编程环境：Unity 5.6.5 f1（64 位） Microsoft Visual Studio 2010

源文件网址：https://github.com/Tacyqi/Jump

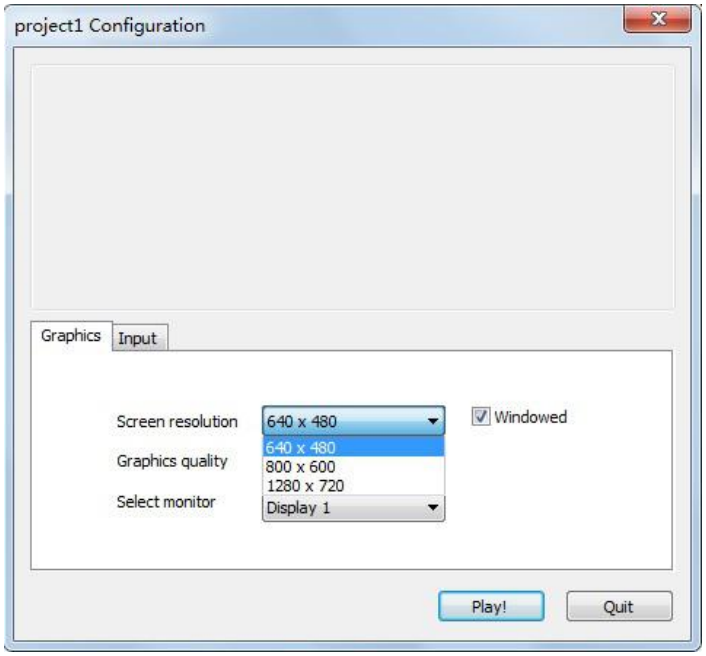
源文件名：project1

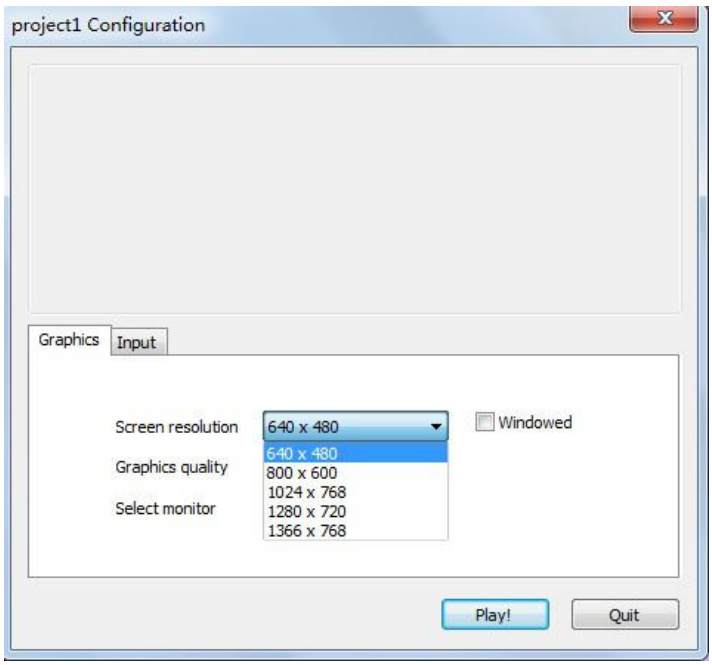
CONTENT

1. 游戏的使用说明

① 应用程序的使用说明（Jump.exe）

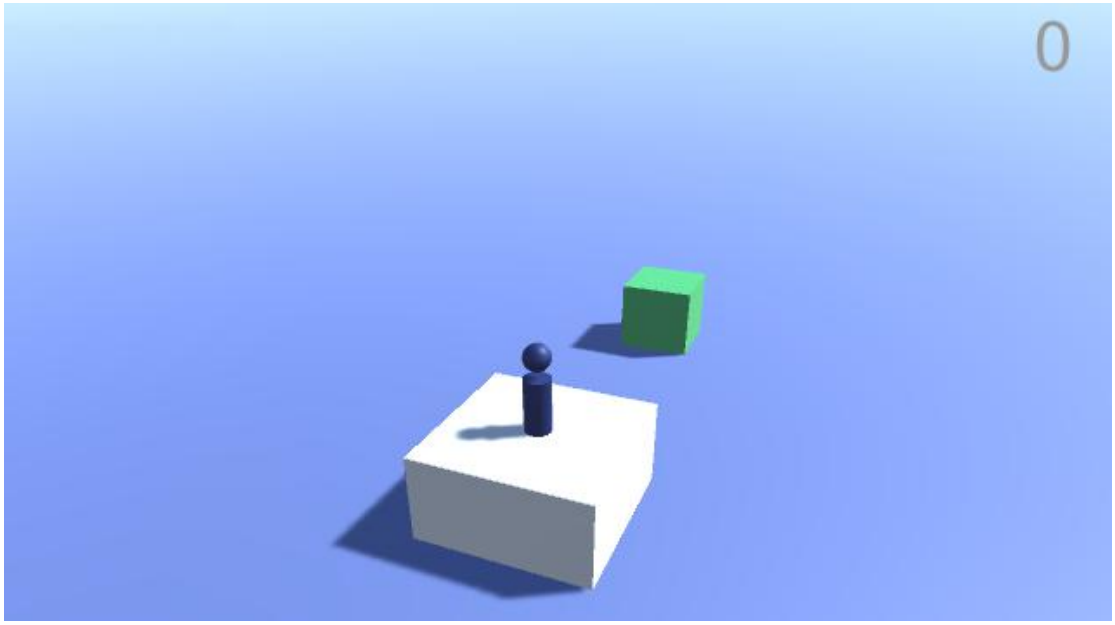
在文件夹中，提供了 2 个版本的应用程序（32 位和 64 位），您可以根据需要下载相应的版本。双击其中一个版本的 Jump.exe 应用程序即可运行游戏。在这里，我们可以设置不同的屏幕大小和画质进行游戏。但值得注意的是，当我们打开的是全屏模式时，点击键盘的 win 键即可回到电脑的主界面。





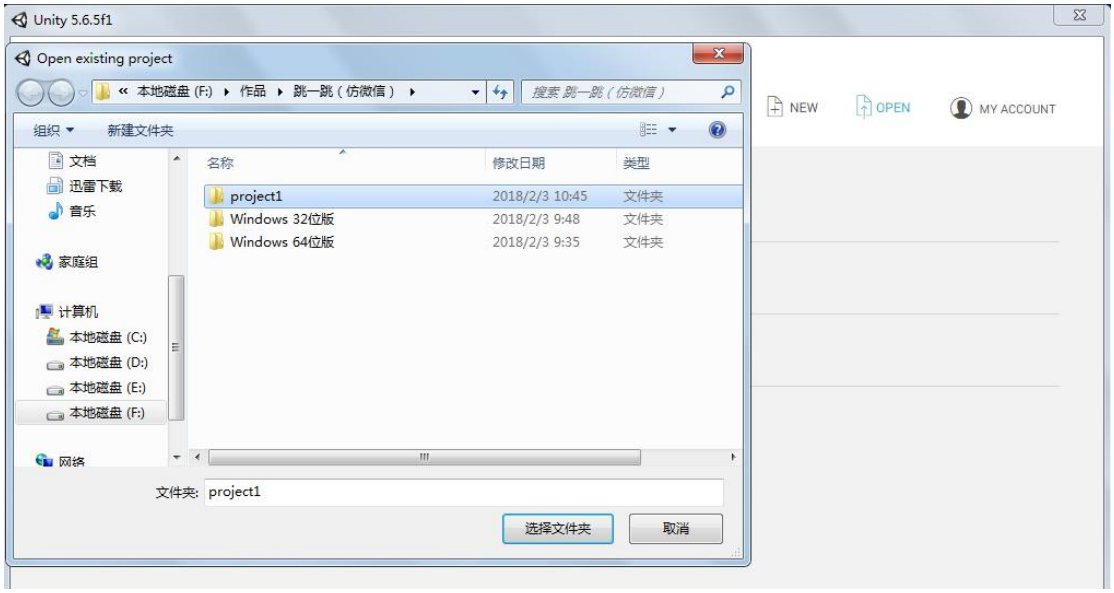
游戏玩法：

打开应用程序后，我们会看到如下界面，点击**空格键**就可对小人进行控制和跳跃。界面的右上角为总计分，每次游戏结束后，分数和界面都会回到最初的位置。祝游戏愉快！



②源文件的使用说明（project1）

打开 Unity（Unity 5.x 和 Unity 2017.x 版本应该都可以运行），选择 Open（打开），在打开的界面中选择 project1 文件夹，即可进入编辑器并可以对其编辑。

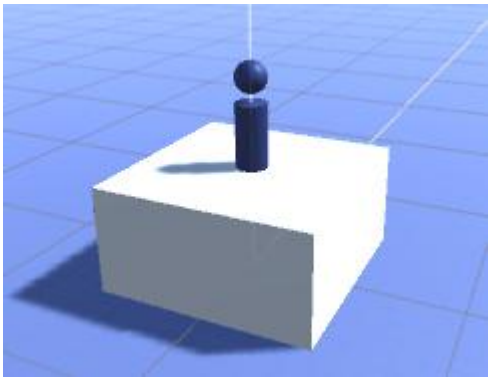


2. 游戏及源代码的分析

在“跳一跳”的小游戏中，我们会有十二个功能需求：角色及场景创建，角色跳跃，台子自动生成，相机跟随，死亡判定及重新开始游戏，分数统计，角色蓄力的粒子效果，角色蓄力效果，角色蓄力台子效果，台子的随机大小/颜色，台子随机生成方向，加分飘分效果。

（1）角色及场景创建

在游戏开始制作之初，我们会对游戏中的角色和场景进行创建。首先我们先创建一个 Cube 物体，作为盒子；再创建一个 Plane 平面，作为地面，并更名为”Ground”；最后创建一个空物体，里面包含了 Sphere 球体（更名为”Head”）和 Cylinder 圆柱体（更名为“Body”）作为小人的身体。我们可以通过添加材质球的方法对物体的颜色进行更改，完成最终的效果。



（2）角色跳跃

为了实现小人跳跃等功能，我们需要给小人添加刚体组件并创建 C#脚本（命名为”Player.cs”）进行控制，决定角色跳跃的部分代码是：

```
public float Factor;
private Rigidbody _rigidbody;   //定义组件
private float _startTime;   //定义开始时间

void Start()
{
    _rigidbody = GetComponent<Rigidbody>();   //获取组件
}

void Update () {
//检测按下空格键的时间
if (Input.GetKeyDown(KeyCode.Space))
```

```
{
    _startTime = Time.time; //计算按下的时间
}
//检测松开空格键的时间
if (Input.GetKeyUp(KeyCode.Space))
{
      var elapse = Time.time - _startTime; //定义从按下到松开的差值
      OnJump(elapse);
 }
}

//根据时长来决定跳跃距离的远近
void OnJump(float elapse)
{
   _rigidbody.AddForce(new Vector3(0,1,0) * elapse * Factor,ForceMode.Impulse);
}
```

在运行游戏的过程中，发现小人由于重心作用一跳就倒，此时我们将小人”Body”部分中的 Capsule Collider 移除，添加一个 Box Collider，并在 Start 函数中添加_rigidbody.centerOfMass = Vector3.zero; 语句来修改小人的重心。

（3）台子自动生成

台子的自动生成需要初始台子和定义台子与台子之间的随机距离，并且当小人跳到下一个台子后便生成下一个台子。其中主要代码为：

```
public float MaxDistance = 3;    //定义初始的最大距离
public GameObject Stage;    //定义初始台子
private GameObject _currentStage;    //定义现在所在的台子
private Collider _lastCollisionCollider; //上一次反生碰撞的 Collider

//定义生成台子的方法
void SpawnStage()
{
      var stage = Instantiate(Stage);
      //当前只定义了台子沿 x 轴方向移动，在设置台子随机生成方向时会有更改
     stage.transform.position = _currentStage.transform.position + new
Vector3(Random.Range(1.1f,MaxDistance),0,0);

}

  //判断 player 是否跳到了台子上（是否与刚体碰撞）
```

```
void OnCollisionEnter(Collision collision)
{
        Debug.Log(collision.gameObject.name);
      //判断是否在同一个台子反生碰撞，如果在同一个台子则不生成下一个台子
      if (collision.gameObject.name.Contains("Cube")&& collision.collider != 
_lastCollisionCollider)
      {
            _lastCollisionCollider = collision.collider;
            _currentStage = collision.gameObject;
             SpawnStage();
       }
}
```

编程中需要注意：要避免小人在同一个台子发生碰撞生成多个台子的情况。

（4）相机跟随

小人跳到下一个台子后，我们需要相机自动跟随到下一个台子。在这里，我们用到了 Unity 的 DOTween 插件。其中主要代码为：

```
public Transform Camera;
private Vector3 _cameraRelativePosition; //相机的相对位置

void Start ()
{
      _rigidbody = GetComponent<Rigidbody>();    //获取组件
      _rigidbody.centerOfMass = Vector3.zero;
      _currentStage = Stage;
      _lastCollisionCollider = _currentStage.GetComponent<Collider>();
      SpawnStage();

     _cameraRelativePosition = Camera.position - transform.position;   //相机的相对
位置 = 相机的位置 - player 的位置
}

//判断 player 是否跳到了台子上（是否与刚体碰撞）
 void OnCollisionEnter(Collision collision)
{
            Debug.Log(collision.gameObject.name);
            if (collision.gameObject.name.Contains("Cube")&& collision.collider != 
_lastCollisionCollider)
            {
```

```
                _lastCollisionCollider = collision.collider;
                _currentStage = collision.gameObject;
                SpawnStage();
                MoveCamera(); //用 MoveCamera 方法来进行移动
            }
        }

    void MoveCamera()
    {
          Camera.DOMove(transform.position + _cameraRelativePosition, 1);
     }
```

（5）死亡判定及重新开始游戏

当小人落到地面时，我们判断游戏结束并重新开始游戏。我们点击 File--Build Settings，在打开的界面中把当前场景添加到”Scenes in bulid”中。这部分主要的代码为：

```
 void OnCollisionEnter(Collision collision)    //判断 player 是否跳到了盒子上（是否与刚体碰撞）
    {
        Debug.Log(collision.gameObject.name);
        if (collision.gameObject.name.Contains("Cube")&& collision.collider != _lastCollisionCollider)
        {
            _lastCollisionCollider = collision.collider;
            _currentStage = collision.gameObject;
            SpawnStage();
            MoveCamera();
        }

        if (collision.gameObject.name == "Ground")
        {
            //本局游戏结束，重新开始
            SceneManager.LoadScene(0);
        }
    }
```

当我们运行程序时，发现游戏失败重新开始时，界面的光线会变成灰色。此时我们需要打开 Windows--Lighting--Settings，在打开的界面中，点击 Generate Lighting 将光照信息进行保存。

（6）分数统计

这部分我们需要将总分数显示在界面中。在编辑器中我们新建一个 Text，并将 Text 的位置放置在界面的右上角，初始值设为 0。这部分的主要代码为：

```
public Text ScoreText;
private int _score;

//判断 player 是否跳到了台子上（是否与刚体碰撞）
void OnCollisionEnter(Collision collision)
{
        Debug.Log(collision.gameObject.name);
        if (collision.gameObject.name.Contains("Cube")&& collision.collider !=
_lastCollisionCollider)
        {
            _lastCollisionCollider = collision.collider;
            _currentStage = collision.gameObject;
            SpawnStage();
            MoveCamera();
                _score++; //每次小人碰撞不同的台子就会将分数加一
                ScoreText.text = _score.ToString();
        }

        if (collision.gameObject.name == "Ground")
        {
            //本局游戏结束，重新开始
            SceneManager.LoadScene(0);
        }
    }
```

（7）角色蓄力的粒子效果

在完成之前的基础功能后，我们对小人添加蓄力粒子效果。用到了 Unity 中例子系统的知识。在 Unity 编辑器中，我们在”Player”下创建一个 Particle System（粒子系统），更改 Particle System 下的 Shape 为 Hemisphere（半球），同时更改粒子的大小、范围和颜色，从而达到比较满意的效果。这部分的主要代码为：

```
public GameObject Particle;

void Update () {
        //检测按下空格键的时间
```

```
        if (Input.GetKeyDown(KeyCode.Space))
        {
            _startTime = Time.time; //计算按下的时间
            Particle.SetActive(true);
        }
        //检测松开空格键的时间
        if (Input.GetKeyUp(KeyCode.Space))
        {
            var elapse = Time.time - _startTime; //定义从按下到松开的差值
            OnJump(elapse);
            Particle.SetActive(false);
        }

    }
```

（8）角色蓄力效果

这部分我们要设置小人在蓄力时下蹲的动作。小人分为头部和身体，头部在蓄力时会沿 Y 轴向下移动，身体在蓄力时会缩放并向下移动。主要代码为：

```
public Transform Head;   //定义小人的头部
public Transform Body;   //定义小人的身体

void Update () {
        //检测按下空格键的时间
        if (Input.GetKeyDown(KeyCode.Space))
        {
            _startTime = Time.time; //计算按下的时间
            Particle.SetActive(true);
        }
        //检测松开空格键的时间
        if (Input.GetKeyUp(KeyCode.Space))
        {
            var elapse = Time.time - _startTime; //定义从按下到松开的差值
            OnJump(elapse);
            Particle.SetActive(false);
            //还原头部和身体的位置
            Body.transform.DOScale(0.1f, 0.2f);
            Head.transform.DOLocalMoveY(0.276f, 0.2f);
        }
```

```
        if (Input.GetKey(KeyCode.Space))
        {
            //调整身体的缩放变化
            Body.transform.localScale += new Vector3(1, -1, 1) * 0.05f *
Time.deltaTime;
            //调整头部的位置
            Head.transform.localPosition += new Vector3(0, -1, 0) * 0.1f *
Time.deltaTime;
        }

    }
```

（9）角色蓄力台子效果

这部分我们要设置台子受到压力时的蓄力效果。其中主要代码为：

```
void Update () {
    //检测按下空格键的时间
    if (Input.GetKeyDown(KeyCode.Space))
    {
        _startTime = Time.time; //计算按下的时间
        Particle.SetActive(true);
    }
    //检测松开空格键的时间
     if (Input.GetKeyUp(KeyCode.Space))
     {
            var elapse = Time.time - _startTime; //定义从按下到松开的差值
            OnJump(elapse);
            Particle.SetActive(false);

            Body.transform.DOScale(0.1f, 0.2f);
            Head.transform.DOLocalMoveY(0.276f, 0.2f);

            _currentStage.transform.DOLocalMoveY(0.25f, 0.2f);
            //_currentStage.transform.DOScale(new Vector3(1, 0.5f, 1), 0.2f);
            _currentStage.transform.DOScaleY(0.5f,0.2f);
     }

     if (Input.GetKey(KeyCode.Space))
     {
```

```
            //调整身体的缩放变化
          Body.transform.localScale += new Vector3(1, -1, 1) * 0.05f * Time.deltaTime;
            //调整头部的位置
        Head.transform.localPosition += new Vector3(0, -1, 0) * 0.1f * Time.deltaTime;

                _currentStage.transform.localScale += new Vector3(0, -1, 0) * 0.15f *
Time.deltaTime;
                _currentStage.transform.localPosition += new Vector3(0, -1, 0) * 0.15f
* Time.deltaTime;    //将台子的 Y 轴位置进行调整
            }

    }
```

（10）台子的随机大小/颜色

这部分我们更改台子的大小和颜色来增加游戏的难度。主要代码为：

```
 void SpawnStage()
    {
        var stage = Instantiate(Stage);
        stage.transform.position = _currentStage.transform.position + new
Vector3(Random.Range(1.1f,MaxDistance),0,0);
        //随机修改盒子的大小
        var randomScale = Random.Range(0.5f,1);
        stage.transform.localScale = new Vector3(randomScale, 0.5f, randomScale);
        //随机修改盒子的颜色（重载函数或重载方法）
        stage.GetComponent<Renderer>().material.color = new
Color(Random.Range(0f, 1), Random.Range(0f, 1), Random.Range(0f, 1));
    }
```

（11）台子随机生成方向

这部分我们更改了台子的随机生成方向（X 轴和 Z 轴两个方向）和小人沿台子的跳跃方向，其中主要的代码为：

```
Vector3 _direction = new Vector3(1, 0, 0); //设初始的位置沿 x 轴的正方向

//判断小人跳跃的方向
void OnJump(float elapse)
```

```
{
        _rigidbody.AddForce((new Vector3(0,1,0) + _direction)* elapse *
Factor,ForceMode.Impulse);
  }

void SpawnStage()
      {
            var stage = Instantiate(Stage);
            stage.transform.position = _currentStage.transform.position + _direction *
Random.Range(1.1f,MaxDistance);

            //随机修改盒子的大小
            var randomScale = Random.Range(0.5f,1);
            stage.transform.localScale = new Vector3(randomScale, 0.5f, randomScale);
            //随机修改盒子的颜色（重载函数或重载方法）
            stage.GetComponent<Renderer>().material.color = new
Color(Random.Range(0f, 1), Random.Range(0f, 1), Random.Range(0f, 1));

     }

//判断 player 是否跳到了盒子上（是否与刚体碰撞）
void OnCollisionEnter(Collision collision)
      {
            Debug.Log(collision.gameObject.name);
            if (collision.gameObject.name.Contains("Cube")&& collision.collider !=
_lastCollisionCollider)
            {
                  _lastCollisionCollider = collision.collider;
                  _currentStage = collision.gameObject;
                  RandomDirection();
                  SpawnStage();
                  MoveCamera();

                  _score++;
                  ScoreText.text = _score.ToString();
            }

            if (collision.gameObject.name == "Ground")
            {
                  //本局游戏结束，重新开始
                  SceneManager.LoadScene(0);
            }
     }
```

```
//定义随机生成方向的方法
 void RandomDirection()
    {
        var seed = Random.Range(0, 2); //只会生成 0,1 两个值
        if (seed == 0)
        {
            _direction = new Vector3(1, 0, 0);
        }
        else
        {
            _direction = new Vector3(0, 0, 1);
        }
    }
```

（12）加分飘分效果

这部分我们对界面进行加分和飘分的效果设置。在设置飘分时我们需要注意将分数的初始位置设置为当前小人所在的位置。首先我们在 Unity 场景编辑器中创建一个新的 Text（命名为”Score”）并更改大小和位置。其中更改的代码为：

```
public Text SingleScoreText; //每次得分显示的成绩
private bool _isUpdateScoreAnimation;    //bool 值的初始值为 false
private float _scoreAnimationStartTime;

void OnCollisionEnter(Collision collision)    //判断 player 是否跳到了盒子上（是否与刚体碰撞）
    {
        Debug.Log(collision.gameObject.name);
        if (collision.gameObject.name.Contains("Cube")&& collision.collider != _lastCollisionCollider)
        {
            _lastCollisionCollider = collision.collider;
            _currentStage = collision.gameObject;
            RandomDirection();
            SpawnStage();
            MoveCamera();
            ShowScoreAnimation();

            _score++;
            ScoreText.text = _score.ToString();
        }
```

```
            if (collision.gameObject.name == "Ground")
            {
                //本局游戏结束，重新开始
                SceneManager.LoadScene(0);
            }
        }

private void ShowScoreAnimation()
{
        _isUpdateScoreAnimation = true;
        _scoreAnimationStartTime = Time.time;
}

//更新 Score 动画
void UpdateScoreAnimation()
{
        //判断动画结束时间
       if (Time.time - _scoreAnimationStartTime > 1)
                _isUpdateScoreAnimation = false;

        //将 3D 空间位置转化为 2D 平面位置，并且将文字的初始位置设为当前小人所在的位置（在这里，需要找到渲染三维的摄像机）
        var playerScreenPos = RectTransformUtility.WorldToScreenPoint(Camera.GetComponent<Camera>(), transform.position);
            SingleScoreText.transform.position = playerScreenPos + Vector2.Lerp(Vector2.zero, new Vector2(0, 200), Time.time - _scoreAnimationStartTime); //分数随着时间向上移动
            SingleScoreText.color = Color.Lerp(Color.black, new Color(0, 0, 0, 0), Time.time - _scoreAnimationStartTime);     //分数的颜色渐渐变淡
        }
```