AI for Games

Doron Nussbaum

- Introduction
- Movement and path planning
- Games and trees
  - MinMax
  - Decision trees
- Finite State Machines
- Agents
- Other

Intelligence Definition

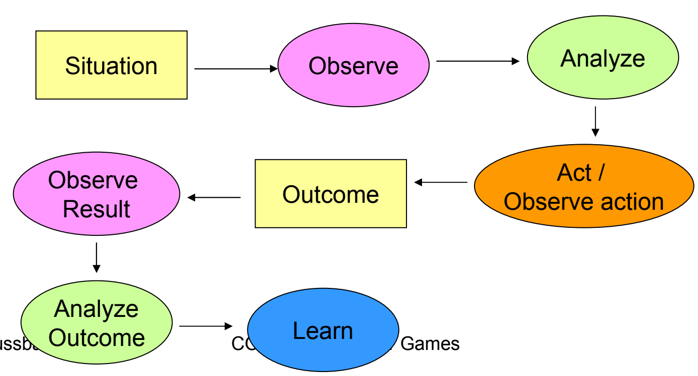
- "The ability to acquire and apply knowledge and skills" (Oxford dictionary)

- Ability to understand, reason, grasp issues and complex problems/ideas as well as ability to learn from experience of self and others

- Artificial Intelligence
  - An attempt by machines (computers) to be intelligent

Humans and Intellegence

- We have the ability to:
  - Learn → look at a situation



AI

- In Academia – understanding intelligence (theory)
  - Thought process
    - Behaviour prediction
  - Comprehension
    - Machine vision
  - Natural language –
    - understanding semantics
    - Carry out a conversation

- Engineering – Use AI to solve real problems
  - Searching (Google)
  - Stock market predictions

Are we there yet?

- Not Yet!

- Large number of applications
  - Machine vision
  - Speech recognition
  - Stock market prediction
  - …

- Spectrum of domain is hard to achieve
- Some specific areas are promising
  - Chess
  - Carrying out a conversation on specific topics (e.g., whether, daily activities)

- What is in the way
  - Unscripted actions
  - Not always logical
  - Emotions

What about Game AI?

- Game AI is different
  - Provide entertainment
  - Attract the player
  - Should be realistic
- PACMAN
  - Is it AI?
- AI in games provides
  - Interaction with the player
  - Level of unpredictability
  - Somewhat no repetition in the game

http://free-extras.com/images/pacman_game-1973.htm

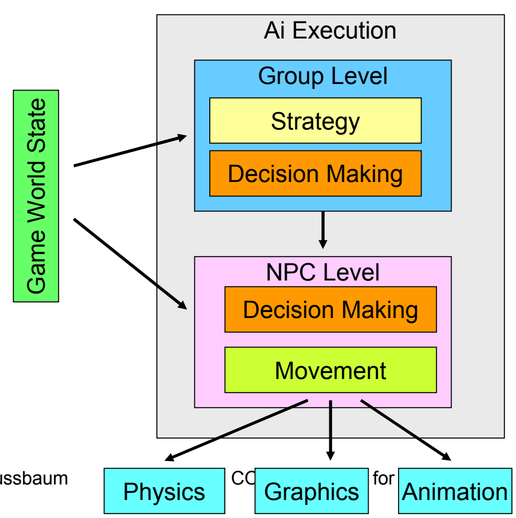
What Should Game AI Be

- Should be good
  - Cannot be too smart – should have some built in flaws
  - Provide fun
- Provide good perception (no obvious flaws)
  - Should not look dumb
  - No unintended flaws – cannot be defeated using a “secret path”t
- Must be fast (real time)
- If possible configurable
  - Not hard coded by programmer
- Can adjust to different level of players

What about Game AI?

- Must be smart, but purposely flawed
  - Loose in a fun, challenging way
- No unintended weaknesses
  - No "golden path" to defeat
  - Must not look dumb
- Must perform in real time (CPU)
- Configurable by designers
  - Not hard coded by programmer
- "Amount" and type of AI for game can vary
  - RTS needs global strategy, FPS needs modeling of individual units at "footstep" level
  - RTS most demanding: 3 full-time AI programmers
  - Puzzle, street fighting: 1 part-time AI programmer
  - All of project 2. ☺

AI Model

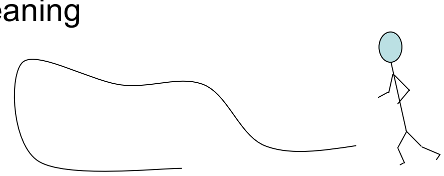


Movement

- Movement relates to the way an NPC moves in the world
  - Wander
  - Seek
  - Chase / home in / zoom to target
  - Flee

Wander

- NPC moves in the world
  - Without a particular goal
  - Usually aimlessly (no logic in movement)
  - Scouts an area (not exploring)
    - Guarding
    - Cleaning



wander

- Address motion
- Address orientation

- Set up a target
  - Set up a path
  - Path can be a sequence of short line segments.

- Update the motion
  - Speed
  - Velocity
  - Orientation

- Use regular motion equations (distance, speed, velocity)

Seek

- Similar to wander (something in mind)
- Searching for something
  - Treasure
  - Enemy
  - Weapon

- Create constraints – (when is the target visible?)
  - Distance constraints
  - Visibility – colour, size, text
  - Type – searching for a box, target is a sphere

Seek

- Address motion
- Address orientation

- Set up a search pattern
  - Wander
  - Logical scan – in circles, side to side, moving in a maze

- Update the motion
  - Speed
  - Velocity
  - Orientation

Chase

- Target is visible
  - Known location of target
  - Target may be stationary or in motion

- Realistic actions
  - Change of direction is affected by speed
  - Cannot change direction on the spot (e.g., a car)

- Issues
  - Overshooting the target

Chase

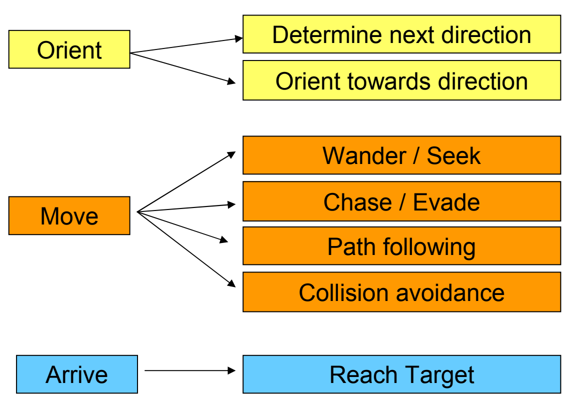
- Address motion
- Address orientation

- Set up a static path
  - Path can be direct
  - Path needs to avoid obstacles

- Set up a chase path (dynamic path)
  - Follow the path
  - Zoom in on current location
  - Prediction of future location

- Update the motion
  - Speed
  - Velocity
  - Orientation

Steeting



Path Planning

- Determine how to move in space
  - Wander – move aimlessly
  - Seek – move to a particular location
- What does one have
  - Start point
  - Target point
  - Obstacles
  -

Path Planning

- What is missing?
- Free space!!!
  - Space in which one can move freely
- This may not be trivial
  - Object has area/volume (2D/3D)

Path Planning

- Motion –
  - Assume that object is a point →
  - Move a point in the free space

- Creating free space
  - Convert object to a point
  - Enlarge obstacles accordingly (e.g., Minkowski sums)

Path Planning (last)

- How to move in free space?
  - Hard – not knowing where to go, when to turn

- Visibility
  - Move in a shortest path "notion"

- Attempt to convert the space into a graph

Path Planning

- Types of problems that may be of interest
- Guards placement
  - How many guards are needed to guard area
  - Where to position guards so that the guarded area is covered
- Guarding path
  - Is there a path that a guard can see the guarded area
    - All the time
    - At least once during the motion
  - How many guards are needed?
  - What should the paths be?
- Gaming –
  - Each guard is an autonomous object
  - Place less guards then needed – give the player a chance

Path Planning

- Most of the time path planning is to reach a goal.
  - Shortest path
  -
- What is the meaning of shortest path?

Path Planning

- Input:
  - a graph
    - Vertices
    - Edges
    - Weights
  - Start point
  - End point (in most cases)
- Output: a path (possible ∅)

- Algorithm?
  - Path traversing algorithms?

Path traversing Algorithms

- Breadth First Search (BFS)
  - Explore closest neighbourhood first

- Depth First Search (DFS)
  - Explore furthest neighbourhood first

Path Planning

- Algorithms for path planning
  - Best first
  - Dijkstra shortest path
  - A*
  - Hierarchical shortest path

Constructing Graphs

- Depending on the world
  - Grid (cell based)
    - Four connected
    - Eight connected (3x3)
    - 16 connected (5x5)
  - Vector based
    - TIN
    - Obstacles
    - Free space

Best First

- Usually used on a grid based graphs
- Idea
  - Attempt to move as “fast” as possible to the target (Greedy algorithm)
  - Attraction to the target/goal position

Dijkstra Shortest Path

- Search for the target around the start point until target is found.
  - No relation to the target point
- Algorithm properties
  - Triangle inequality
    $cost(\pi(u,w)) <= cost(\pi(u,v)) + cost(\pi(v,w))$
  - $\delta(u)$ is the minimum $cost(\pi(u,v))$

Dijkstra Shortest Path

- Let (v,u) be an edge in G
  - $\delta(u)$ <= $\delta(v)$ + weight(u,v)

- If $v_0,v_1,\ldots,v_k$ be a shortest path from v_0 to v_k then $v_i,\ldots,v_j$ is a shortest path from v_i to v_j where 0<= i,j <= k and $i<j$

- $v_i \leftarrow \infty$
- $s \leftarrow 0$
- Insert all vertices to a priority queue Q
- While ($Q \neq \emptyset$)
  - $u \leftarrow$ top(Q)
  - for all v which are neighbours of u
    - if cost($\pi(s,u)$)+weight(u,v) < cost($\pi(s,v)$) then
      - cost($\pi(s,v)$) $\leftarrow$ cost($\pi(s,u)$)+weight(u,v)
      - Update parent of u
      - Update Q with new cost of v

What is the “problem” with Dijkstra?

What is the “problem” with Dijkstra?

- A BFS algorithm
- Search everywhere without any relatioship to the target

- Solution ?

What is the “problem” with Dijkstra?

- A BFS algorithm
- Search everywhere without any relatioship to the target

- Solution
  - Combine Best First and Dijkstra

A* Path Algorithm

- A heuristic algorithm
- Attempts to take the best of both worlds
  - The BFS behaviour of Dijkstra
  - The DFS behaviour of Best First

A* Path Algorithm

- Modify the comparison statement of Dijkstra priority queue algorithm
- Instead of extracting u such that $\text{cost}(\pi(s,u))$ is the minimum in Q
- Use $\text{cost}(\pi(s,u)) + \text{Estimate}(u,t)$

A* Path Algorithm

- What kind of estimate one can use?

Large Graphs

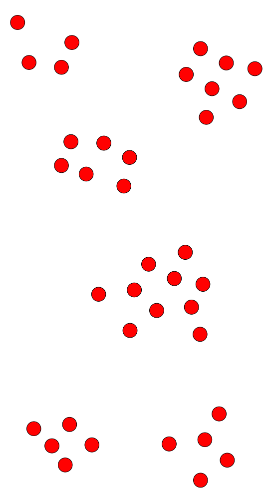
- What can be done with large graphs?
- How can many queries be handled?
- Scalability
  - Domain size
  - Number of players
- Solution
  - Speed up queries
  - Merge queries

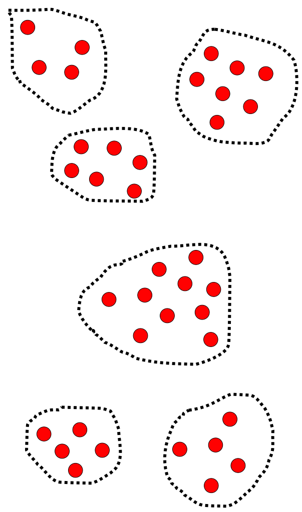
Hierarchical Graphs

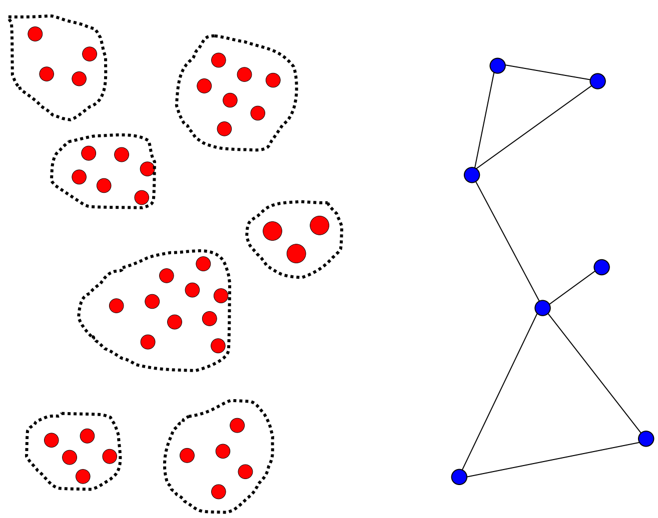
- Create a hierarchy
- Similar to a highway system
  - Neighbourhood roads
  - City roads
  - Regional roads

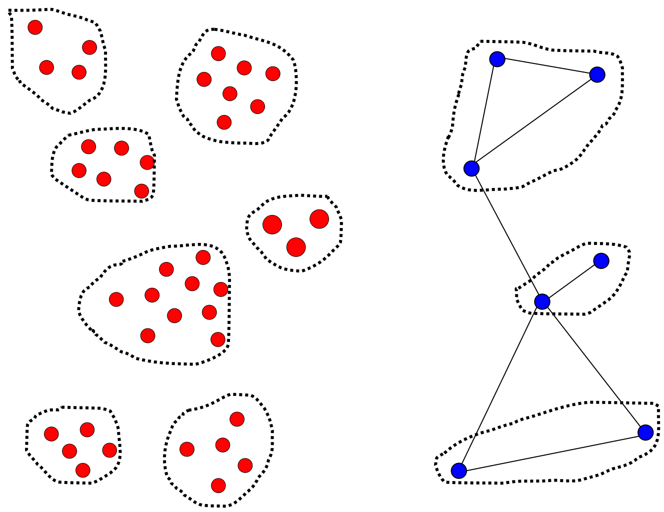
Graph Hierarchy

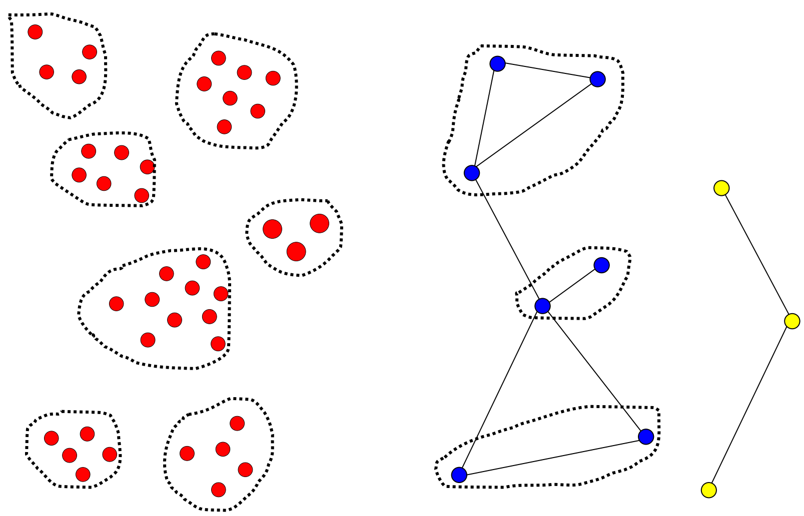
- Designate some paths as “high” level paths (highways)
  - Few connections
  - Few edges
- Create a representation of the space
  - Possibly a sequence of representations G, G^1, G^2…
- Operation
  - Gravitate to a high level representation
  - Find path on high level
  - Convert path back to a low level “real path”

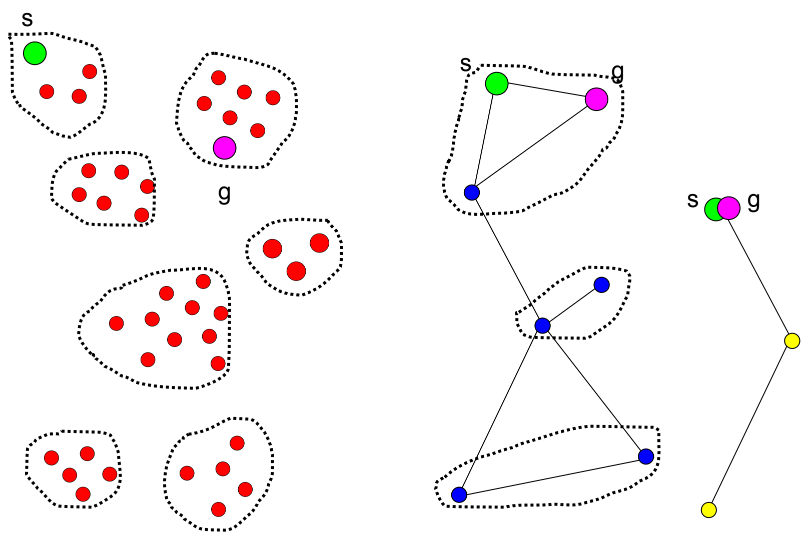












Game Playing Programs

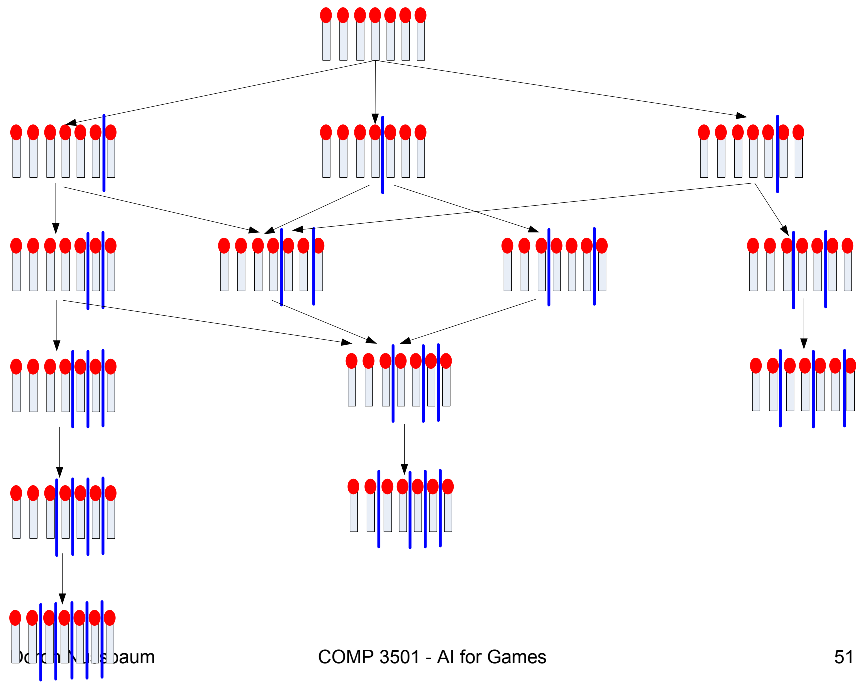
- Game playing programs obey some rules:
  - Visibility – is the board visibile/partial visible
  - Turns – is the game played in order
  - Chance/Luck – what is the nature of a “move”
    - Governed by probability (card game, dice)

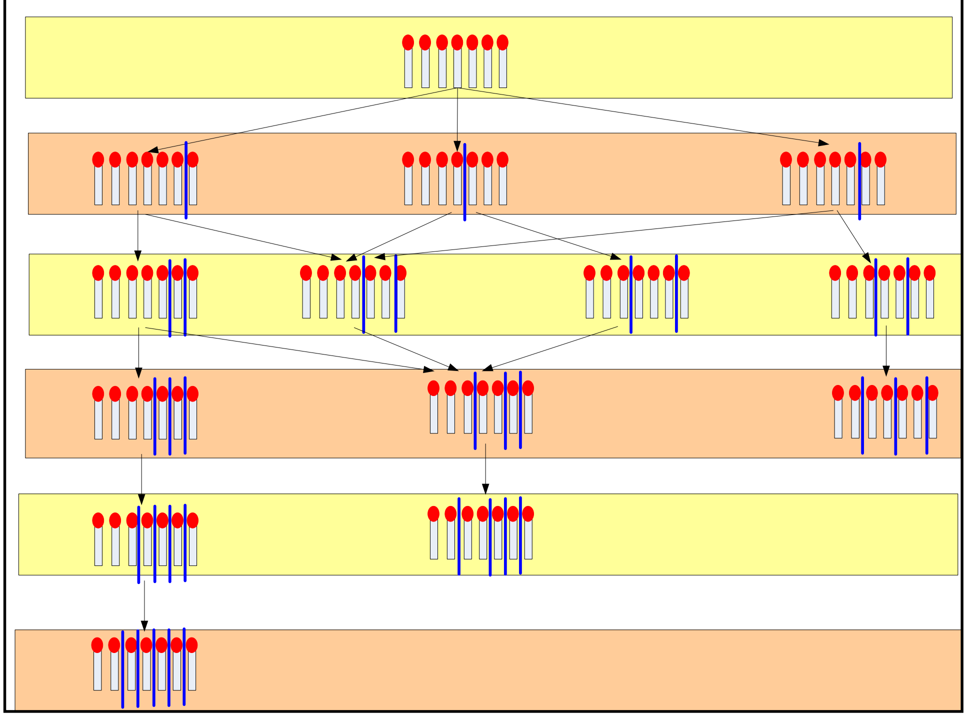
How to play?

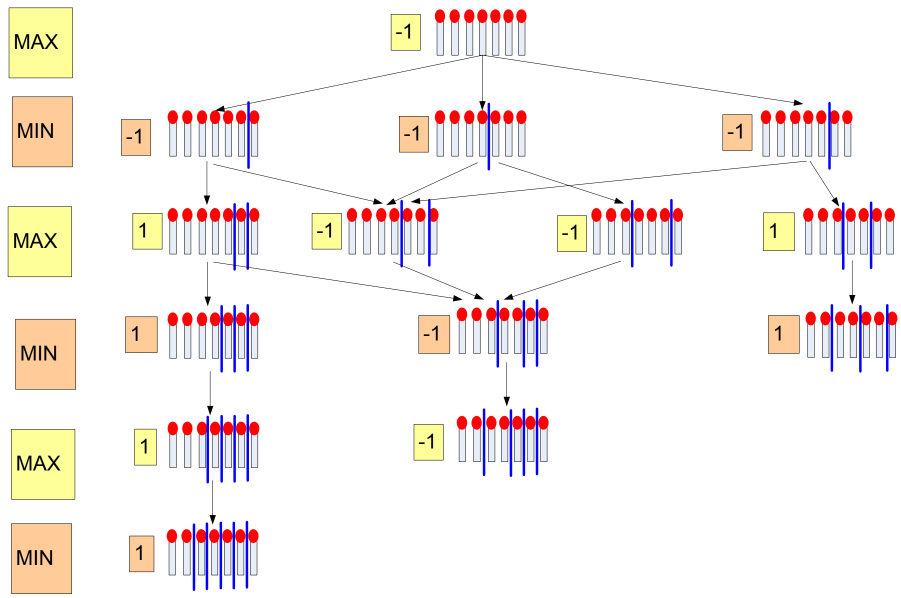
- Deterministic game
- Create a tree with all possible moves
- Search the tree for the best move
  - DFS
- Assuming that each player plays for the best move then each try to follow a path that guarantees a “victory”

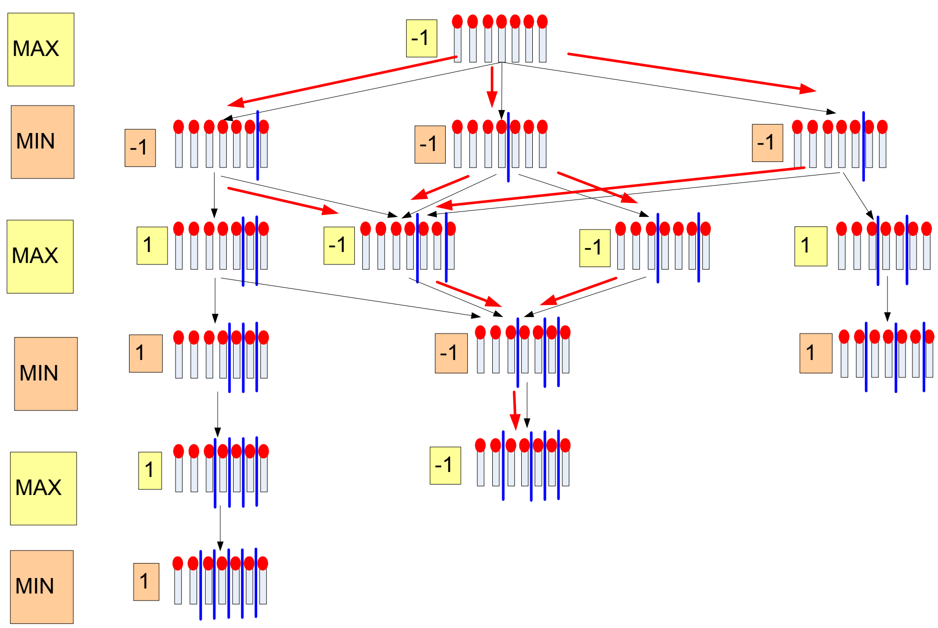
Division Nim Game

- Given a pile of matches
- Each player in his/her turn takes one pile and divides it into two piles of different size
- Game ends when there is non other option to play
- For example given 6 matches one can divide into
  - 1-5
  - 2-4






MAX
MIN
MAX
MIN
MAX
MIN
-1
-1
-1
-1
1
-1
-1
1
1
-1
1
1
-1
1


MAX
MIN
MAX
MIN
MAX
MIN
-1
-1
-1
-1
1
-1
-1
1
1
-1
1
1
-1
1

$$\text{minimax}(v) = \begin{cases} value(v) & \text{if v is a leaf} \\ \min\limits_{u \in \text{children}(v)} \{\text{minimax}(u)\} & \text{if } v \text{ is a MIN node} \\ \max\limits_{u \in \text{children}(v)} \{\text{minimax}(u)\} & \text{if } v \text{ is a MAX node} \end{cases}$$


```
In: node v in the tree
Out: value of v

MiniMax(v)
if (children(v) = ∅) then
    return value(v)
else if label(v) = MIN then
    d ← +∞
    for all u ∈ children(v) do
        d ← min{d,MiniMax(u)}
    endfor
    return d
else
    d ← -∞
    for all u ∈ children(v) do
        d ← max{d,MiniMax(u)}
    endfor
    return d
endif
```

Complexity

- Assuming a branching factor is *b* and a depth of the search tree is *k* then the tree size is

$$1+b+b^2+\cdots+b^k=\frac{1-b^{k+1}}{1-b}=\frac{b^{k+1}-1}{b-1}$$

- Need to traverse the tree (DFS/post order)
- $O(b^k)$

MiniMax Trees

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity $O(b^k)$
- Space complexity $O(b^k)$ (depth-first exploration)

- For chess, b ≈ 35, m ≈100 for "reasonable" games
→ exact solution completely infeasible

Partial Trees

- At times it is not possible to build a full tree
  - Time complexity $O(b^k)$
  - Space complexity $O(b^k)$
- For example - Chess game
  - Branching factor - b ≈ 35,
  - A reasonable game is k ≈ 100
    - Exact solution completely infeasible
    - $35^{100} \approx 2^{500}$

Evaluation Function

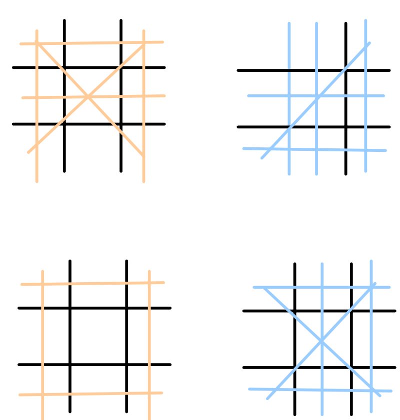
- Solution
  - Build a partial tree
- Question –
  - What value to give a node in the "middle" of the tree?
- Develop an evaluation function to assess the state of the game

Searching

- if search depth limit was reached
  - Evaluate/Compute state of current position
  - Return value
- else
  - if MAX level then
    - apply MiniMax to each child
    - return maximum of results
  - else // MIN level
    - apply MiniMax to each child
    - return minimum of results

Evaluate Tic-Tac-Toe

- Evaluate the number of free moves that are available to win

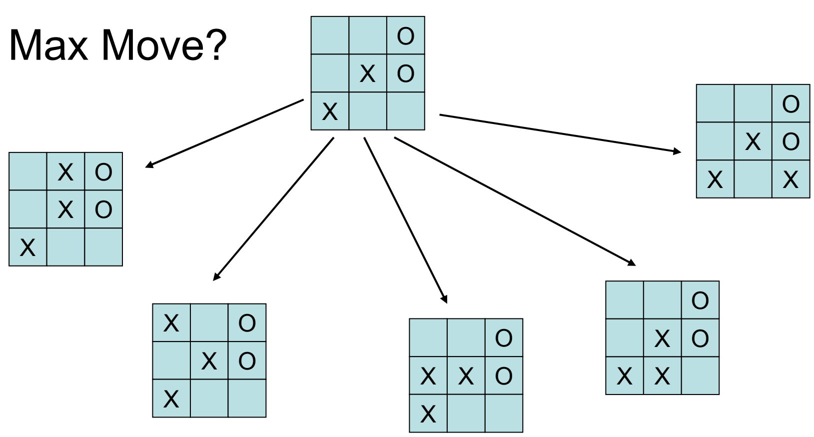


- Eval(p) –
  number of directions for Max - number of directions for Min
- Eval(p) - ∞ if Max wins
- Eval(p) - -∞ if Min wins
- Eval(p) - 6 – 5 = 1

X	O	

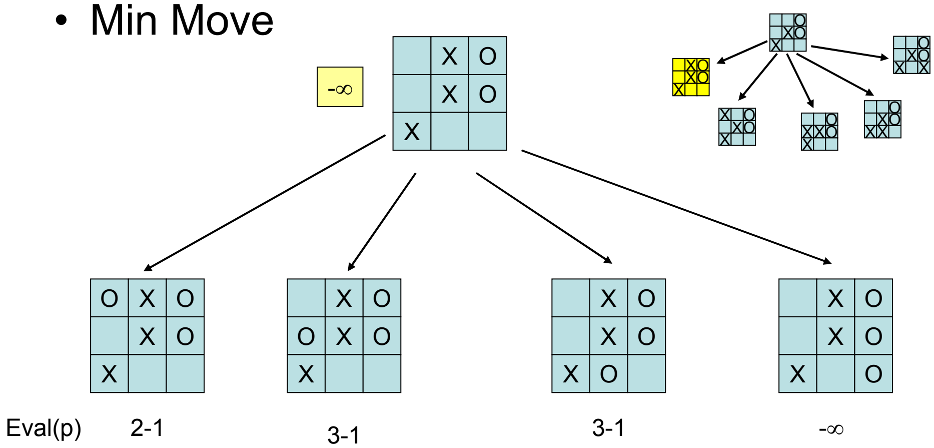
Example – Assume 2 steps ahead

- Max Move?



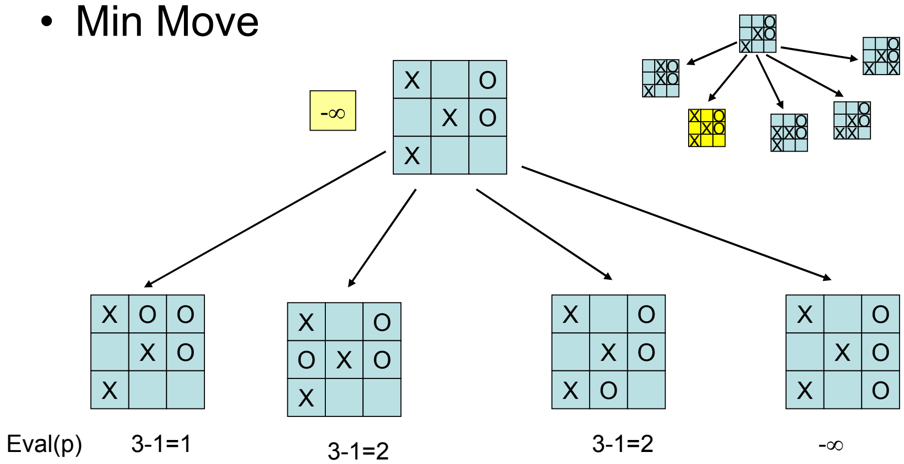
Example – Assume 2 steps ahead

- Min Move



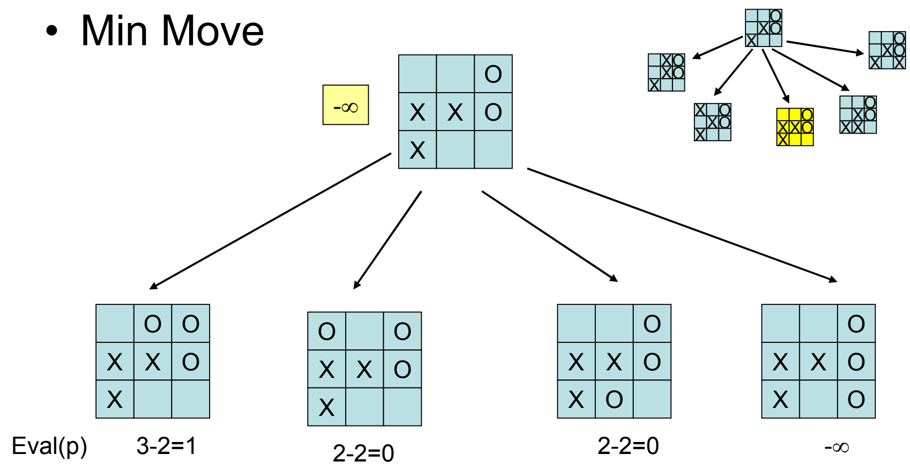
Example – Assume 2 steps ahead

- Min Move



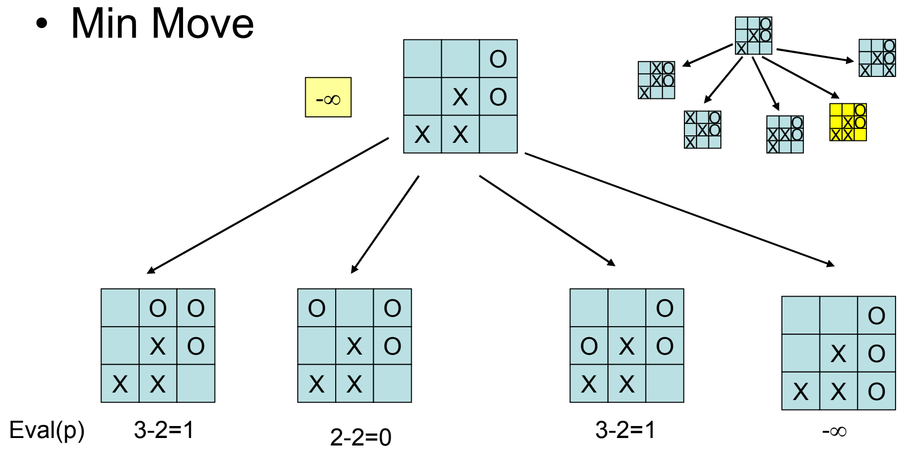
Example – Assume 2 steps ahead

- Min Move



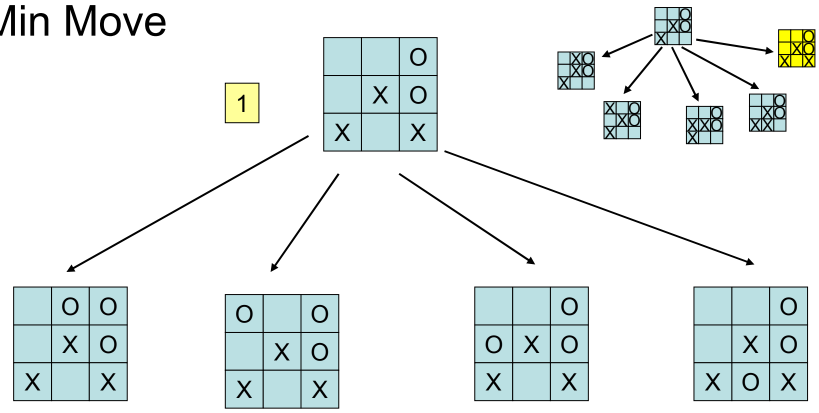
Example – Assume 2 steps ahead

- Min Move



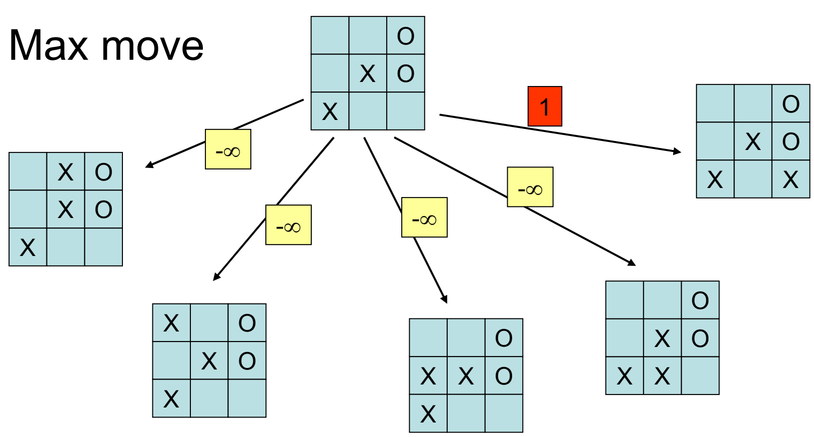
Example – Assume 2 steps ahead

- Min Move



Eval(p) 3-1=2 2-1=1 3-1=2 2-1=1

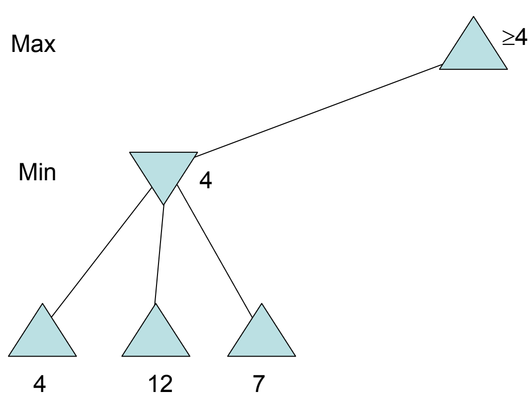
- Max move



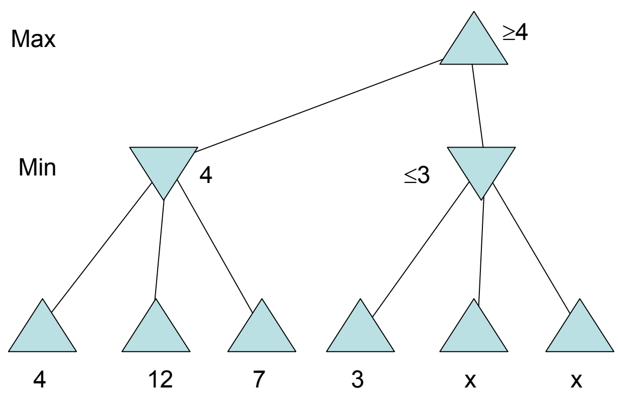
Pruning the tree

- Tree can be quite large
  - Evaluation consumes computing cycles
- Trimming the tree → saves work
- Alpha-beta pruning (α-β pruning)

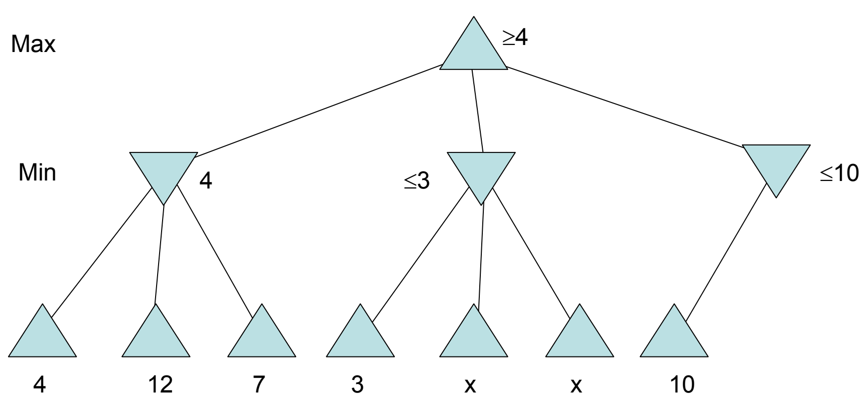
Example



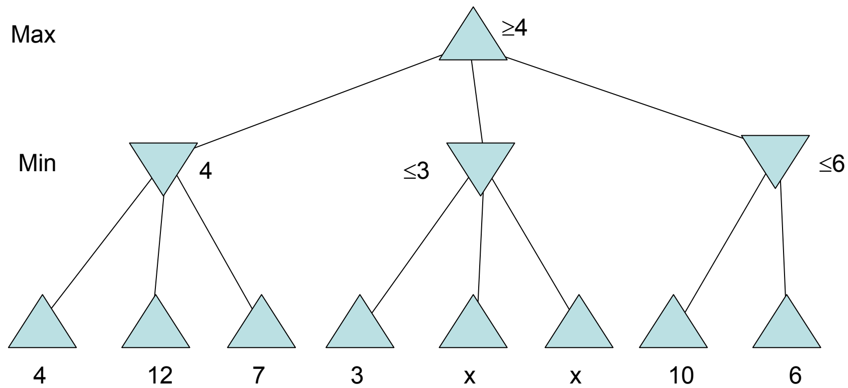
Example



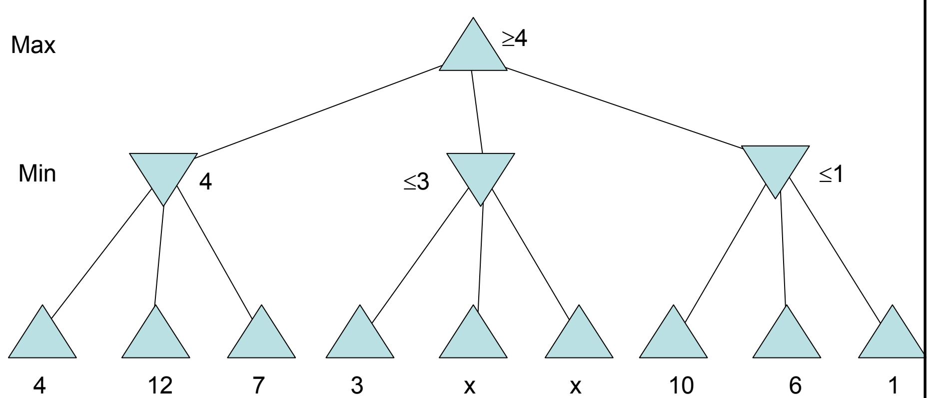
Example



Example



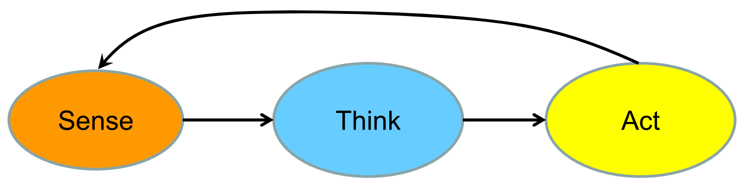
Example



Game Agents

- There are several ways of representing non-player-characters (NPC)
  - Game agents is one option which is easy to relate to
  - In most cases it is an enemy, or an ally
  - Sometimes it is neutral (background NPC)

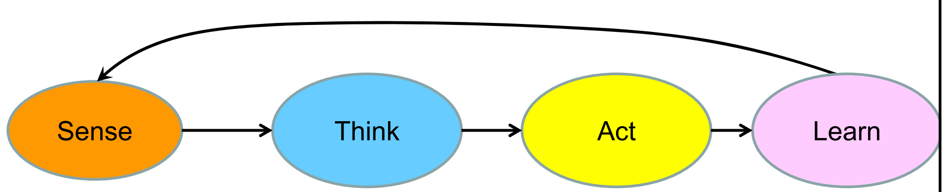
- Agents operate through a sense-think-act cycle



Game Agents

- There are several ways of representing non-player-characters (NPC)
  - Game agents is one option which is easy to relate to
  - In most cases it is an enemy, or an ally
  - Sometimes it is neutral (background NPC)

- Agents operate through a sense-think-act-learn cycle



Sensing

- Agents use sensing to gather world information:
  - Barriers (obstacles)
  - Environment (forest, lake, urban setting)
  - Opponents (position and state)
  - Other objects/agents
- Sensing acts as a stimulator to the agent which may trigger an action

- Agents can query the game for information (cheating)
  - Impose similar constraints on data as player
  - Cannot see through walls
  - Sensing has limitation (hearing, vision)
  - Can only use what they have experienced

Sensing - Vision

- A fundamental sense in gaming
  - An agent may ask to get a list of all enemies
    - Not all enemies are known to agent at that point
    - Time consuming to provide a list of all
    - Not all enemies are relevant at this point
  - Limit knowledge to what is visible
- Visibility is compute intensive and can be complicated
  - Partial visibility of objects
  - Complex terrain

Sensing - Vision

- Game must be able to answer queries fast
  - Is the object within viewing range of agent?
  - Is the objet within viewing angle of agent?
  - Is the object unobstructed?
- Compute a vector to each object
  - Check magnitude
  - Check angle (dot-product) to determine field of view
  - Check if obstcured

Sensing - Hearing

- Player crawls (tip-toe) past an enemy → enemy does not hear
  - Run by enemy → enemy hears
  - Player suddenly shoots a gun at night → agents may hear it and start to move towards sound (e.g., urban war)
- Implement as event driven
  - When player performs action then notify agents within range
    - Obstacles (compute intensive)
  - Variations –
    - Other sounds muffle player
    - Agents are listening → increase range of sound

Sensing - Communication

- Agents can communicate with each other
  - Transferring sensed knowledge to other agents
    - Instant – using a two-way radio
    - Slow – agents runs to other agents and informs
    - Via sound – shouting
- Agents senses data from other agents
  - Vision – signs, colours, actions, not seing
  - Sound
- Reaction time
  - Sensing may take time (e.g., instant reaction to alarm)
  - Build in an artificial delay (e.g., using timer)

Thinking

- Evaluate the gathered information
  - The crux of the AI system
- Can be complex or simple (as required)

- Generally two ways

1. Pre coded expert knowledge
   - Typically handcrafted - a sequence of “if-then” rules with added randomness
2. Search algorithm for optimal/best solution
   1. MinMax trees

Thinking

- Expert Knowledge (Finite State Machine, Decision Trees)
  - Appealing
    - Simple to create
    - Natural (can think about own behaviour)
      - If “I am tired” then “find a place to sleep”
    - Embodies common sense and knowledge of domain
      - Is enemy weaker than me? → attack
      - Is enemy stronger than me? → run away and/or seek help
  - Often not scalable
    - Can be complex with many rules
    - Changing one rule may affect others
  - Still sufficient as most agents address a limited domain
    - carry out very limited functionality

Thinking

- Search
  - Look ahead and see what to do next
    - Path finding (A*)
    - Next move (MinMax)
  - Works well with existing knowledge
    - Visible obstacles, current location
  - Machine learning
    - Evaluate past actions → use for future actions
    - Techniques are improving but
      - Compute intensive
      - Domain specific

Acting

- Sensing and thinking are invisible to player
  - No matter how sophisticated the these two steps the player is oblivious to them
- Acting is visible to user
  - Game must demonstrate its sophistication via realizable actions
- Common actions
  - Pickup weapon
  - Fire
  - Communicate with player
  - animation

Acting

- Learning and remembering
  - May not be important when agent is short lived
  - If agent is a live for more than 30s can be helpful
  - If player always attacks from left then shield to the left
- Implementation does not have to be complex
  - Gather stats on player behaviour (direction of attack)
  - Information can fade (limited capacity for past)
  - Simple things such as using last player’s location to start a search
  - Can be done at the game level –
    - Too many agents are killed in an area → create a smell, colour the area in blood, leave part of agents on the ground (armoury, part of weapons)

Acting

- Making agents stupid
  - Many cases agents can be superior and dominate the player
    - Agent always makes headshot or takes cover
    - Agents runs faster
    - Can be considered as cheating (inside trader)
  - Dumb the agent by giving “human” traits
    - Less accuracy
    - Longer reaction time
    - Engaging the player one at a time
    - Make mistakes
    - Make unnecessary moves
    - Run out of ammunition at critical time → force to retreat

Acting

- Agent Cheating
  - Ideally agents should not be superior by having an unfair advantage
    - Better attributes
    - More attributes
    - Inside knowledge
  - Agents may resort to “cheating” at a high level in order to challenge the player.
    - Player should be aware of it → player will be ready
    - May challenge the player
    - Compute intensive reasons
    - Development time
  - Let the player know upfront