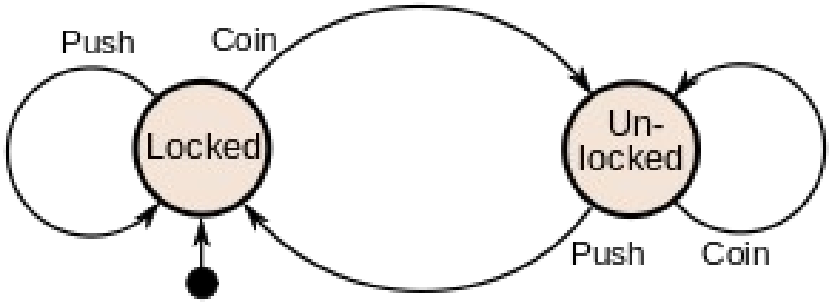
Games AI

Lecture 1.2

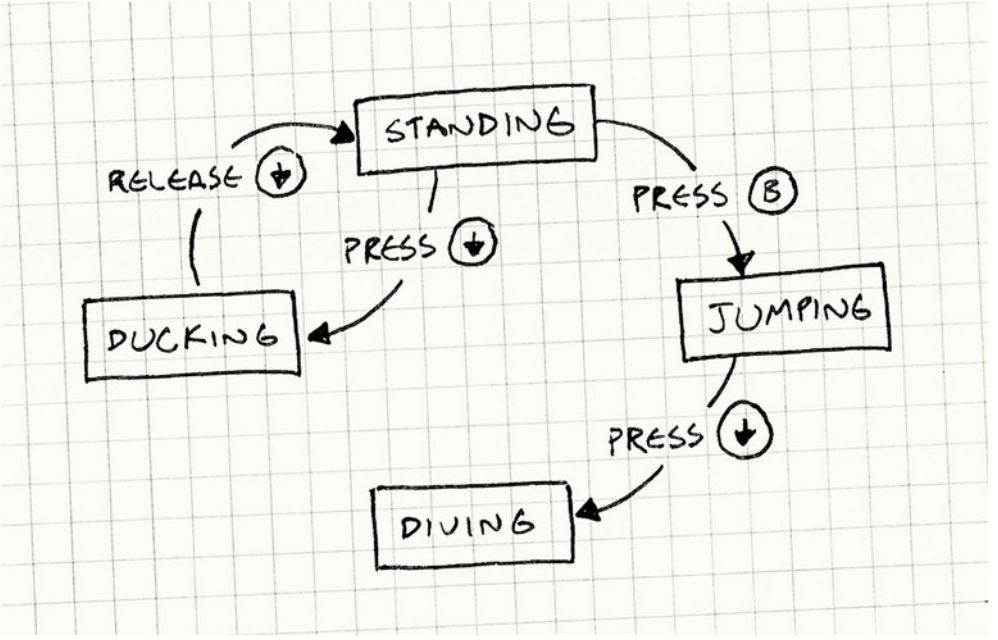
Finite State Machines

- Finite State Machines
  - Simple idea
  - Widely used
  - Several variations

- Agent can be in one of a finite set of states
- States can be connected by transitions
- Transitions are triggered by events

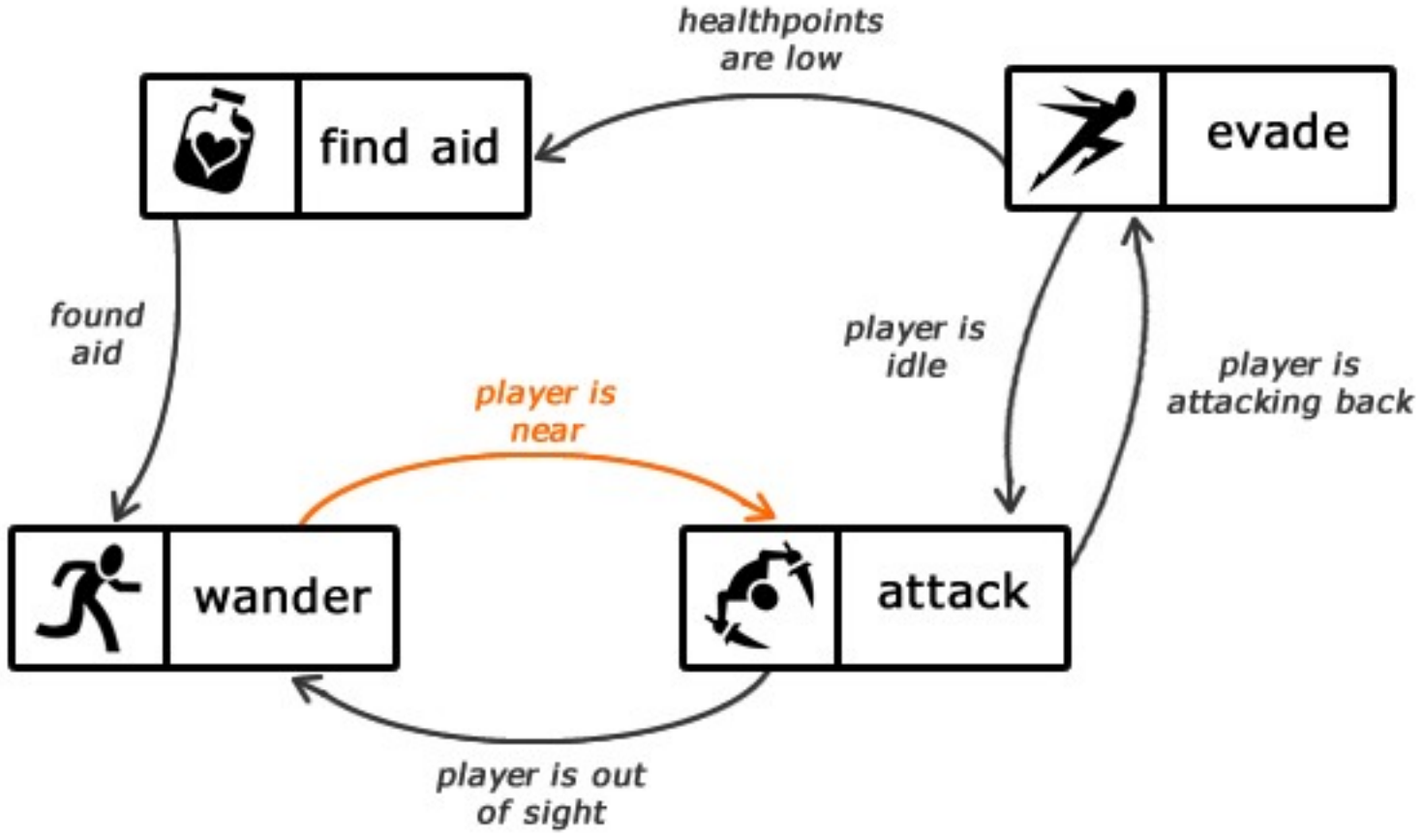


- Break down complex systems into a discrete set of states
- Each state performs a narrowly defined task



- Benefits
  - Easily understood
  - Small and easy to code
  - Negligible overhead
  - Various extensions to the idea
- Limitations
  - Simple formalism means can’t handle complexity

Finite state machines



- Where can you see FSMs in use in games?
  - Opponent AI
  - Animation
  - UI
  - Game design and mechanics
  - and more...

- Lets design some FSMs together for agents that you might want to implement
  - Turret?
  - Mob?
  - Bullet hell?

- Implementing an FSM (1)
  - Switch statement

```
let currentState: State

switch (currentState)
{
    case State.Idle:
        //do something
        break;
    case State.Attack:
        // attack the player
        break;
    case State.Flee:
        // run away
        break;
}
```

- Pros
  - Simple
- Cons
  - State update code all in one place

- Implementing an FSM (2)
  - Encapsulate in a class
  - Implement each state as a function or method
  - Use a property to determine active state
  - Call active state function per game update

```
export class FSM
{
    private activeState: () => void;

    public setState(state: () => void) :void
    {
        this.activeState = state;
    }

    public update(): void
    {
        if (this.activeState != null)
            this.activeState();
    }
}
```

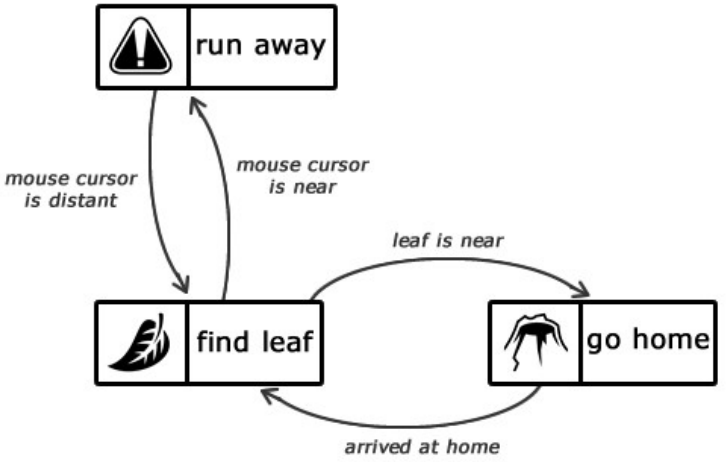
- Problem
  - Potential for invalid state transitions

- A rethink:
  - Store current state
  - Accept events (e.g. function calls to public interface)
  - Lookup next state
  - Change state
  - Run update for current state

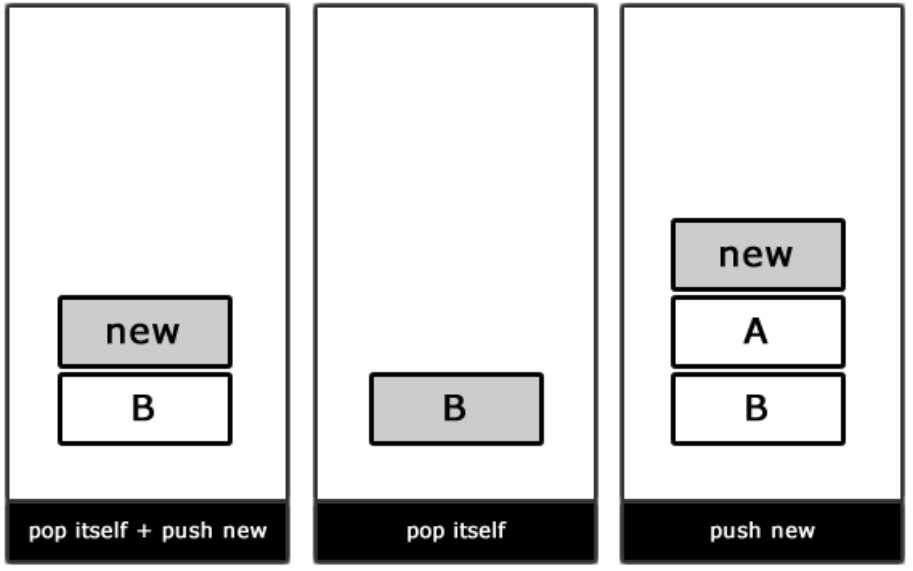
- Go and implement an FSM AI
  - Start with a simple implementation
  - Get it doing something cool
  - Improve your implementation to make it more reliable
- Then think about how your code can scale and fit into a larger AI system
  - What limitations do you find with FSMs?


More State Machines
30
84
+85
GeorgeROFLington
FATAL
Bernie Noel

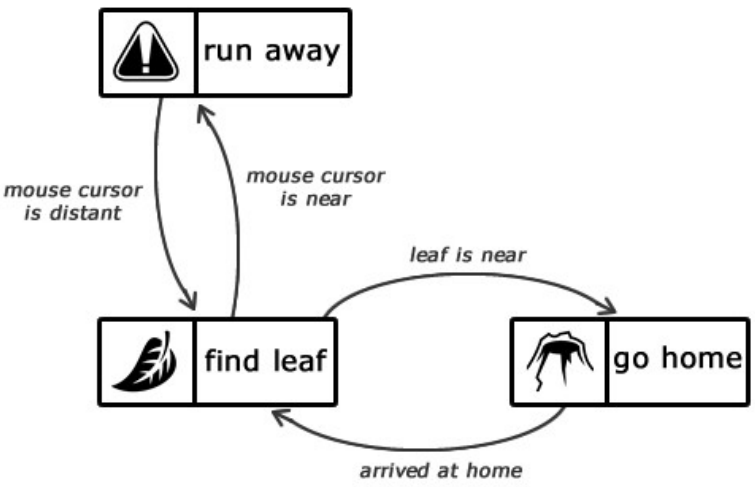
- Problem: FSMs don’t store history
  - Need to use duplicate states
  - Can quickly get unmanageable



- Solution 1
  - Stack-based FSM
  - Top of stack points to current state
  - State transitions: push, pop, pop and push
  - How would we implement our “run away” state?



- Solution 2:
  - Hierarchical State Machines
  - States can contain sub-FSMs
  - Each state stores the current state for its internal FSM
  - How would we update our ant state machine?



- State machines are a widely used tool, but best when combined with other techniques
- State/event oriented, hard to design goal-oriented behaviour
- Adding concurrency and non-determinism can increase sophistication/realism

- Implement a hierarchical FSM
  - How does it compare to a standard FSM for creating more complex behaviours?
  - Make your code robust and reusable
    - You might want to use these techniques in your assessment