Promising AI Techniques (3 of 3)

- *Genetic algorithms*
  - Search and optimize based on evolutionary principles
  - Good when "right" answer not well-understood
  - E.g. – may not know best combination of AI settings. Use GA to try out
  - Often expensive, so do offline
- *N-Gram statistical prediction*
  - Predict next value in sequence (e.g.- 1818180181 … next will probably be 8)
  - Search backward n values (usually 2 or 3)
  - Example
    - Street fighting (punch, kick, low punch…)
    - Player does low kick and then low punch. What is next?
    - Uppercut 10 times (50%), low punch (7 times, 35%), sideswipe (3 times, 15%)
    - Can predict uppercut or, proportionally pick next (e.g.- roll dice)

Outline

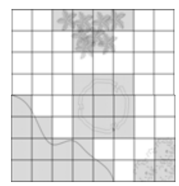
- Introduction (done)
- Common AI Techniques (done)
- Promising AI Techniques (done)
- Pathfinding (A*) (next)
- Finite State Machines
- Summary

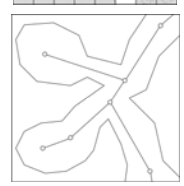
Pathfinding

- Often seems obvious and natural in real life
  - E.g. Get from point A to B → go around lake
- For a computer controlled player, may be difficult
  - E.g. Going from A to B go through enemy base
- Want to pick "best" path
- Need to do it in real-time
- Why can't we just figure it out ahead of time (i.e. before the game starts)?

Representing the Space

- System needs to understand the level
  - But not full information, only relevant information (e.g. is it passable, not water vs. lava vs. tar…)
- Common representations
  - 2d Grid
    - Each cell passable or impassible
    - Neighbors automatic via indices (8)
  - Waypoint graph
    - Connect passable points
    - Neighbors flexible (but needs to be stored)
    - Good for arbitrary terrain (e.g. 3d)





Finding a Path

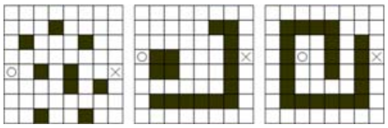
- Path – a list of cells, points or nodes that agent must traverse to get to from start to goal
  - Some are better than others → measure of *quality*
- Algorithms that guarantee path called *complete*
- Some algorithms guarantee *optimal* path
- Others find no path (under some situations)

Random Trace (Simple Algorithm)

- Agent moves towards goal
- If goal reached, then done
- If obstacle
  - Trace around obstacle clockwise or counterclockwise (pick randomly) until free path towards goal
- Repeat procedure until goal reached
- (Humans often do this in mazes)

Random Trace (continued)

- How will Random Trace do on the following maps?



- Not a *complete* algorithm
- Found paths are unlikely to be optimal
- Consumes very little memory

Understanding A*

- To understand A*
- – Combines breadth-first, best-first, and Dijkstra
- These algorithms use nodes to represent candidate paths
- m_pParent used to chain nodes sequentially together to represent path
  - List of absolute coordinates, instead of relative directions

```
class PlannerNode {
public:
PlannerNode *m_pParent;
int m_cellX, m_cellY;
...
};
```

Breadth-First (1 of 2)

Overview

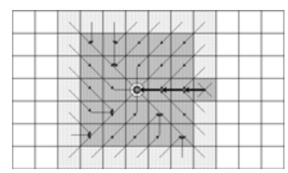
- Use two lists: *open* and *closed*
- Open list keeps track of promising nodes
- Closed list keeps nodes that are visited, but don't correspond to goal
- When node examined from open list
  - Take off
  - Check to see if reached goal
- If not reach goal
  - Create additional nodes
  - Place on closed list

Overall Structure

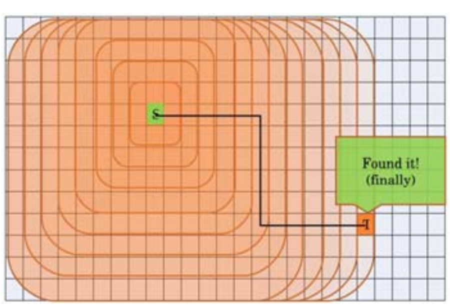
- Create start point node – push onto open list
- While open list is not empty
  - A. Pop node from open list (call it currentNode)
  - B. If currentNode corresponds to goal done
  - C. Create new nodes (successors nodes) for cells around currentNode and push them onto open list
  - D. Put currentNode onto closed list

Breadth-First (2 of 2)

- Search from center
- Goal was 'X'
- Open list → light grey
  - Have not been processed
- Closed list → dark grey
  - Not goal and have been processed
- Arrows represent parent pointers
- Path appears in **bold**



Breadth-First in Action

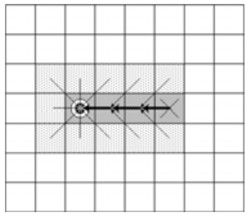


Breadth-First Characteristics

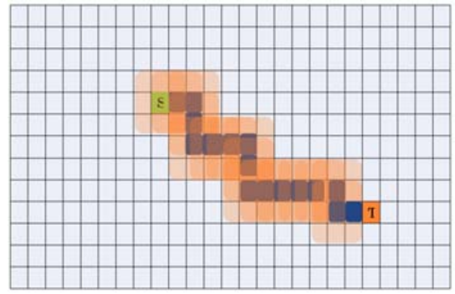
- Exhaustive search
- – Systematic, but not clever
- Consumes substantial amount of CPU and memory
- Guarantees to find paths that have fewest number of nodes in them
  - *Complete* algorithm
  - But not necessarily shortest distance!

Best-First (1 of 2)

- Uses problem specific knowledge to speed up search process
  - Not an exhaustive search, but a *heuristic search*
- Head straight for goal
- Computes distance of every node to goal
- Algorithm same as breadth first
  - But use distance as priority value
  - Use distance to pick next node from open list

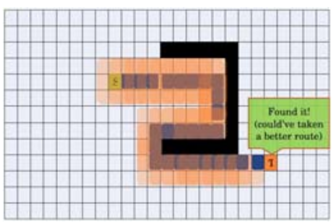


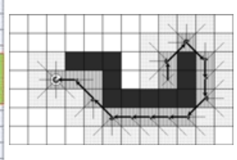
Best-First in Action



Looks pretty good! But perfect?

Best-First (2 of 2)





(Sub-optimal paths)

Best-First Characteristics

- Heuristic search
- Uses fewer resources than breadth-first
- On average, much faster than breadth-first search
- Tends to find good paths
  - No guarantee to find most optimal path
- *Complete* algorithm

Dijkstra's Algorithm

- Disregards distance to goal
  - Keeps track of cost of every path
  - Unlike best-first, no heuristic guessing
- Computes accumulated cost paid to reach a node from start
  - Uses cost (called "given cost") as priority value to determine next node in open list
- Use of cost allows it to handle other terrain
  - E.g. mud that "slows" or "downhill"

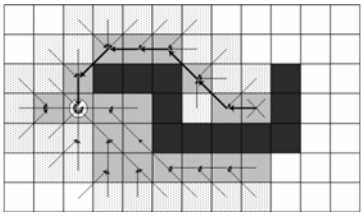
Dijkstra Characteristics

- Exhaustive search
- At least as resource intensive as Breadth-First
- Always finds the optimal path
  - No algorithm can do better
- *Complete* algorithm

A*

- Use best of Djikstra and Best-First
- Both heuristic cost (estimate) and given cost (actual) to pick next node from open list

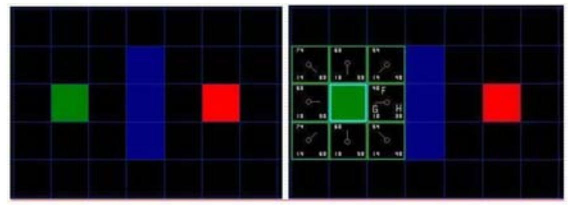
Final Cost = Given Cost + (Heuristic Cost * Heuristic Weight)



(Avoids Best-First trap!)

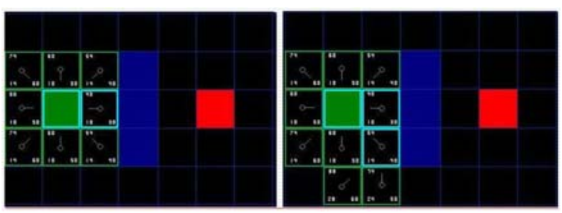
A* Internals (1 of 3)

- Green: start
- Red: goal
- Blue: barrier
- G: 10 for ver/horiz, 14 for diagonal
- H: distance * 10



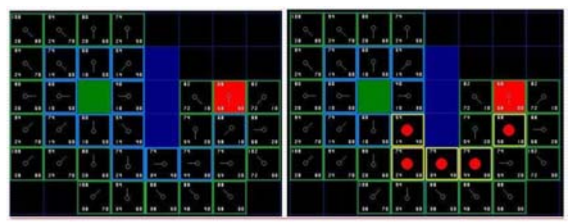
A* Internals (2 of 3)

- Now check for the low F value in OPEN
  - In this case NE = SE = 54, so choose SE
- Going directly to SE is cheaper than E->SE
  - Leave start as the parent of SE, and iterate

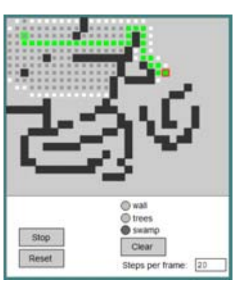


A* Internals (3 of 3)

- Keep iterating until reach goal and OPEN is empty
- Follow parent links to get short path



A* Demo



http://www.antimodal.com/astar/

A* Characterisitics

- Heuristic search
  - Weight can control 0 then like Dijkstra, large then like best-first
- On average, uses fewer resources than Dijkstra and Breadth-First
- "Good" heuristic guarantees it will find the most optimal path
  - "Good" as long as doesn't overestimate actual cost
  - For maps, good is "as a bird flies" distance (best-case)
- *Complete* algorithm