Artificial Intelligence

CIS 399 - Intro to Game Programming
Chris GauthierDickey
What is AI?

• How can we answer this question?
  – Quite philosophical and treads on people’s belief systems

• In terms of a game, it encompasses everything from path-finding, to reasoning about the virtual world.
Established Techniques

• The two most common techniques are:
  – Cheating: we give AI more information than the human has in order to compete with the human’s reasoning ability
  – Finite State Machines: the most basic building block of most game AI (you should take Automata Theory)
Cheating and its Problems

• Cheating is a simple solution to a complex problem.
  – For example, when a space ship is looking for enemies, we might not restrict the search to those on its radar.

• Players can recognize cheating though and may get discouraged.
Finite State Machines

• Modeled by a set of states which the AI may be in at any time.
  – For example: chase_enemy, flee, patrol

• Simple to implement and debug
• Each state may have a complex set of actions and reactions
Chasing and Evading

• The basic chase algorithm:

```java
if (predator.X > prey.X)
    predator.X--;  
else if (predator.X < prey.X)
    predator.X++;  

if (predator.Y > prey.Y)
    predator.Y--;  
else if (predator.Y < prey.Y)
    predator.Y++;  
```
Basic Evade Algorithm

• How can we turn our chase algorithm into an evade algorithm?

• Just reverse predator.X and predator.Y with prey.X and prey.Y!

• Obviously, this is very basic.
Interception

- Calculate closing velocity: \( V_r = V_{prey} - V_{predator} \).
- Calculate range to close: \( S_r = S_{prey} - S_{predator} \).
- Calculate time to close: \( t_c = \frac{|S_r|}{|V_r|} \).
- Calculate interception point: \( S_t = S_{prey} + (V_{prey}) \times t_c \).
- Steer towards interception point!
Planned Paths

• Sometimes we need to follow a planned path
  – patrols
  – race cars driving around a track

• We simply store an array that has the series of steps needed to achieve the path
  – In a continuous world, this means storing steering vectors, thrust values, and time for each operation
Potential Functions

• Potential functions can be used to customize how a predator chases a prey.

• The Lenard-Jones potential function describes how molecules attract and repel each other.
  
  \[ U = -\frac{A}{d^n} + \frac{B}{d^m} \]
  
  – \(-\frac{A}{d^n}\) is the attractive force
  – \(+\frac{B}{d^m}\) is the repulsive force
Potential Functions continued...

• When $U$ is negative, we are attracted and when $U$ is positive, we are repulsed. $A$ and $B$ changes the strength of these forces. $n$ and $m$ change the attenuation.

• Setting $B$ to 0 is like designing a homing missile that will ram the object.

• Setting $A$ to 0 is like telling the predator to avoid the prey at all cost.
More Potential

• How do we use U?
• We find the vector between predator and prey and scale it by U.
• We then find the point at the end of this vector when added to the predator’s position.
• We steer in that direction.
Multiple Potentials

• We can take all of the potential vectors and add them together to figure out where to maneuver to.

• In our game, potential vectors from meteors have $A = 0$, so our ships steer clear of meteors. We sum all the vectors, including those to prey, and we then steer in that direction.
Flocking

• In the classic sense, we refer to Craig Reynolds’s boids, where each unit follows three rules:
  – *Cohesion*: Each unit steers towards the average position of its neighbors
  – *Alignment*: Each unit aligns itself along the average heading of its neighbors
  – *Separation*: Each unit steers to avoid colliding with neighbors
More Flocking

• How do we know neighbors?
  – Each unit has a visibility radius and visibility arc
  – Units that fall in that volume are neighbors
  – Smaller arcs lead to single file flocking and easily broken flocks
  – 360 degree arcs look unnatural
Obstacles and Leaders

• We can use ‘feelers’ which detect obstacles and steer away from them
• We can also assign leaders towards which flocks follow
A* Path Finding

• Certainly, in a simple world, we can use chase algorithms and random movement to discover players
• We can also drop bread-crumbs from the player and let predators discover them
• For more complex worlds with obstacles, A* is optimal (again, overkill though on simple worlds)
What is A*

Add starting node to list
while open list is not empty
{
    current node = node from open list with lowest cost
    if current node = goal then
        path complete
    else
        move current node to closed list
        examine each node adjacent to the current node
        for each adjacent node
        ...
    
}
A* Algorithm Continued

if it isn’t on the open list and
it isn’t on the closed list and
it isn’t an obstacle then
move it to the open list and calculate cost

- Cost calculated by number of steps from starting node + heuristic value
  - heuristic is often line-of-sight hops to goal
  - We can add to the cost by terrain costs or other influences such as the number of kills the player has achieved over a particular node