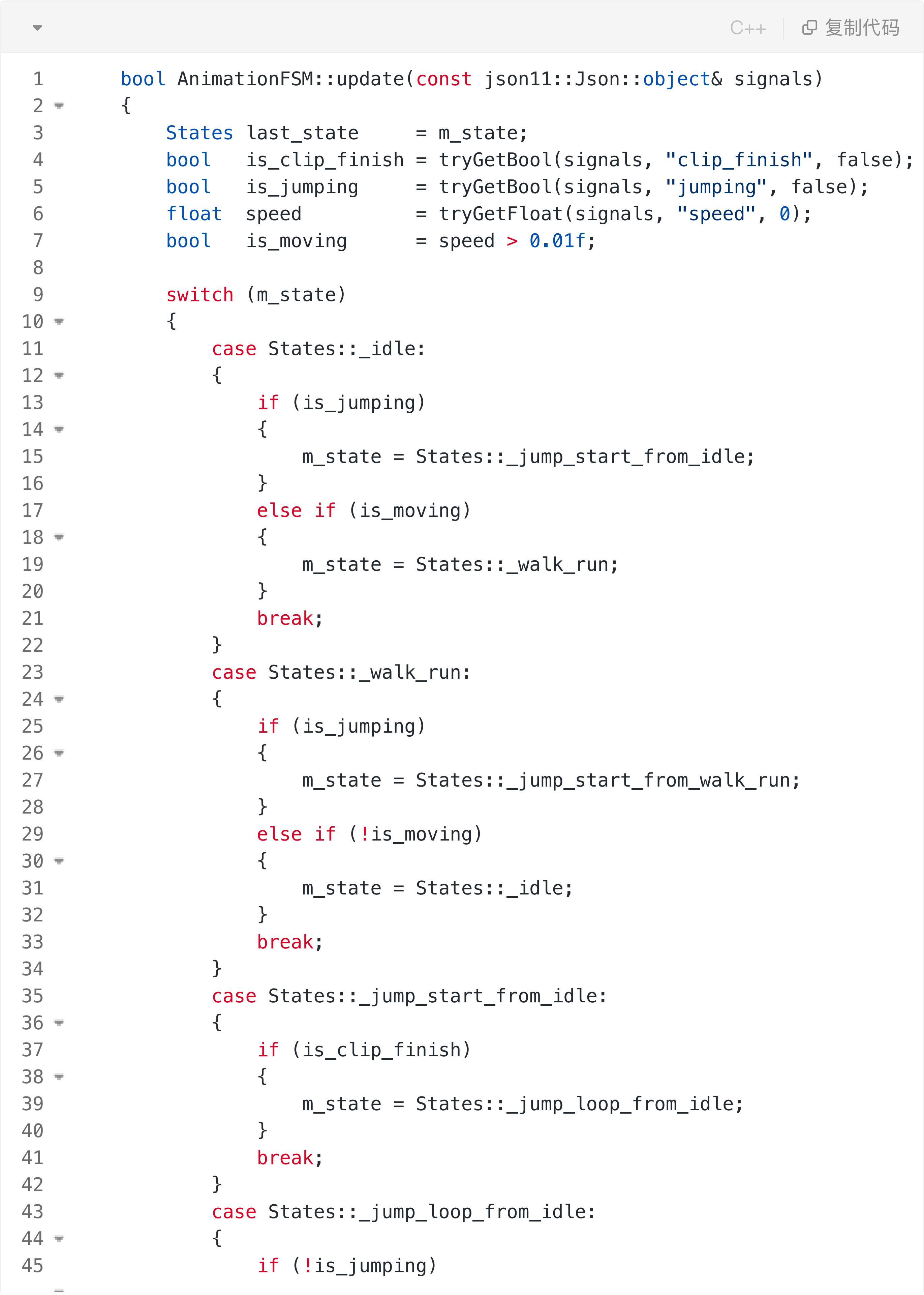
Games104_homework3_report

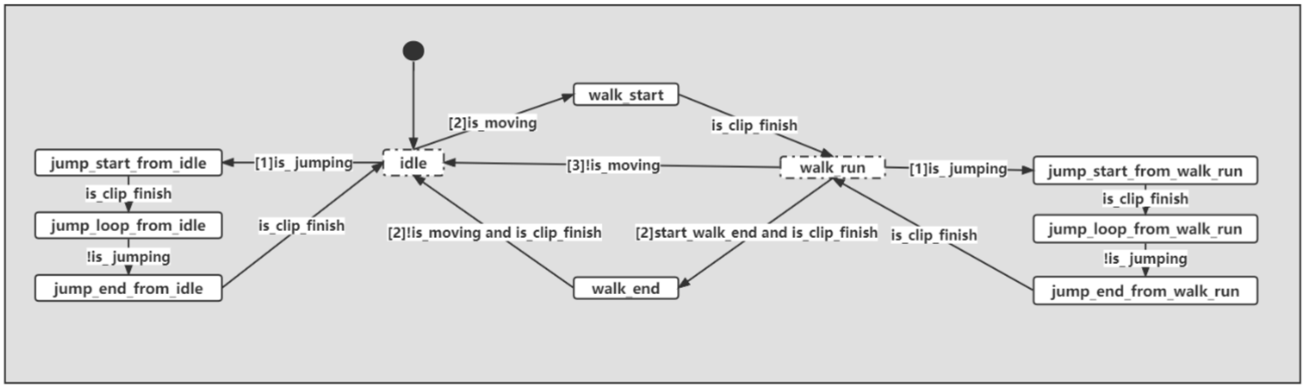
1. 状态机实现思路



```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;

    switch (m_state)
    {
        case States::_idle:
        {
            if (is_jumping)
            {
                m_state = States::_jump_start_from_idle;
            }
            else if (is_moving)
            {
                m_state = States::_walk_run;
            }
            break;
        }
        case States::_walk_run:
        {
            if (is_jumping)
            {
                m_state = States::_jump_start_from_walk_run;
            }
            else if (!is_moving)
            {
                m_state = States::_idle;
            }
            break;
        }
        case States::_jump_start_from_idle:
        {
            if (is_clip_finish)
            {
                m_state = States::_jump_loop_from_idle;
            }
            break;
        }
        case States::_jump_loop_from_idle:
        {
            if (!is_jumping)
            {
                m_state = States::_jump_end_from_idle;
            }
            break;
        }
        case States::_jump_end_from_idle:
        {
            if (is_clip_finish)
            {
                m_state = States::_idle;
            }
            break;
        }
        case States::_jump_start_from_walk_run:
        {
            if (is_clip_finish)
            {
                m_state = States::_jump_loop_from_walk_run;
            }
            break;
        }
        case States::_jump_loop_from_walk_run:
        {
            if (!is_jumping)
            {
                m_state = States::_jump_end_from_walk_run;
            }
            break;
        }
        case States::_jump_end_from_walk_run:
        {
            if (is_clip_finish)
            {
                m_state = States::_walk_run;
            }
            break;
        }
        default:
            break;
    }
    return last_state != m_state;
}
```

- 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。



- 状态机是根据给的状态转换图，每个state判断达成条件进入另一个state，if和else if顺序根据给的优先级排序
- 发现代码中 `start_walk_end` 永远是false，而且带入 `_walk_start` 和 `_walk_stop` 后动画表现不对，所以做出了优化修改

2. AnimationPose混合

```cpp
void AnimationPose::blend(const AnimationPose& pose)
{
    // Loop each bone
    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];

        const float& weight_one = m_weight.m_blend_weight[i];
        const float& weight_two = pose.m_weight.m_blend_weight[i];
        float sum_weight = weight_one + weight_two;

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

1. 两个pos，遍历每根bone
2. 算出pos1和pos2之间的权重，给后续位置、旋转、缩放进行插值使用
3. 旋转插值时使用 `slerp` 解决钝角插值bug
4. `m_weight.m_blend_weight[i]` 等于 `sum_weight` 是pos的总权重，用来累加做归一化，可以在其他pos混合时使用

3. side pass 实现思路

```cpp
        // side pass
        if (physics_scene->sweep(m_rigidbody_shape,
                                 world_transform.getMatrix(),
                                 horizontal_direction,
                                 horizontal_displacement.length(),
                                 hits))
        {
            Vector3 total_normal = Vector3::ZERO;
            for (auto it = hits.begin(); it != hits.end(); it++)
            {
                total_normal += (*it).hit_normal.normalisedCopy();
            }
            total_normal.z = 0.0f;

            float   sliding_distance  = total_normal.crossProduct(horizontal_displacement).z;
            Vector3 sliding_direction = Vector3(-total_normal.y, total_normal.x, total_normal.z);
            final_position += sliding_direction * sliding_distance;
        }
        else
        {
            final_position += horizontal_displacement;
        }
```

1. 通过 cast shape查询到hits数组
2. 遍历hits收集所有命中的法线计算出一个 `total_normal`
3. 通过 `total_normal` 和 `horizontal_dispacement` 进行 `crossProduct` 模拟出滑动向量
4. 再通过 `total_normal` 的yx方向计算出滑动方向
5. 最后计算出滑动点