Homework 3

The video is recorded with phone. My hardware does not support recording & playing simutaneously.

For full code, see the end of each task section.

Task 1

Implementation

I simply followed the FSM diagram provided.

There's one problem with FSM design: the signal `is_jumping` is linked to `is_touch_ground`. Thus when implementing the feature of walking up/down stairs, if for a split second the character is not touching ground, then the jump movement immediately starts. It looks a bit weird.

Task 1 - Code

Full Code of color_grading.frag

```
switch (m_state)
{
        case States::_idle:
        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_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;
            }
            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;
```

Task 2

Implementation

Given the definition of `Vector3::lerp(...)`, the argument `alpha` is the weight of second input. In this case, since the first input shoud be current pose, e.g. `bone_trans_one`, then `alpha` is weight of input pose `bone_trans_two`. Therefore `cur_weight` is the weight of `bone_trans_two` divided by sum.

After blending, `m_weight.m_blend_weight[i]` needs to be set to the new weight, which is `sum_weight`.

Followed the lecture videos, rotation has to be interpolated with `shortest_path=True`. Otherwise the animation can flip.

Task 2 - Code

```
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] / 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);
    }
}
```

Task 3

Implementation

Horizontal collision is divided into the following situations:

- Situation 1: walking into an object with large hit height (e.g. wall)

To implement blocking/sliding when walking into wall, the horizontal direction of movement has to change. I retrieve the horizontal part of hit directions, then substract it from original horizontal movement direction.

- Situation 2: walking into an object with small hit height
    - Situation 2.1: the object itself is high (e.g. when jumping into a wall then falling off)

      This is a special occasion happens when jumping off from a high object. In this case the character is not touching groud. Therefore, as long as `m_is_touch_ground` is false, the character should move the same as in situation 1.

    - Situation 2.2: the object itself is low (e.g. walking up stairs)

      If the height difference between hit position and character position is lower than threshold (e.g. stair height), the object is considered as a stair and the character can continue moving. Horizontal movement stays unchanged, but the vertical position needs to be set to stair height.

    - Situation 2.2.1: the character touches the stairs, but the body is not there yet.

      In this case, the real movement is the distance between `hit_position` and current position `world_transform.m_position`. Horizontal movement should stay unchanged, but vertical movement should be proportionally scaled given the real movement.

    - Situation 2.2.2: The character's movement brings the chara upstairs

      Directly move the chara to `hit_position`.
- Situation 3: not hitting any object

  Horizontal movement stays unchanged, i.e. `final_position += horizontal_displacement;`.

Task 3 - Problems

When the character walks upstairs, sometimes there's spontanenous jumping. It might be due to the FSM design.

Task 3 - Code

```
// side pass
if (physics_scene->sweep(m_rigidbody_shape,
    world_transform.getMatrix(),
    horizontal_direction,
    horizontal_displacement.length(),
    hits))
{
    // Move to opposite direction if hit
    // Hit normal is in the same direction as horizontal_direction
    Vector3 opposite_direction = Vector3();
    Vector3 hit_position       = Vector3(0, 0, -1);
    for (auto& hit : hits)
    {
```

```
        opposite_direction += hit.hit_normal;
        hit_position = (hit.hit_position.z > hit_position.z) ? hit.hit_position :
hit_position;
    }
    float hit_diff = hit_position.z - world_transform.m_position.z;
    if (hit_diff < 0.3 && hit_diff > 0 && m_is_touch_ground == true)
    {
        // Walking up a stair
        m_is_touch_ground     = true;
        float hit_distance = final_position.distance(hit_position);
        if (hit_distance > horizontal_displacement.length())
        {
            // Chara touches the stairs, but the body is not there yet.
            Vector3 real_movement = hit_position - world_transform.m_position;
            final_position += horizontal_direction *
horizontal_displacement.length();
            // Real z should be proportional to real movement
            final_position.z = world_transform.m_position.z +
            real_movement.length() / hit_distance *
        horizontal_displacement.length();
        }
        else
        {
            // When the chara is in air when going up stairs
            final_position = hit_position;
        }
    }
    else
    {
        // Hitting a wall, or jumping into a wall then falling off
        opposite_direction.z = 0;
        opposite_direction.normalise();
        horizontal_direction -= opposite_direction;

        // No need to normalize horizontal_direction, since it will hugely
increase displacement
        final_position += horizontal_displacement.length() * horizontal_direction;
    }
}
else
{
    final_position += horizontal_displacement;
}
```