

Games104_homework3_report

1. 状态机实现思路
2. AnimationPose混合
3. side pass 实现思路

1. 状态机实现思路

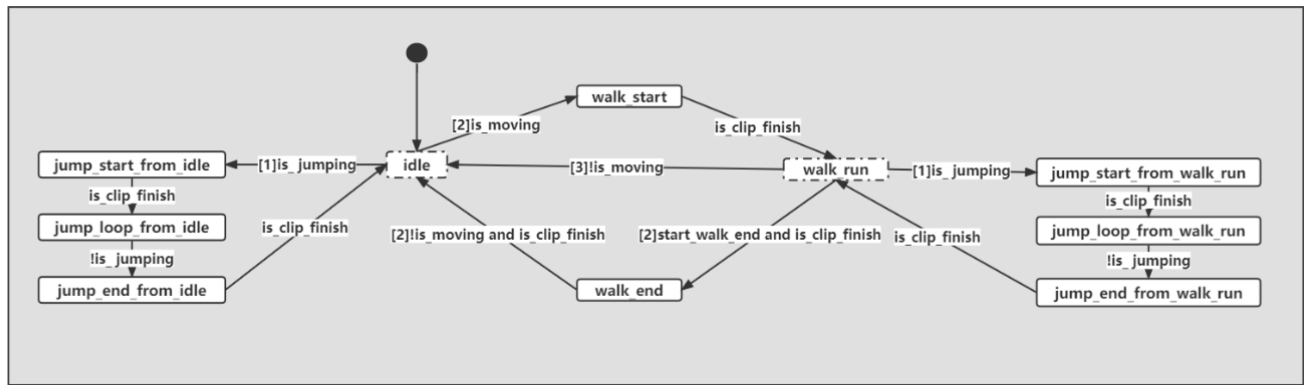
```
1  bool AnimationFSM::update(const json11::Json::object& signals)
2  {
3      States last_state    = m_state;
4      bool    is_clip_finish = tryGetBool(signals, "clip_finish", false);
5      bool    is_jumping    = tryGetBool(signals, "jumping", false);
6      float   speed         = tryGetFloat(signals, "speed", 0);
7      bool    is_moving     = speed > 0.01f;
8
9      switch (m_state)
10     {
11         case States::_idle:
12         {
13             if (is_jumping)
14             {
15                 m_state = States::_jump_start_from_idle;
16             }
17             else if (is_moving)
18             {
19                 m_state = States::_walk_run;
20             }
21             break;
22         }
23         case States::_walk_run:
24         {
25             if (is_jumping)
26             {
27                 m_state = States::_jump_start_from_walk_run;
28             }
29             else if (!is_moving)
30             {
31                 m_state = States::_idle;
32             }
33             break;
34         }
35         case States::_jump_start_from_idle:
36         {
37             if (is_clip_finish)
38             {
39                 m_state = States::_jump_loop_from_idle;
40             }
41             break;
42         }
43         case States::_jump_loop_from_idle:
44         {
45             if (!is_jumping)
```

```

46         {
47             m_state = States::_jump_end_from_idle;
48         }
49         break;
50     }
51     case States::_jump_end_from_idle:
52     {
53         if (is_clip_finish)
54         {
55             m_state = States::_idle;
56         }
57         break;
58     }
59     case States::_jump_start_from_walk_run:
60     {
61         if (is_clip_finish)
62         {
63             m_state = States::_jump_loop_from_walk_run;
64         }
65         break;
66     }
67     case States::_jump_loop_from_walk_run:
68     {
69         if (!is_jumping)
70         {
71             m_state = States::_jump_end_from_walk_run;
72         }
73         break;
74     }
75     case States::_jump_end_from_walk_run:
76     {
77         if (is_clip_finish)
78         {
79             m_state = States::_walk_run;
80         }
81         break;
82     }
83     default:
84         break;
85 }
86 return last_state != m_state;
87 }

```

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

```

1 void AnimationPose::blend(const AnimationPose& pose)
2 {
3     // Loop each bone
4     for (int i = 0; i < m_bone_poses.size(); i++)
5     {
6         auto& bone_trans_one = m_bone_poses[i];
7         const auto& bone_trans_two = pose.m_bone_poses[i];
8
9         const float& weight_one = m_weight.m_blend_weight[i];
10        const float& weight_two = pose.m_weight.m_blend_weight[i];
11        float sum_weight = weight_one + weight_two;
12
13        if (sum_weight != 0)
14        {
15            float cur_weight = weight_two / sum_weight;
16            m_weight.m_blend_weight[i] = sum_weight;
17            bone_trans_one.m_position = Vector3::lerp(bone_trans_one.m_pos
18            ition, bone_trans_two.m_position, cur_weight);
19            bone_trans_one.m_scale = Vector3::lerp(bone_trans_one.m_sca
20            le, bone_trans_two.m_scale, cur_weight);
21            bone_trans_one.m_rotation = Quaternion::sLerp(cur_weight, bone
22            _trans_one.m_rotation, bone_trans_two.m_rotation,true);
23        }
24    }
25 }

```

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

```

1      // side pass
2      if (physics_scene->sweep(m_rigidbody_shape,
3                               world_transform.getMatrix(),
4                               horizontal_direction,
5                               horizontal_displacement.length(),
6                               hits))
7      {
8          Vector3 total_normal = Vector3::ZERO;
9          for (auto it = hits.begin(); it != hits.end(); it++)
10         {
11             total_normal += (*it).hit_normal.normalisedCopy();
12         }
13         total_normal.z = 0.0f;
14
15         float sliding_distance = total_normal.crossProduct(horizontal_displacement).z;
16         Vector3 sliding_direction = Vector3(-total_normal.y, total_normal.x, total_normal.z);
17         final_position += sliding_direction * sliding_distance;
18     }
19     else
20     {
21         final_position += horizontal_displacement;
22     }

```

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_displacement` 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