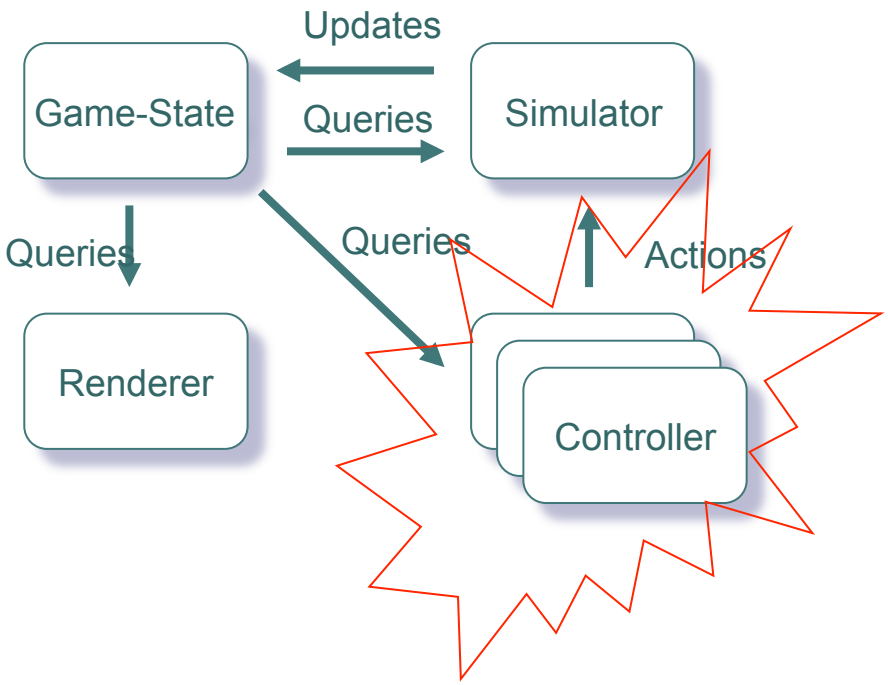
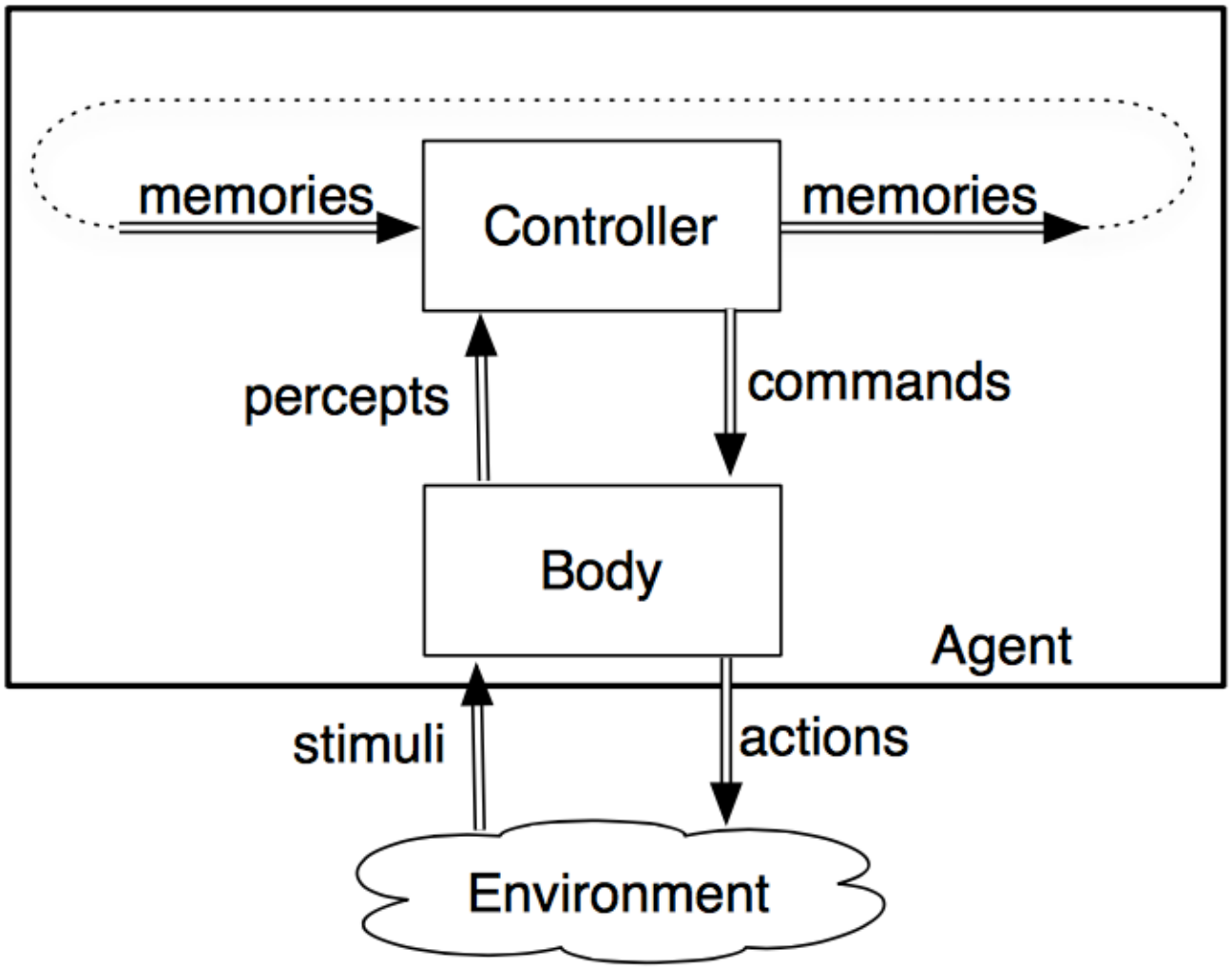
Game Engine Architecture



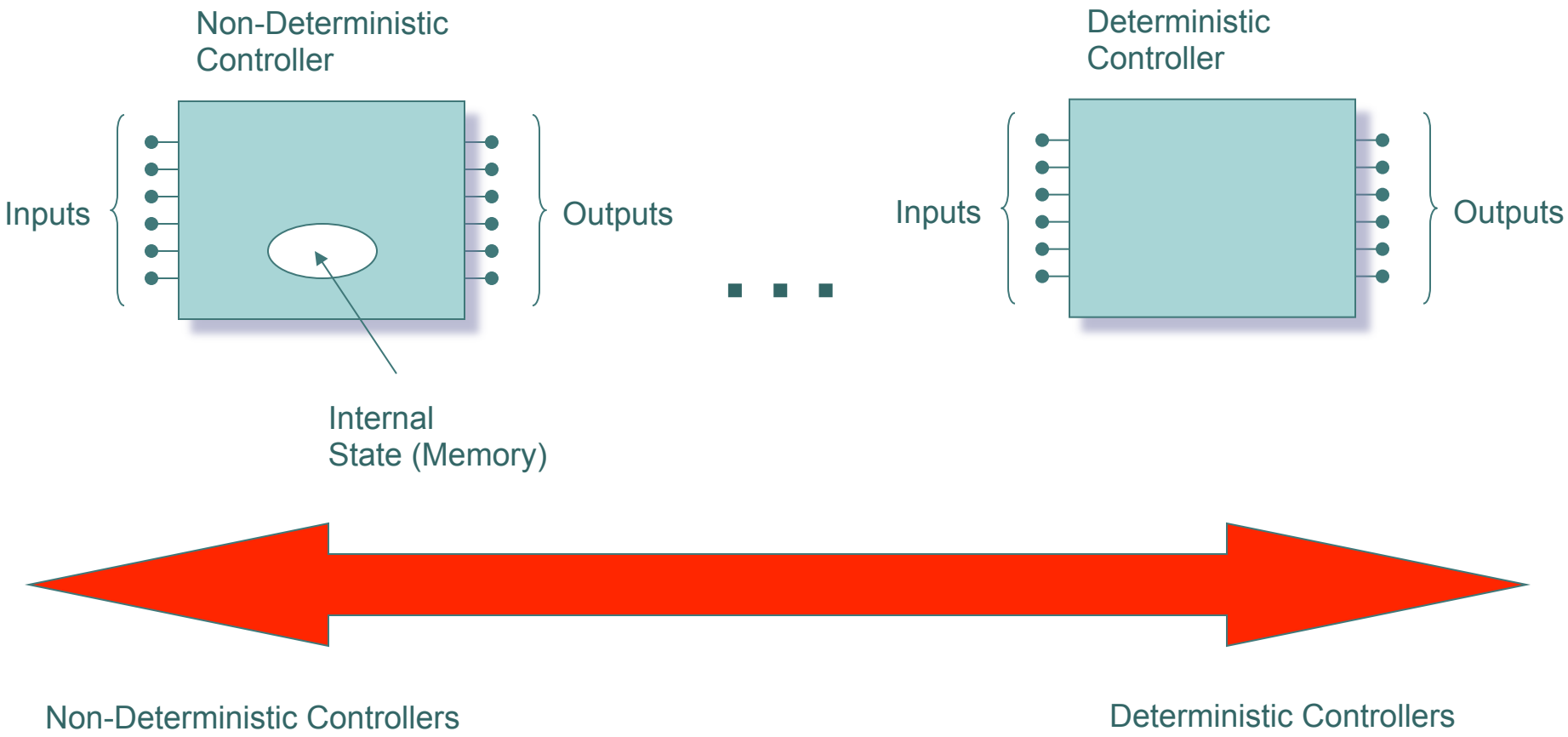
Source: AI for Computer Games, John D. Funge, AK Peters, 2004.

- Game-State – the game-state represents the current state of the world. It knows about all the objects in the world and provides access to them so that they can be queried by all the other components for information about their current state.
- Simulator – the simulator encodes the rules of how the game-state changes based on the "game physics." Together with a set of animations the simulator is responsible for generating a character's moves in response to the actions chosen by the associated controller.
- Renderer – together with the game's geometry and texture maps, the renderer is responsible for rendering a depiction of the game-state; usually with images and sound.
- Controller – each character in the game has at least one controller ("brain") associated with it. The controller is responsible for selecting actions. For player characters the controller interprets joystick interactions. For NPCs the controller consists of the AI and low-level control.

Another View of a Controller



Controller Architectures



Controller Architectures

- Non-Deterministic Controllers:
  - Can easily accommodate very complex behaviors/ actions.
  - Can be computationally very complex.
  - Different reaction (outputs) to the same situations (inputs) – non-deterministic behavior.
- Deterministic Controllers:
  - Can only implement *reflexive behavior* (behavior that only depends on the current set of inputs - no memory/history/internal state).
  - Computationally usually very simple (lookup table).
  - Same reaction (outputs) to the same situations (inputs) – deterministic behavior

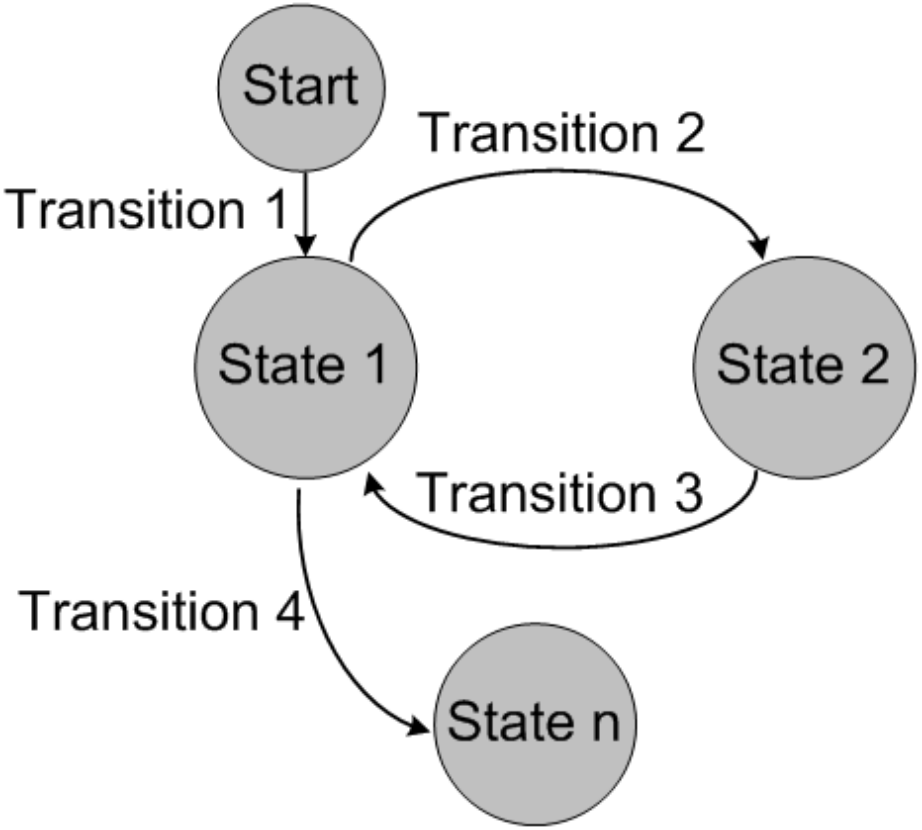
Finite State Machines as Controllers

- FSM is one of the simplest and most basic AI models.
- An FSM consists of
  - Inputs
  - States
  - State transitions

State transition table

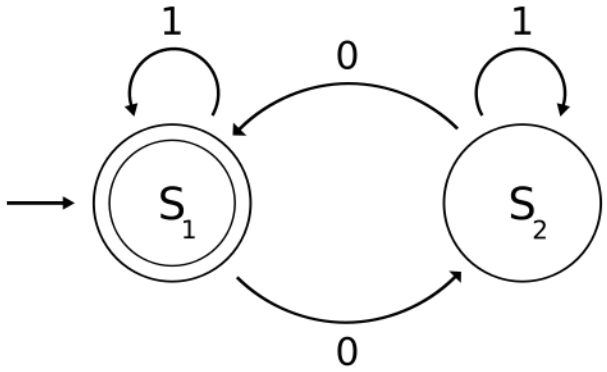
Current state → Input ↓	State A	State B	State C
Input X	...	...	...
Input Y	...	State C	...
Input Z	...	...	...

Finite State Machines



Finite State Machines

- From a CS perspective we are used to FSMs as recognizers of some language
- What language does the following FSM recognize?

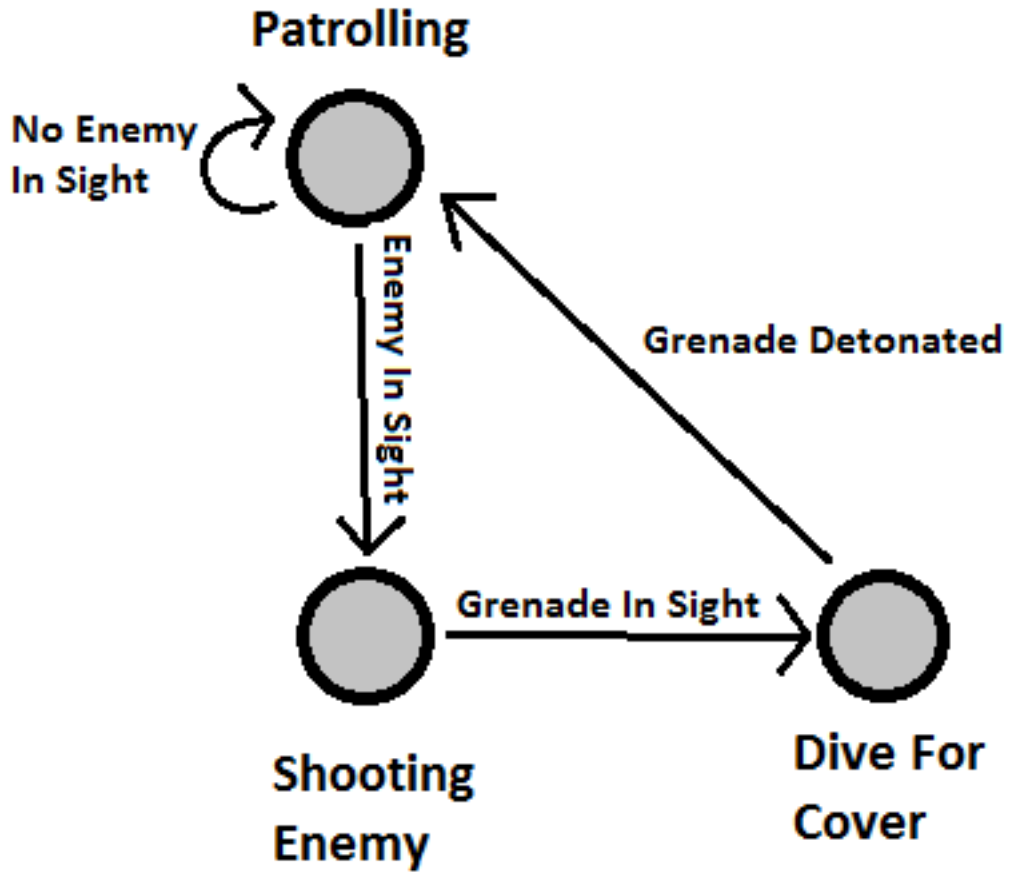


Finite State Machines

- We are interested in generating behavior given a state and an input

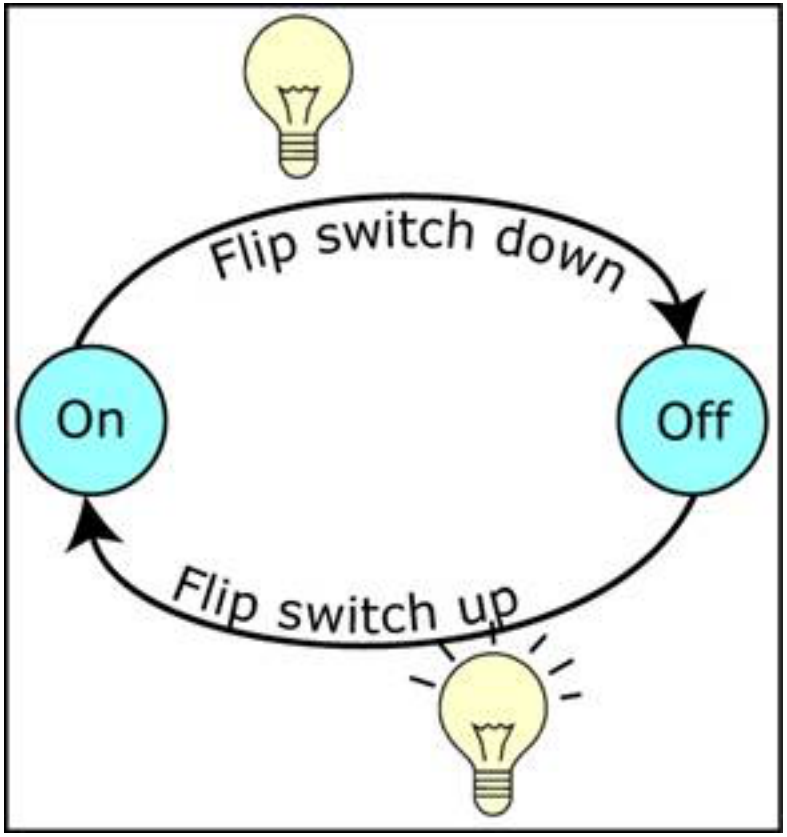
➔ Transducers

Transducers



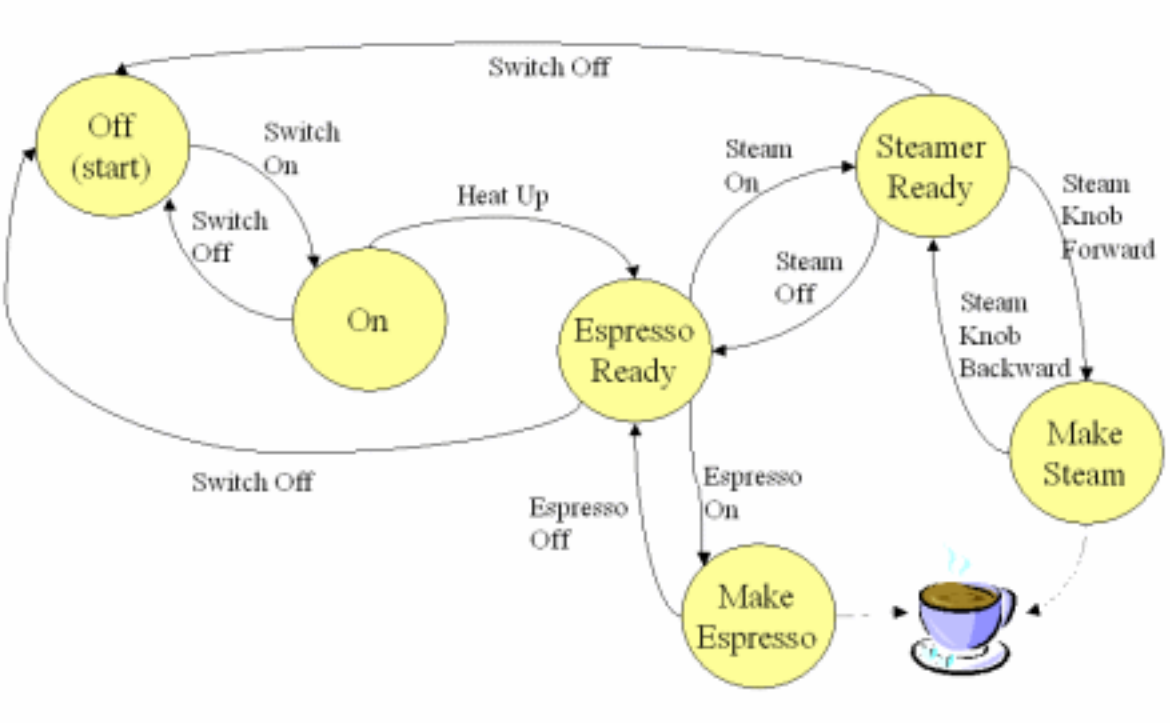
http://martindevans.me/heist-game/2013/04/16/Finite-State-Machines-(Are-Boring)/

Transducers



http://www.ai-junkie.com/architecture/state_driven/tut_state1.html

Transducers



https://msdn.microsoft.com/en-us/library/aa478972.aspx

FSM Basics

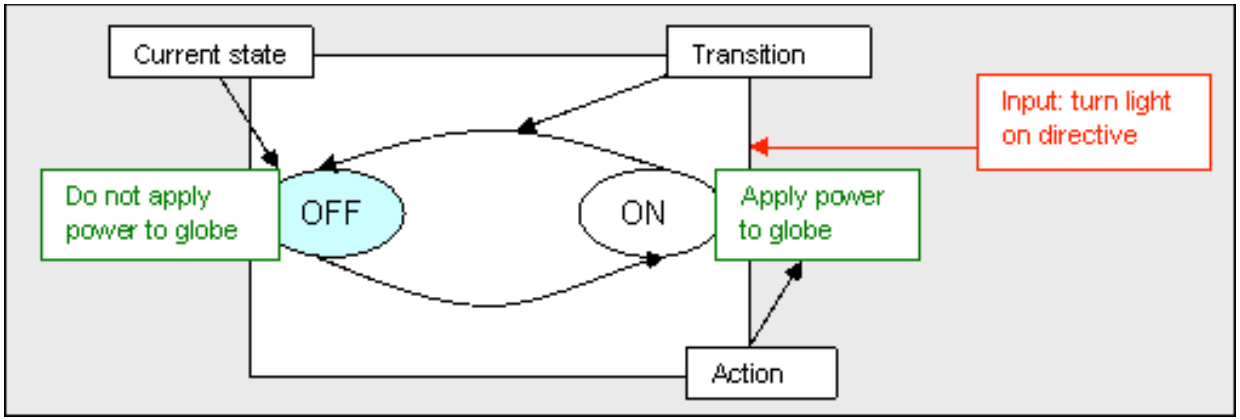
- A FSM consists of the following 4 components:
  - States which define behavior and may produce actions
    - exclusive acions on states -- Moore Machine
  - State transitions which are movement from one state to another and may produce actions
    - exclusive actions on transitions -- Mearly Machine
  - Rules/conditions/labels which must be met to allow a state transition
  - Input events which may trigger rules/satisfy conditions/match labels and lead to state transitions

See the tutorial online: http://ai-depot.com/FiniteStateMachines/FSM.html

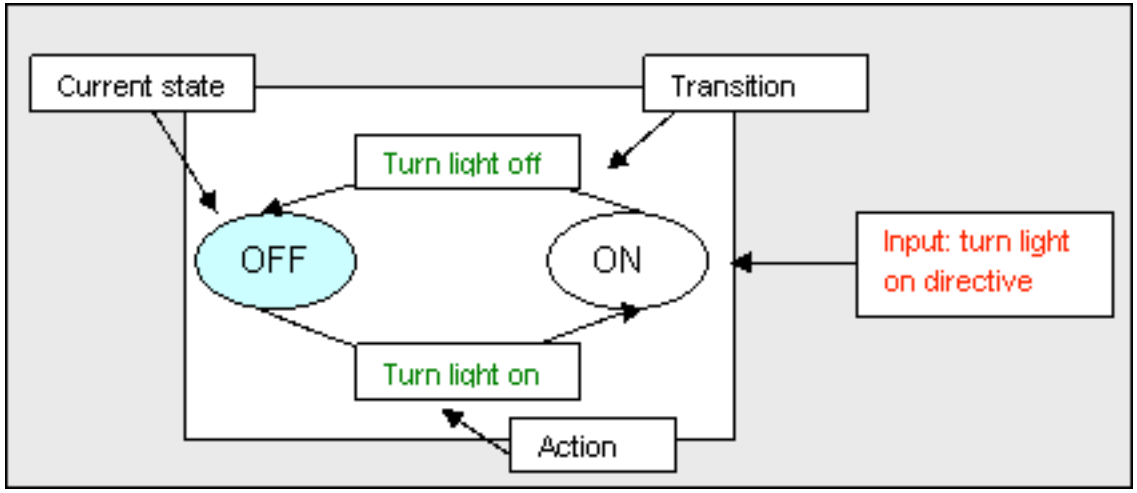
Moore vs. Mearly

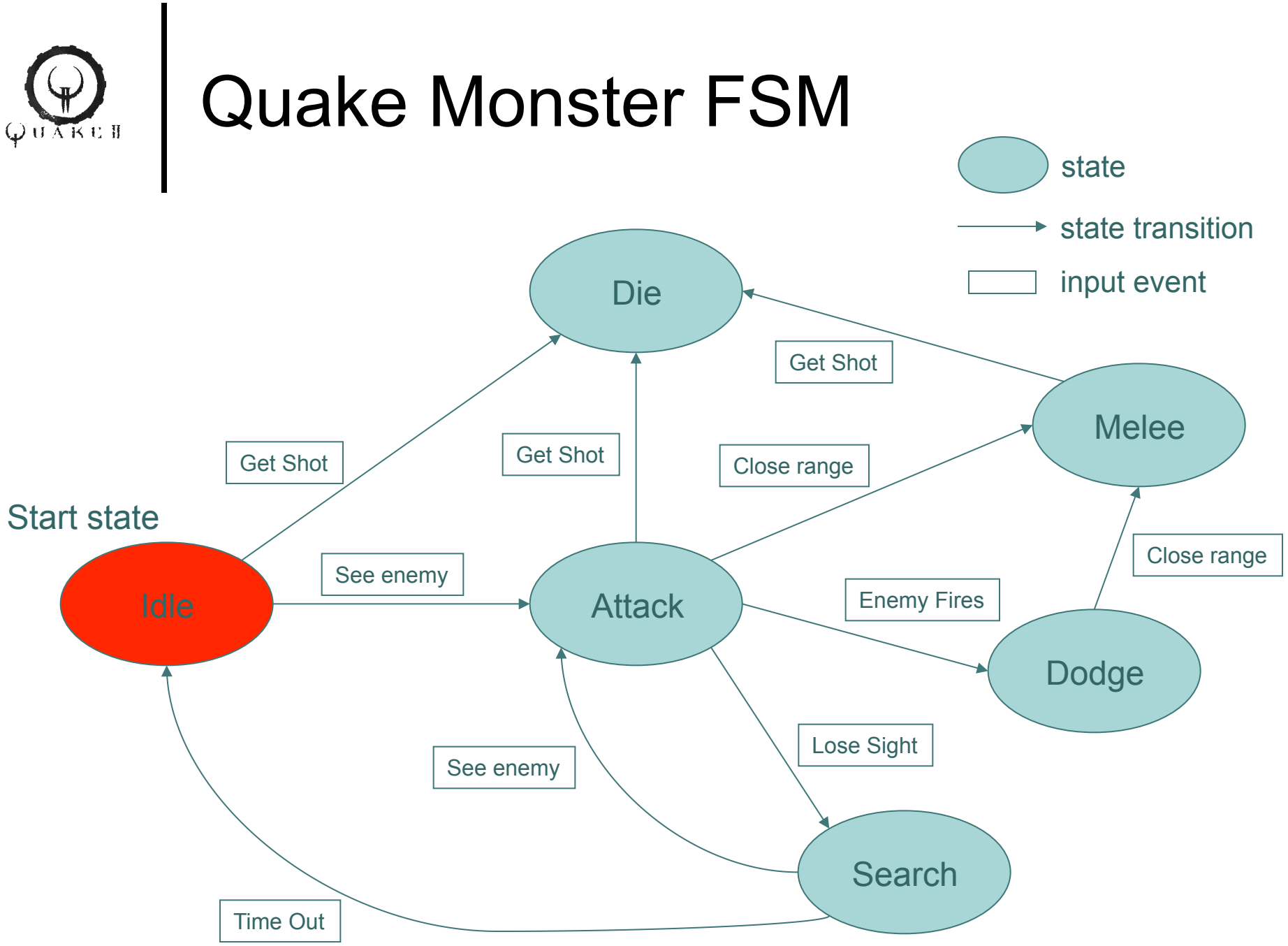
Problem: Switching a light on and off.

Moore Machine

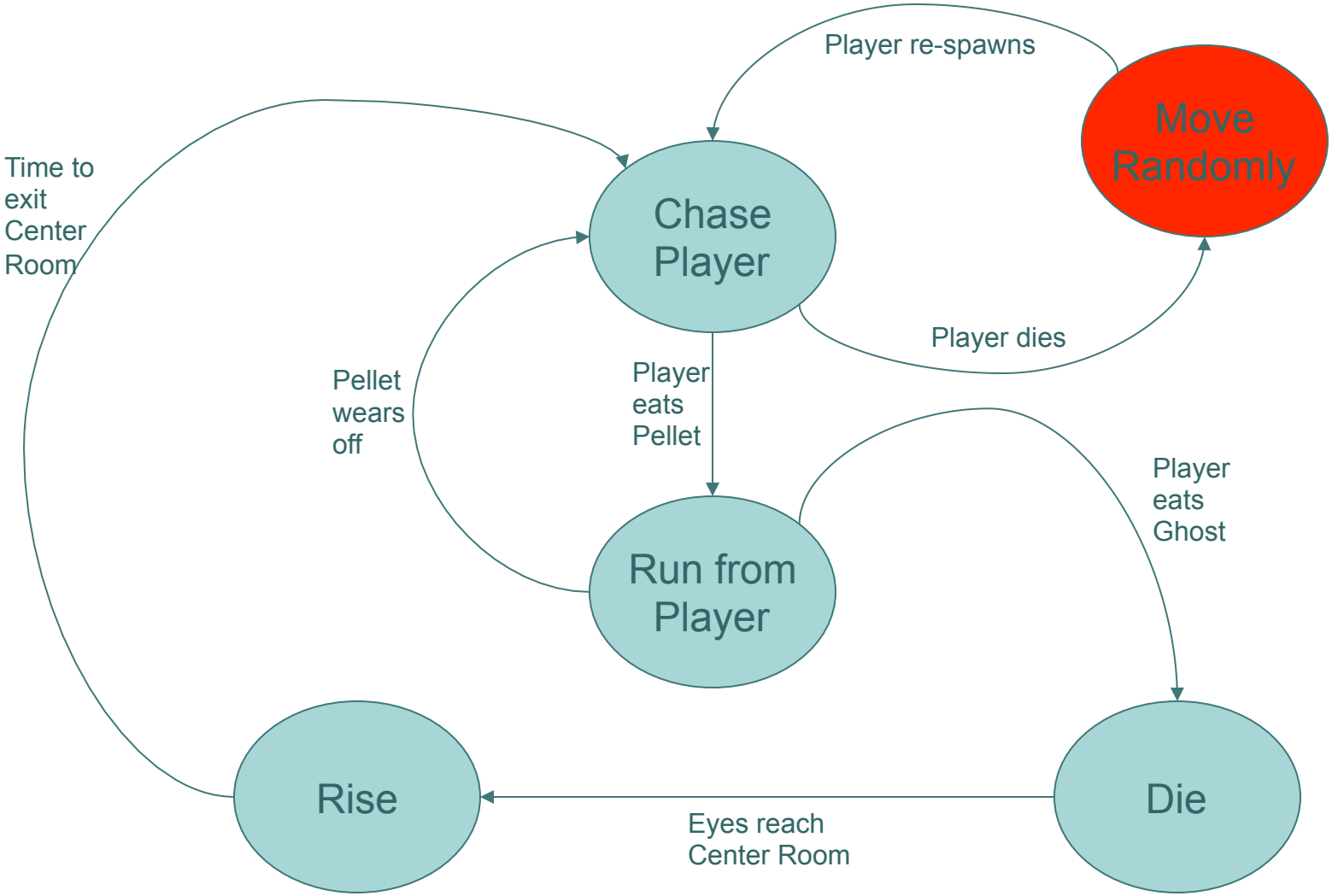


Mearly Machine




Quake Monster FSM
state
state transition
input event
Die
Melee
Attack
Idle
Dodge
Search
Start state
Get Shot
Get Shot
Get Shot
Close range
Close range
See enemy
Enemy Fires
See enemy
Lose Sight
Time Out

Another Example: FSM for Ghost in Pac-Man



Disadvantages of FSM

- May be too predictable
- Large FSM with many states and transitions can be difficult to manage and maintain. The graph may start to look like “spaghetti.”
- State oscillation. States may be too rigid. Conditions are too crisp.
  - For example, there are two states: Flee and Idle.
  - The condition for being in the Flee state is to be within a distance 5.0 from the enemy. The condition for being in the Idle state is to be greater than 5.0 from the enemy.
  - Say, the object is 4.9 from the enemy. It is in Flee state, so it runs away. Now it is 5.1, so it is in Idle state. It randomly moves around, and goes to 4.9 and gets into the Flee state again etc.

Implementation

```
class Monster {
  int state; // 0: Idle 1: Attack 2: Melee 3: Dodge 4: Search 5: Die
             // perhaps should use an enum type here….

  Monster() {state = 0;}

  void Iterate() {
     switch(state) {
     case 0: Move(rand()); break;
     case 1: Chase(); Shoot(); break;
     case 2: Melee(); break;
     case 3: Dodge(); break;
     case 4: FindEnemy(); break;
      case 5: Die(); break;
     }}

  void HandleInput(int eventID) {
      if (eventID== E_INJURED) {
          state = 5;
  }  else if (eventID== E_SPOTTED_ENEMY) {
         if ((state==0) || (state==4)) {
            state = 1;
         }
      }
  }
  void Shoot(){…}
  void Melee() {…}
  void Dodge() {…}
  void Chase() {…}
  void FindEnemy() {…}
  void Die() {…}
};
```

Example FSM

- What would a FSM look like that make a quagent walk back and forth in a quake room?

Assignments

- Read Chap 2
- Read FSM online tutorial (see course website)
- Try to implement Programming Exercise #1 with a FSM