DM842

Computer Game Programming: AI

Lecture 6

Decision Making

Christian Kudahl

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Other Ideas in Pathfinding

2. Decision Making
   - Decision Trees
   - State Machine
   - Behavior Trees

Outline

1. Other Ideas in Pathfinding

2. Decision Making
    - Decision Trees
    - State Machine
    - Behavior Trees

Open Goal Pathfinding

- Multiple goals could cause problems. How should heuristic behave?
- Probably ideal that goal is decided in decision making and heuristic act according to single goal.

Dynamic Pathfinding

- Environment is changing in unpredictable ways or its information is incomplete.
- Replan from scratch each time new information is collected or
- Replan only the part that has changed $\rightsquigarrow$ D*. Requires a lot of storage space.

Memory-Bounded Search

- Try to reduce memory needs
  - Iterative-deepening A^* (IDA*)
  - Simplified Memory Bounded A^* (SMA*)

Iterative Deepening A*

- IDA*
  - Idea from classical Uniformed Iterative Deepening
    depth-first search where the max depth is iteratively increased
  - skip open and closed list
  - depth-first search with cutoff on the f-cost
  - cutoff set on the smallest f-cost of nodes that exceeded the threshold at the previous iteration
  - very simple to implement but less efficient
  - good variant for goal-oriented action planning in decision making

Properties of IDA*

Complete Yes
Time complexity Exponential
Space complexity linear
Optimal Yes. Also optimal with non-consistent heuristic (but admissability is needed)

Simple Memory-Bounded A*

Use all available memory

- Follow A* algorithm and fill memory with new expanded nodes
- If new node does not fit
  - remove stored node with worst f-value
  - propagate f-value of removed node to parent
- SMA* will regenerate a subtree only when it is needed
  the path through subtree is unknown, but cost is known

Properties of SMA*

Complete Yes, if there is enough memory for a solution path
Time Same as A* if enough memory to store the shortest path tree
Space Use available memory
Optimal Yes, if enough memory to store the best solution path

In practice, good trade-off between time and space requirements

Other Issues

Interruptible Pathfinding

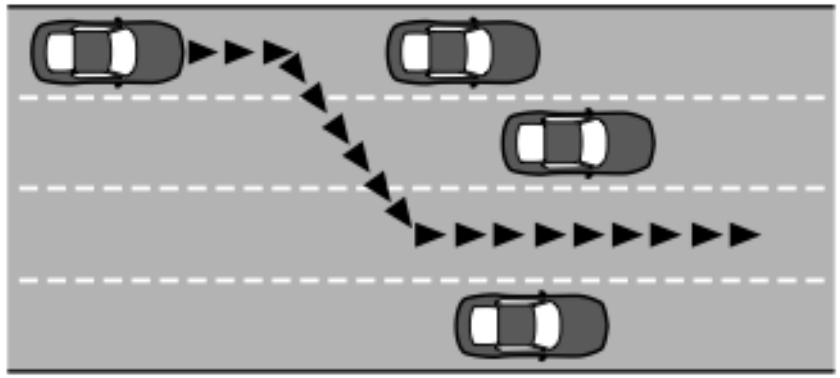
- rendering needs to run every 1/60 or 1/30 of a second ($= 0.6ms$)
- A* algorithm can be easily stopped and resumed.
- data required to resume are all contained in the open and closed lists.

In Real Time Strategy games: possible many requests to pathfinding at the same time

- serial $\rightsquigarrow$ problems for time, parallel $\rightsquigarrow$ problems for space
- central pool of pathfinding + path finding queue (FIFO).
- information from previous pathfinding runs could be useful to be stored (especially valid for hierarchical pathfinding)

Continuous Pathfinding

Vehicle pathfinding: eg, police car pursuing a criminal
Split down by placing a node every few yards along the road
path = a period of time in a sequence of adjacent lanes.



But cars are moving. Depending on the speed the gap may be there or not.

- A* in a graph where nodes represent states rather than positions
- a node has two elements: a position and a time.
- an edge exists between two nodes if the end node can be reached from the start node and if the time it takes to reach the node is correct.
- two different nodes may represent the same position

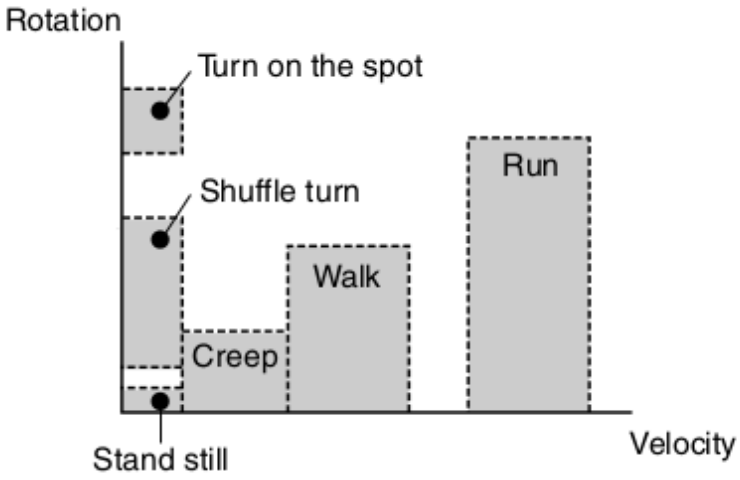
- graph created dynamically: connections, so they are built from scratch when the outgoing connections are requested from the graph.

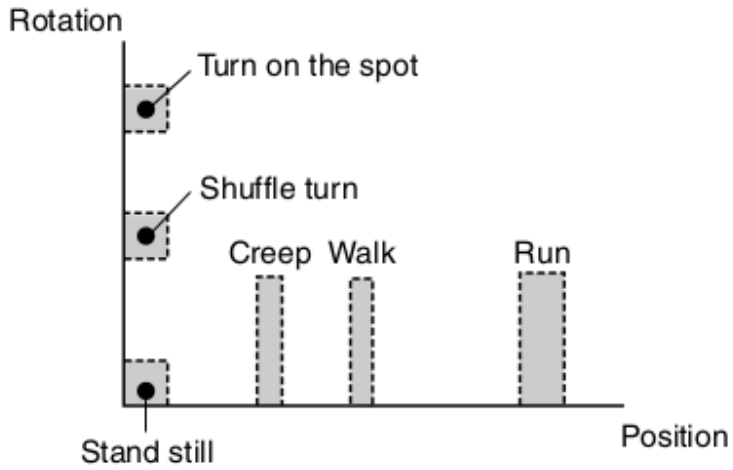
- retrieving the out-going connections from a node is a very time-consuming process $\rightsquigarrow$ avoid A* versions that need recalculations

- It should be used for only small sections of planning.
  Eg, plan a route for only the next 100 yards or so. The remainder of the route planned on intersection-by-intersection basis.
  Hierarchical Pathfinding, with the continuous planner being the lowest level of the hierarchy.

Movement Planning

- If characters are highly constrained, then the steering behaviors might not produce sensible results. Eg: urban driving.
- Chars have, eg, walk animation, run animation, or sprint animation
- Animations need specific conditions for being believable
- Plan sequence of animations to reach a large scale maneuver



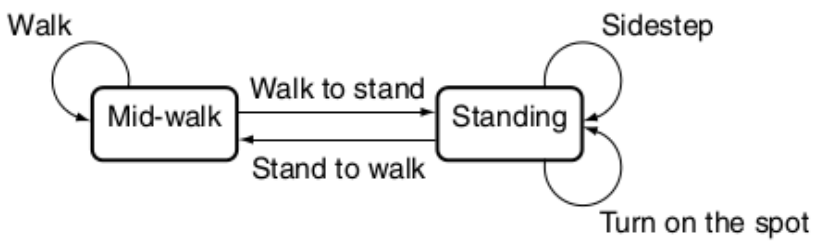


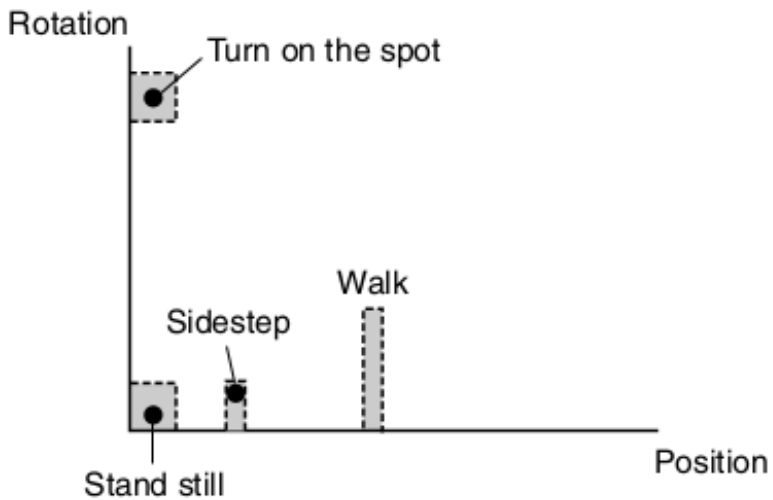
- Movement planning uses a graph representation. Each node of the graph represents both the position and the state of the character at that point, ie, the velocity vector, that determines the set of allowable animations that can follow
- Connections in the graph represent valid animations; lead to nodes representing the char after the animation
- Route returned consists of a set of animations
- If the velocities and positions are continuous, then infinite number of possible connections. Heuristic only returns the best successor nodes for addition to the open list.
- Similarly to continuous pathfinding, graph is generated on the fly.

Example

Walking bipedal character
Animations: walk, stand to walk, walk to stand, sidestep, and turn on the spot.
They can be applied to a range of movement distances
Positions: Each animation starts or ends from one of two positions: mid-walk or standing still.
Some positions in the environment are forbidden

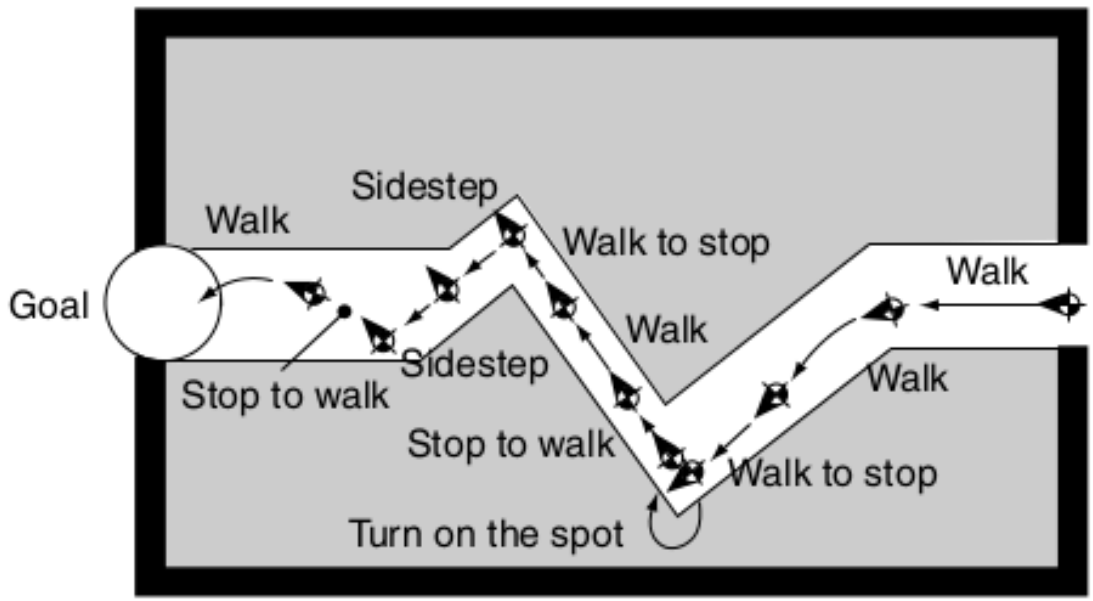
State machine: positions $\equiv$ states and transitions $\equiv$ animations.





Goal: range of positions with no orientation.

Result from A*:



Outline

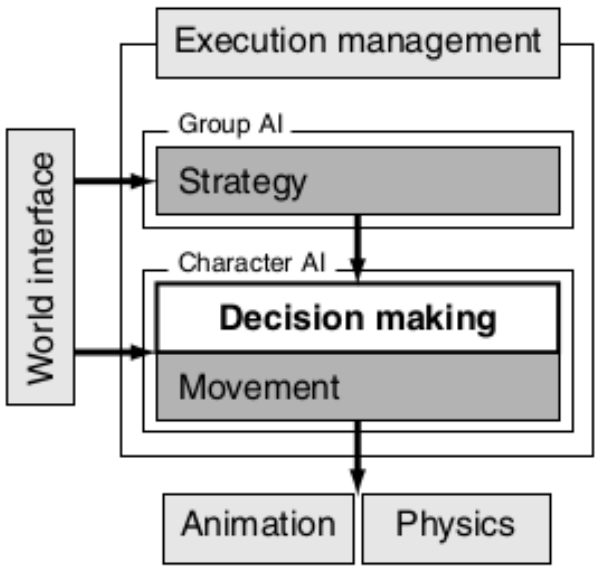
1. Other Ideas in Pathfinding

2. Decision Making
    - Decision Trees
    - State Machine
    - Behavior Trees

Decision Making

Decision Making: ability of a character to decide what to do.
We saw already how to carry out that decision (movement, animation, ...).
From animation control to complex strategic and tactical AI.

- state machines,
- decision trees
- rule-based systems
- fuzzy logic
- neural networks



Input internal and external knowledge
Output action

Knowledge representation:

- External knowledge identical for all algorithms
  Message passing system.
  Eg, `danger` is a constant at the character. Every new object needs to define when to send message `danger` and the character will react.

- Internal knowledge algorithm dependent

- Actions:
  Objects notify which actions they are capable of by means of flags.
  For goal oriented behavior, every action has a list of goals that will be achieved
  Alternatively, actions as objects with associated data such as state of world after action, animations, etc. Actions are then associated to objects.

The Toolchain

- AI-related elements of a complete toolchain
- Custom-designed level editing tools to be reused in multiple games
- Each object in the game world has a set of data associated with it that controls behavior
  Eg, data type “to be avoided” / “to be collected”.
- Different characters require different decision making logic and behavior
- Allowing level designers to have access to the AI of characters they are placing without a programmer requires specialist AI design tools.

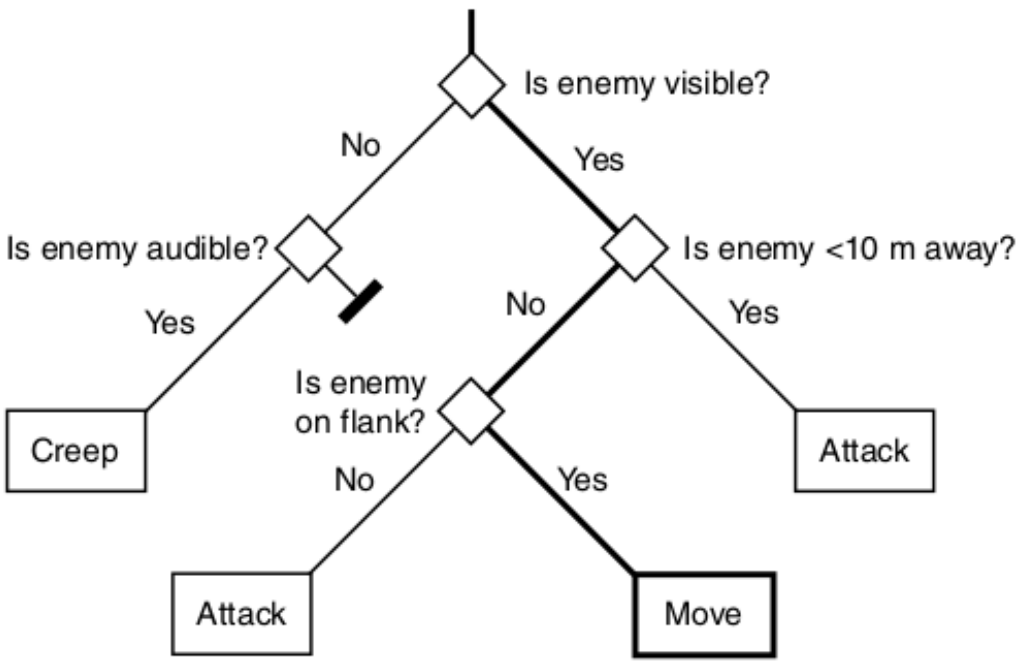
Outline

1. Other Ideas in Pathfinding

2. Decision Making
   - Decision Trees
   - State Machine
   - Behavior Trees

Decision Trees

- Tree made up of connected decision points.
- Each choice is made based on the character's knowledge.
- At each leaf of the tree an action is attached
- Typically binary tree (multibranches are equivalent) but more generally directed acyclic graph (DAG).



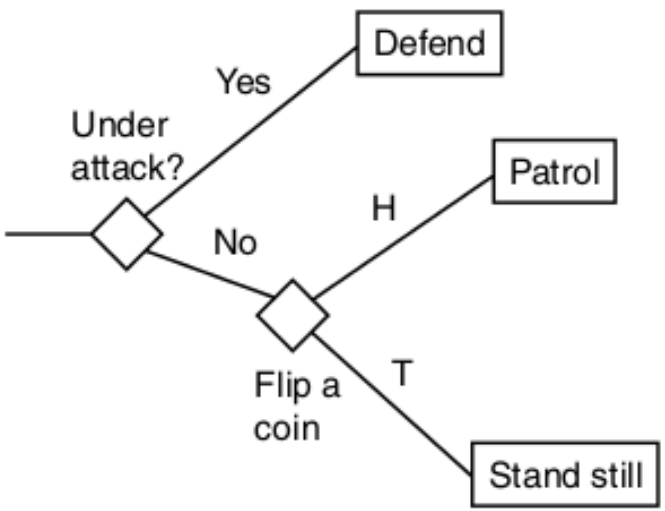
Combinations of decisions are obtained by the structure of the tree. Eg: AND, OR

Decision trees can express any function of the input attributes.
E.g., for Boolean functions, truth table row $\rightarrow$ path to leaf

Execution time depends on decisions
Eg, checking if any enemy is visible may involve complex ray casting sight checks through the level geometry.

Random Decision Trees

- Some element of random behavior choice adds unpredictability, interest, and variation
- Requires some care if the choice is made at every frame to yield stable behavior $\rightsquigarrow$ keep track of last decision



- Add a time-out information, so the agent changes behavior occasionally.

Outline

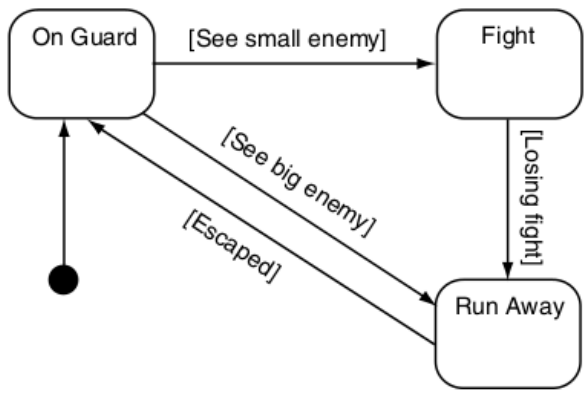
1. Other Ideas in Pathfinding

2. Decision Making
    - Decision Trees
    - State Machine
    - Behavior Trees

Finite State Machines

An FSM is an algorithm used for parsing text, eg, tokenize the input code into symbols that can be interpreted by the compiler.

- States: actions or behaviors. Chars are in exactly one of them at any time.
- Transitions: a set of associated conditions, if they are met the char changes state
- Initial state for the first frame the state machine is run



In a decision tree, the same set of decisions is always used, and any action can be reached through the tree.

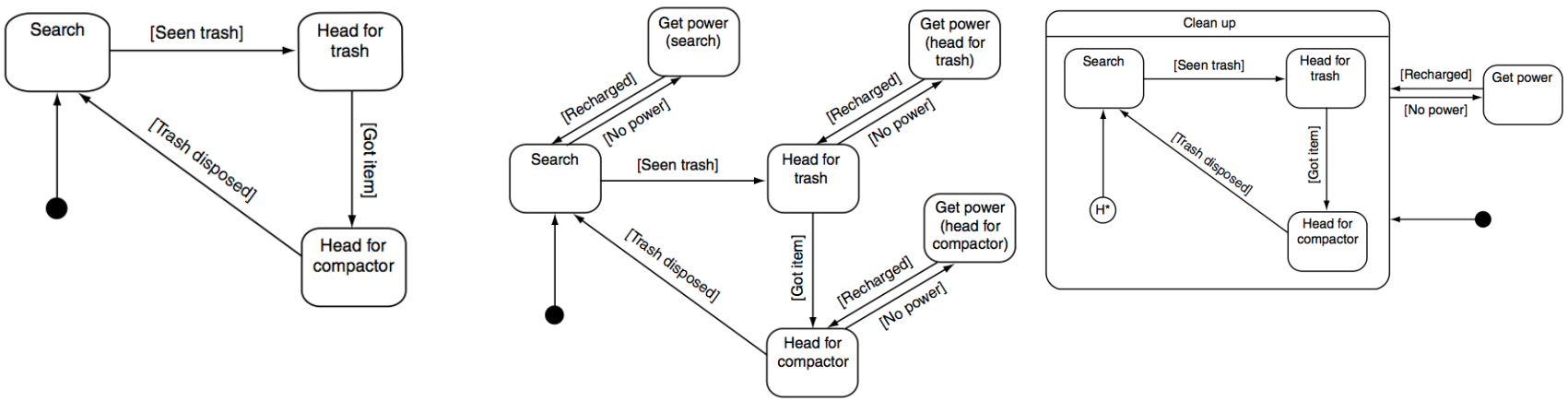
In a state machine, only transitions from the current state are considered, so not every action can be reached.

General State Machines

- set of possible states
- current state
- set of transitions
- at each iteration (normally each frame), the state machine's update function is called.
- checks if any transition from the current state is triggered
- the first transition that is triggered is scheduled to fire
  (some actions related to transition are executed)

Hierarchical State Machines

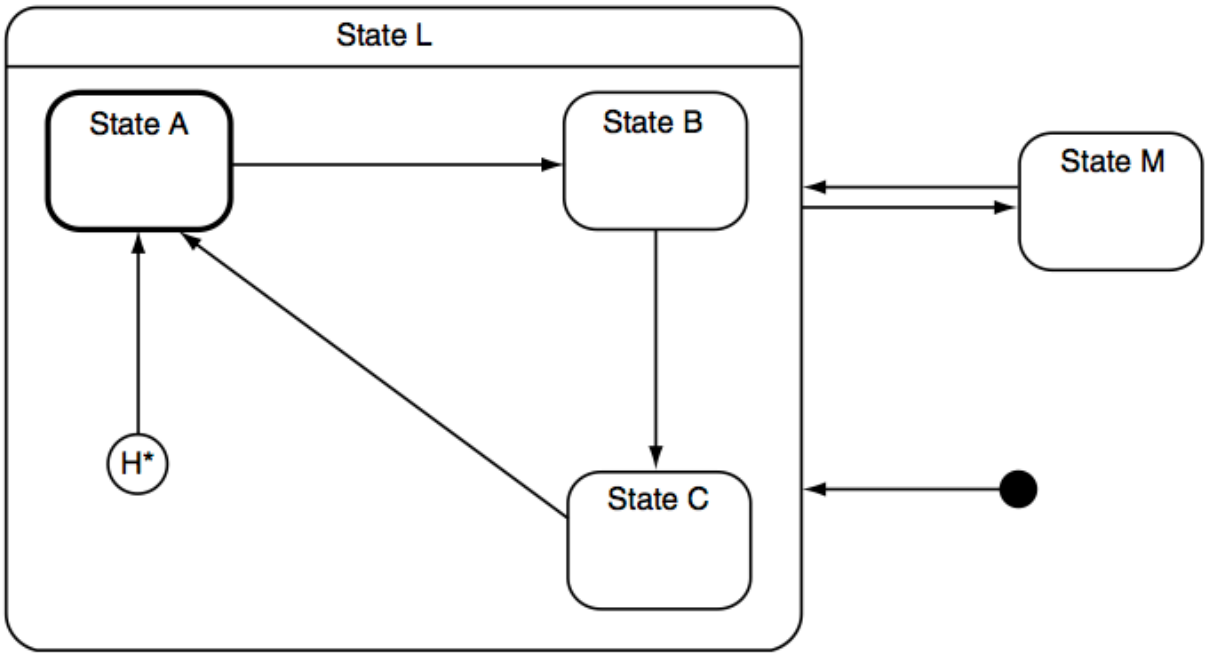
- Alarm mechanism: something that interrupts normal behavior to respond to something important.
- Representing this in a state machine leads to a doubling in the number of states.
- Instead: each alarm mechanism has its own state machine, along with the original behavior.



We can add transitions between layers of machines

Hierarchical State Machines

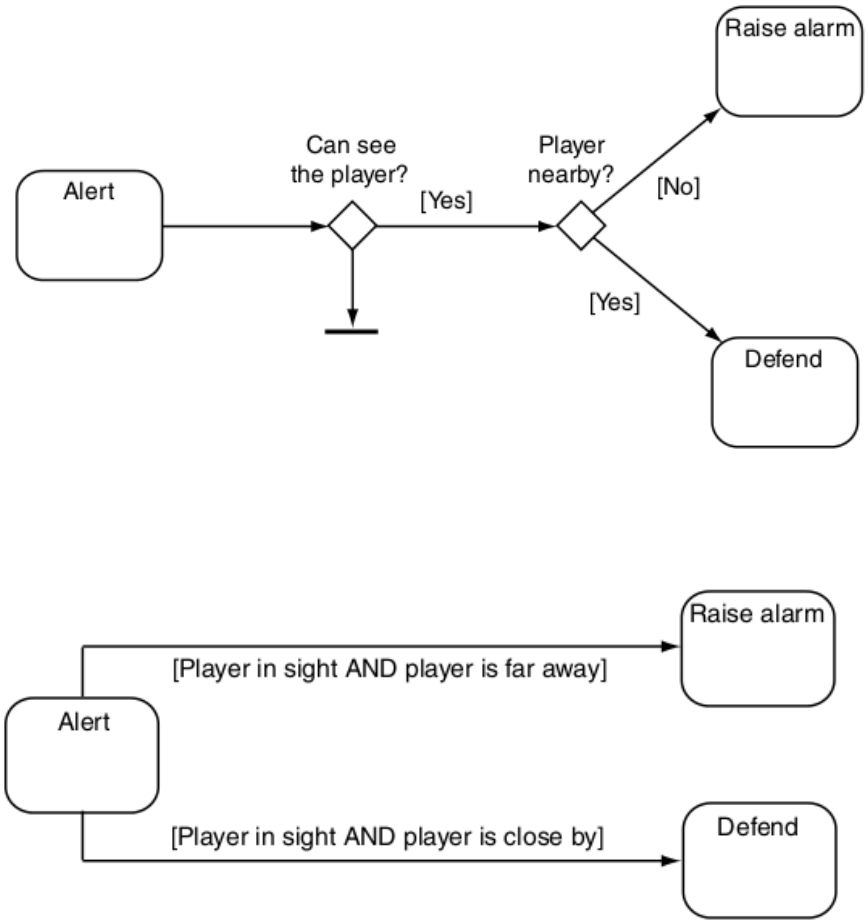
In a hierarchical state machine, each state can be a complete state machine in its own right $\rightsquigarrow$ recursive algorithm



A triggered transition may be: (i) to another state at current level, (ii) to a state higher up, or (iii) to a lower state

Combining DT and SM

Decision trees can be used to implement more complex transitions



Outline

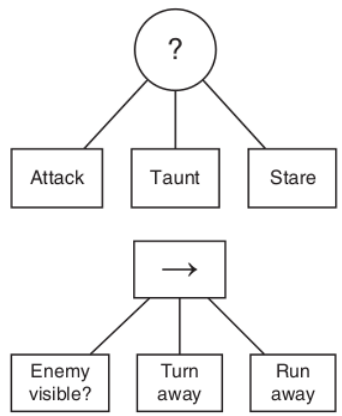
1. Other Ideas in Pathfinding

2. Decision Making
    - Decision Trees
    - State Machine
    - Behavior Trees

Behavior Trees

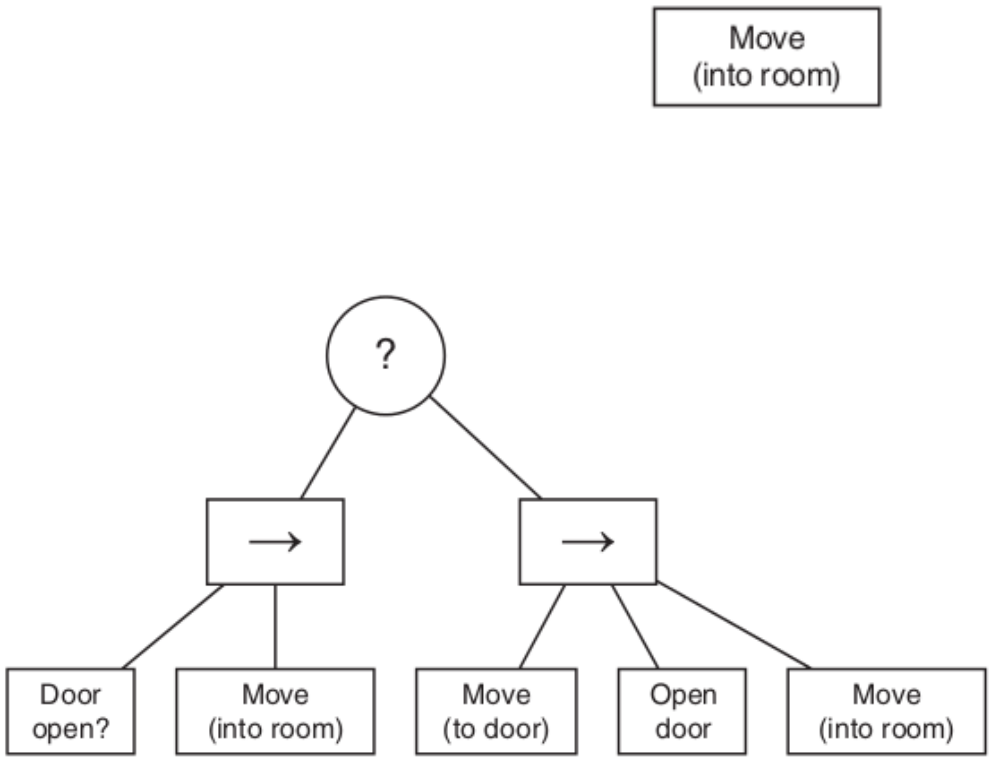
- synthesis of: Hierarchical State Machines, Scheduling, Planning, and Action Execution.
- state: task composed of sub-trees
- tasks are Conditions, Actions, Composites
- tasks return `true, false, error, need more time`
- Actions: animation, character movement, change the internal state of the character, play audio samples, engage the player in dialog, pathfinding.
- Conditions are logical conditions
- behavior trees are coupled with a graphical user interface (GUI) to edit the trees.

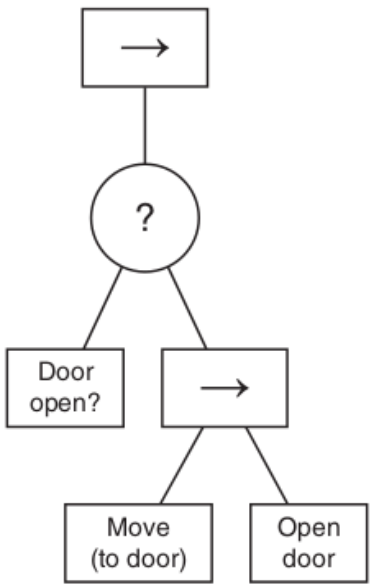
- Both Conditions and Actions sit at the leaf nodes of the tree. Branches are made up of Composite nodes.
- Composites: two main types: Selector and Sequence
- Both run each of their child behaviors in turn and decide whether to continue through its children or to stop according to the returned value.
- Selector returns immediately with a success when one of its children succeeds. As long as children are failing, it keeps on trying. If no children left, returns failure. (used to choose the first of a set of possible actions that is successful) Eg: a character wanting to reach safety.
- Sequence returns immediately with a failure when one of its children fails. As long as children are succeeding, it keeps on trying. If no children left, returns success. (series of tasks that need to be undertaken)



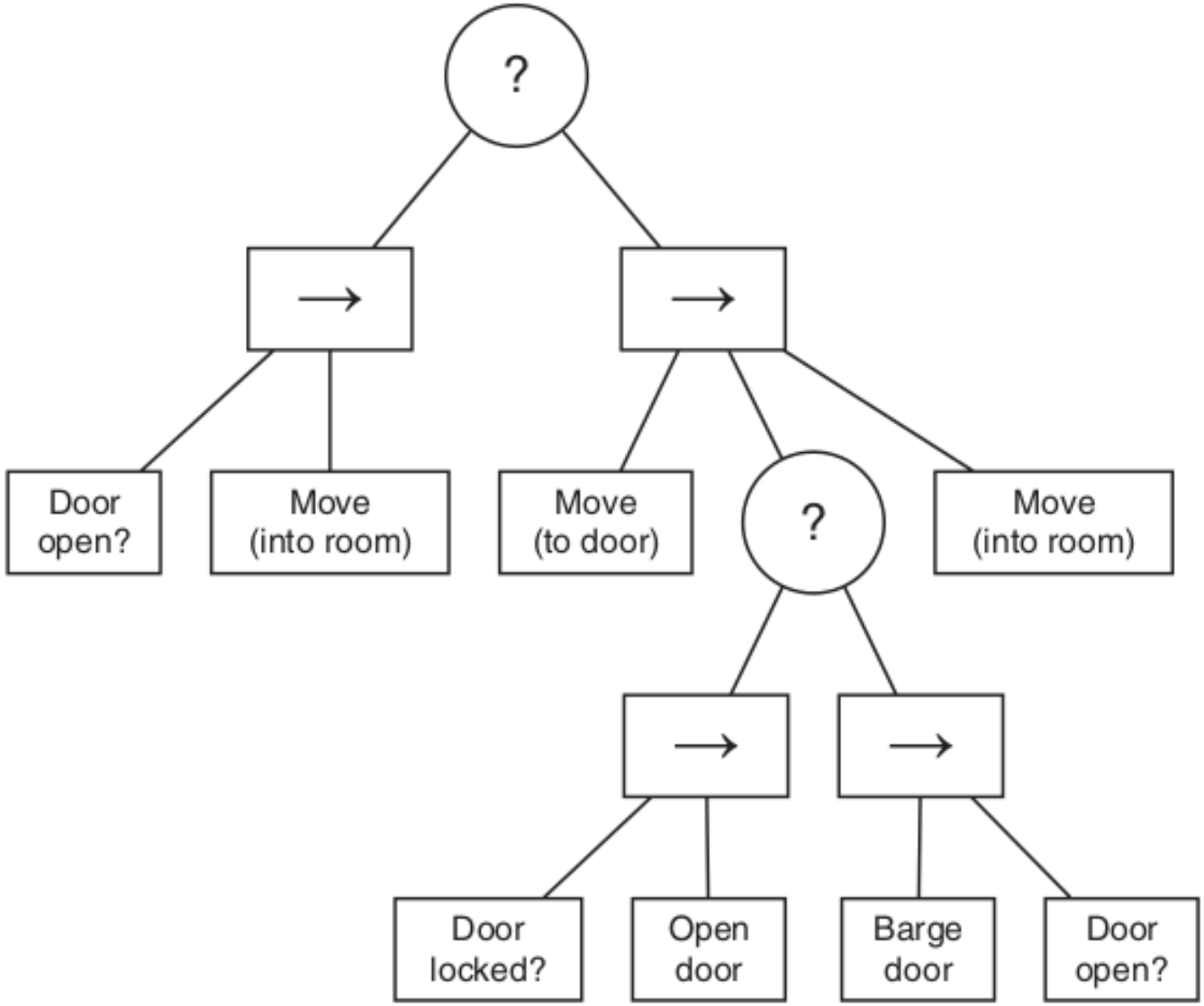
Developing Behaviour Trees

- get something very simple to work initially





Condition task in a Sequence acts like an IF-statement.
If the Sequence is placed within a Selector, then it acts like an IF-ELSE-statement


?
→
→
Door open?
Move (into room)
Move (to door)
?
Move (into room)
→
→
Door locked?
Open door
Barge door
Door open?

- behaviour trees implement a sort of reactive planning. Selectors allow the character to try things, and fall back to other behaviors if they fail. (look ahead only via actions)

- depth-first search

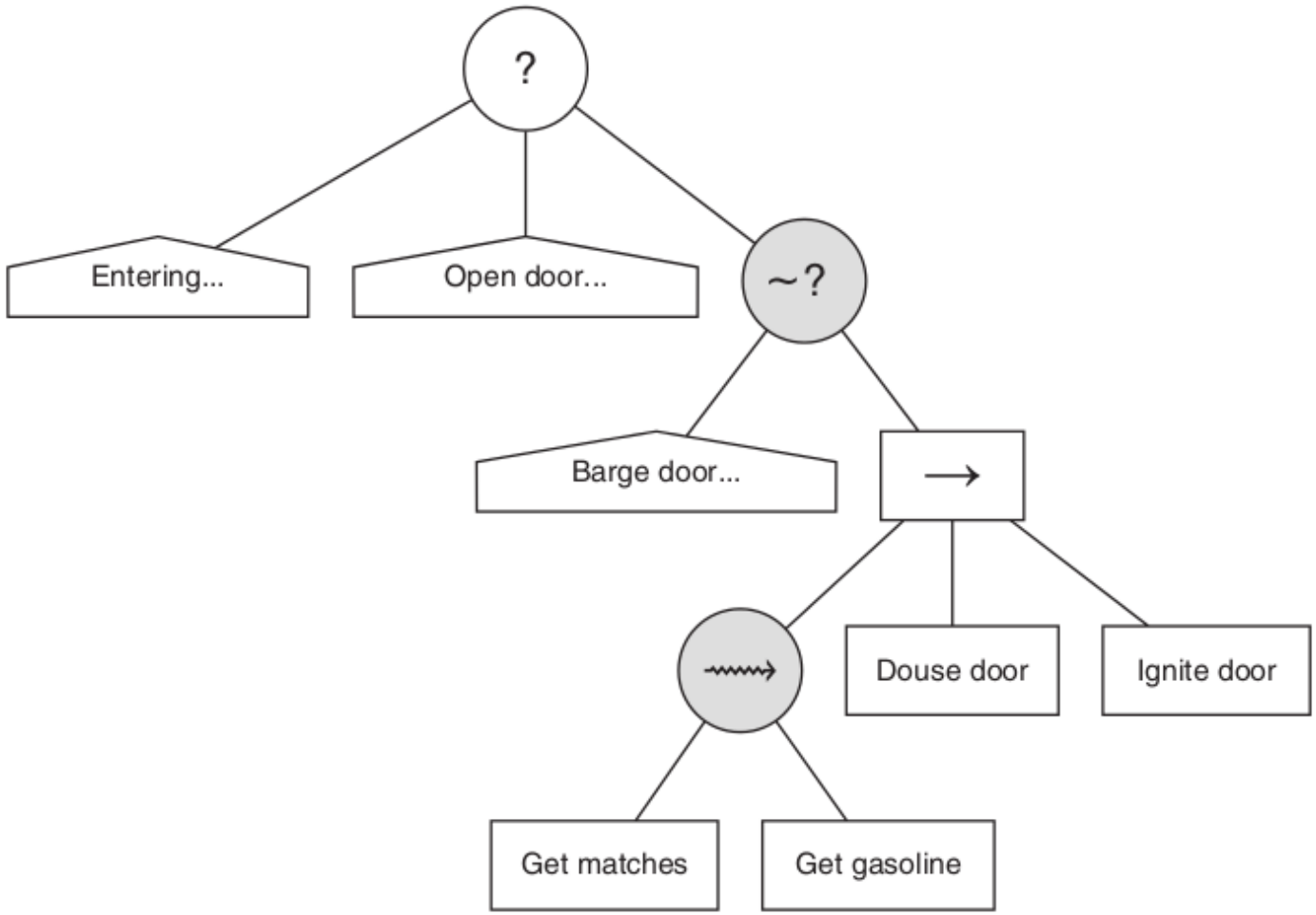
- could be written as state machines or decision trees but more complicated

Non-Deterministic Composite Tasks

- In some cases, always trying the same things in the same order can lead to predictable AIs.
- Selectors: eg, if alternative ways to enter the door, no relevant order
- Sequences: eg, collect components, no relevant order
- Some parts may be strictly ordered, and others can be processed in any order.

```
class NonDeterministicSelector (Task):
  children
  def run():
    shuffled = random.shuffle(children)
    for child in shuffled:
      if child.run(): break
    return result
```

```
class NonDeterministicSequence (Task):
  children
  def run():
    shuffled = random.shuffle(children)
    for child in shuffled:
      if not child.run(): break
    return result
```

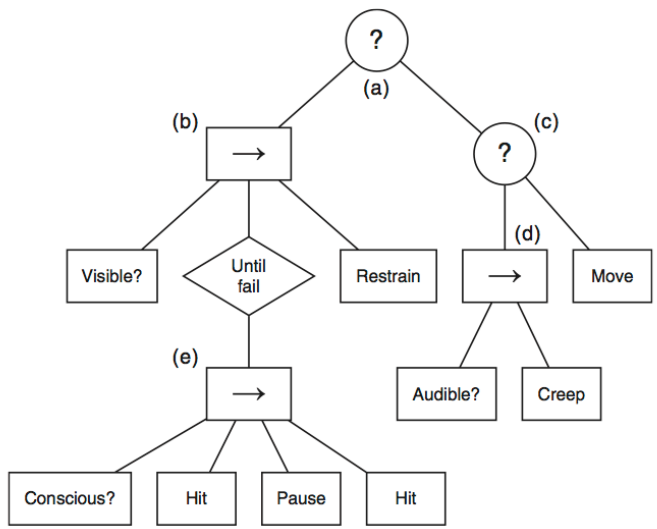

?
Entering...
Open door...
~?
Barge door...
→
Douse door
Ignite door
Get matches
Get gasoline

Decorators

- The decorator pattern is a class that wraps another class, modifying its behavior (from object-oriented software engineering).
- Composite that has one single child task and modifies its behavior in some way.

Like filters that:

- limit the number of times a task can be run (eg, does not insist with some action)
- keep running a task until it fails
- negation



Resume

1. Other Ideas in Pathfinding

2. Decision Making
   - Decision Trees
   - State Machine
   - Behavior Trees