《清华骑士》测试文档

争取不熬夜的软工小队

（谷海桥、李世蛟、卢志永、丁思勤、尧廷松、牛艺钧、尹依晨、介可得、冯思淼）

白盒测试员、文档编写者：丁思勤

目录

1 测试安排

根据课程要求，我们利用黑盒测试和白盒测试的方法对《清华骑士》进行了单元测试和集成测试。本部分简要说明测试安排。

白盒测试目的是全面检查程序内部逻辑结构，要求对所有逻辑路径进行测试。单元测试用于检验被测代码的一个很小的、很明确的功能是否正确。我们将白盒测试和单元测试结合，用于测试各脚本中关键功能函数的正确性。

黑盒测试不考虑内部逻辑结构，是以用户的角度，从输入数据与输出数据的对应关系出发进行测试的，主要看程序是否能适当地接收输入数据而产生正确的输出信息。集成测试是在单元测试的基础上，将所有模块按照设计要求组装成为子系统或系统进行的测试，主要检查各模块能否正确协同工作。我们将黑盒测试和集成测试结合，用于测试游戏界面与游戏流程。

按照最初的设计，《清华骑士》分为“人物”、“怪物”、“道具”、“地图”、“UI”五大功能模块进行开发，在测试中我们有所调整，道具部分细分为“可消耗道具”、“武器道具”、“Buff 道具”，并且“UI”部分需要依附于另外几个部分进行测试。根据不同的模块的特点，选用的测试方法不同，具体见“测试内容”的安排。

2 测试内容

人物部分有较多的逻辑控制代码，白盒测试和黑盒测试并重。进行白盒测试部分有人物加载测试、人物血量控制测试、背包测试、武器栏测试、Buff 栏测试、闪现测试。进行黑盒部分测试有移动测试、动画测试、血条 UI 测试、体力条 UI 测试、攻击测试。

怪物部分和人物部分类似，也需要白盒测试和黑盒测试并重，且部分功能需要分为普通怪物和 Boss 分别测试。进行白盒测试部分有怪物加载测试、目标获取测试、攻击方向获取测试、追逐测试、怪物血量控制测试、死亡测试、技能测试。进行黑盒测试的部分有移动测试、血条 UI 测试、攻击测试。

道具测试部分主要是测试道具的消耗和使用效果，以白盒测试为主，分为道具生成测试、血药测试、力量药测试、Buff 道具测试。黑盒部分主要是道具捡起测试、道具 UI 加载

及变化测试。

武器道具由于其特殊性，需要独立测试。白盒测试部分包括武器加载测试、近战攻击测试、远程攻击测试。黑盒部分包括瞄准测试、位置变化测试、切换测试。

地图测试以黑盒测试为主，分为地形生成测试、场景传送测试、小地图 UI 测试。

各部分测试内容汇总如下表：

功能模块	测试内容	测试方法
人物	人物加载测试	白盒+单元测试
	人物血量控制测试	
	背包测试	
	武器栏测试	
	Buff 栏测试	
	闪现测试	
	移动测试	黑盒+集成测试
	动画测试	
	血条 UI 测试	
	体力条 UI 测试	
	攻击测试	
怪物	怪物加载测试	白盒+单元测试
	目标获取测试	
	攻击方向获取测试	
	追逐测试	
	怪物血量控制测试	
	死亡测试	
	技能测试	
	移动测试	黑盒+集成测试
	血条 UI 测试	
	攻击测试	
道具	道具生成测试	白盒+单元测试
	血药测试	

	力量药测试	
	Buff 道具测试	
	道具捡起测试	黑盒+集成测试
	道具 UI 加载及变化测试	
武器	武器加载测试	白盒+单元测试
	近战攻击测试	
	远程攻击测试	
	瞄准测试	黑盒+集成测试
	位置变化测试	
	切换测试	
地图	地形生成测试	黑盒+集成测试
	场景传送测试	
	小地图 UI 测试	

表 1 测试内容及方法

3 测试环境

测试在 Windows 10 中的 Unity 2020.1.9f1c1 平台上进行，白盒测试利用其 Test Runner 中 Edit Mode 功能，黑盒测试利用其 Play 功能。

4 测试详情

4.1 人物

在已经准备好的 Test Scene 中放入待测试的 MahouPrefabs，测试脚本[SetUp]部分中写上加载游戏资源并初始化的代码如下：

```
[SetUp]
0 个引用
public void SetUp()
{
    LogAssert.ignoreFailingMessages = true;

    MahouAgent = GameObject.Find("MahouPrefabs").GetComponent<CharacterAgent>();
    var initialGame = new Initialization();
    initialGame.Awake();
    MahouAgent.Awake();

    var uiHealthBar = new UIHealthBar();
    uiHealthBar.Awake();

    var uiDashBar = new UIDashBar();
    uiDashBar.Awake();
    Debug.Log("Set up.");
}
```

4.1.1 人物加载测试

加载部分测试人物的 index 和各属性值是否正确读取。代码如下。

```
[UnityTest]
[Order(0)]
[Description("测试人物的index、各属性值是否正确加载")]
0 个引用
public IEnumerator CharacterLoadTest()
{
    LogAssert.ignoreFailingMessages = true;

    Assert.IsNotNull(MahouAgent);
    Assert.AreEqual(0, MahouAgent.characterIndex);
    Assert.AreEqual(1, MahouAgent.actualLiving.TimeInvincible);
    Assert.AreEqual(1, MahouAgent.actualLiving.AttackAmount);
    Assert.AreEqual(100f, MahouAgent.actualLiving.CurrentHealth);
    Assert.IsNotNull(MahouAgent.WeaponPrefab);
    Debug.Log(MahouAgent.WeaponPrefab);
    yield return null;

    LogAssert.ignoreFailingMessages = false;
}
```

4.1.2 人物血量控制测试

人物的血量增减由 ChangeHealth()函数控制，我们要测试血量是否符合预期，并且不超过上下限。由于我们设置了“无敌状态”，在无敌状态下无法扣血，能够加血，我们需要测试加血、扣血是否与无敌状态冲突。代码如下。

```
[UnityTest]
[Order(1)]
0 个引用
public IEnumerator CharacterHealthTest()
{
    LogAssert.ignoreFailingMessages = true;


    // 初始血量应为100，有1s的无敌
    Assert.AreEqual(100f, MahouAgent.actualLiving.CurrentHealth);
    Assert.AreEqual(MahouAgent.actualLiving.MaxHealth, MahouAgent.actualLiving.CurrentHealth);
    Assert.IsTrue(MahouAgent.actualLiving.State.HasStatus(new InvincibleState()));

    // 过1s以上，应该无敌状态消除
    var frameCount = 0;
    while ((frameCount++) <= 900)
    {
        MahouAgent.Update();
        yield return null;
    }
    frameCount = 0;
    Assert.IsFalse(MahouAgent.actualLiving.State.HasStatus(new InvincibleState()));

    // 测试伤害，扣20点血，受伤后应进入无敌
    MahouAgent.ChangeHealth(-20f);
    Assert.AreEqual(80f, MahouAgent.actualLiving.CurrentHealth);
    Assert.IsTrue(MahouAgent.actualLiving.State.HasStatus(new InvincibleState()));

    // 无敌状态能够加血，血量上限不超过最大血量
    MahouAgent.ChangeHealth(700f);
    Assert.AreEqual(100f, MahouAgent.actualLiving.CurrentHealth);
    Assert.AreEqual(MahouAgent.actualLiving.MaxHealth, MahouAgent.actualLiving.CurrentHealth);

    // 等待无敌时间解除
    while ((frameCount++) <= 900)
    {
        MahouAgent.Update();
        yield return null;
    }
    frameCount = 0;
    Assert.IsFalse(MahouAgent.actualLiving.State.HasStatus(new InvincibleState()));

    // 就算伤害值超过当前血量，血量最低也是0，不过人已经没了
    MahouAgent.ChangeHealth(-700f);
    Assert.AreEqual(0f, MahouAgent.actualLiving.CurrentHealth);
    Assert.IsTrue(MahouAgent.IsDead());

    LogAssert.ignoreFailingMessages = false;
}
```

4.1.3 背包测试

该部分主要测试背包中物品的添加、移除、使用后数量的是否符合预期。代码如下。

```
[UnityTest]
[Description("测试背包物品的添加、移除、使用")]
0 个引用
public IEnumerator InventoryTest()
{
    LogAssert.ignoreFailingMessages = true;

    // 人物初始化时，应该带有物品
    Assert.IsFalse(MahouAgent.IsInventoryEmpty());

    // 清空背包
    MahouAgent.CleanInventory();
    Assert.IsTrue(MahouAgent.IsInventoryEmpty());

    // 添加10枚硬币
    MahouAgent.InventoryAddItem(new Coin { Amount = 10 });
    Assert.IsFalse(MahouAgent.IsInventoryEmpty());
    Assert.AreEqual(10, MahouAgent.GetItemAmount(new Coin { }));

    // 添加2个血瓶
    MahouAgent.InventoryAddItem(new HealthPotion { Amount = 2 });
    Assert.AreEqual(2, MahouAgent.GetItemAmount(new HealthPotion { }));

    // 满血状态下应该不能使用血瓶，其数量不消耗
    MahouAgent.UseItem(new HealthPotion { });
    Assert.AreEqual(2, MahouAgent.GetItemAmount(new HealthPotion { }));

    // 非满血状态，应该能使用，数量减1
    MahouAgent.actualLiving.State.ClearStatus();
    MahouAgent.ChangeHealth(-1);
    MahouAgent.UseItem(new HealthPotion { });
    Assert.AreEqual(1, MahouAgent.GetItemAmount(new HealthPotion { }));

    // 清空背包
    MahouAgent.CleanInventory();
    Assert.IsTrue(MahouAgent.IsInventoryEmpty());

    yield return null;
    LogAssert.ignoreFailingMessages = false;

}
```

4.1.4 Buff 栏和武器栏测试

该部分和背包测试类似，不过使用和移除逻辑略有不同，故主要测试物品的添加。代码如下。

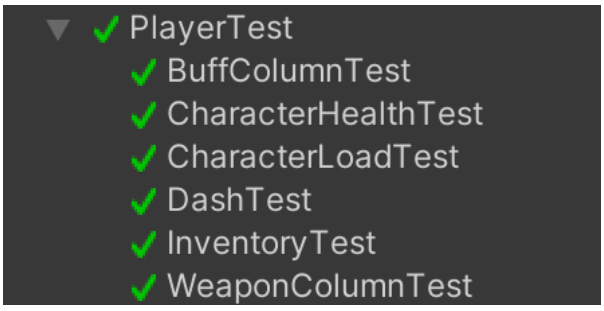
```
[UnityTest]
0 个引用
public IEnumerator BuffColumnTest()
{
    LogAssert.ignoreFailingMessages = true;

    // 测试是否能正确添加无敌状态物品
    MahouAgent.CleanBuffColumn();
    Assert.AreEqual(0, MahouAgent.GetBuffItemAmount(new InvincibleItem { }));
    MahouAgent.BuffColumnAddItem(new InvincibleItem { });
    Assert.AreEqual(1, MahouAgent.GetBuffItemAmount(new InvincibleItem { }));

    yield return null;
    LogAssert.ignoreFailingMessages = false;

}

[UnityTest]
0 个引用
public IEnumerator WeaponColumnTest()
{
    LogAssert.ignoreFailingMessages = true;

    // 初始人物手上有"咸鱼"武器，武器栏不为空
    Assert.IsTrue(MahouAgent.WeaponPrefab != null);
    Debug.Log(MahouAgent.WeaponPrefab);
    Assert.IsFalse(MahouAgent.IsWeaponColumnEmpty());

    // 清空武器栏
    MahouAgent.CleanWeaponColumn();
    Assert.IsTrue(MahouAgent.IsWeaponColumnEmpty());

    // 加入一把枪
    MahouAgent.WeaponColumnAddItem(new Gun { });
    Assert.IsFalse(MahouAgent.IsWeaponColumnEmpty());
    Assert.AreEqual(1, MahouAgent.GetWeaponAmount(new Gun { }));

    // 切换手上的武器，咸鱼应该到了武器栏中
    // 不知道为什么它会报错，故就不在这个里面测切换
    //MahouAgent.SwapeWeapon();
    //Debug.Log(MahouAgent.WeaponPrefab);

    yield return null;
    LogAssert.ignoreFailingMessages = false;
}
```

4.1.5 闪现测试

该部分测试闪现的 CD 是否能如预期回复，且回复后能否释放闪现，释放闪现后是否重置 CD。代码如下

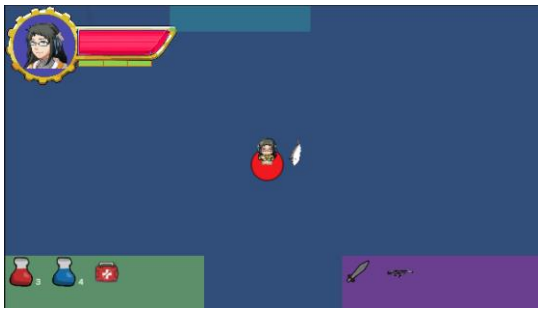
```
[UnityTest]
0 个引用
public IEnumerator DashTest()
{
    MahouAgent.ClearDashBar();
    Assert.AreEqual(0, MahouAgent.dashBar);
    var frameCount = 0;
    while ((frameCount++) <= 300)
    {
        MahouAgent.Update();
        yield return null;
    }
    frameCount = 0;

    Assert.AreEqual(301, MahouAgent.dashBar);
    MahouAgent.ForceDash();
    Assert.AreEqual(1, MahouAgent.dashBar);

    // 不知为什么Dash后位置不变
    Debug.Log(MahouAgent.rigidbody2d.position);

    yield return null;
}
```

人物部分白盒测试结果全部通过。



接下来是人物的黑盒测试部分。

4.1.6 移动及动画测试

<table>
<tr><td>  </td><td>  </td></tr>
<tr><td>初始位置</td><td>向左移动</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>向上移动</td><td>向下移动</td></tr>
</table>

WSAD 正确控制人物方向，并且人物正确随鼠标位置改变，动画也正常播放。

4.1.7 血条、体力条 UI 测试

受攻击后血量减少，按空格进行闪现后位置确实变化，并且体力值减少。体力随时间均匀回复。

4.1.8 攻击测试

鼠标左键能够正确进行攻击，上图中持枪射出子弹。

4.2 怪物

在已经准备好的 Test Scene 中放入待测试的 MahouPrefabs、“多元鸡分”和“拉格朗日”，测试脚本[SetUp]部分中写上加载游戏资源并初始化的代码如下：

```
[SetUp]
0 个引用
public void SetUp()
{
    LogAssert.ignoreFailingMessages = true;
    DuoYuanJiFenAgent = GameObject.Find("多元鸡分").GetComponent<MonsterAgent>();
    LagrangeAgent = GameObject.Find("拉格朗日").GetComponent<BossAgent>();

    var initialGame = new Initialization();
    initialGame.Awake();

    DuoYuanJiFenAgent.Start();
    LagrangeAgent.Start();
    Debug.Log("Set up.");
}
```

4.2.1 怪物加载测试

该部分测试怪物能否正确加载，其位置和部分属性值是否正确。代码如下。

```
[UnityTest]
[Order(0)]
[Description("测试怪物能否正确加载")]
0 个引用
public IEnumerator MonsterLoadTest()
{
    Assert.IsNotNull(DuoYuanJiFenAgent);
    Assert.AreEqual(2, DuoYuanJiFenAgent.monsterIndex);
    Assert.AreEqual(new Vector3(0, 2, 0), DuoYuanJiFenAgent.transform.position);
    Assert.IsNotNull(DuoYuanJiFenAgent.bulletPrefab);

    Assert.IsNotNull(LagrangeAgent);
    Assert.AreEqual(5, LagrangeAgent.monsterIndex);
    Assert.AreEqual(new Vector3(3, 4, 0), LagrangeAgent.transform.position);
    Assert.IsNotNull(LagrangeAgent.bulletPrefab);

    Debug.Log("怪物成功加载");
    yield return null;
}
```

4.2.2 目标获取、攻击方向、追逐测试

怪物的主要功能是攻击玩家，因此要能正确获取玩家位置，进行追逐和攻击。代码如下。

```
[UnityTest]
[Order(1)]
[Description("测试怪物能否正确找到目标")]
0 个引用
public IEnumerator FindCharacterTest()
{
    // 怪物一开始没有找玩家
    Assert.IsFalse(DuoYuanJiFenAgent.target != null);
    Assert.IsFalse(LagrangeAgent.target != null);

    // 经过1帧后，找到玩家
    var frameCount = 0;
    while ((frameCount++) <= 0)
    {
        DuoYuanJiFenAgent.Update();
        LagrangeAgent.Update();
        yield return null;
    }
    Assert.IsTrue(DuoYuanJiFenAgent.target != null);
    Assert.IsTrue(LagrangeAgent.target != null);
    Debug.Log("成功找到目标");

    yield return null;
}

[UnityTest]
[Order(2)]
[Description("测试能否获得正确的攻击方向")]
0 个引用
public IEnumerator AttackDirTest()
{
    var frameCount = 0;
    while ((frameCount++) <= 0)
    {
        DuoYuanJiFenAgent.Update();
        LagrangeAgent.Update();
        yield return null;
    }
    var targetPosition = DuoYuanJiFenAgent.target.transform.position;
    var monsterPosition = DuoYuanJiFenAgent.transform.position;
    var lagrangePosition = LagrangeAgent.transform.position;
    Assert.AreEqual((targetPosition - monsterPosition).normalized, DuoYuanJiFenAgent.GetAttackDirection());
    Assert.AreEqual((targetPosition - lagrangePosition).normalized, LagrangeAgent.GetAttackDirection());
}

[UnityTest]
[Order(3)]
[Description("测试怪物状态机能否变为"追逐"")]
0 个引用
public IEnumerator ChasingTest()
{
    var frameCount = 0;
    while ((frameCount++) <= 9)
    {
        DuoYuanJiFenAgent.Update();
        LagrangeAgent.Update();
        yield return null;
    }
    Assert.IsTrue(DuoYuanJiFenAgent.IsChasing());
    Assert.IsTrue(LagrangeAgent.IsChasing());

    yield return null;
}
```

4.2.3 血量和死亡测试

怪物和人物所使用的血量控制逻辑一致，代码也基本一致，不再列出。不同的是某些

怪物死亡后会分裂成小怪，有些怪物死亡后会掉落金币。测试代码如下：

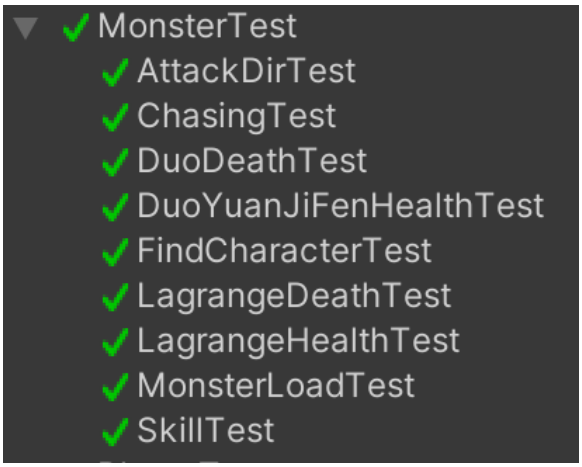
```
[UnityTest]
[Order(5)]
[Description("测试Boss血量空后会不会被清除，由于Edit Mode下无法调用Destroy，故检测它是否掉落金币")]
0 个引用
public IEnumerator LagrangeDeathTest()
{
    LogAssert.ignoreFailingMessages = true;

    // 初始地图中无金币
    Assert.IsFalse(GameObject.Find("Coin(Clone)"));

    // Boss死亡后掉落金币
    LagrangeAgent.ChangeHealth(-800f);
    Assert.AreEqual(0f, LagrangeAgent.actualLiving.CurrentHealth);
    Assert.IsTrue(LagrangeAgent.IsDead());
    Assert.IsTrue(GameObject.Find("Coin(Clone)"));

    // 恢复血量，为了通过后续测试
    LagrangeAgent.ChangeHealth(800f);

    yield return null;
    LogAssert.ignoreFailingMessages = false;
}

[UnityTest]
[Order(6)]
[Description("测试多元鸡分死亡后能否产生3个微多元鸡分")]
0 个引用
public IEnumerator DuoDeathTest()
{
    LogAssert.ignoreFailingMessages = true;

    Assert.IsFalse(GameObject.Find("微多元鸡分(Clone)"));

    DuoYuanJiFenAgent.ChangeHealth(-40f);
    Assert.AreEqual(0f, DuoYuanJiFenAgent.actualLiving.CurrentHealth);
    Assert.IsTrue(DuoYuanJiFenAgent.IsDead());

    // 找到所有的微多元鸡分，应该有3个
    var objs = new List<GameObject>();
    foreach ( GameObject go in GameObject.FindObjectsOfType(typeof(GameObject)))
    {
        if (go.name == "微多元鸡分(Clone)")
        {
            objs.Add(go);
        }
    }
    Assert.AreEqual(3, objs.Count());

    // 恢复血量，为了通过后续测试
    DuoYuanJiFenAgent.ChangeHealth(40f);

    yield return null;
    LogAssert.ignoreFailingMessages = false;
}
```

4.2.4 技能测试

和人物不同，不同怪物配有不同技能，下面的代码测试怪物是否装备预期的技能。

```
[UnityTest]
[Order(4)]
[Description("测试怪物有没有指定的skill")]
0 个引用
public IEnumerator SkillTest()
{
    // 多元鸡分有远程攻击技能、分裂技能
    var duoSkills = DuoYuanJiFenAgent.GetCurrentSkills();
    Skill splitSkill = duoSkills.FirstOrDefault(e => e is SplitSkill);
    Assert.IsNotNull(splitSkill);
    Skill duoMissleSkill = duoSkills.FirstOrDefault(e => e is MissleAttackSkill);
    Assert.IsNotNull(duoMissleSkill);

    // 拉格朗日有远程攻击、强化攻击、火球攻击、激光攻击技能
    var lagSkills = LagrangeAgent.GetCurrentSkills();
    Skill laserSkill = lagSkills.FirstOrDefault(e => e is LaserSkill);
    Assert.IsNotNull(laserSkill);
    Skill lagMissleSkill = lagSkills.FirstOrDefault(e => e is MissleAttackSkill);
    Assert.IsNotNull(lagMissleSkill);
    Skill barrageSkill = lagSkills.FirstOrDefault(e => e is BarrageSkill);
    Assert.IsNotNull(barrageSkill);
    Skill fierceSkill = lagSkills.FirstOrDefault(e => e is FierceSkill);
    Assert.IsNotNull(fierceSkill);

    yield return null;

}
```

以上是怪物的白盒测试代码，测试结果通过。

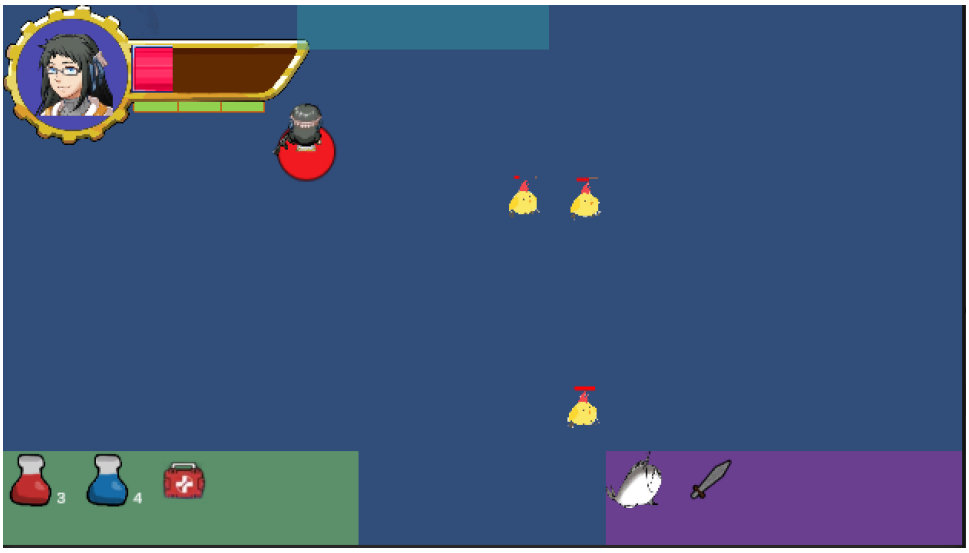


接下来是怪物的黑盒测试部分。

4.2.5 移动和攻击测试

这张图已经表现得很明显了。

4.2.6 血条 UI 测试



由上图可见，多元鸡分死亡后分裂成 3 个微多元鸡分，怪物受攻击后头顶血条减少。

4.3 道具

在已经准备好的 Test Scene 中放入待测试的 MahouPrefabs，测试脚本[SetUp]部分中写上加载游戏资源并初始化的代码如下：

```
[SetUp]
0 个引用
public void SetUp()
{
    LogAssert.ignoreFailingMessages = true;

    MahouAgent = GameObject.Find("MahouPrefabs").GetComponent<CharacterAgent>();
    var initialGame = new Initialization();
    initialGame.Awake();
    MahouAgent.Awake();

    var uiHealthBar = new UIHealthBar();
    uiHealthBar.Awake();

    var uiDashBar = new UIDashBar();
    uiDashBar.Awake();
    Debug.Log("Set up.");
}
```

4.3.1 道具生成测试

本部分测试道具生产函数 ItemAgent.GenerateItem (Vector3 position, Item item)是否能在指定地点生成指定物品，代码如下。

```
[UnityTest]
[Description("测试物品生成函数生成的物品的位置和数量是否正确")]
0 个引用
public IEnumerator ItemGenerateTest()
{
    ItemAgent.GenerateItem(new Vector3(5, 5), new HealthPotion { Amount = 10 });

    var item = GameObject.Find("HealthPotion(Clone)");
    Assert.AreEqual(new Vector3(5, 5, 0), item.transform.position);
    Assert.AreEqual(10, item.GetComponent<ItemAgent>().Item.Amount);
    yield return null;
}
```

4.3.2 血药测试

本部分主要测试两点：1.血药能否回复预期的血量；2.使用后也不能超过最大血量。测试代码如下。

```
[UnityTest]
[Description("测试血瓶和血包能不能加人物血量")]
0 个引用
public IEnumerator HealthItemTest()
{
    LogAssert.ignoreFailingMessages = true;

    // 准备测试，先把血量扣到80
    MahouAgent.actualLiving.State.ClearStatus();
    Assert.IsFalse(MahouAgent.actualLiving.State.HasStatus(new InvincibleState()));
    MahouAgent.ChangeHealth(-20f);
    Assert.AreEqual(80f, MahouAgent.actualLiving.CurrentHealth);
    MahouAgent.actualLiving.State.ClearStatus();
    Assert.IsFalse(MahouAgent.actualLiving.State.HasStatus(new InvincibleState()));

    MahouAgent.CleanInventory();

    // 测试使用血瓶，血瓶每次回复10%最大生命值，但使用完之后没有无敌
    MahouAgent.InventoryAddItem(new HealthPotion { });
    MahouAgent.UseItem(new HealthPotion { });
    Assert.AreEqual(90f, MahouAgent.actualLiving.CurrentHealth);
    Assert.IsFalse(MahouAgent.actualLiving.State.HasStatus(new InvincibleState()));

    // 血扣到20
    MahouAgent.ChangeHealth(-70f);
    MahouAgent.actualLiving.State.ClearStatus();

    // 测试使用血包，血瓶每次回复70%最大生命值，但使用完之后没有无敌
    MahouAgent.InventoryAddItem(new Medkit { });
    MahouAgent.UseItem(new Medkit { });
    Assert.AreEqual(90f, MahouAgent.actualLiving.CurrentHealth);
    Assert.IsFalse(MahouAgent.actualLiving.State.HasStatus(new InvincibleState()));

    // 90点血再使用血包，血量也不会超过100
    MahouAgent.InventoryAddItem(new Medkit { });
    MahouAgent.UseItem(new Medkit { });
    Assert.AreEqual(100f, MahouAgent.actualLiving.CurrentHealth);

    yield return null;
    LogAssert.ignoreFailingMessages = false;

}
```

4.3.3 力量药测试

这部分主要测试力量药能不能正确增加人物的攻击力，攻击力无上限。代码如下。

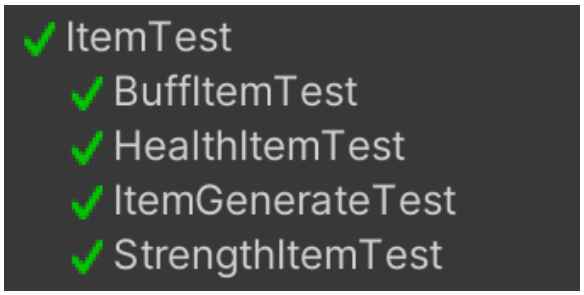
```
[UnityTest]
[Description("测试力量药水能不能加人物攻击")]
0 个引用
public IEnumerator StrengthItemTest()
{
    LogAssert.ignoreFailingMessages = true;

    // 初始人物攻击力应该为1
    Assert.AreEqual(1, MahouAgent.actualLiving.AttackAmount);

    MahouAgent.CleanInventory();
    MahouAgent.InventoryAddItem(new StrengthPotion { });

    // 使用力量药后，攻击力应该为1.2
    MahouAgent.UseItem(new StrengthPotion { });
    Assert.IsTrue(Mathf.Approximately(1.2f, MahouAgent.actualLiving.AttackAmount));

    yield return null;
    LogAssert.ignoreFailingMessages = false;
}
```

4.3.4 Buff 道具测试

游戏中有一个特殊道具，捡起后人物获得无限时长的无敌，这部分测试该道具的效果。代码如下

```
[UnityTest]
0 个引用
public IEnumerator BuffItemTest()
{
    LogAssert.ignoreFailingMessages = true;

    // 先去除人物初始化时的无敌状态
    MahouAgent.BuffColumnAddItem(new InvincibleItem { });
    MahouAgent.actualLiving.State.ClearStatus();
    Assert.IsFalse(MahouAgent.actualLiving.State.HasStatus(new InvincibleState()));

    // 测试无敌物品是否能给人物无敌，且无敌时间是无穷
    MahouAgent.UseBuffItem(new InvincibleItem { });
    Assert.IsTrue(MahouAgent.actualLiving.State.HasStatus(new InvincibleState()));
    Assert.IsTrue(float.IsNaN(MahouAgent.actualLiving.State.StateDuration[new InvincibleState()]));

    MahouAgent.actualLiving.State.ClearStatus();

    yield return null;
    LogAssert.ignoreFailingMessages = false;
}
```

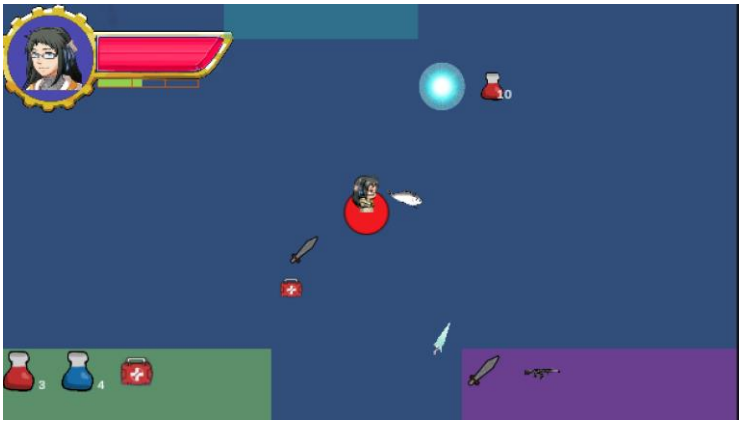
以上是道具部分的白盒测试部分，结果全部通过。



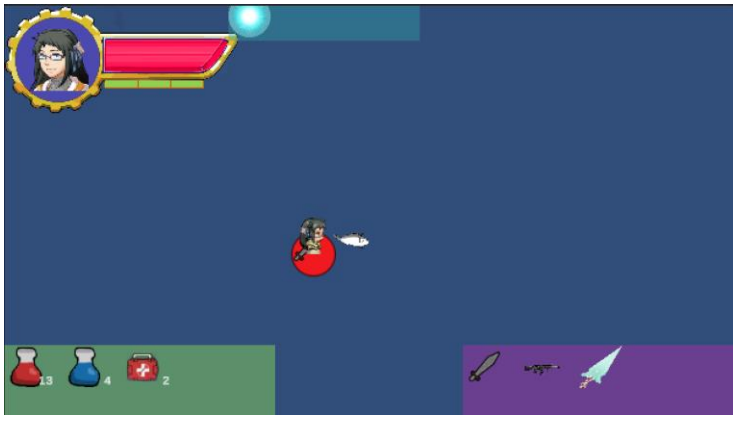
接下来是道具部分的黑盒测试。

4.3.5 道具捡起、道具 UI 加载及变化测试

在场景中生成一些道具如下

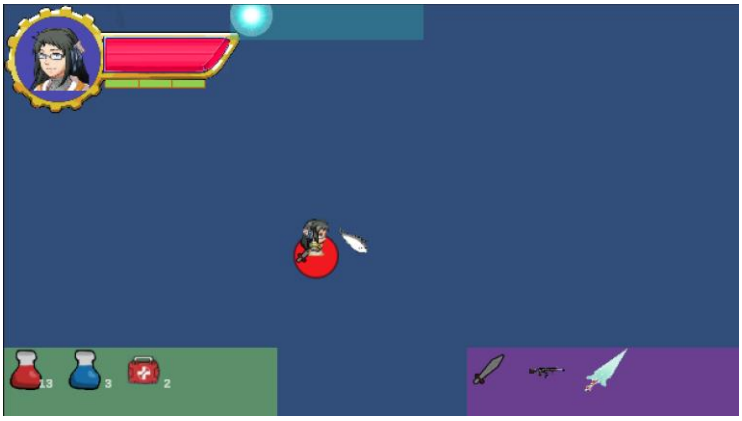


捡起后结果如下



UI 部分的道具栏物品数量、武器栏内武器、上方 buff 栏都正确改变。值得注意的是由于武器栏中本来就有 Sword，故无法再捡起。

由于现在满血，不能使用回血道具，使用一瓶加攻击的道具，结果是背包内改物品数量减少 1。



4.4 武器

在已经准备好的 Test Scene 中放入待测试的 MahouPrefabs、GunPrefab、SwordPrefab，测试脚本[SetUp]部分中写上加载游戏资源并初始化的代码如下：

```
[SetUp]
0 个引用
public void SetUp()
{
    LogAssert.ignoreFailingMessages = true;
    var initialGame = new Initialization();
    initialGame.Awake();

    MahouAgent = GameObject.Find("MahouPrefabs").GetComponent<CharacterAgent>();
    MahouAgent.Awake();

    gun = GameObject.Find("Gun").GetComponent<WeaponAgent>();
    gun.TestAwake(MahouAgent);

    sword = GameObject.Find("Sword").GetComponent<MeleeWeaponAgent>();
    sword.TestAwake(MahouAgent);

    Debug.Log("Set up.");
}
```

4.4.1 武器加载测试

本部分测试武器信息是否被正确加载，近战武器没有“子弹”，远程武器有。代码如下。

```
[UnityTest]
[Order(0)]
[Description("测试能否正确加载武器及其信息")]
0 个引用
public IEnumerator WeaponLoadTest()
{
    LogAssert.ignoreFailingMessages = true;

    Assert.IsNotNull(gun);
    Assert.AreEqual(4, sword.itemIndex);
    Assert.IsNull(sword.bulletPrefab);

    Assert.IsNotNull(sword);
    Assert.AreEqual(5, gun.itemIndex);
    Assert.IsNotNull(gun.bulletPrefab);
    yield return null;

    LogAssert.ignoreFailingMessages = false;
}
```

4.4.2 远程攻击测试

本部分测试以 Gun 远程武器能否发射其子弹，场景中本没有“子弹”，检测调用其 Attack()函数后是否出现“子弹”。代码如下。

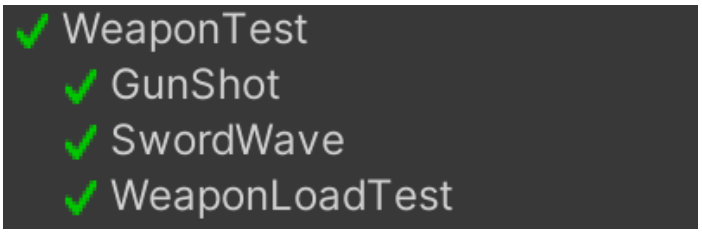
```
[UnityTest]
[Order(1)]
[Description("测试能否正确发射子弹")]
0 个引用
public IEnumerator GunShot()
{
    LogAssert.ignoreFailingMessages = true;
    Assert.IsNull(GameObject.Find("BulletPrefab(Clone)"));

    gun.Attack();
    Assert.IsNotNull(GameObject.Find("BulletPrefab(Clone)"));
    yield return null;

    LogAssert.ignoreFailingMessages = false;
}
```

4.4.3 近战攻击测试

本部分测试以 Sword 为代表的近战武器是否能确挥舞。代码如下。

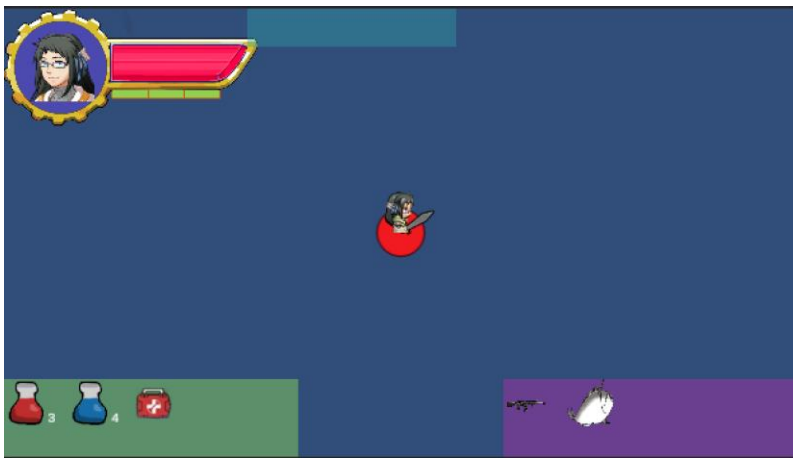
```
[UnityTest]
[Order(2)]
[Description("测试能否挥刀")]
0 个引用
public IEnumerator SwordWave()
{
    LogAssert.ignoreFailingMessages = true;

    bool finishFlag = false;

    var frameCount = 0;
    while ((frameCount++) <= 200)
    {
        // 挥刀过程中返回false，完成一次挥刀后sword.Attack()才返回ture
        if (sword.Attack() == true)
        {
            finishFlag = true;
        }
        yield return null;
    }
    Assert.IsTrue(finishFlag);
    yield return null;

    LogAssert.ignoreFailingMessages = false;
}
```

以上是武器部分白盒测试，结果通过。



至此，所有的白盒+单元测试得到通过。接下来是武器的黑盒测试。

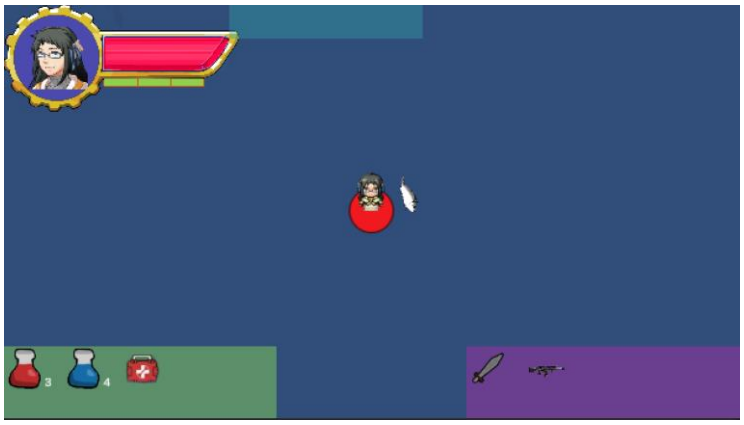
4.4.4 切换测试

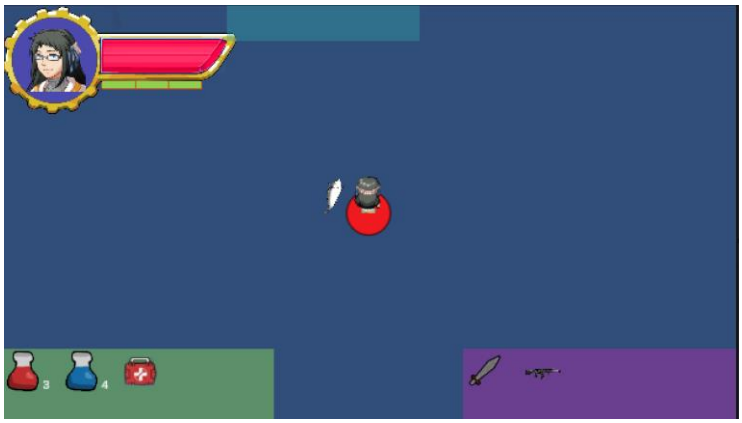


切换前，人物手上拿着“刀”，左下角武器栏中有“枪”和“咸鱼”。

按“H”切换武器，人物手上拿到了“枪”，“刀”加入武器栏某位。

4.4.5 武器瞄准和位置变化测试





武器指向会随着鼠标旋转，而且鼠标 y 方向高于人物，则武器位置调整刀左边，低于人物则调整到右边，与人物动画配合，确保武器在人物左侧。

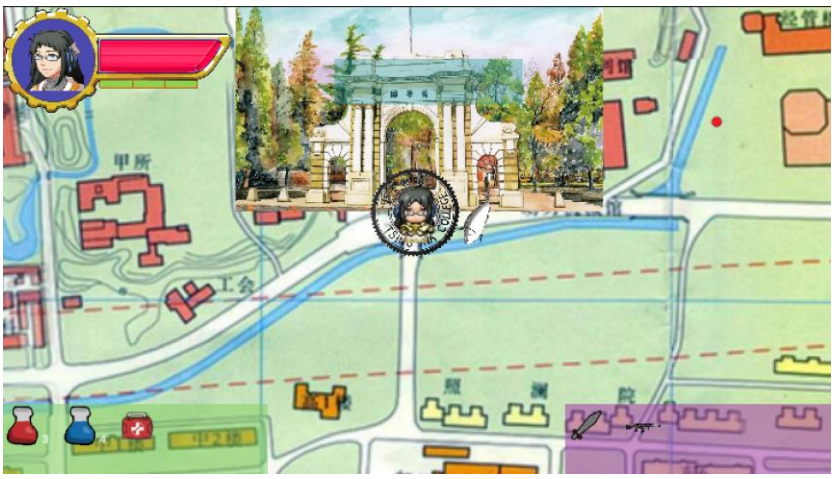
4.5 地图

地图部分主要是黑盒测试。

4.5.1 地形生成测试

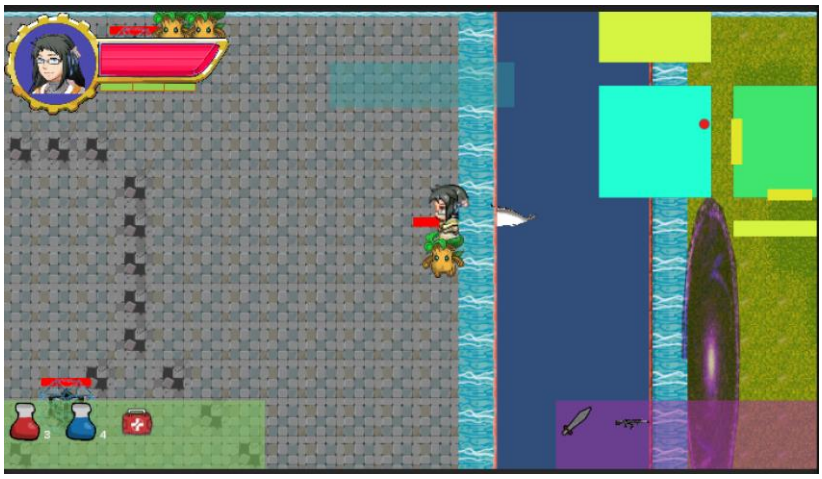
上图中生成了代有墙壁地形的房间，人物和怪物均无法越过墙壁。四周的“黑洞”用于不同房间的传送。

4.5.2 场景传送测试



初始在主地图（清华校园）中，站在地图中校徽处 3 秒，被传送到关卡中。

4.5.3 小地图 UI 测试



关卡中右上角为小地图，矩形色块表示房间，红点显示人物当前位置，黄色矩形显示可以传送的边。

5 吐槽

吐槽一下，由于 Edit Mode 只负责测试局部代码的逻辑，不负责游戏资源的加载及初始化，因此为了白盒测试我们把很多初始化函数可见性由 private 改为 public，以在测试代码中能够显式调用。破坏了封装，这不好。