General Game Playing

Introduction

Michael Genesereth
Computer Science Department
Stanford University
Game Playing

Human Game Playing
• Intellectual Activity
• Competition

Computer Game Playing
• Testbed for AI
• Limitations
Limitations of Game Playing for AI

Narrowness
  Good at one game, not so good at others
  Cannot do anything else

Not really testing intelligence of machine
  Programmer does all the interesting analysis / design
  Machine simply follows the recipe
General Game Players are systems able to play arbitrary games effectively based solely on formal descriptions supplied at “runtime”.

Translation: They don’t know the rules until the game starts.

Must figure out for themselves: legal moves, winning strategy in the face of incomplete info and resource bounds
Versatility
Novelty
International GGP Competition
Annual GGP Competition
Held at AAAI or IJCAI conference
Administered by Stanford University
(Stanford folks not eligible to participate)
Winners

2005 - ClunePlayer - Jim Clune (USA)
2006 - FluxPlayer - Schiffel, Thielscher (Germany)
2007 - CadiaPlayer - Bjornsson, Finsson (Iceland)
2008 - CadiaPlayer - Bjornsson, Finsson (Iceland)
2010 - Ary - Mehat (France)
2011 - TurboTurtle - Schreiber (USA)
2012 - CadiaPlayer - Bjornsson, Finsson (Iceland)
2013 - TurboTurtle - Schreiber (USA)
2014 - Sancho - Draper (USA), Rose (UK)
2015 - Galvanise - Emslie
2016 - WoodStock - Piette (France)
GGP-05 Winner Jim Clune
International GGP Competition
GGP-07, GGP-08, GGP-12 Winners
Carbon versus Silicon
Human Race Being Defeated
Game Description
Multiplicity of Games
Finite Synchronous Games

Environment
- Environment with finitely many states
- One initial state and one or more terminal states
- Each state has a unique goal value for each player

Players
- Fixed, finite number of players
- Each with finitely many moves

Dynamics
- Finitely many steps
- Only one player moves on each step
- Environment changes only in response to moves
Good News: Since all of the games that we are considering are finite, it is possible in principle to communicate game information in the form of state graphs.

Problem: Size of description. Even though everything is finite, these sets can be large.

Solution:
Exploit regularities / structure in state graphs to produce compact encoding
States

```
cell(1,1,x)  
cell(1,2,b)  
cell(1,3,b)  
cell(2,1,b)  
cell(2,2,o)  
cell(2,3,b)  
cell(3,1,b)  
cell(3,2,b)  
cell(3,3,x)  
control(o)
```
<table>
<thead>
<tr>
<th>cell(1,1,x)</th>
<th>cell(1,2,b)</th>
<th>cell(1,3,b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell(2,1,b)</td>
<td>cell(2,2,o)</td>
<td>cell(2,3,b)</td>
</tr>
<tr>
<td>cell(3,1,b)</td>
<td>cell(3,2,b)</td>
<td>cell(3,3,x)</td>
</tr>
<tr>
<td>control(o)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **mark(1,3)**

<table>
<thead>
<tr>
<th>cell(1,1,x)</th>
<th>cell(1,2,b)</th>
<th>cell(1,3,o)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell(2,1,b)</td>
<td>cell(2,2,o)</td>
<td>cell(2,3,b)</td>
</tr>
<tr>
<td>cell(3,1,b)</td>
<td>cell(3,2,b)</td>
<td>cell(3,3,x)</td>
</tr>
<tr>
<td>control(x)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
init(cell(1,1,b))
init(cell(1,2,b))
init(cell(1,3,b))
init(cell(2,1,b))
init(cell(2,2,b))
init(cell(2,3,b))
init(cell(3,1,b))
init(cell(3,2,b))
init(cell(3,3,b))
init(control(x))

legal(P,mark(X,Y)) :-
  true(cell(X,Y,b)) &
  true(control(P)) & X#b & Y#b

legal(x,noop) :-
  true(control(o))

legal(o,noop) :-
  true(control(x))

next(cell(M,N,P)) :-
  does(P,mark(M,N)) &
  true(cell(M,N,Z)) & Z#b &
  (M#J | N#K)

next(cell(M,N,Z)) :-
  does(P,mark(M,N)) &
  true(cell(M,N,Z)) & Z#b &
  true(control(P))

next(cell(M,N,b)) :-
  does(P,mark(M,N)) &
  true(cell(M,N,Z)) & Z#b &
  true(control(P))

next(control(x)) :-
  true(control(o))

next(control(o)) :-
  true(control(x))

row(M,P) :-
  true(cell(M,1,P)) &
  true(cell(M,2,P)) &
  true(cell(M,3,P))

column(N,P) :-
  true(cell(1,N,P)) &
  true(cell(2,N,P)) &
  true(cell(3,N,P))

diagonal(P) :-
  true(cell(1,1,P)) &
  true(cell(2,2,P)) &
  true(cell(3,3,P))

diagonal(P) :-
  true(cell(1,3,P)) &
  true(cell(2,2,P)) &
  true(cell(3,1,P))

terminal :- line(P)

terminal :- ~open

goal(x,100) :- line(x)
goal(x,50) :- draw
goal(x,0) :- line(o)
goal(o,100) :- line(o)
goal(o,50) :- draw
goal(o,0) :- line(x)

goal(x,100) :- line(x)
goal(x,50) :- draw
goal(x,0) :- line(o)
goal(o,100) :- line(o)
goal(o,50) :- draw
goal(o,0) :- line(x)

line(P) :- row(M,P)
line(P) :- column(N,P)
line(P) :- diagonal(P)

open :- true(cell(M,N,b))
draw :- ~line(x) & ~line(o)
What we see:

\[
\text{\texttt{next}}(\text{\texttt{cell}}(M,N,x)) : - \\
\text{\texttt{does}}(\text{\texttt{white}}, \text{\texttt{mark}}(M,N)) & \\
\text{\texttt{true}}(\text{\texttt{cell}}(M,N,b))
\]

What the player sees:

\[
\text{\texttt{next}}(\text{\texttt{welcoul}}(M,N,\text{\texttt{himenoiding}})) : - \\
\text{\texttt{does}}(\text{\texttt{himenoiding}}, \text{\texttt{dukepse}}(M,N)) & \\
\text{\texttt{true}}(\text{\texttt{welcoul}}(M,N,\text{\texttt{lorenchise}}))
\]
Game Playing
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cell(1,1,b)</td>
<td>cell(1,2,b)</td>
<td>cell(1,3,b)</td>
</tr>
<tr>
<td>cell(2,1,b)</td>
<td>cell(2,2,b)</td>
<td>cell(2,3,b)</td>
</tr>
<tr>
<td>cell(3,1,b)</td>
<td>cell(3,2,b)</td>
<td>cell(3,3,b)</td>
</tr>
</tbody>
</table>

control(x)
mark(1,1)
mark(1,2)
mark(1,3)
mark(2,1)
mark(2,2)
mark(2,3)
mark(3,1)
mark(3,2)
mark(3,3)
State Update

cell(1,1,b)  
cell(1,2,b)  
cell(1,3,b)  
cell(2,1,b)  
cell(2,2,b)  
cell(2,3,b)  
cell(3,1,b)  
cell(3,2,b)  
cell(3,3,b)  
cell(1,1,b)  
cell(1,2,b)  
cell(1,3,x)  
cell(2,1,b)  
cell(2,2,b)  
cell(2,3,b)  
cell(3,1,b)  
cell(3,2,b)  
cell(3,3,b)  
to control(x)  
mark(1,3)  
control(o)
Complete Game Graph Search
Incomplete Game Tree Search

How do we evaluate non-terminal states?
General Heuristics
  Goal proximity (everyone)
  Maximize mobility (Barney Pell)
  Minimize opponent’s mobility (Jim Clune)
Monte Carlo Search

Monte Carlo Tree Search

UCT - Uniform Confidence Bounds on Trees
Second Generation GGP

Monte Carlo Search
Offline Processing of Game Descriptions
  Compile to do the search faster
  Reformulate problem to decrease size of search space

*What human programmers do in creating game players*
Conversion of logic to traditional programming language
Simple, widely published algorithms
several orders or magnitude speedup
no asymptotic change

Conversion to Field Programmable Gate Arrays (FPGAs)
several more orders of magnitude improvement
Hodgepodge = Chess + Othello

Analysis of joint game:
Branching factor as given to players: \( a \times b \)
Fringe of tree at depth \( n \) as given: \((a \times b)^n\)
Fringe of tree at depth \( n \) factored: \( a^n + b^n \)
Reformulation Opportunities

Examples
  Factoring, e.g. Hodgepodge
  Bottlenecks, e.g. Triathalon
  Symmetry detection, e.g. Tic-Tac-Toe
  Dead State Removal

Trade-off - cost of finding and using structure vs savings
  Sometimes cost proportional to size of description
  Sometimes savings proportional to size of game tree
Automatic Programming

```java
public class CreateObjectDemo {
    public static void main(String[] args) {
        // create a point object and two rectangle objects
        Point origin_one = new Point(23, 94);
        Rectangle rect_one = new Rectangle(origin_one, 100, 200);
        Rectangle rect_two = new Rectangle(50, 100);

        // display rect_one's width, height, and area
        System.out.println("Width of rect_one: " + rect_one.width);
        System.out.println("Height of rect_one: " + rect_one.height);
        System.out.println("Area of rect_one: " + rect_one.area());

        // set rect_two's position
        rect_two.origin = origin_one;

        // display rect_two's position
        System.out.println("X Position of rect_two: " + rect_two.origin.x);
        System.out.println("Y Position of rect_two: " + rect_two.origin.y);

        // move rect_two and display its new position
        rect_two.move(40, 72);
    }
}
```
Algorithmic Expertise

Knuth in a Box
Game Theory

\[
\begin{array}{cc|cc}
  & a & b \\
  a & 4 & 3 \\
  b & 2 & 1 \\
\end{array}
\]

\[
\begin{array}{cc|cc}
  & a & b \\
  a & 4 & 1 \\
  b & 3 & 2 \\
\end{array}
\]
Demoralizing the Opponent
Fooling the Opponent
Conclusion
General Game Playing is not a game

Someone said:

I did my Master's thesis on dots and boxes!

Nobody laughed...
Serious Business
Theory of Intelligence

Dimensions of Intelligence
- Representation of the World
- Correct and efficient reasoning
  - Rationality with incomplete info and resource bounds

Generality
- Not just ability to perform well on specific tasks
- But also ability to perform well in general
- Test of intelligence, not just test of knowledge
The main advantage we expect the advice taker to have is that its behavior will be improvable merely by making statements to it, telling it about its … environment and what is wanted from it.

- John McCarthy 1958
The potential use of computers by people to accomplish tasks can be “one-dimensionalized” into a spectrum representing the nature of the instruction that must be given the computer to do its job. Call it the what-to-how spectrum. At one extreme of the spectrum, the user supplies his intelligence to instruct the machine with precision exactly how to do his job step-by-step. ... At the other end of the spectrum is the user with his real problem. ... He aspires to communicate what he wants done ... without having to lay out in detail all necessary subgoals for adequate performance.

- Ed Feigenbaum 1974
The General Problem Solver demonstrates how generality can be achieved by factoring the specific descriptions of individual tasks from the task-independent processes.
A human being should be able to change a diaper, plan an invasion, butcher a hog, conn a ship, design a building, write a sonnet, balance accounts, build a wall, set a bone, comfort the dying, take orders, give orders, cooperate, act alone, solve equations, analyze a new problem, pitch manure, program a computer, cook a tasty meal, fight efficiently, die gallantly. 

*Specialization is for insects.*
Course Details
<table>
<thead>
<tr>
<th>Month</th>
<th>Day</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>April</td>
<td>5</td>
<td>Introduction</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>Game Description</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>Game Playing</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>Incomplete Search</td>
</tr>
<tr>
<td>May</td>
<td>3</td>
<td>Statistical Search</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Logical Optimization</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>Materialization and Reformulation</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>Game Tree Reformulation, e.g. Factoring</td>
</tr>
<tr>
<td>June</td>
<td>7</td>
<td>Final Competition</td>
</tr>
</tbody>
</table>

*You are here.*
Teams

Composition

3 people each (2 or 4 okay with good reason)

Names:
  Pansy Division
  The Pumamen
  Team Camembert
  Mighty Bourgeoisie
  Greedy Bastards
  Red Hot Chili Peppers
  Michael Genesereth
  /*^*/
  X Æ A-12

Identifiers:
  pansy_division
  punamen
  camembert
  bourgeoisie
  greedybastards
  peppers
  michael_genesereth
  happy
  x_ash_a_12
Technology

Language
  Java
  ***Javascript***
  Fortran

Operating System
  Mac OS
  Unix
  Linux

Hardware
  Whatever you like … but …
  Able to access course website
You do not have to win competitions to get a perfect score, but your players must play correctly and illustrate weekly lessons.

No curve. Grades are based completely on mastery of subject matter as demonstrated via components above.

*Grades in this course are generally quite high (because people tend to work hard).*
http://cs227b.stanford.edu