Intelligent Game Agents

CMSC425.01 Spring 2019
Administrivia

• Exam being graded …

• Project 2b concepts out, write up soon (add animations to 2a)
Today’s questions

How, and why, should you make game agents intelligent
Thoughts on game AI

• What is game AI for?
Thoughts on game AI

• What is game AI for?
  
  Major game opponents

  Individual game units

  Richer world of NPCs
Discussion question

• Do you want your opponents to be

Game AI, made better

Humans, through better networked games
What does AI mean here?

• How code Starcraft
  Hive mind?
  Individual zerg?
What does AI mean here?

- How code Starcraft
  - Hive mind?
  - Hard coded?
  - Not adaptive
  - Individual zerg?
  - A* plus "attack"

- Observation:
  - Not that intelligent but powerful gameplay
Review: examples

- A*

- Minowski sum of obstacles
A*  

- Pick next node to expand based on sum of distance so far and heuristic

\[ f(u) = d[u] + h(u) = d[u] + \text{dist}(u, t) \]
A-Star(G, s, t) {
    foreach (node u) { // initialize
        d[u] = +infinity; mark u undiscovered
    }
    d[s] = 0; mark s discovered // distance to source is 0
    repeat forever { // go until finding t
        let u be the discovered node that minimizes d[u] + dist(u,t)
        if (u == t) return d[t] // arrived at the destination
        else {
            for (each unfinished node v adjacent to u) {
                d[v] = min(d[v], d[u] + w(u,v)) // update d[v]
                mark v discovered
            }
            mark u finished // we’re done with u
        }
    }
}
A* Search – Each entry is $d[u] : f(u)$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(u)$</td>
<td>15</td>
<td>13</td>
<td>15</td>
<td>17</td>
<td>12</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Init</td>
<td>0:15</td>
<td>∞:13</td>
<td>∞:15</td>
<td>∞:17</td>
<td>∞:12</td>
<td>∞:10</td>
<td>∞:9</td>
<td>∞:8</td>
<td>∞:5</td>
<td>∞:0</td>
</tr>
<tr>
<td>1: $s$</td>
<td>0</td>
<td>8:13</td>
<td>–</td>
<td>2:17</td>
<td>3:12</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>2: $d$</td>
<td>↓</td>
<td>8:13</td>
<td>–</td>
<td>2:17</td>
<td>3</td>
<td>5:10</td>
<td>6:9</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>3: $e$</td>
<td>↓</td>
<td>8:13</td>
<td>–</td>
<td>2:17</td>
<td>↓</td>
<td>5</td>
<td>6:9</td>
<td>7:8</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>4: $f$</td>
<td>↓</td>
<td>8:13</td>
<td>–</td>
<td>2:17</td>
<td>↓</td>
<td>6</td>
<td>7:8</td>
<td>–</td>
<td>15:0</td>
<td>–</td>
</tr>
<tr>
<td>5: $t$</td>
<td>↓</td>
<td>8:13</td>
<td>–</td>
<td>2:17</td>
<td>↓</td>
<td>7:8</td>
<td>–</td>
<td>–</td>
<td>15</td>
<td>–</td>
</tr>
<tr>
<td>Final</td>
<td>0</td>
<td>8</td>
<td>∞</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>∞</td>
<td>15</td>
</tr>
</tbody>
</table>
Good heuristics

• For A* to compute correctly the heuristic \( h(u) \) must be:

  • Admissible: \( h(u) \) never overestimates the graph distance from node \( u \) to goal \( t \)

  • Consistent: \( h(u') \leq \delta(u',u'') + h(u'') \)

• *Goldilocks* – heuristics must be not too high, not too low
$H(A) = 11$

$H(B) = 7$

$H(D) = 8$

$H(E) = 50$

$H(G) = 8$

$H(H) = 50$
\[
\begin{align*}
H(A) &= 0 \\
d(A+H(A)) &= 7 \\
H(B) &= 0 \\
H(G) &= 0 \\
H(E) &= 2 \\
d(E)+H(E) &= 10 \\
H(D) &= 8 \\
d(D)+H(D) &= 10 \\
H(T) &= 8 \\
\end{align*}
\]
Algorithm: Computing Minowski sum

• Input: two polygons
• Output: polygon of M-sum

• Algorithm:
  • Take each edge in CCW direction
  • Sort by angle
  • Combine
Finding paths in polygonal configuration space

• Version 1: Navmesh
• Others?

• Version 8: Rapidly-expanded Random Trees (RRTs)
Decision making

• Reactive decision making: Respond
  • Decision trees
  • Finite State Machines
  • Behavior trees

• Proactive decision making: Plan
  • Not this semester
  • Can use variation of A* on space of operators on world state
  • Robot planning
  • Done by game designers implicitly
Decision trees

• Structured if

• Can
  • Randomize
  • Share subtrees
  • Have branching factor > 2
Finite State Machine

• Organize behaviors in graph
• Set transitions on state of game

(a)

(b)

<table>
<thead>
<tr>
<th></th>
<th>Small enemy</th>
<th>Big enemy</th>
<th>Losing</th>
<th>Escaped</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fight</td>
<td></td>
<td></td>
<td>Run</td>
<td></td>
</tr>
<tr>
<td>Run</td>
<td></td>
<td></td>
<td></td>
<td>Guard</td>
</tr>
<tr>
<td>Losing the fight</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Escaped</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Finite State Machine

• State can be
  • Behavior - character actions
  • Emotional state - predisposition (confidence/fear, anger, health, energy)

(a)

On Guard
See small enemy
Fight
See big enemy
Losing the fight
Run Away
Escaped
(Start state)

(b)

<table>
<thead>
<tr>
<th></th>
<th>Small enemy</th>
<th>Big enemy</th>
<th>Losing</th>
<th>Escaped</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guard</td>
<td>Fight</td>
<td>Run</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Fight</td>
<td>–</td>
<td>–</td>
<td>Run</td>
<td>–</td>
</tr>
<tr>
<td>Run</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Guard</td>
</tr>
</tbody>
</table>
Controlling FSM complexity

• State with branches to many others
Controlling FSM complexity

• Hierarchical FSM (StateCharts)
  • Superstates + generalized transitions
• Part of UML (if you don’t know, look up ...)

![Statechart Diagram]

- On Guard
- Fight
- Run Away
- Guarding
- Small enemy
- Big enemy
- Losing
- Escaped
- Hungry
- Full
- Get food
- (Start)
Tuning FSMs

- Variations on one character template
- EG, Orcs in LOTR
- Massive Software
  - Each agent owns character profile
  - Randomize when populating game
Behavior trees

• Lightweight way to design action plans

• Plan
  • Sequence of actions
    1) Go to door
    2) Use key open door
    3) Go through door
  • With preconditions
    2)* Must have key
Behavior trees

• Lightweight way to design action plans
• Plan
  • Sequence of actions
    1) Go to door
    2) Use key open door
    3) Go through door
  • With preconditions
    2)* Must have key
Sequential AND node

\[
A \quad B \quad C \quad D
\]

evaluate sequentially until failure

(a)

fail!

(b)
Sequential OR node

(a) Test sequentially until success

(b) Success!
Goal: Move into room

- Door wide open?
- Move into room
- Move to door
- Move into room
  - Door unlocked?
  - Open door
  - Break down door
- Open door