CSc 165 Lecture Note Slides

Artificial Intelligence & NPCs In Games

Dr. John Clevenger
Computer Science Dept.
CSUS

Overview

- Structure of AI Systems
- Overview of Basic AI Techniques
  - State Machines, Rule Systems, State-Space Searching, Genetic Algorithms, Neural Networks
- Agent-based AI
  - Tracking/Chasing/Evading/Patrolling/Hiding
- Tactical AI
  - Path Finding
  - Group Dynamics (Flocking and Boids)
  - Scenario Analysis

Two Fundamental Types of "AI"

- "Agent"
  - Enemies
  - Non-Player Characters (NPCs)
  - "Extras"
- "Controllers"
  - Provide tactical reasoning, group dynamics
  - Not a "character" per se

Basic Structure of an "AI"

- Sensor (input) system
- Memory
- Reasoning/Analysis Core
- Action/Output System

Sensor (Input) Functions

- Agent-based:
  - Where is the player?
  - Where is he looking?
  - What does the world look like (e.g. buildings)?
- Tactical:
  - What is the balance of power in different regions?
  - What resources do I have?
  - What types of resources exist?
  - How is the world organized?

Memory Functions

- Some data are simple to store:
  - “state” (standing, walking, running…), location, resource inventory, etc.
- Other data are more complex/abstract:
  - ‘balance of power’
  - previously-applied strategies

Reasoning/Analysis Core

- Uses sensory input & memory to make decisions about "what to do next"
- Can use a variety of techniques:
  - State Machines
  - Rule Systems
  - State Space Searches
  - Genetic Algorithms
  - Neural Networks
- Can occupy as much as 50% of the game update function!

John Clevenger
CSc Dept, CSUS

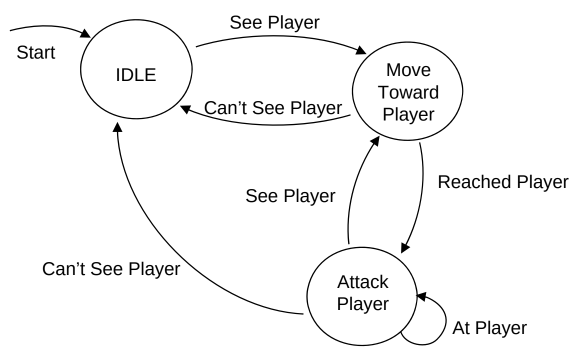
Action/Output System

- Agent:
  - controls actions (animation, movement, etc.)
- Controller:
  - forwards action instructions to group members
- Provides both individual characters and the overall game with "personality"

John Clevenger
CSc Dept, CSUS

Basic AI Techniques

- Finite State Machines (FSMs)



Basic AI Techniques (cont.)

- Rule Systems
  - Define system behavior using *rules*:

    condition1 → action1
    condition2 → action2
    …
  - Rule sets are parsed looking for first true condition
  - Sequences of rules imply *priority*
  - Behavior sometimes easier to specify than with FSMs
  - The basis of so-called "Expert Systems"

Basic AI Techniques (cont.)

- Example Rule System:

```
next to enemy → fight
close to enemy & stronger → chase
close to enemy → flee
have pending command from leader →
          execute command
teammate fighting & have gun →
          shoot at enemy
stay still
```

After Core Techniques and Algorithms in Game Programming, Daniel Sanchez-Crespo Dalmau

Basic AI Techniques (cont.)

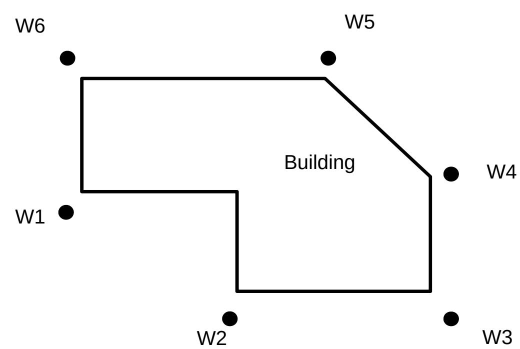
- **Other Techniques:**
  - **State Space Searching**
    - Propagate each possible "move" into the future and evaluate the outcome
  - **Genetic Algorithms**
    - Develop agents with new characteristics by merging the best properties of existing agents
  - **Neural Networks**
    - Define a combinatorial network mapping system characteristics (inputs) to actions (outputs)
    - Network can be "trained" (improved) via feedback

Agent Actions

- Tracking
- Chasing
- Evasion
- Patrolling
- Hiding
- Attacking
- Shooting

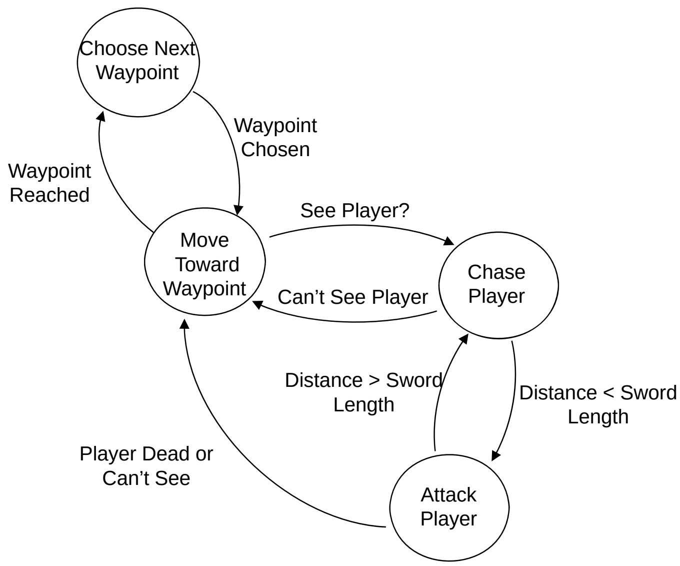
Creating Agent Behavior

- Example: Patrolling/Chasing/Attacking



Waypoints Around A Building

Agent Control Using FSM's



After a diagram in Core Techniques and Algorithms in Game Programming, Daniel Sanchez-Crespo Dalmau

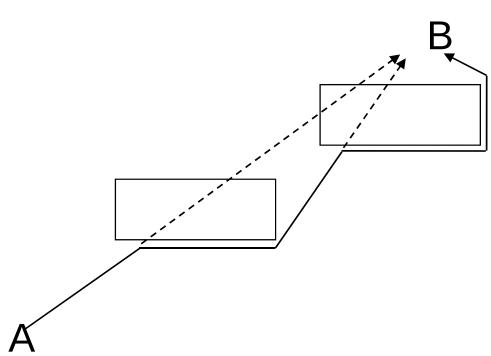
Tactical AI

- **Tactic: a sequence of operations designed to achieve a goal**
- **Tactics Involve:**
  - Initial state
  - Goal state
  - Plan for moving from one state to the other

Path Finding

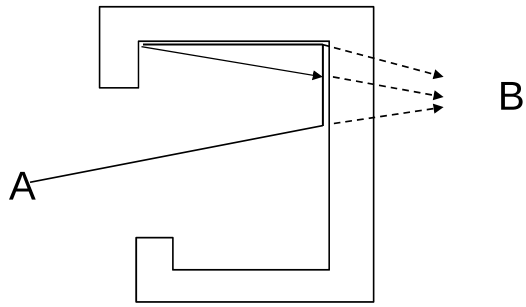
- Initial State: location “A”
- Goal State: location “B”
- Need: plan to move from “A” to “B”
  - How to get from A to B
  - How to move around any obstacles on the way
  - Find *shortest* path
  - Find the path *quickly*

Crash and Turn



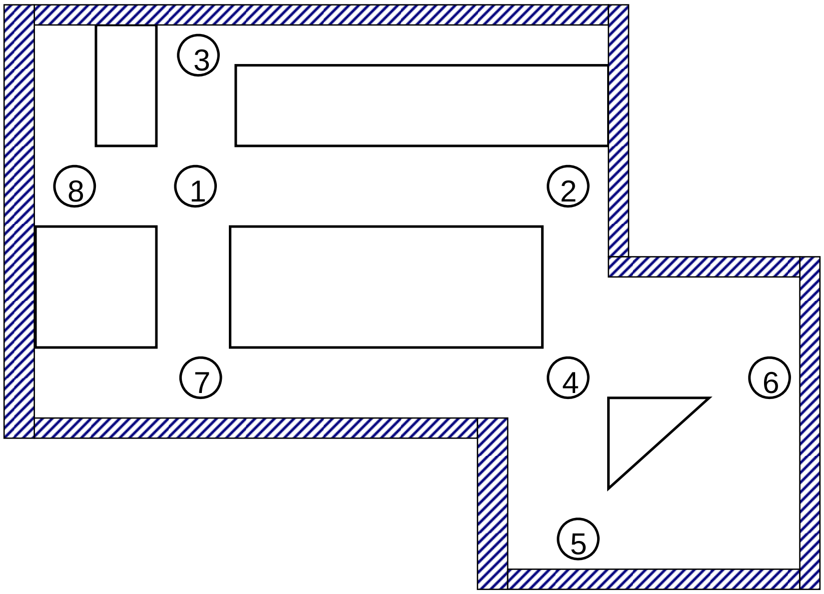

```
compute line of sight to target;
while (not at target){
  if (not blocked)  {
    move along line of sight;
  }
  else  {
    turn facing parallel to
         blocker;
    while (adjacent to blocker){
      move forward;
    }
    compute line of sight to
         target;
  }
}
```

Crash and Turn Failures:



John Clevenger
CSc Dept, CSUS

Map Graphs



John Clevenger
CSc Dept, CSUS

Dijkstra's Algorithm

```
computePaths (Graph g, weights w, startNode s) {
   //initialize
   for each node in g {
      distance[node] = MAX;
      previousNode = NULL;
   }
   Distance[startNode] = 0;
   Visited = empty set;
   Pending = all nodes of Graph g;

   while (Pending != empty) {
      u = Pending.removeHead();
      Visited.add(u);

      for each node v connected to u {
         if ( distance[v] > distance[u] + w(u,v) ) {
            distance[v] = distance[u] + w(u,v) ;
            previousNode[v] = u;
         }
      }
   }
}
```


Group Tactics

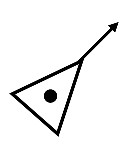
- Group: a collection of *units* (individual NPC's)
- Difficult to coordinate separate AIs
- Need higher level tactical control
- Examples:
  - Flocks of birds
  - Schools of fish
  - Crowds of people
  - Soldiers
  - Monsters …

BOIDS

- A model for "Artificial Life"
  - Craig Reynolds, Computer Graphics V21,4, SIGGRAPH 1987 (pp. 25-34)
- Defines specific rules for "flocking behavior"
  - Can be used for simple animal group behavior
  - Also works for group dynamics
    - Large-scale strategy games
    - Traffic on highways
    - Pedestrian/crowd behavior

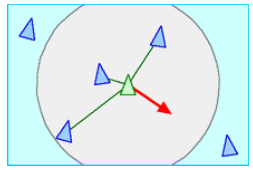
Boid Premise

- Apparently complex group behavior can be modeled by
  - A relatively simple set of rules
  - A simple entity (a "boid") with simple attributes:
    - Speed and heading (velocity)
    - Position

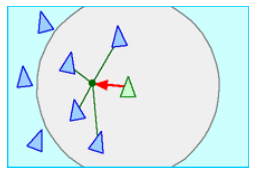


Basic Boid Rules

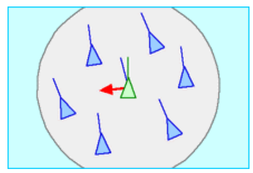
- Separation



- Cohesion



- Alignment



(After Reynolds)

Implementing Boid Rules

```
initializeBoids();

update (time) {
   Vector v1, v2, v3;

   for (each boid b)  {
     v1 = getCohesionVector(b);
     v2 = getSeparationVector(b);
     v3 = getAlignmentVector(b);

     b.velocity = b.velocity + v1 + v2 + v3 ; //vector addition
     b.position = b.position + b.velocity ;  //point+vector→ point
  }
}
```

Boid
Vector velocity Point position

Boid Cohesion

```
Vector getCohesionVector(Boid b){
    //calculate flock center
    Point center = 0;
    for (each boid n) {
       center = center +  n.position ;
    }
    center = center.divideBy(numBoids);

     //find a vector that moves a “little bit”
     //(e.g. 1%) toward center
    Vector change = (center – b.position) / 100 ;

    return change;
}
```


Boid Separation

```
Vector getSeparationVector(Boid b) {
    Vector change = 0;

    for (each boid n) {
       dist = n.position – b.position ;
       if ( abs (dist) < MIN_DIST) {
           change = change – dist ;
       }
    }

    return change;
}
```

Boid Alignment

```
Vector getAlignmentVector(Boid b) {
    //find flock average direction vector
    Vector avgDir = 0;
    for (each boid n) {
       avgDir = avgDir + (n.velocity);
    }
    avgDir = avgDir / numBoids ;

   //return a small fraction of the difference
  //between this boid & avg
  Vector change = (avgDir – b.velocity)  /  8;
  return change;
}
```

John Clevenger
CSc Dept, CSUS

Additional Boid Rules

- Goal Setting
- Speed Limiting
- Space Bounding
- Perching
- Scattering
- Obstacle Avoidance
- Strong Attraction or Repulsion

John Clevenger
CSc Dept, CSUS

Implementing New Rules

```
update (time) {
   Vector v1, v2, v3, v4, v5….;

   for (each boid b) {
       v1 = getCohesionVector(b);
       v2 = getSeparationVector(b);
       v3 = getAlignmentVector(b);

       v4 = getRule4Vector(b);
       v5 = getRule5Vector(b);
       …

       b.velocity = b.velocity + v1 + v2 + v3 + v4 + v5 + … ;

       b.position = b.position + b.velocity ;
   }
}
```

John Clevenger
CSc Dept, CSUS

Example Implementations

```
Vector getWindVector (Boid b) {
    Vector wind = new Vector (windDir);
    return wind ;
}
```

```
Vector moveTowardGoal (Boid b) {
    Vector goal = new Vector (goalPoint);

    return (goal – b.position) / 100 ;
}
```

John Clevenger
CSc Dept, CSUS

Limit Rules

```
void limitVelocity (Boid b) {
    if (b.velocity  >  MAX_VELOCITY) {
      b.velocity =
         ( b.velocity / magnitude(b.velocity) ) * MAX_VELOCITY;
    }
}
```

```
void boundPosition (Boid b) {
  if (b.position is outside boundary) {
    alter position toward the inward direction ;
  }
}
```

John Clevenger
CSc Dept, CSUS

Applying Limit Rules

```
update(): {
  for (each boid b) {
    …invoke rules

    b.velocity = b.velocity + v1 + v2 + v3 + v4 + v5 + … ;
    limitVelocity(b);

    b.position = b.position + b.velocity ;
    boundPosition(b);
  }
}
```

John Clevenger
CSc Dept, CSUS

"Perching"

In boundPosition:

```
if (position.Y closeTo
        ground) {
  position.Y = ground;
  perching = true ;
  timeRemaining = randNum ;
}
```

```
update(): {
  for (each boid b)     {
    if (b.isPerching()) {
       if (timeRemaining > 0)
          timeRemaining-- ;
       else
          perching = false;
    }

    if ( ! b.isPerching() ) {
       … invoke rules
       … update boid velocity
          & position ;
    }
  }
}
```


More Rule Possibilities

- Scattering
- Obstacle Avoidance
- Strong Attraction
  - (e.g. seek prey)
- Strong Repulsion
  - (e.g. avoid predator)