Decision Making

Jeff Wilson

What all can agents do?

- Move…
- But also:
  - Gather resources
  - Attack
  - Access computer terminal
  - Open door
  - Etc.
- Steering behavior blending can sometimes address multiple behaviors, but often agents need higher levels of control supported by a decision making process
  - for instance, to avoid steering vectors that negatively interfere, cancel each other out, etc.

Decision Making: Pre-planned behaviors

- Programmer determines *recipe* for agent behaviors
  - In other words, the programmer implements the plan(s)
- All contingencies anticipated as part of this *recipe*
- If something was missed, the agent is unable to adapt
- Decisions are made by conditional logic
- Can include deterministic on non-deterministic conditions

Types of Decision Making

- **Reactive**
  - Response directly to environmental state
  - Pre-planned conditional actions
  - Most RT game AIs are reactive
- **Deliberative**
  - Models environment (discretized)
  - Inferences made to determine plan that can achieve goal state
- **Reflective**
  - Learn from experience

Pre-planned Behaviors: Types

- Production Rules (Simple version/Fixed Priority Arbiter)
- FSMs (variants)
- Behavior Trees

Production Rules (Simple Version/Fixed Priority Arbiter)

- Define actions to be executed in response to conditions
- Simple architecture
- Difficult to organize
- Can be difficult to understand behavior from looking at rules
- Rules can conflict (ambiguous situations)
- Action sequences require a series of rules with stateful triggers

Production Rules (Simple Version/Fixed Priority Arbiter)

If(big enemy in sight) run away

If(enemy in sight) fire

If(---) ----

Age of Kings example

```
(defrule
       (building-type-count-total castle less-than 1)
       (can-build castle)
=>
       (build castle)
       (chat-local-to-self "castle"))
```

Production Rules (Simple version/Fixed Priority Arbiter)

- How to select among multiple valid rules?
  - Advanced production rules systems utilize an **arbiter** that selects from matching rules (will be revisited)
  - **Fixed Priority Arbiter**: For simple version, we will just evaluate the first one that matches from an ordered list
  - Priority weighting for rules or sensor events
    - Like firewall rules
  - Random selection
- Can be stateless

Example of random selection with Fixed Priority

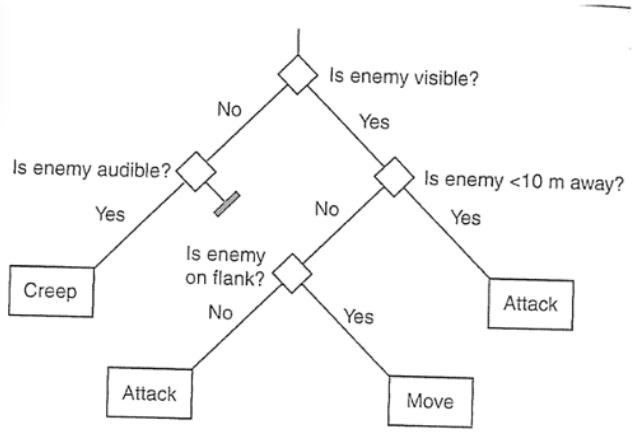
If((big enemy in sight) AND (rand() > thresh1)) run away

If((enemy in sight) AND (rand() > thresh2)) fire

If(---) ----

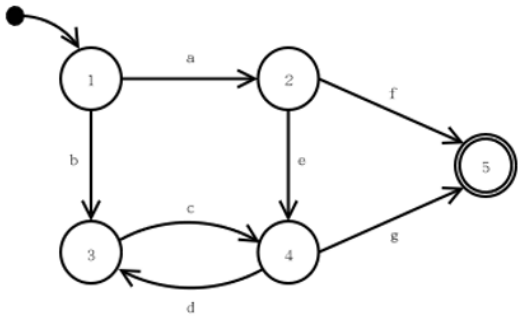
Simple Decision Tree (Manually Created)

- Nested if statements organized logically (for now)
- Good for simple decision making
- But quickly becomes a mess to manage manually
- (To be revisited later)



Finite State Machine (FSM) Theory

- A (model of a) device which has
  - a finite number of states (*S*)
  - an input vocabulary (*I*)
  - a transition function T(*s*,*i*) → *s'*
  - a start state $s_0 \in S$
  - zero or more final states $F \subset S$
- Behavior
  - Can only be in one state at a given moment in time
  - Can make transitions from one state to another or to cause an output (action) to take place.



∈ - element of

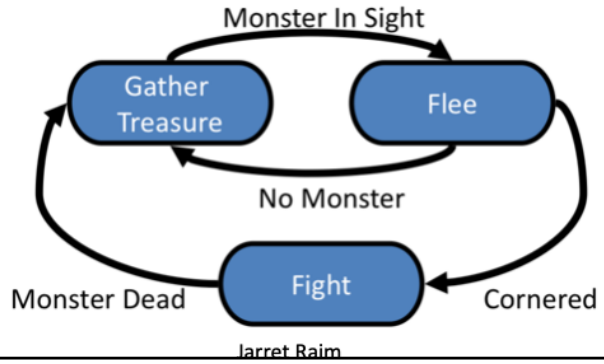
⊂ - proper subset / strict subset

Finite State Machine (FSM)

- Each state represents some desired behavior
- Can poll the world, or respond to events (more on this later)
- Support actions that depend on state & triggering event (Mealy) as well as entry & exit actions associated with states (Moore)

FSM

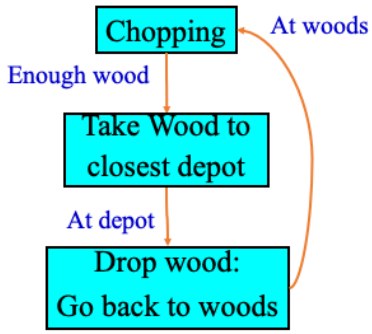
- Character AI modeled as sequence of mental states
- World events (can) force a change in state
- Mental model easy to grasp (for all)



Jarret Raim

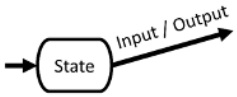
Finite State Machines (FSMs)

- States: Action to take
  - Chase
  - Random Walk
  - Patrol
  - Eat
- Transitions:
  - Time
  - Events
  - Completion of action



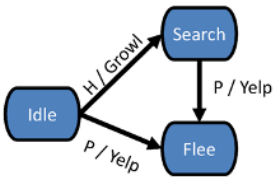
Mealy & Moore

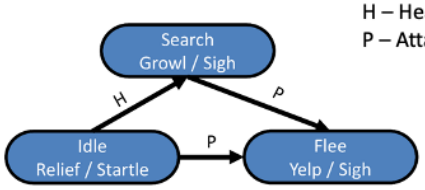
Mealy Output =
F(state, input)



Moore Output =
F(state)

Inputs
H – Hear
P – Attacked/See Big Enemy



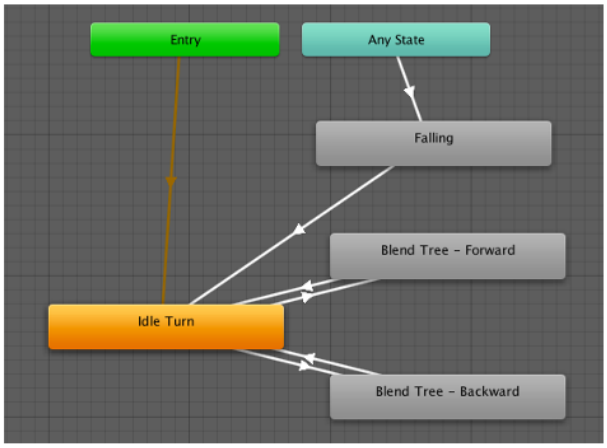


Mealy output values determined by current state and current inputs
Moore output determined just by current state

H – Hear
P – Attacked or see big enemy

FSM

- ***Any State*** *or* ***Global*** – Convenience notation
- Allows for simplified implementation and visualization

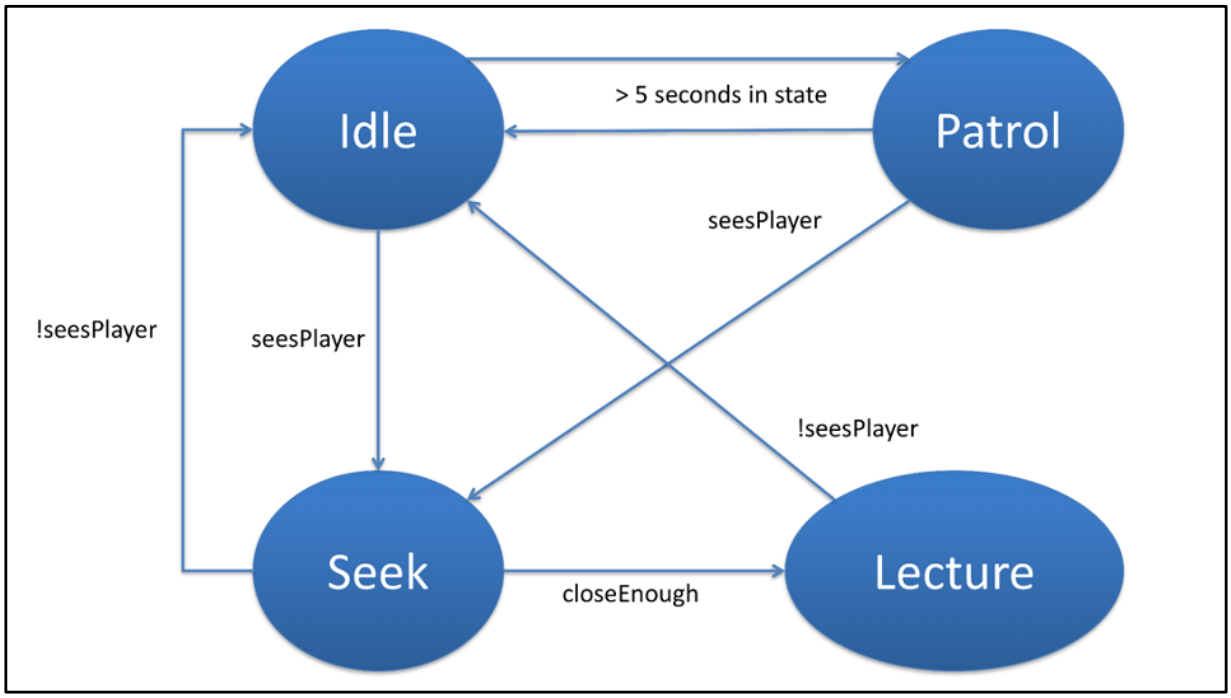


FSM Exercise

Using the graph form, design/visualize an FSM for a security guard that patrols a school after hours. If it sees the player it needs to seek the player and give them a lecture till the player can slip away. Otherwise switch between Idle and Patrol.

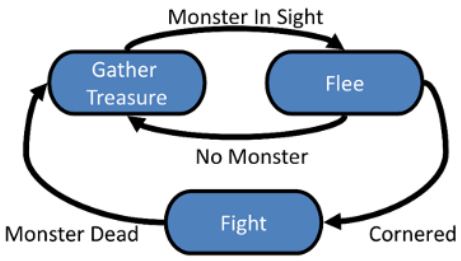
States = {Idle, Patrol, Seek, Lecture}

Feel free to make up variables.


Idle
Patrol
Seek
Lecture
> 5 seconds in state
seesPlayer
!seesPlayer
seesPlayer
!seesPlayer
closeEnough

State Transition Table

Current State	Condition	State Transition
Gather Treasure	Monster	Flee
Flee	Cornered	Fight
Flee	No Monster	Gather Treasure
Fight	Monster Dead	Gather Treasure

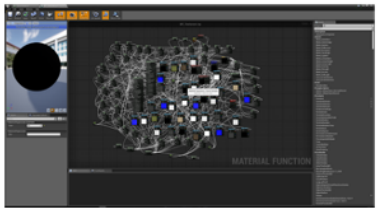


FSM representation with table

State Machine Problems

- Predictable
  - Sometimes a good thing
  - If not, use fuzzy or probabilistic state machines to help some
- Simplistic
  - Use hierarchies of FSM's (Half-Life)
    - Use of FSM stack
    - Simplifies development/debugging

FSM: Disadvantages



- When it fails, fails hard:
  - A transition from one state to another requires forethought (get stuck in a state or can't do the "correct" next action)
- Number of states can grow fast
  - Exponentially with number of events in world (multiple ways to react to same event given other variables)
- Number of transitions can grow even faster
- Doesn't work easily with sequences of actions/memory

FSM: Advantages

- Ubiquitous (not only in digital games)
- Quick and simple to code
- (can be) easy to debug
- Fast: Small computational overhead
- Intuitive
- Flexible
- Easy for designers without coding knowledge

FSM: Implementation (procedural)

```
public enum AIState
{
  Patrol,
  GoToAmmoDepot,
  AttackPlayerWithProjectile,
  InterceptPlayer,
  AttackPlayerWithMelee,
  ChasePlayer
  //TODO more? states…
};

public AIState aiState;

// Use this for initialization
void Start ()
{
  aiState = AIState.Patrol;
}
```

```
void Update ()
{
  //state transitions that can happen from any state might happen here
  //such as:
  //if(inView(enemy) && (ammoCount == 0) &&
  //  closeEnoughForMeleeAttack(enemy))
  //  aiState = AIState.AttackPlayerWithMelee;

  //Assess the current state, possibly deciding to change to a different state
  switch (aiState) {
    case AIState.Patrol:
    //if(ammoCount == 0)
    //    aiState = AIState.GoToAmmoDepot;
    //else
    //  SteerTo(nextWaypoint);
    break;

    case AIState.GoToAmmoDepot:
    //SteerToClosestAmmoDepot()
    break;
    //... TODO handle other states
    default:
    break;
  }
}
```

FSM: Implementation (Object Oriented)

File: FSMState.cs

```
abstract public class FSMState  <T>  {
  abstract public void Enter (T entity);
  abstract public void Execute (T entity);
  abstract public void Exit(T entity);
}
```

File: EnterMineAndDigForNugget.cs

```
using UnityEngine;

public sealed class EnterMineAndDigForNuggets :  FSMState<Miner> {

  static readonly EnterMineAndDigForNuggets instance =
    new EnterMineAndDigForNuggets();
  public static EnterMineAndDigForNuggets Instance {
    get {    return instance;  }
  }
  static EnterMineAndDigForNuggets() { }
  private EnterMineAndDigForNuggets() { }

  public override void Enter (Miner m) {
    if (m.Location != Locations.goldmine) {
      Debug.Log("Entering the mine...");
      m.ChangeLocation(Locations.goldmine);
    }
  }
  public override void Execute (Miner m) {
    m.AddToGoldCarried(1);
    Debug.Log("Picking ap nugget and that's..." +
      m.GoldCarried);
    m.IncreaseFatigue();
    if (m.PocketsFull())
      m.ChangeState(VisitBankAndDepositGold.Instance);
  }
  public override void Exit(Miner m) {
    Debug.Log("Leaving the mine with my pockets full...");
  }
}
```

https://blog.playmedusa.com/a-finite-state-machine-in-c-for-unity3d/

FSM: Parameterized Transitions

- Often states need information on which to act
- Simple solution may just reference various member variables of agent
- But as FSM becomes more complex, it can become challenging to make sure member variables are correctly set (e.g. activeTarget, currentPath, etc.)
- Parameterizing "ChangeState()" methods can make state requirements explicit, enforce that info be provided, and provide precondition enforcement opportunity
- With OO approach, generics/templates language concepts useful:

```
public abstract class State1Param  < T >          : StateBase { public abstract void Enter(T p); }
public abstract class State2Params < T0, T1 >     : StateBase { public abstract void Enter(T0 p0, T1 p1); }
public abstract class State3Params < T0, T1, T2 >: StateBase { public abstract void Enter(T0 p0, T1 p1, T2 p2); }
```

See http://zalods.blogspot.com/2018/11/simple-yet-powerful-fsm-implementation.html

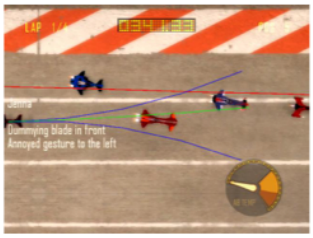
For more detais

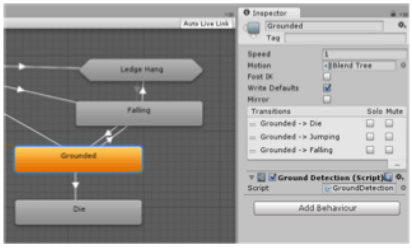
FSM: Changing State Safely

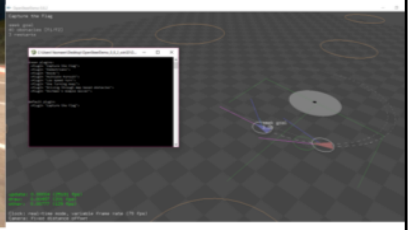
- State changes can normally happen anywhere, including within the "UpdateState()" of a particular FSM state
- This can create troublesome situations where current state changes state but executes further code
- It's likely that a programmer will eventually shoot themselves in the foot with unexpected side effects from calling "EnterState()" and then executing further code in the exited state
- Therefore, enforcing a mechanism for enforcing **deferred state changes** may be desirable.
- For instance, only allow a state change request as part of an "UpdateState()" return value that specifies either: "*stay in the current state*" or "*change to this new state (and here are some parameters for the call)*"

Debugging FSMs

- Offline Debugging
  - Logging
  - Verbosity Levels
- Online Debugging
  - Graphical representation is modified based on AI state
  - Command line to modify AI behavior on the fly.

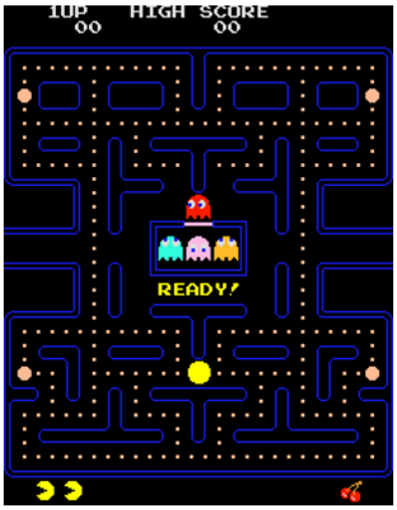






FSM Examples: Pac Man

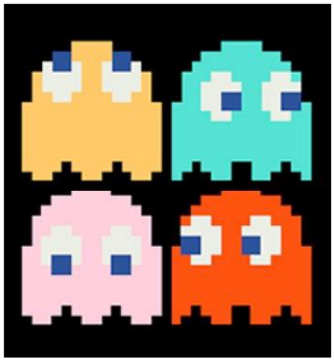
- States ?
- Transitions ?



https://www.gamasutra.com/view/feature/3938/the_pacman_dossier.php?print=1

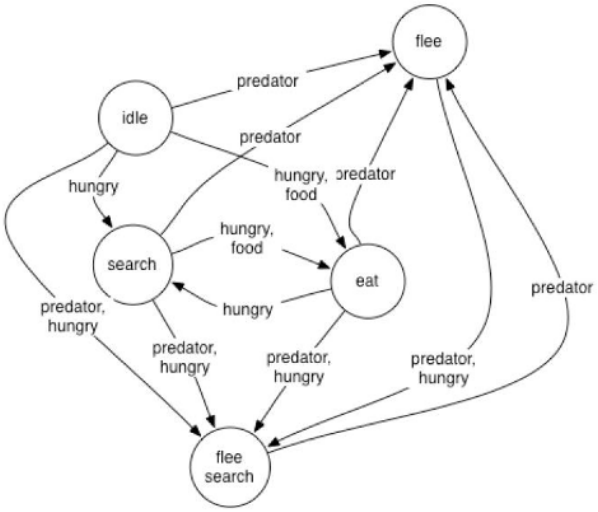
FSM Examples: Pac Man

- Red: Shadow, blinky
  - "pursuer" or "chaser"
  - Chase target: pac-man's tile
- Pink: Speedy, pinky
  - "ambusher"
  - Chase target: pac-man's tile + 4 ahead
- Blue: Bashful, inky
  - "whimsical"
  - Chase target: double vector from red ghost to (pac + 2 ahead)
- Orange: Pokey, Clyde
  - "feigning ignorance"
  - Chase target: dist < 8 ? scatter tile : pac-man's tile



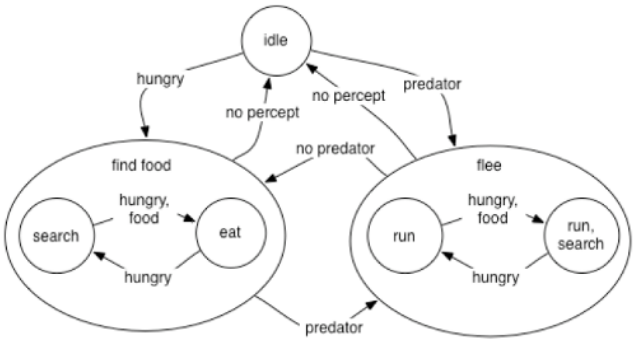
Chase, Scatter, or Frightened
http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior

FSM Examples: Prey scenario



Hierarchical FSM Example

- Equivalent to regular FSMs, adding recursive multi-level evaluation
- Easier to think about encapsulation
- Hierarchical approach addresses *entry, update, exit* and *any* (wildcard) at multiple levels
- But how to deal with transition from lower level state?



Hierarchical FSM

- Changes in world state event (input vocabulary) can be used for transitions at any level in hierarchy
- If a low-level state does not handle a potential transition event, the unhandled event is dealt with at a higher level
- Allows designer to avoid duplicating transitions
- …but you need a way for low-level state to transition out once high level transition has been identified…

http://aigamedev.com/open/article/hfsm-gist/

Hierarchical FSM

- Recursive algorithm
  - Highest level transitions are always honored, bypassing lower level updates
  - Hierarchical states remember what child state they are in
  - All actions, whether associated with entry, update, or exit are deferred. These are collated in order of recursive evaluation and only executed once the entire HFSM is evaluated
  - Furthermore, transitions that change levels in the hierarchy are deferred when transitioning up (recursively chained when going down)

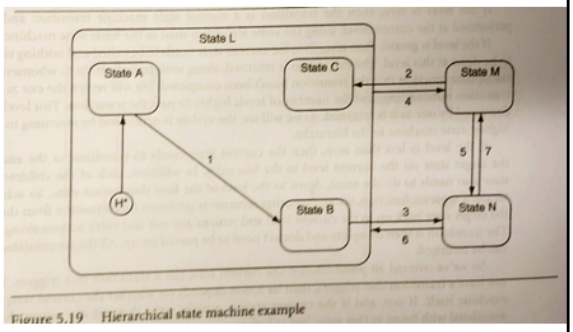
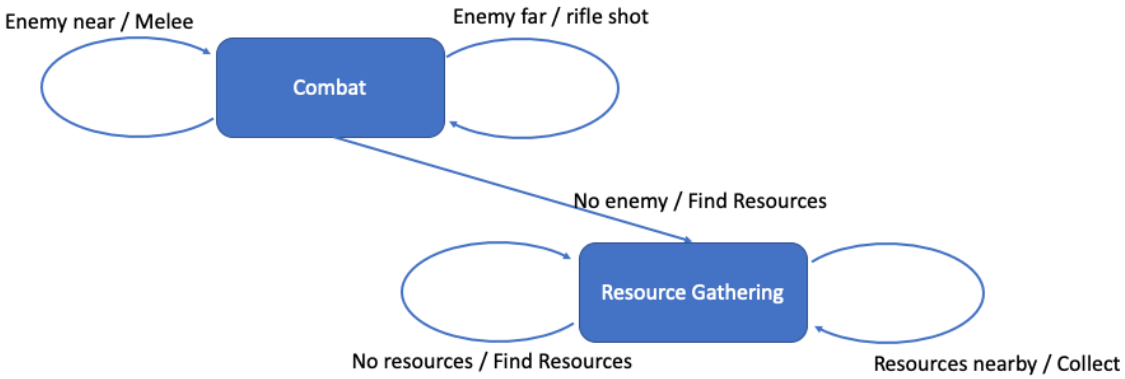


Figure 5.19 Hierarchical state machine example

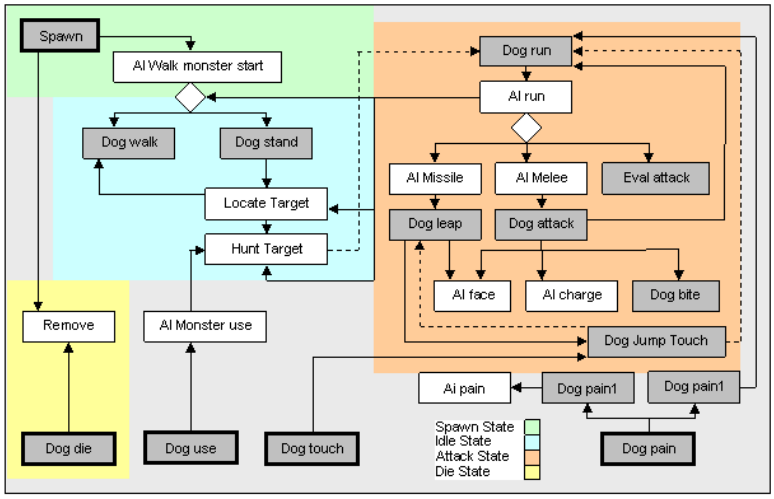
Artificial Intelligence for Games (2nd ed) Millington and Funge: page 323 Examples

Hierarchical FSMs

- Mealy can be used in a somewhat hierarchical way (but only one level deep)



FSM: Quake Rottweiler

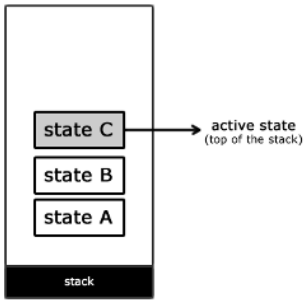


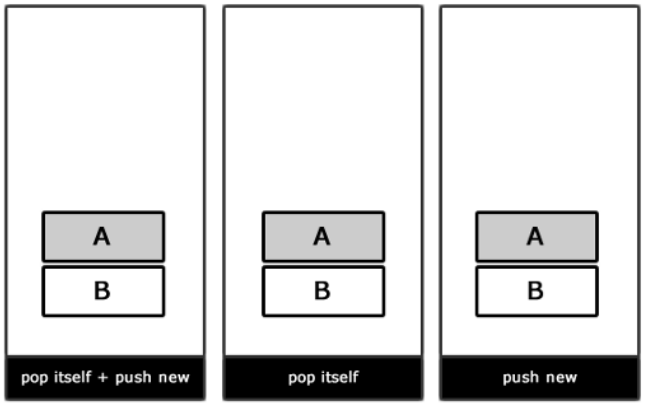
http://ai-depot.com/FiniteStateMachines/FSM-Framework.html

Some hierarchical patterns demonstrated here

FSM: Stack

- Must make sure there is always a state on the stack (maybe root state that never pops but pushes new states every time it is top?)
- Pushing/Popping can introduce challenges in how to transition game/character state from one state to the next





https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867

FSM: Stack Problem

- Memory leak potential
- Pop + Push New – Helps, provided that it is used by programmer
- May need restrictions enforced such as:
  - Cannot have two of the same state on the stack at the same time (or from same hierarchy family)
  - Pop + Push New required for state changes among sibling states within a hierarchical level (cannot push a sibling state without first popping self)

FSM: Transition Problem

FSM: Transition Problem

- HFSMs and FSM Stacks provide ways to generalize transitions
- However, this is easier said than done
- Real games involve complex character states, often animation based, that do not have obvious transition opportunities to new state behaviors
- A high level HFSM transition may require unique sub-state cleanup to transition out (the developer must plan for all possibilities)
- Typical to transition out of low level states to a neutral state (e.g. character standing in place), but can be awkward
- Animation blending/interpolation, physics-based ragdoll simulation can also help
- Agent interacting with artifacts can also be a challenge (don't forget to get rid of that live grenade!)

FSM: Transition Problem

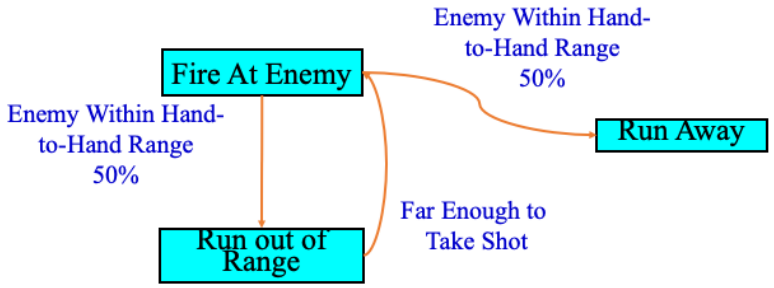
- Practical FSM implementations may need additional support for multi-frame transitions, deferring, blocking transitions (or additional states and transitions)
- Example: An agent attack animation must finish first with a deferred transition
- Example: Agent must sheath sword before beginning conversation with friendly NPC

Probabilistic State Machines

- Personalities
  - Change probability that character will perform a given action under certain conditions

	Aggressive	Passive
Attack	50%	5%
Evade	5%	60%
Random	10%	10%
Flock	20%	20%
Pattern	15%	5%

Probabilistic Example

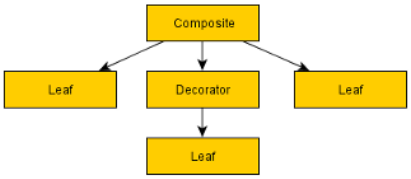


Probabilistic State Machines

- Other aspects:
  - Sight
  - Memory
  - Curiosity
  - Fear
  - Anger
  - Sadness
  - Sociability
- Modify probabilities on the fly?

Behavior Trees

- Some aspects of state machines
- More declarative
- Enforces understandable structure
- Components
  - Composite
  - Decorator
  - Leaf

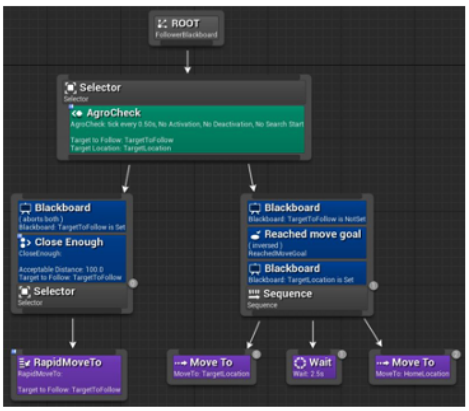


States:
- Success
- Failure
- Running

http://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php

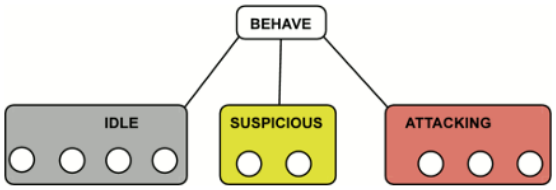
Behavior Trees

- Simple **reactive** planning
- Tree of behaviors specify what an agent should do under all circumstances (manually provided)



Behavior Trees

- Transitions are externalized from states
- Instead what you normally think of states in a FSM become behaviors
- This allows easy reuse of behaviors as context-specific transitions are decoupled



http://aigamedev.com/open/article/bt-overview/

Behavior Trees: Composite

- Composite Node:
  - One or more children
  - Sequence (AND), Selector (OR), or Random
  - Short circuiting of Boolean logic
  - Returns success or fail (based on children returns typically)

Behavior Trees: Decorator

- One child
- Change return of child
- Terminate the child
- Control logic (e.g. repeat on child)
- Examples:
  - Inverter (like ! / NOT)
  - Succeeder (always true, runs child but doesn't care about success/failure)
  - Repeater
  - Repeat until fail

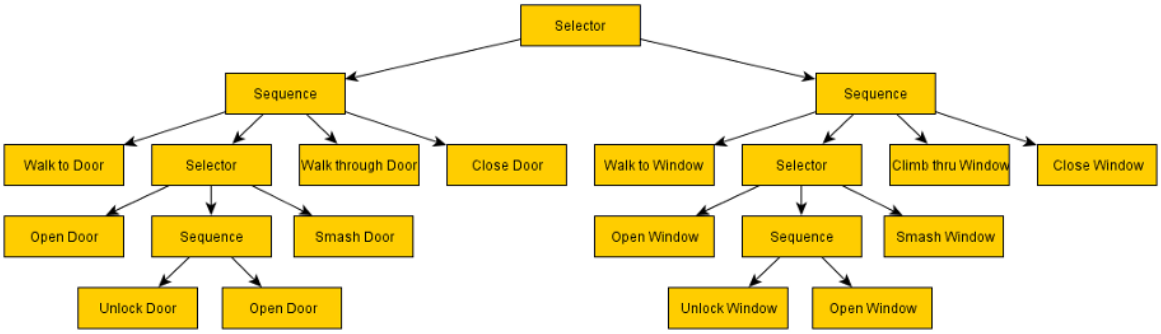
Behavior Trees: Leaf

- Game logic
  - Library
  - custom
- Returns Success, Fail, or Processing
- Init() – called first visit
- Process() –called until complete
- Parameters

Behavior Trees: One action per tick

- Each node keeps track of current child to execute
- At start of tick, walk the tree to find our current node. – If in it last frame continue, otherwise reset it
- Nodes return Success, Fail, or Processing

Behavior Tree: Example

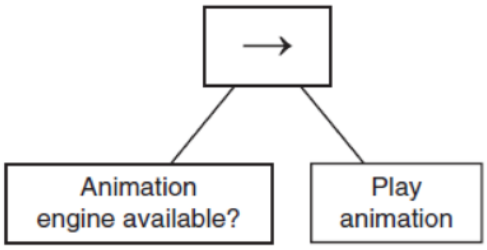


Behavior Trees: Non-deterministic

- Strict order == predictable
- Introduce random selector or sequence nodes
- Can also introduce prioritized versions which select or sequence according to world state (can be less predictable, more efficient, etc)

Behavior Trees: Semaphores

- Check for restricted resources
  - Keeps a tally of available resources and number of users – e.g. animation engine, pathfinding pool, etc.
- Typically provided in a language library



Behavior Trees

- Very popular/ubiquitous (Bungie's Halo 2 – 2004)
- Simple reactive planning that is synthesis of: HFSM, Scheduling, Planning
- Easy to design, alter, and reuse (parts of)
- Easy to understand
- Easy for non-programmers to create (usually with visual tools or high-level script)
- Instead of *state*, employ *behaviors* or *tasks*
- Composable, self contained
- Fail gracefully

Decision Making: Time Dependency

- New decisions can be made as often as the simulation updates
- What happens if update rate changes? (E.g. variable frame rate?)
- Especially noticeable on probabilistic decisions
- Consider: steering behavior updates every frame…
- …but decision-making on a fixed interval
- And/or time dependent probabilities