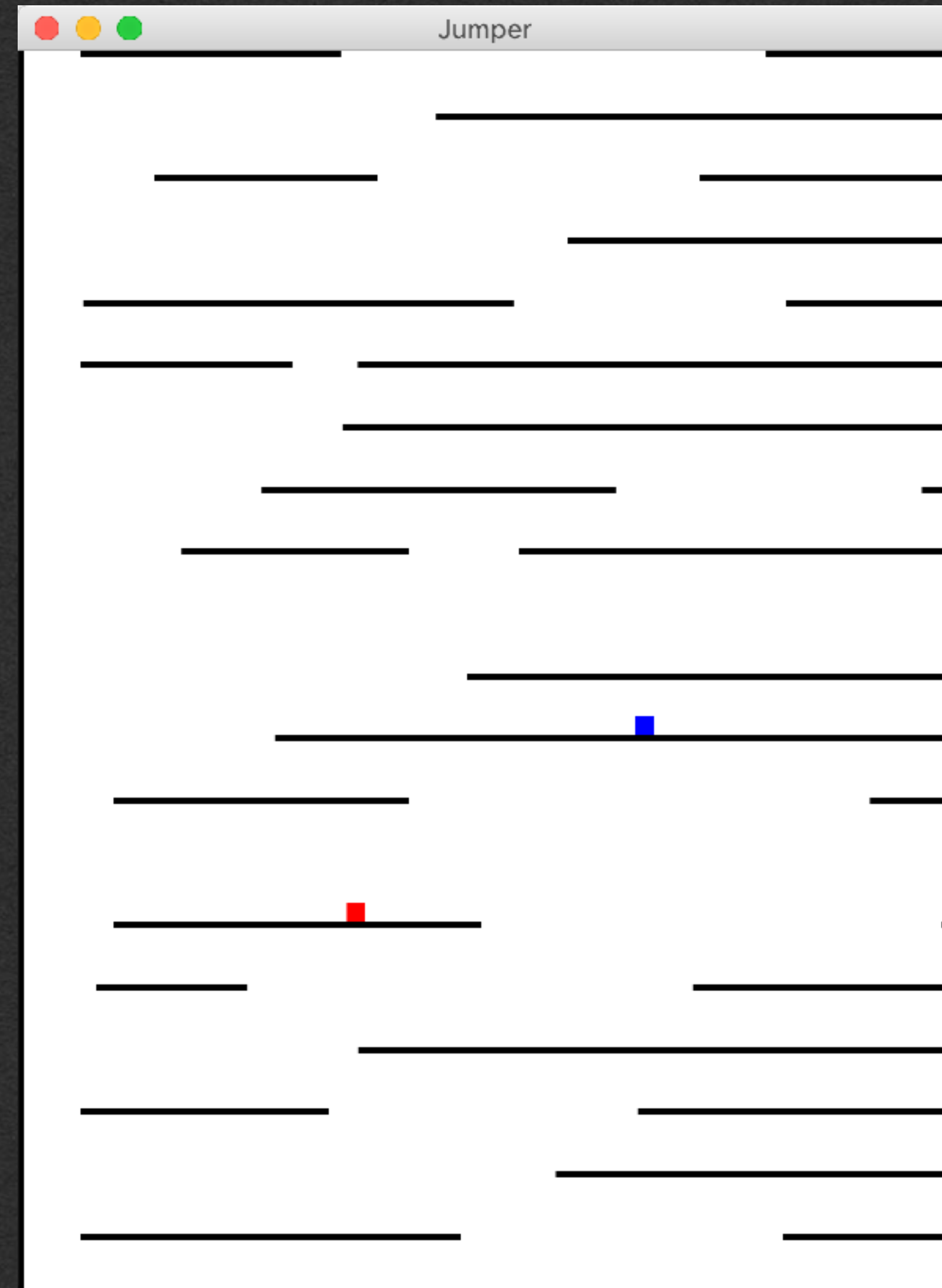


State Pattern

Jumper Example

Jumper

- 2 Player vertical scrolling platform
- Screen scrolls up as the players climb the platforms
- The bottom of the screen is game over
- **Goal:** Climb faster than the other player



Jumper - Player

How does the player move?

- User inputs
- States! <-- Good stuff

Only 3 inputs to control each player

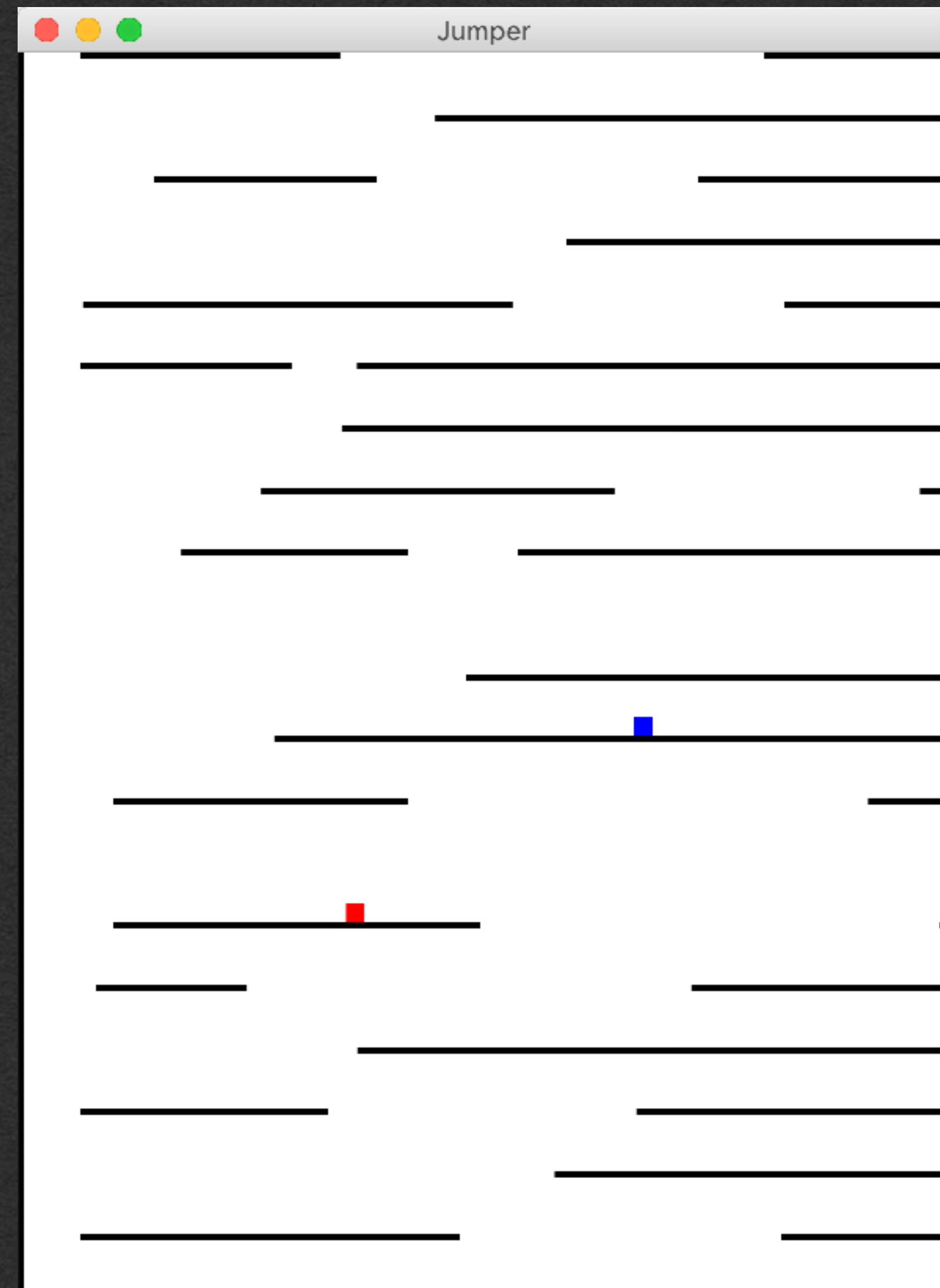
- Left button
- Right button
- Jump button

Player 1:

- a, d, w

Player 2:

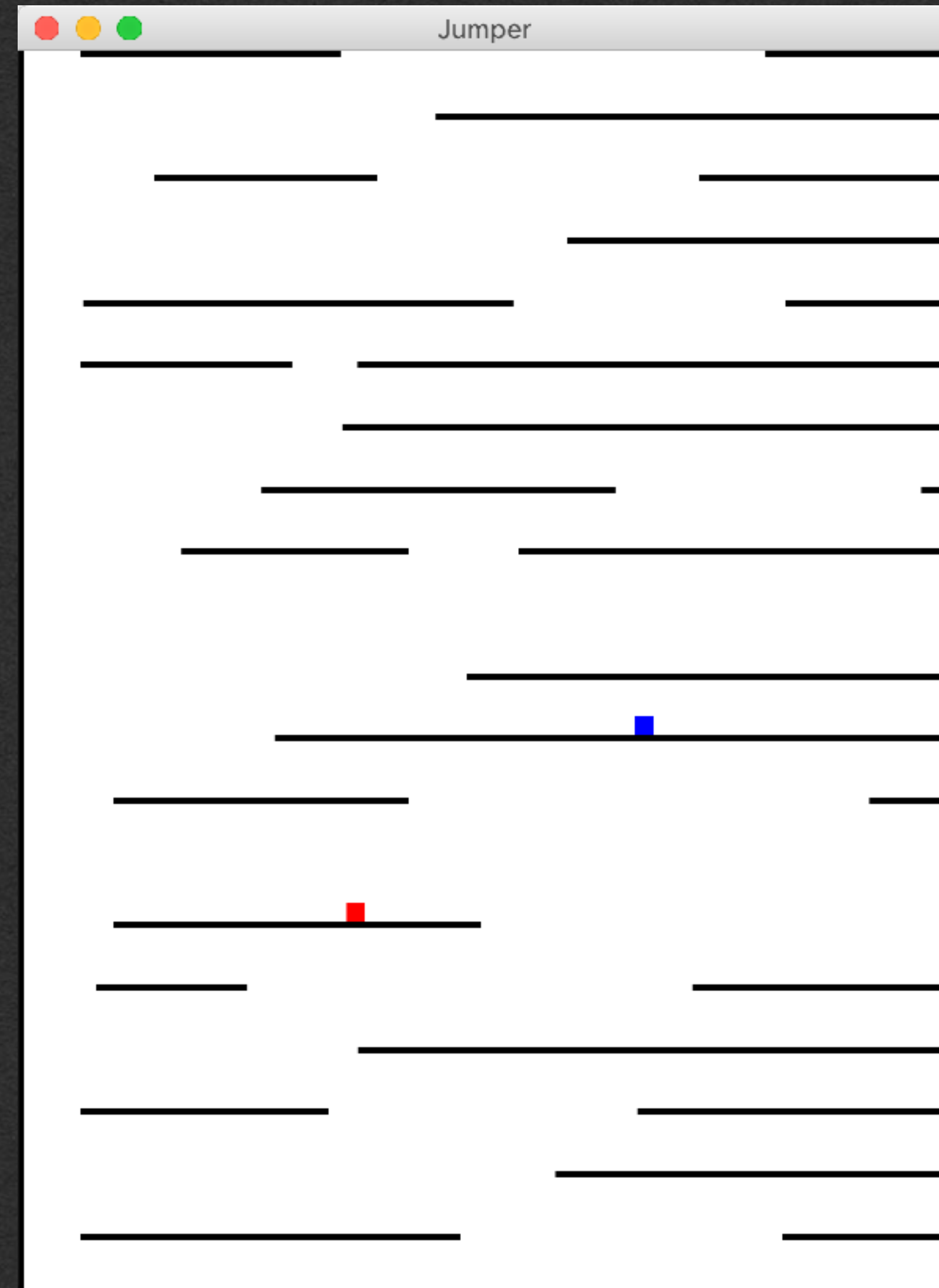
- Left, right, up arrows



Jumper Player Behavior

Each player should

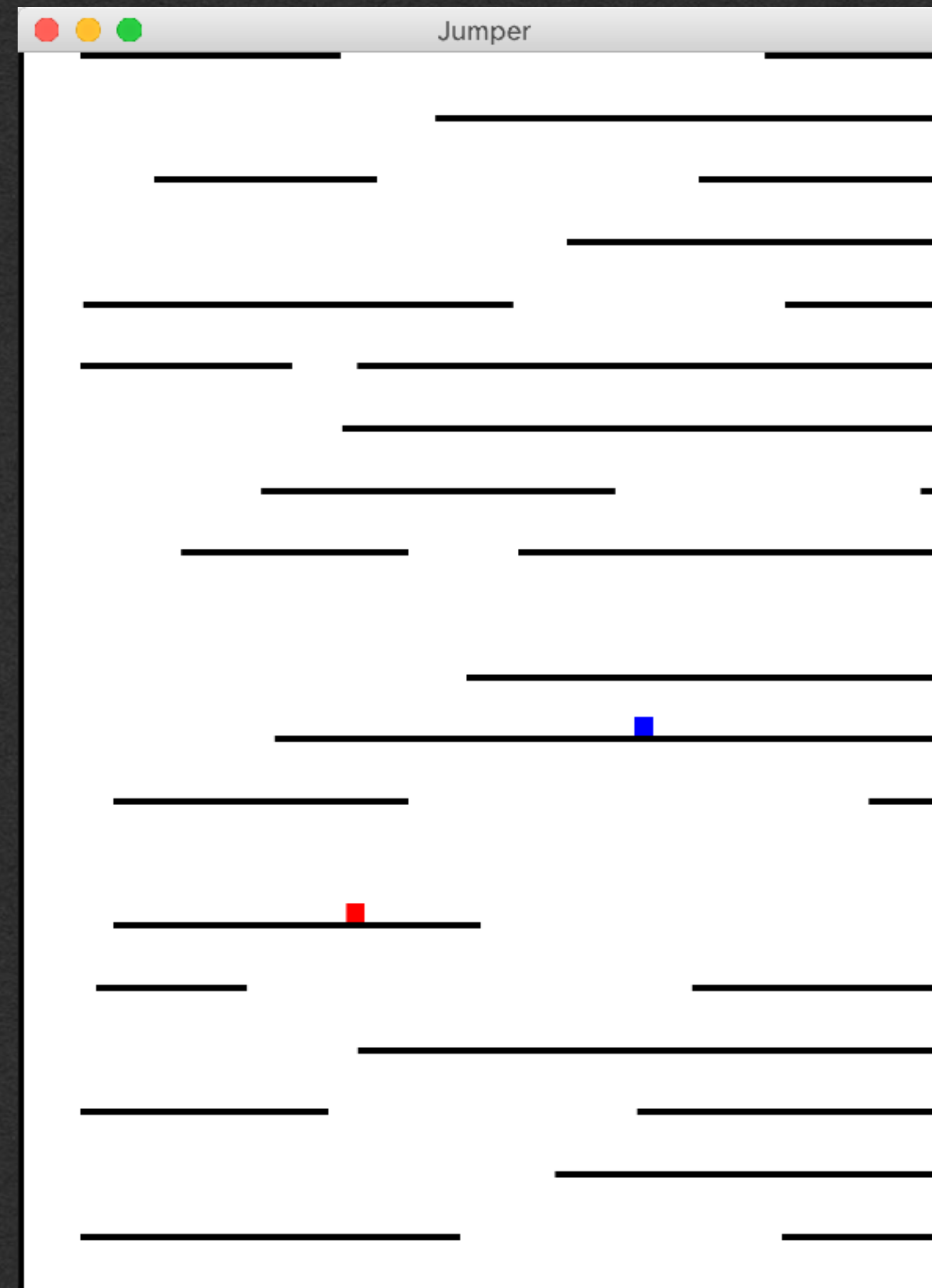
- Walk left and right when keys are pressed
- Jump when jump is pressed
- Jump higher if walking instead of standing still
- Jump at different heights based on how long the jump button is held after a jump
- Move left and right slower while in the air if the direction is changed
- Jump through platforms while jumping up
- Land on platforms while falling down
- Fall if walked off a ledge
- Block all inputs if the bottom of the screen is reached



Player behavior

We could write all this behavior without the state pattern

- Code will likely be hard to follow
- Difficult to add new features



Jumper Player Behavior

Each player should

- Walk left and right when keys are pressed
- Jump when jump is pressed
- Jump higher if walking instead of standing still
- Jump at different heights based on how long the jump button is held after a jump
- Move left and right slower while in the air if the direction is changed
- Jump through platforms while jumping up
- Land on platforms while falling down
- Fall if walked off a ledge
- Block all inputs if the bottom of the screen is reached

How to implement these features?

- Write your API
 - What methods will change behavior depending on the current state of the object
 - These methods define your API and are declared in the state abstract class
- Decide what states should exist
 - Any situation where the behavior is different should be a new state
- Determine the transitions between states

Jumper Player Behavior

Each player should

- Walk left and right **when keys are pressed**
- Jump **when jump is pressed**
- Jump higher if walking instead of standing still
- Jump at different heights based on **how long the jump button is held** after a jump
- Move left and right slower while in the air **if the direction is changed**
- Jump through platforms while jumping up
- **Land on platforms** while falling down
- Fall if **walked off a ledge**
- Block **all inputs** if the bottom of the screen is reached

How to implement these features?

- Write your API
 - What methods will change behavior depending on the current state of the object

API:

- left/right/jump pressed or released
- 6 methods
- Land on a platform

Jumper Player Behavior

Each player should

- **Walk** left and right when keys are pressed
- **Jump** when jump is pressed
- Jump higher if **walking** instead of **standing** still
- **Jump** at different heights based on how long the jump button is held **after a jump**
- Move left and right slower while **in the air** if the direction is changed
- Jump through platforms while **jumping up**
- Land on platforms while **falling down**
- **Fall** if **walked** off a ledge
- Block all inputs if the **bottom of the screen is reached**

How to implement these features?

- Decide what states should exist

States:

- Standing
- Walking
- Jumping/Rising
- Falling
- Dead (Bellow Screen)

Jumper Player Behavior

Each player should

- **Walk left and right when keys are pressed**
- **Jump when jump is pressed**
- Jump higher if walking instead of standing still
- Jump at different heights based on how long the jump button is held after a jump
- Move left and right slower while in the air if the direction is changed
- Jump through platforms while jumping up
- **Land** on platforms while falling down
- **Fall if walked off a ledge**
- **Block all inputs if the bottom of the screen is reached**

How to implement these features?

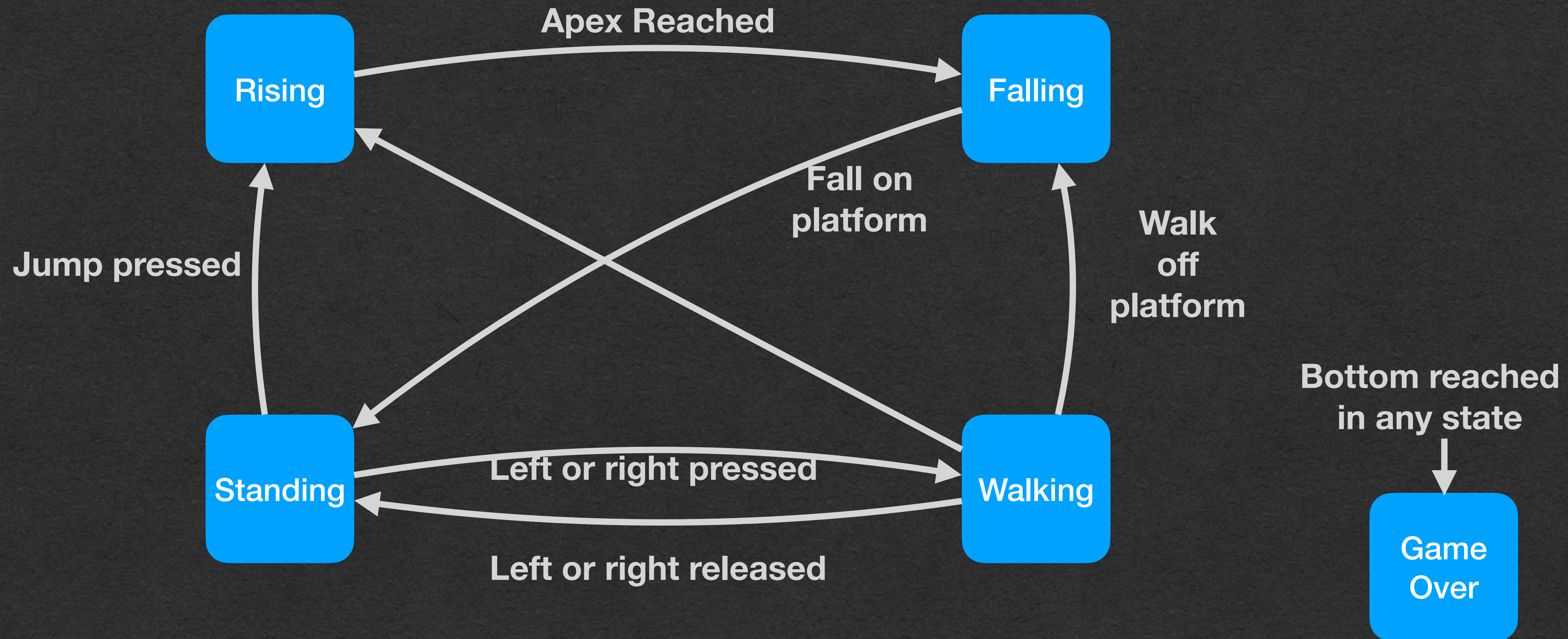
- Determine the transitions between states

State Transitions:

- Standing -> Walking
 - left/right pressed
- Walking -> Standing
 - left/right released
- Walking/Standing -> Jumping
 - Jump pressed
- Falling -> Standing
 - Land on a platform
- Walking -> Falling
 - Walk off a platform
- Jumping -> Falling
 - Apex of jump reached
- Any -> GameOver
 - Reach the bottom of the screen

Jumper Player Behavior

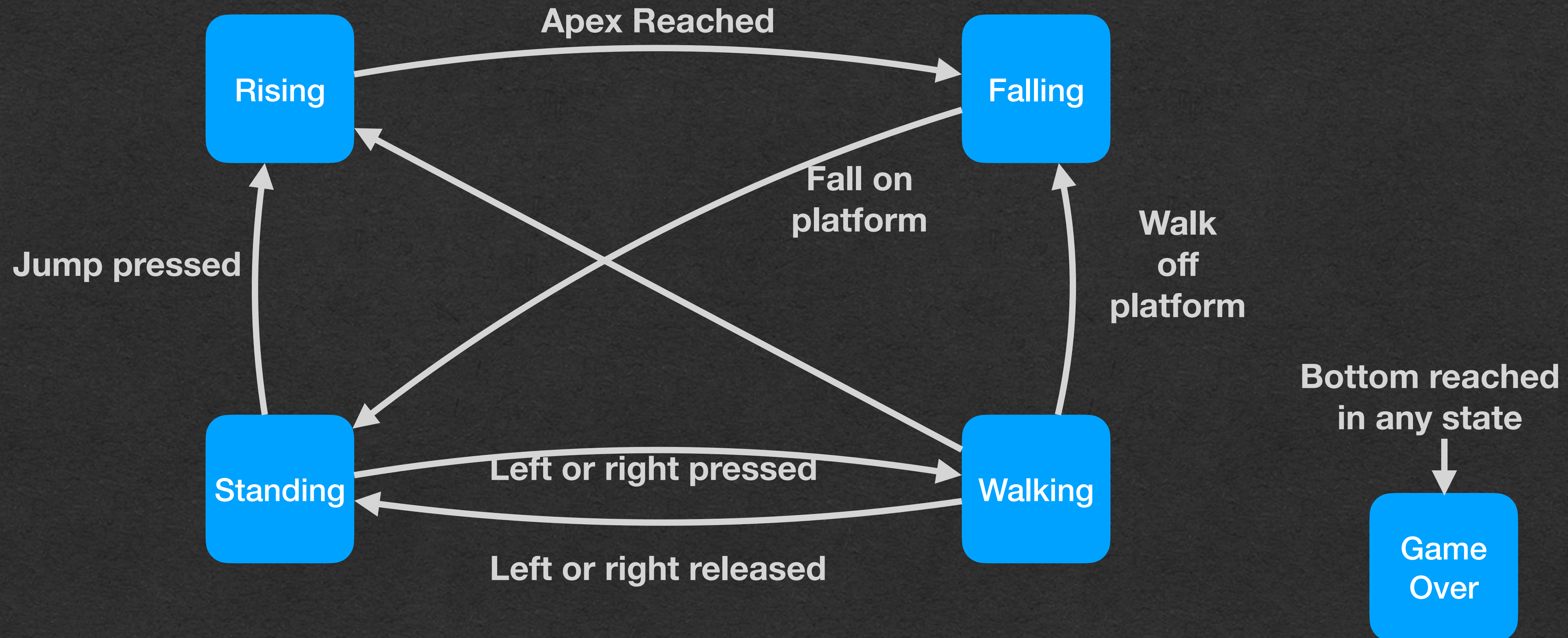
Let's visualize the states and transitions in a state diagram



Jumper Player Behavior

For each state, implement the API methods with the desired behavior in that state

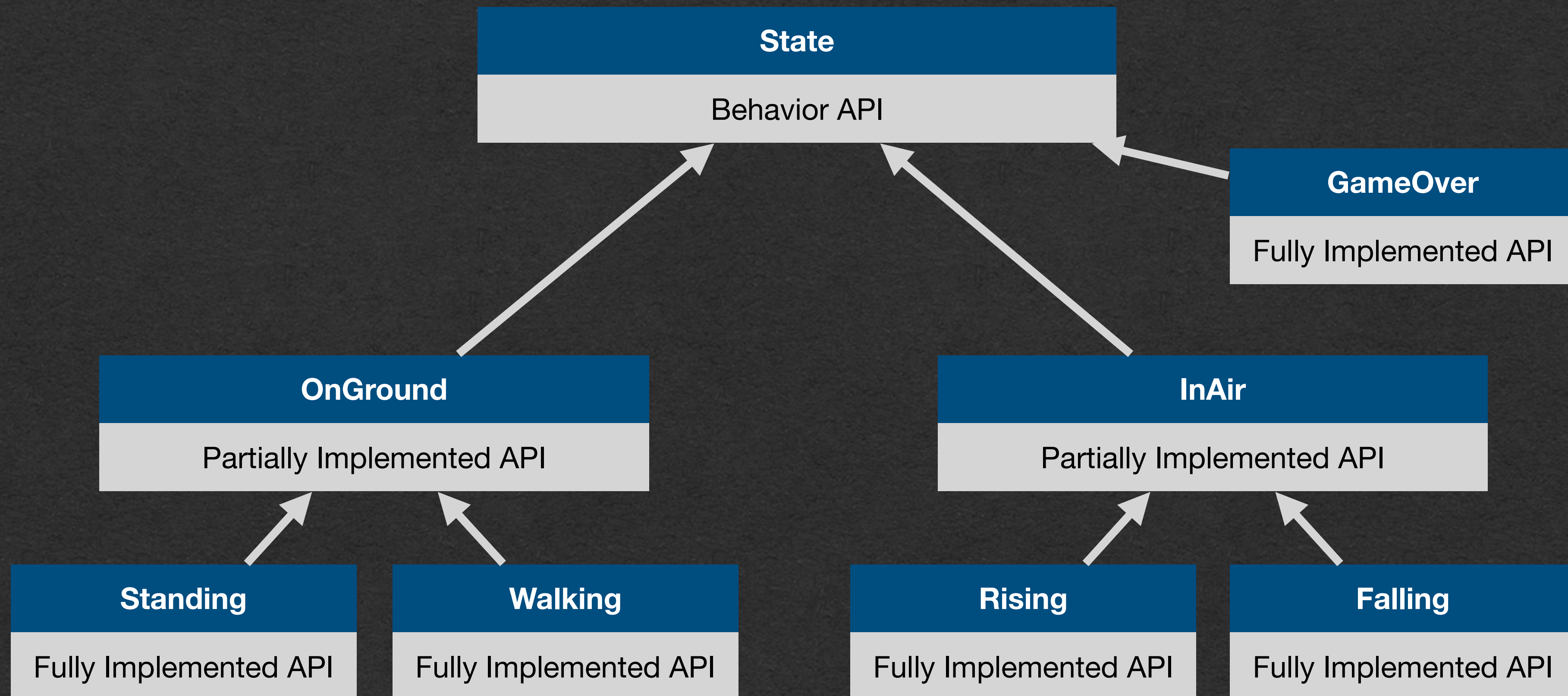
- Add default behavior in the state subclass



Jumper Player Behavior

Use inheritance to limit duplicate code

- Factor out common behavior between states into new classes



Adding Functionality

Task: Add a double jump to Jumper

- How can we add a double jump?
 - Players can jump 1 additional time while in the air
- With poor design
 - This could be extremely difficult!
 - May required modifying a significant amount of existing code
- With our state pattern
 - No problem at all

Adding Functionality

Task: Add a double jump to Jumper

- Add functionality to existing states
 - Rising and Falling states now react to the jump button by jumping again (Set velocity.z to the jump velocity)
- We'll add new states
 - RisingAfterDoubleJump/FallingAfterDoubleJump
 - Extend Rising/Falling respectively
 - Override the jump button press to do nothing
- Update state transitions
 - Press jump from Rising/Falling transitions to the respective AfterDoubleJump state
 - Reaching the apex in RisingAfterDoubleJump transitions to FallingAfterDoubleJump (Not Falling)

Adding Functionality

Task: Add a double jump to Jumper

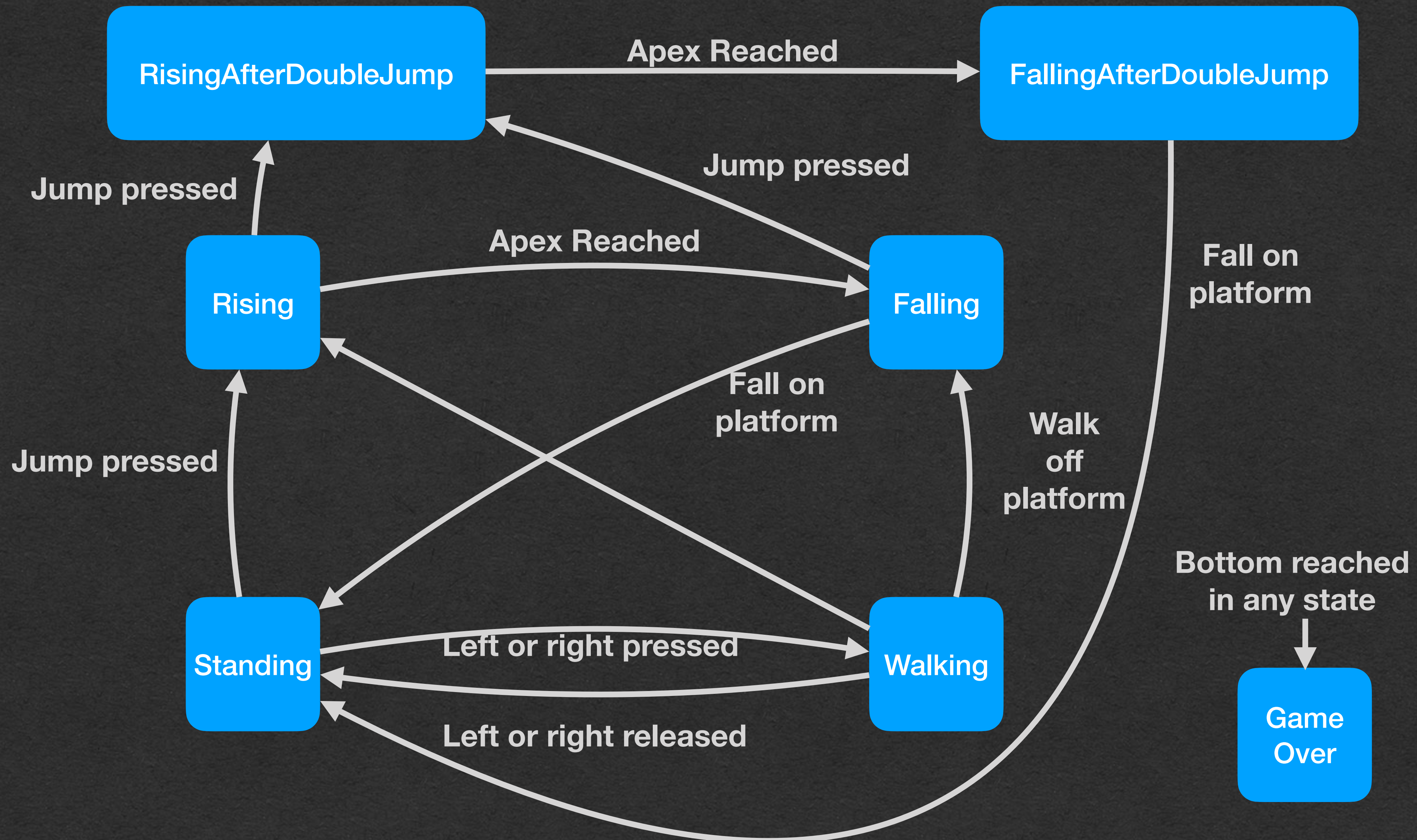
- This task could have been completed with a boolean flag instead of using new states

```
var usedDoubleJump = false

override def jumpPressed(): Unit = {
  if(!this.usedDoubleJump) {
    player.velocity.z = player.standingJumpVelocity
    this.usedDoubleJump = true
  }
}
```

- If this approach is used for many features the code will be harder to maintain
- **More to the point:** What if your professor says you can't use conditionals, but you have a situation where a button should only work once?
- Try adding more states

Jumper Player Behavior



Jumper Player Behavior

