GAMES104 作业 3　　作者：Tiger

一、 状态机 update 函数代码的截图及实现思路讲解

```cpp
bool AnimationFSM::update(const json11::Json::object& signals)
    {
        States last_state     = m_state;
        bool   is_clip_finish = tryGetBool(signals, "clip_finish", false);
        bool   is_jumping     = tryGetBool(signals, "jumping", false);
        float  speed          = tryGetFloat(signals, "speed", 0);
        bool   is_moving      = speed > 0.01f;
        bool   start_walk_end = false;

        // std::cout << speed << std::endl;

        switch (m_state)
        {
            case States::_idle:
                /**** [0] ****/
                if (is_jumping)
                    m_state = States::_jump_start_from_idle;
                else if (is_moving)
                    m_state = States::_walk_start;
                break;
            case States::_walk_start:
                /**** [1] ****/
                if (is_clip_finish)
                    m_state = States::_walk_run;
                break;
            case States::_walk_run:
                /**** [2] ****/
                if (!is_moving)
                    m_state = States::_idle;
                else if(is_jumping)
                    m_state = States::_jump_start_from_walk_run;
                else if (start_walk_end && is_clip_finish)
                    m_state = States::_walk_stop;
                break;
            case States::_walk_stop:
                /**** [3] ****/
                if (!is_moving && is_clip_finish)
                    m_state = States::_idle;
                break;
            case States::_jump_start_from_idle:
                /**** [4] ****/
                if (is_clip_finish)
                    m_state = States::_jump_loop_from_idle;
                break;
            case States::_jump_loop_from_idle:
                /**** [5] ****/
                if (!is_jumping)
                    m_state = States::_jump_end_from_idle;
                break;
            case States::_jump_end_from_idle:
                /**** [6] ****/
                if (is_clip_finish)
                    m_state = States::_idle;
                break;
            case States::_jump_start_from_walk_run:
                /**** [7] ****/
                if (is_clip_finish)
                    m_state = States::_jump_loop_from_walk_run;
                break;
            case States::_jump_loop_from_walk_run:
                /**** [8] ****/
                if (!is_jumping)
                    m_state = States::_jump_end_from_walk_run;
                break;
            case States::_jump_end_from_walk_run:
                /**** [9] ****/
                if (is_clip_finish)
                    m_state = States::_walk_run;
                break;
            default:
                break;
        }
        return last_state != m_state;
    }
```

思路讲解：

按照作业要求中的状态机示意图设置条件跳转即可。

发现的可改进之处：

当前工程中，start_walk_end 似乎没有结束 signal，原代码中一直给的是 false，推测
是因为目前没有走动结束的动画资源。

另外，可能目前动画资源数量较少，有些动画不够流畅，比如跳跃结束后，如果角色还有速度，似乎会“打滑”一段距离。

二、 AnimationPose::blend 代码截图及实现思路讲解

```
void AnimationPose::blend(const AnimationPose& pose)
{
    for (int i = 0; i < m_bone_poses.size(); i++)
    {
        auto&       bone_trans_one = m_bone_poses[i];
        const auto& bone_trans_two = pose.m_bone_poses[i];

        float sum_weight = m_weight.m_blend_weight[i] + pose.m_weight.m_blend_weight[i];
        if (sum_weight != 0)
        {
            float cur_weight = pose.m_weight.m_blend_weight[i];
            m_weight.m_blend_weight[i] = sum_weight;
            float ratio = cur_weight/sum_weight;
            bone_trans_one.m_position  = Vector3::lerp(bone_trans_one.m_position,
bone_trans_two.m_position, ratio);
            bone_trans_one.m_scale     = Vector3::lerp(bone_trans_one.m_scale,
bone_trans_two.m_scale, ratio);
            bone_trans_one.m_rotation  = Quaternion::nLerp(ratio,
bone_trans_one.m_rotation, bone_trans_two.m_rotation, true);
        }
    }
}
```

思路讲解：

首先将两帧动画按权重计算出混合比例，利用库里提供的插值方法进行混合。
这两帧的权重会累积后存储，用于和再下一帧的动画混合。
不断重复这一过程即可。

三、 CharacterController::move 代码截图及实现思路讲解

```
// side pass
        if (physics_scene->sweep(
            m_rigidbody_shape,
            world_transform.getMatrix(),
            horizontal_direction,
            horizontal_displacement.length(),
            hits))
        {
        //    final_position += hits[0].hit_distance * horizontal_direction;

            float elastic_ratio = 0.25f;
            Vector3 desired_direction =
Vector3::lerp(horizontal_direction.project(hits[0].hit_normal),
horizontal_direction.reflect(hits[0].hit_normal), elastic_ratio);
            desired_direction.normalise();
            Vector3 desired_displacement = horizontal_displacement.length() *
desired_direction;

            hits.clear();
            if (physics_scene->sweep(
                m_rigidbody_shape,
                world_transform.getMatrix(),
                desired_direction,
                desired_displacement.length(),
                hits))
            {
                final_position += hits[0].hit_distance * desired_direction;
            }
            else
            {
                final_position += desired_displacement;
            }
        }
        else
        {
            final_position += horizontal_displacement;
        }

        hits.clear();
```

思路讲解：

首先类比垂直的阻挡检测，进行一次水平方向的 sweep。

为了实现前进碰到墙壁可以自动调整位移方向，如果碰到物体，则将方向投影到平行物体的方向。

这里，还额外增加了一个与反射方向按弹性比例混合的过程，主要是为了防止过于靠近墙壁而卡在其中或滑移不了的情况。

接下来，再对新方向的移动进行类似的阻挡检测，如果发现还是碰撞，则说明角色处在墙角位置，不进行移动。否则，按照新方向移动即可。

其他行为，如跳起后空中碰到墙壁可以落回地面，也已包含在上述逻辑中。

运行情况可参考文件夹中的视频。