General Game Playing
Metagaming

Michael Genesereth
Computer Science Department
Stanford University

Metagaming

Metagaming is match-independent game processing, i.e. game processing that is done independent of any particular opponent or any particular state.

Objective of metagaming - to optimize performance in playing specific matches of the game.

Usually done offline, i.e. during the *startclock* or *between moves or in parallel with regular game play*.

*What human programmers do in **creating** game players*

Headstart

Headstart

Headstart = game tree search during the start clock.

Ordinary PTS

```
function start (r,rs,sc,pc)
 {role = r;
  library = definemorerules([],rs.slice(1));
  roles = findroles(library);
  state = findinits(library);
  startclock = parseInt(sc);
  playclock = parseInt(pc);
  var reward = parseInt(findreward(role,state,library));
  tree = makenode(state,findcontrol(state,library),reward);
  return 'ready'}

function play (move)
 {if (move!==nil)
     {tree = subtree(move,tree); state = tree.state};
  if (findcontrol(state,library)!==role) {return false};
  var deadline = Date.now()+playclock*1000;
  while (Date.now()<deadline) {process(tree)};
  return selectaction(tree)}
```

PTS With Headstart

```
function start (r,rs,sc,pc)
 {role = r;
  library = definemorerules([],rs.slice(1));
  roles = findroles(library);
  state = findinits(library);
  startclock = parseInt(sc);
  playclock = parseInt(pc);
  var reward = parseInt(findreward(role,state,library));
  tree = makenode(state,findcontrol(state,library),reward);
  var deadline = Date.now()+startclock*1000;
  while (Date.now()<deadline) {process(tree)};
  return 'ready'}

function play (move)
 {if (move!==nil)
     {tree = subtree(move,tree); state = tree.state};
  if (findcontrol(state,library)!==role) {return false};
  var deadline = Date.now()+playclock*1000;
  while (Date.now()<deadline) {process(tree)};
  return selectaction(tree)}
```

Big Switch

Bigswitch

Bigswitch is using the startclock to analyze a game and using the results to *adjust* runtime play.

Could also be run in parallel with game play or during opponent's turn.

Analyzing Games

Deductive analysis based on rules

Run depth charges and do statistical analysis

Properties and Choices

Properties

- Single Player vs Multiple Players (Hunter vs Alquerque)
- Intermediate state values? (Alquerque vs Connect Four)
- Branching factor? (Connect Four vs Minorities)
- Depth? (Alquerque vs Nine Board)
- Cost of expanding? (Tic Tac Toe vs Skirmish)

Choices:

- Search Strategy (ID vs PTS)
- Selection Function (exploitation vs exploration)
- Evaluation Function (mobility, state values, depth charges)
- Opponent Model (minimax vs maximax)

Subgoal Ordering

Game Optimization

Game Optimizations with standard interpreter:

Logical Optimization, e.g. reordering subgoals

Objective: compute game tree faster

Game Optimizations with different interpreters:

Grounding

Symbolizing

Objective: compute game tree faster

Game Reformulation:

Pruning game trees

Decomposing games into independent subgames

*Objective: decrease the **size** of the game tree*

Logical Optimization

Logical Optimization is the transformation of a logic program into an equivalent logical program with better computational properties.

Types:

→ Subgoal Ordering

Subgoal Elimination

Rule Elimination

Materialization

Simplification

Example

```
  SEND
+MORE
-----
MONEY
```

Solution

```
digit(1)      digit(6)
digit(2)      digit(7)
digit(3)      digit(8)
digit(4)      digit(9)
digit(5)      digit(0)

solution(S,E,N,D,M,O,R,Y) :-
  digit(S) & digit(E) & digit(N) & digit(D) &
  digit(M) & digit(O) & digit(R) & digit(Y) &
  S!=0 & E!=S & N!=S & N!=E & D!=S & D!=E & D!=N &
  M!=0 & M!=S & M!=E & M!=N & M!=D &
  O!=S & O!=E & O!=N & O!=D & O!=M &
  R!=S & R!=E & R!=N & R!=D & R!=M & R!=O &
  Y!=S & Y!=E & Y!=N & Y!=D & Y!=M & Y!=O & Y!=R
  evaluate(S*1000+E*100+N*10+D,U) &
  evaluate(M*1000+O*100+R*10+E,V) &
  evaluate(M*10000+O*1000+N*100+E*10+Y,W) &
  evaluate(plus(U,V),W)
```

Analysis

Data

```
digit(1)      digit(6)
digit(2)      digit(7)
digit(3)      digit(8)
digit(4)      digit(9)
digit(5)      digit(0)
```

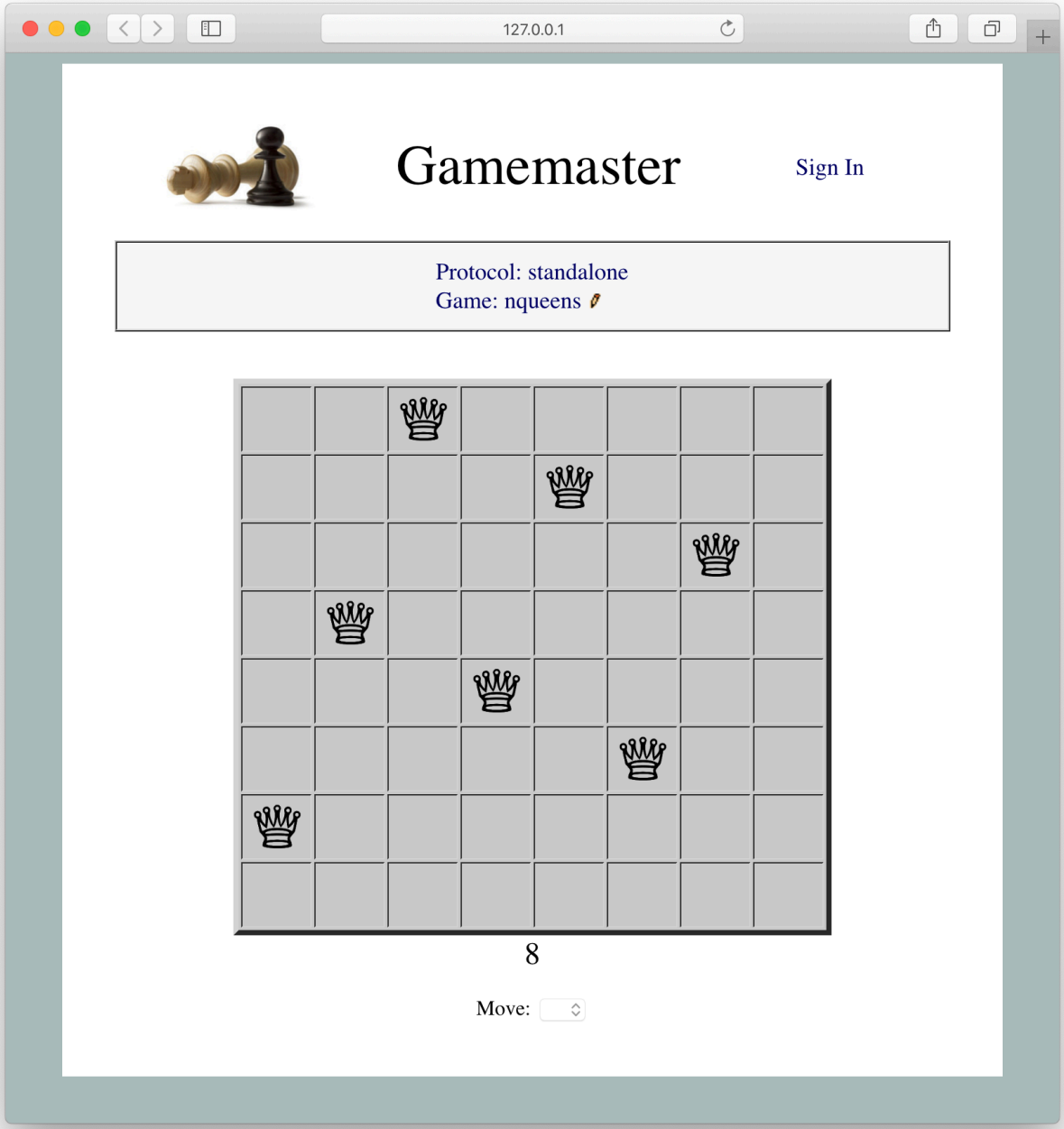
Rule

```
goal(S,E,N,D,M,O,R,Y) :-
  digit(S) & digit(E) & digit(N) & digit(D) &
  digit(M) & digit(O) & digit(R) & digit(Y) & ...
```

Analysis

10x10x10x10x10x10x10x10 = 10^8 = 100,000,000 instances

8 Queens


127.0.0.1
Gamemaster
Sign In
Protocol: standalone
Game: nqueens
8
Move:

Goal Description

```
digit(1)      digit(5)
digit(2)      digit(6)
digit(3)      digit(7)
digit(4)      digit(8)

base(cell(X,Y)) :- digit(X) & digit(Y)

solution(C1,C2,C3,C4,C5,C6,C7,C8) :-
  cell(C1) & cell(C2) & cell(C3) & cell(C4) &
  cell(C5) & cell(C6) & cell(C7) & cell(C8) &
  ~attacks(C1,C2) & ... & ~attacks(C7,C8)
```

Analysis

64 x 64 x 64 x 64 x 64 x 64 x 64 x 64 =
281,474,976,710,656 instances

Subgoal Ordering

Original Rule

```
solution(X,Y) :- p(X) & r(X,Y) & q(X)
```

Reformulation

```
solution(X,Y) :- p(X) & q(X) & r(X,Y)
```

Analysis

Original Rule

```
solution(X,Y) :- p(X) & r(X,Y) & q(X)
```

$$(n\text{^}2 + 2n) + n*((n\text{^}2 + 2n) + n*(n\text{^}2 + 2n)) =$$
$$n\text{^}4 + 3n\text{^}3 + 3n\text{^}2 + 2n$$

Reformulation

```
solution(X,Y) :- p(X) & q(X) & r(X,Y)
```

$$(n\text{^}2 + 2n) + n*((n\text{^}2 + 2n) + 1*(n\text{^}2 + 2n)) =$$
$$2n\text{^}3 + 5n\text{^}2 + 2n$$

Subgoal Ordering Technique

Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove *first* subgoal, add to new query, and repeat.

Output the new query.

Example:

```
solution(X,Y) :- p(X) & r(X,Y) & q(X)
solution(X,Y) :-
```

Subgoal Ordering Technique

Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove *first* subgoal, add to new query, and repeat.

Output the new query.

Example:

```
solution(X,Y) :- p(X) & r(X,Y) & q(X)
solution(X,Y) :- p(X)
```

Subgoal Ordering Technique

Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove *first* subgoal, add to new query, and repeat.

Output the new query.

Example:

```
solution(X,Y) :- p(X) & r(X,Y) & q(X)
solution(X,Y) :- p(X) & q(X)
```

Subgoal Ordering Technique

Accept query rule as input.

(1) Create new query with head of input and empty body.

(2) Iterate through subgoals. On encountering one with all variables bound in subgoals of new query, add to new query and remove from original query. If none found, remove *first* subgoal, add to new query, and repeat.

Output the new query.

Example:

```
solution(X,Y) :- p(X) & r(X,Y) & q(X)
solution(X,Y) :- p(X) & q(X) & r(X,Y)
```

Computational Analysis

Data

```
digit(1)      digit(6)
digit(2)      digit(7)
digit(3)      digit(8)
digit(4)      digit(9)
digit(5)      digit(0)
```

Rule

```
solution(S,E,N,D,M,O,R,Y) :-
  digit(S) & digit(E) & digit(N) & digit(D) &
  digit(M) & digit(O) & digit(R) & digit(Y) & ...
```

Analysis

10x10x10x10x10x10x10x10 = 10^8 = 100,000,000 cases

111,111,110 unifications

Running time ~ minutes

Subgoals Reordered

```
solution(S,E,N,D,M,O,R,Y) :-
  digit(S) & S!=0 &
  digit(E) & E!=S &
  digit(N) & N!=S & N!=E &
  digit(D) & D!=S & D!=E & D!=N &
  digit(M) & M!=0 & M!=S & M!=E & M!=N & M!=D &
  digit(O) & O!=S & O!=E & O!=N & O!=D & O!=M &
  digit(R) & R!=S & R!=E & R!=N & R!=D &
             R!=M & R!=O &
  digit(Y) & Y!=S & Y!=E & Y!=N & Y!=D &
             Y!=M & Y!=O & Y!=R &
  evaluate(S*1000+E*100+N*10+D,X) &
  evaluate(M*1000+O*100+R*10+E,Y) &
  evaluate(M*10000+O*1000+N*100+E*10+Y,Z) &
  evaluate(plus(X,Y),Z)
```

Computational Analysis

Rule

```
solution(S,E,N,D,M,O,R,Y) :-
  digit(S) & S!=0 &
  digit(E) & E!=S &
  digit(N) & N!=S & N!=E &
  digit(D) & D!=S & D!=E & D!=N &
  digit(M) & M!=0 & M!=S & M!=E & M!=N & M!=D &
  digit(O) & O!=S & O!=E & O!=N & O!=D & O!=M &
  digit(R) & R!=S & R!=E & R!=N & R!=D &
             R!=M & R!=O &
  digit(Y) & Y!=S & Y!=E & Y!=N & Y!=D &
             Y!=M & Y!=O & Y!=R & ...
```

Analysis

10x9x8x7x6x5x4x3 = 1,814,400 cases

5,989,558 / 7,921,010 unifications

Interpreted ~ 40 seconds

Computational Analysis

Rule

```
goal(S,E,N,D,M,O,R,Y) :-
  digit(D) &
  digit(E) & mutex(E,D) &
  digit(Y) & mutex(Y,D,E) &
             evaluate(remainder(plus(D,E),10),Y) &
  digit(S) & distinct(S,0) & mutex(S,D,E,Y) &
  digit(N) & mutex(N,D,E,Y,S) &
  digit(M) & distinct(M,0) & mutex(M,D,E,Y,S,N) &
  digit(O) & mutex(O,D,E,Y,S,N,M) &
  digit(R) & mutex(R,D,E,Y,S,N,M,O) & ...
```

Analysis

438,728 / 7,921,010 unifications

Interpreted ~ 3 seconds

Subgoal Removal

Original Rule:

```
solution(X,Y) :- p(X,Y) & q(Y) & q(Z)
```

Equivalent Reformulation:

```
solution(X,Y) :- p(X,Y) & q(Y)
```

Not covered in this course.

Rule Removal

Redundant Rules:

```
solution(X) :- p(X,b) & q(b) & r(Z)
solution(X) :- p(X,Y) & q(Y) & r(Z)
```

Non-Redundant Rules:

```
solution(X) :- p(X,b) & q(b) & r(Z)
solution(X) :- p(X,Y) & q(Y) & r(c)
```

Not covered in this course.

Materialization

Materialization is the process of precomputing a view relation and storing the result so that subsequent queries can be answered by lookup rather than computation.

Advantage: Look up is often much cheaper than computation. Efficiency advantage if the relation is queried often.

Disadvantages: Relation may be *large* (e.g. third generation relatives of US population) or even *infinite* (e.g. the set of all primes) and/or *expensive* to compute (e.g. factors of a large number) and may be accessed only infrequently.

Not covered in this course.

Basic Idea

Simplification is the process of substituting static facts into rules and eliminating true subgoals and eliminating rules with false subgoals.

Not covered in this course.

Metagamers

Not Secure — gamemaster.stanford.edu

Gamemaster

genesereth
Sign Out

Developer

Metagamer	Action
grounder	Launch
materializer	Launch
metagamer	Launch
optimizer	Launch
pruner	Launch
simplifier	Launch
symbolizer	Launch
viper	Launch

Optimizer

Not Secure — gamemaster.stanford.edu

Gamemaster

General Game Playing

Optimizer

Game Description:

```
s(X,Y,Z) :- q(X,Y) & r(Y,Z) & p(X)

h(X,Y,Z) :- g(X,Y) & g(Y,Z) & g(X,W)

j(Y) :- g(X,Y) & p(Y)
j(Y) :- g(X,Y)

p(a)
p(b)
p(c)

g(a,a)
g(a,b)
g(b,c)
g(b,d)
g(c,a)
g(c,b)
g(d,c)
```

Order Subgoals | Prune Subgoals | Prune Rules

Analysis:

```
s(X,Y,Z) :-
  q(X,Y) &
  p(X) &
  r(Y,Z)

h(X,Y,Z) :-
  g(X,Y) &
  g(Y,Z) &
  g(X,W)

j(Y) :-
  g(X,Y) &
  p(Y)

j(Y) :- g(X,Y)

p(a)
p(b)
```

Including Optimization Code

```
<script src='http://epilog.stanford.edu/javascript/
epilog.js'></script>

<script src='http://gamemaster.stanford.edu/javascript/
localstorage.js'></script>

<script src='http://gamemaster.stanford.edu/metagaming/
optimizer.js'></script>

<script src='http://gamemaster.stanford.edu/gameplaying/
pts.js'></script>

<script src='http://gamemaster.stanford.edu/reasoning/
general.js'></script>
```

Using Optimization Code

```
function start (r,rs,sc,pc)
 {role = r;
  var rules = rs.slice(1);
  library = definemorerules([],rules);
  roles = findroles(library);
  state = findinits(library);
  startclock = numberize(sc);
  playclock = numberize(pc);
  return 'ready'}

function start (r,rs,sc,pc)
 {role = r;
  var rules = fixrules(rs.slice(1));
  library = definemorerules([],rules);
  roles = findroles(library);
  state = findinits(library);
  startclock = numberize(sc);
  playclock = numberize(pc);
  return 'ready'}
```

Other

Game Optimization

Game Optimizations with standard interpreter:

Logical Optimization, e.g. reordering subgoals

Objective: compute game tree faster

Game Optimizations with different interpreters:

Grounding

Symbolizing

Objective: compute game tree faster

Game Reformulation:

Pruning game trees

Decomposing games into independent subgames

*Objective: decrease the **size** of the game tree*

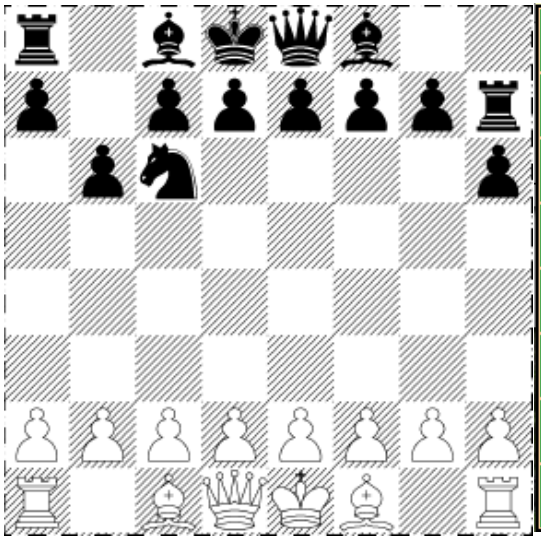
Grounding (Week 7)

```
row(M,R) :- cell(M,1,R) & cell(M,2,R) & cell(M,3,R)
col(N,R) :- cell(1,N,R) & cell(2,N,R) & cell(3,N,R)
```

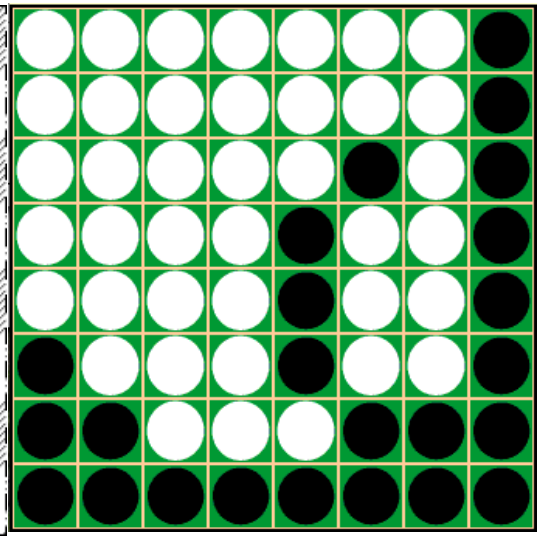
```
row(1,x) :- cell(1,1,x) & cell(1,2,x) & cell(1,3,x)
row(2,x) :- cell(2,1,x) & cell(2,2,x) & cell(2,3,x)
row(3,x) :- cell(3,1,x) & cell(3,2,x) & cell(3,3,x)
row(1,o) :- cell(1,1,o) & cell(1,2,o) & cell(1,3,o)
row(2,o) :- cell(2,1,o) & cell(2,2,o) & cell(2,3,o)
row(3,o) :- cell(3,1,o) & cell(3,2,o) & cell(3,3,o)

col(1,x) :- cell(1,1,x) & cell(2,1,x) & cell(3,1,x)
col(2,x) :- cell(1,2,x) & cell(2,2,x) & cell(3,2,x)
col(3,x) :- cell(1,3,x) & cell(2,3,x) & cell(3,3,x)
col(1,o) :- cell(1,1,o) & cell(2,1,o) & cell(3,1,o)
col(2,o) :- cell(1,2,o) & cell(2,2,o) & cell(3,2,o)
col(3,o) :- cell(1,3,o) & cell(2,3,o) & cell(3,3,o)
```

Game Reformulation (Week 8)

Hodgepodge = Chess + Othello



Branching factor: *a*



Branching factor: *b*

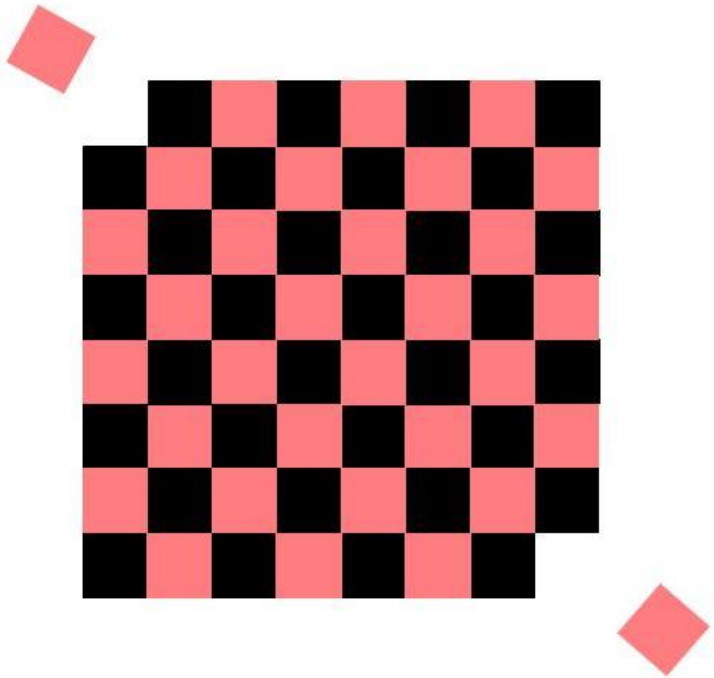
Analysis of joint game:

- 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

Symbolic Reasoning

Symbolic Reasoning

Automatic Theorem Proving



Not covered in this course.

Automatic Programming

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

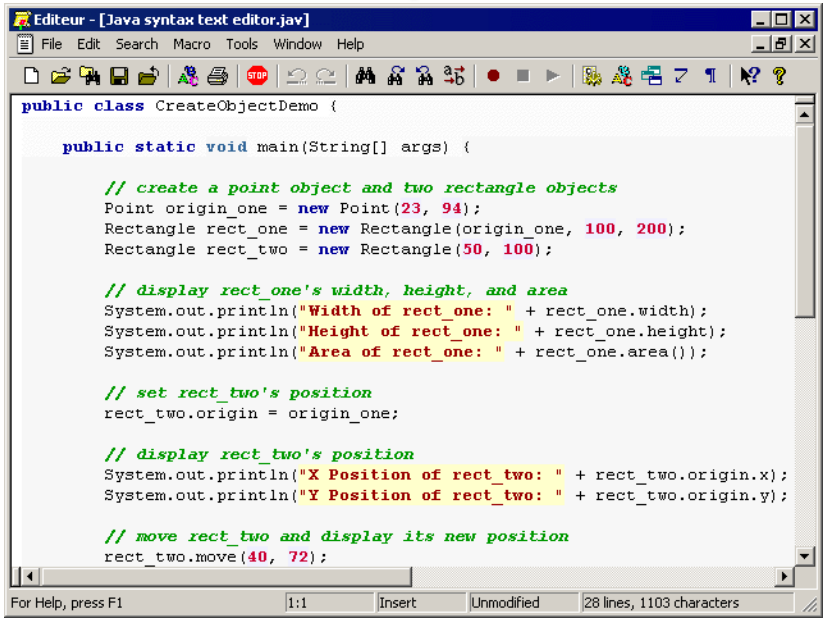
Not covered in this course.

Automatic Programming

```
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)) &
    (M#J | N#K)

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

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


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)
```

```
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))

draw :- ~line(x) &
        ~line(o)
```



Algorithmic Expertise

Knuth in a Box

General Game Playing

General Game Playing
HELLESPONT ALHAMBRA
Mike

GENERAL
GAME
PLAYING