GAME AI:

The Art of Faking It

Topics

General Gaming Theory

Agents

Finite State Automata

Improvement Algorithms

Natural Language Understanding

Probability and Utility

Planning Algorithms

Team AI

Searching and Shortest Path

Time Constraints

Scheduling

Gaming Theory

[Note: The foils concerning game theory are basically a synopsis of an article first published in *Interactive Fantasy*. It can be found on the Web at *http://www.crossover.com/~costik/nowords.html*]

What is a Game?

A Game Is Not Simply a Puzzle.

A puzzle is static. A game is interactive.

A Game Is Not Simply a Toy.

A toy is interactive. But a game has goals.

A Game Is Not Simply a Story.

A story is linear. A Game is not.

A Game Demands Participation.

Gaming Theory

Necessary Conditions for a Game

Decision Making

A player must make choices.

Goals

A player must have objectives.

Opposition

A player must struggle.

Managing Resources

A player must be able to balance diverse variables.

Gaming Theory

Game-Enhancing Features

Cooperation

Allowing players to assist one another.

Atmosphere

Developing a “look or feel” to the game.

Simulation

Constructing a game which mirrors some aspect of reality.

Variety of Encounter

Doing the same thing over and over and over again is boring.

Gaming Theory

Game Enhancing Features Continued

Position Identification

Empathizing with the protagonist.

Role-Playing

Enabling the player to take on a persona.

Socializing

Creating a meta-game environment.

Narrative Tension

Building up increasing levels of difficulty.

Agents

What is an agent?

Levels of Agent Complexity:

No Agent

Pure simulation

Examples: *Breakout*, a flight simulator

Oblivious Agent

Fixed patterns

Examples: *River Raid, Space Invaders*

Random Oblivious Agent

Randomly switching between patterns

Example: *Galaga*

Agents

Response Agent

Stimulus actions. Agent's action is determined by player's action.

Example: *Doom*

Goal-Directed Agent

Searches through possible actions. Weighs options.

Examples: chess, othello

Autonomous Agent

Has multiple goals. Uses some heuristic to make decisions.

Example: *Bolo*

Cooperating Agents

Inter-agent communication to reach "global" decisions.

Example: war simulations

Agents

Cyborgs

A computer assisted player.

Example: *Bolo*

Humans

A determined *homo sapien* at the controls.

Example: Bart

Programmers

A person who has written the game and has all the cheat codes.

Example: Us and our AI assignment.

FSA’s

The Return of Theory

What are FSA’s

FSA’s are computing machines.

They have state variables.

Generate New State Variables and Output.

This has what to do with games and AI?

Well.....

FSA’s

State Variables = Agent Internal Variables

Agent Internal Variables can represent:

Amount of Fuel
Amount of Ammo
Amount of Armor
or
Aggressiveness
Remorse
Happiness

Combine Agent Internal Variables with Sensor Input to pick action, and change Agent Internal Variables.

So Aggressive Agent responds differently than Happy Agent.

Seems like a good model of agent behavior.

The Continuing Saga of FSA's

How To Choose Action?

Action given by behaviors.

Behaviors compete for control of agent.

Releasing Mechanism evaluates input for behavior.

Value Behavior =
  Combination (Release Mechanism Values, Internal Variables)

Behaviors can inhibit other behaviors.

Behavior with best value drive agent.

Other behaviors allowed to make suggestions.

Genetic Algorithms

The latest greatest in AI

Genetic Algorithm evolution inspired algorithm

Other Evolutionary Algorithms are:

- Evolutionary Programming
- Evolutionary Strategy
- Classifier System

All of these follow the same basic algorithm.

They simulate the process of biologic evolution.

Very useful for optimization problems

Evolutionary Pseudo Code

```
t = 0
//initialize a random population of solutions
initpopulation P(t)

//evaluate fitness of all initial individuals in population
evaluate P(t)

//test for determination criterion (time, fitness, etc.)
while not done do

    //time goes by
    t = t + 1

    //select group for off-spring
    P’ = selectparents P(t);

    //create offspring by either recombining parent “genes” and
    // mutating or just mutating
    makeoffspring P’

    //evaluate fitness of offspring
    evaluate P’

    //pick survivors till next generation

    P = survivor P,P’
```

More Evolutionary Matters

So now that we know the basic algorithm for all Evolutionary Algorithms
what are the differences between them.

Genetic Algorithms:

-Demand problem be encoded into bits.

-Take a page from human reproduction and has parent bits swapped around in creating offspring.

-Parent population generally discarded.

Evolutionary Programming:

-Doesn't demand encoding of problem. Can use real values used
in program

-No recombination of parent genes. Simply perturb values so small changes very likely, big changes not likely

-Survival either decided in tournament or strictly best N solution survive.Usually Tournament

Evolutionary Strategies:

-Also works on real values.

-Parent carries information on how much child should be perturbed.
This also changed from generation to generation.

-Survival is usually deterministic, and also just children.

Classifier System:

-System used to describe agent behavior combining genetic algorithm
and a reinforcement learning algorithm.

Classifier Systems

Classisfier Systems is a way of developing and teaching a very simple agent.

Agent made of:
 Sensors
 Message Board
 Set of rules that run behavior

Agents grown by Genetic Algorithms

Rules encoded as bits.
First n bits sensor inputs
last m bits agent action to input

Genetic Algorithm used to try find optimal rules.

Classifier Systems can also learn, not just evolve

Learning Classifier System

Use reinforcement learning to improve individual agents

Rules given value to bid for the right to drive agent.

Rule with highest bid drives the agent. Loses amount value bid.

Agent gets pay back from environment. Payback to driving rule.

Successful rule gain ability to control agent
Bad rule loses ability to control agent

Hopefully before agent is dead, good rules in charge

Genetic Algorithm and Games

Not Widely Used in present games

Could Use in Development

Not obviously useful in game

Maybe send AI farm with game

Maybe use in a level based game

Reinforcement Learning might be usable

Machine Learning

teaching and old dog new tricks

Learning Agent needs four parts:

Performance Element:

Makes decision on external action.

Learning Element:

Makes changes to the Performance Element to try and improve it.

Critic:

Evaluates the agent on a constant standard of performance. Says when need improvement.

Problem Generator:

Makes suggestions to the Performance Element that may not be optimal but get new responses that help with learning process.

Decision Trees

Decision Tree is a tree in which Internal nodes are variable names.

Depending on value of variable pick a child of the node.

Until reach a leaf which is the action to be taken.

Decision Trees do have problems not very good at counting decision

ex. bad at returning if even number of inputs is 1, or a majority of inputs is 1

Decision Trees Learn

Method of learning is trying to find pattern in trial set.

Use heuristic to make tree from trials

Put important variables at top.

Important based how many trials variable classifies

Pseudo Code:

function Decision-Tree-Learning(examples,attributes,defaults)

  if examples is empty then return default

   else if all examples have the same classification, return classification.

   else if attributes is empty then return Majority-Value(examples)

   else best<-Choose-Attribute(attributes,examples)

     tree<-a new decision tree with root test best

     for each value v_i of best

       $examples_i$<-{elements of examples with best=v_i}

   subtree<-Decision-Tree-Learning($examples_i$,attributes-best,

   Majority-Value($examples_i$)

Neural Nets

Neural Nets are a simplified model of the way biologists believe the human brain works.

Neuron are replaced by Nodes. Synapse and Axons are replaced by links. The excititory and inhibitory abilities of input are replaced by weights on the value from a link.

Nodes are linked together.
Input to the node is a weighted sum.
If greater than a threshold the node outputs

The nodes are networked together (usually in either two or three layer networks.).

Input and output encoded.

Links start with random values.
Network learns by fiddling with weights.

Amount of fiddling based difference from right answer.

Problems With Neural Nets

Neural Nets are not the ultimate solve everything.

Like Decision Trees certain things they're just not good at.

Once out of training mode, don't really learn.

Not real intuitive how to get the behavior you want

Very difficult to program and get in stable state.

Machine Learning and Games

Not used by most games if any

Most of require a training process to learn

Not real clear how to know what the right answer is

If got to learn, need to keep from getting to good.

However, having a game that got better, or learned to play the way you liked would give you a cool game.

Natural Language Understanding

Can a Computer Understand English?

In Theory...

The meaning of words depends upon their context.

The trigram model: context can be reduced to the previous two words.

Syntax aids in deciphering semantics.

Probabilities are assigned to groups of words.

Time and space requirements are prohibitively large.

The Turing Test

In Practice...

Different game paradigms: *Zork*, *Ultima*, MUDs.

Simple command parsing.

Not-so-simple command parsing.

Noun-based questioning.

Menu-driven conversation with pre-recorded voice.

Simulating different languages in a game.

M-x psychoanalyze-pinhead

```
ELIZA: I am the psychotherapist.  Please, describe
your problems.  Each time you are finished talking,
type RET twice.

ZIPPY: If I had heart failure right now, I couldn't
be a more fortunate man!!

ELIZA: Why do you say that?

ZIPPY: We just joined the civil hair patrol!

ELIZA: What makes you believe you just joined the
civil hair patrol?

ZIPPY: PUNK ROCK!!  DISCO DUCK!!  BIRTH CONTROL!!

ELIZA: Maybe your plans have something to do with
this.
```

Searching Algorithms

[Note: Check out *http://www.essi.fr/~waldura/graphs.html* with the HotJava browser to see animations of these two searches.]

Depth First

```
dfs-visit (Graph G, Vertex u) {

        the vertex u is painted gray

        for all white successors v of u {

            dfs-visit(G, v)

        }

        u is painted black

  }


dfs (Graph G) {

        all vertices of G are first painted white

        dfs-visit(G, root of G)

  }
```

Searching Algorithms

Breadth First

```
bfs (Graph G) {

     all vertices of G are first painted white

      the graph root is painted gray and put in a queue

     while the queue is not empty {

          a vertex u is removed from the queue

          for all white successors v of u {

                v is painted gray
                v is added to the queue

          }

          u is painted black

     }

  }
```

Shortest Path Algorithms

[Note: These algorithms were stolen from *http://www.lis.pitt.edu/~john/shorpath.htm*. Check out that page for a more detailed discussion.]

Crash and Turn

1) Try to move in a straight line to the endpoint. Always remember the previous tile coordinates.

2) If a collision occurs, rotate right until a free passage is found, then move to that tile.

3) Repeat the previous two steps until the goal is reached or a collision with the previous tile occurs.

4) If we collide with our previous position, change to a left-hand rule.

Benefits: Simple to implement. Works OK in sparse environments.

Drawbacks: Relatively stupid. Will get stuck in loops.

Modifications:

Intelligently deciding whether to use the right-hand rule or the left-hand rule.

Precalculating before initiating move.

Scanning both directions.

Shortest Path Algorithms

Divide and Conquer

1) Take the current position and the destination position.

2) Draw a straight line between them.

3) Take the mid point of the line. If it is on an obstacle, perturb the point in some way (spiral out, for example) until it isn't.

4) Set this point as the new destination and iterate until you have a destination close enough to only require a simple move.

Benefits: Relatively simple. Concentrates on closer tiles.

Drawbacks: Doesn't work well on cluttered terrain or on terrain with long linear features.

Shortest Path Algorithms

A* Algorithm

[Note: A detailed description of the A* algorithm can be found at *http://www.himolde.no/~lonning/astar/astar.htm*.]

f = "goodness function"

g = cost of getting from the initial state to the present state

h = hypothetical cost of going from the present state to the final state

$\boldsymbol{f = g + h}$

You have a priority queue (probably implemented as a heap) that is prioritized by the how far you are from the start state (g) plus your estimation (h) of how far it is to the goal (Manhattan distance).

Prime the queue by putting in your start state.

Loop

If queue is empty you cannot reach the goal.

Pop off the first one.

Check to see if it's the goal- you win!

Add the unvisited neighboring grid squares (that aren't walls).

Keep a representation of the grid and mark each square with "an arrow" that points along the path to the goal.

Probability and Utility

The Mathematics of Fun

Games knowledge is not perfect
Agents occasionally must guess and takes chances.
To help make intelligent guesses we offer a quick refresher on the basics of probability.

Probability Values range from 0..1.

P(A) = probability of an event occurring

P(A or B)= P(A) + P(B) - P(A and B)

$$P(A|B) = \frac{P(A \text{ and } B)}{P(B)}$$

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Now That we know that basics of finding the probability of an event how do we use this to find the value of the event?

Utility

Utility is the value that the user places on reaching a certain outcome.

ex. A raffle gives away a Large Screen TV and a Trip to Hawaii.
To us a U(trip)>U(tv), however to someone in Hawaii U(tv) > U(trip).

Expected Value is the sum of the utility *probability for each possible result

$$EUV = \sum_i U(result_i) \times P(result_i)$$

Principle of Maximum Expected Utility - rational agent picks action with biggest EUV.

There are other principles to drive decision making.

Econ. Game Theory

Formal study of how to make choices in completely unrealistic situations.

The theory revolves around close study of two person games. Usually defined by a table like so.....

Table 1: Game Payoffs

A	Player A move 1	Player A move 2
Player B move 1	5	-10
Player B move 2	2	1

maxi-min principle: pick the move with the maximum minimum.

In the above game Player B would pick move 2.

Planning

What Organized People Do

Planners not real fast, a plan space search

If problem and plan space kept small, usable

If a very large problem, Cheat

Now on with the plan........

Planning

Start State: need to define the state at the start of the plan.

ex. agent at(home), have(money), have(car),~have(food)

Goal State: describe wanted at the end of a successful plan.

ex. agent at(home),have(money),have(food),have(car)

Operator Definition: the possible operations are defined in terms of preconditions necessary to use operator, and the result of using the operator

ex operator drive_to(x)

Precond: at(car), have(car)

result: at(x)

Planner search from goal to init.

Pick steps that fit in plan, til hit problem

Then backtrack.

Planning Pseudo Code

function Planner(intial, goal, operators)

plan<- Make-Minimal-Plan(intial,goal)

loop

    if Solution?(plan) then return plan

    S_{need}, c<-Select-SubGoal(plan)

    Choose-Operator(plan,operators,Sneed,c)

    Resolve-Threats(plan)

function Select-SubGoal(plan)

Pick a plan step S_{need} from STEPS(plan)

with a precondition c that has not been achieved

return S_{need},c

procedure Choose-Operator(plan,operators, S_{need},c)

  choose a step S_{add} from operators or Steps(plan) that has c as an effect

  if there is no such step then fail

  add a causal link S_{add}->S_{need} to LINKS(plan)

  add the ordering constraint S_{add}<S_{need} to ORDERINGS(plan)

  if S_{add} is a newly added step from operators then

      add S_{add} to STEPS(plan)

      add Start<S_{add}<Finish to ORDERINGS(plan)

procedure Resolve-Threats(plan)

for each S_{threat} that threatens a link S_i->S_j in LINKS(plan)

choose either

PROMOTION:add $S_{threat}<S_i$ to ORDERINGS(plan)

DEMOTION:add $S_j<S_{threat}$ to ORDERINGS(plan)

if not CONSISTENT(plan) then fail

Techniques in Creating AI TeamWork

1. Formalized Teamwork

   Teamwork is hard coded in.

   Agents are forced to coordinate.

2. Emergent Teamwork

   Agents don’t coordinate activity.

   Agents only worry about there own behavior.

   Complex group behavior occurs

Formalized Team AI

1. Hierarchy of AI

(aka The Van Damn method)

Multiple levels of agents.

Agent at the top direct lower agents.

Behind the scenes high abstraction agent.

or

Onscreen character that directs them.

ex. Have unseen general agent who directs were armies go.
Soldiers in charge of getting there.

ex. Powerful monster with horde of little monsters he directs.

More Hierarchy.

Why use hierarchy set-up?

Allows different levels of abstractions. Makes algorithm design easier.

Keeps down number of agents and comparisons down. Usually making Computation quicker and easier.

Fairly easy to force the behavior the programmer wants.

Can be made to work quickly.

Even More Hierarchy

Why not always use?

Not always possible to construct an appropriate hierarchy.

Not opportunistic.

Have to have algorithm that deals with lot of issues.

Changing the algorithm can require lots of work.

Black Board System

What is a blackboard System?

AI architecture that emulates the process that occurs when a group of engineers tries to solve a problem around a blackboard.

Problem is displayed on the blackboard.

Engineer shouts out the suggestions.

Manager decides what is added to the board.

BlackBoard System parts:

BlackBoard: central message board where state of the problem is stored.

Knowledge Sources: Separate AI's and agents that study the board to make suggestions.

Control component: Serves as a control over which KS's are allowed to write to the BlackBoard, what areas should be pursued. Keeps focused on goal.

Model very flexible

Blackboard can be divided to improve search

Events used to trigger KS action

The knowledge sources very heterogeneous.

Black Board Systems

Why use this model?

Allows problem decisions to be broken down into manageable pieces.

Allows for very modular construction. Easy to change.

Allows opportunistic problem solving.

Fairly realistic model of way cooperation works

Why not always use?

Fairly bulky communication procedure that may slow things down.

Controlling were the problem solving process goes is tricky.

Not always immediately obvious how to force behavior you want

Emergent Group Behavior

Agents that take a look at what's around them.

Take action with no communication.

Group of agents result in group action.

Best example of this is Craig Reynolds flocking algorithm. (it's more a herding, schooling algorithm. There are no flying V's.)

Boids governed by three basic needs.

Satisfing needs result in group behavior.

The boids are governed by three needs (listed in order of importance).

1. avoid colliding with nearby flockmates
2. stay at the same velocity of nearby flockmates.
3. stay close to nearby flockmates.

Emergent Teamwork

boid get vector for each need.

vector combine in average sum. or used as priority allow.

boids use limited sense input to model real world.

For demonstration of this algorithm see:

www.fusebox.com/cb/alife.html

Emergent Teamwork

Reasons to use this method?

- Fairly short simple code.
- No communication overhead making fairly quick.
- Can produce extremely interesting behavior.

Problems with this method?

- Very hard to come up with algorithm.
- Hard to tweak values to get the behavior you want.
- Figuring why it's not doing a behavior very hard.

Time Constraints

- I am an AI. I am thinking, I am thinking, I am thinking...

BAM! I've been shot!

- That AI should have been more reactive...

1. Coping with a limited time in which the AI can compute without having to make a decision.

2. Maximum reactivity == Minimum hairy planning.

3. Reactivity does not imply lack of planning.

4. Sometimes you have to hack things to make them work faster. "Scripts" vs. Planning.

Time Constraints - II

- How to deal with time?

  1. Break up your processing into discrete “turns.”

  2. Each turn, test whether certain “danger” events have occurred. If so, deal with them before continuing with your long term goals.

- What if I can’t think quickly enough?

  1. Perhaps you need to simplify your algoritm?

  2. “If you are running a BHP (Big Hairy Planner), try doing a small part of the processing at each turn.” --lpk

  3. Computing with a state machine, try making a fixed number of transitions per turn.

  4. Explore Anytime Algorithms, a class of planning algorithms which will come up with a plan regardless of how long it is allowed to run. [Dean 88]

Scheduling

- When does the AI think?

  1. Round-robin -- part of your event loop

  2. Idle-processing -- only when time is available

- How long?

  1. Program your AI’s such that they run quickly. It’s the best you can do, short of some major hacks.

- Should I use threads?

  1. No, not unless you know what you’re doing. Debugging is a pain.

  2. In the average case, a multi-threaded algorithm takes at least as much time as a single-threaded one, if not more.

Scheduling - II

- An example AI class:

```
class AI {

    // ...

    void gameTick();
    // This function is called
    // once every 1/60th of a second.
    // Don't take too long!

    void somethingHappened(Event e);
    // An event ocurred that the AI
    // should be aware of.

    // ...

};
```

Pay attention! This is relevant to the AI assignment.

Know Your World

- An agent in a dynamic environment.

  1. Beware of invalid references. Smart pointers?

  2. Remember: things in the world are changing. You know what happens when you *assume*...

- Visibility / Limiting the enemy's awareness.

  1. Make the AI's "know" their limitations. Assume benevolence.

  2. Make the AI's have a special interface to the world; one which intentionally limits their perception. Assume malevolence.

Motivations and Goals

- What does the AI want to do?

  1. Destroy a target. Protect the Moon-Base.

  2. Destroy the players. The Enemy Must Die.

  3. Help the players. A Friend in Need.

  4. Nothing. Attack them that attack you.

- Where are the goals encoded?

  1. The mission objectives.

  2. The Game Engine(tm). What's that?

Weapons and Generality

- What do I do with this axe?

  1. It is a lot harder to write a totally general AI than it is to write a simple one. A no brainer, right? Don't try it.
  2. Hard-code abilities and how to deal with them. Your character should be good at using its tools.
  3. Avoid learning at all costs! This doesn't mean "don't alter your plans," but "don't force your AI to learn what you could code in twenty lines."

- Who do I kill with this axe?

  1. This is more of a motivation/goal.
  2. Learning can be applied, especially with respect to who to be afraid of. Stay away from the theif until you have the nasty knife!