HW3

本次作业具体内容

1. 在 Pilot 小引擎代码中找到 pilot/engine/source/runtime/function/animation/pose.cpp，找到blend函数补充代码，实现动画融合。
2. 在 Pilot 小引擎代码中找到 pilot/engine/source/runtime/function/animation/animation_FSM.cpp，找到update函数补充代码，实现机器人走跑跳状态机。
3. 在 Pilot 小引擎代码中找到 pilot/engine/source/runtime/function/controller/character_controller.cpp，找到move函数修改代码，利用SceneQuery实现具有相对真实物理表现的character controller：如可跳至平台上，跳起后空中碰到墙壁可以落回地面，前进碰到墙壁可以自动调整位移方向等。

打分细则：

1. [25 分] 正确实现状态机update，通过编译并实现跳跃动画的状态切换。
2. [25 分] 正确实现blend，通过编译并成功实现。
3. [50 分] 修改move，通过编译并成功体现一些真实物理表现，完成对应功能可获得对应分数
    1. [25分]水平移动时可以被墙壁挡住
    2. [25分]跳起后空中碰到墙壁可以落回地面

切到作业分支

```
1 git remote add upstream https://github.com/BoomingTech/Piccolo.git
2 git fetch upstream
3 git checkout -b homework02-rendering upstream/games104/homework02-rendering
```

删掉原来build出来的sln文件

```
1 cmake -S . -B build
```

原理

第11章 动画系统

代码框架

第一部分 状态机

名字	AnimationComponent		
功能	每个拥有骨骼动画的物体都拥有一个		

	AnimationComponent，它存储了每一个骨骼动画所需的数据及动画当前的状态。		
成员	• m_animation_res存储了骨骼及动画资源 `AnimationComponentRes`	m_skeleton_file_path指定骨骼资源	
		m_clips指定所有的动画ClipBase资源	ClipBase • ClipBase是一个拥有名字，长度 有BasicClip、BlendState、Bl 三个子类 • BasicClip是普通的动画clip资源 个普通动画的clip文件路径，与 系，clip的长度等 • BlendState是基本的混合动画资 了一系列普通动画clip，以及他 • BlendSpace1D是一维的混合空 BlendState，它指定了一系列普 并通过一个命名float变量在运行 混合权重
	• m_animation_fsm是一个动画状态机实例 `AnimationFSM` 是一个简易的状态机	AnimationComponent将会调用状态机的update函数，将接收集到的信号传递给状态机 状态机根据signals以及update函数内部的逻辑更新当前状态m_state，update函数已经留空 AnimationComponent将会调用getClipBaseName获得需要播放的ClipBase的名字(它可能是BasicClip、BlendState、BlendSpace1D)并将计算该ClipBase的结果应用到物体的骨骼。完成update函数后跳跃将看到动画切换 根据状态机示意图 （其中边说明中的[x]代表优先级，数字越小优先级越高） 完成代码。	
	• 每一帧AnimationComponent的tick函数将会被调用，它是该动画组件运算的入口		
	• 外部系统可以通过updateSignal传递一些信号		
位置	engine\source\runtime\function\framework\com		

	ponent\animation\animation_component.h		

第二部分 动画融合

如果当前AnimationComponent的ClipBase是一个BlendState，它将通过blend函数计算结果（结果是一个AnimationPose） 它首先计算出所有BlendState列表里的所有Clip的当前的Pose，然后通过AnimationPose::blend将这些结果Pose混合到一个pose上	`AnimationPose`	AnimationPose存储骨骼Transform数组及对应的权重（通过下标对应）	Transform的定义
		AnimationPose::blend函数已经挖空，完成函数可以看到走到跑过渡的效果（可以根据注释的提示，也可以不理会注释）	

第三部分 利用SceneQuery实现具有相对真实物理表现的character controller

玩家控制角色运动涉及功能层中input、motor、character controller三个系统中的逻辑。	玩家按 下ASWD（前后左右移动）、Shift（跑）、空格键（跳）后，Input系统对应处理产生游戏指令 m_game_command。
	motor系统根据各游戏指令属性，计算出motor层逻辑期望的运动位移 m_desired_displacement。具体计算流程分散在motor_component.cpp的若干函数中。motor系统分别计算水平向速度、竖直方向速度、水平移动方向、逻辑期望位移，最终计算出移动位置。同 学们可以阅读motor_component.cpp相关函数代码以理解跳跃状态相关逻辑。
	character controller收到motor的逻辑期望位移，在物理场景中进行场景请求计算出符合物理规律的实际运动后的位置。 目前实现的版本实现了竖直方向上的检测，可以跳至平台上。character controller通常的实现方式是分离水平和竖直方向上的检测pass来分阶段处理。 controller内部会维护touch_ground等状态，使motor层能够根据controller在物理场景中的结果改变跳跃状态等。 同学们需要改写CharacterController::move函数利用射线检测或形状扫描等场景请求来实现如下功能： 水平移动时可以被墙壁挡住 跳起后空中碰到墙壁可以落回地面（目前如果在下落阶段碰到墙壁会卡在墙上）

	感兴趣的同学可以继续尝试实现水平移动碰到墙面不仅仅是阻挡前进而且可以自动修正运动方向、auto-stepping或者沿斜面下滑等行为，不作为打分作业内容

完成过程

状态机

只要根据提示补充updata函数即可

动画融合

AnimationComponent::tick()函数中，会计算当前ClipBase的播放时间，然后判断ClipBase是否需要混合。

```
                else if (clip.getTypeName() == "BlendState")
                {
                    auto blend_state = static_cast<BlendState*>(clip);
                    blend(m_ratio, blend_state);// m_ratio是播放时长，blend_state
                }
```

BlendState是这样定义的：

```
    REFLECTION_TYPE(BlendState)
    CLASS(BlendState : public ClipBase, Fields)
    {
        REFLECTION_BODY(BlendState);

    public:
        int                      m_clip_count;
        std::vector<std::string> m_blend_clip_file_path;
        std::vector<float>       m_blend_clip_file_length;
        std::vector<std::string> m_blend_anim_skel_map_path;
        std::vector<float>       m_blend_weight;
        std::vector<std::string> m_blend_mask_file_path;
        std::vector<float>       m_blend_ratio;
        virtual ~BlendState() override {}
        virtual float getLength() const override
        {
            float length = 0;
            for (int i = 0; i < m_clip_count; i++)
            {
                auto curweight = m_blend_weight[i];
                length += curweight * m_blend_clip_file_length[i];
            }
            return length;
        }
    };
```

```
    void AnimationComponent::blend(float desired_ratio, BlendState* blend_state)
    {
```

```
3
4           for (auto& ratio : blend_state->m_blend_ratio)
5           {
6               ratio = desired_ratio;
7           }
8           auto blendStateData = AnimationManager::getBlendStateWithClipData(*blend
9           std::vector<AnimationPose> poses;
10          for (int i = 0; i < blendStateData.m_clip_count; i++)
11          {
12              AnimationPose pose(blendStateData.m_blend_clip[i],
13                                 blendStateData.m_blend_weight[i],
14                                 blendStateData.m_blend_ratio[i],
15                                 blendStateData.m_blend_anim_skel_map[i]);
16              m_skeleton.resetSkeleton();
17              m_skeleton.applyAdditivePose(pose);
18              m_skeleton.extractPose(pose);
19              poses.push_back(pose);
20          }
21          for (int i = 1; i < blendStateData.m_clip_count; i++)
22          {
23              for (auto& pose : poses[i].m_weight.m_blend_weight)
24              {
25                  pose = blend_state->m_blend_weight[i];
26              }
27              poses[0].blend(poses[i]);
28          }
29
30          m_skeleton.applyPose(poses[0]);
31          m_animation_result = m_skeleton.outputAnimationResult();
32      }
```

如果当前AnimationComponent的ClipBase是一个BlendState，它将通过blend函数计算结果（结果是一个AnimationPose） 它首先计算出所有BlendState列表里的所有Clip的当前的Pose，BlendState是由AnimationComponent传进来的，然后通过 AnimationPose::blend将这些结果Pose混合到一个pose上，这个函数被挖空了。

物理表现

不让它卡在墙上

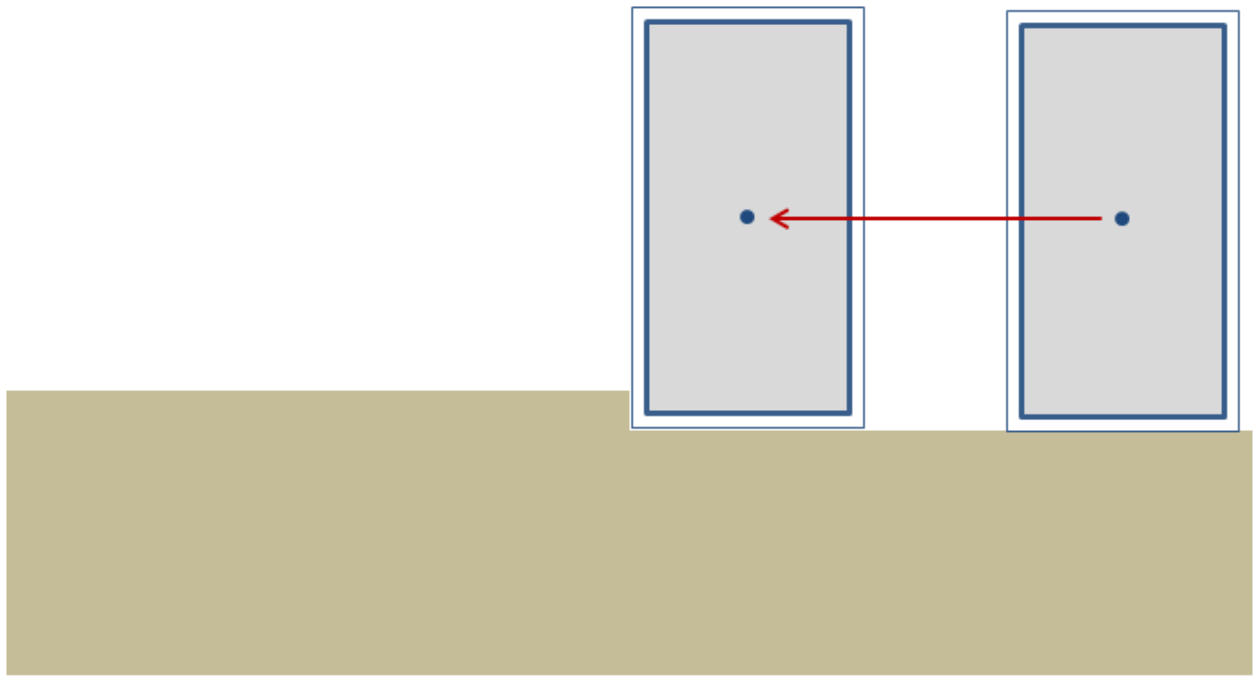
```
1 sweep(
2             m_rigidbody_shape,
3             world_transform.getMatrix(),
4             horizontal_direction,
5             horizontal_displacement.length(),
6             hits)
```

未完成

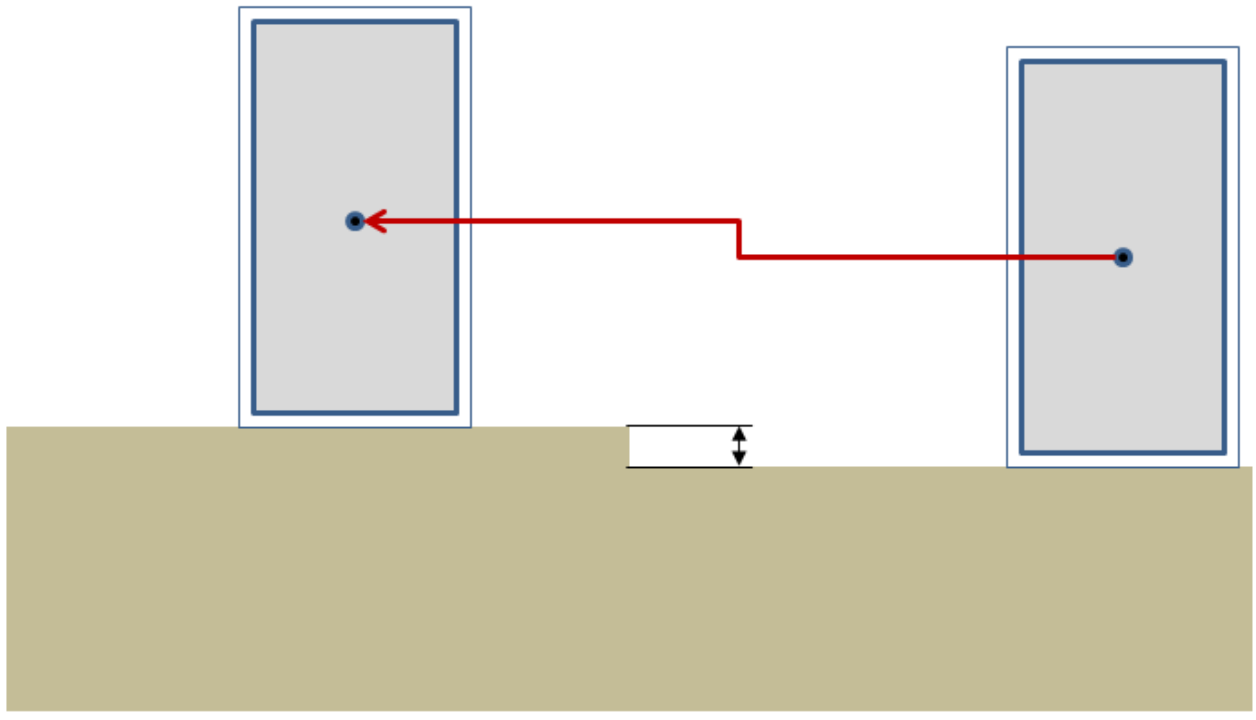
Auto Stepping

如果没有 `auto-stepping` ， `box-controlled` 角色很容易卡在地面网格的轻微高度上。 在下图中，一小步将完全停止角色。 感觉很不自然，因为在现实世界中，角色会不假思索地越过这个小障

碍。



这就是自动步进使我们能够做到的。 没有玩家的任何干预(即没有他们考虑)，盒子正确地越过小障碍物。



但是，如果障碍物太大，即其高度大于 `stepOffset` 参数，则控制器无法自动爬升，角色卡住(这次是正确的)：

“攀爬”(例如，越过这个更大的障碍)也可能在未来实现，作为自动步进的延伸。 `step offset` 在 `PxControllerDesc::stepOffset` 中定义，稍后可通过 `PxController::getStepOffset()` 函数获得。

一般来说， `step offset` 偏移量应尽可能小。

前进碰到墙壁自动调整位移方向往侧边移动

未完成

滑步问题

未完成