Google AI Challenge: Planet Wars

Gábor Melis
gmelis@franz.com

Franz Inc.

January, 2011
About this presentation

1. Planet wars intro
2. Implementation/AI
3. Meta ramblings

<table>
<thead>
<tr>
<th>Rank</th>
<th>Username</th>
<th>Country</th>
<th>Organization</th>
<th>Language</th>
<th>Elo Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bocs/macko</td>
<td></td>
<td>Other</td>
<td>Lisp</td>
<td>3765</td>
</tr>
<tr>
<td>2</td>
<td><em>jouri</em></td>
<td></td>
<td>Other</td>
<td>C++</td>
<td>3665</td>
</tr>
<tr>
<td>3</td>
<td>Slin:-</td>
<td></td>
<td>Lund University</td>
<td>Java</td>
<td>3524</td>
</tr>
<tr>
<td>4</td>
<td>Astek</td>
<td></td>
<td>Other</td>
<td>C#</td>
<td>3501</td>
</tr>
<tr>
<td>5</td>
<td>jimrogerz</td>
<td></td>
<td>Microsoft</td>
<td>C#</td>
<td>3500</td>
</tr>
<tr>
<td>6</td>
<td>Account</td>
<td></td>
<td>Other</td>
<td>C++</td>
<td>3498</td>
</tr>
<tr>
<td>7</td>
<td>george</td>
<td></td>
<td>Other</td>
<td>C++</td>
<td>3494</td>
</tr>
<tr>
<td>8</td>
<td>GreenTea</td>
<td></td>
<td>Dnipropestrovsk National University</td>
<td>Java</td>
<td>3489</td>
</tr>
<tr>
<td>9</td>
<td>asavis</td>
<td></td>
<td>Other</td>
<td>Java</td>
<td>3480</td>
</tr>
<tr>
<td>10</td>
<td>bix0/4ever</td>
<td></td>
<td>Other</td>
<td>Java</td>
<td>3476</td>
</tr>
<tr>
<td>11</td>
<td>protocoloscon</td>
<td></td>
<td>Rukes Band</td>
<td>C++</td>
<td>3469</td>
</tr>
<tr>
<td>12</td>
<td>dmj111</td>
<td></td>
<td>Other</td>
<td>Python</td>
<td>3467</td>
</tr>
<tr>
<td>13</td>
<td>davidfluo</td>
<td></td>
<td>Other</td>
<td>Python</td>
<td>3463</td>
</tr>
<tr>
<td>14</td>
<td>Raschi</td>
<td></td>
<td>Other</td>
<td>Java</td>
<td>3459</td>
</tr>
<tr>
<td>15</td>
<td>BaronTrezo</td>
<td></td>
<td>Rukes Band</td>
<td>C++</td>
<td>3441</td>
</tr>
<tr>
<td>16</td>
<td>thedreamer</td>
<td></td>
<td>Other</td>
<td>C++</td>
<td>3439</td>
</tr>
<tr>
<td>17</td>
<td>wagstaff</td>
<td></td>
<td>Other</td>
<td>Haskell</td>
<td>3431</td>
</tr>
<tr>
<td>18</td>
<td>medrimonia</td>
<td></td>
<td>University of Bordeaux 1</td>
<td>Python</td>
<td>3424</td>
</tr>
<tr>
<td>19</td>
<td>smloh1</td>
<td></td>
<td>Other</td>
<td>Java</td>
<td>3422</td>
</tr>
<tr>
<td>19</td>
<td>shengas</td>
<td></td>
<td>Other</td>
<td>Python</td>
<td>3422</td>
</tr>
</tbody>
</table>
2nd Google AI Challenge: Planet Wars (http://ai-contest.com/)
Couple of thousand contestants
Several supported programming languages (C++, Python, Java, Lisp, Go, etc)
Simple real-time strategy game

Agile and effective tools are needed.
Planet Wars: Rules

- enemy ships cancel each other out in battle
- planets produce some ships per turn
- neutral planets: short term sacrifice for long term gain
Planet Wars: Stealing

Taking over a neutral planet costs as many ships as there are defenders.
It is an oft used tactic to wait for the enemy to take the neutral, lose ships to neutral forces, and then take the planet from him on the next turn.
If ships stay put until they are needed for defense or attack, then they may be too far from the action when they are finally needed.
Planet Wars: Multi-planet moves

By combining forces of multiple planets the target planet can be taken earlier or defended later.
Implementation: Difficulties

- position evaluation
- practically unbounded number of possible moves
- how to test playing strength
Future is a possible sequence of states of a planet. In the simplest case the future is calculated from ships already en route in the game.

;;; A future is a particular sequence of states of a planet. It’s represented by an OWNERS and a N-SHIPS array.
(defclass future ()
  ((planet :initarg :planet :reader planet)
   (owners :initarg :owners :reader owners)
   (n-ships :initarg :n-ships :reader n-ships)
   ;; Number of ships player 2 lost when attacking neutrals minus the number of ships player 1 lost when attacking neutrals in this future.
   (balance :initarg :balance :reader balance)))
strength is a piecewise linear function of time
assume that there are no hidden changepoints
score: difference of accumulated growths
### Full attack lemma

Assuming that there are no neutral planets and Player 2 can take none of the planets of Player 1 when both players continuously send all possible ships to the contested planet, then Player 2 can take none of the planets of Player 1 even if allowed to attack multiple planets simultaneously in any pattern.

- Is this even true?
- In any case full attack future based evaluation is extremely useful.
Implementation: Move generation

- A smallish number of candidate moves must be selected.
- Moves are assembled from per-planet steps.
- A step is a set of orders targeting the same planet.
Implementation: Steps

- the *need* of a planet is the number of ships per turn needed to take over or defend that planet
- we try to satisfy the need of the target planet from the *surpluses* of friendly planets
- once we have steps for all planets they are scored by the normal evaluation function and the most promising ones combined into a composite move (subject to validity)
Implementation: Surplus

- try to control non-linearity
- most notable non-linearity is at ownership changes
- definition of surplus:
  The surplus of player P at planet A at time t is the number of ships that can be sent away on that turn from the defending army without:
  - making any scheduled order from planet A invalid
  - causing the planet to be lost anytime after that (observing only the fleets already in space)
  - bringing an imminent loss closer in time
Implementation: Redistribution

Just a small tweak to an extremely simple scoring function:

;;; The score of a future (of a planet) is simply the difference of
;;; growths captured by the players adjusted by the balance of the
;;; future (that is, taking into account the ships lost when capturing
;;; neutrals).
;;; Give a very slight positional penalty every turn for every enemy
;;; ship. When FUTURE is a FULL-ATTACK-FUTURE then this has the effect
;;; of preferring positions where the friendly ships are near the
;;; enemy.
(defun score (future player)
  (let ((owners (owners future))
        (n-ships-per-turn (n-ships future))
        (growth (growth (planet future)))
        (score (* (player-multiplier player) (balance future)))
        (opponent (opponent player)))
    (dotimes (i (length owners))
      (let ((owner (aref owners i))
            (n-ships (aref n-ships-per-turn i)))
        (cond ((= owner player)
               (incf score growth))
              ((= owner opponent)
               (decf score growth))
              (when (= player 1)
               (decf score (* 0.0000000000001d0
                             (- (the fixnum *n-turns-till-horizon*) i)
                             n-ships)))))
  score))
the neutral planets are the blind spot of the position evaluator
if the bot cannot take and keep a high growth planet it may go and take a low growth one leaving the first one to the opponent
this can lead to quick losses
solution: alpha-beta
but there are dangers ...
Questions? (first round)

Stay tuned.
Meta: Search

- A walk on states of solution space (often need to record states too)
  - Guided by heuristics in non-trivial cases
- States get evaluated
there are practically infinite possible actions to take
  - a very good move generator is needed
  - fast evaluation of moves is needed
Meta: Development as search

- Basics
  - version control (git)
  - unit testing
- Evaluation
  - understand what and why the bot does
  - how much does it lose?
    - test playing strength
  - fix all bugs before moving on Often hard to distinguish genuine bugs from algorithmic weaknesses.
- Move generator
  - why does it lose?
    - analyze lots of games
  - greedy heuristic (good for testing)
Meta: Levels of development as search

- With finite memory, information about visited states is lost. Danger of endless loops, making no progress.
  1. think quickly just jotting down main ideas in a few words
  2. talk to your rubber duck
  3. expand on ideas until “executing” them in head
  4. think coding, code thinking

- keep a record of progress (org-mode, version control, etc)
Meta: Why Lisp?

- code can be refactored quickly
- no risk of having to rewrite it in another language to speed it up
- faster testing, debugging in interactive development environment
- those who can test more ideas have a big advantage
- and also those who can place more useful bugs in the code
- no, I haven’t written a DSL
Meta: What to pack for a space war?

- much time and energy sources
- effective time management
- a good notepad
- one pack of meta-heuristics
- a heap of parens
Still hungry for more?

- source code repository http://quotenil.com/git/?p=planet-wars.git
- the code is tested on Linux with these Common Lisp implementations:
  - SBCL (http://sbcl.org)
- contest web site: http://ai-contest.com
Questions? (second round)

PROGRAMMING
You're doing it completely wrong.