|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | Bool jump ，bool dash（是否下滑），bool ground（是否在地面上） | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果  实际结果 | 测试状态（P/F） |
| 1 | 把PlayerControl中ground = true，不在下滑，调用jump函数 | Ground = true  Dash = false | Jump = true | Jump = true | P |
| 2 | 把PlayerControl中ground = true，正在下滑，调用jump函数 | Ground = true  Dash = true | Jump = false | Jump = false | P |
| 3 | 把PlayerControl中ground = false，不在下滑，调用jump函数 | Ground = false  Dash = false | Jump = false | Jump = false | P |

**白盒测试（Unity Test Runner）**

**1.人物jump测试**（类似判定表法的穷举）

|  |
| --- |
| [Test]  public void Jump1() {  // Use the Assert class to test conditions.  PlayerControl playerControl = new PlayerControl();  Assert.IsFalse(playerControl.jump);  playerControl.grounded = true;  playerControl.dash = false;  playerControl.jump = true;  bool canJump = false;  if (playerControl.grounded == true&&playerControl.dash == false&&playerControl.jump == true)  canJump= true;  Assert.IsTrue(canJump);  } |

|  |
| --- |
| [Test]  public void Jump2()  {  // Use the Assert class to test conditions.  PlayerControl playerControl = new PlayerControl();  Assert.IsFalse(playerControl.jump);  playerControl.grounded = true;  playerControl.dash = true;  playerControl.jump = true;  bool canJump = false;  if (playerControl.grounded == true && playerControl.dash == false && playerControl.jump == true)  canJump = true;  Assert.IsFalse(canJump);  } |

|  |
| --- |
| [Test]  public void Jump3()  {  // Use the Assert class to test conditions.  PlayerControl playerControl = new PlayerControl();  Assert.IsFalse(playerControl.jump);  playerControl.grounded = false;  playerControl.dash = false;  playerControl.jump = true;  bool canJump = false;  if (playerControl.grounded == true && playerControl.dash == false && playerControl.jump == true)  canJump = true;  Assert.IsFalse(canJump);  } |

**2.人物攻击测试**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | Bool Attack | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | 把PlayerControl中的 Attack 变量设置成true，调用 Attack 函数 | Attack = true | Attack = false | Attack = false | P |

|  |
| --- |
| [Test]  public void Attack()  {  PlayerControl player = new PlayerControl();  Assert.IsFalse(player.attack);  Button\_attack.attack = true;  player.attack = Button\_attack.attack;  Assert.IsTrue(player.attack);  } |

**3.人物火球攻击测试**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | Bool FireBall | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | 把PlayerControl中的 FireBall 变量设置成true，调用 FireBall 函数 | FireBall = true | FireBall = false | FireBall = false | P |

|  |
| --- |
| [Test]  public void FireBall()  {  PlayerControl player = new PlayerControl();  Assert.IsFalse(player.fireballAttack);  Button\_fire.fireball = true;  player.fireballAttack = Button\_fire.fireball;  Assert.IsTrue(player.fireballAttack);  } |

**4.人物行走测试**（类似判定表法的穷举，最后一个例子使用边界值分析法）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | Bool faceRight（人物朝向），float h （人物准备要走的走向 h=1为右，h=-1为左，h=0静止不调用后面的移动命令） | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果  实际结果 | 测试状态（P/F） |
| 1 | FaceRight = true，人物朝向右边，将像右走 | h = 1 | FaceRight = true | FaceRight = true | P |
| 2 | FaceRight = true，人物朝向右边，将像左走 | h = -1 | FaceRight = false | FaceRight = false | P |
| 3 | FaceRight = false，人物朝向左边，将像左走 | h = -1 | FaceRight = false | FaceRight = false | P |
| 4 | FaceRight = false，人物朝向左边，将像右走 | h = 1 | FaceRight = true | FaceRight = true | P |
| 5 | FaceRight = false，人物朝向左边，人物保持静止 | h=0 | FaceRight = false | FaceRight = false | P |

|  |
| --- |
| [Test]  public void Move1()  {  PlayerControl playerControl = new PlayerControl();  Assert.IsTrue(playerControl.facingRight);  float h = 1;  if (h > 0 && !playerControl.facingRight)  {  playerControl.facingRight = true;  }  else if (h < 0 && playerControl.facingRight)  {  playerControl.facingRight = false;  }  Assert.IsTrue(playerControl.facingRight);  } |

|  |
| --- |
| [Test]  public void Move2()  {  PlayerControl playerControl = new PlayerControl();  Assert.IsTrue(playerControl.facingRight);  float h = -1;  if (h > 0 && !playerControl.facingRight)  {  playerControl.facingRight = true;  }  else if (h < 0 && playerControl.facingRight)  {  playerControl.facingRight = false;  }  Assert.IsFalse(playerControl.facingRight);  } |

|  |
| --- |
| [Test]  public void Move3()  {  PlayerControl playerControl = new PlayerControl();  Assert.IsTrue(playerControl.facingRight);  playerControl.facingRight = false;  float h = -1;  if (h > 0 && !playerControl.facingRight)  {  playerControl.facingRight = true;  }  else if (h < 0 && playerControl.facingRight)  {  playerControl.facingRight = false;  }  Assert.IsFalse(playerControl.facingRight);  } |

|  |
| --- |
| [Test]  public void Move4()  {  PlayerControl playerControl = new PlayerControl();  Assert.IsTrue(playerControl.facingRight);  playerControl.facingRight = false;  float h = 1;  if (h > 0 && !playerControl.facingRight)  {  playerControl.facingRight = true;  }  else if (h < 0 && playerControl.facingRight)  {  playerControl.facingRight = false;  }  Assert.IsTrue(playerControl.facingRight);  } |

|  |
| --- |
| [Test]  public void Move5()  {  PlayerControl playerControl = new PlayerControl();  Assert.IsTrue(playerControl.facingRight);  playerControl.facingRight = false;  float h = 0;  if (h > 0 && !playerControl.facingRight)  {  playerControl.facingRight = true;  }  else if (h < 0 && playerControl.facingRight)  {  playerControl.facingRight = false;  }  Assert.IsFalse(playerControl.facingRight);  } |

5.**人物下滑测试**（类似判定表法的穷举）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | Bool Dash（是否下滑），Bool ground（是否在地面上的检测） | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | 设置ground = true，调用Dash（）函数 | Dash = true | Dash = false | Dash = false | P |
| 2 | 设置ground = false，调用Dash（）函数 | Dash = true | Dash = false | Dash = false | P |

|  |
| --- |
| [Test]  public void Dash1()  {  PlayerControl player = new PlayerControl();  player.grounded = true;  player.dash = true;  bool canDash = false;  if(player.grounded == true && player.dash == true)  {  canDash = true;  }  Assert.IsTrue(canDash);  } |

|  |
| --- |
| [Test]  public void Dash2()  {  PlayerControl player = new PlayerControl();  player.grounded = false;  player.dash = true;  bool canDash = false;  if (player.grounded == true && player.dash == true)  {  canDash = true;  }  Assert.IsFalse(canDash);  } |

**6.怪物血量变化测试**（等价类划分法，第一个例子是一个正常数值。第二个例子选择一个非正常数值）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | Float Masterhealth | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | 设置 Masterhealth 初始值，然后调用takedamage（）伤害函数 | Masterhealth = 100f  Takedamage（10） | Masterhealth = 90f | Masterhealth = 90f | P |
| 2 | 设置 Masterhealth 初始值，然后调用takedamage（）伤害函数 | Masterhealth = 100f  Takedamage（110） | Masterhealth = 0f | Masterhealth = 0f | P |

|  |
| --- |
| [Test]  public void Frog\_Health1()  {  // Use the Assert class to test conditions.  Frog\_Health frog = new Frog\_Health();  frog.health = 100f;  frog.TakeDamage\_Test(10f);  Assert.AreEqual(90f, frog.health);  } |

|  |
| --- |
| [Test]  public void Frog\_Health2()  {  // Use the Assert class to test conditions.  Frog\_Health frog = new Frog\_Health();  frog.health = 100f;  frog.TakeDamage\_Test(110f);  Assert.AreEqual(0f, frog.health);  } |

**7.人物血量变化测试**（等价类划分法，第二例用了边界值分析法）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | Float Health，float largest\_Health | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | 设置 Health 初始值，然后调用takedamage（）伤害函数 | Health = 100f  takedamage（10） | Health = 90f | Health = 90f | P |
| 2 | 设置 Health 初始值，然后调用takedamage（）伤害函数 | Health = 100f  Takedamage（110） | Health = 0f | Health = 0f | P |
| 3 | 设置 Health 初始值，设置largest\_Health，然后调用takedamage（）伤害函数 | Health = 95f  largest\_Health  =100f  Takedamage（-10） | Health = 100f | Health = 100f | P |
| 4 | 设置 Health 初始值，设置largest\_Health，然后调用takedamage（）伤害函数 | Health = 85f  largest\_Health  =100f  Takedamage（-10） | Health = 95f | Health = 95f | P |

|  |
| --- |
| [Test]  public void Player\_Health1()  {  // Use the Assert class to test conditions.  PlayerHealth health = new PlayerHealth();  health.TakeDamage\_T(10f);  Assert.AreEqual(90f, health.health);  } |

|  |
| --- |
| [Test]  public void Player\_Health2()  {  // Use the Assert class to test conditions.  PlayerHealth health = new PlayerHealth();  health.TakeDamage\_T(110f);  Assert.AreEqual(0f, health.health);  } |

|  |
| --- |
| [Test]  public void Player\_Health3()  {  // Use the Assert class to test conditions.  PlayerHealth health = new PlayerHealth();  health.health = 95f;  health.TakeDamage\_T(-10f);  Assert.AreEqual(100f, health.health);  } |

|  |
| --- |
| [Test]  public void Player\_Health4()  {  // Use the Assert class to test conditions.  PlayerHealth health = new PlayerHealth();  health.health = 85f;  health.TakeDamage\_T(-10f);  Assert.AreEqual(95f, health.health);  } |

1. **按键测试**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | Button\_Attack.attack | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | Button\_Attack.attack初值为false，调用click（）之后变成true | Button\_Attack.attack = false | Button\_Attack.attack = true | Button\_Attack.attack = true | P |

|  |
| --- |
| [Test]  public void Button\_Attack()  {  Button\_attack.attack = false;  Assert.IsFalse(Button\_attack.attack);  Button\_attack button = new Button\_attack();  button.Click();  Assert.IsTrue(Button\_attack.attack);  } |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | Button\_Dash .dash | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果  实际结果 | 测试状态（P/F） |
| 1 | Button\_Dash .dash 初值为false，调用click（）之后变成true | Button\_Dash .dash = false | Button\_Dash .dash = true | Button\_Dash .dash = true | P |

|  |
| --- |
| [Test]  public void Button\_Dash()  {  Button\_dash.dash = false;  Assert.IsFalse(Button\_dash.dash);  Button\_dash button = new Button\_dash();  button.Click();  Assert.IsTrue(Button\_dash.dash);  } |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | Button\_fire.fireball | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | Button\_fire.fireball 初值为false，调用click（）之后变成true | Button\_fire.fireball = false | Button\_fire.fireball = true | Button\_fire.fireball = true | P |

|  |
| --- |
| [Test]  public void Button\_Fire()  {  Button\_fire.fireball = false;  Assert.IsFalse(Button\_fire.fireball);  Button\_fire button = new Button\_fire();  button.Click();  Assert.IsTrue(Button\_fire.fireball);  } |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | Button\_jump.jump | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | Button\_jump.jump 初值为false，调用click（）之后变成true | Button\_jump.jump = false | Button\_jump.jump = true | Button\_jump.jump = true | P |

|  |
| --- |
| [Test]  public void Button\_Jump()  {  Button\_jump.jump = false;  Assert.IsFalse(Button\_jump.jump);  Button\_jump button = new Button\_jump();  button.Click();  Assert.IsTrue(Button\_jump.jump);  } |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | backToGame.canBackToGame | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | backToGame.canBackToGame 初值为false，调用click（）之后变成true | backToGame.canBackToGame = false | backToGame.canBackToGame = true | backToGame.canBackToGame = true | P |

1. **跳转测试**

|  |
| --- |
| [Test]  public void BackToGame()  {  BackToGame backToGame = new BackToGame();  Assert.IsFalse(backToGame.canBackToGame);  backToGame.onClick\_T();  Assert.IsTrue(backToGame.canBackToGame);  } |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | canToMenu | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | canToMenu 初值为false，调用click（）之后变成true | canToMenu = false | canToMenu = true | canToMenu = true | P |

|  |
| --- |
| [Test]  public void ToMenu()  {  ToMenu toMenu = new ToMenu();  Assert.IsFalse(toMenu.canToMenu);  toMenu.OnClick\_T();  Assert.IsTrue(toMenu.canToMenu);  } |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | canToNewGame | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | canToNewGame 初值为false，调用click（）之后变成true | canToNewGame = false | canToNewGame = true | canToNewGame = true | P |

|  |
| --- |
| [Test]  public void ToNewGame()  {  ToNewGame toNewGame = new ToNewGame();  Assert.IsFalse(toNewGame.canToNewGame);  toNewGame.OnClick\_T();  Assert.IsTrue(toNewGame.canToNewGame);  } |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | canToSetting | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | canToSetting 初值为false，调用click（）之后变成true | canToSetting = false | canToSetting = true | canToSetting = true | P |

|  |
| --- |
| [Test]  public void ToSetting()  {  ToSettings toSetting = new ToSettings();  Assert.IsFalse(toSetting.canToSetting);  toSetting.onClick\_T();  Assert.IsTrue(toSetting.canToSetting);  } |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | canToStore | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | canToStore 初值为false，调用click（）之后变成true | canToStore = false | canToStore = true | canToStore = true | P |

|  |
| --- |
| [Test]  public void ToStore()  {  ToStore toStore = new ToStore();  Assert.IsFalse(toStore.canToStore);  toStore.OnClick\_T();  Assert.IsTrue(toStore.canToStore);  } |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 测试数据 | Active | | | | |
| 操作步骤 | 操作描述 | 数 据 | 期望结果 | 实际结果 | 测试状态（P/F） |
| 1 | Active 初值为false，调用click（）之后变成true | Active = false | Active = true | Active = true | P |

|  |
| --- |
| [Test]  public void QuitCancel()  {  QuitCancel quit = new QuitCancel();  Assert.IsFalse(quit.Active);  quit.onClick\_T();  Assert.IsTrue(quit.Active);  } |