
互動程式設計III
Interactive Programming Design Integration

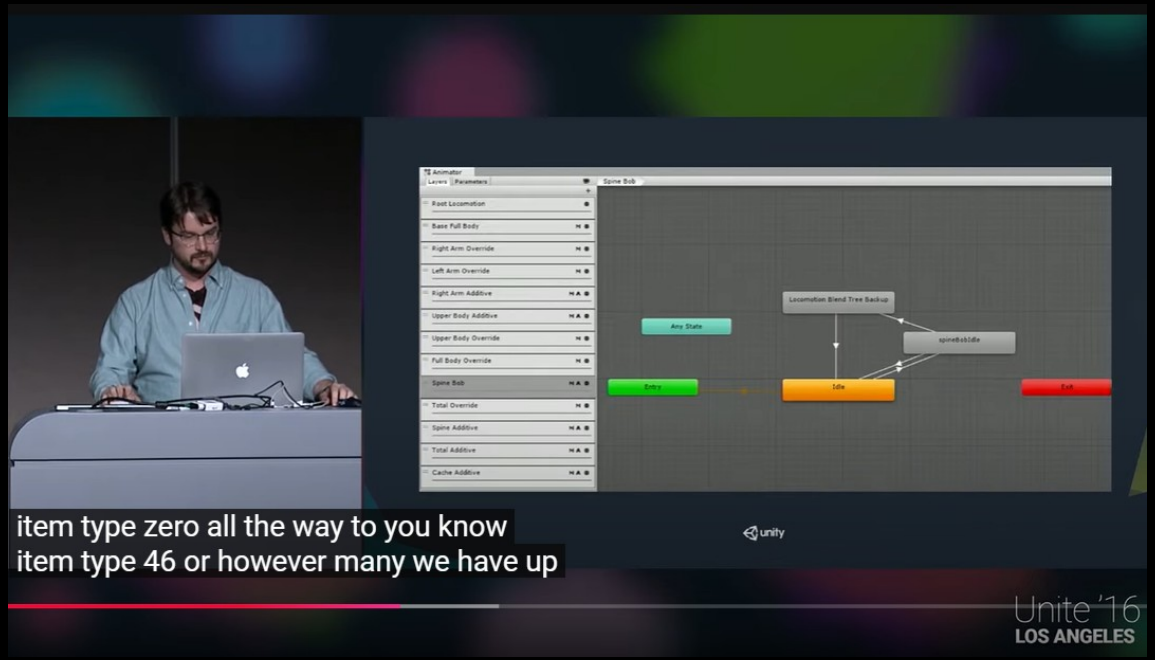
互動程式設計III

Interactive Programming Design Integration

- Player Control
  - FSM + Animator
  - Fall State
  - Roll State
  - Locomotion State
  - Ground Sensor

Problem #1 – Animator is Prone to be Over-designed

- The more delicate you want your character to be, the more easily its controller will be over-designed.
- Watch this video 12:02 ~ 13:34
  - What is the problem this structure solved?
  - What is the problem this structure generated?



https://youtu.be/8VgQ5PpTqjc?si=hP7ztP5KvtrEUTlE&t=722

Problem #2 - FSM Synchronization

- Basic usage: simple C# state-less logic + Animator FSM
  - Easily to get character twitch/mulfunction. Hard to debug.
- Advanced usage: C# FSM + Animator FSM
  - Super hard to design, maintain and edit.
- Blending of the 2 method: C# FSM + transition-less Animator
  - Take Unity animator as a collection of animations.

Even if you designed a bug-free animator, you still need to adequately sync it with your character interaction logic.

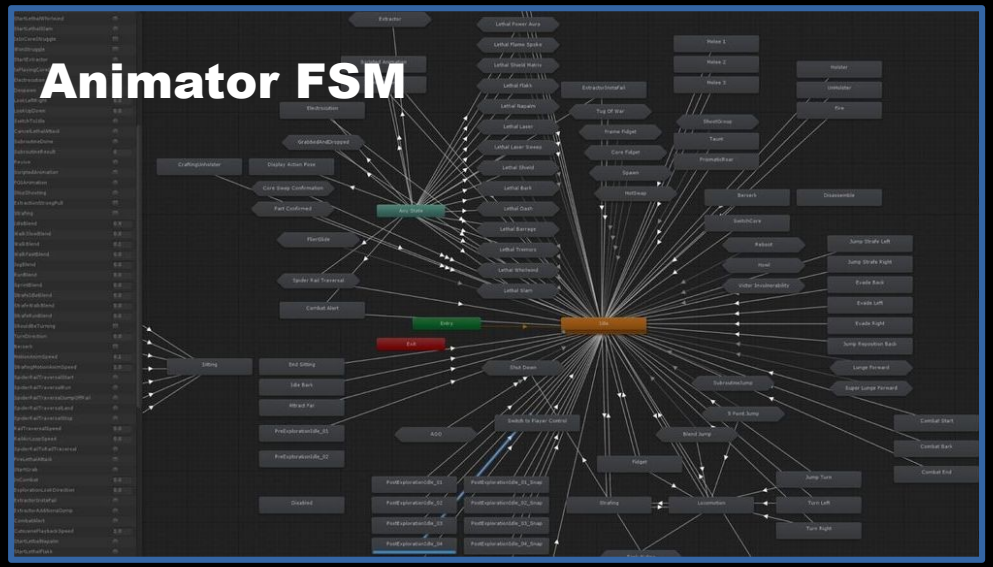
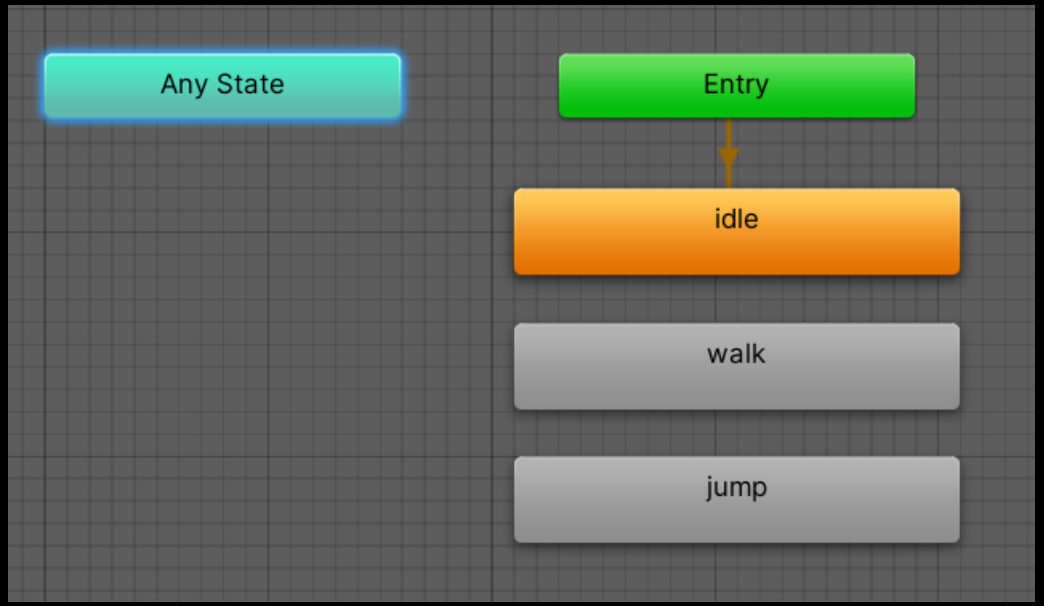


Image source:
https://www.rokoko.com/insights/get-started-with-unity-character-animation

Transition-less Animator

- Drag all the required animation into the animator.
- Make sure the name is corrected (should not be mixamo.com).
- No need to wire any transition.



IDLE/WALK/JUMP Logic

- Define a enum with 3 states in PlayerController class: IDLE, WALK, JUMP. ❶
- Add a tiggerEnter flag for OnEnter logic. ❷
- Initialize the newVelocity to zero. ❸
- calculate movingVecH/V. ❹
- Regulate the movingVec to pointing to either horizontal or vertical. ❺

```
1   using UnityEngine;
2
3   public class PlayerController : MonoBehaviour
4   {
5       public enum STATE {
6           IDLE,
7           WALK,
8           JUMP
9       }
10      [SerializeField]
11      private STATE state;
12
13      public float velocity = 3.0f;
14      public float jumpThrust = 3.0f;
15      public GameObject model;
16
17      private Vector3 movingVec;
18      private Animator anim;
19      private Rigidbody rigid;
20      [SerializeField]
21      private bool isThrust = false;
22      private bool triggerEnter = false;
23
24      void Awake() {
25          anim = model.GetComponent<Animator>();
26          rigid = GetComponent<Rigidbody>();
27          state = STATE.IDLE;
28      }
29
30      void Update() {
31          // Initialize newVelocity to zero.
32          Vector3 newVelocity = Vector3.zero;
33          // Moving Vector calculation.
34          float movingVecH = Vector3.Dot(movingVec, Vector3.right);
35          float movingVecV = Vector3.Dot(movingVec, Vector3.forward);
36
37          // Restrict to 4 direction movements.
38          if (Mathf.Abs(movingVecH) >= Mathf.Abs(movingVecV)) {
39              movingVec = movingVecH * Vector3.right;
40          }
41          else {
42              movingVec = movingVecV * Vector3.forward;
43          }
```

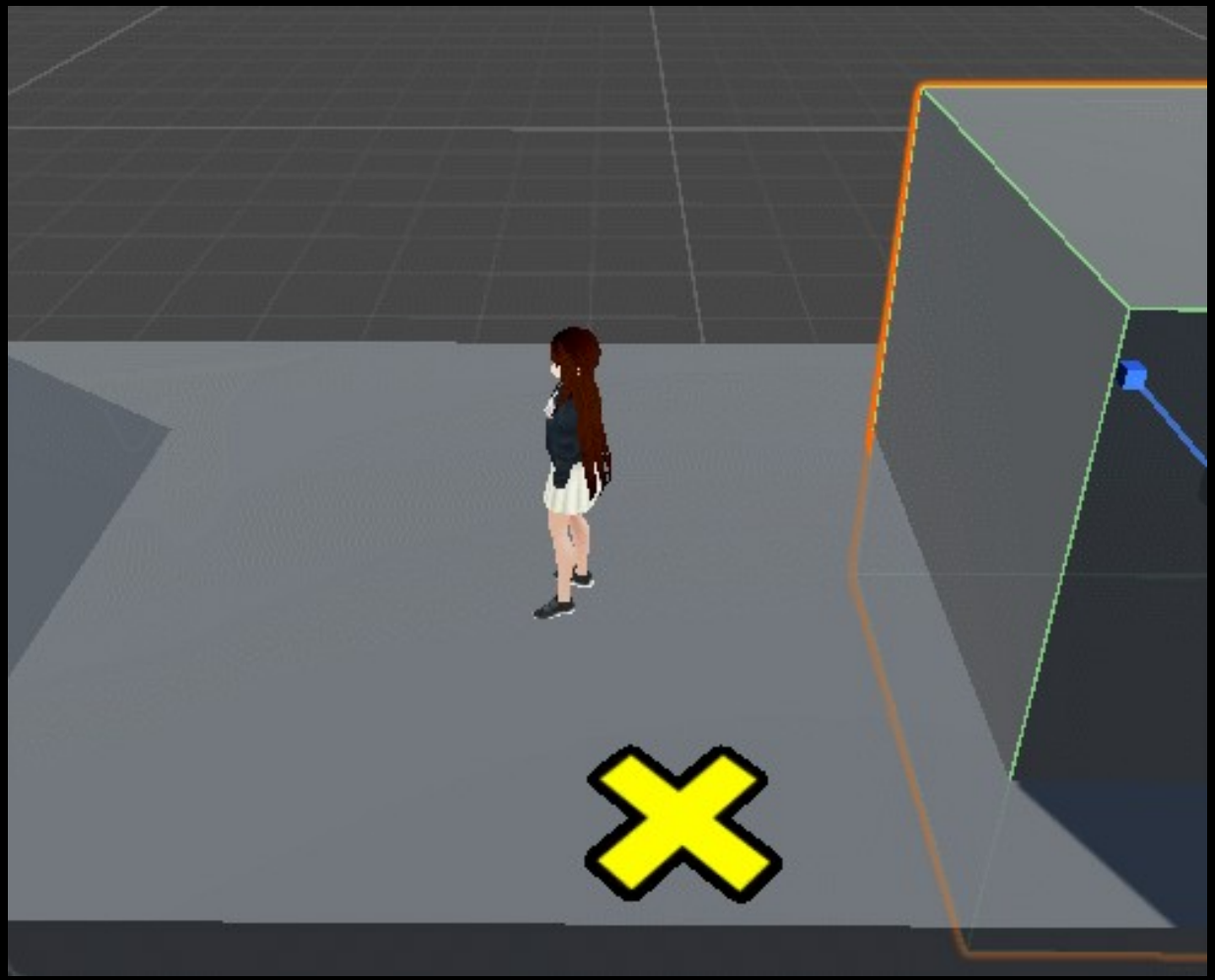
- The huge switch
  - Use anim.CrossFadeFixedTime()[*] instead of anim.Play() to blend into target animation. ❶
  - Recover the triggerEnter in side OnEnter codes. ❷
  - Remember to set newVelocity to rigid.velocity in both of the OnEnter and IsState parts in JUMP state❸. Otherwise newVelocity will be asigned to zero by line#32. And the player will lose all planar velocity when it start jumping.

```
44
45          switch (state) {
46              case STATE.IDLE:
47                  if (triggerEnter) {
48                      anim.CrossFadeInFixedTime("idle",0.1f);
49                      triggerEnter = false;
50                      break;
51                  }
52                  if (movingVec.magnitude > 0.1f) {
53                      GoToState(STATE.WALK);
54                      break;
55                  }
56                  newVelocity = Vector3.zero;
57                  //anim.SetBool("isWalking", false);
58                  break;
59              case STATE.WALK:
60                  if (triggerEnter) {
61                      anim.CrossFadeInFixedTime("walk", 0.1f);
62                      triggerEnter = false;
63                      break;
64                  }
65                  if (movingVec.magnitude <= 0.1f) {
66                      GoToState(STATE.IDLE);
67                      break;
68                  }
69                  //anim.SetBool("isWalking", true);
70                  // Calculate new velocity
71                  newVelocity = movingVec * velocity;
72                  model.transform.forward = Vector3.Slerp(
73                      model.transform.forward, movingVec, 0.1f);
74                  break;
75              case STATE.JUMP:
76                  if (triggerEnter) {
77                      anim.CrossFadeInFixedTime("jump", 0.1f);
78                      triggerEnter = false;
79                      newVelocity =rigid.velocity;
80                      break;
81                  }
82                  newVelocity = rigid.velocity;
83                  break;
84              default:
85                  break;
86          }
```

- Comment out all the animator parameter get/set. We are going to use CrossFadeFixedTime() to assign animation directly. ❶

```
87          // Applying thrust
88          newVelocity.y = rigid.velocity.y + (isThrust ? 1.0f : 0) * jumpThrust;
89          rigid.velocity = newVelocity;
90          isThrust = false;
91      }
92
93      private void GoToState(STATE targetState) {
94          state = targetState;
95          triggerEnter = true;
96      }
97
98      public void Move(Vector3 vector) {
99          movingVec = vector;
100     }
101
102     public void Jump(bool _isThrust) {
103         if (_isThrust) {
104             if (state == STATE.IDLE || state == STATE.WALK) {
105                 isThrust = true;
106                 GoToState(STATE.JUMP);
107                 //anim.SetTrigger("triggerJump");
108                 //anim.SetBool("isGround", false);
109             }
110         }
111     }
112
113     public void OnCollisionEnter(Collision collision) {
114         GoToState(STATE.IDLE);
115         //anim.SetBool("isGround",true);
116     }
117 }
118
```

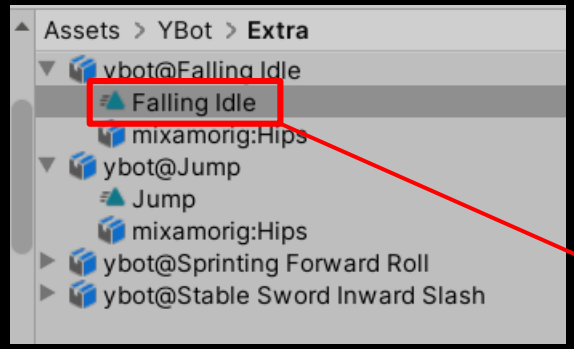
- Try it. It should work like before.

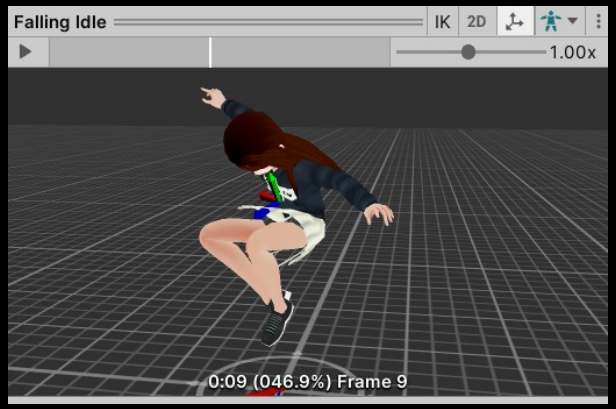


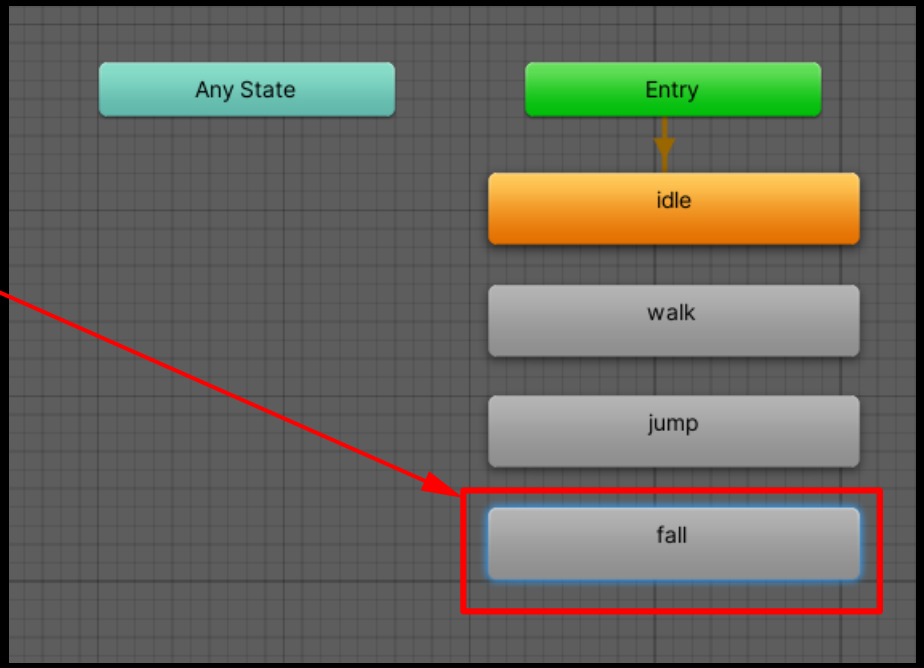
- Player Control
  - FSM + Animator
  - Fall State
  - Roll State
  - Locomotion State
  - Ground Sensor

Add a Fall State

- When jump action is finished but the player is still in air, the player controller should enter a FALL state.
- Find the falling animation "Falling Idle" and drag it into the animator AC.
  - This will create a new state. Name it "fall" state.







FALL Logic#1 - AnimatorStateInfo

- We are going to monitor the progress of jump animation.
- Method#1: get the current progress in Player Controller.
  - Get the state info by anim.GetCurrentAnimatorStateInfo(). [*]
  - Read current progress from stateInfo.normalizedTime.[*]
  - Compare current state name by stateInfo.IsName().[*]
- Add enum value for FALL state. ❶
- Get current animator state info. ❷
- If normalized time is greater than 80% (0.8f), and current state is "jump", then switch to FALL state. ❸
- Add logic for FALL state. ❹
- Add OnCollisionEnter to detect ground touching. ❺

```
using Unity.VisualScripting;
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    public enum STATE {
        IDLE,
        WALK,
        JUMP,
        FALL
    }
```

❶

```
    void Update() {
        // Initialize newVelocity to zero.
        Vector3 newVelocity = Vector3.zero;
        // Moving Vector calculation.
        float movingVecH = Vector3.Dot(movingVec, Vector3.right);
        float movingVecV = Vector3.Dot(movingVec, Vector3.forward);
        AnimatorStateInfo stateInfo;
```

❷

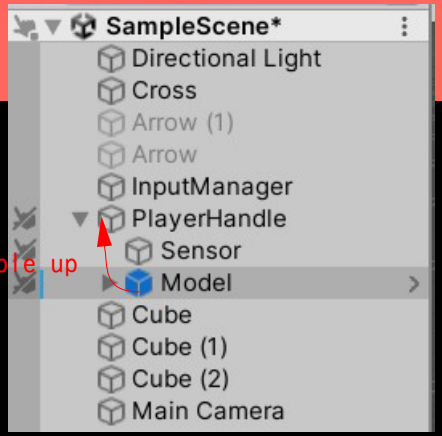
```
        break;
    case STATE.JUMP:
        if (triggerEnter) {
            anim.CrossFadeInFixedTime("jump", 0.1f);
            triggerEnter = false;
            newVelocity =rigid.velocity;
            break;
        }
        newVelocity = rigid.velocity;
        stateInfo = anim.GetCurrentAnimatorStateInfo(0);
        if (stateInfo.normalizedTime > 0.8f && stateInfo.IsName("jump")) {
            GoToState(STATE.FALL);
        }
        break;
    case STATE.FALL:
        if (triggerEnter)
        {
            anim.CrossFadeInFixedTime("fall", 0.1f);
            triggerEnter = false;
            newVelocity = rigid.velocity;
            break;
        }
        newVelocity = rigid.velocity;
        break;
```

❸ ❹

```
    public void OnCollisionEnter(Collision collision) {
        GoToState(STATE.IDLE);
        //anim.SetBool("isGround",true);
    }
```

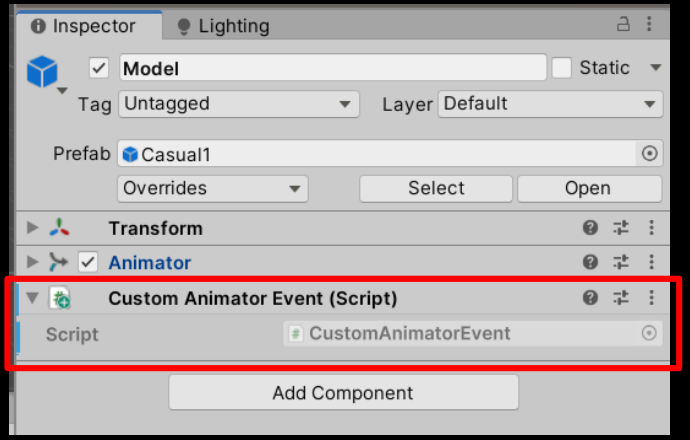
Fall Logic#2 - Animation Event

- Method#2: Add an animator event: When the event is triggerd, the sibling monobehaviour component next to the Animator component will receive a message with the same name.
  - Add a new C# script on Model called CustomAnimationEvent.cs. ❶
  - Just bubble up the message from Model to Player Handle.
    - You can't keep the same name of this animation event. So inside OnJumpEvent we call another OnJumpEnd in the parent gameObject. ❷
  - Write a method called OnJumpEnd() in Player Controller. ❸



❷

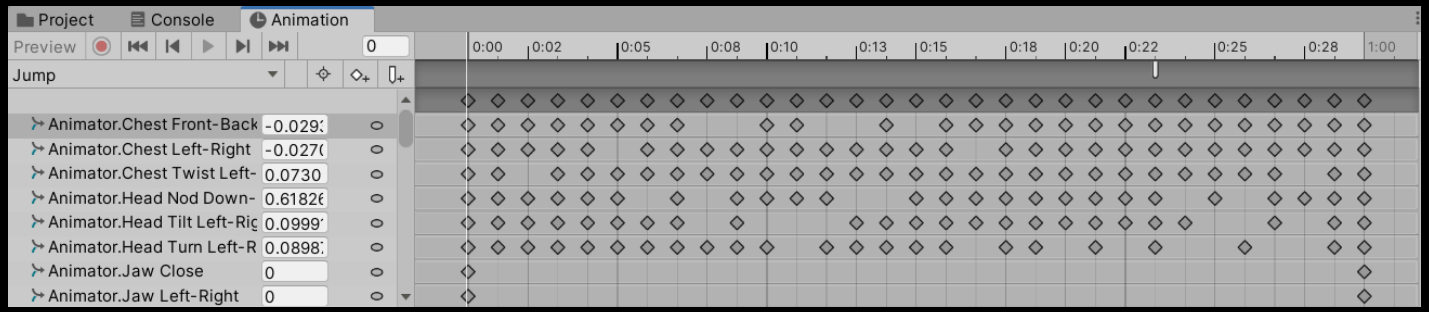
```
1  using UnityEngine;
2
3  public class CustomAnimatorEvent : MonoBehaviour
4  {
5      public void OnJumpEvent()
6      {
7          SendMessageUpwards("OnJumpEnd");
8      }
9  }
```

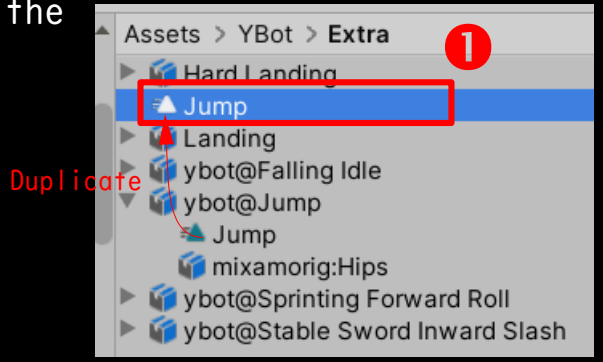


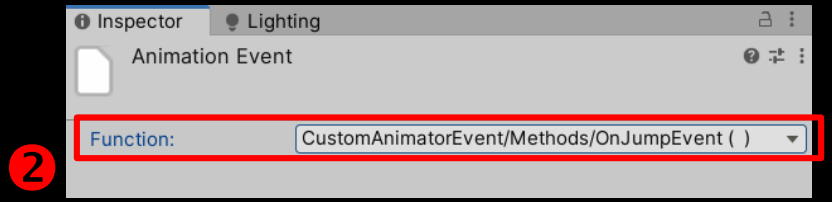
❸

```
129  public void OnCollisionEnter(Collision collision) {
130      GoToState(STATE.IDLE);
131      //anim.SetBool("isGround",true);
132  }
133
134  public void OnJumpEnd() {
135      GoToState(STATE.FALL);
136  }
137  }
```

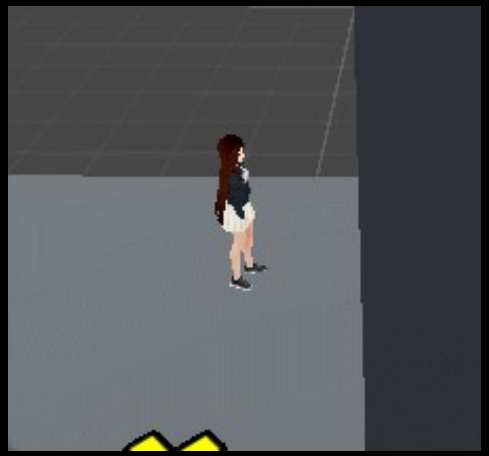
- Add an animator event
  - Duplicate the Jump animation from fbx so that the new once can be edited. Assign the new Jump animation to jump state. ❶
  - Edit the new Jump animation. Add an animation event and bind to OnJumpEvent (around frame#23). ❷
    - When the event is triggerd, the sibling monobehaviour component next to the Animator component will receive a message with the same name.



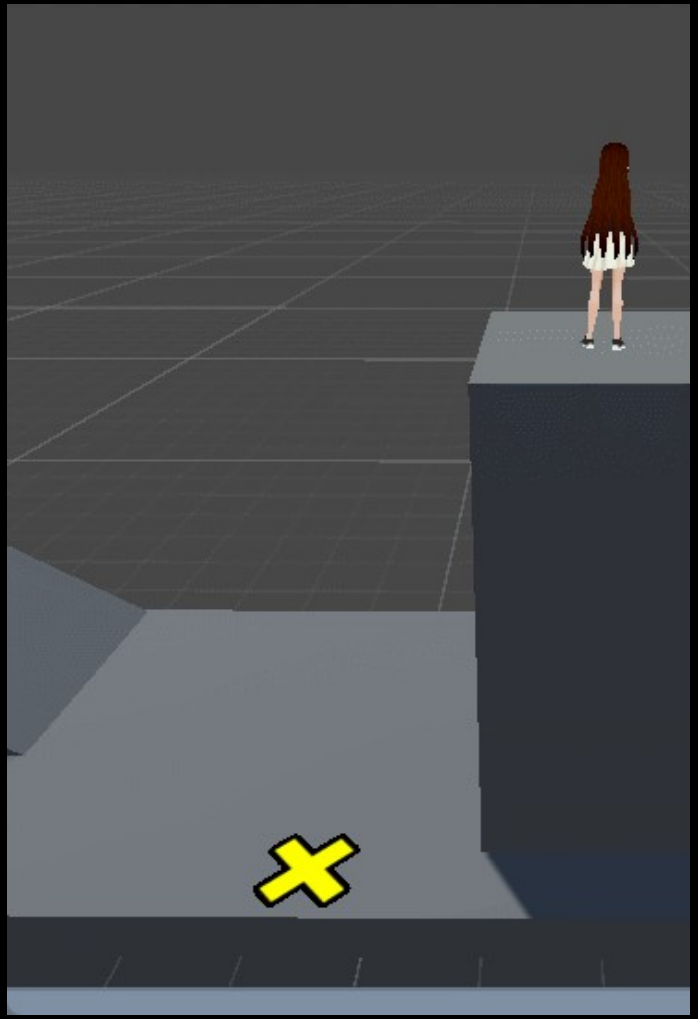




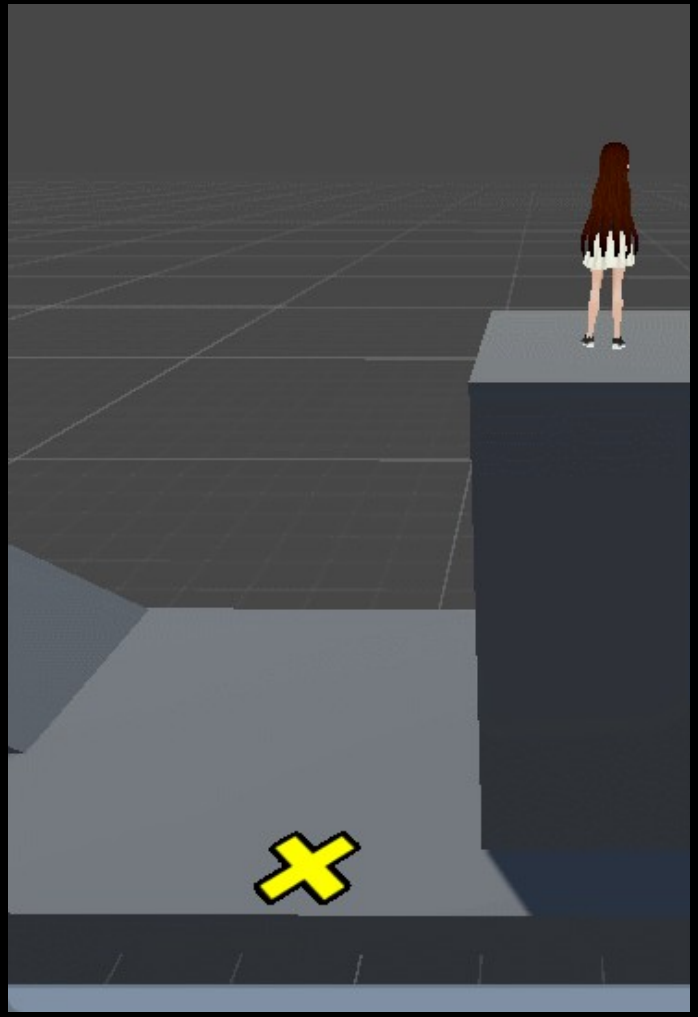
- Try it. Both should work properly.
- But if you try to jump successively you'll get error animation. (why?)
  - Solve this bug by yourself.



Successive jumps



Method#1



Method#2

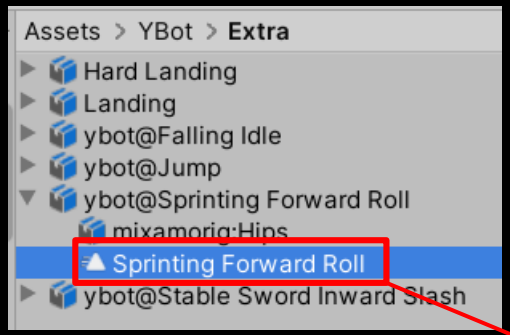
互動程式設計III

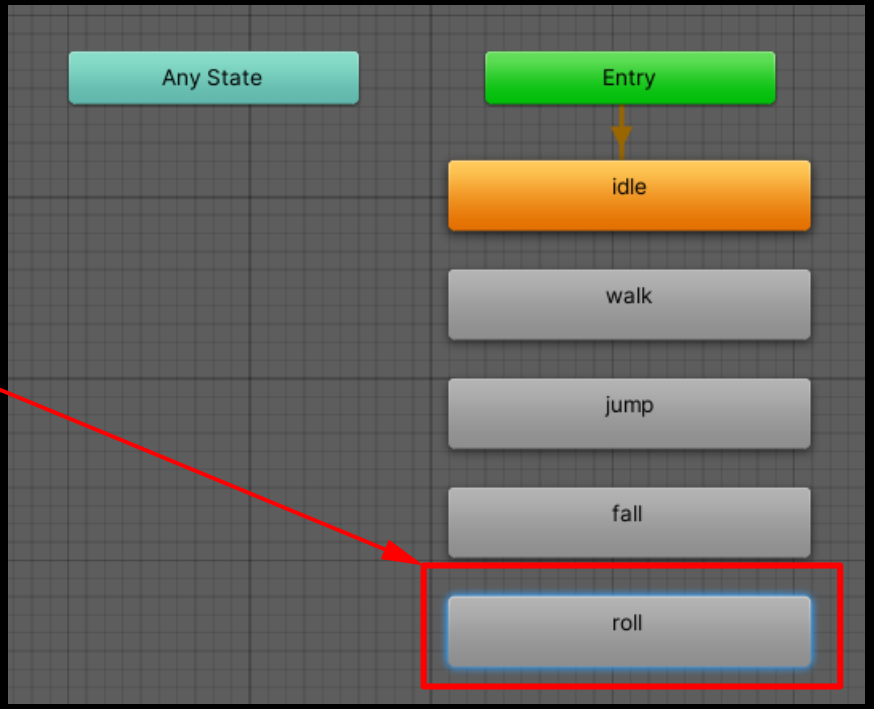
Interactive Programming Design Integration

- Player Control
  - FSM + Animator
  - Fall State
  - Roll State
  - Locomotion State
  - Ground Sensor

Add a Roll State

- The fall action finishes while the player is touching ground. We want the player to roll on the ground to reduce its momentum.
- Find the falling animation "Sprinting Forward Roll" and drag it into the animator AC.
  - This will create a new state. Name it "roll" state.





ROLL Logic

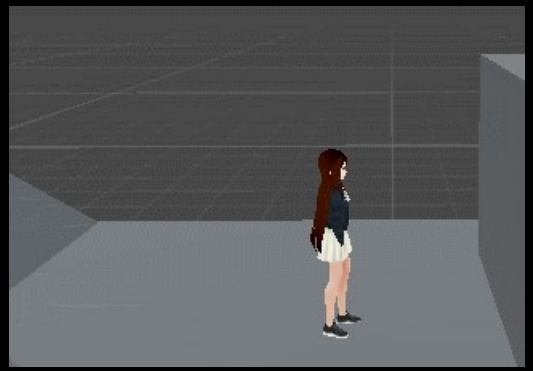
```
99                  newVelocity = rigid.velocity;
100                 break;
101             case STATE.ROLL:
102                 if (triggerEnter) {
103                     anim.CrossFadeInFixedTime("roll", 0.1f);
104                     triggerEnter = false;
105                     newVelocity = rigid.velocity;
106                     break;
107                 }
108                 stateInfo = anim.GetCurrentAnimatorStateInfo(0);
109                 newVelocity = rigid.velocity;
110                 if (stateInfo.normalizedTime > 0.8f && stateInfo.IsName("roll"))
111                 {
112                     GoToState(STATE.IDLE);
113                 }
114                 break;
115             default:
116                 break;
117         }
```

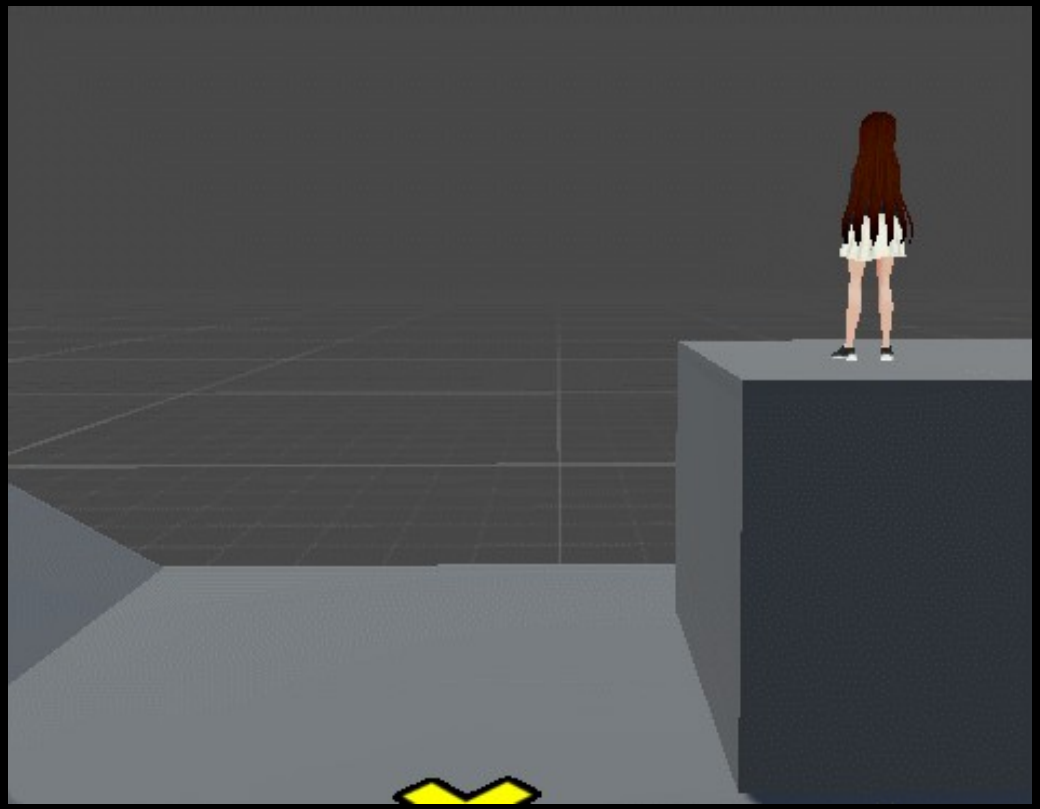
```
1   using UnityEngine;
2
3   public class PlayerController : MonoBehaviour
4   {
5       public enum STATE {
6           IDLE,
7           WALK,
8           JUMP,
9           FALL,
10          ROLL
11      }
```

- Add enum value for ROLL state. ❶
- Add logic for ROLL state
  - Crossfade to roll animation state. ❷
  - Get current animator state info. If normalized time is greater than 80% (0.8f), and current state is "roll", then switch to FALL state. ❸
  - Edit OnCollisionEnter to include FALL → ROLL transition. ❹

```
137                 GoToState(STATE.JUMP);
138             }
139         }
140     }
141
142     public void OnCollisionEnter(Collision collision) {
143         if (state == STATE.JUMP) {
144             GoToState(STATE.IDLE);
145         }
146         else if (state == STATE.FALL) {
147             GoToState(STATE.ROLL);
148         }
149     }
150 }
```

- Try it.
  - See if you can change the moving direction when falling and rolling.
  - But sometime even if you are jumping on the baseline floor, the player still enters ROLL state.

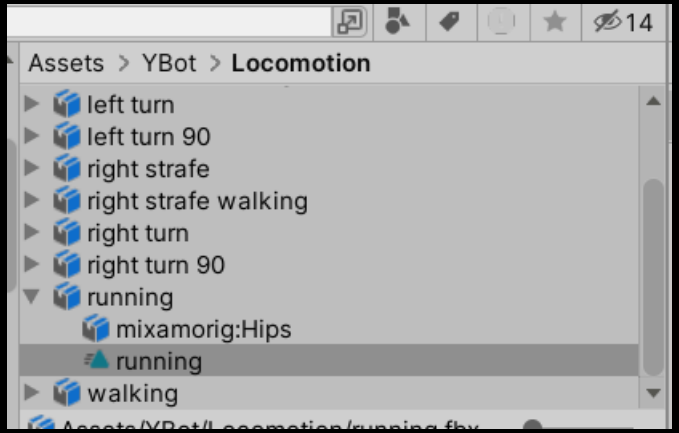


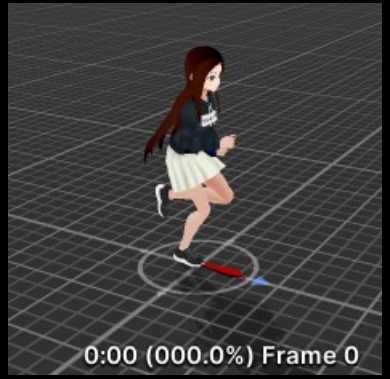


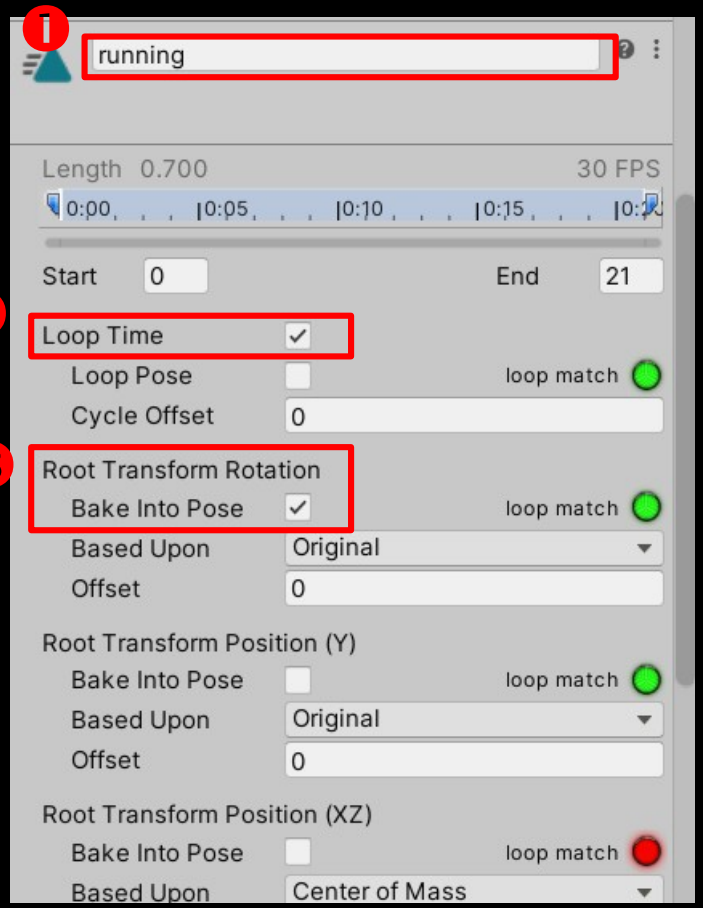
- Player Control
  - FSM + Animator
  - Fall State
  - Roll State
  - Locomotion State
  - Ground Sensor

Prepare the Running Animation

- Check the import settings of the running animation:
  - Rename the animation from mixamo.com to running. ❶
  - Check Loop Time ❷
  - Root Transfomr Rotation: check Bake Into Pose ❸

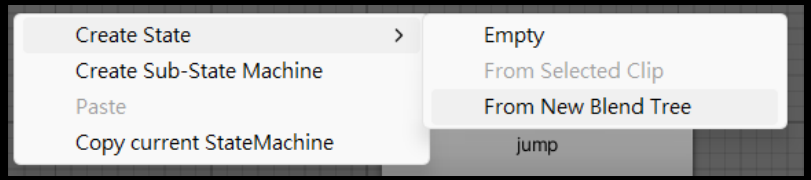


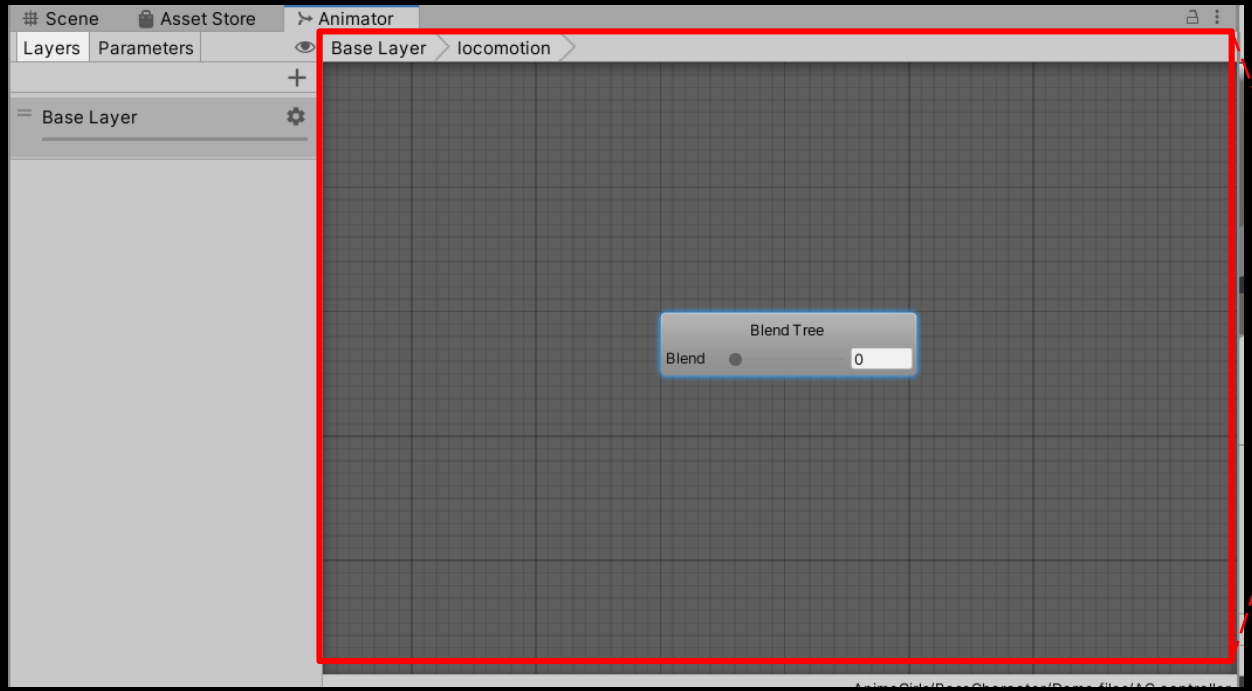


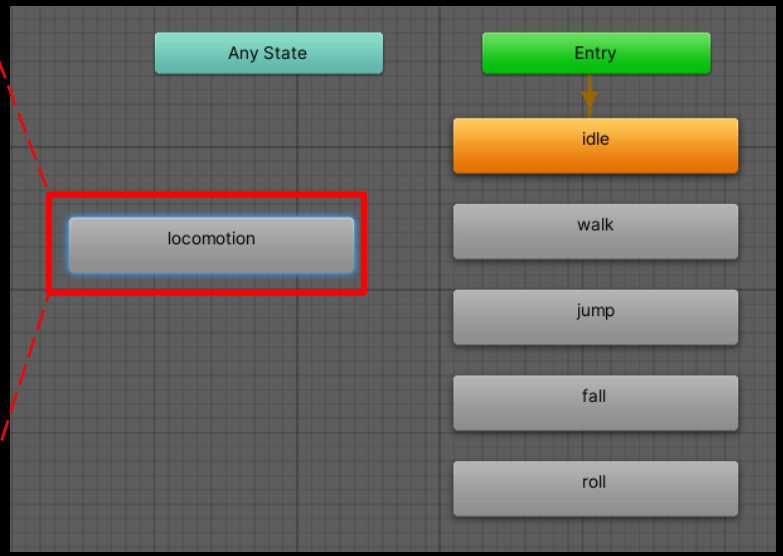


Add a 1D Blend Tree

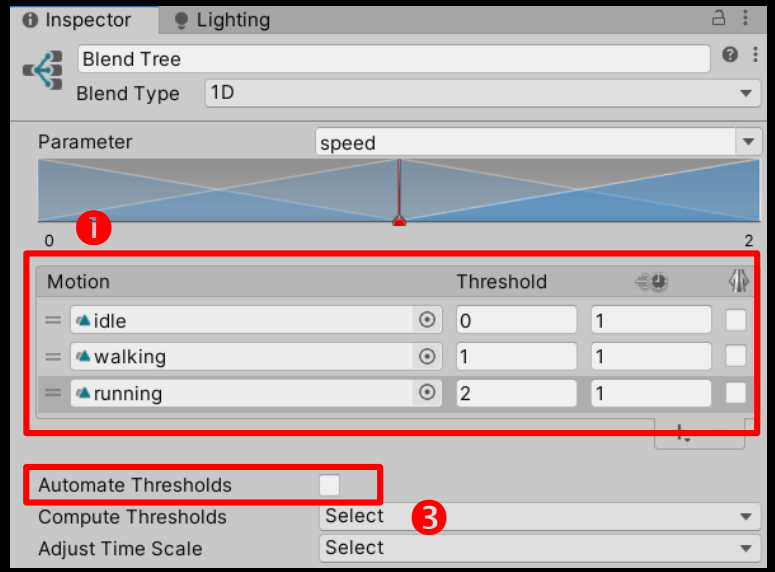
- We are going to use a blend tree to blend between walk and run animations.
  - Add a Blend Tree and call it "locomotion".
  - Double click locomotion to open it.

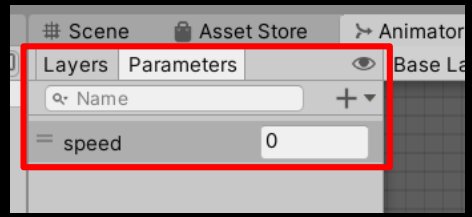


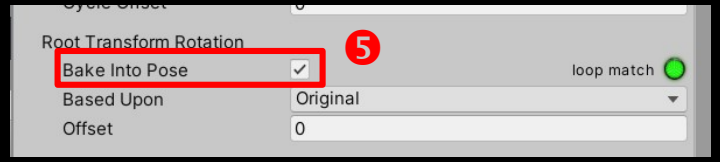


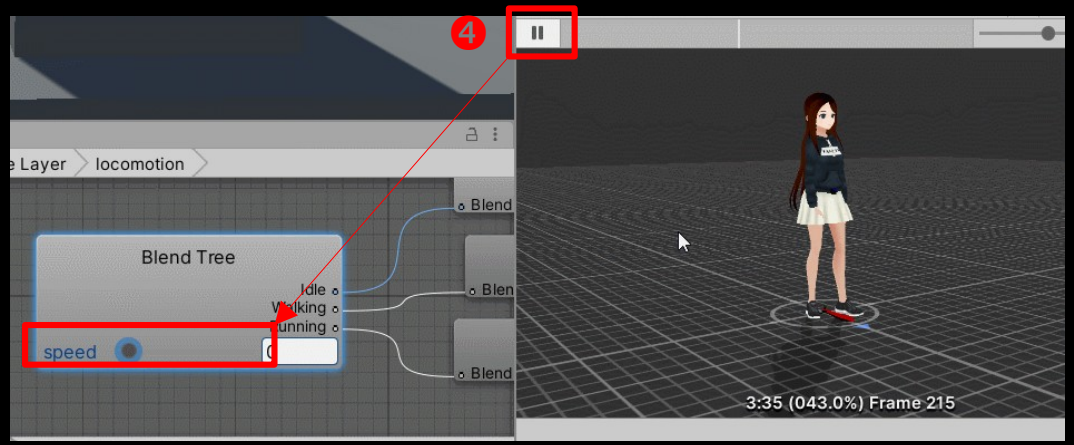


- Add 3 motion fields in the Motion list.
  - Lock the inspector and drag idle, walking, running in the list. ❶
  - When you create the ID blend tree, an animator parameter called Blend will also be created. Rename this parameter ”speed”. ❷
  - Disable auto threshold and set the threshold to 0, 1 and 3 accordingly. ❸
  - Play the preview and try to move the slider on blend tree to see the effect. ❹
  - If you find the animated model has weird rotation, you can bake the animation Root Transform Rotation into pose. ❺









LOCOMOTION Logic

- Select the WALK in STATE enum, right click and choose rename to rename it as LOCOMOTION. ❶
- Add an isRunning flag. ❷
- Use the following ternary condition operator to simplify the code:
  - Expression: (isRunning? 3.0f : 1.0f)
  - When isRunning equals true, the expression returns 3.0f.
  - Otherswise the expression returns 1.0f. ❸
  - Use this expression to modulate both of the animation and newVelocity.
- Add a Run method for setting isRunning. ❹

```
19  private Vector3 movingVec;
20  private Animator anim;
21  private Rigidbody rigid;
22  [SerializeField]
23  private bool isThrust = false;
24  [SerializeField]
25  private bool isRunning = false;
26  private bool triggerEnter = false;
```

```
3   public class PlayerController : MonoBehaviour
4   {
5       public enum STATE {
6           IDLE,
7           WALK,
8           JUMP
9           FALL
10          ROLL
11      }
```

WALK

重新命名會更新 1 個檔案中的 4 個參考。

包括註解(C)

```
3   public class PlayerController : MonoBehaviour
4   {
5       public enum STATE {
6           IDLE,
7           LOCOMOTION,
8           JUMP,
9           FALL,
10          ROLL
11      }
```

```
63          case STATE.LOCOMOTION:
64              if (triggerEnter) {
65                  anim.CrossFadeInFixedTime("locomotion", 0.1f);
66                  triggerEnter = false;
67                  break;
68              }
69              if (movingVec.magnitude <= 0.1f) {
70                  GoToState(STATE.IDLE);
71                  break;
72              }
73              anim.SetFloat("speed", isRunning ? 3.0f : 1.0f);
74              // Calculate new velocity
75              newVelocity = movingVec * velocity * (isRunning? 3.0f: 1.0f);
76              model.transform.forward = Vector3.Slerp(
77                  model.transform.forward, movingVec, 0.1f);
78              break;
```

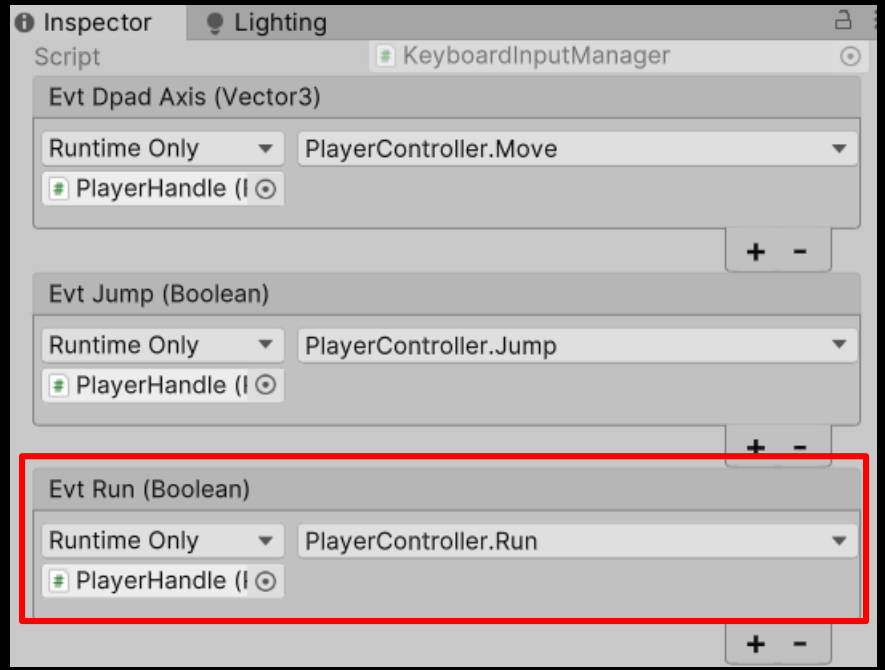
```
142
143 public void Run(bool _isRunning) {
144     isRunning= _isRunning;
145 }
```

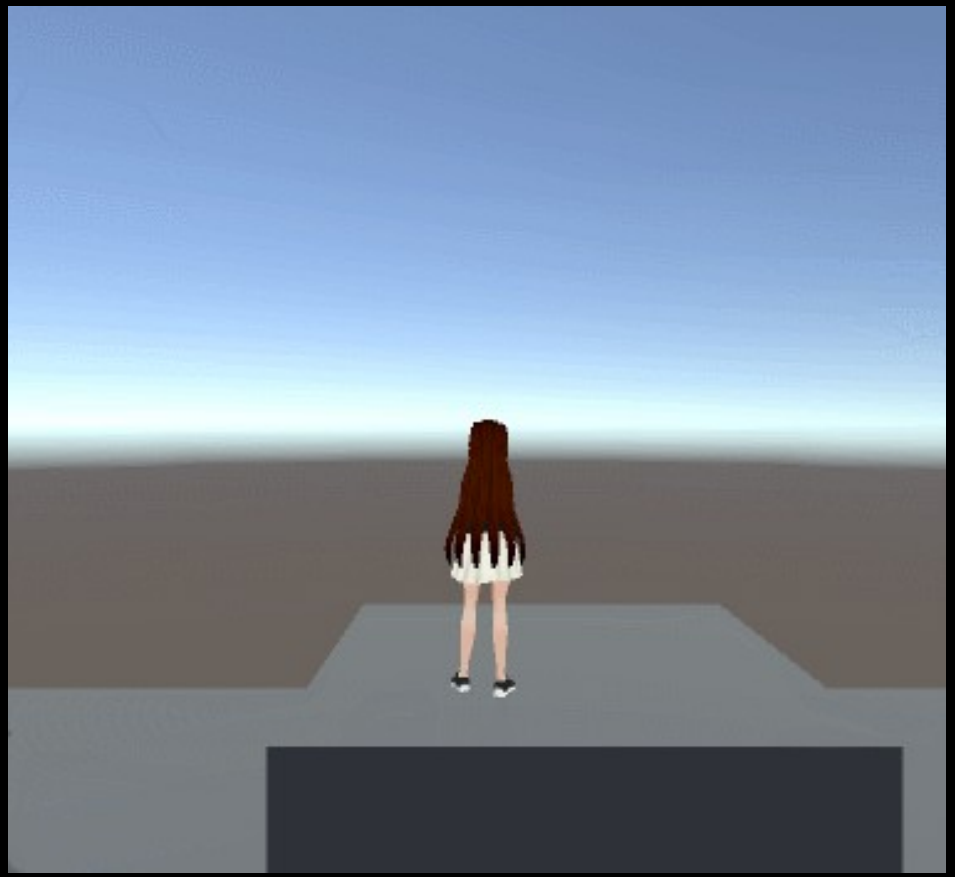
- We also need to update the InputManager and KeyboardInputManager
  - InputManager ❶
    - Add evtRun
    - Add CalculateRun()
  - KeyboardInputManager ❷
    - Add a temp variable run to save the status of run key.(right shift)
    - Invoke evtRun and pass the run key status.

```
1   using UnityEngine.Events;
2   using UnityEngine;
3
4   public abstract class InputManager : MonoBehaviour
5   {
6       public UnityEvent<Vector3> evtDpadAxis;
7       public UnityEvent<bool> evtJump;
8       public UnityEvent<bool> evtRun;
9
10      protected abstract void CalculateDpadAxis();
11      protected abstract void CalculateJump();
12      protected abstract void CalculateRun();
13      protected abstract void PostProcessDpadAxis();
14
15      private void Update()
16      {
17          CalculateDpadAxis();
18          CalculateJump();
19          CalculateRun();
20          PostProcessDpadAxis();
21      }
22  }
```

```
1   using UnityEngine;
2   public class KeyboardInputManager : InputManager
3   {
4       private Vector3 axis;
5       private bool jump;
6       private bool run;
7
8       protected override void CalculateDpadAxis()...
30
31      protected override void CalculateJump()...
36
37      protected override void CalculateRun()
38      {
39          run = Input.GetKey("right shift");
40          evtRun?.Invoke(run);
41      }
42
43      protected override void PostProcessDpadAxis()...
47  }
```

- Bind the Run method to KeyboardManagerInput. Try it.
- Important! Due to the keyboard's hardware circuitry cost, not all of the 3-key combos are supported by your keyboard.
  - The more combinations a keyboard supports, the higher the manufacturing costs.

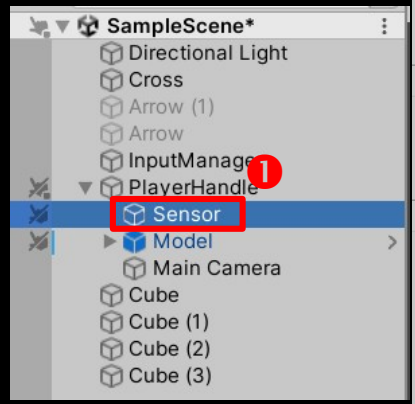


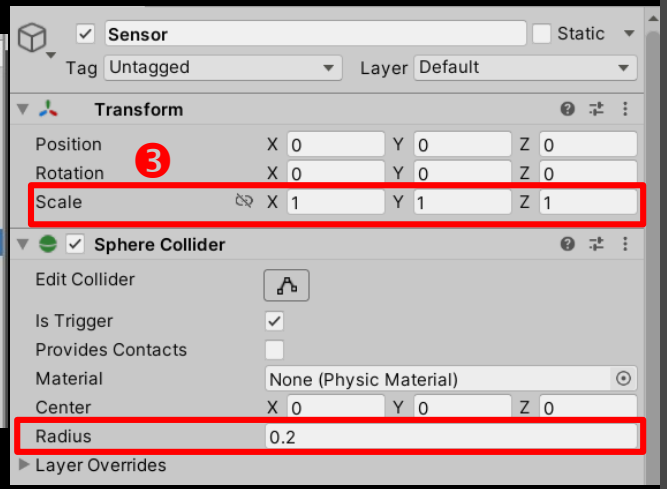


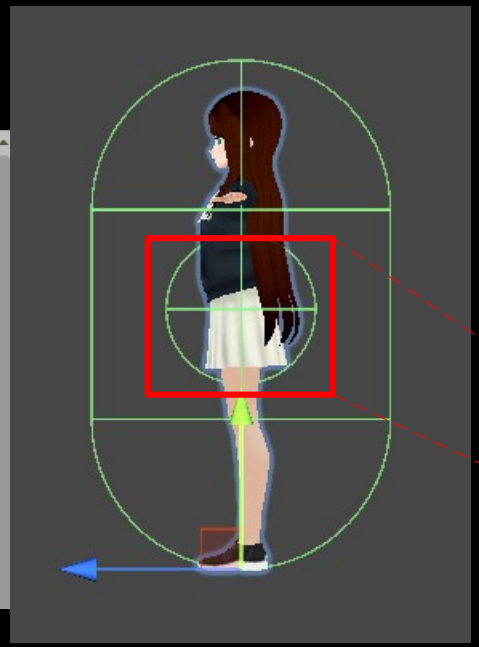
- Player Control
  - FSM + Animator
  - Fall State
  - Roll State
  - Locomotion State
  - Ground Sensor

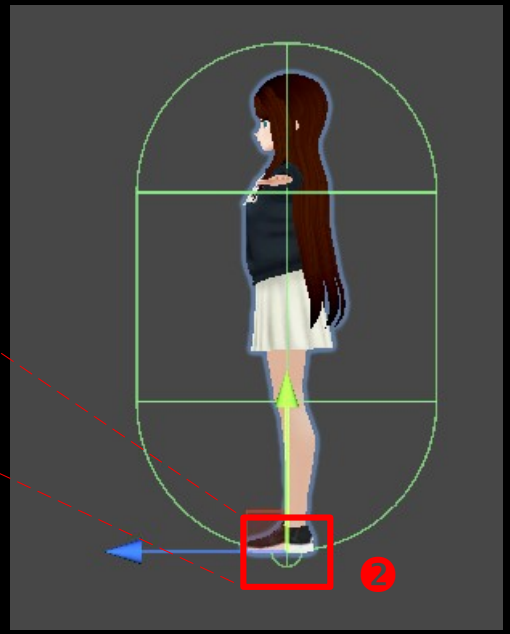
Sense the Ground

- Since we have a Sensor gameObject in our player structure❶. We are going to utilize it to sense the ground.
  - If there isn't any ground object below, the player should change from LOCOMOTION state to FALL state.
- Move the sensor object to (0, 0, 0)❷
- Scale to (1,1,1,) and Radius to 0.2. Set the collider on Sensor to be a trigger.❸









Ground Sensor Logic

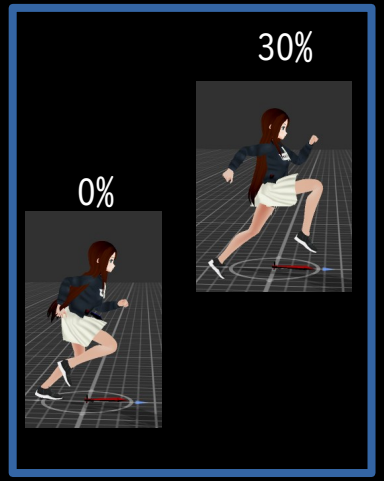
- Because the Player Handle has a rigidbody, so all the colliders attached to its children will also be considered as part of the rigidbody. This is called compound collider.[*]
- Add a GroundSensor.cs to the Sensor gameObject.
  - Create a Unity event evtGround for player controller to subscribe this on ground event.
  - Sense the groud collider by physics trigger.
    - Count the colliders that are not tagged as ”Ground”.
  - Update the grounding information periodically.

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

public class GroundSensor : MonoBehaviour
{
    public UnityEvent<bool> evtGround;
    [SerializeField]
    private List<Collider> colliders = new List<Collider>();

    private void Update()
    {
        evtGround?.Invoke((colliders.Count > 0) ? true : false);
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Ground")
        {
            colliders.Add(other);
        }
    }

    private void OnTriggerExit(Collider other)
    {
        colliders.Remove(other);
    }
}
```

Modify Player Controller

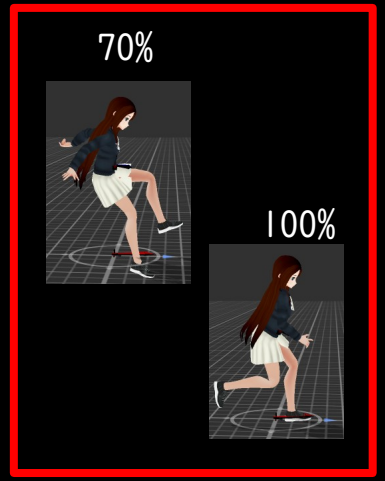
- Comment out the OnCollisionEnter.
- Add IsGround method
  - If the player in on ground:
    - And if it's in JUMP state, check if the animation is also playing jump state and exceeds 70% progress. If so, switch to IDLE state.
    - Or if it's in FALL state, just lands and ROLL without checking.
  - If the player is NOT on ground:
    - And if it's in IDLE or LOCO state, just FALL without checking.

IsGround, but
This is NOT landing



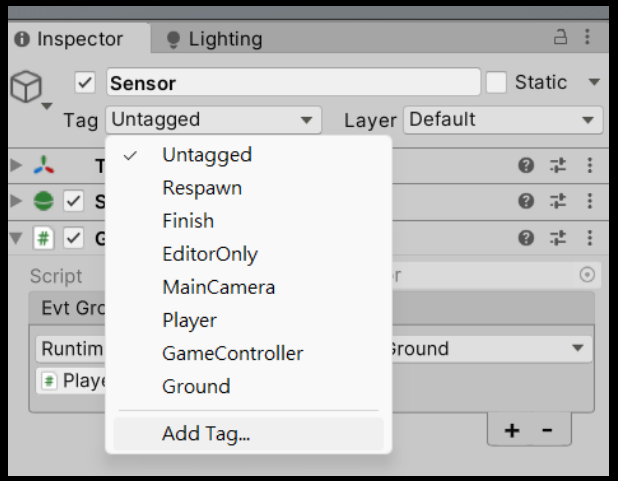
Is NOT ground 30% ~ 70...

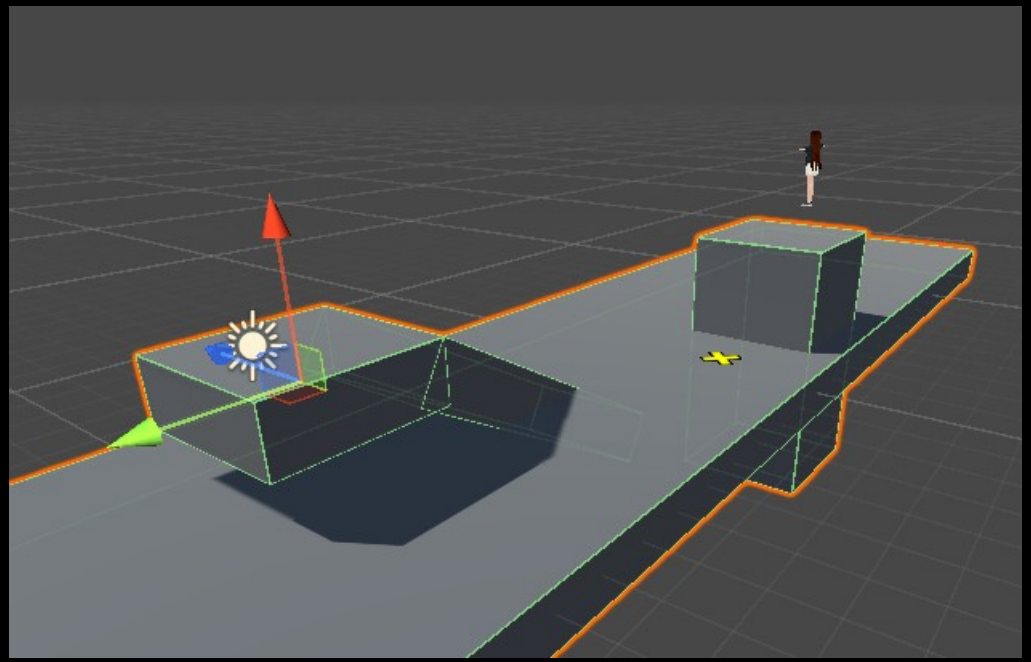
IsGround, and
This is landing

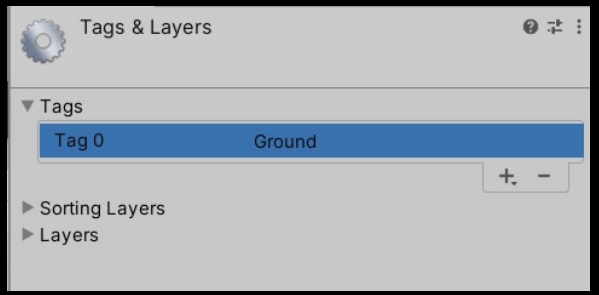


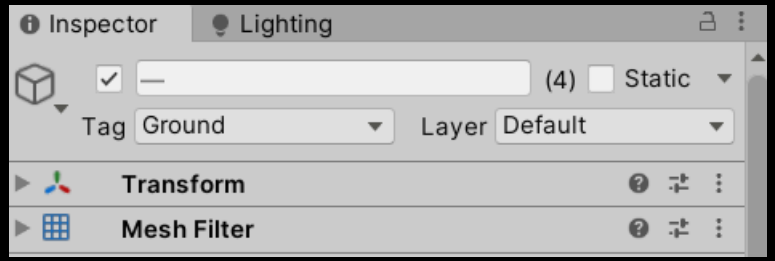

```
//public void OnCollisionEnter(Collision collision) {
//    if (state == STATE.JUMP) {
//        GoToState(STATE.IDLE);
//    }
//    else if (state == STATE.FALL) {
//        GoToState(STATE.ROLL);
//    }
//}

public void IsGround(bool _isGround)
{
    if (_isGround)
    {
        if (state == STATE.JUMP)
        {
            AnimatorStateInfo stateinfo = anim.GetCurrentAnimatorStateInfo(0);
            if (stateinfo.normalizedTime > 0.7f && stateinfo.IsName("jump"))
            {
                GoToState(STATE.IDLE);
            }
        }
        else if (state == STATE.FALL)
        {
            GoToState(STATE.ROLL);
        }
    }
    else
    {
        if (state == STATE.IDLE || state == STATE.LOCOMOTION)
        {
            GoToState(STATE.FALL);
        }
    }
}
}
```

- Create a tag called ”Ground”. Becareful to make the first letter a capital G.
- Select all the Cube gameObjects and set their tags to Ground.

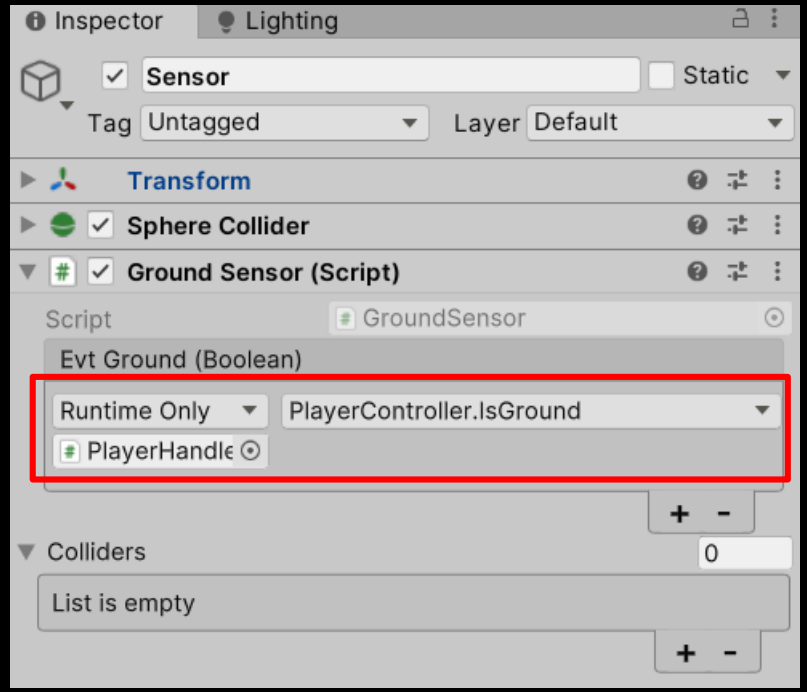


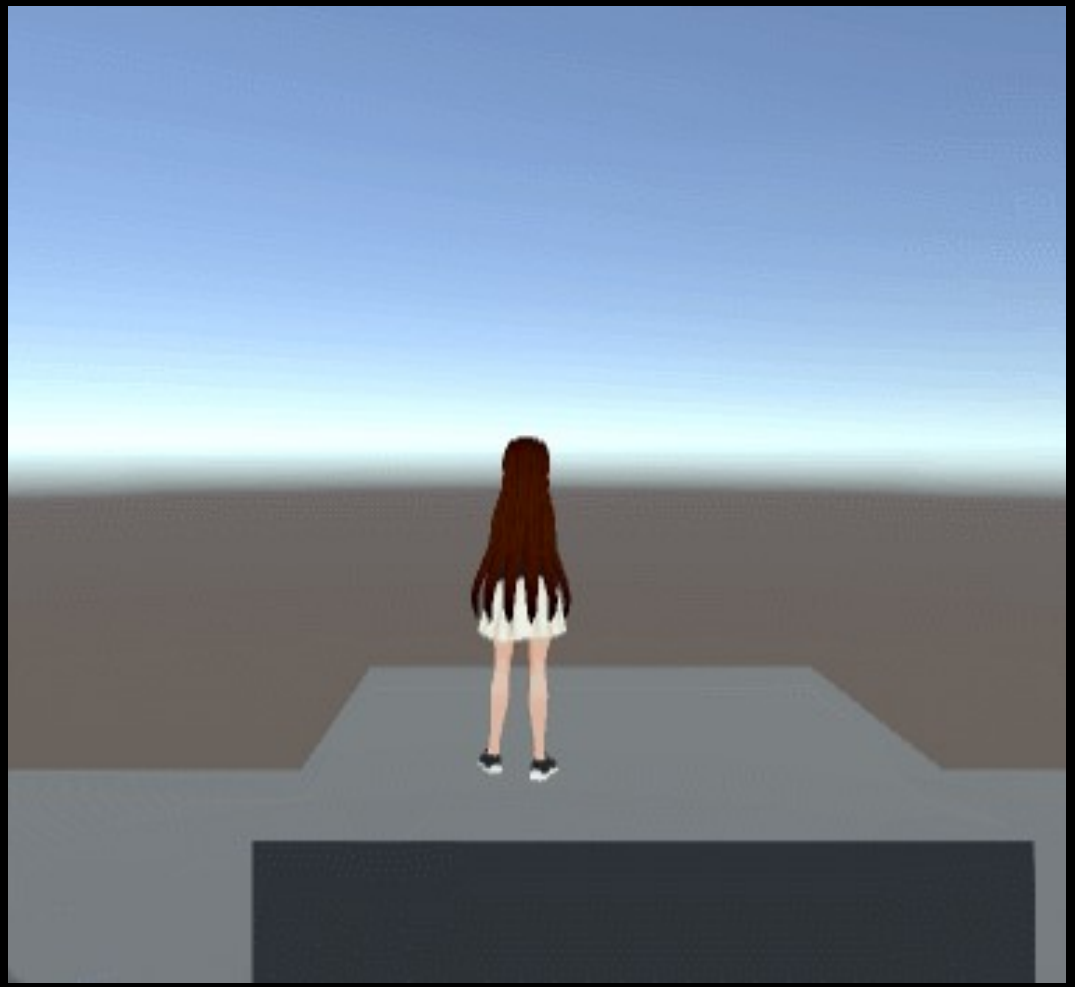






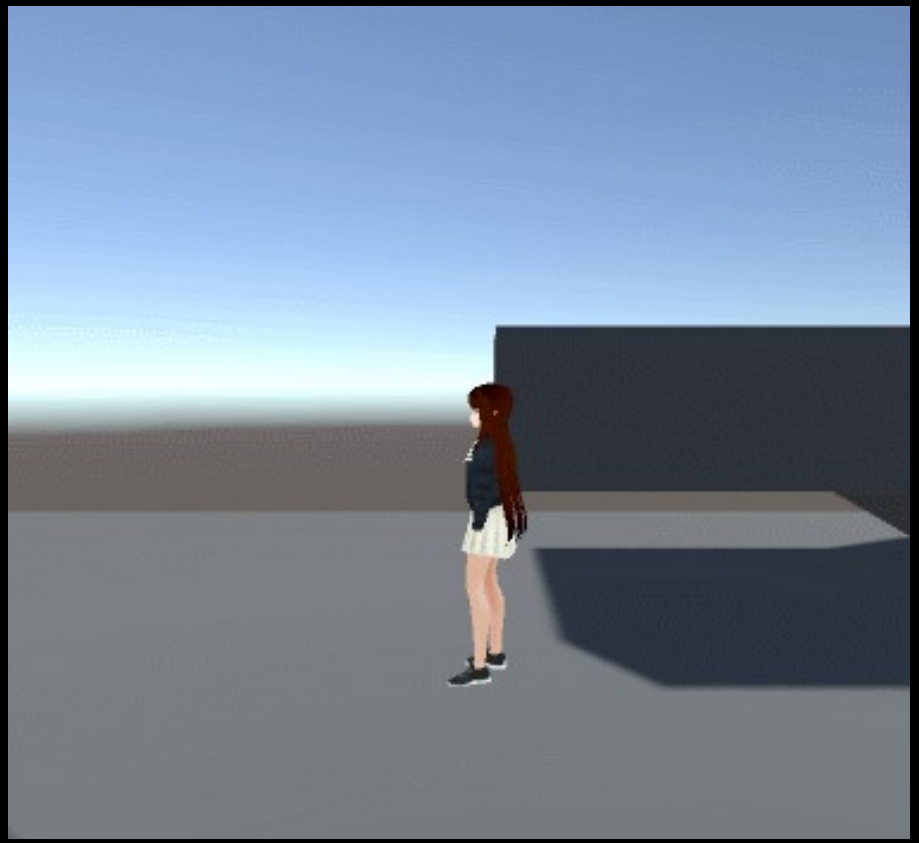
- Bind the IsGround method to Sensor. Try it.
- BUG!! If you press jump key in a burst, the player will jump but shows falling animation. (why?)
  - How to prevent user jumping in a burst?





A Little Question for You

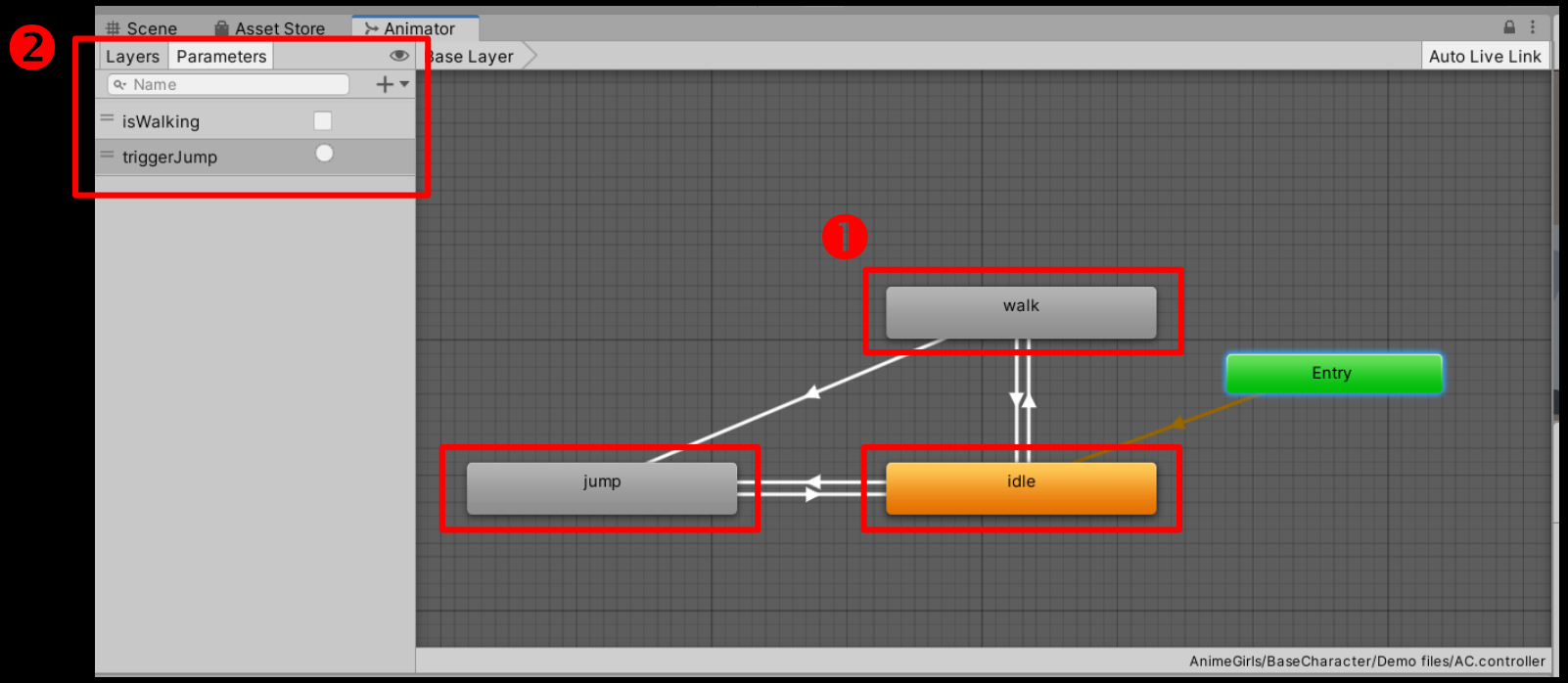
- When turning from WALK to RUN, the animation changes suddenly. How to make this change smoother both in movement and animation?



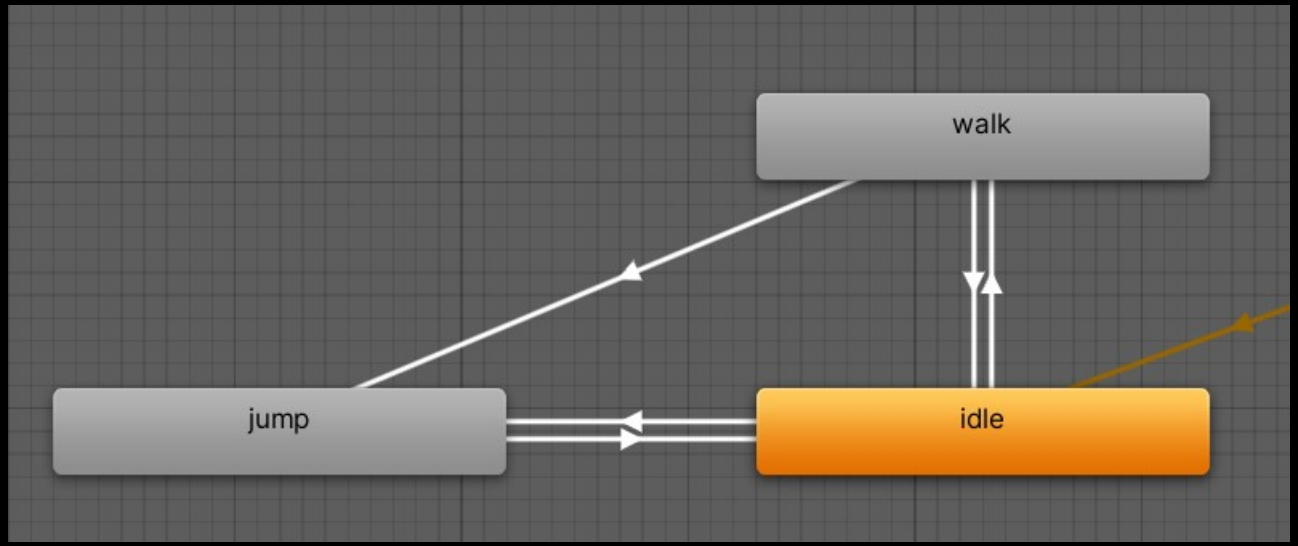
Q&A

Animator

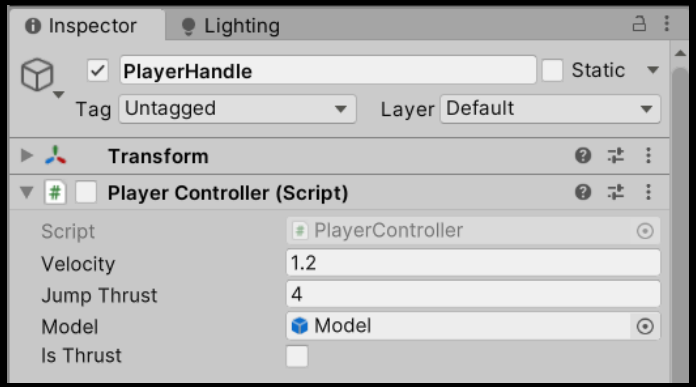
- The 1st version animator
  - 3 states: idle, walk, jump❶
- Add 2 new Animator parameters:❷
  - IsWalking (bool): whether the player is walking
  - TriggerJump (trigger): fired when player starts jumping



- Add transitions
  - Planar movement
    - idle → walk: exit time OFF, isWalking == true
    - walk → idle: exit time OFF, isWalking == false
  - Airborne
    - idle → jump: exit time OFF, triggerJump
    - walk → jump: exit time OFF, triggerJump
    - jump → idle: exit time ON



- Try it.
  - Turn off the Player Controller first.
  - Select the PlayerHandle in scene to see FSM flowing.



- Regulate movingVec to be either horizontal or vertical. ❶
- IDLE state logic: ❷
  - If the movingVec has a magnitude larger than 0.1f, then switch to WALK state. And break the state logic execution here.
  - Once we are under IDLE state, we should set anim param "isWalking" to false.
- WALK state logic: ❸
  - If the movingVec has a magnitude smaller than 0.1f, then switch to IDLE state. And break the state logic execution here.
  - Once we are under WALK state, we should set anim param "isWalking" to true.
  - Update the newVelocity by movingVec.
  - Gradually turn the model by Slerp().
- Jump state logic: ❹
  - Just copy and keep the current velocity.

```
❶ // Restrict to 4 direction movements.
❶ if (Mathf.Abs(movingVecH) >= Mathf.Abs(movingVecV)) {
❶     movingVec = movingVecH * Vector3.right;
❶ }
❶ else {
❶     movingVec = movingVecV * Vector3.forward;
❶ }

  switch (state) {
❷     case STATE.IDLE:
❷         if (movingVec.magnitude > 0.1f) {
❷             GoToState(STATE.WALK);
❷             break;
❷         }
❷         anim.SetBool("isWalking", false);
❷         break;
❸     case STATE.WALK:
❸         if (movingVec.magnitude <= 0.1f) {
❸             GoToState(STATE.IDLE);
❸             break;
❸         }
❸         anim.SetBool("isWalking", true);
❸         // Calculate new velocity
❸         newVelocity = movingVec * velocity;
❸         model.transform.forward = Vector3.Slerp(
❸             model.transform.forward, movingVec, 0.1f);
❸         break;
❹     case STATE.JUMP:
❹         newVelocity=rigid.velocity;
❹         break;
      default:
          break;
  }
```

- Apply jump thrust to newVelocity❶
- Update Jump()❷
  - When player hit jump button, check whether current state is IDLE or WALK first.
    - If yes, set isTrust to true and switch state to JUMP.
    - Set anim trigger "triggerJump" to trigger animation transition.

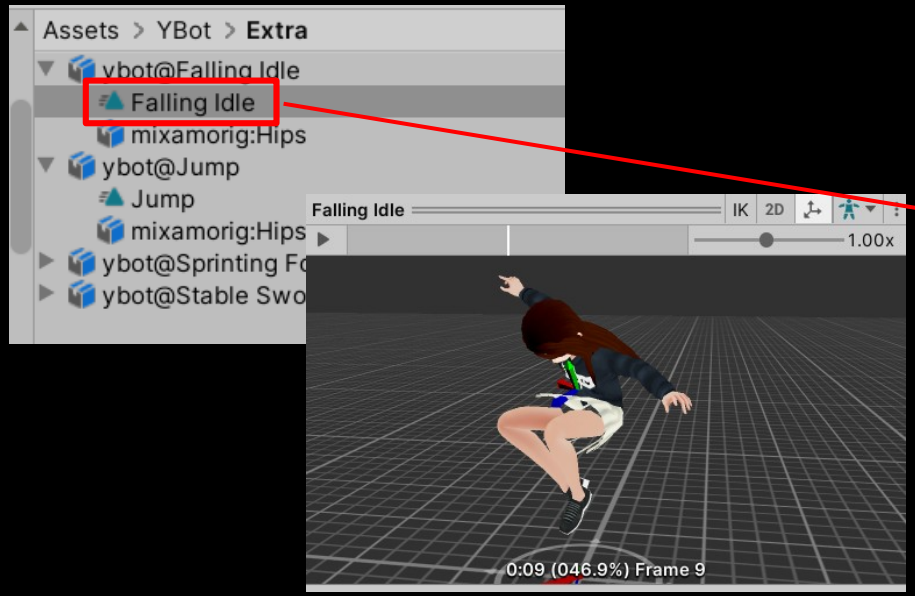
```
70      ❶   // Applying thrust
71          newVelocity.y = rigid.velocity.y + (isThrust ? 1.0f : 0) * jumpThrust;
72          rigid.velocity = newVelocity;
73          isThrust = false;
74      }
75
76      private void GoToState(STATE targetState) {
77          state = targetState;
78          triggerEnter = true;
79      }
80
81      public void Move(Vector3 vector) {
82          movingVec = vector;
83      }
84
85      ❷   public void Jump(bool _isThrust) {
86          if (_isThrust) {
87              if (state == STATE.IDLE || state == STATE.WALK) {
88                  isThrust = true;
89                  GoToState(STATE.JUMP);
90                  anim.SetTrigger("triggerJump");
91              }
92          }
93      }
94
95      public void OnCollisionEnter(Collision collision) {
96          GoToState(STATE.IDLE);
97      }
98  }
```

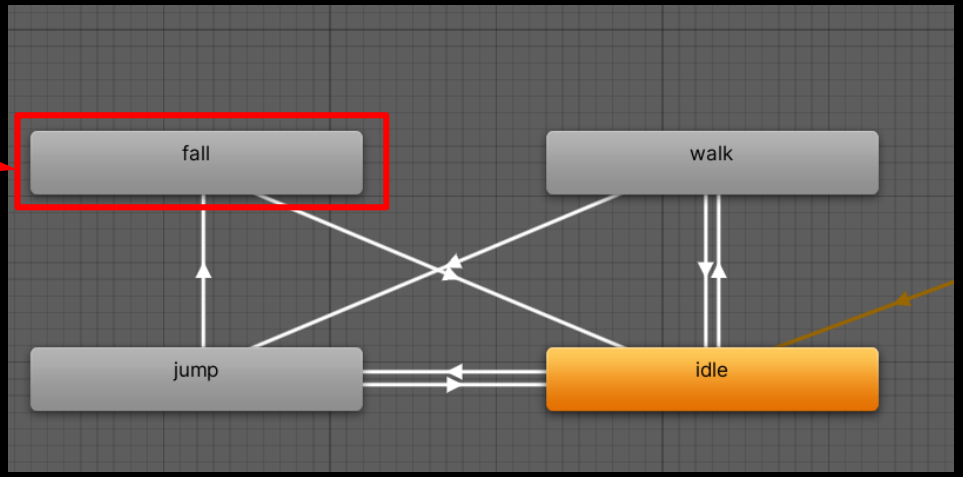
- Try it. Everything should work like before.

- Player Control
  - FSM + Animator
  - Fall State

Add a Fall State

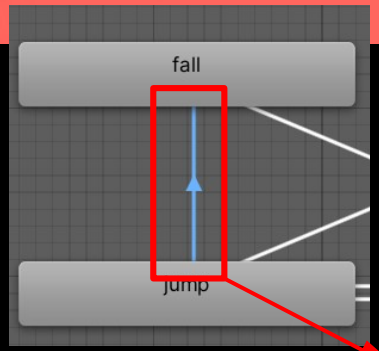
- Find the falling animation ”Falling Idle” and drag it into the animator AC.
  - This will create a new state. Name it ”fall” state.

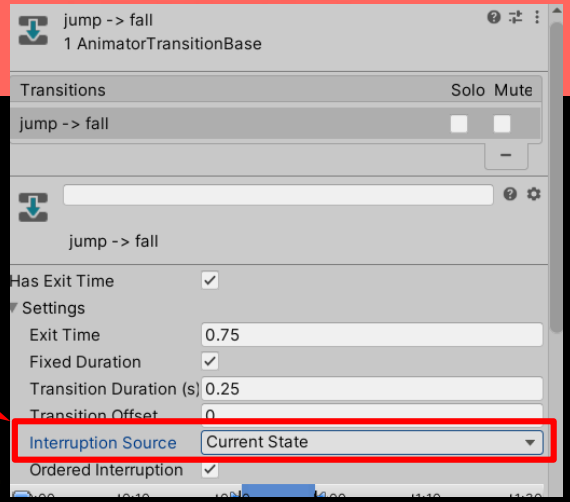


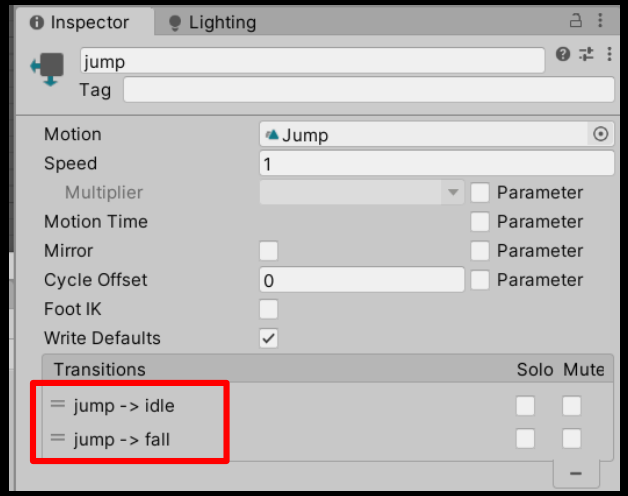


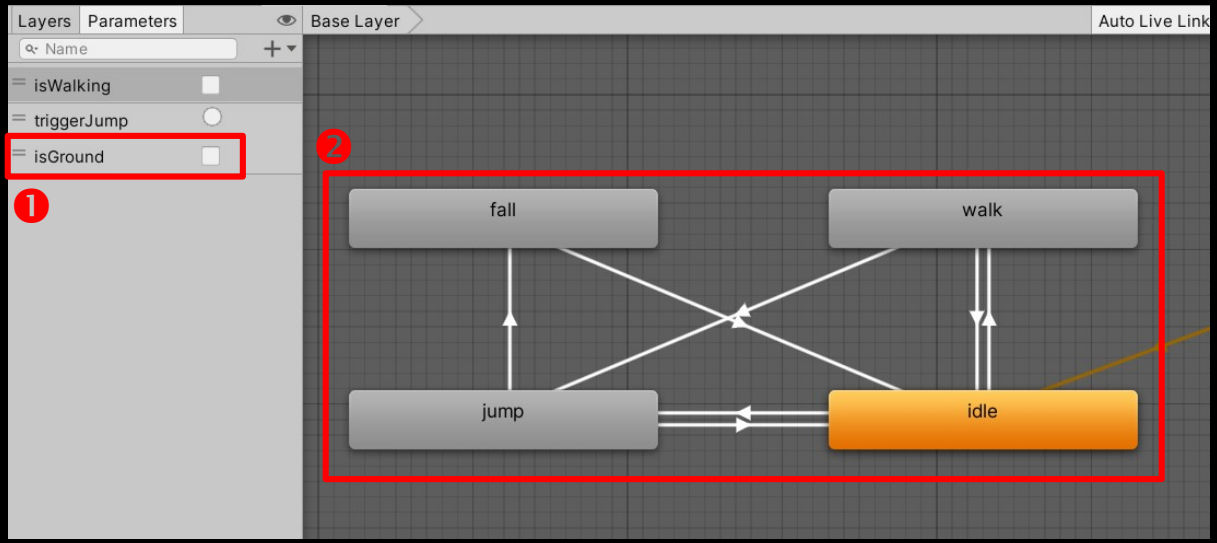
Add a Fall State

- Add a bool animator parameter isGround. ❶
- Edit state transitions: ❷
  - jump → idle: Exit time OFF, isGround == true
  - jump → fall: Exit time ON
  - fall → idle: Exit time OFF, isGround == true
- Set the priority of transitions from jump:
  - Jump to idle should has a higher priority! ❸ (Why? Think about it)
  - Set the interruption source to Current State. ❹



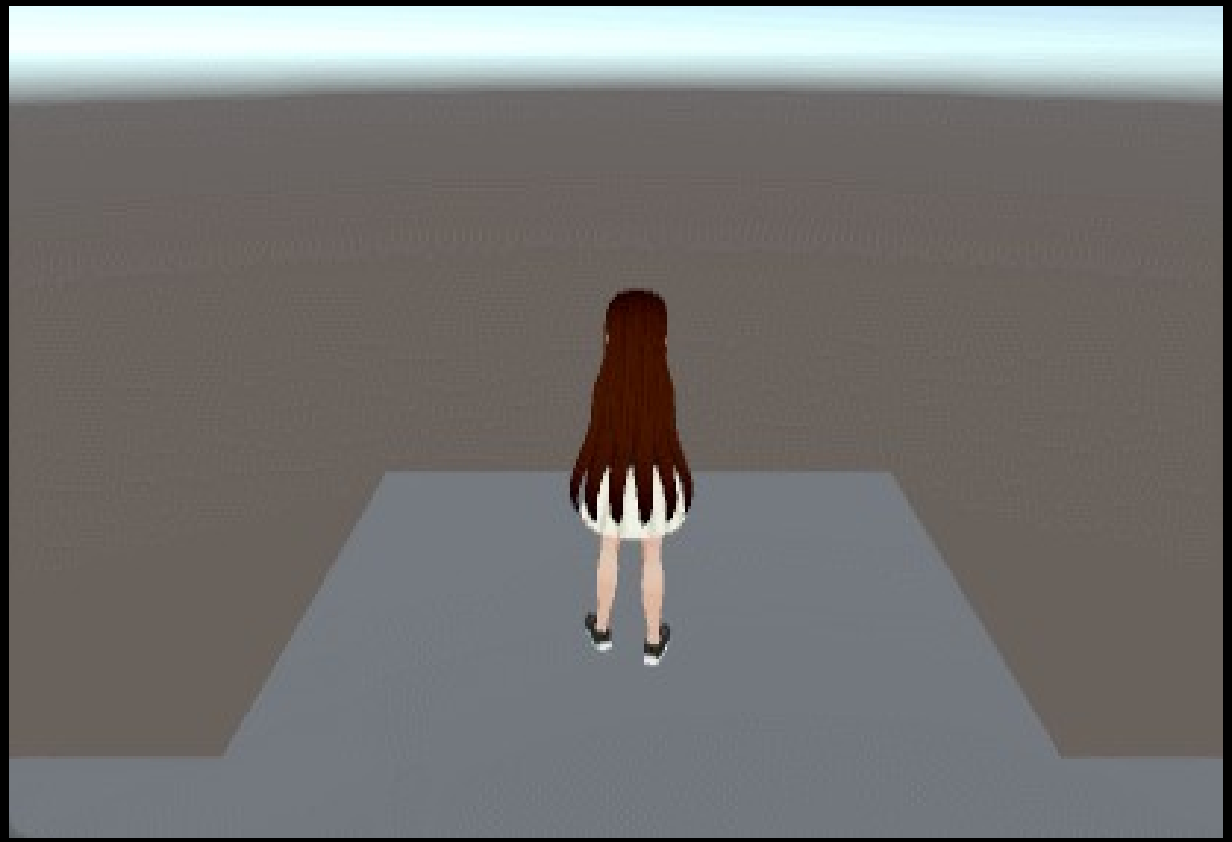






https://unity.com/blog/engine-platform/state-machine-transition-interruptions

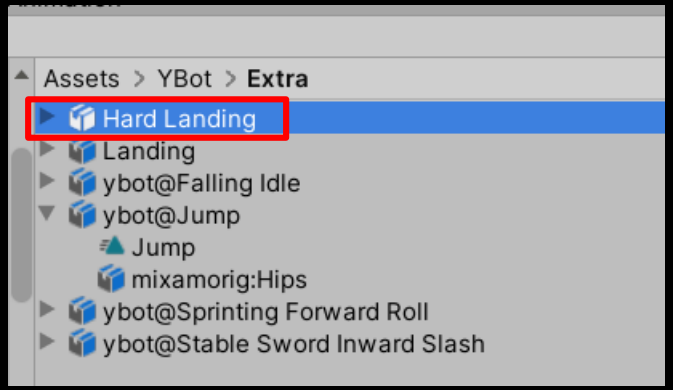
- Try it. Everything should work like before.

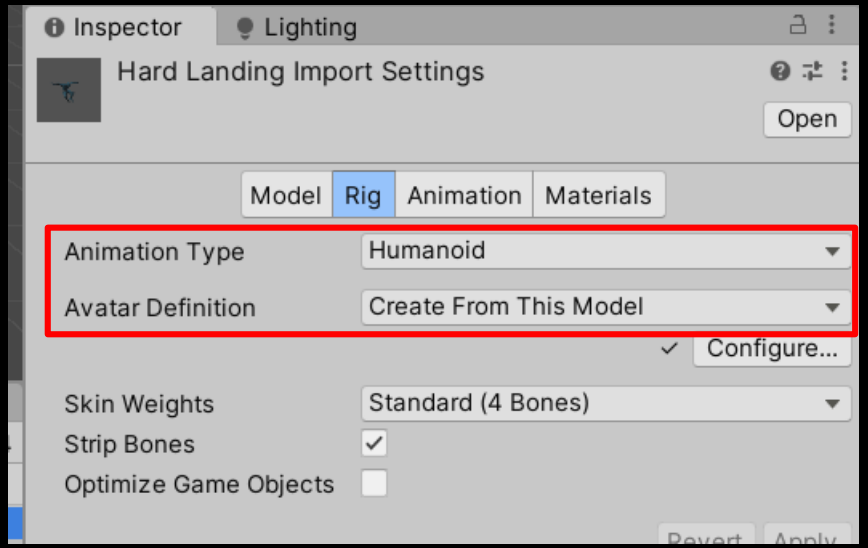


- Player Control
  - FSM + Animator
  - Fall State
  - Land State

Add a Land State

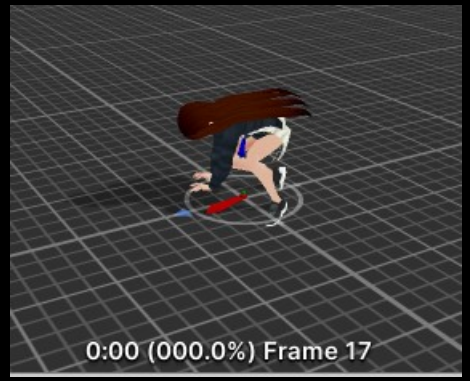
- Find the Hard Landing animation in Extra folder. Or download it from Mixamo (download it with Skin)
  - Select the fbx and set it import configurations:
    - Animation type set to Humanoid.
    - Avatar definition set to Create from this model.

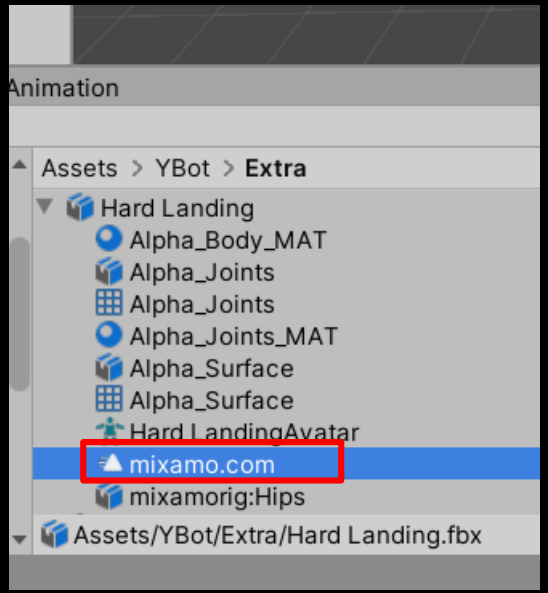


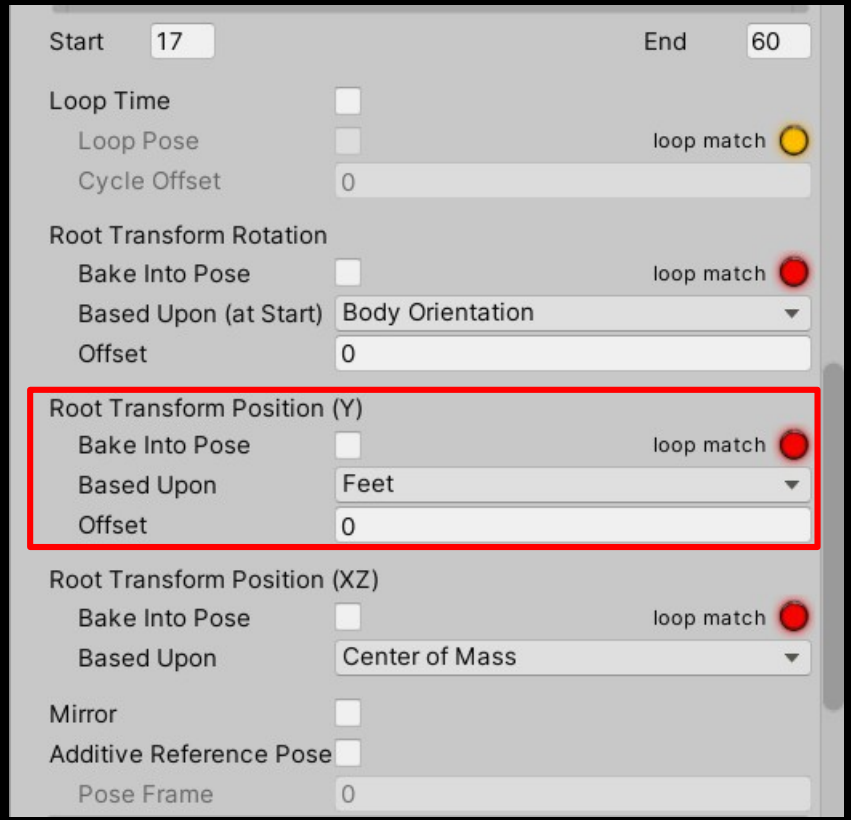


Add a Land State

- Edit the animation
  - Set Root Transform Position(Y): Based Upon to Feet.

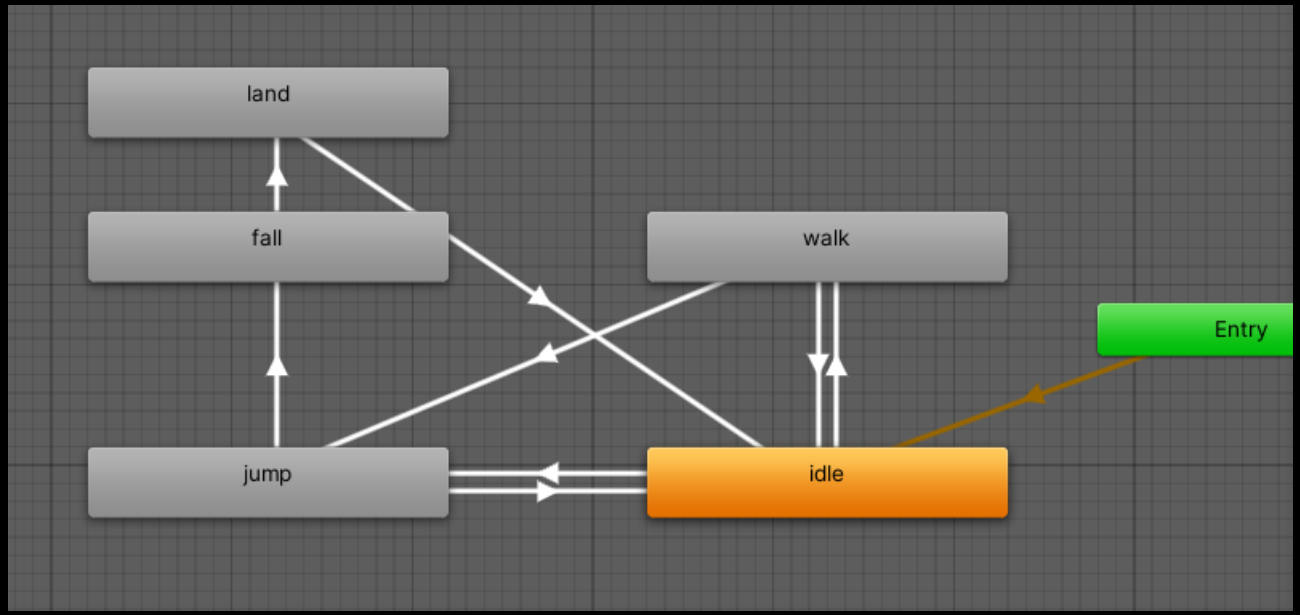






Add a Land State

- Drag Hard Landing animation into animator to create a new state called land.
- Edit transitions:
  - Delete: fall → idle
  - fall → land: Exit time OFF, isGround == true
  - land → idle: Exit time ON
  - jump → idle: Exit time OFF, isGround == true



Modify Player controller

- We are going to design a state to "lock" the player without letting the player to move.
- Add a new enum value LAND. ❶
- Add LAND state logic in switch❷
  - Just reset the newVelocity to zero.
- Add OnLandStart() and OnLandEnd()❸
  - We are planing to call OnLandStart when landing animaiton starts to play. And call OnLandEnd when landing animation is finished.
  - To do so, we are going to write some "Behaviour" script on animator state later.
- Modify OnCollisionEnter❹
  - Do not go to IDLE directly. OnLandEnd will do that instead.
  - Just send isGround info to animator.

```
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    public enum STATE {
        IDLE,
        WALK,
        JUMP,
        LAND
    }
    [SerializeField]
    private STATE state;
```

❶

```
            case STATE.JUMP:
                newVelocity=rigid.velocity;
                break;
            case STATE.LAND:
                newVelocity = Vector3.zero;
                break;
            default:
                break;
        }
```

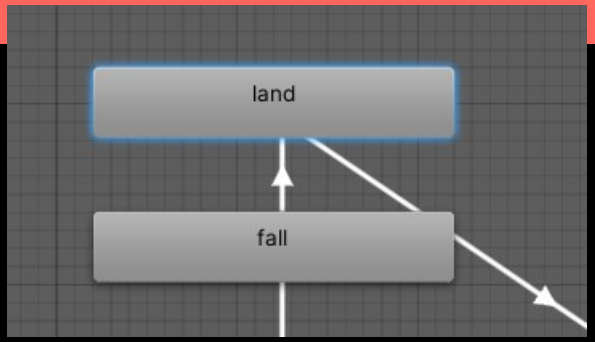
❷

```
                anim.SetBool("isGround", false);
            }
        }
    }

    public void OnLandStart()
    {
        GoToState(STATE.LAND);
    }

    public void OnLandEnd()
    {
        GoToState(STATE.IDLE);
    }

    public void OnCollisionEnter(Collision collision) {
        //GoToState(STATE.IDLE);
        anim.SetBool("isGround",true);
    }
}
```

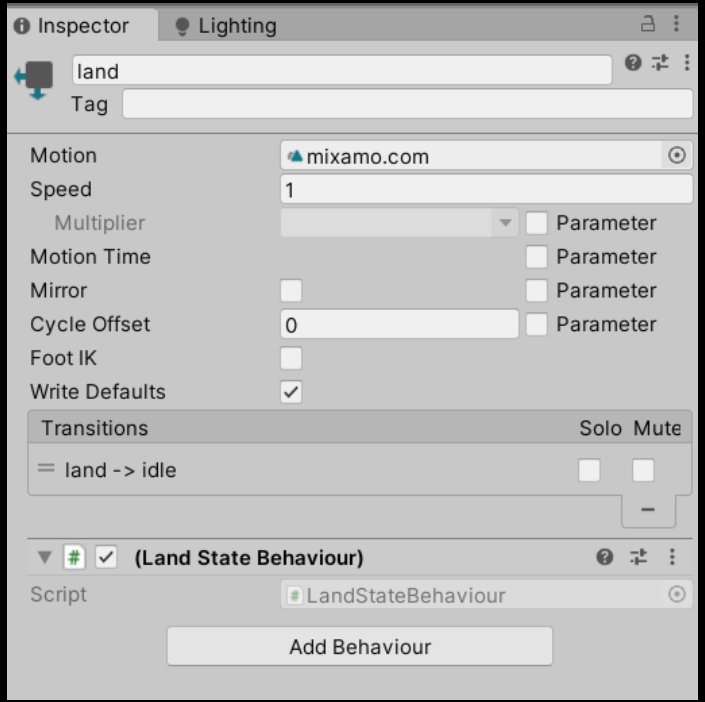
❸ ❹

Add State Behaviour

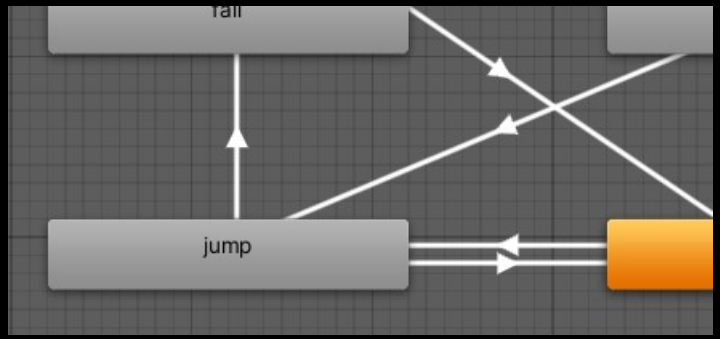
- Select land state in Animator AC. Click Add Behaviour to add C# new script and bind it to the state.
  - Name it "LandStateBehaviour"
  - Turn on OnStateEnter and OnStateExit. We use SendMessageUpwards to invoke methods on Player Controller.




```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LandStateBehaviour : StateMachineBehaviour
{
    // OnStateEnter is called when a transition starts and the state machine starts to evaluate this state
    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        animator.SendMessageUpwards("OnLandStart");
    }

    // OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks
    //override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    //{
    //
    //}

    // OnStateExit is called when a transition ends and the state machine finishes evaluating this state
    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        animator.SendMessageUpwards("OnLandEnd");
    }

    // OnStateMove is called right after Animator.OnAnimatorMove()
    //override public void OnStateMove(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
```

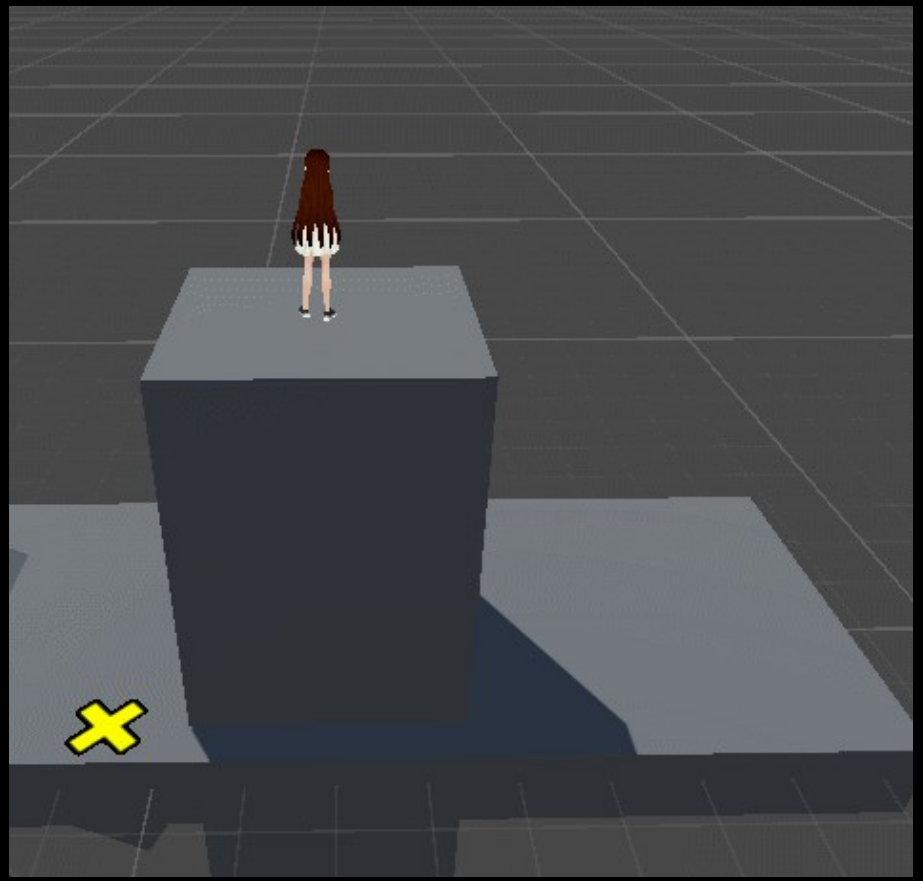


- Add another Behaviour for jump state
  - This one only need to turn on OnStateExit().




```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class JumpStateBehaviour : StateMachineBehaviour
{
    // OnStateEnter is called when a transition starts and the state machine starts to evaluate this state
    //override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    //{
    //
    //}

    // OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks
    //override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    //{
    //
    //}

    // OnStateExit is called when a transition ends and the state machine finishes evaluating this state
    override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    {
        animator.SendMessageUpwards("OnLandEnd");
    }

    // OnStateMove is called right after Animator.OnAnimatorMove()
    //override public void OnStateMove(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
    //{
    //    // Implement code that processes and affects root motion
    //}
```

- Try it. The player state will be locked in LAND state until its animation finished playing.



Add Behaviour Script to Jump State

- Method#2: Write a Behaviour script to monitor OnStateEnter/OnStateUpdate/OnStateExit.
  - We want to intercept the OnStateExit() of animator jump state.
  - Select jump state and click Add Behaviour to add a new state script. Name this state script as JumpStateBehaviour.cs.
  - Edit JumpStateBehaviour.cs. Uncomment the OnStateUpdate method.You can