CS501 – Artificial Intelligence, Fall 2007

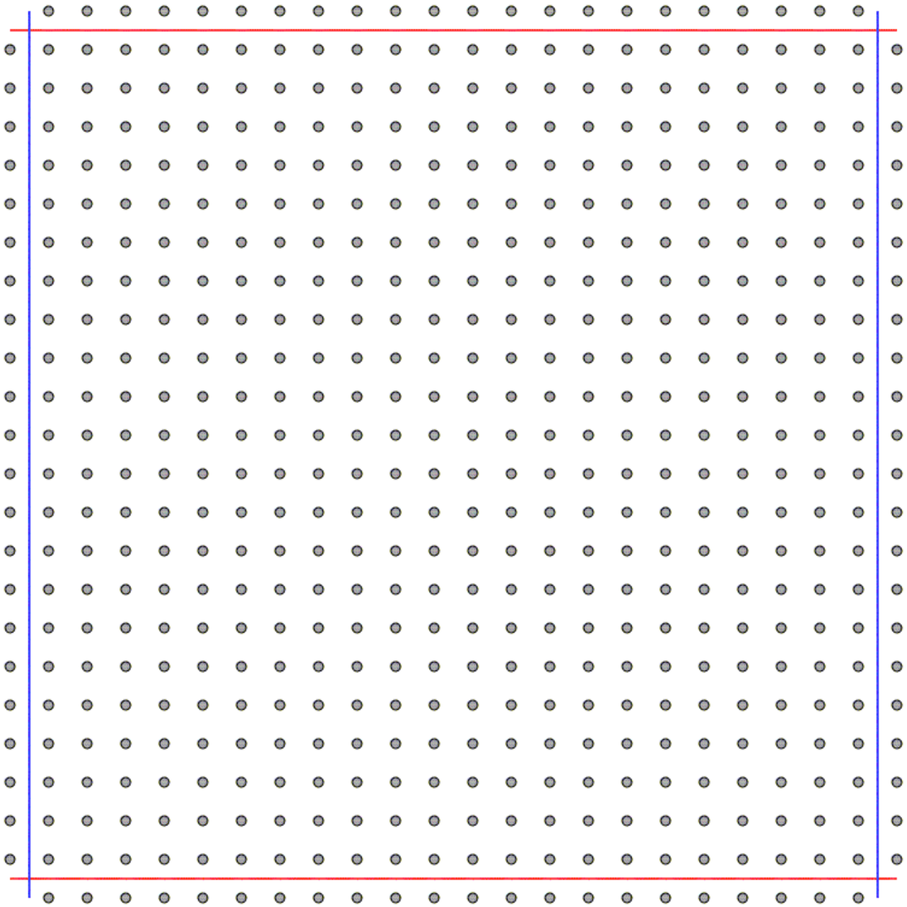
"TwixTalicious"

THANH DANG LANFRANCO MUZI ALEXANDER ROSS

Portland State University, November 26th, 2007

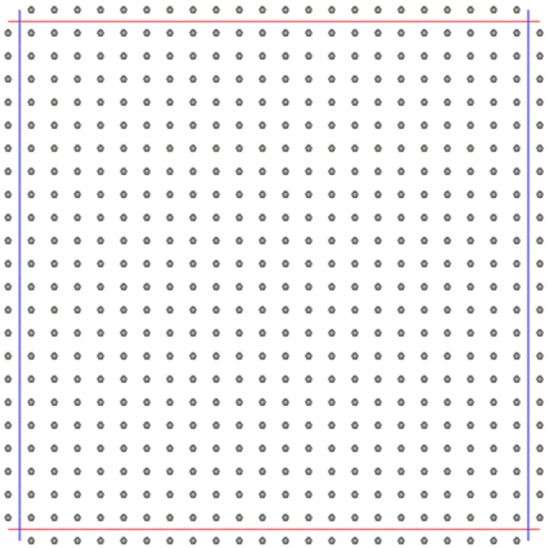
TwixT

- Two players (Red and Blue) on 24×24 grid of nodes



TwixT – rules of the game

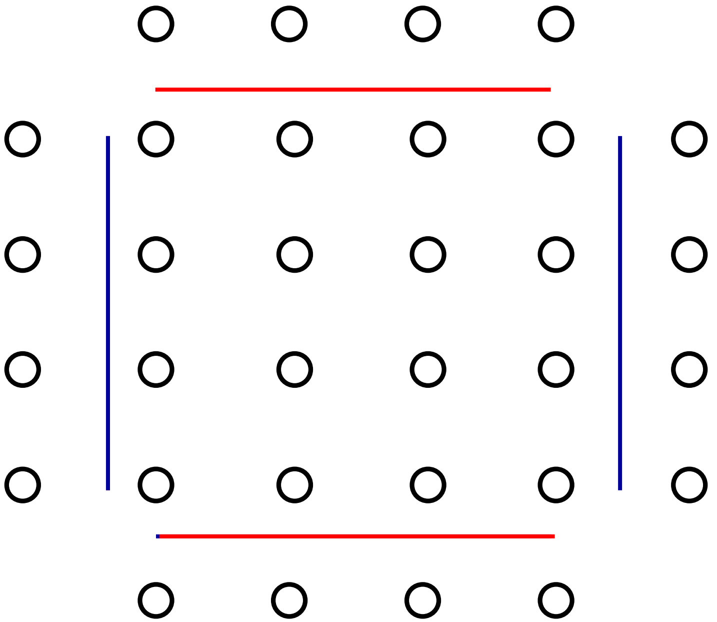
- Topmost and bottommost rows reserved for Red
- Leftmost and rightmost rows reserved for Blue
- At each move, player claims a node



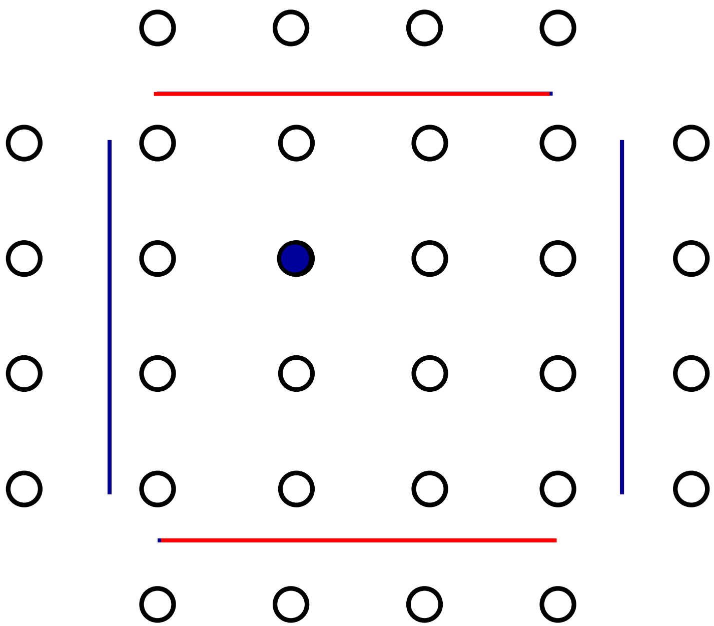
- Two nodes owned by one player may be connected if they are a knight's move apart

Goal: connect your reserved rows of the board with a path of connected nodes

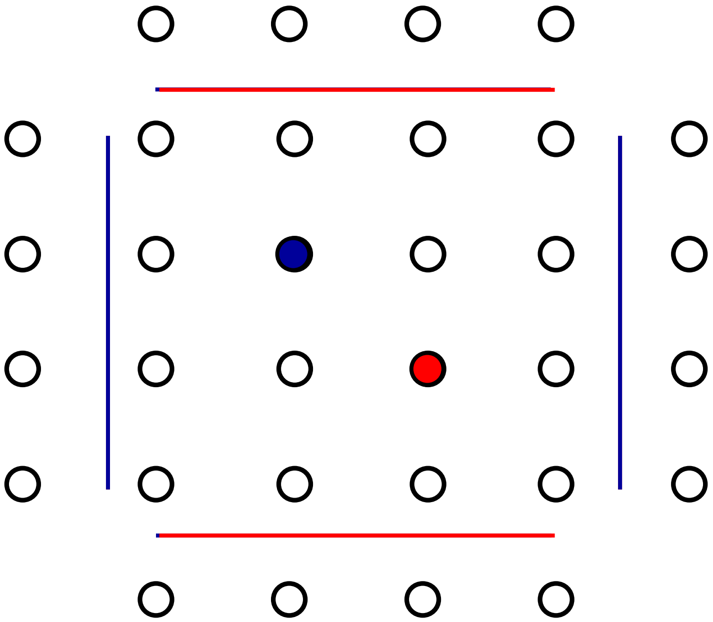
Sample game



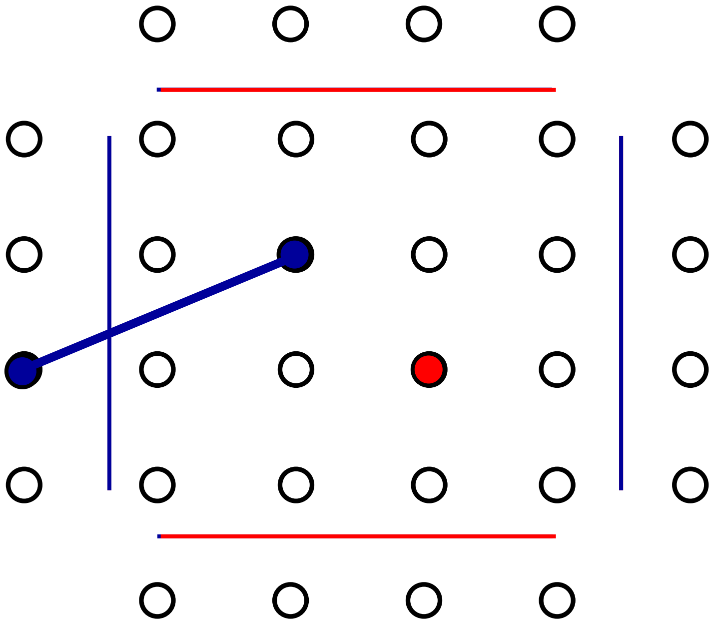
Sample game



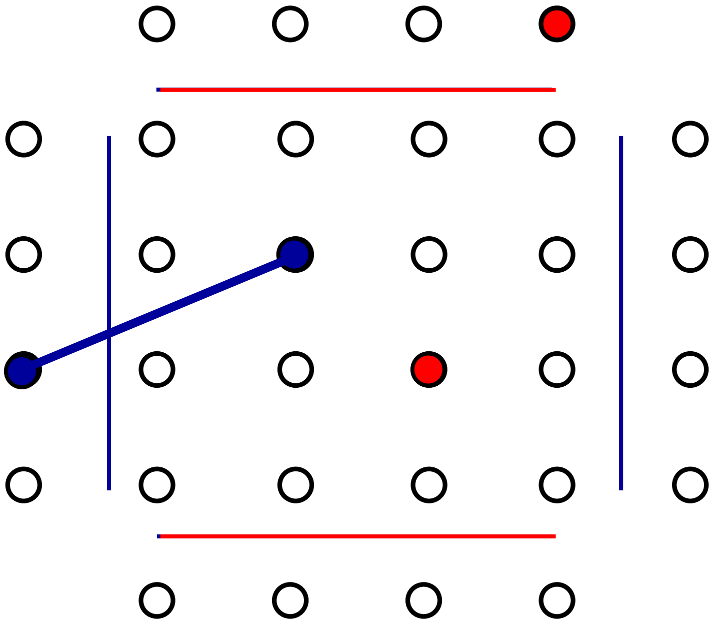
Sample game



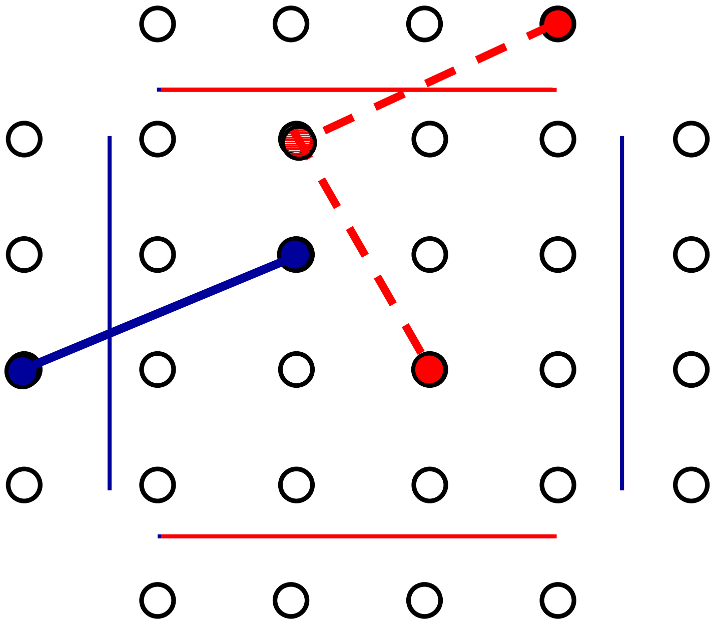
Sample game



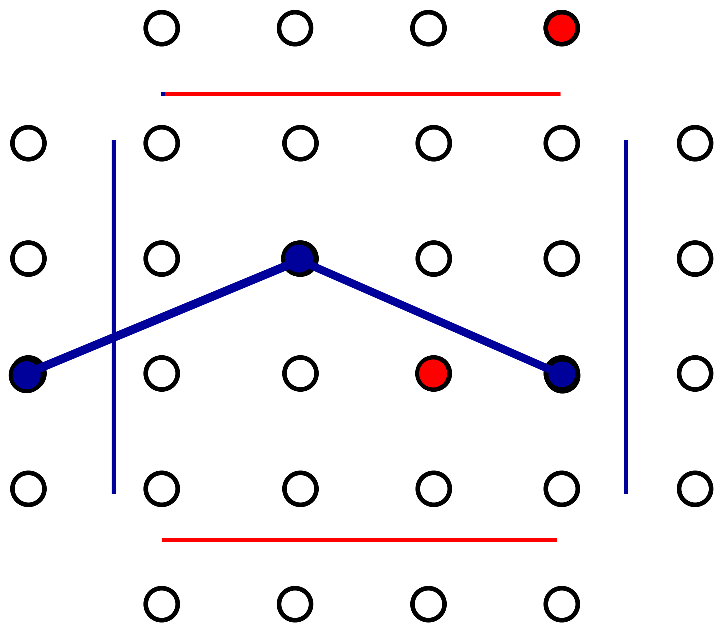
Sample game



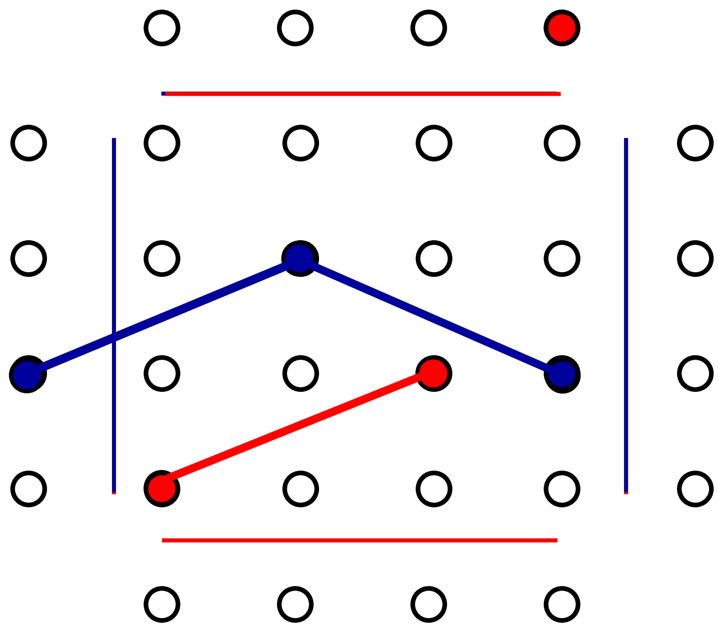
Sample game



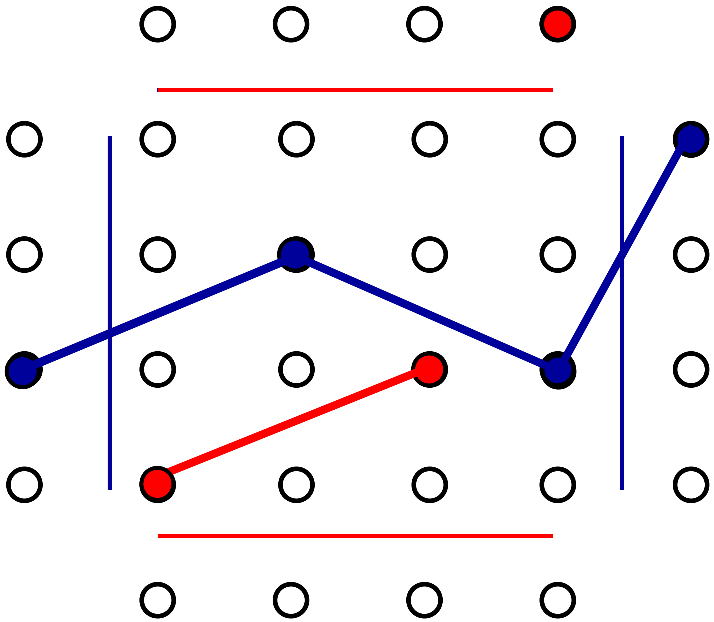
Sample game



Sample game



Sample game



Blue

Wins!

Our task

a) Write a program that improves its playing strategy by learning as it plays.

b) Compare its performance with that of a player based on a fixed heuristic

What we did

- Adapted preexisting Python classes (A. Ross) to represent the game of TwixT
- Built web server to play the game (code for rendering game, managing game state)
- The computer players

The computer players

- Non-learning player (fixed heuristic)
- Learning players:
  - Perceptron-based
  - 2-layer Neural Network-based

  Based on a value function defined as a weighted summation of a number of *features* (more later)

The value function

$$h(s) = \tanh\left(\sum_{i=1}^{N} w_i f_i\right) + w_g g$$

s = state of the board

f_i = value of *feature i*

w_i = weight of feature *i*

g = *"General criterion"*

Samuel, A. L., 1959. Some studies in machine learning using the game of checkers. IBM Journal of Research and Development 3 (3), 210-229

Features

•Defined by us

•Some have obvious "meaning" in terms of proceeding towards goal (general goal). *High value = advantage in game*

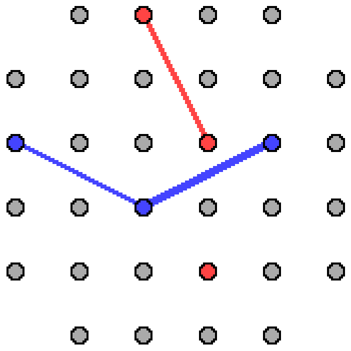
•The meaning of others is unclear (helpful when others close to zero?)

Features – f_1

Let:

•*P* be a player

•*s* = side of the board.



***f_1* (*P*)** = *(total length of bridges owned by P)/ s^2*

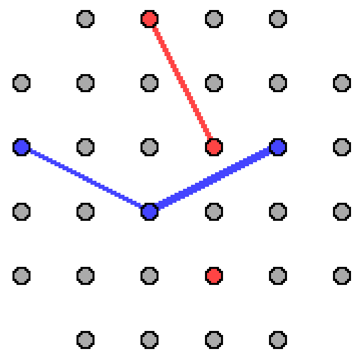
Features – f_2, f_3, f_4

Let:

• *P'* be *P*'s opponent

$$f_2(P) = f_1(P) - f_1(P')$$

$f_3(P)$ = *distance covered by projections of P's bridges along the direction of the opponent's edges*



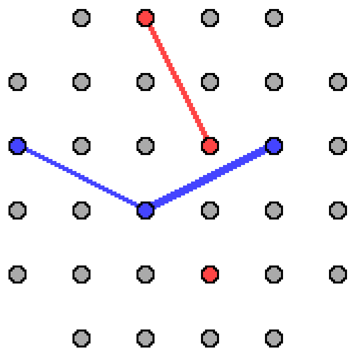
$f_4(P) = 1 - f_3(P')$

Features – f_5, f_6, f_7

Let:

•*Ncm* = number of potentially connectable nodes

•*Ncn* = number of connected nodes



$$f_5(P) = Ncm(P)/Ncn(P)$$

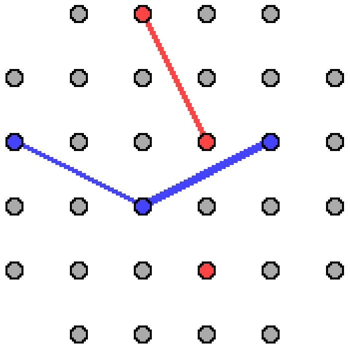
$f_6(P) = +\infty$ if winning move for P, *0* o/w

$f_7(P) = -\infty$ if winning move for P', *0* o/w

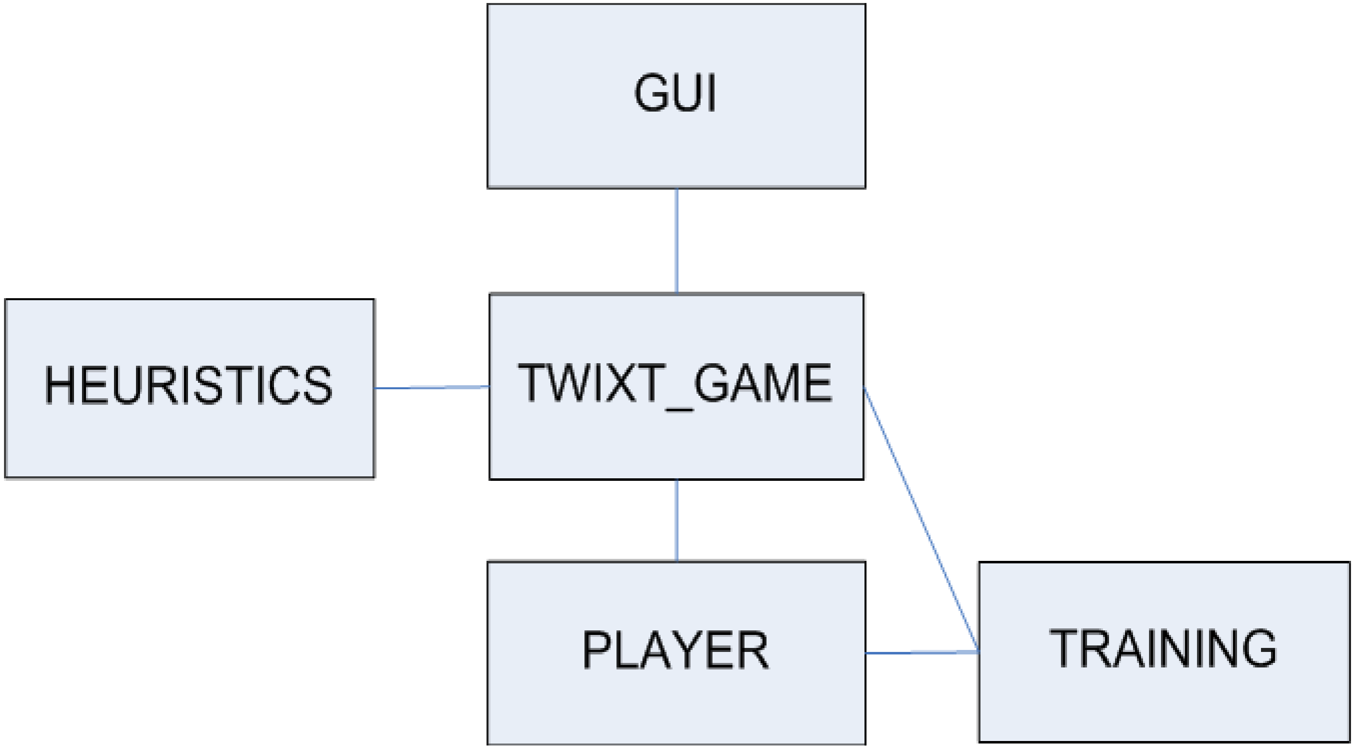
Features - *g*

$$g\,(P) = f_3\,(P) - f_3\,(P')$$

It is always assumed that it is to the player's advantage to be "ahead" of opponent

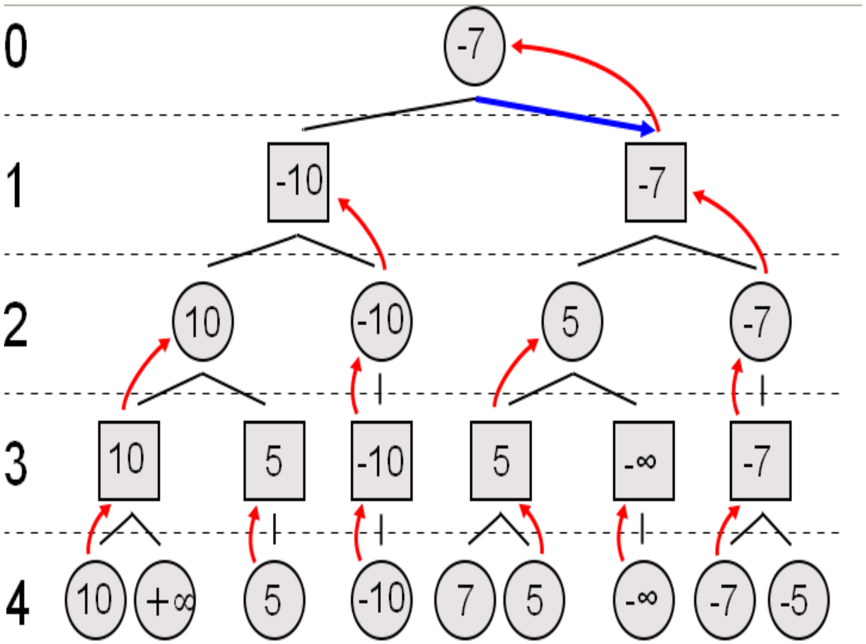


System Architecture



Learning Techniques (1) MiniMax Search

- *2 Level Search*



Source: http://upload.wikimedia.org/wikipedia/commons/6/6f/Minimax.svg

Learning Techniques (2)
Neural Network

- Game score

$$h(s) = \tanh\left(\sum_{i=1}^{N} w_i f_i\right) + w_g g$$

- Weights update

$$w_i(t+1) = w_i(t) + n * (d - h) * f_i$$

- Weight for dominant criterion is fixed

Training Process

- Random fixed weights vs. dominant criterion
- Learning perceptron vs. dominant criterion
- Random fixed weights vs. Learning Percep.

Results

Random fixed weights vs. dominant criterion

50 generations

Random Fixed weights	Dominant criterion	draw
50	0	0

Results

Learning perceptron vs. dominant criterion

50 generations

Learning Perceptron	Dominant criterion	draw
43	0	7

Results

Random fixed weights vs. Learning Percep.

50 generations

Random fixed weights	Learning Perceptron	draw
50	0	0

Results

- Our testing shows that there is a strong bias for the first player.
- Likely a result of the very small board size we trained on (6x6). First player is guaranteed a win by playing a certain strategy.
- Even by placing the learning player in the second position, could not achieve a win for player 2.

Results

- Depth of search was limited to 2 states. This was purely limited by available horsepower.
  - This limits the sophistication play rather severely and also favors player 1.
- Another problem is that we likely need more features.

Conclusion

- We have a lot of room for improvement.
- Improve code efficiency to increase search depth and board size.
- Or possibly purchase new supercomputer.
- Add (and test) new features.