General Game Playing

Introduction

Game Playing

Human Game Playing

- Intellectual Activity
- Skill Comparison

Playing strategy games like chess and checkers couples intellectual activity with competition. By playing games, we can exercise and improve our intellectual skills. The competition adds excitement and allows us to compare our skills to those of others.

Game Playing

Human Game Playing

- Intellectual Activity
- Skill Comparison

Computer Game Playing

- Testbed for AI

The same motivation accounts for interest in Computer Game Playing as a testbed for Artificial Intelligence. Programs that think better should be able to win more games, and so we can use competitions as an evaluation technique for intelligent systems. Unfortunately, building programs to play specific games has limited value in AI.

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

(1) To begin with, specialized game players are very narrow. They are often good at one game but not so good at others. Deep Blue may have beaten the world chess champion, but it has no clue how to play checkers; it cannot even balance a checkbook. (2) A second problem with specialized game playing systems is that they do only part of the work. Most of the interesting analysis and design is done in advance by their programmers. The systems themselves might just as well be tele-operated.

All is not lost. Many believe that the idea of game playing can be used to good effect to inspire and evaluate good work inArtificial Intelligence, but it requires moving more of the mental work to the computer itself. This can be done by focussing attention on General Game Playing.

General Game Playing

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 uncertainty and resource bounds

General game players are systems able to play arbitrary games based solely on formal game descriptions supplied at "runtime". In a typical general game playing session, the players know nothing about the game in advance. Once the game begins, they receive a game description. Based solely on this description, they must figure out how to play the game legally and effectively. Furthermore, they must deal with uncertainty (about the actions of the other players) and resource bounds (in the form of a game clock that limits their computation time).

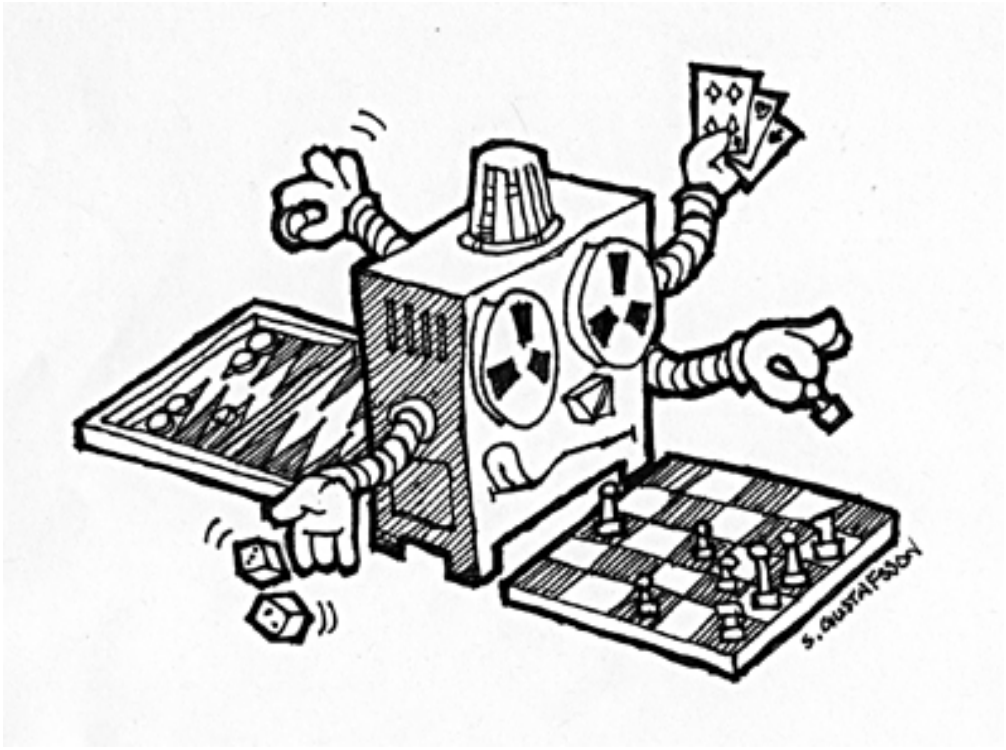
Technology

Unlike specialized game players (e.g. Deep Blue), they do not use algorithms designed in advance for specific games.

Artificial Intelligence Technologies
- knowledge representation
- deduction, reformulation, induction, …
- rational behavior w/ uncertainty, resource bounds

Given this arrangement, general game players cannot rely on algorithms designed in advance for specific games. General game playing expertise depends on intelligence on the part of the game player and not just intelligence of the programmer of the game player. In order to perform well, general game players typically incorporate ideas from multiple subareas of Artificial Intelligence technologies, such as knowledge representation, reasoning, and rational decision making.

Variety of Games



The upshot is that general game players are able to play different kinds of games. They can play simple games (like Tic-Tac-Toe) and complex games (like Chess). Games with simultaneous play (like Diplomacy) and games with alternating play (like Risk). Games with complete information or games with incomplete information (e.g. Battleship). Games with different numbers of players (Blocks World, Chess, Chinese Checkers). Games with or without communication among the players (e.g. Bughouse Chess). Zero-sum games and cooperative games.

International GGP Competition

Annual GGP Competition

Annual GGP Competition
- Held at AAAI or IJCAI conference
- Administered by Stanford University
- (Stanford folks not eligible to participate)

The main driving force of the General Game Playing community is the annual International General Game Playing competition. The competition is typically held in conjunction with the national conference on Artificial Intelligence sponsored by the AAAI (the Association for the Advancement of Artificial Intelligence) or the biennial IJCAI (the International Conference on Artificial Intelligence). It is administered by Stanford University, so those of you at Stanford are not eligible to participate.

History

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)

The competition began in 2005 and has run annually ever since (with the exception of 2009), and it has been won by players from various different countries - including the US, Germany, France, and Iceland.

GGP-05 Winner Jim Clune

/42

Here is a photo of Jim Clune (the winner of the first GGP competition) being congratulated by the AAAI president Ron Brachman. He was obviously pleased at his player having won. However, that smile might be due in part to the $10,000 he received for winning the competition.

GGP-06 Winners

/42

Here we have Michael Thielscher and Stephan Schiffel - winners of the competition at AAAI-06.

GGP-07, GGP-08, GGP-12 Winners

/42

On the left, Hilmar Finsson and Yngve Bjornsson, the winners in 2007, 2008, and 2012, together with Stephan Schiffel and Michael Thielscher, the winners in 2006, and Jim Clune, the winner in 2005.

GGP-11, GGP-13 Winner

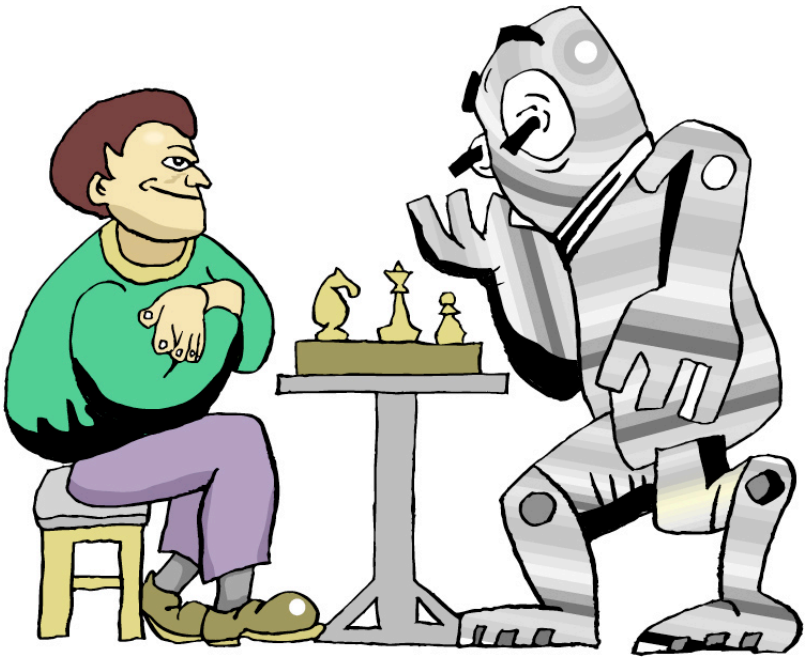
Here we have Hilmar Finsson congratulating a beaming Sam Schreiber at the end of GGP-11.

Other Games, Other Winners

/54

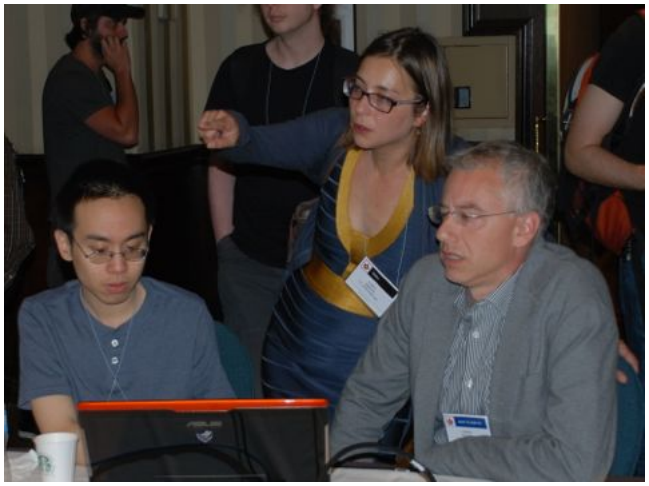
The contests are exciting, and even the spectators get in on the action. Without getting into specifics, let me just say that some people have come away from these competitions wealthier than beforehand.

Carbon versus Silicon



In addition, to the regular GGP competition, there is now an annual battle between Carbon and Silicon. Unfortunately for us, the humans are not doing that well.

Human Race Being Defeated



/42

For example, in 2012, CadiaPlayer handily defeated the human race (represented by Chris Welty shown here with some advisors). In keeping with the double-elimination format of the competition, the contestants played two different games. In the first, CadiaPlayer handily defeated the human in a well-played match of Dual Connect 4. It then won a match of Platform Jumpers. As a consolation prize, the human was awarded two bottles of Scotch, in part to ease his disappointment at letting down the human race.

Game Description

Finite Synchronous Games

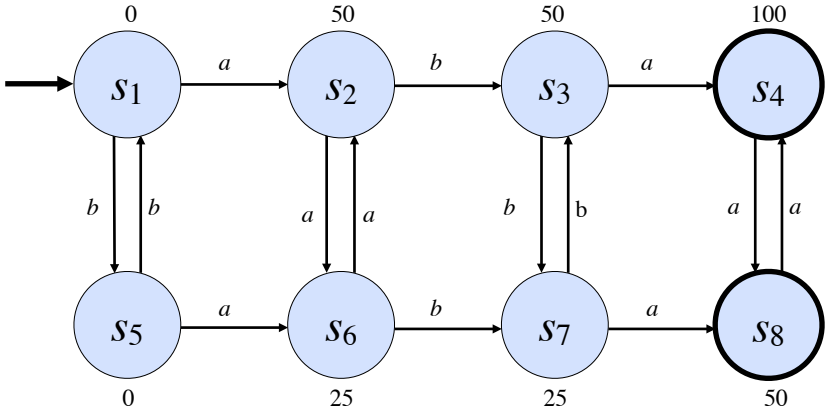
Environment
- Environment with finitely many states
- One initial state and one or more terminal states
- Each state has a unique goal value for each player

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

Dynamics
- Finitely many steps
- All players move on all steps (some no ops)
- Environment changes only in response to moves

Despite the variety of games treated in General Game Playing, all games share a common abstract structure. Each game takes place in an environment with finitely many states, with one distinguished initial state and one or more terminal states. In addition, each game has a fixed, finite number of players; each player has finitely many possible actions in any game state, and each state has an associated goal value for each player. The dynamic model for general games is synchronous update: all players move on all steps (although some moves could be "no-ops"), and the environment updates only in response to the moves taken by the players.

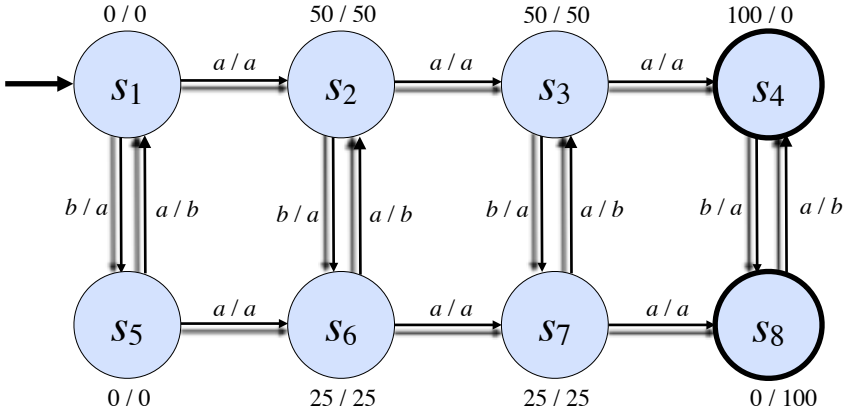
Single Player Game



/42

Given this common structure, we can think of a game as a state graph, like the one shown here. In this case, we have a game with one player, with eight states (named s1, ... , s8), with one initial state (s1), with two terminal states (s4 and s8). The numbers associated with each state indicate the values of those states. The arcs in this graph capture the transition function for the game. For example, if the game is in state s1 and the player does action a, the game will move to state s2. If the player does action b, the game will move to state s5.

Multiple Player Game



a

/42

In the case of multiple players with simultaneous moves, the arcs become multi-arcs, with one arc for each combination of the players' actions. Here is an example of a simultaneous move game with two players. If in state s1 both players perform action a, we follow the arc labelled a / a. If the first player does b and the second player does a, we follow the b / a arc. We also have different goals for the different players. For example, in state s4, player 1 gets 100 points whereas player 2 get 0 points; and, in state s8, the situation is reversed.

Direct Description

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.

The state graph captures the essential information about a game. Since all of the games that we are considering are finite, it is possible, in principle, to describe such games in the form of state graphs.

Problem and Solution

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

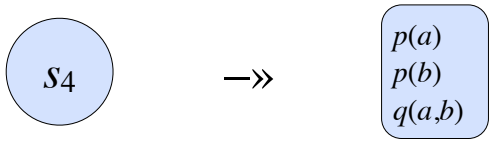
Solution:

Exploit regularities / structure in state graphs to produce compact encoding

Unfortunately, such explicit representations are not practical in all cases. Even though the numbers of states and actions are finite, these sets can be extremely large; and the corresponding graphs can be larger still. For example, in chess, there are thousands of possible moves and more than 10^30 states. Fortunately, we can solve this problem by exploiting regularities in the game to produce compact encodings.

Structured States and Actions

Monolithic states –» structured states

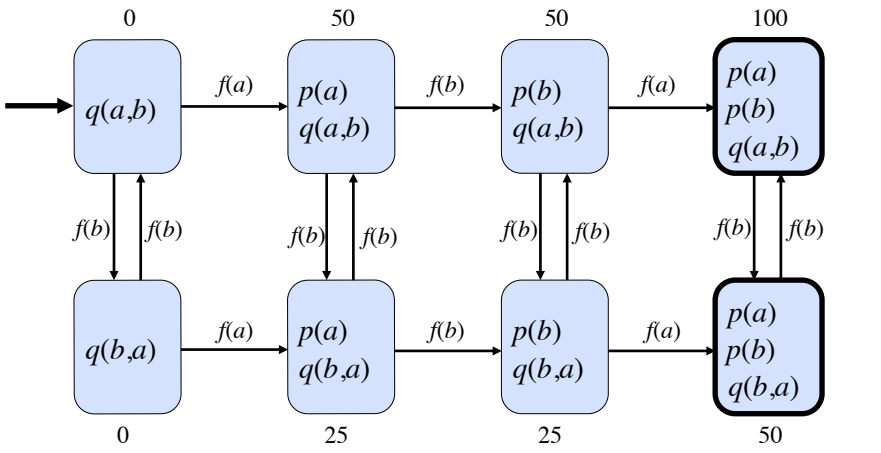


Monolithic actions –» structured actions

$$a \text{ –» } f(a,b)$$

In practice, we rarely think of states as monolithic entities. More frequently, we characterize states in terms of propositions that are true in those states. Similarly, actions often have structure.

Structured State Machine



a

/42

This leads to the notion of a structured state machine like the one shown here. The overall structure is the same as in the simple state machine shown earlier, but in this case, we have revealed the structure of the states. Now, by itself this does not help us since the graph is the same size as before. However, actions frequently affect only some of the propositions that are true in states and leave others unchanged. By exploiting these limitations on the effects of actions, we can often encode state graphs compactly by writing logical rules in place of explicit graphs.

Game Description Language

Game Description Language (or GDL) is a formal language for encoding the rules of games.

Game rules written as sentences in Symbolic Logic.

GDL is widely used in the research literature and is used in virtually all General Game Playing competitions.

GDL extensions are applicable in real-world applications such as Enterprise Management and Computational Law.

GDL is a formal language for encoding games in this way. As we shall see, it is based on Symbolic Logic. It is widely used in the research literature and is used in virtually all General Game Playing competitions. Moreover, it forms the basis for some more expressive variants that have significant value in real-world applications, such as Enterprise Management and Computational Law.

Example

Tic-Tac-Toe

```
role(x)
role(o)

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))

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

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

```
next(cell(M,N,P)) :-
    does(P,mark(M,N))

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

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

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

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

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

terminal :- line(P)
terminal :- ~open

goal(x,100) :- line(x) & ~line(o)
goal(x,50) :- ~line(x) & ~line(o)
goal(x,0) :- ~line(x) & line(o)
goal(o,100) :- ~line(x) & line(o)
goal(o,50) :- ~line(x) & ~line(o)
goal(o,0) :- line(x) & ~line(o)
```

```
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))

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

open :-
true(cell(M,N,b))
```

Here we see an example of GDL, in this case the rules for the game of Tic-Tac-Toe. We discuss the specifics of GDL in our next lesson. For now, the details are unimportant. The one thing to note here is that this one page of rules fully describes a game of thousands of states. That's a significant saving over the state graph. The improvement in more complex games can be even more dramatic. For example, it is possible to describe the rules of Chess in just four pages of rules like these.

Game Management

Game Management

Game Management is the process of administering a game in General Game Playing.

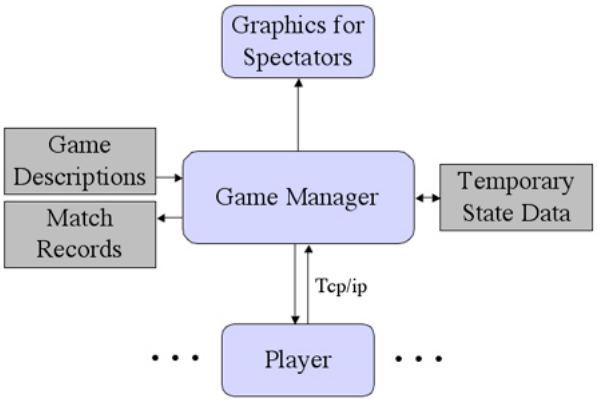
Match = instance of a game.

Components:

- Game Manager
- Game Playing Protocol

Game Management is the process of administering a game in a general game playing setting. More properly, it should be called match management, as the issue is how to manage individual matches of games, not the games themselves. However, everyone seems to use the phrase “Game Management”, and so we are stuck with it. In this segment, we start with an overview of the General Game Playing environment illustrating the central role of the Game Manager. We then discuss the General Game Playing communication protocol.

Game Manager



Here is a diagram of a typical general game playing ecosystem. At the center of the ecosystem is the game manager. The game manager maintains a database of game descriptions, maintains some temporary state for matches while they are running, and maintains a database of match results. The game manager communicates with game players using the Internet's TCP/ip protocol. It also provides a user interface for users who want to schedule matches, and it provides graphics for spectators watching matches in progress.

The process of running a match goes as follows. Upon receiving a request to run a match, the Game Manager's first sends a start message to each player to initiate the match. Once game play begins, it sends play messages to each player to obtain their moves and simulates the results. This part of the process repeats until the game is over. The Manager then sends stop messages to each player.

General Game Playing Protocol

Start

Manager sends Start message to players

Start(id, $role$, $description$, $startclock$, $playclock$)

The start message lists the name of the match, the role the player is to assume (e.g. white or black in chess), a formal description of the associated game (in GDL), and the start clock and play clock associated with the match. The start clock determines how much time remains before play begins. The play clock determines how much time each player has to make each move once play begins. Upon receiving a start message, each player sets up its data structures and does whatever analysis it deems desirable in the time available. It then replies to the Game Manager that it is ready for play. Having sent the start message, the game manager waits for replies from the players. Once it has received these replies OR once the start clock is exhausted,

General Game Playing Protocol

Start

Manager sends Start message to players

Start(*id*, *role*, *description*, *startclock*, *playclock*)

Play

Manager sends Play messages to players

Play(*id*, *actions*)

Receives plays in response

On each step, the Game Manager sends a play message to each player. The message includes information about the actions of all players on the preceding step. (On the first step, the argument is nil.) On receiving a play message, players spend their time trying to decide their moves. They must reply within the amount of time specified by the match's play clock. The Game Manager waits for replies from the players. If a player does not respond before the play clock is exhausted, the Game manager selects an arbitrary legal move. In any case, once all players reply or the play clock is exhausted, the Game manager takes the specified moves or the legal moves it has determined for the players and determines the next game state. It then evaluates the termination condition to see if the game is over. If the game is not over, the game manager sends the moves of the players to all players and the process repeats.

General Game Playing Protocol

Start

Manager sends Start message to players

start(*id*, *role*, *description*, *startclock*, *playclock*)

Play

Manager sends Play messages to players

play(*id*, *actions*)

Receives plays in response

Stop

Manager sends Stop message to players

stop(*id*, *actions*)

Once a game is determined to be over, the Game Manager sends a stop message to each player with information about the last moves made by all players. The stop message allows players to clean up any data structures for the match. The information about previous plays is supplied so that players with learning components can profit from their experience. Having stopped all players, the Game manager then computes the rewards for each player, stores this information together with the play history in its database, and ceases operation.

Game Playing

Game Playing

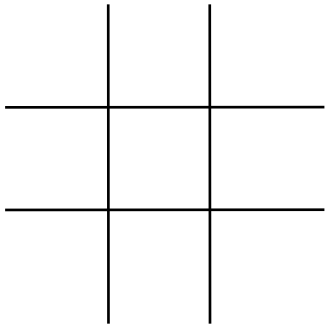
Logical reasoning in searching game tree:
- Initial state
- Legal actions for each player in each state
- State resulting from execution of legal action
- Value of each state for each player
- Determination of whether state is terminal

Easy to convert from logic to other representations
- Simplicity of logical formulation
- Simple, widely published algorithms
- 3-4 orders or magnitude speedup
- no asymptotic change

Having a formal description of a game is one thing; being able to use that description to play the game effectively is something else entirely. The player must be able to compute the initial state of the game. It must be able to compute which moves are legal in every state. It must be able to determine the state resulting from a particular combination of moves It must be able to compute the value of each state for each player. And it must be able to determine whether any given state is terminal.

Since game descriptions are written in symbolic logic, it is obviously necessary for a game player to do some amount of automated reasoning. There are two extremes here. (1) One possibility is for the game player to process the game description interpretively throughout a game. (2) The second possibility is for the player to use the description to devise a specialized program and then use that program to play the game. This is effectively automatic programming. As this is just an introduction, we will discuss the first possibility and leave it to you to think about the second possibility and various hybrid approaches.

Initial State



To start with, a player can use the game description to determine the initial state. In the case of Tic-Tac-Toe, we have a board with nine empty cells.

Legal Moves

White's moves:

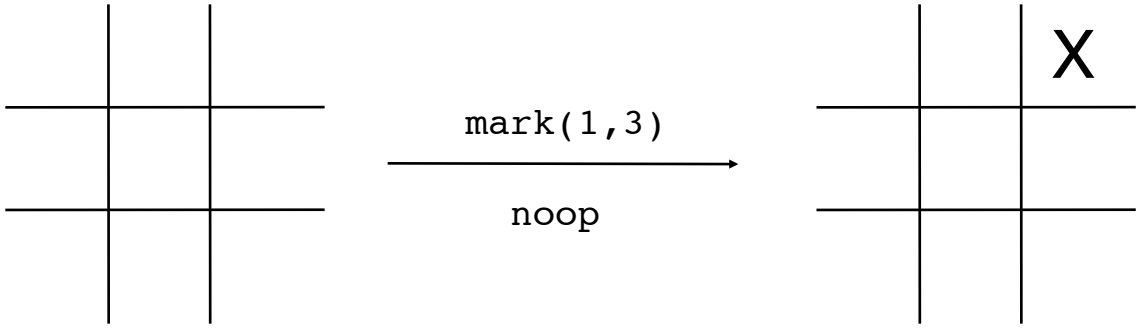
```
mark(1,1)
mark(1,2)
mark(1,3)
mark(2,1)
mark(2,2)
mark(2,3)
mark(3,1)
mark(3,2)
mark(3,3)
```

Black's moves:

```
noop
```

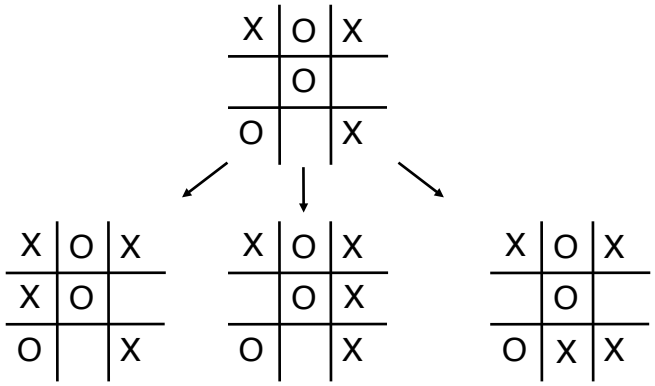
Given a state, like the one we just saw, a player can use the game description to compute the legal moves for each of the players. In this case, the white player can mark any of the nine cells. And the black player do nothing; in other words, it must execute the noop action.

State Update



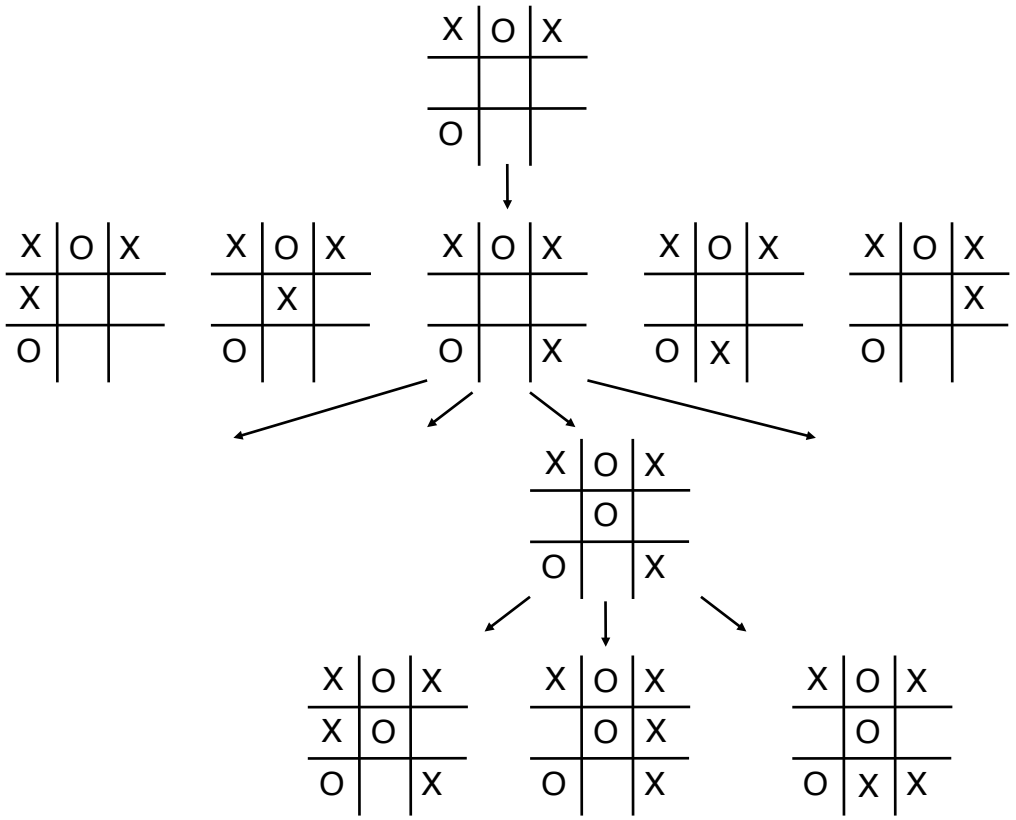
Given a state and the players' actions, a player can compute the next state using the update rules. In the case shown here, if the white player plays the mark(1,3) action in the initial state, the result is a state in which there is an X in the upper right corner.

Game Tree Expansion



One way for a player to decide on a course of action in a match is to use these two computations repeatedly to expand the game tree. Starting in a known state, it computes the legal actions for itself and its opponents, as previously discussed. For each combination of actions of the players, it simulates the actions to obtain the next state and thereby expand the tree. Here we see the TTT tree expanded one level.

Game Tree Search



Repeating this, a player can expand the tree to two levels, three levels, and so forth, until it encounters terminal states on every branch, such as the one here in the middle of the bottom row. By examining the various branches, it can choose the one that produces the best payoff. Of course, this choice depends on the move of the other players, and it must consider all possible opponent moves or make some assumptions about things that the other players will or will not do. In principle, this procedure allows a player to identify the best possible strategy to play the game.

Resource Limitations

Large state spaces

- ~5000 states in Tic-Tac-Toe
- $>10^{30}$ states in Chess

Limited Resources

- Memory
- Time (start clock, move clock)

Unfortunately, even in cases where there is a clearcut solution, the tree may be so large as to make it practically impossible for any player to expand the game tree. In TTT, there are just 5000 states, a manageable number; but there are more than 10^30 states in chess. Using this approach, the player would run out of time and memory long before finishing.

Incremental Search

Incremental Search

- Expand Game Graph incrementally
- As much as time allows
- Minimax/etc. evaluation on non-terminal states using an evaluation function of some sort

But how do we evaluate non-terminal states?

In traditional game playing, the rules are known in advance; and the programmer can invent game-specific evaluation functions. Not possible in GGP.

The alternative is to do incremental search, on each move expanding the tree as much as possible and then making a choice based on the apparent value of non-terminal states. In traditional game playing, where the rules are know in advance, the programmer can invent game-specific evaluation functions to help in this regard. For example, in chess, we know that states with higher piece count and greater board control are better than ones with less material or lower control. Unfortunately, it is not possible for a GGP programmer to invent such game-specific rules in advance, as the game's rules are not known until the game begins. The program must evaluate states for itself.

Guaranteed Evaluation Functions

Ideas

* Novelty with reversibility
* Goal-monotonic observables
* Bad states, useless moves

<insert your good idea here>

The good news is that there are some evaluation techniques that always work. For example, there is no harm preferring new states to states that have previously been seen, provided that there is a way to get back to the original states. Also, if a player is able to determine that some observable condition corresponds to distance from the goal, then it is good to minimize that quantity. Suppose the player were in a cave trying to get out. If it saw a brighter light in one tunnel than another, it might go for the brighter light. Finally, there are some states that can be determined to be bad even if other states are not known to be good. For example, stepping off the roof of a tall building is probably not the best way to get to the store (at least not in the real world).

Non-Guaranteed Evaluation Functions

Ideas

- Goal proximity (everyone)
- Maximize mobility (Barney Pell)
- Minimize opponent's mobility (Jim Clune)
- <insert your good idea here>

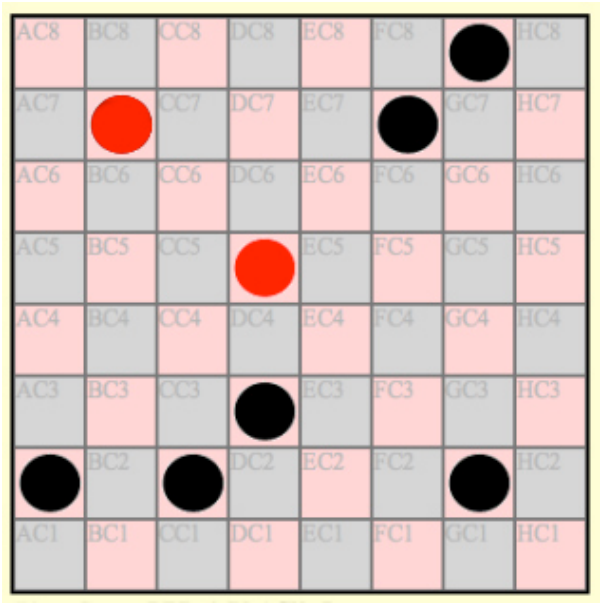
Another possibility is to use non-guaranteed heuristics. A number of such heuristics have been proposed over the years.

Goal proximity is an example. Proponents of this heuristic argue that, all other things being equal, it is a good idea to prefer states that are closer to goal states than states that are farther away. Distance here is usually judged by similarity between states, that is the number of facts in common in the descriptions of the two states.

Mobility is another general heuristic. Proponents argue that, all other things being equal, it is better to move to a state that affords the player greater mobility, that is more possible actions. Better than being boxed into a corner. Symmetrically, proponents of mobility argue that it is good to minimize the mobility of one's opponents.

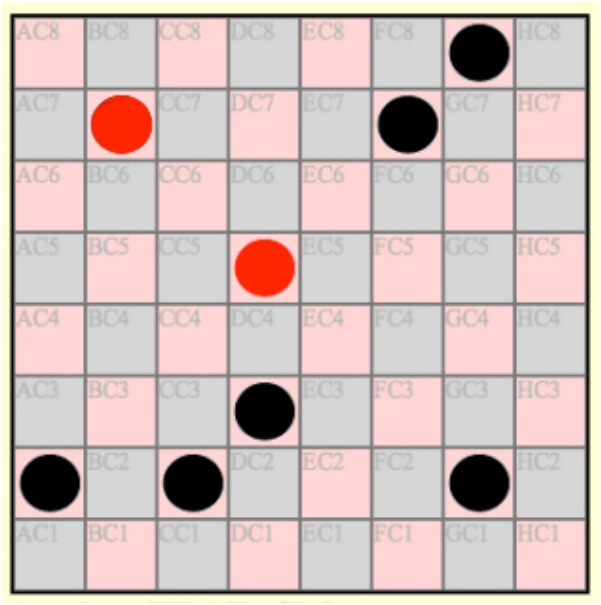
All of these heuristics have been shown to be effective in some games. Unfortunately, they are only heuristics. They frequently fail, sometimes with comical consequences.

GGP-06 Final - Cylinder Checkers



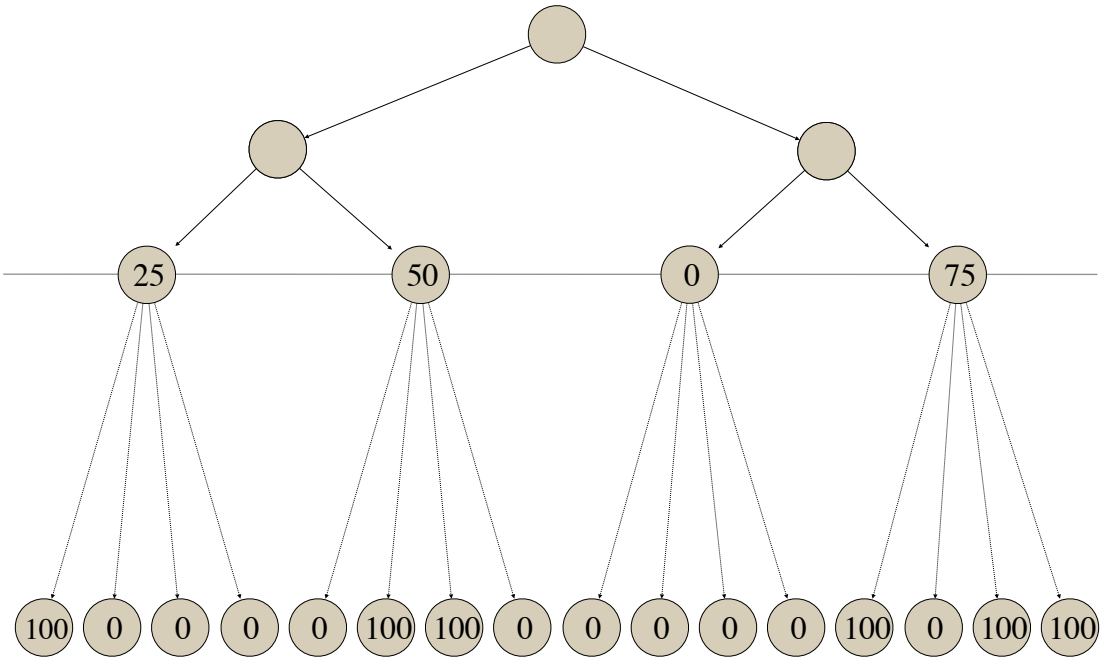
The final match of GGP-06 is an example. The game was cylinder checkers, i.e. checkers played on a cylinder. Recall that, in checkers, a player is permitted to move one of his ordinary pieces (pieces that are not kings) one square forward on each turn. Here red is moving from top to bottom and black is moving from bottom to top. If a piece is blocked by an opponent's player, he can "jump" that player if there is an empty square on the other side. Moreover, the player *must* make such a jump if one is available. The objective of the game is to take all or as many of the opponent's pieces as possible while preserving one's own pieces. Here is a snapshot of the game. It is red's turn to play. What should he do? And what do you think he did?

GGP-06 Final - Cylinder Checkers



Here's a hint. The player in this case was Cluneplayer, and it had decided, for some reason or other, that limiting the opponent's mobility was a good heuristic. If it were to move the rearmost piece, black would have multiple moves. However, if it were to move the piece in front, black would be forced to capture its piece. In other words, it would have at most one move. Clearly, moving the forward piece minimizes the opponent's mobility, so that is what Cluneplayer did. Actually, the whole match played out this way, with red giving black captures at every opportunity. It was sad to watch but also a little comical. The moral is that, while non-guaranteed heuristics are sometimes useful, they are not always useful.

Monte Carlo



48

An alternative to evaluation functions like these is Monte Carlo Search. The basic idea of is simple. The player expands the tree a few levels. Then, rather than using a local heuristic to evaluate a state, it makes some probes from that state to the end of the game by selecting random moves for all players. It sums up the total rewards for all such probes and divides by the number of probes to obtain an estimated utility for that state. It can then use these expected utilities in comparing states and selecting actions. Monte Carlo and its variants have proven highly successful in general game playing, and virtually every general game playing program today uses some variant of Monte Carlo search.

Metagaming

Metagaming is the process of reasoning about games and, by extension, game players and game playing.

Extremely broad definition:
- game design and game analysis
- games in general as well as specific games
- *what programmers do in creating specific / general players*

Done *offline*, i.e. during the start clock or between moves or in parallel with regular game play.

This discussion of game tree search and heuristics reveals just how difficult the GGP problem is. Monte Carlo works amazingly well, but it too breaks down badly in certain cases. Fortunately, there is another, complementary approach to general game playing that has tremendous power, and that is *metagaming*.

Metagaming is problem solving in the world of games. It involves reasoning about games and, by extension, game players and game playing. As stated, this is an extremely general definition. It includes both game design and game analysis. It includes reasoning about games in general as well as reasoning about specific games and specific matches of specific games. Significantly, it includes what programmers do in devising programs to play specific games as well as what programmers do in devising general game playing programs. Metagaming is usually done offline, during the brief period after a player receives the game rules and game play begins; or sometimes it is done in parallel with game tree search.

Metagaming Specifics

Differences from Game tree search
- more information
- less information, i.e. more general
- goal - create / optimize player to play games effectively

Techniques:
- Analysis of propositional nets and rule graphs
- Proofs using logic
- Compilation into machine language and/or FPGAs

In General Game Playing, we are primarily interested in those types of metagaming that can be automated. This raises the question of the distinction between ordinary game playing and metagaming. Can we distinguish the two? It is not that easy, but there is a difference. To begin with, ordinary game tree search can be viewed as a degenerate form of metagaming, one in which the metagamer must find the best actions for a specific role in a specific game starting in a specific state. By contrast, in some cases, metagaming sometimes involves information and goals that are different from the specifics of game tree search.

* To begin with, metagaming can take into account information other than the game description. For example, it might take into account its past experience. For example, in a round robin tournament where total return matters, it might select a different strategy than in an elimination ladder, where beating the opponent's score is what matters most.

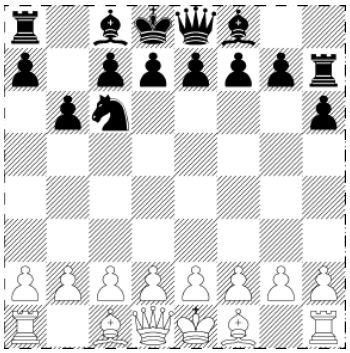
* Metagaming is also sometimes done with *less* information than is used in match play, e.g. without information about role, initial state, goals, or termination. As a result, metagaming is more general, deriving conclusions that apply across different games and different players.

* The goal of metagaming is broader than that of game tree search. It is not so much concerned with selecting the actions of a specific player in a specific game, but rather it is concerned with devising a game tree search program or optimizing an existing program to search the game tree (without actually searching the tree itself).

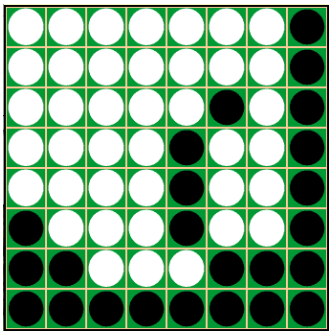
Whether or not the concept of automated metagaming can be distinguished from game tree search, there is no doubt that the concept is used to good effect in many general game playing programs.

Hodgepodge

Hodgepodge = Chess + Othello



Branching factor: *a*



Branching factor: *b*

Analysis of joint game:

Branching factor as given to players: $a*b$

Fringe of tree at depth n as given: $(a*b)^n$

Fringe of tree at depth n factored: a^n+b^n

One example of metagaming is game decomposition, also called factoring. Consider the game of Hodgepodge. Hodgepodge is actually two games glued together. Here we show chess and othello, but it could be any two games. One move in a joint game of Hodgepodge corresponds to one move in each of the two constituent games. Winning requires winning at least one of the two games while not losing the other. What makes Hodgepodge interesting is that it is factorable, that is it can be divided into two independent games. Realizing this can have dramatic benefit. To see this, consider the size of the game tree for hodgepodge. Suppose that one game tree has branching a and the other has branching factor b. Then the branching factor of the joint game is a times b, and the size of the fringe of the game tree at level n is (a*b)^n. However, the two games are independent. Moving in one subgame does not affect the state of the other subgame. So, the player really should be searching two smaller game trees, one with branching factor a and the other with branching factor b. In this way, at depth n, there would be only a^n+b^n states. This is a huge decrease in the size of the search space.

Reformulation Opportunities

Finding Interesting Structure in Games:
- Factoring, e.g Hodgepodge
- Bottlenecks, e.g. Triathalon
- Symmetry detection, e.g. Tic-Tac-Toe
- Dead State Removal

Trade-off - cost of finding structure vs savings
- *Sometimes* cost proportion to size of description
- *Sometimes* savings proportional to size of the game tree

Factoring is just one example of game reformulation. There are many others. For example, it is sometimes possible to find symmetries in games that cut down on search space. In some games, there are bottlenecks that allow for a different type of factoring. Consider, for example, a game made up of one or more subgames in which it is necessary to win one game before moving on to a second game. In such a case, there is no need to search to a terminal state in the overall game; it is sufficient to limit search to termination in the current subgame. These examples are extreme cases, but there are many simpler everyday examples of finding structure of this sort that can help in curtailing search.

The trick in metagaming is to analyze and/or reformulate a game without expanding the entire game tree. The interesting thing about general game playing is this - that sometimes the cost of analysis is proportional to the size of the description rather than the size of the game tree, as in teh examples we have just seen. In such cases, players can expend a little time and gain a lot in search savings.

Philosophical Remarks

General Game Playing is not a game

Critique: Game playing is *frivolous*.

Serious Applications:

- Enterprise Management
- Computational Law

Funders:

- Darpa
- SAP

As we shall see, GGP is an interesting application in its own right. It is intellectually engaging and more than a little fun. But it is much more than that. It serves an analog for applications of logic in other areas, such as business and law, science and engineering. More fundamentally, it raises questions about the nature of intelligence and serves as a laboratory in which to evaluate competing approaches to artificial intelligence.

Testbed for Theories of Intelligence

Characteristics of GGP
- game descriptions contain full information
- which determine optimal behavior

Useful for evaluating theories of intelligence
- effects of representation
- incompleteness of information
- resource bounds

More fundamentally, general game playing has value as a testbed for theories of intelligence. Game descriptions provide full information about a world and determine optimal strategies as a baseline for evaluating agent behavior. By its nature, the GGP setting can be used to evaluate problem solving strategies and by extension theories of intelligence, by taking into account representation, incompleteness of information, and resource bounds.

John McCarthy

*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. To make these statements will require little, if any, knowledge of the program or the previous knowledge of the advice taker.*

It was in 1958 that John McCarthy invented the concept of the "advice taker". The idea was simple. He wanted a machine that he could program by description. He would describe the intended environment and the desired goal, and the machine would use that information in determining its behavior. There would be no programming in the traditional sense. McCarthy presented his concept in a paper that has become a classic in the field of AI. ... READ

Ed Feigenbaum

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.*

An ambitious goal! But that was a time of high hopes and grand ambitions. The idea caught the imaginations of numerous subsequent researchers -- notably Bob Kowalski, the high priest of logic programming, and Ed Feigenbaum, the inventor of knowledge engineering. In a paper written in 1974, Feigenbaum gave his most forceful statement of McCarthy's ideal. ... READ

Robert Heinlein

computer/robot

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.

One final remark. Some have argued that the way to achieve intelligent behavior is through specialization. That may work so long as the assumptions one makes in building such systems are true. For general intelligence, however, general intellectual capabilities are needed, and such systems should be capable of performing well in a wide variety of tasks. To paraphrase the words of Robert Heinlein ... READ ... Those of us who are more interested in artificial intelligence than artificial insects agree with Heinlein.


GENERAL
GAME
PLAYING