# Google AI Challenge: Planet Wars

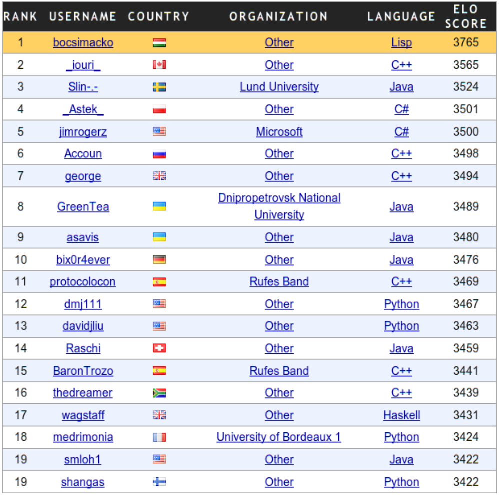
Gábor Melis
gmelis@franz.com

Franz Inc.

January, 2011

# About this presentation

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

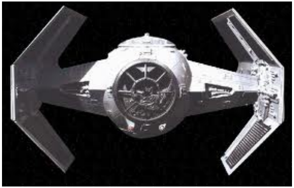


| RANK | USERNAME | COUNTRY | ORGANIZATION | LANGUAGE | ELO SCORE |
|---|---|---|---|---|---|
| 1 | bocsimacko | 🇭🇺 | Other | Lisp | 3765 |
| 2 | _iouri_ | 🇨🇦 | Other | C++ | 3565 |
| 3 | Slin-.- | 🇸🇪 | Lund University | Java | 3524 |
| 4 | _Astek_ | 🇵🇱 | Other | C# | 3501 |
| 5 | jimrogerz | 🇺🇸 | Microsoft | C# | 3500 |
| 6 | Accoun | 🇷🇺 | Other | C++ | 3498 |
| 7 | george | 🇬🇧 | Other | C++ | 3494 |
| 8 | GreenTea | 🇺🇦 | Dnipropetrovsk National University | Java | 3489 |
| 9 | asavis | 🇺🇦 | Other | Java | 3480 |
| 10 | bix0r4ever | 🇩🇪 | Other | Java | 3476 |
| 11 | protocolocon | 🇪🇸 | Rufes Band | C++ | 3469 |
| 12 | dmj111 | 🇺🇸 | Other | Python | 3467 |
| 13 | davidjliu | 🇺🇸 | Other | Python | 3463 |
| 14 | Raschi | 🇨🇭 | Other | Java | 3459 |
| 15 | BaronTrozo | 🇪🇸 | Rufes Band | C++ | 3441 |
| 16 | thedreamer | 🇿🇦 | Other | C++ | 3439 |
| 17 | wagstaff | 🇬🇧 | Other | Haskell | 3431 |
| 18 | medrimonia | 🇫🇷 | University of Bordeaux 1 | Python | 3424 |
| 19 | smloh1 | 🇺🇸 | Other | Java | 3422 |
| 19 | shangas | 🇫🇮 | Other | Python | 3422 |

# (-o-) With parens into spacewar (-o-)

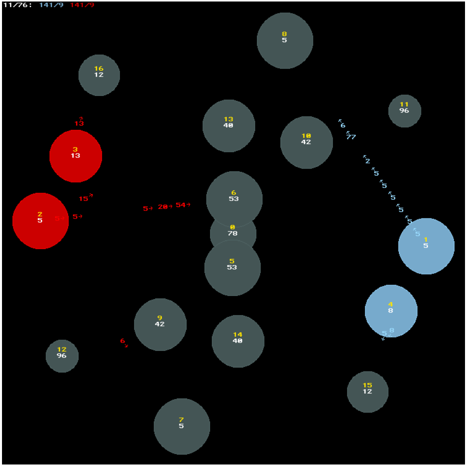
- 2nd Google AI Challenge: Planet Wars (http://ai-contest.com/)
- Couple of thousand contestants
- Several supported programming languages (C++, Python, Java, Lisp, Go, etc)
- Simple real-time strategy game

Agile and effective tools are needed.



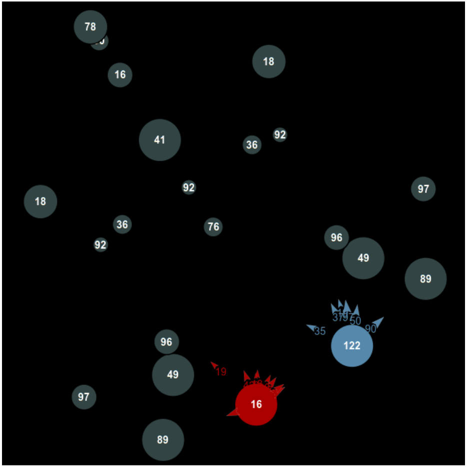
# Planet Wars: Rules

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



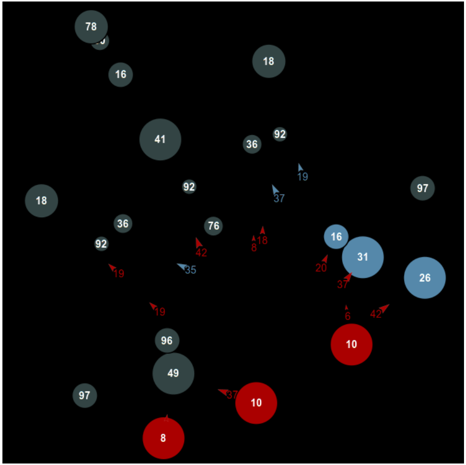
# Planet Wars: Stealing

Taking over a neutral planet costs as many ships as there are defenders.



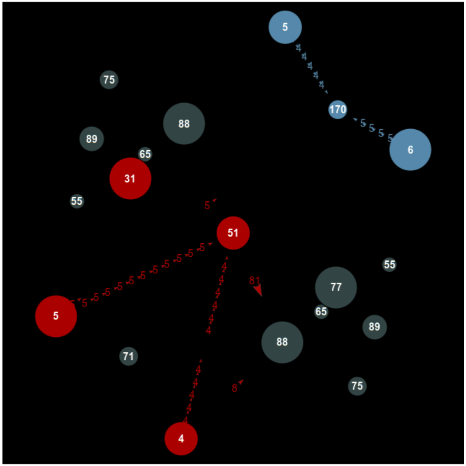
# Planet Wars: Stealing 2

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.



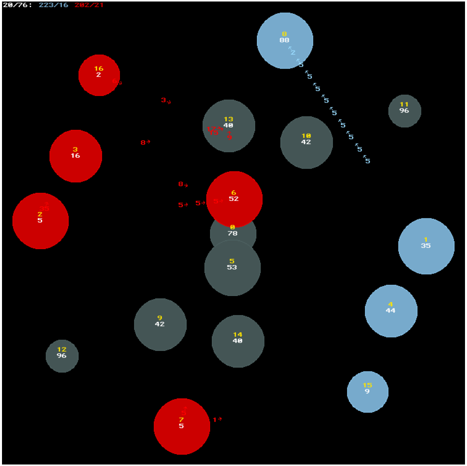
# Planet Wars: Redistribution

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

# Implementation: Future

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)))
```

# Implementation: Future based evaluation

- strength is a piecewise linear function of time
- assume that there are no hidden changepoints
- score: difference of accumulated growths

# Implementation: Full attack evaluation

## Full attack lemma

Assuming that there are no neutral planets and Player 2 can take none of the planets of Player 1 when both player 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 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 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 (* 0.000000000001d0
                                (- (the fixnum *n-turns-till-horizon*) i)
                                n-ships)))))))
    score))
```

# Implementation: Alpha-beta?

- 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

# Meta: Solution space

- 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

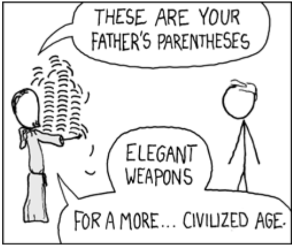
- Basics
  - version control (git)
  - unit testing
- Evaluation
  - understand what and why the bot does
  - how much does it lose?
    - test playing strength
  - fix all bugs before moving on Often hard to distinguish genuine bugs from algorithmic weaknesses.
- Move generator
  - *why* does it lose?
    - 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


THESE ARE YOUR FATHER'S PARENTHESES
ELEGANT WEAPONS
FOR A MORE... CIVILIZED AGE.

# 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:
  - Allegro CL (Free Express Edition: http://www.franz.com/downloads/clp/survey)
  - SBCL (http://sbcl.org)
- contest web site: http://ai-contest.com

# Questions? (second round)