Development of Games

Lecture 15, part 2

Game trees, rules and other methods of knowledge representation and inference

Outline

- Game trees
- Rules
- Finite state machines
- Pathing

Game Tree Search

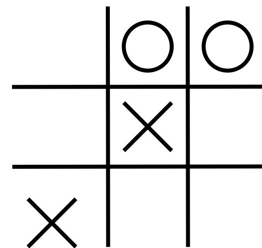
- View moves in game as decomposition operators.
- The nodes in the search space are positions.
- The edges in the search space are the moves we use to get there.
- Perform a search of the state space.
- Our aim is to win.
- In practice space is too large to search completely - normally search to a fixed depth.

Heuristic

- We can’t search entire space for a win, so we need to estimate how good a position is for us.
- *Evaluation function* estimates how good positions are for us.
- Assume positions which are good for us are bad for our opponent and vice versa:
  - positions with the highest (i.e. maximum) evaluation are best for us,
  - positions with the lowest (i.e. minimum) evaluation are best for our opponent.

An Example Evaluation Function

- For example in noughts and crosses, score
  - 1000 if we win, -1000 if we lose,
  - 100 if we have centre, -100 if opponent has centre
  - 10 for each corner we have, -10 for each corner opponent has,
  - 5 for each edge we have, -5 for each edge opponent has.

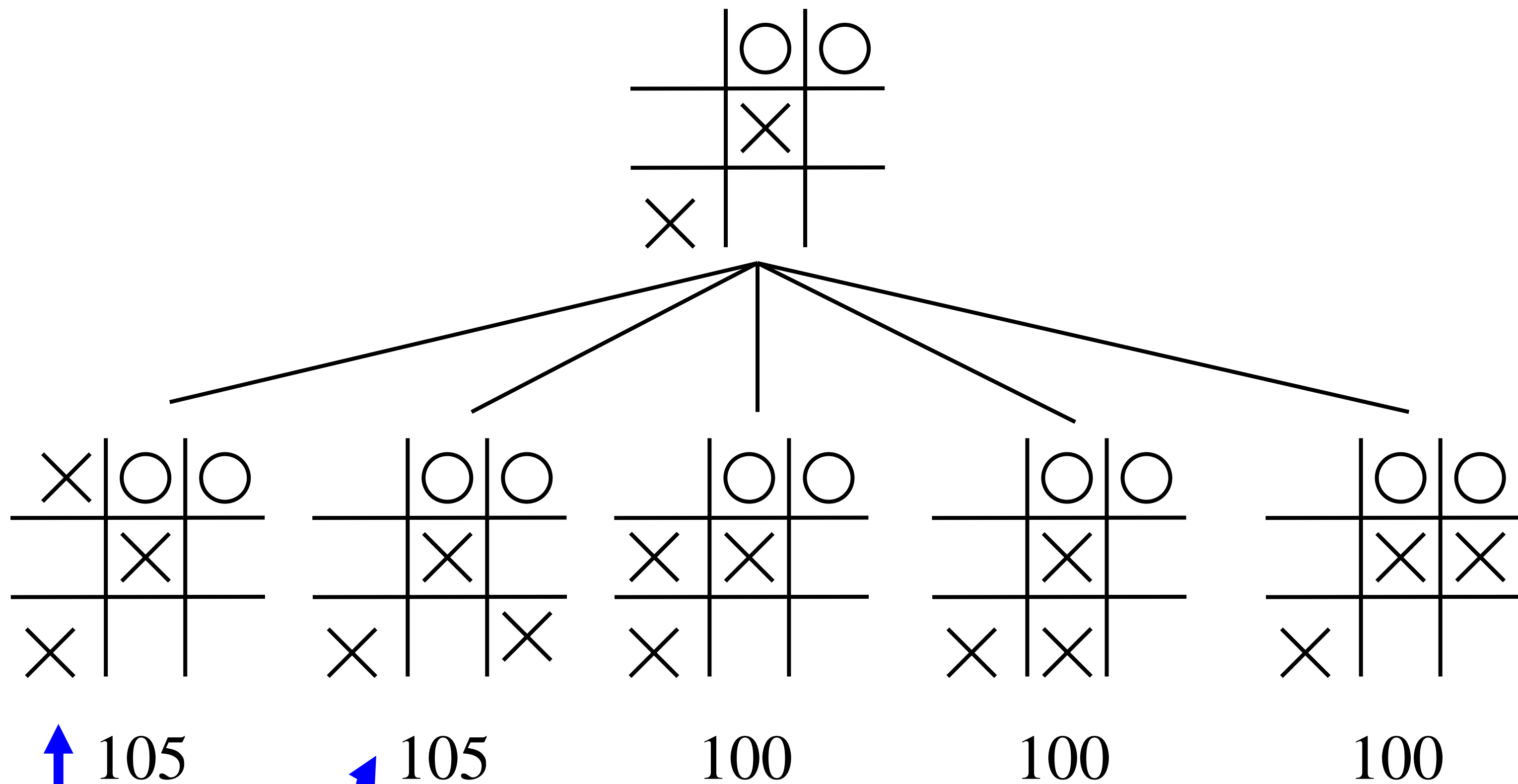


Score for X is 100 (centre) + 10 (corner) - 10 (opponent corner) - 5 (opponent edge) = 105.

Shallow Search

- In a given position, the computer can choose next move by:
  - generating all successor positions which can be reached in one move from the given position,
  - applying evaluation function to each of them,
  - playing move which leads to successor position with highest value of evaluation function.

- For example...



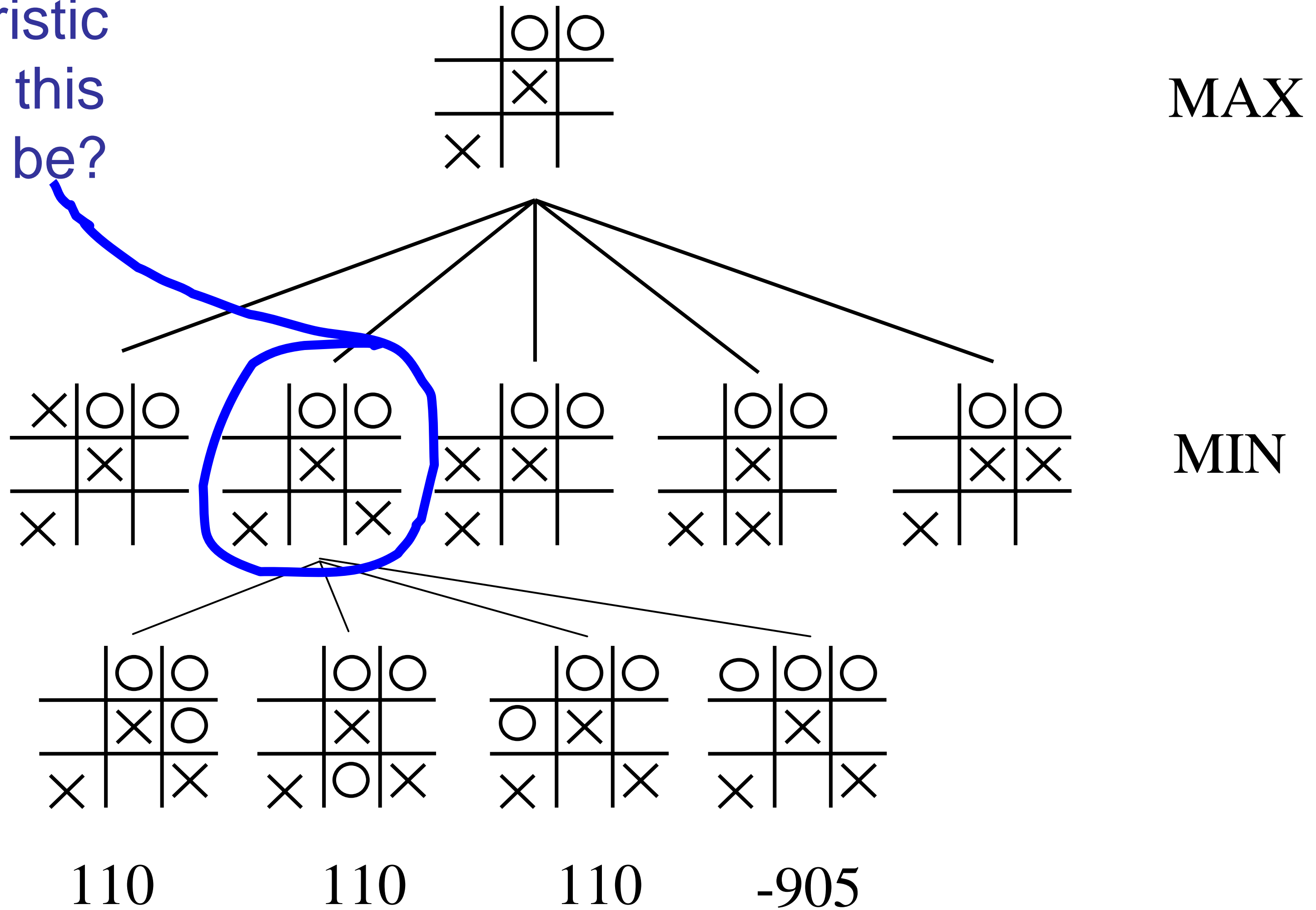
- The computer (X) would play either of these 2 moves (and the heuristic does not say which).
- But if it plays the second of them, it will lose.

More Search

- The computer fell into a trap because it didn’t take account of the moves its opponent could make.
- We would like to find a way to give the second position a lower heuristic value.
- So, let’s look at the original position again and search deeper in the move tree:

Searching deeper

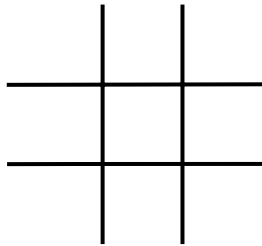
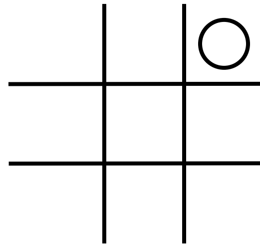
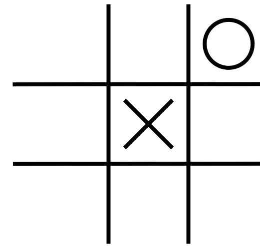
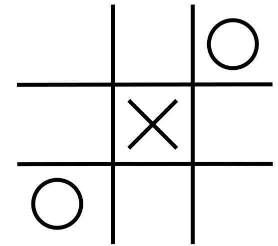
What should the heuristic value of this position be?



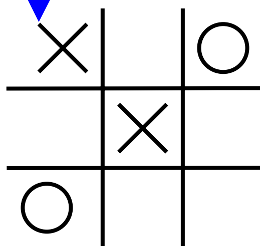
Adjusting Evaluation Function

- Now, how good is the circled position for X?
- What is its heuristic value?
- It is O's move in this position.
  - Assume O will play the move which gives the best position for O.
  - Assume O uses the same evaluation function as X.
  - Then O will pick the win, with evaluation -905.
- So X adjusts the evaluation function of the circled node to -905
  - which is the minimum of 110, 110, 110, -905.

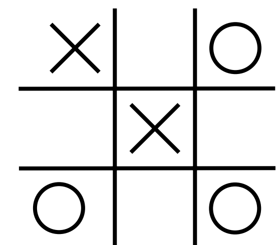
- We may need to look even deeper:
  - after the following sequence of moves:

…the computer would continue by playing in the corner, since that gives the highest heuristic value (90).



…but now O trivially wins by playing in the other corner.

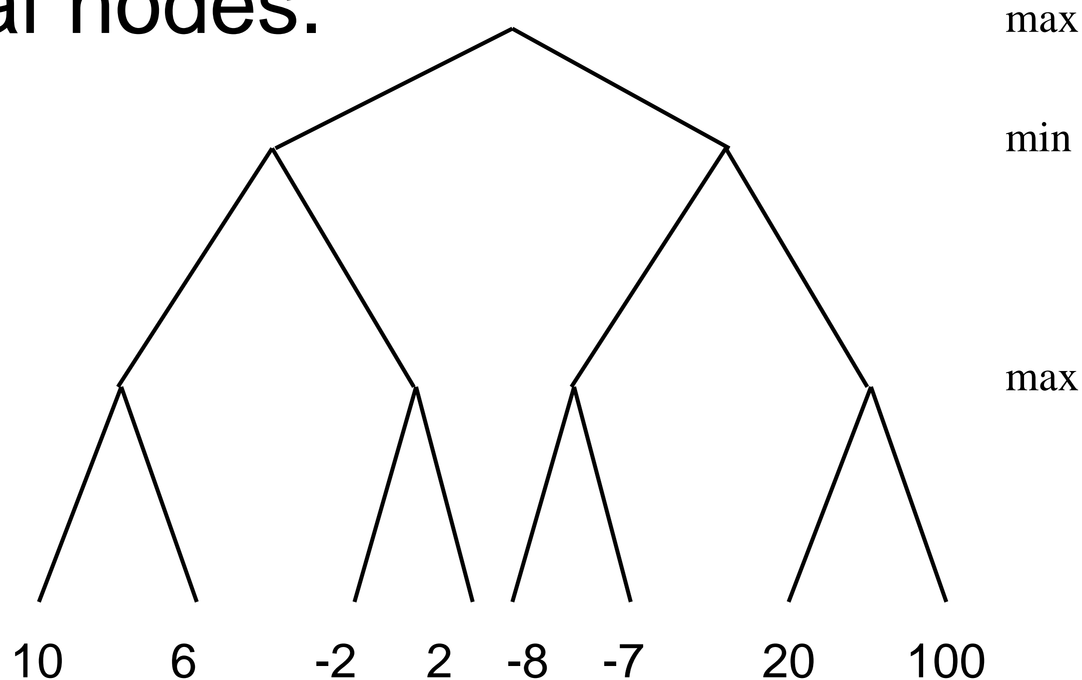


Andrey V.Gavrilov
Kyung Hee University

Minimax

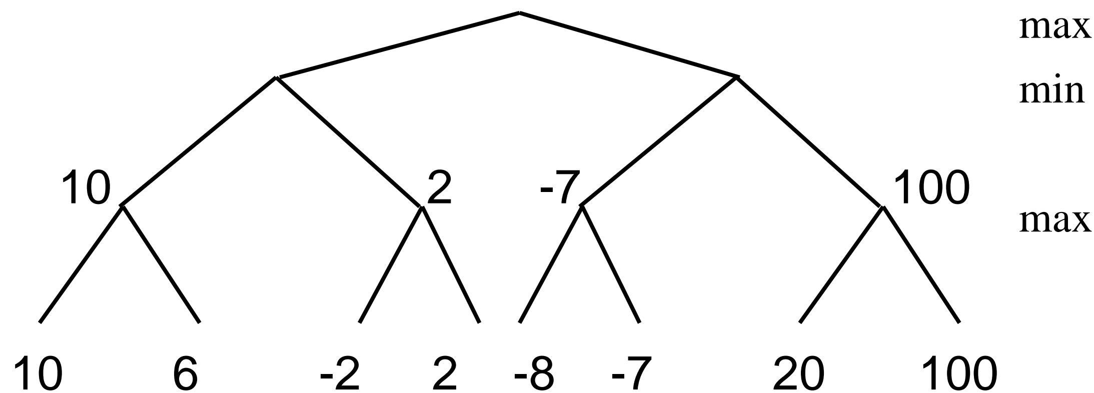
- *Minimax* algorithm searches to a fixed depth.
- Adjusts evaluation of positions depending on what the other player can do next:
  - Draw state space tree to fixed depth.
  - Root node is “ours” - we will pick child which has highest score for us. Call this a *max node*.
  - Immediate children of root node, i.e. at 1st level belong to opponent who wants to pick move with worst i.e. lowest score for us. Call this a *min node*.

- Continue down through levels like this, alternating *min nodes* with *max nodes*.
- Calculate evaluation function only at leaf nodes.



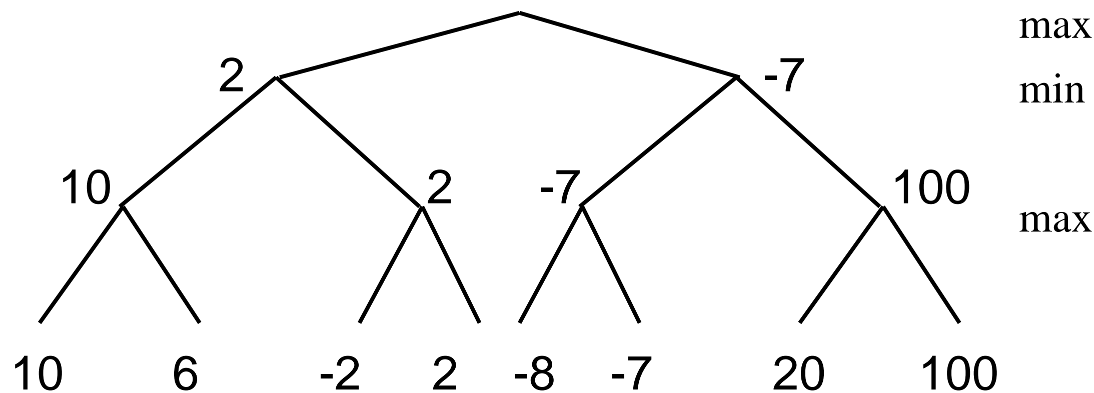
Propagating Evaluation

- At the *max* nodes immediately above the leaves, the player there will choose the move giving *maximum* evaluation.
- The heuristic value of each of these nodes is the *maximum* of the values of its children:



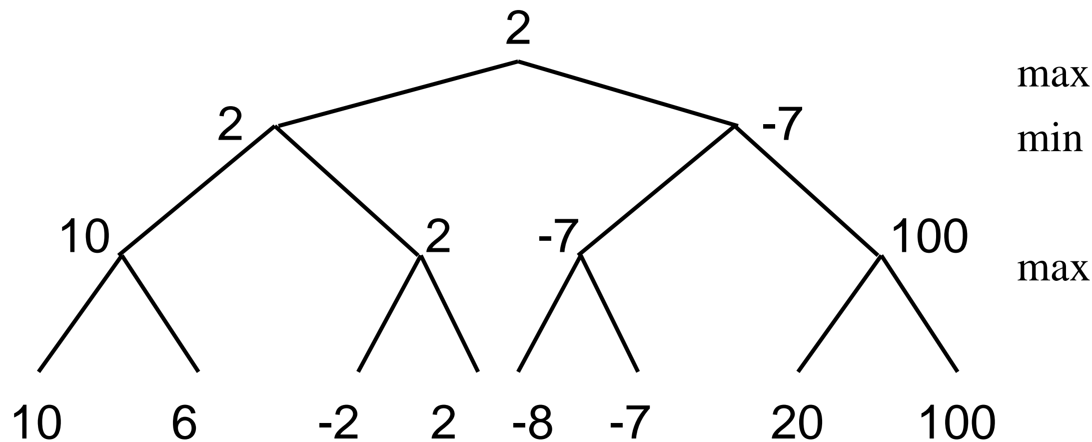
Propagating Evaluation

- At the *min* nodes at the next level, the player there will choose the move giving *minimum* evaluation.
- The heuristic value of each of these nodes is the *minimum* of the values of its children:



Propagating Evaluation

- Finally, the top node is a *max* node.
- The heuristic value of the top node is the *maximum* of the values of its children:



- The computer will play the move which leads to this maximum value.

Calculating Minimax

- Consider a node *n* in the search tree.
  - IF *n* is at bottom of tree, then *h(n)* = *eval*(*n*), where *eval* is our simple evaluation function.
  - If *n* is a MIN node with e.g. 3 children *a,b,c*, then *h(n)* = minimum of *h(a), h(b), h(c).*
  - If *n* is a MAX node with e.g. 3 children *a,b,c*, then *h(n)* = maximum of *h(a), h(b), h(c).*
- The selected move is the move at the root node which has the highest value of *h*.

Alpha-beta

- Alpha-beta is an optimisation of minimax.
- Maintains cutoff values, called *alpha* and *beta*, at each node which allows some parts of the search space to be pruned.
- Performance depends on order in which positions are evaluated.
- Alpha-beta is the algorithm used in most game-playing programs.
- Pseudo code in *http://www.xs4all.nl/~verhelst/chess/search.html*

Where Alpha-Beta Fails

- Alpha-beta is widely used but not always appropriate:
  - In some games the search space is too big, e.g. in Go there are hundreds of moves to consider at each level in the search.
  - In some games (e.g. Go) it is hard to get a reasonable evaluation of a position.
  - Most card games, e.g. poker, bridge (for both the above reasons).
  - Psychological games.

Rules

Rule - (I, A, P, A->B, F)

- I – identifier of rule (number or name)
- A – area of using of rule
- P – condition using of rule
- A – condition of rule
- B – conclusion of rule
- F – tail conditions (any comments or additional actions)
- A->B – core of rule, may be different kinds of interpretation

More simple rules

Rule - (I, A->B)

- May be implemented in games as table

Examples of possible rules in Games

- If *resource=oil* and *resource=steel* and *knowledge=combustion* and *access=sea* then *opportunity_of_build=ship*
- If *my_force<enemy_force – level_of_spirit* then *attack*
- If *enemy_unit_defeated* then *level_of_spirit + delta*
- If *destroyed* > 70% then *go_to_repair*

Kinds of interpretation of rule

- Logical
  - A – logical function with &,V, not
  - If one is true then rule are executing
- Probabilistic
  - A – logical function with &,V, not
  - Rule are executing with any probability
- Threshold
  - A - set of features, which are adding with weights and rule are executing if addition is more then any threshold (as in model of neuron)

Kinds of inference

- Backward chaining
  - From goal to facts (as in Prolog or as in top-down method of grammatical analyzing)
- Forward chaining
  - From facts to goal (as in bottom-up method of grammatical analyzing)

Forward chaining inference

match-resolve-act cycle

The match-resolve-act cycle is the algorithm performed by a forward-chaining inference engine.
It can be expressed as follows:

loop
1. match all condition parts of condition-action rules against working memory and collect all the rules that match;
2. **if** more than one match, *resolve* which to use;
3. perform the action for the chosen rule

until action is STOP **or** no conditions match

Step 2 is called conflict resolution. There are a number of conflict resolution strategies.

Conflict resolution strategies

Specificity Ordering

If a rule's condition part is a superset of another, use the first rule since it is more specialized for the current task.

Rule Ordering

Choose the first rule in the text, ordered top-to-bottom.

Data Ordering

Arrange the data in a priority list. Choose the rule that applies to data that have the highest priority.

Size Ordering

Choose the rule that has the largest number of conditions.

Recency Ordering

The most recently used rule has highest priority. The least recently used rule has highest priority. The most recently used datum has highest priority.

The least recently used datum has highest priority

Context Limiting

Reduce the likelihood of conflict by separating the rules into groups, only some of which are active at any one time. Have a procedure that activates and deactivates groups.

Backward chaining inference

In backward chaining, we work back from possible conclusions of the system to the evidence, using the rules backwards. Thus backward chaining behaves in a goal-driven manner.

Backward chaining uses stack for store current goals (order of searching of tree) for possibility to select alternative path in case fail.

When backward chaining is better?

- It is needed to prove one goal, and what is goal is known preliminary
- Initial number of facts is enough large
- Number of query of facts during inference is enough small

When forward chaining is better?

- We preliminary don’t know what will be decision from several possible (its may be strongly differ between them)
- Part of time for dialog (query of facts) is relatively small in differ with part for generation of facts from other sources
- During inference some hypothesis may be generated
- It is needed to make decision in real time as answer on appearance of facts

Representation of uncertainty in rules

- Facts with confidence
  - Confidence may be (0,1), (-1,1), (0,100), (0,10)
  - Are processing (during checking of condition) in compliance with formulas of fuzzy logic
- Rules with confidence
  - Confidence *Conf* is corresponding to any conclusion
  - It means that if confidence of condition is 1 (100%), then fact-conclusion is appending to base of facts with confidence *Conf*

- Advantages of rules as method for knowledge representation
  - Flexibility
  - Possibility of nonmonotonic reasoning
  - Easy understandability
  - Easy append to knowledge base
- Disadvantages
  - Low level of structuring, so it is difficult to explore of knowledge base
  - Orientation on consistent solving of task
  - Without special program support may be problems with knowledge integrity during its expanding
- Disadvantages in Games
  - May be too large and to demand many resources

Encoded List Processing a pattern of behaviours (Cont.)

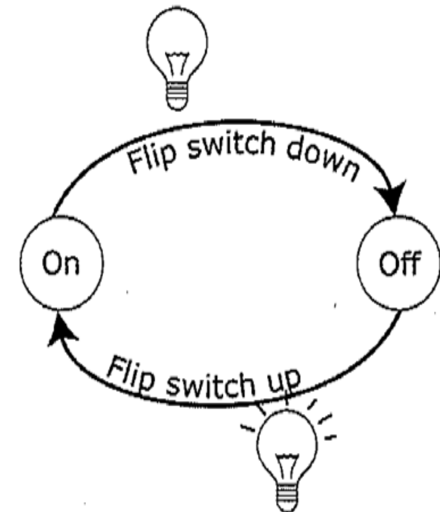
- A set of pre-recorded patterns or lists of behaviours that they have either learned from experience or are instinctive.
  - A pattern is a sequence of steps we perform to accomplish a task.
  - For example, when you drive to work, school, or your house, you are following a pattern. You get into your car, start it, drive to the destination, stop the car, turn it off, get out, and finally do whatever it is you're going to do.
  - Many intelligent creatures have pre-recorded patterns or lists of behaviours that they have either learned from experience or are instinctive.

Encoded List Processing
a pattern of behaviours

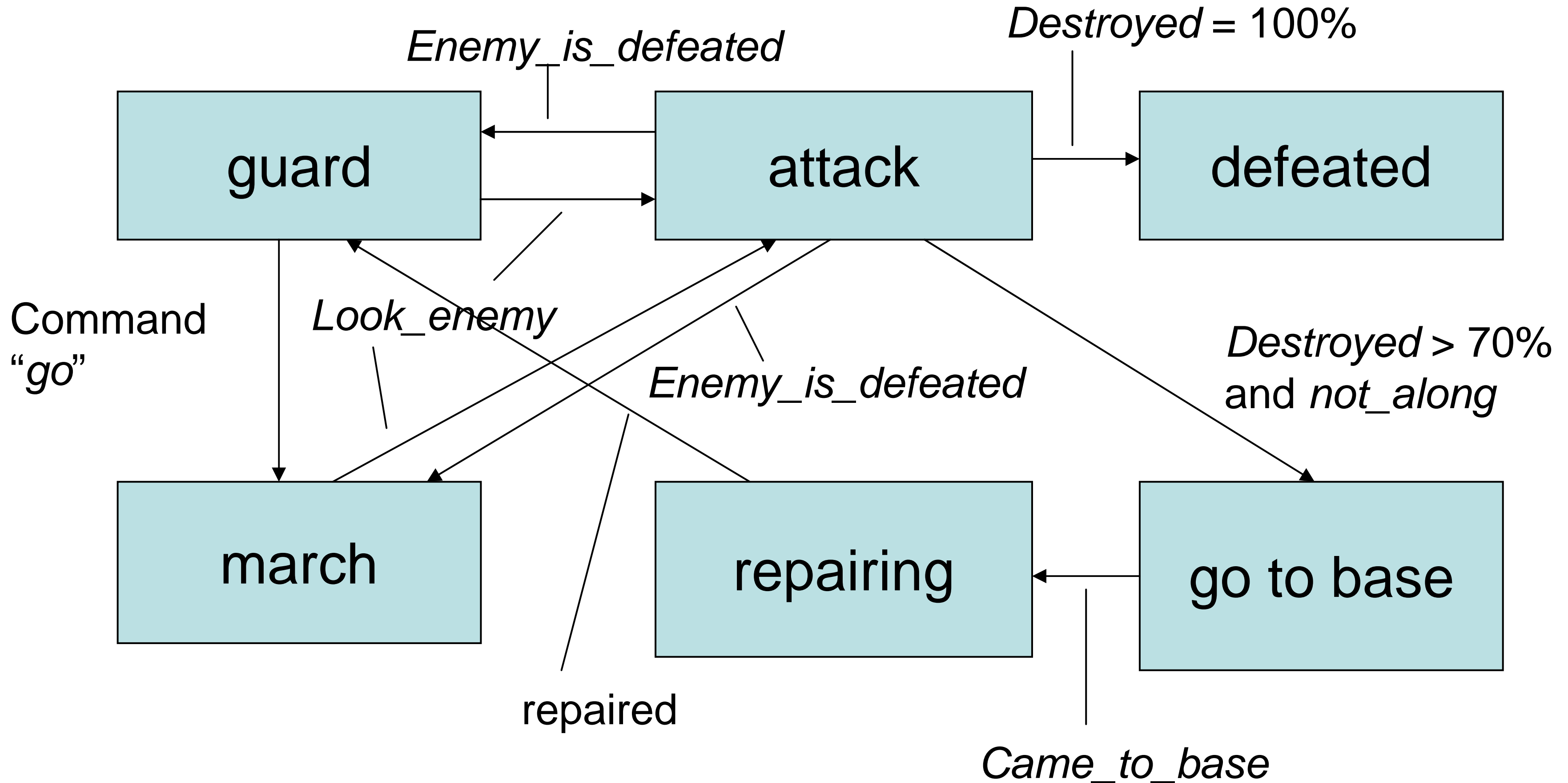
- Patterns are a good way to implement seemingly complex processes in game AI.
  - In fact, many games today still use patterns for much of the game logic.
  - Simply by using an input array to a list processor.
- An encoded list may have the following set of valid instructions:
  - Turn right * Turn left * Move forward * Move backward * Sit still * Fire weapon.
  - 6^16 different possible patterns or roughly
  - 2.8 trillion different behaviors.

Finite State Machines (FSM)

- *A finite state machine is a device, or a model of a device, which has a finite number of states it can be in at any given time and can operate on input to either make transitions from one state to another or to cause an output or action to take place. A finite state machine can only be in one state at any moment in time.*



Example of possible finite machine, described behavior of unit in Game



Fuzzy State Machines (FuSM)

- Similar to the FSM except that a given set of stimuli maps, not only to a specific response, but also to a set of possible responses.
- Incorporating ideas from fuzzy logic, this method brings great flexibility into the games domain.
- This method can therefore generate a variety of responses to a given set of stimuli.

Fuzzy State Machines (FuSM) (Cont.)

- For example, “if the opponent is too strong, run away.” Here the words “too strong” are simple to judge in terms of true or false and there could be many possible responses to the situation.
- A variety of methods have been developed to select an appropriate response, such as
  - probability weights, or
  - threshold to trigger an action.

Chase Algorithm

- The **chase algorithm** is a classic example of a deterministic algorithm.
- the coordinates of the bad guys and the coordinates of the player as inputs into a deterministic algorithm that outputs direction changes or direction vectors for the bad guys in real time.
- If we wanted to reverse the logic and make the bad guy run then the conditional logic could be inverted or the outcome increment operators could be inverted.

A* Pathing Algorithm

- A* is an algorithm that allows the computer to work out a path from A to B (navigating around obstacles). It is essentially part of graph theory, though you wouldn’t really believe it if you looked at the code. A* is also used in chess games, route-finding software and a whole variety of applications. A* is computationally expensive though, so be warned!
- Essentially, there is a set of nodes (map locations and a cost) called OPEN and another set called CLOSED. Each time through the main loop, you pick out the best element from OPEN (where "best" means "the one with the lowest cost"), and you look at its neighbours. You then put any unvisited neighbours into the OPEN.
- The cost of a node is the sum of the current cost of walking from the start to that node and the heuristic estimate of the cost from that node to the goal.

Pseudo code

Insert starting point into OPEN set (it is recommended you use a HEAP structure for the set).

while the OPEN set is not empty:

Get the "Best" node from the OPEN set (i.e. The one with the least cost).

Exit from loop if we're at the goal node

Insert all valid neighbours into the OPEN set (with Distance already travelled + Heuristic Cost and Direction that the cell was moved into)

Delete the "Best" node from the OPEN set and insert into the CLOSED

loop

if the OPEN set is not empty, then

Backtrack from goal node to start node using the CLOSED set and directions.

else

No path is possible

end if