Advanced Gameplay Programming

Week 2

HW Review

Decoupling Patterns

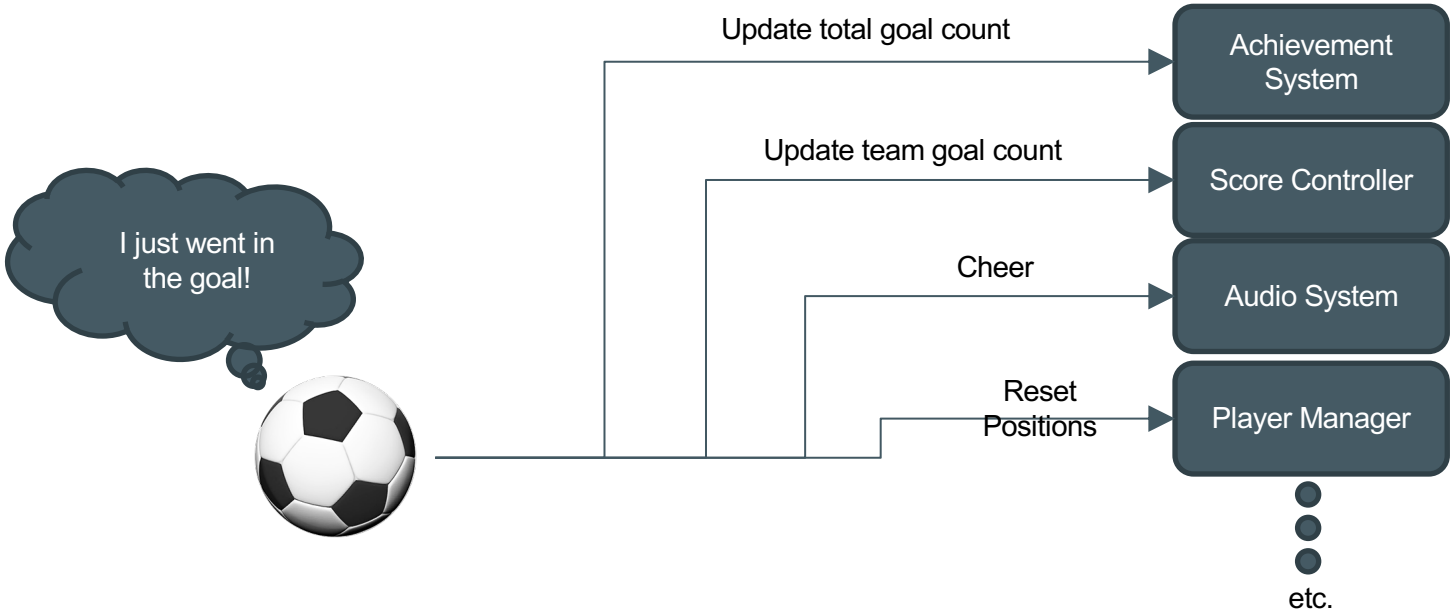
Event Managers

Events

Why use events?

- You need to communicate to a lot of different objects, or a lot of different objects need to communicate with you.
    - Points
    - Player Mode Changed
    - Game Ending
- You want state changes to be communicated without coupling all your content together.

Programming w/Out Events



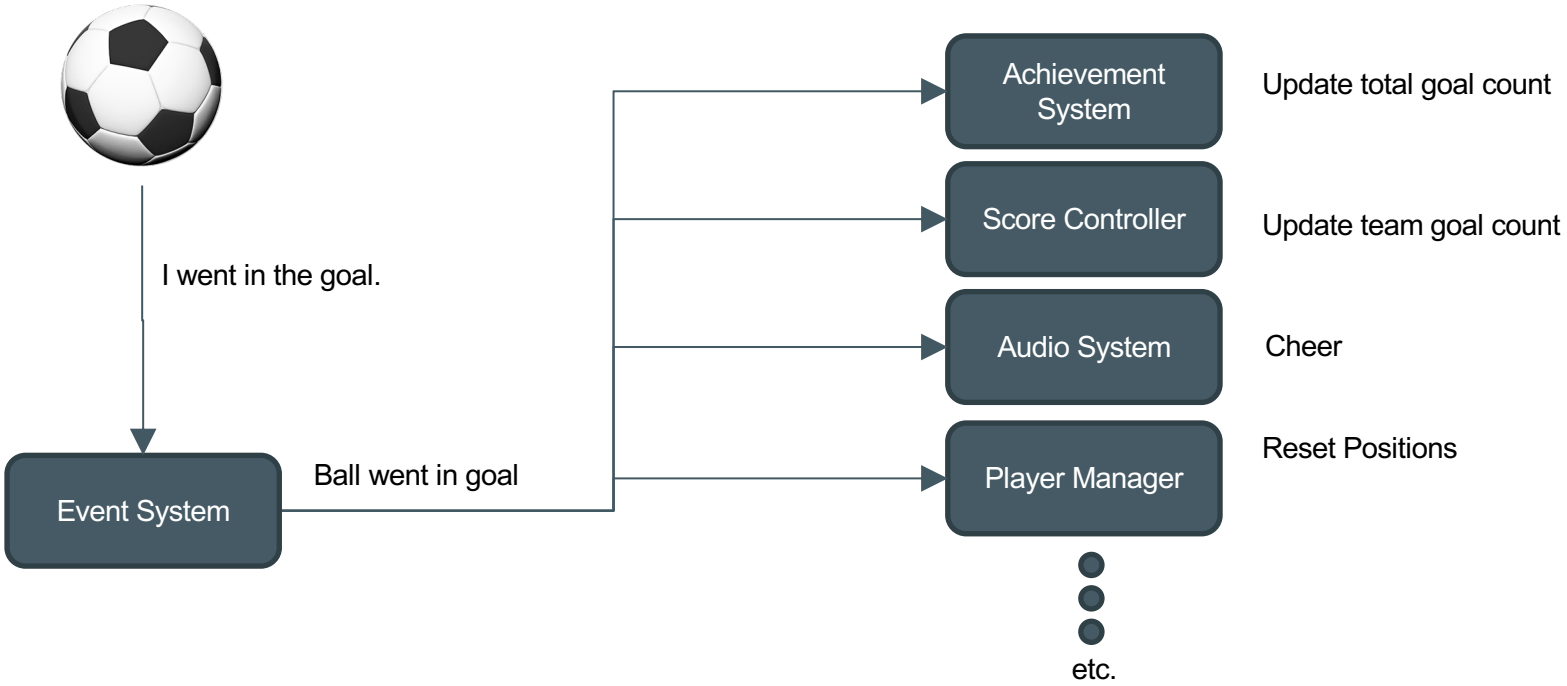
In Ball Class

```
public void OnTriggerEnter(Collider other) {
    if (other.tag == “Goal”) {
        Services.Achievements.GoalScored();
        Services.Achievements.TeamScored(scoringTeam);
        Services.ScoreManager.UpdateScore(scoringTeam, 1);
        Services.Audio.StartSoundEffect(“cheer1.wav”);
        Services.Players.ResetPositions();
        … etc.
    }
}
```

Why is this bad?

- Ball class connected every system
- Difficult to read
- Difficult to update and make changes
- Difficult to debug.

Programming with Events



In ball class

```
public void onTriggerEnter(Collider other) {
        if (other.tag == “Goal”) {
    Services.Events.Fire<GoalScored>(other.name);
      }
}
```

In each of the other classes:

```
private void OnGoalScored() {
    // do what happens when a goal is scored
}
```

Why is this better?

- Code is modular
- Code is decoupled
- Code is easy to read
- Code is simpler / easier to understand

Why is this worse?

- Debugging becomes harder.
- If you don’t handle registering and unregistering for events, you can have crashes or bizarre behavior
- It’s easy to overuse this pattern.

Example Events

- Player Died
- Goal Scored
- Item picked up
- Enemy Spawned
- Game Over
- Level Over
- Basically, anything *multiple systems* would care about.

Code Example - In “ScoreManager.cs”

```
private void Start()
{
    Services.EventManager.Register<GoalScored>(IncrementTeamScore);
}

private void OnDestroy()
{
    Services.EventManager.Unregister<GoalScored>(IncrementTeamScore);
}

private void IncrementTeamScore(AGPEvent e)
{
    // what happens to increment score
}
```

Code Example – Firing an Event

```
public class GoalScored : AGPEvent
{
    // whatever state variables you need
    bool foo;
    int bar;

    public GoalScored(bool fooIn, int barIn)
    {
        this.foo = fooIn;
        this.bar = barIn;
    }
}

Services.EventManager.Fire(new GoalScored(gameObjectName == "Blue Goal"));
```

Important Notes

- BroadcastMessage / SendMessage
- You have to remember to **unregister handlers** when the object is destroyed

Finite State Machines

Where's the bug?

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.B))
    {
            princessRigidBody.velocity = JUMP_VELOCITY;
            setGraphics(IMAGE_JUMP);
    }
}
```

How do you fix it?

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.B) && !isJumping)
    {
            isJumping = true;
            princessRigidBody.velocity = JUMP_VELOCITY;
            setGraphics(IMAGE_JUMP);
    }
}
```

Where's the bug now?

```
if (Input.GetKeyDown(KeyCode.B) && !isJumping)
{
    // Jump if not jumping...
}
else if (Input.GetKeyDown(KeyCode.S))
{
    if (!isJumping)
        setGraphics(IMAGE_DUCK);
}
else if (Input.GetKeyUp(KeyCode.S))
{
    setGraphics(IMAGE_STAND);
}
```

Add another flag...

```
if (Input.GetKeyDown(KeyCode.B) && !isJumping && !isDucking)
{
    // Jump...
}
else if (Input.GetKeyDown(KeyCode.S) && !isJumping) {
    isDucking = true;
    setGraphics(IMAGE_DUCK);
}
else if (Input.GetKeyUp(KeyCode.S) && isDucking) {
    isDucking = false;
    setGraphics(IMAGE_STAND);
}
```

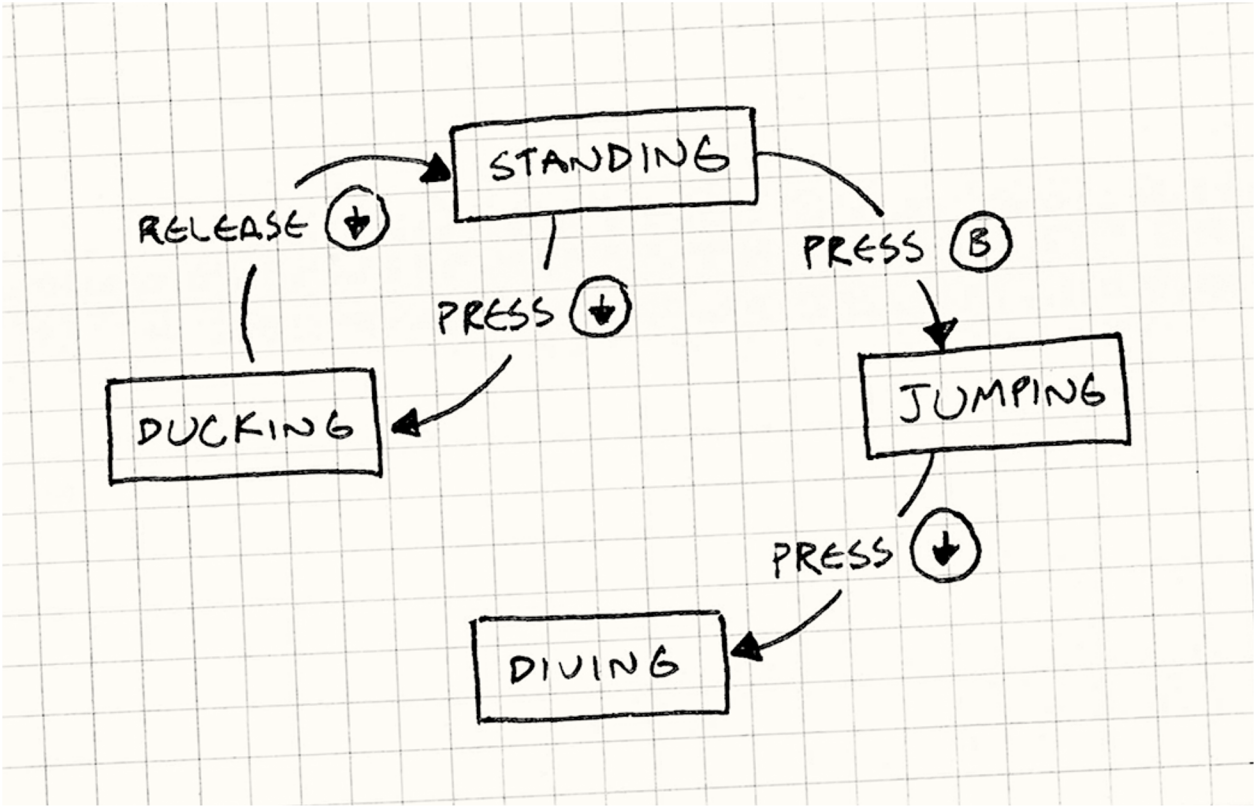
Where's the bug now?

```
else if (Input.GetKeyDown(KeyCode.S)) {
    if (!isJumping) {
        isDucking = true;
        setGraphics(IMAGE_DUCK);
    } else {
        isJumping_ = false;
        setGraphics(IMAGE_DIVE);
    }
}
```

Something Is Wrong

- This movement code is already dozens of lines with multiple conditionals.
- Every time we touch this code, it breaks.
- We still have a lot of stuff we have to include
  - walking
  - running
  - other attacks
- Our code doesn’t look anything like what we mean.

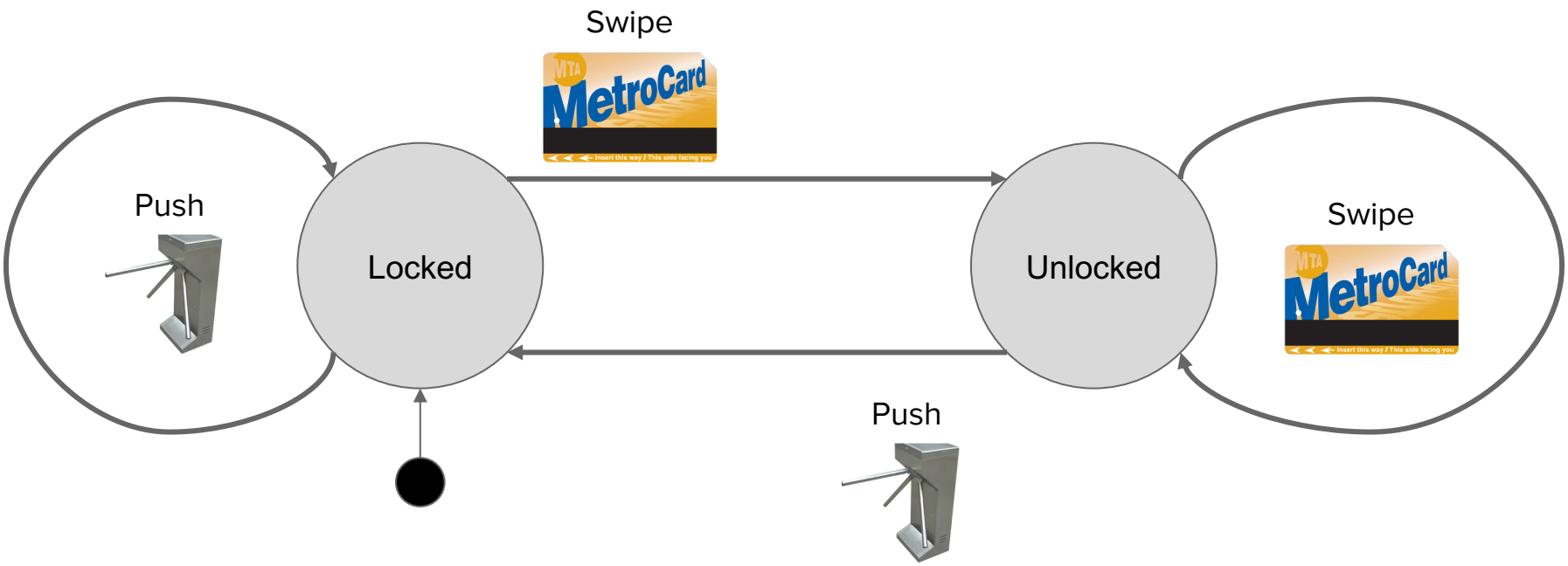
What Do We Want?



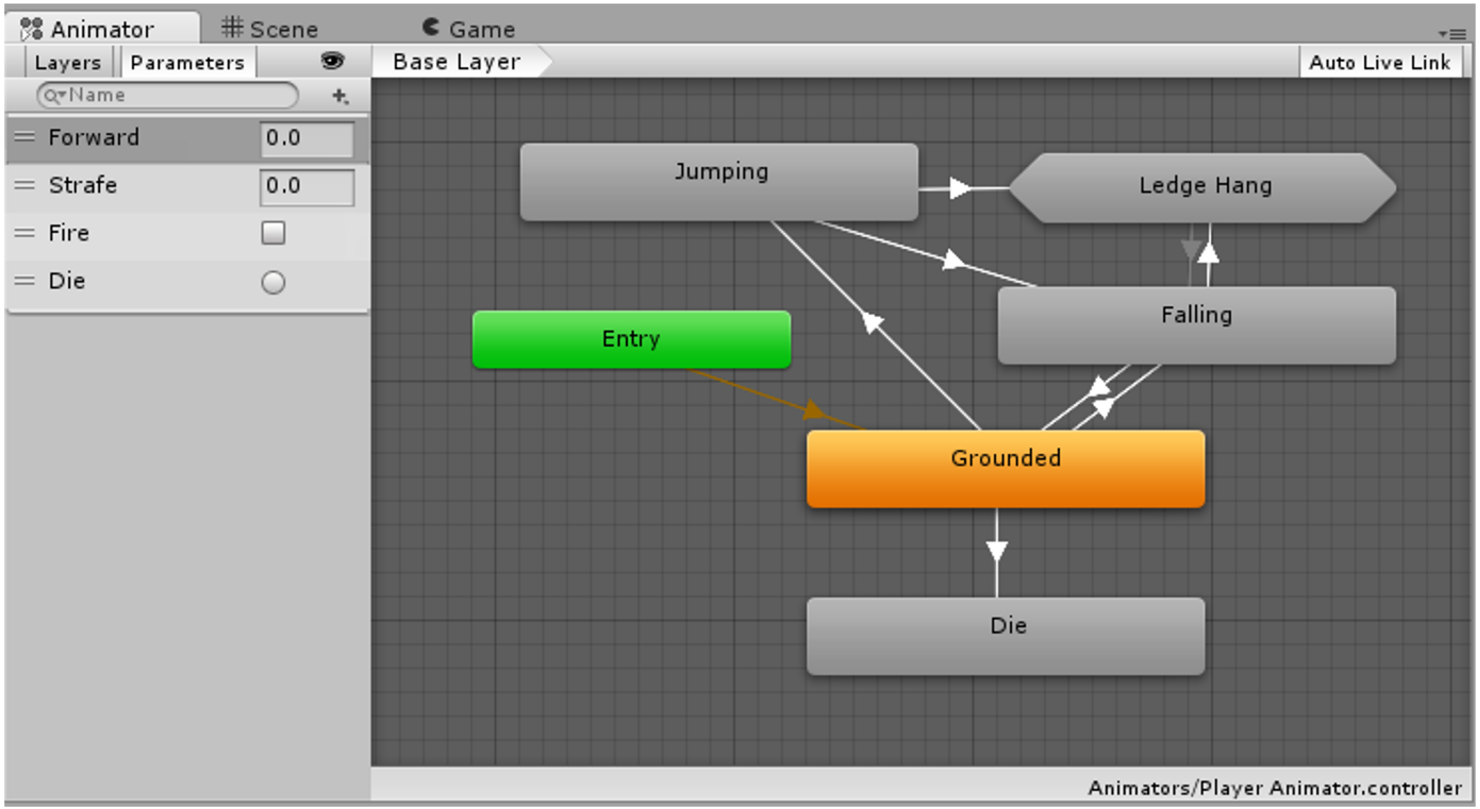
Finite State Machine

- You have a fixed set of states that the machine can be in.
- The machine can only be in one state at a time.
- A sequence of inputs or events is sent to the machine.
- Each state has a set of transitions, each associated with an input and pointing to a state.

What is a Finite State Machine (FSM)?



Have We Seen This Before?



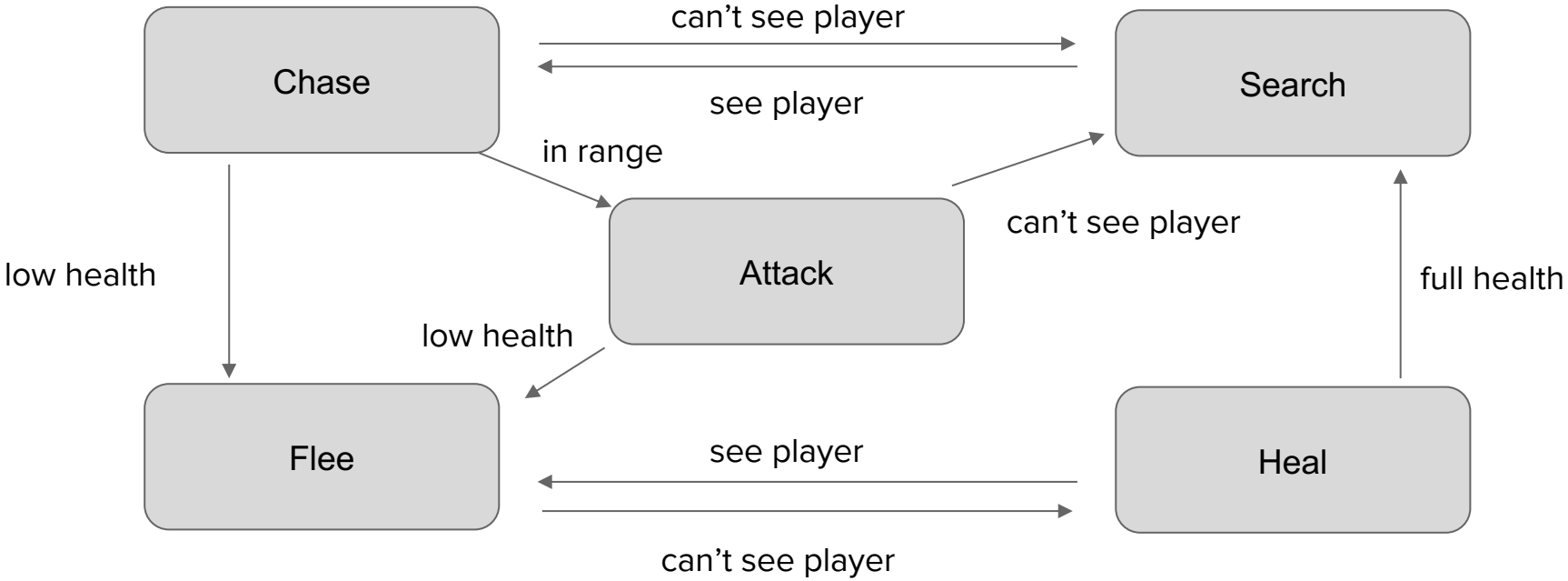
Code Example – Creating State Machine

```
private FiniteStateMachine<AIPlayer> _AIPlayerStateMachine;

void Start()
{
    _AIPlayerStateMachine = new FiniteStateMachine<AIPlayer>(this);
    _AIPlayerStateMachine.TransitionTo<IdleState>();
}

void Update()
{
    _AIPlayerStateMachine.Update();
}
```

Code Example – Defining States

```
private abstract class AIPlayerState : FiniteStateMachine<AIPlayer>.State
{
    public override void OnEnter()
    {
        // initialization
    }
    public override void Update()
    {
        // update
    }
    public override void OnExit()
    {
        // on exit
    }
}
```

Does it need to be limited to Inputs?



When is this useful?

State machines help untangle complicated code by enforcing a structure on it.

All you've need to track is:

- a fixed set of states
- a single current state
- hardcoded transitions

You have an entity whose behavior

- changes based on some internal state
- that state can be rigidly divided into a small set of options
- responds to inputs or events over time

When is this not useful?

State machines help untangle complicated code by enforcing a structure on it.

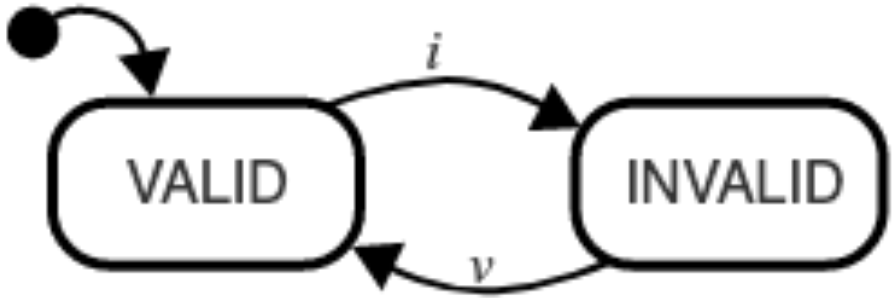
- This structure doesn't fit anything more complicated, or that can be in multiple states at once.

All you've got is a fixed set of states, a single current state, and some hardcoded transitions.

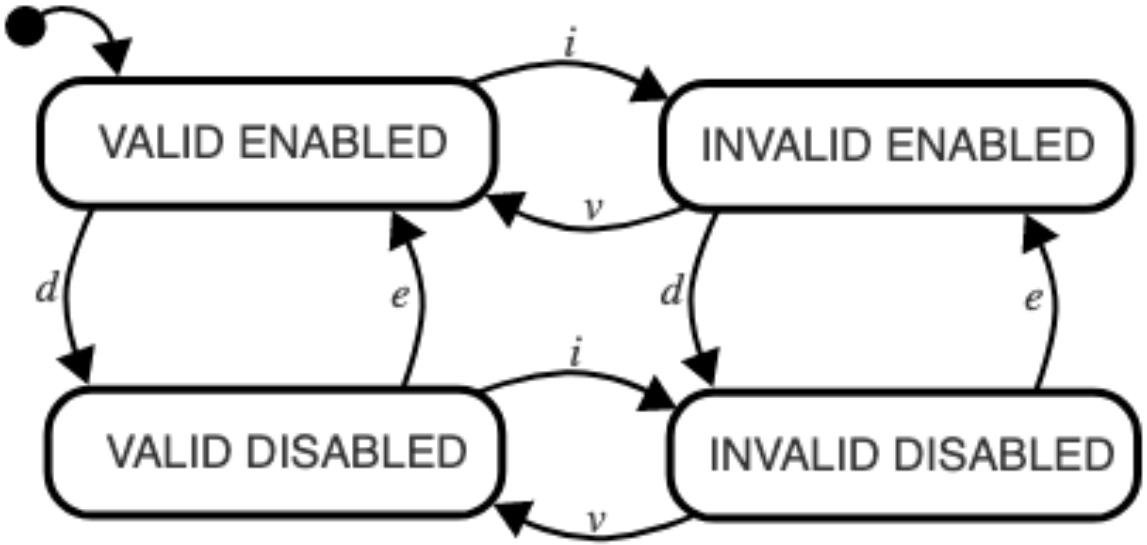
- It's easy to make mistakes in the transitions, putting parts of your game in locked states

This is very limited: It isn't a good solution for (modern) AI.

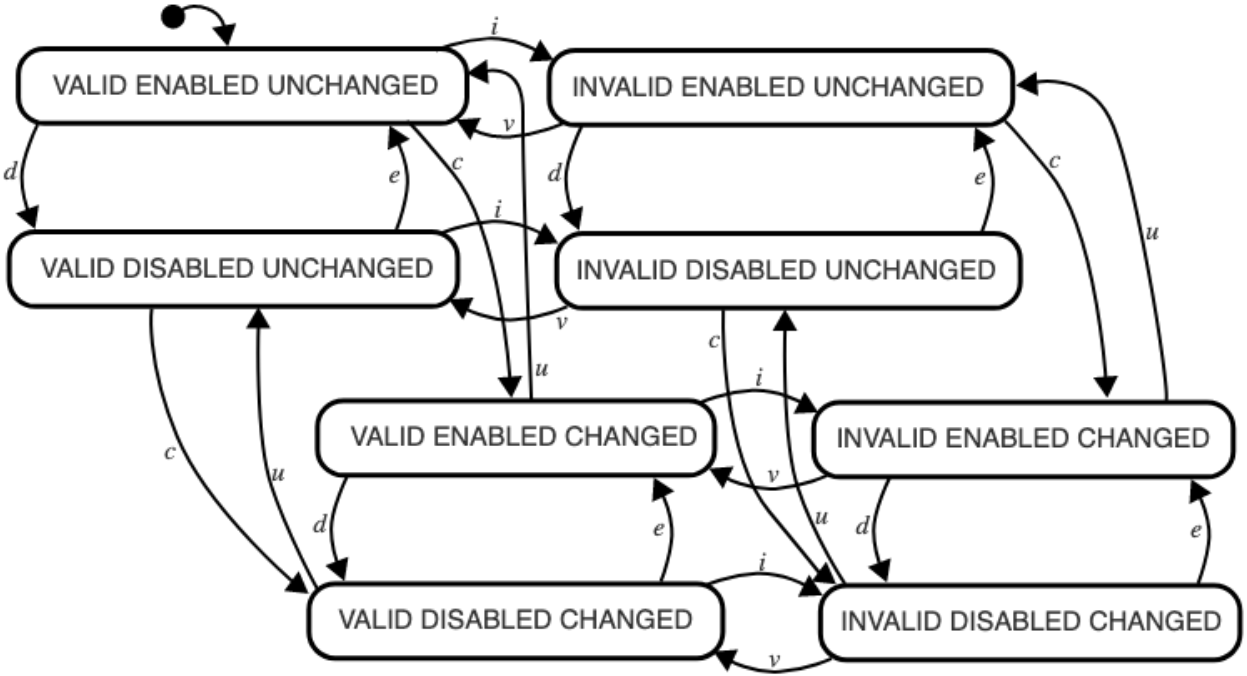
State Explosion



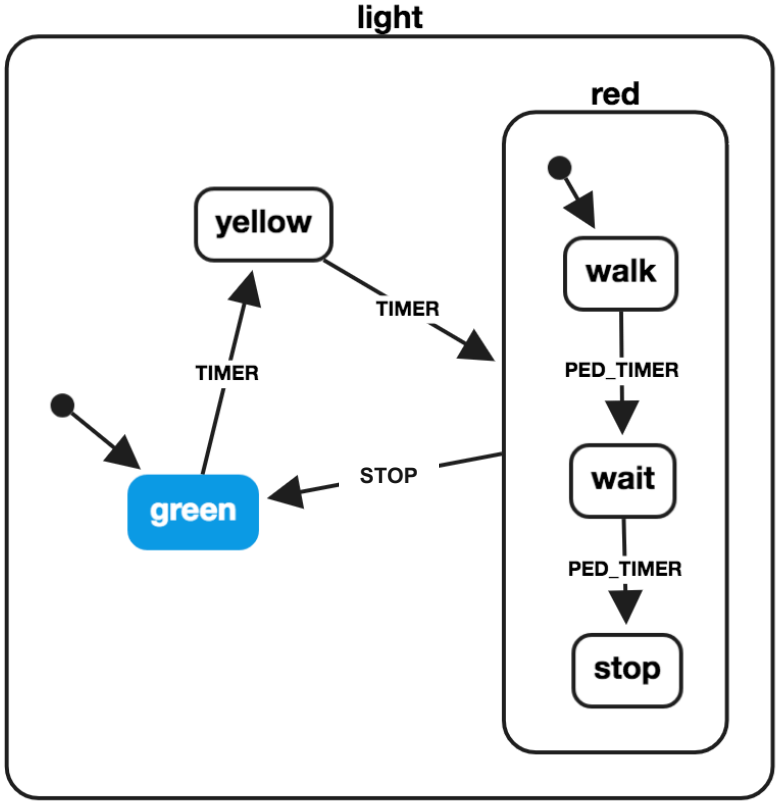
State Explosion

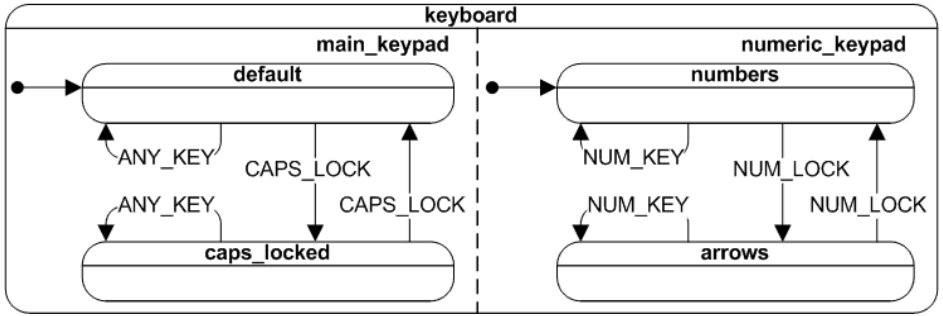


State Explosion



Heirarchical State Machines

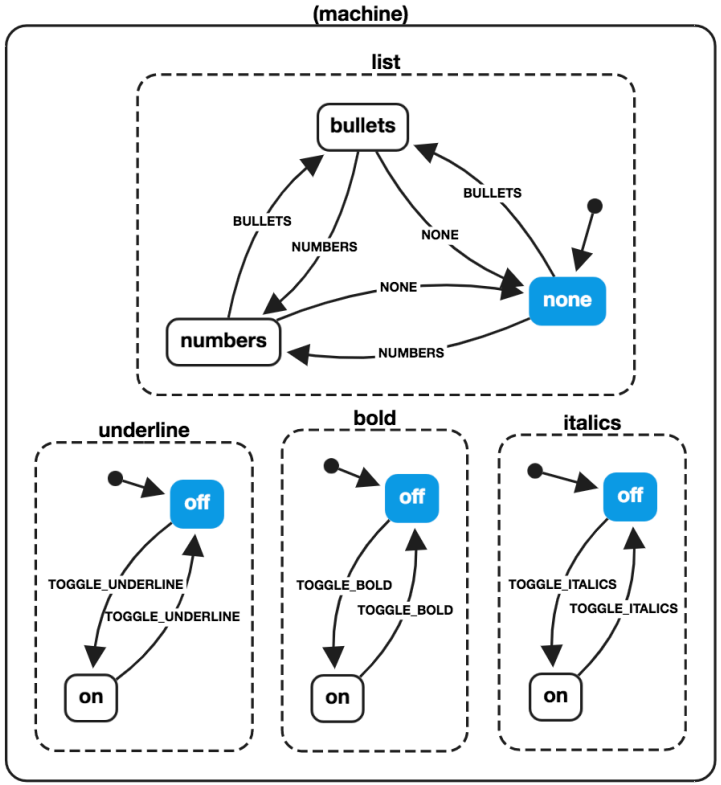




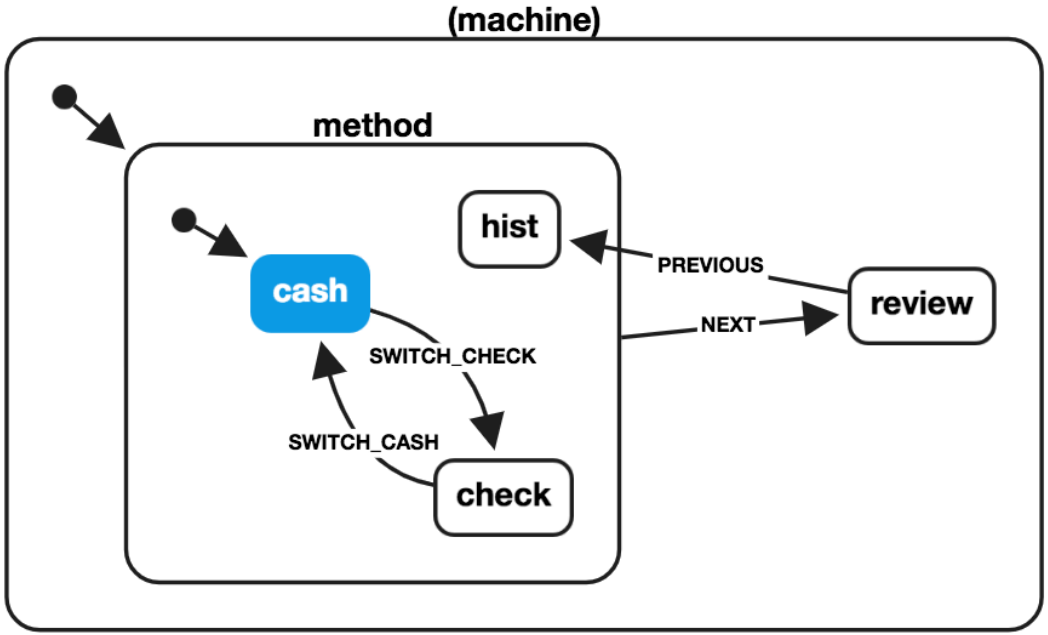
Example Heirarchical State Machines

- Gamestates that have “sub-state” machines
  - Tutorial state w/ steps in tutorial
  - A turn based game w/ multiple phases per turn
- A boss w/multiple “states” that each function differently
- A puzzlebox that you can view at multiple levels / has cascading change behavior
- Multiplayer game w/ individual gameplay that affects what phase the game is in

Parallel State Machines



Historical State Machines



Other possible features

- Concurrent regions
- Event deferral
- Event blocking