52      {
53          return Constants.Instance._values != null && Constants.Instance._values.IsDictionary && Constants.Instan
54      }
55
56      // Token: 0x060031A4 RID: 12708 RVA: 0x000200E9 File Offset: 0x0001E2E9
57      private static fsData GetValue(string key)
58      {
59          return Constants.Instance._values.AsDictionary[key];
60      }
61
62      // Token: 0x060031A5 RID: 12709 RVA: 0x00143780 File Offset: 0x00141980
63      public static float GetFloat(string key)
64      {
65          if (!Constants.CheckKey(key))
66          {
67              return 0f;
68          }
```

Constants.ReadFile

They are read from a JSON file named `constants`. If we run a recursive grep for "constants" in the `Moonlighter_Data` directory, we find a few files. We need to open `resources.assets`. Either use a tool to extract it or open it with a hex editor (e.g. HxD) and search for the string `constants`.

I used [Unity Assets Bundle Extractor](). I needed to install [Microsoft Visual C++ 2010 Redistributable Package (x64)]() before running it.

Sort by `Type` and look for files with the `TextAsset` type.



Name	Container	Type	File ID	Path ID	Size (bytes)	Modified
Unnamed asset		TextMesh	0	30009	68	
Unnamed asset		TextMesh	0	30010	68	
Unnamed asset		TextMesh	0	30011	68	
items		TextAsset	0	3521	214688	
LineBreaking Following Cha...		TextAsset	0	3522	312	
recipes		TextAsset	0	3523	84484	
constants		TextAsset	0	3524	24188	
ngpitems		TextAsset	0	3525	39524	
generatedQuests		TextAsset	0	3526	23752	
LineBreaking Leading Char...		TextAsset	0	3527	144	
DemoGameSlot		TextAsset	0	3528	16224	
rooms		TextAsset	0	3529	1712824	
enemy_stats		TextAsset	0	3530	26604	
piraterecipes		TextAsset	0	3531	59400	
Unnamed asset		SpriteRenderer	0	30261	180	
Unnamed asset		SpriteRenderer	0	30262	180	
Unnamed asset		SpriteRenderer	0	30263	180	

TextAssets

We can dump each file. The base stats are inside the `constants` dump:

```
"kBaseHealth": 100,
```

```
"kMaxHealth": 100,
"kMinHealth": 0,
"kBaseSpeed": 185,
"kMaxSpeed": 250,
"kMinSpeed": 80,
"kBaseIntelligence": 20,
"kMaxIntelligence": 100,
"kMinIntelligence": 10,
"kBaseStrength": 5,
"kMaxStrength": 99999,
"kMinStrength": 5,
"kBaseDefense": 10,
"kMaxDefense": 50,
"kMinDefense": 10,
"kMaxkarma": 1,
"kInitKarma": 0,
"kMinkarma": -1,
"kMaxGold": 999999999,
"kInitGold": 200,
"kMinGold": 0,
"kBaseAbsorption": 0,
"kMaxAbsorption": 0,
"kMinAbsorption": 0,
"kBaseShield": 0,
"kMaxShield": 4,
"kMinShield": 0,

"goldBagSpriteRanges": [ 4000, 16000, 64000, 256000, 512000, 1024000 ],
"bagSize": 20,
"pocketSize": 5,
```

```
"WeaponPushForce": 900,

"willShopModeSpeedFactor": 0.7,
"willDungeonModeSpeedFactor": 1.0,
```

Base stats

There's more stuff here. For example, item drop probabilities.

Other files here contain other things such as items (we can get a list of all items), recipes, and enemy stats. By editing these files, we can change enemy stats, items stats, recipes, and more.

Lesssons Learned

We learned:

- How to edit game logic for Unity games.
- How to use dnSpy's analysis feature.
- Edit IL instructions to increase Will's damage and stats.
- Discovered a hidden stat called Intelligence that does not appear in the game's UI.

I saw some hidden features in the decompiled DLL. In the next part, I will try to enable them.

Posted by Parsia • Jan 27, 2019 • Tags: Moonlighter dnSpy

Cheating at Moonlighter - Part 1 - Save File

Cheating at Moonlighter - Part 3 - Enabling Debug HUD

0 Comments Parsiya

Login

Recommend Tweet Share

Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

Subscribe Add Disqus to your site Disqus' Privacy Policy

DISQUS