

Data Driven Computer Animation

HKU COMP 3360

Tutorial 4: Real-time Character Control

Prof. Taku Komura

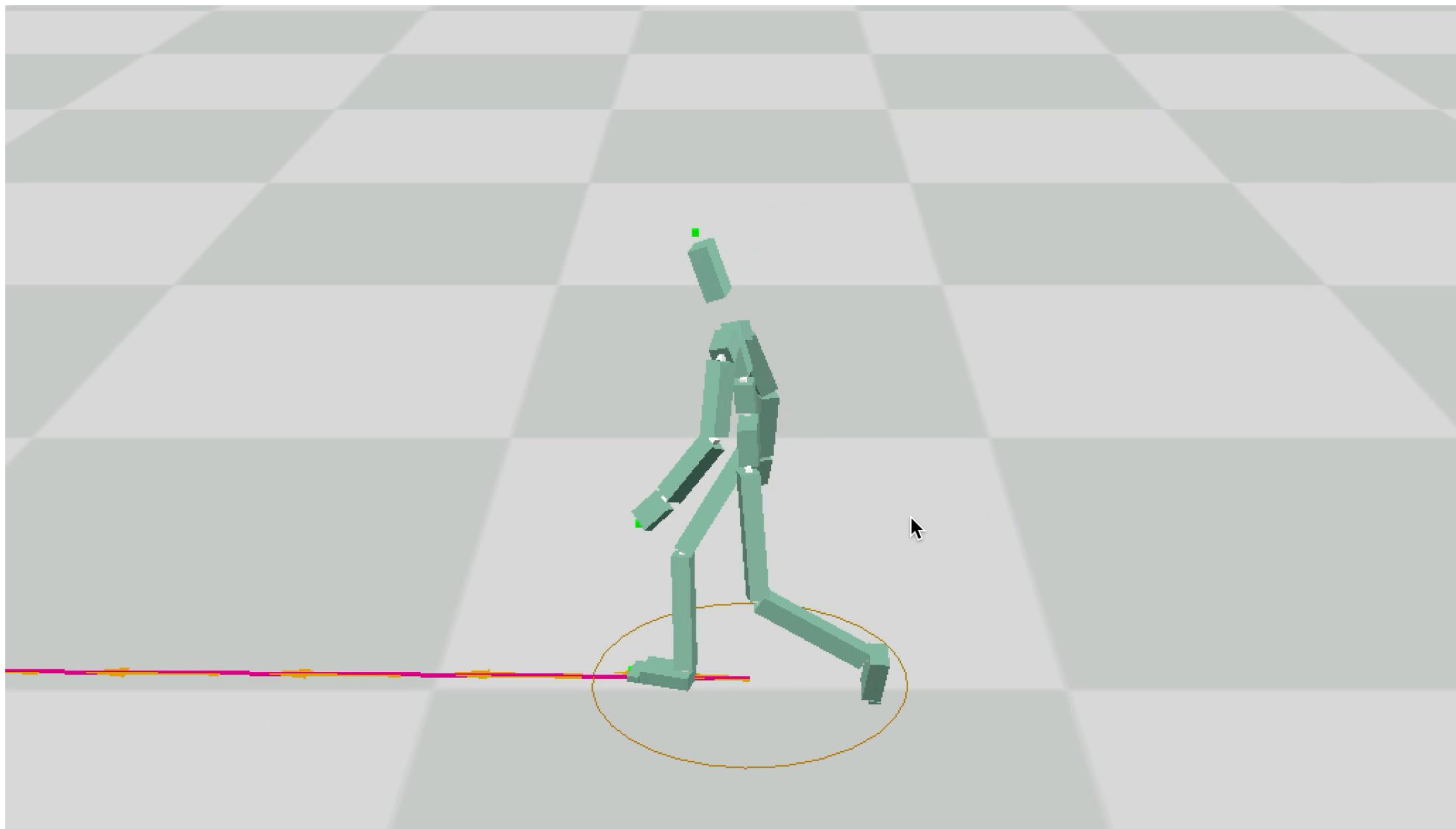
TA: Shi Mingyi (myshi@cs.hku.hk), Zhouyingcheng Liao (zliao@connect.hku.hk)

SECTION 2A, 2023



Agenda

1. The overview of real-time character control
2. State machine
3. Motion matching



Some materials are from:

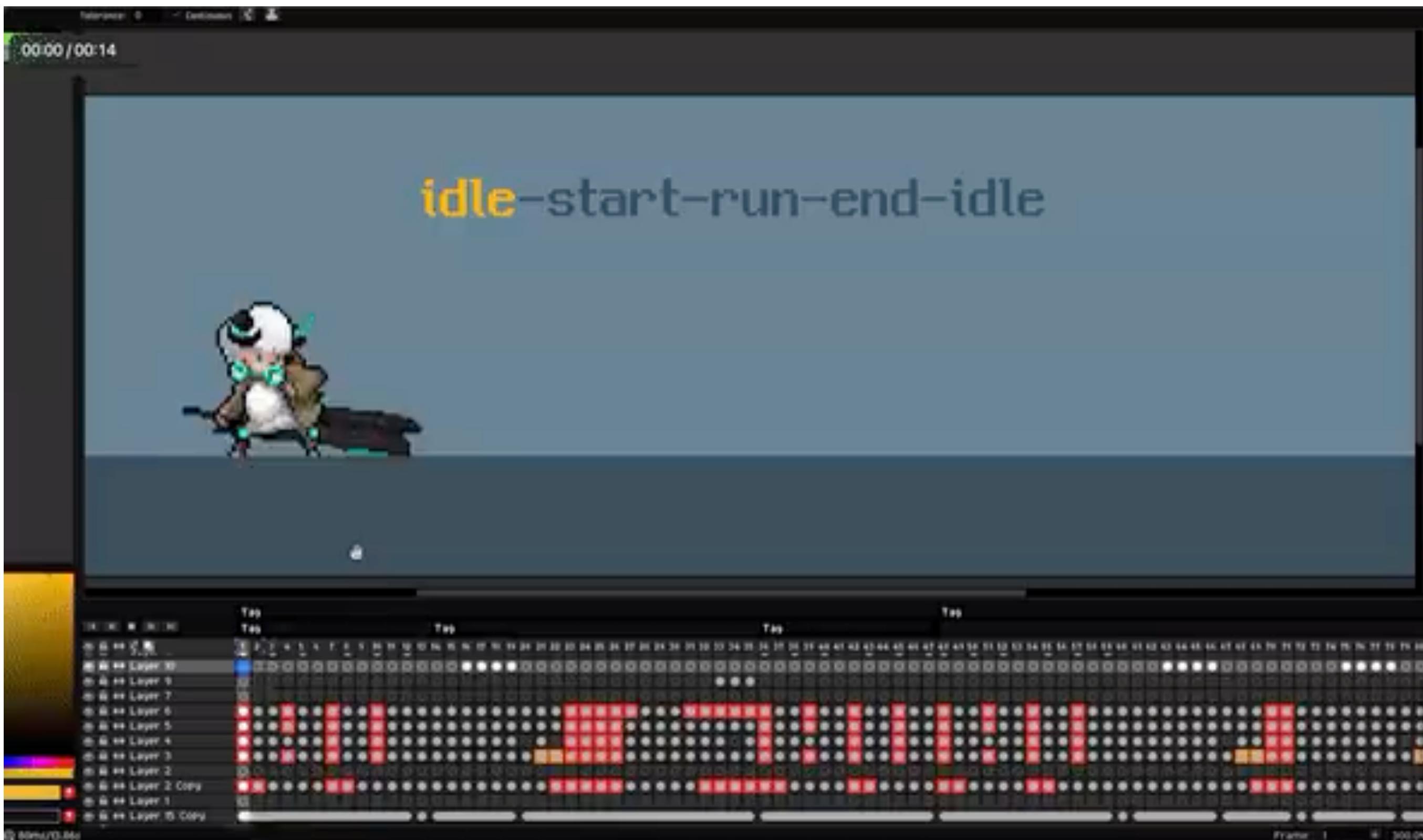


Game Developers Conference (GDC)

1. <https://www.gdcvault.com/play/1026968/-Genshin-Impact-Building-a>
2. <https://www.gdcvault.com/play/1025389/Character-Control-with-Neural-Networks>

Animation

We captured motion clips, label them, and play them in different order



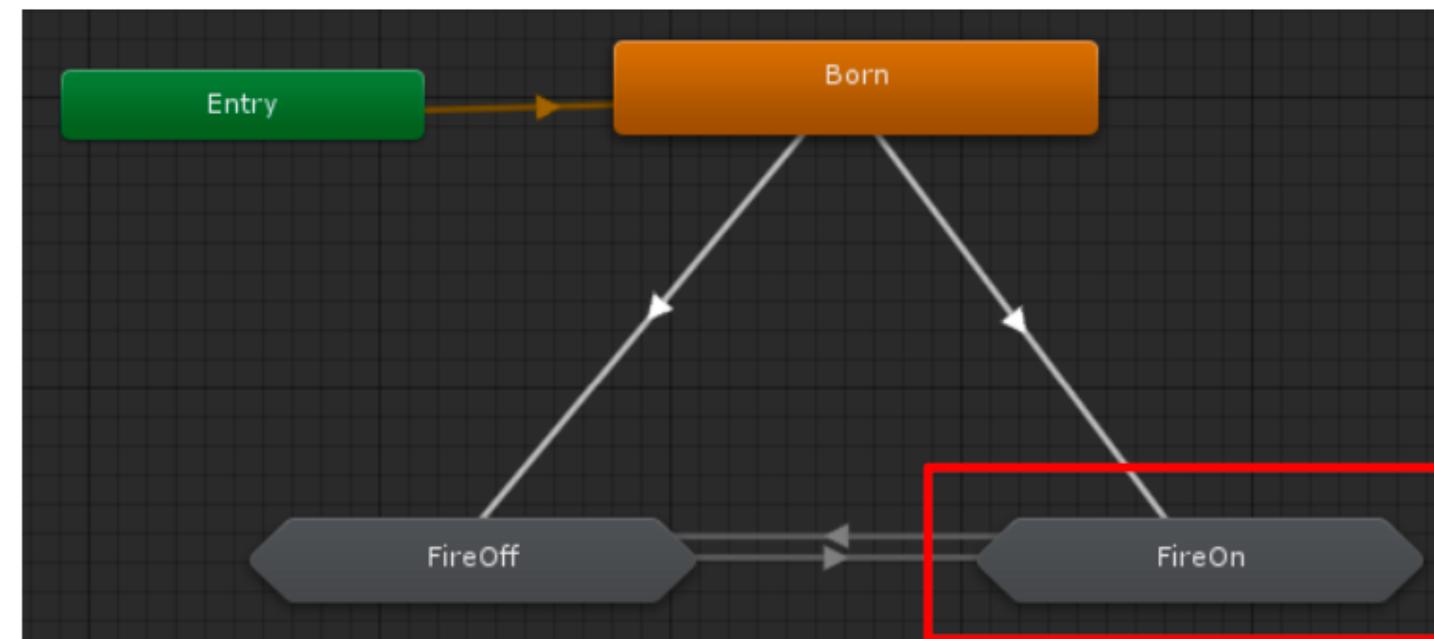
How to make the animation to be controllable?

The character should reflect to different control signals

- from keyboard
- gamepad



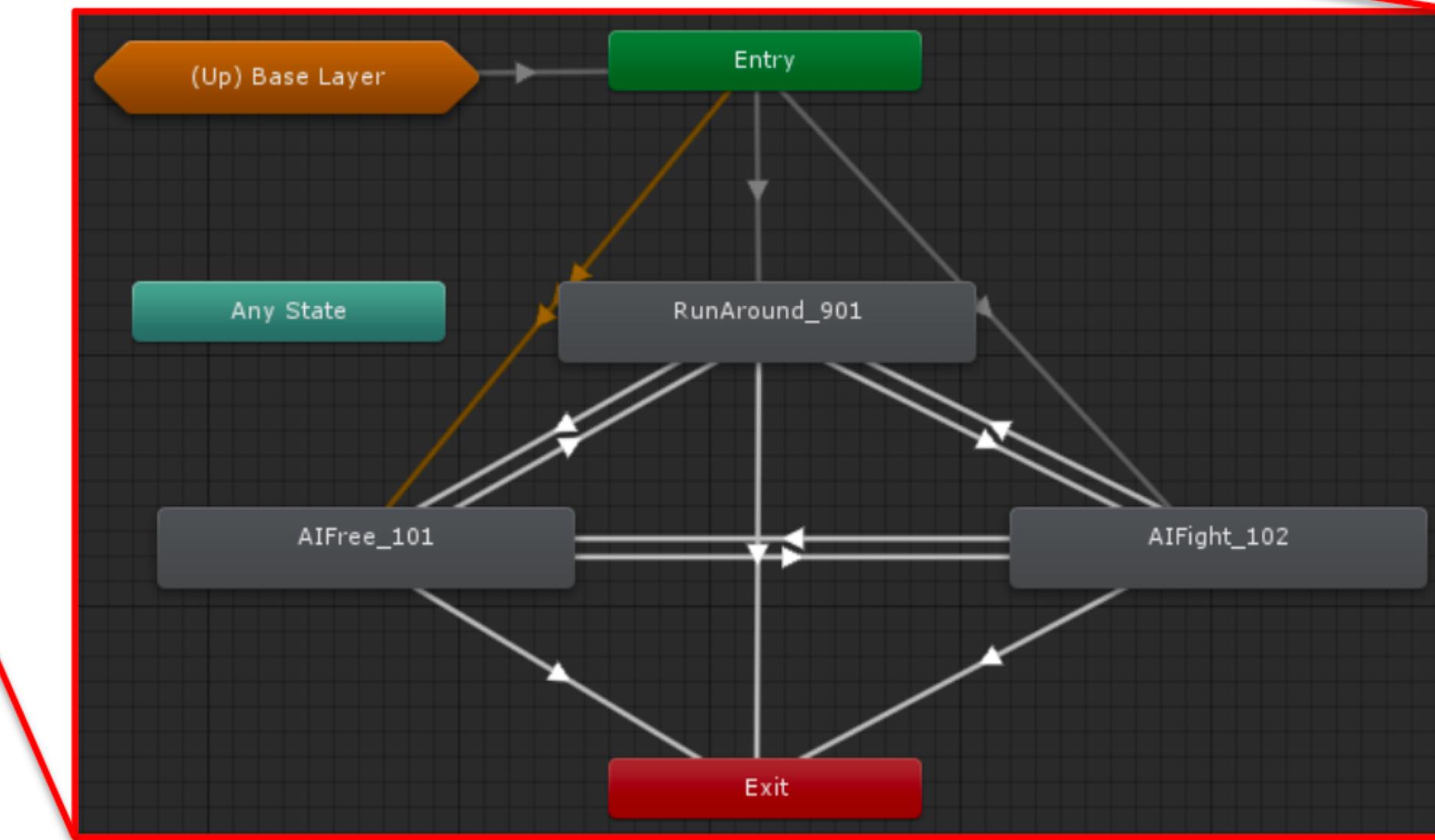
The states are managed by ‘node’



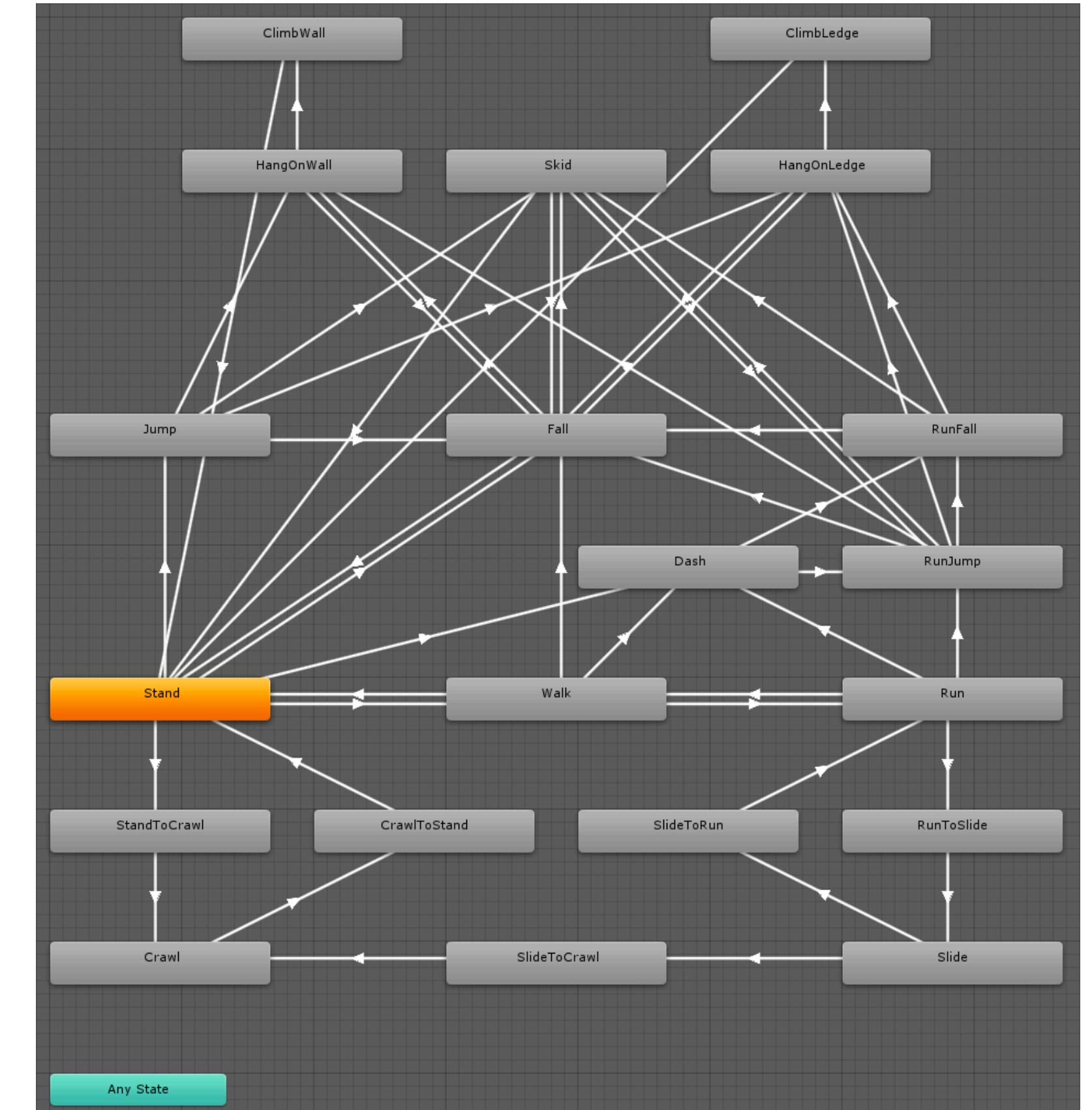
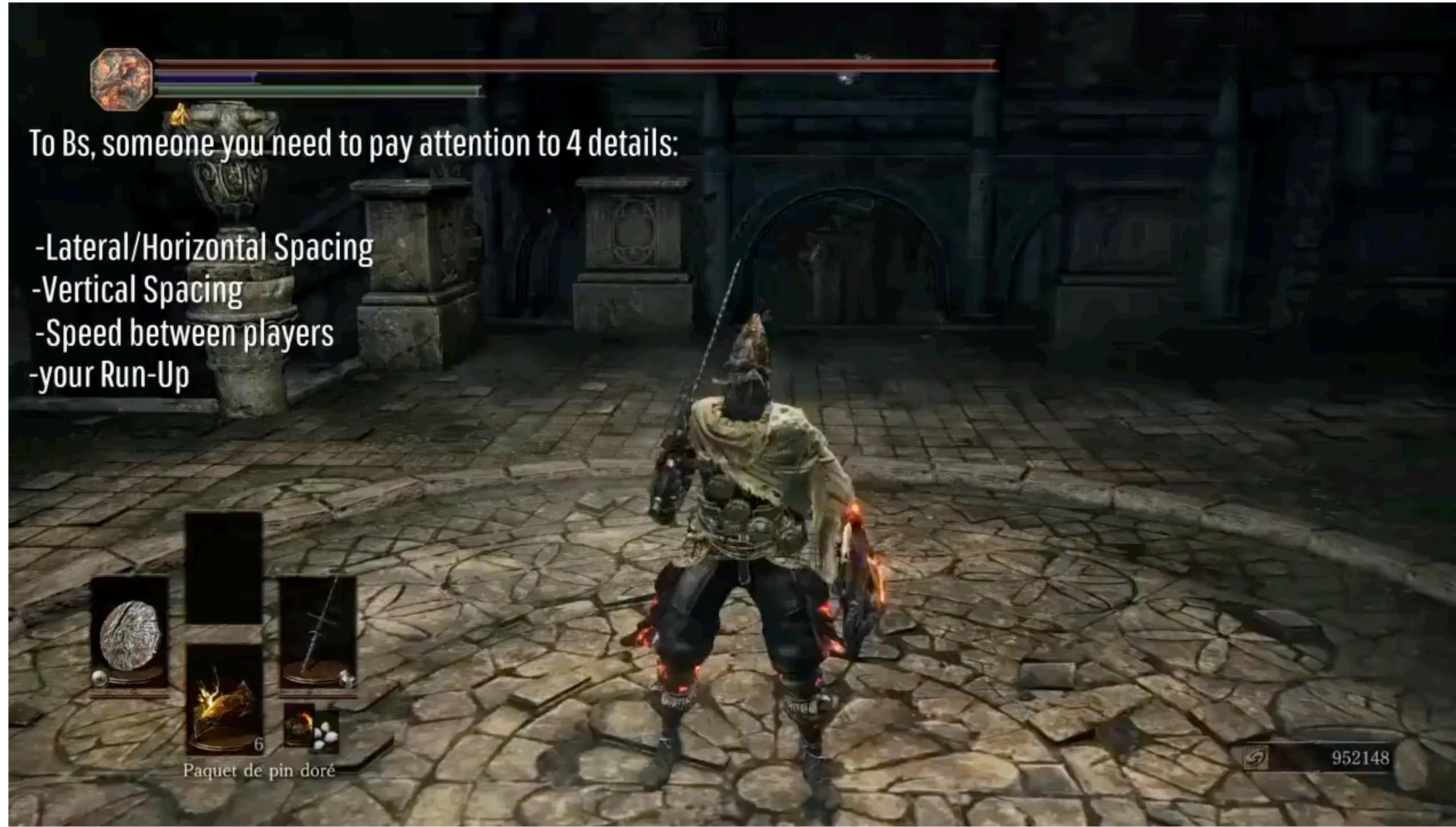
example of fire slime



- A Layered System
- Node can be connected to any other node



Node system contains lots of transition



Controlling system is a combination for lots of motion clips

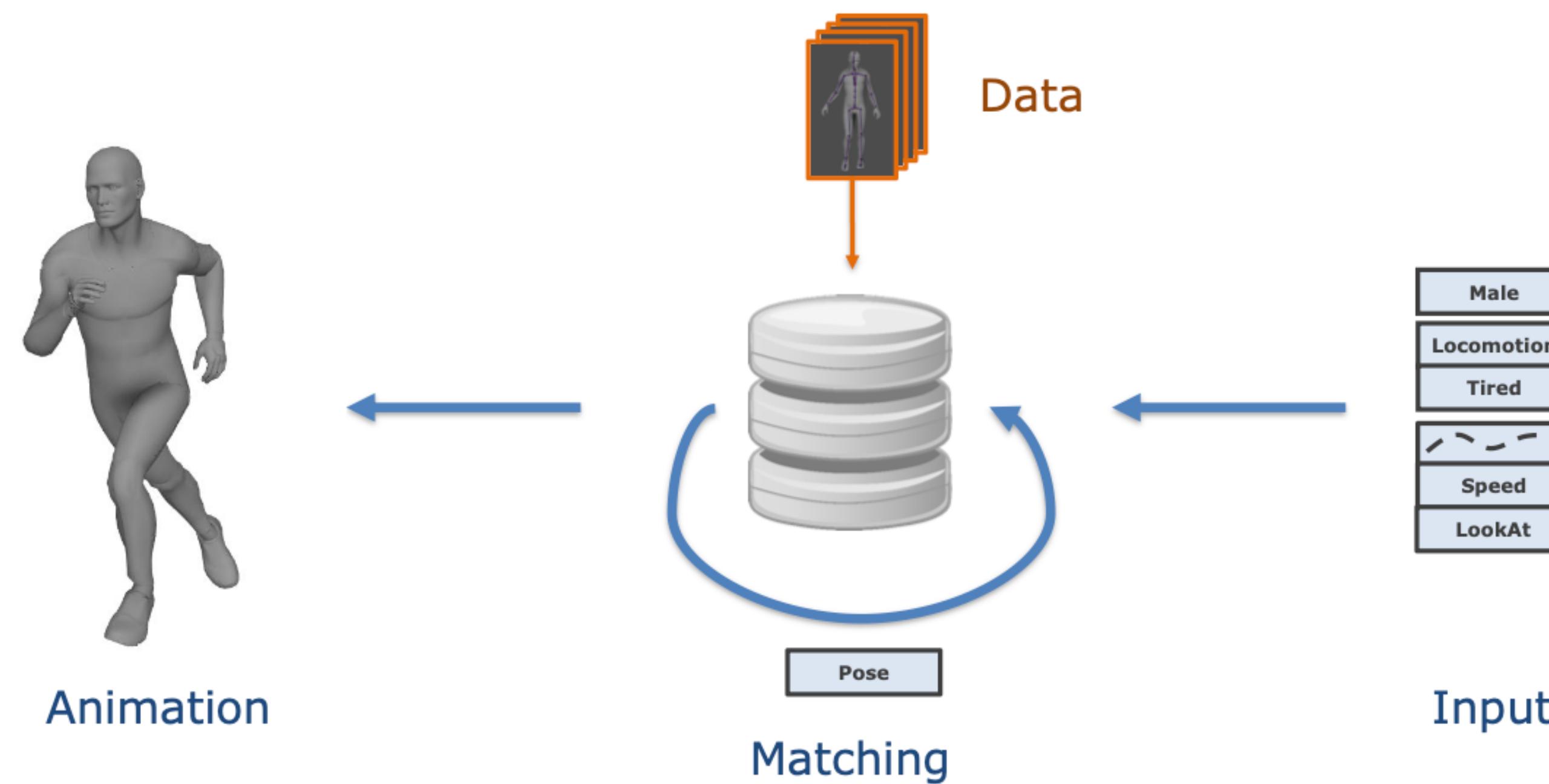
Character State Machine

GDC talk by Daniel Holden

<https://www.youtube.com/watch?v=o-QLSjSSyVk>

PDF download: <https://www.gdcvault.com/play/1025389/Character-Control-with-Neural-Networks>

Programming Details



Animation = Query(Dataset, variables)

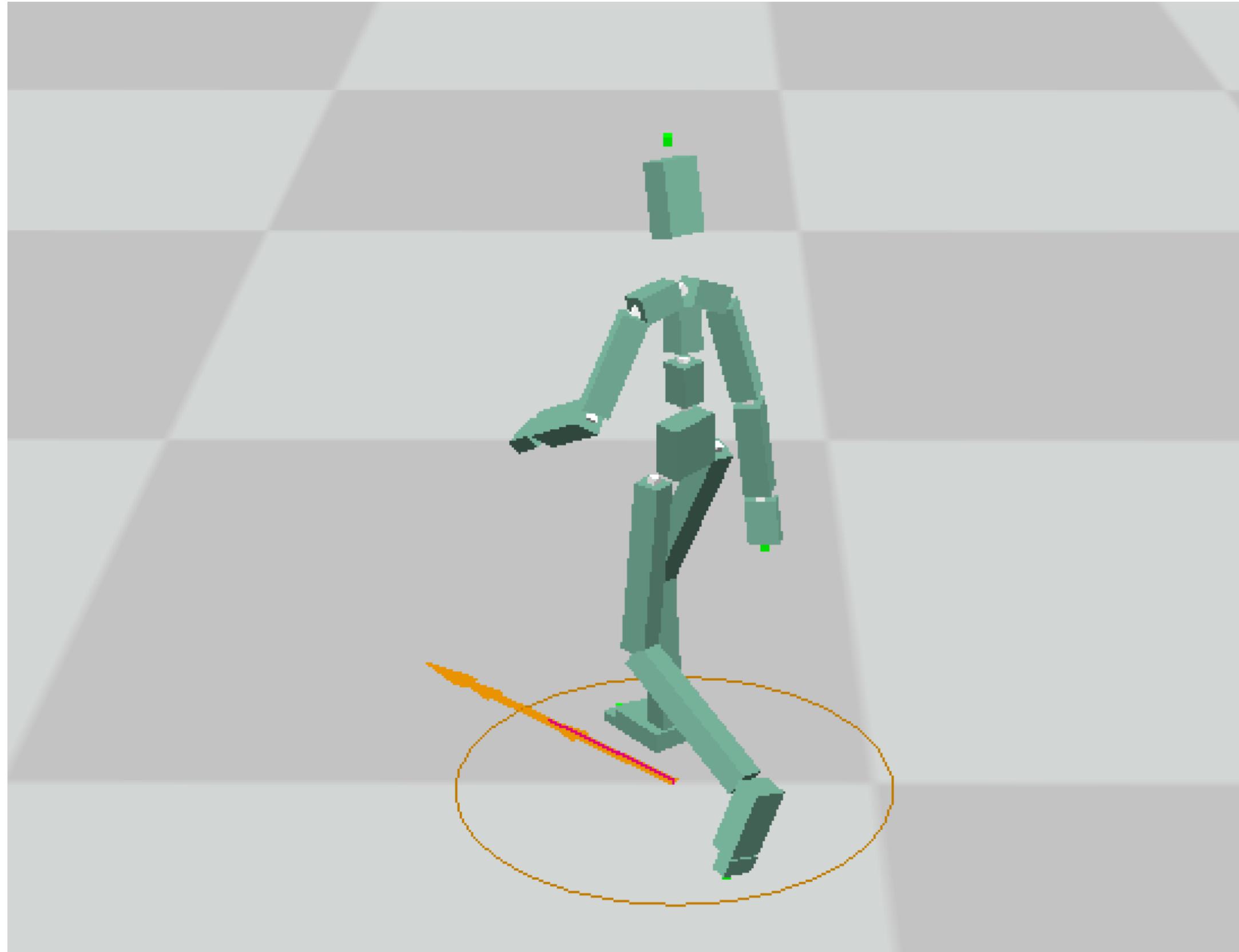
Select **animation** where **variables=xxxxx** from **dataset**

Feature Design

For a motion sequence with n frames

Each frame has its own features:

- Position
- Rotation
- Positional/Angular Velocity
- Future Position/Rotation after 20/40/60 frames

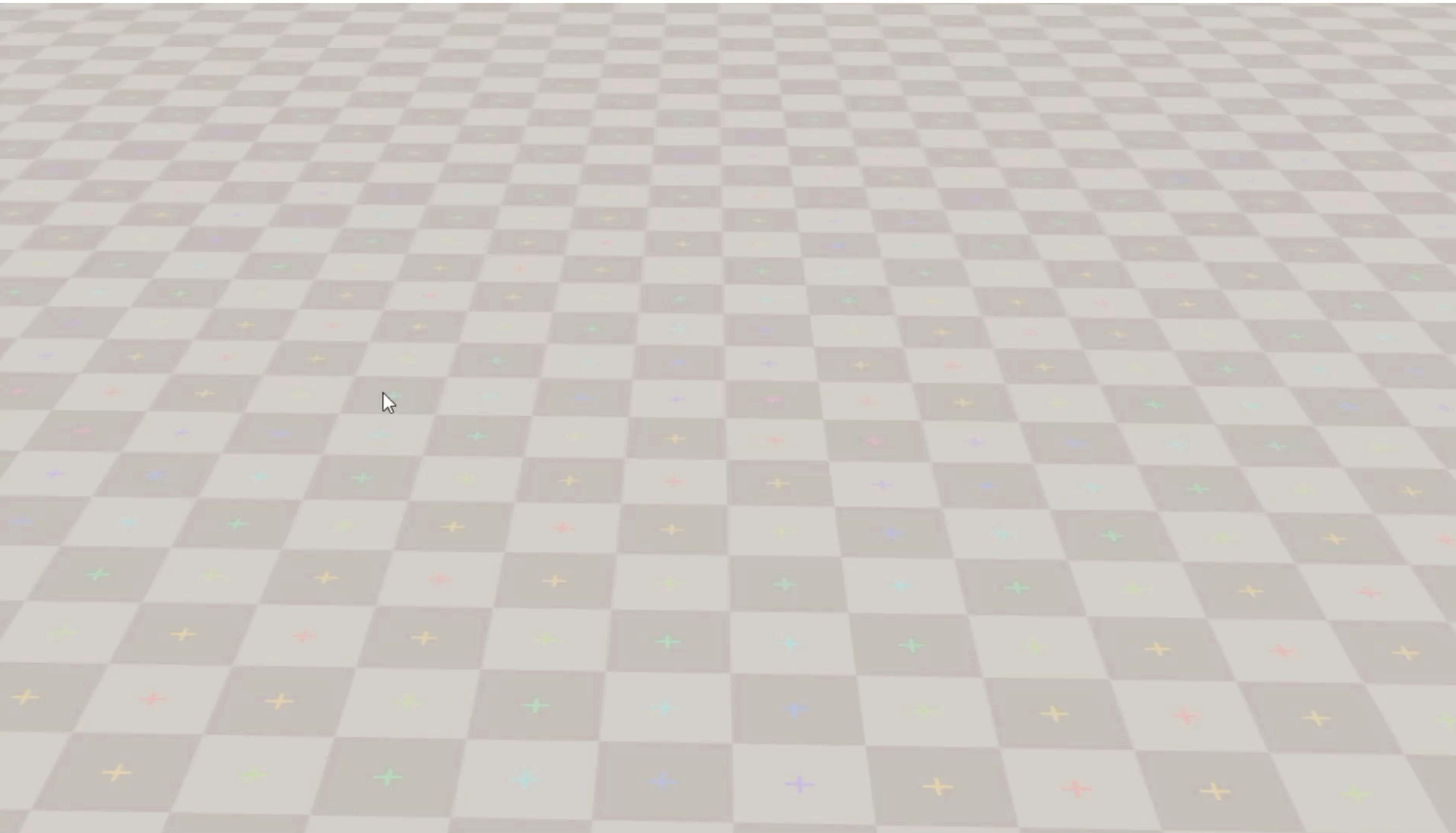


When we control the character

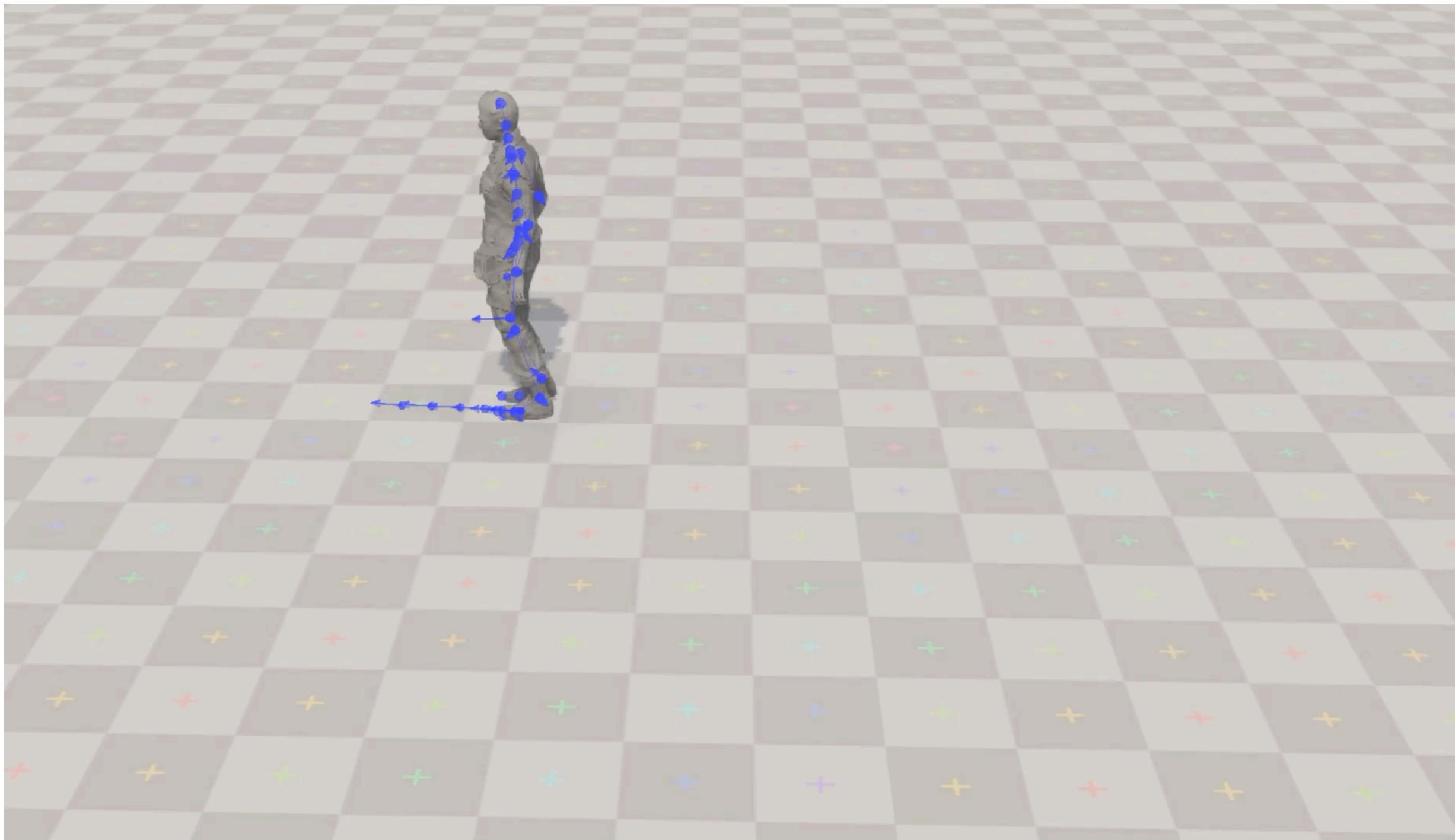
Each frame also has its own features:

- Position
- Rotation
- Positional/Angular Velocity
- Future Position/Rotation after 20/40/60 frames (decided by controller)

Step1: Give control signals



Step2: Analyze the motion data



```
94     # Extract the data terms
95     pos, rot = forward_kinematics_with_channel(self.joint_parent, self.joint_channel, sel
96     rot = align_quat(rot, False)
97     vel = np.zeros_like(pos)
98     vel[1:] = (pos[1:] - pos[:-1])/self.dt
99     vel[0] = vel[-1]
100    avel = np.zeros_like(vel)
101    avel[1:] = quat_to_avel(rot, self.dt)
102    avel[0] = avel[-1]
```

We know:

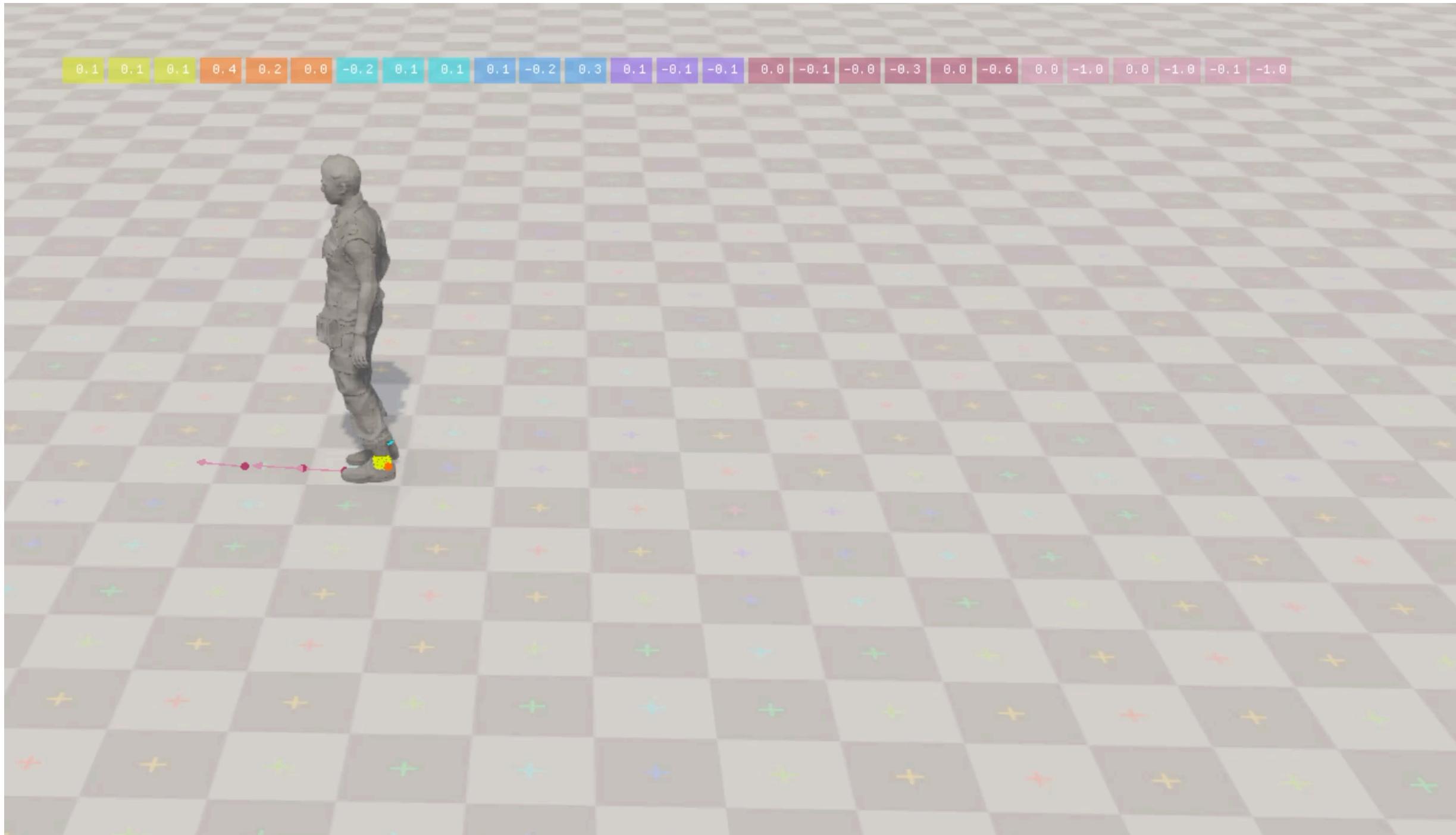
- 1.Joint position
- 2.Joint velocity
- 3.Joint rotations
- 4.Joint angular velocity

...

relative position
waving timing

...

Step3: Design the matching variables

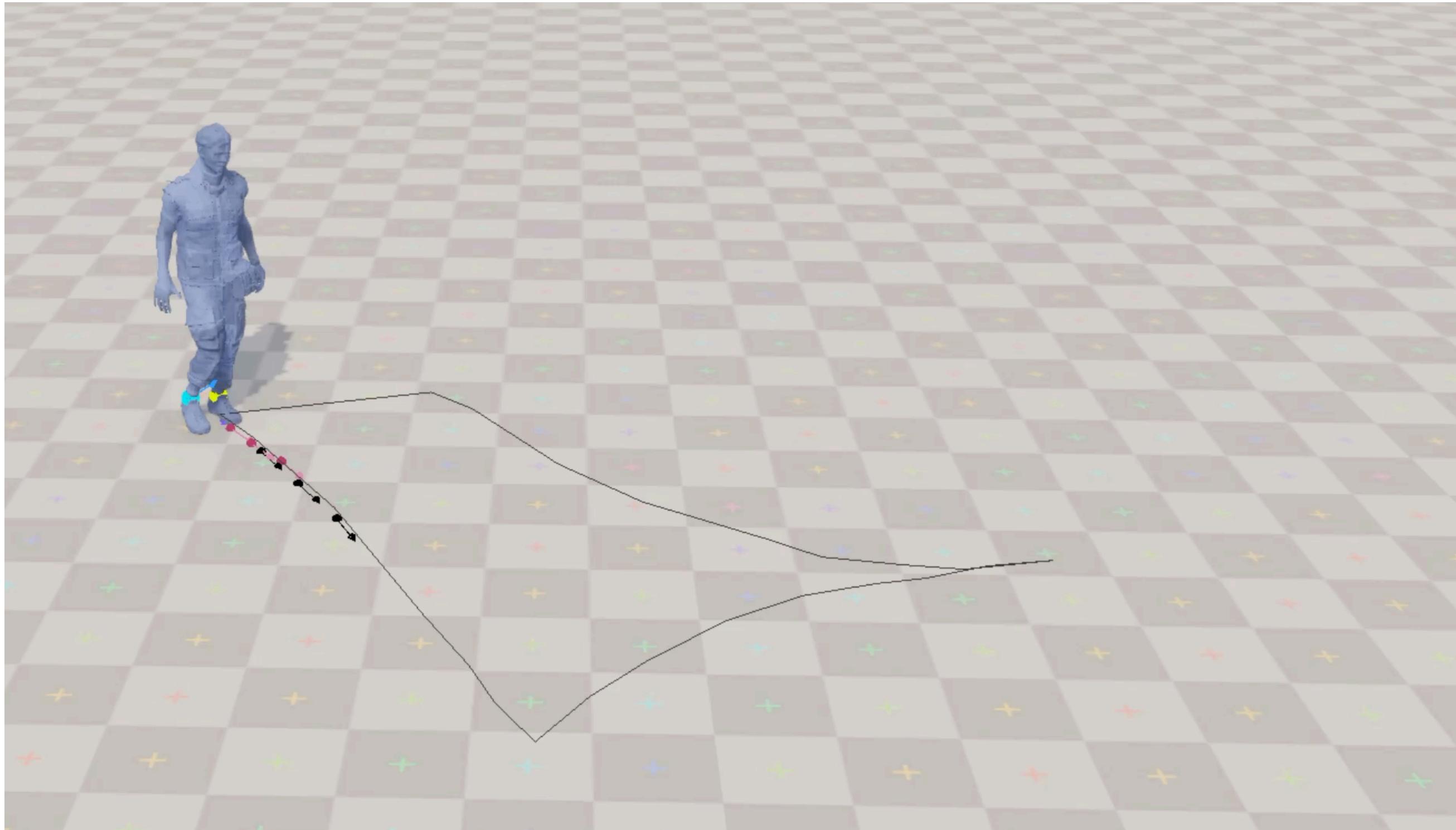


```
19 How can we use these data to match the motion?  
20 ...  
21 feature_mapping = {  
22     'lFootPos': 3,  
23     'rFootPos': 3,  
24     'lFootVel': 3,  
25     'rFootVel': 3,  
26     'lFootRot': 3,  
27     'rFootRot': 3,  
28     'rKneeAVel': 3,  
29     'lKneeAVel': 3,  
30     'lHandPos': 3,  
31     'lHandVel': 3,  
32     'rHandPos': 3,  
33     'rHandVel': 3,  
34     'rKneeAVel': 3,  
35     'lKneeAVel': 3,  
36     'rHipPos': 3,  
37     'lHipPos': 3,  
38     'rHipVel': 3,  
39     'lHipVel': 3,  
40     'hipVel': 3.
```

```
158 for feature_name in self.feature_names:  
159     ...  
160     Extract the position:  
161         features.append(self.extract_offset(root_pos))  
162     Extract the rotation:  
163         features.append(self.extract_rotation(root_rot))  
164     Extract the velocity:  
165         features.append(self.extract_vel(root_vel))  
166     Extract the future position:  
167     ...  
168 ##### Code Start #####  
169  
170     if feature_name == 'lFootPos':  
171         features.append()  
172     elif feature_name == 'lFootVel':  
173         features.append()  
174     elif feature_name == 'trajectoryPos2D':  
175         features.append()  
176     ##### Code End #####  
177  
178     # Extract the future position  
179     # Extract the rotation  
180     # Extract the velocity  
181     # Extract the future position
```

Not all the variables is needed
Find a balance between speed and accuracy

Step4: Matching



```
259 |     normalized_query = self.db.normalize_features(query_feature)
260 |
261 |     # Do the query
262 |     idx = self.db.query_tree.query(normalized_query.reshape(1,-1), k=1)[1][0]
```

Because we only use small dataset
We also store it with OC-tree
So the searching is fast

Programming Details

Prepare:

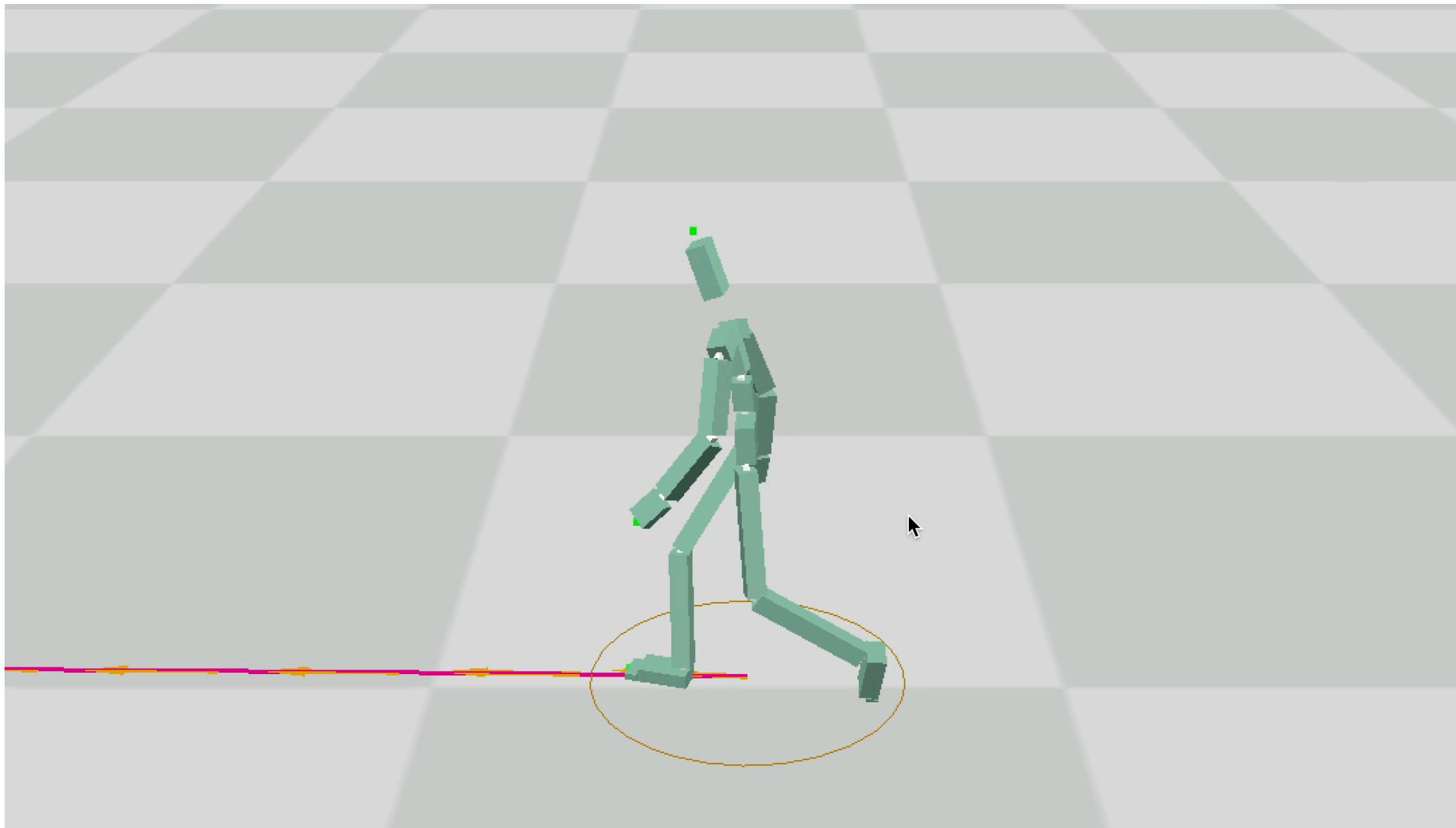
- A motion dataset
 - Oc-tree to accelerate the searching
- A set of feature variables
- A animation system to play the animation

The dataset and animation are prepared in this assignments

you only need to care the variables design

Desired Results

1. Less variables, and better performance (total 15%, 22% - your_variable_num)
2. System analyzation (10%) about variable selection, future frame range, etc.



Overview

- part1_key_framing (30%)
 - - Linear interpolation (10%); Slerp Interpolation (15%)
 - - Report the different performance by giving different numbers (5%)
- part2_concatenate (35%)
 - - Define the search window (10%); Calculate the sim_matrix (10%); Find the real_i and real_j (10%); The shifting on the root joint position (5)
- part3_motion_matching (25%)
 - variable terms (15% <- 22% - your_variable_num) + System analyzation (10%)
- Report (8%) + 2 videos (2%)
 - Including necessary experiment results by **different parameters (4%)** and **your thinking(4%)** for how to produce high quality motions