Games104_homework3_report

Online version: https://nbqmlgi3yg.feishu.cn/docx/doxcncy55zMVa0BlEgNetcdvoNd

The content in the online version may be more updated.

Author: huandzh@gmail.com (huandzh (DHuan) (github.com))

Post Date: 2022-06-09

状态机update函数

以下代码实现的状态机和任务说明中的示意图有所区别，增加了角色停步但又改主意的情况。

代码：

Pilot/engine/source/runtime/function/animation/animation_FSM.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 = tryGetBool(signals, "stop_moving", false);

    // TODO: PA03_2
    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:
            if (is_jumping)
            {
                m_state = States::_jump_start_from_walk_run;
            }
            else if (start_walk_end && is_clip_finish)
            {
                m_state = States::_walk_stop;
            }
            else if (!is_moving)
            {
                m_state = States::_idle;
            }
            /**** [2] ****/
            break;
        case States::_walk_stop:
            if (!is_moving && is_clip_finish)
            {
                m_state = States::_idle;
            }
            else if (!start_walk_end)
```

```
                {
                    m_state = States::_walk_start;
                }
                /**** [3] ****/
                break;
            case States::_jump_start_from_idle:
                if (is_clip_finish)
                {
                    m_state = States::_jump_loop_from_idle;
                }
                /**** [4] ****/
                break;
            case States::_jump_loop_from_idle:
                if (!is_jumping)
                {
                    m_state = States::_jump_end_from_idle;
                }
                /**** [5] ****/
                break;
            case States::_jump_end_from_idle:
                if (is_clip_finish)
                {
                    m_state = States::_idle;
                }
                /**** [6] ****/
                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;
        }
    #ifdef NDEBUG
    #else
```

```
            if (last_state != m_state)
            {
                DEBUG_LOG_DEBUG(getCurrentStateName());
            }
    #endif
            return last_state != m_state;
        }

        std::string AnimationFSM::getCurrentClipBaseName() const
        {
            switch (m_state)
            {
                case States::_idle:
                    return "idle_walk_run";
                case States::_walk_start:
                    return "walk_start";
                case States::_walk_run:
                    return "idle_walk_run";
                case States::_walk_stop:
                    return "walk_stop";
                case States::_jump_start_from_walk_run:
                case States::_jump_start_from_idle:
                    return "jump_start";
                case States::_jump_loop_from_walk_run:
                case States::_jump_loop_from_idle:
                    return "jump_loop";
                case States::_jump_end_from_walk_run:
                case States::_jump_end_from_idle:
                    return "jump_stop";
                default:
                    return "idle_walk_run";
            }
        }

        std::string AnimationFSM::getCurrentStateName() const
        {
            switch (m_state)
            {
                case States::_idle:
                    return PolitNameOf(States::_idle);
                case States::_walk_start:
                    return PolitNameOf(States::_walk_start);
                case States::_walk_run:
                    return PolitNameOf(States::_walk_run);
                case States::_walk_stop:
                    return PolitNameOf(States::_walk_stop);
                case States::_jump_start_from_walk_run:
                    return PolitNameOf(States::_jump_start_from_walk_run);
                case States::_jump_start_from_idle:
                    return PolitNameOf(States::_jump_start_from_idle);
                case States::_jump_loop_from_walk_run:
                    return PolitNameOf(States::_jump_loop_from_walk_run);
```

```
                case States::_jump_loop_from_idle:
                    return PolitNameOf(States::_jump_loop_from_idle);
                case States::_jump_end_from_walk_run:
                    return PolitNameOf(States::_jump_end_from_walk_run);
                case States::_jump_end_from_idle:
                    return PolitNameOf(States::_jump_end_from_idle);
                    ;
                default:
                    return "StatesUnknown";
            }
        }
```

Pilot/engine/source/runtime/function/framework/component/motor/motor_component.cpp

```
void MotorComponent::calculatedDesiredMoveDirection(unsigned int command, const Quater
{
    if (m_jump_state == JumpState::idle)
    {
        Vector3 forward_dir = object_rotation * Vector3::NEGATIVE_UNIT_Y;
        Vector3 left_dir    = object_rotation * Vector3::UNIT_X;

        m_move_state = MoveState::idle;

        if (command > 0)
        {
            m_desired_horizontal_move_direction = Vector3::ZERO;
        }

        if ((unsigned int)GameCommand::forward & command)
        {
            m_desired_horizontal_move_direction += forward_dir;
            m_move_state = MoveState::moving;
        }

        if ((unsigned int)GameCommand::backward & command)
        {
            m_desired_horizontal_move_direction -= forward_dir;
            m_move_state = MoveState::moving;
        }

        if ((unsigned int)GameCommand::left & command)
        {
            m_desired_horizontal_move_direction += left_dir;
            m_move_state = MoveState::moving;
        }

        if ((unsigned int)GameCommand::right & command)
        {
            m_desired_horizontal_move_direction -= left_dir;
            m_move_state = MoveState::moving;
        }

        m_desired_horizontal_move_direction.normalise();

        AnimationComponent* animation_component =
            m_parent_object.lock()->tryGetComponent<AnimationComponent>("AnimationComp
        animation_component->updateSignal("stop_moving", m_move_state == MoveState::id
    }
}
```

实现思路：

常规实现要点

因为任务说明提供了状态机示意图，按图索骥实现状态机是比较直接的。实现要点在于：

1. 当处于某个状态时，在相应的case中进行条件判断
2. 条件满足时，跳转到其他状态
3. 状态间的切换逻辑合理，条件闭合环路（示意图基本保证了这一点）

增加调试日志

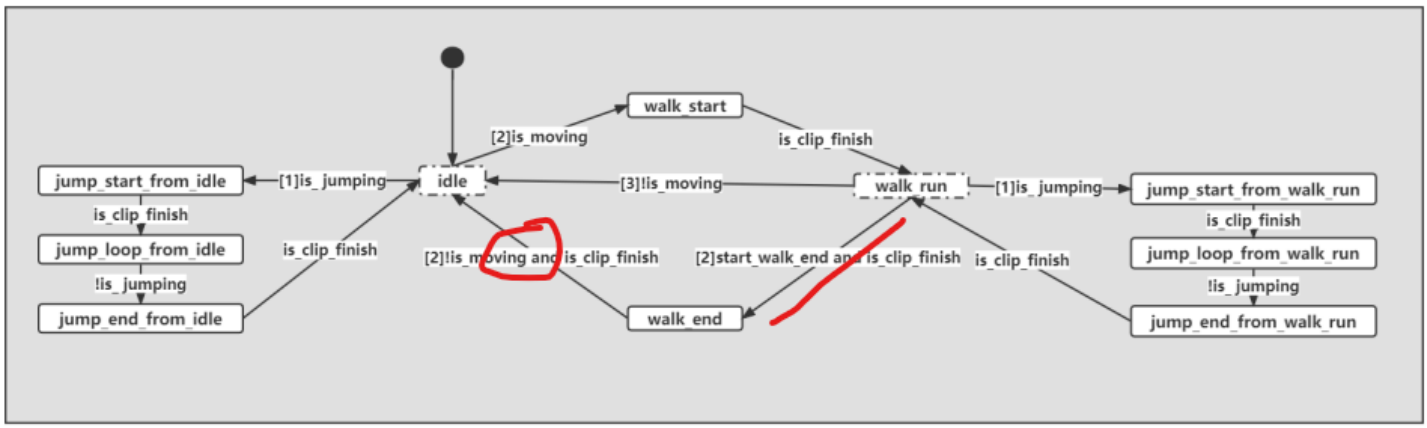
如果单纯看状态机控制的动画效果，会比较难以分清潜在bug的产生原因。比如，动画混合或动画资源本身也可能存在问题，而不是状态机写错了。如果游戏引擎提供状态机编辑功能，开发相应的DEBUG功能也是很重要的。如Unity等，就会在状态机编辑器中高亮显示动画的状态和切换方向。

作业中增加了 `AnimationFSM::getCurrentStateName()` 函数，负责在DEBUG时打印日志。这样通过Play Test，就能够比较容易检验状态切换。

修订任务说明示意图存在问题的地方

观察任务说明中的示意图，我们可以发现，其中的一条路径并没有生效。而且如果生效也可能出现问题。

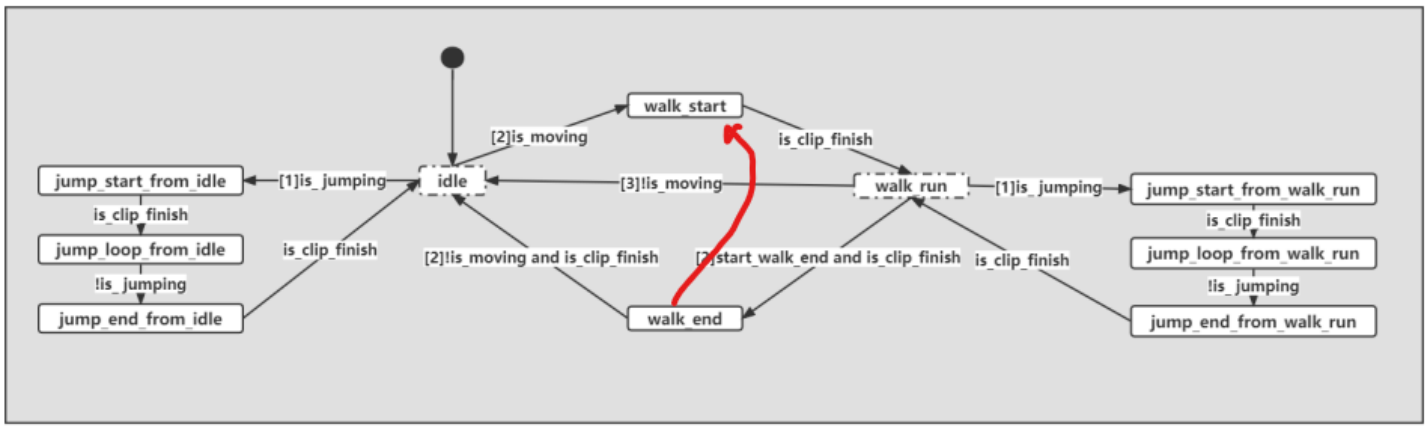
1. `start_walk_end` 为否，因此无法进入 `walk_end` 状态
2. 进入 `walk_end` 状态后，角色如果还有位移，就会一直卡在 `walk_end` 状态



为了解决上述问题，编写了相关的代码

1. 增加了 `m_move_state` 来记录角色是否在逻辑上正要移动
2. 如果逻辑上不移动了，就发出信号，使得 `start_walk_end` 为是；切换 `walk_end` 状态
3. 处于 `walk_end` 状态中时，角色又在逻辑上要移动，就切换到 `walk_start` 状态

修订后的状态机增加了一个从 `walk_end` 状态切换到 `walk_start` 状态的路径：



AnimationPose::blend

代码：

Pilot/engine/source/runtime/function/animation/pose.cpp

```
void AnimationPose::blend(const AnimationPose& pose)
{
    // TODO: PA03_1
    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 = pose.m_weight.m_blend_weight[i] + 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 lerp_ratio = cur_weight / sum_weight;

            ASSERT(lerp_ratio <= 1.0F || lerp_ratio >= 0.0F);

            // blend with lerp_ratio
            bone_trans_one.m_position = Vector3::lerp(bone_trans_one.m_position, bone_tran
            bone_trans_one.m_scale    = Vector3::lerp(bone_trans_one.m_scale, bone_trans_t
            bone_trans_one.m_rotation =
                Quaternion::nLerp(lerp_ratio, bone_trans_one.m_rotation, bone_trans_two.m_
        }
    }
}
```

实现思路：

借鉴 `AnimationPose::extractFromClip` 函数

任务说明中包括了基本的结构，并且 `AnimationPose::extractFromClip` 函数也提供了可借鉴的思路。这段程序的主要任务是计算出正确的pose骨骼权重，并使用权重进行位置、缩放、旋转的插值

1. 首先计算当前骨骼和下个待混合骨骼的合计权重
2. 判断不是0，一方面为0不需要混合，另一方面也会引起除零错误
3. 取下个骨骼的权重为当前权重，计算混合比例
4. 使用 `Vector3::lerp` 混合位置和缩放
5. 使用 `Quaternion::nLerp` 混合旋转四元数
6. 在合适的位置将累计权重和累计混合的位置、缩放、旋转更新到 `m_bone_poses` 和 `m_weight`

CharacterController::move

代码：

Pilot/engine/source/runtime/function/controller/character_controller.cpp

```
Vector3
CharacterController::move(const Vector3& current_position, const Vector3& displacement
{
    std::shared_ptr<PhysicsScene> physics_scene =
        g_runtime_global_context.m_world_manager->getCurrentActivePhysicsScene().lock(
    ASSERT(physics_scene);

    std::vector<PhysicsHitInfo> hits;

    Transform world_transform =
        Transform(current_position + 0.1f * Vector3::UNIT_Z, Quaternion::IDENTITY, Vec

    Vector3 vertical_displacement   = displacement.z * Vector3::UNIT_Z;
    Vector3 horizontal_displacement = Vector3(displacement.x, displacement.y, 0.f);

    Vector3 vertical_direction   = vertical_displacement.normalisedCopy();
    Vector3 horizontal_direction = horizontal_displacement.normalisedCopy();

    Vector3 final_position = current_position;

    m_is_touch_ground = physics_scene->sweep(
        m_rigidbody_shape, world_transform.getMatrix(), Vector3::NEGATIVE_UNIT_Z, 0.10

    hits.clear();

    world_transform.m_position -= 0.1f * Vector3::UNIT_Z;
    // vertical pass
    if (physics_scene->sweep(m_rigidbody_shape,
                             world_transform.getMatrix(),
                             vertical_direction,
                             vertical_displacement.length(),
                             hits))
    {
        DEBUG_LOG_DEBUG("vertical hit");
        final_position += hits[0].hit_distance * vertical_direction;
    }
    else
    {
        final_position += vertical_displacement;
    }

    hits.clear();

    // never count foot in floor as side hit
    world_transform.m_position += 0.05F * Vector3::UNIT_Z;
    // side pass
    float offset = 0.001F;
    if (is_jumping)
    {
        offset = 0.1F;
    }
```

```
        if (physics_scene->sweep(m_rigidbody_shape,
                                 //    /**** [0] ****/,
                                 world_transform.getMatrix(),
                                 //    /**** [1] ****/,
                                 horizontal_direction,
                                 //    /**** [2] ****/,
                                 horizontal_displacement.length() + offset,
                                 hits))
        {
            DEBUG_LOG_DEBUG("side hit");
            final_position += (hits[0].hit_distance - offset) * horizontal_direction;
        }
        else
        {
            final_position += horizontal_displacement;
        }

        return final_position;
    }
```

Pilot/engine/source/runtime/core/base/macro.h

```
#ifdef NDEBUG
#define ASSERT(statement)
#define DEBUG_LOG_DEBUG(...)
#else
#define DEBUG_LOG_DEBUG LOG_DEBUG
#define ASSERT(statement) assert(statement)
#endif
```

实现思路：

`vertical pass` 已经够给出了基本的用法，即把刚体的形状向移动方向扫，判断会不会撞到其他刚体，并记录一系列碰撞点信息。如果碰撞到，就减小实际位移的大小使得角色和其他刚体不会碰撞。

比较容易出问题的地方

- 角色可能跳入墙中。垂直向和水平向会保持在碰撞状态，因而不产生位移，play test时看起来会卡在墙中不断播放jump动画
- 角色在台阶附近跳来跳去，有可能卡脚

增加了一个DEBUG_LOG_DEBUG宏，在DEBUG状态打印碰撞状态信息，用于调试。

解决思路

1. 首先将刚体位置抬起一个小距离0.05F，这样避免胶囊体底部水平向撞地板；因为这个小的垂直距离，也使得角色在跨过一些障碍时能显得比较平顺，避免卡脚
2. 设定一个微小的水平offset，sweep时形状向运动方向多投射这个offset，这样会提前offset的量检测到可能的碰撞
3. 实际位移在角色刚体和水平障碍物之间的距离基础上再缩小offset，使得角色的胶囊体一直和水平障碍物至少能保持offset的距离，也就避免了角色进入墙中
4. 角色跳跃时，这个offset的量适当增大一些

其他

- 写作业中发现的FSM示意图问题已提交issue