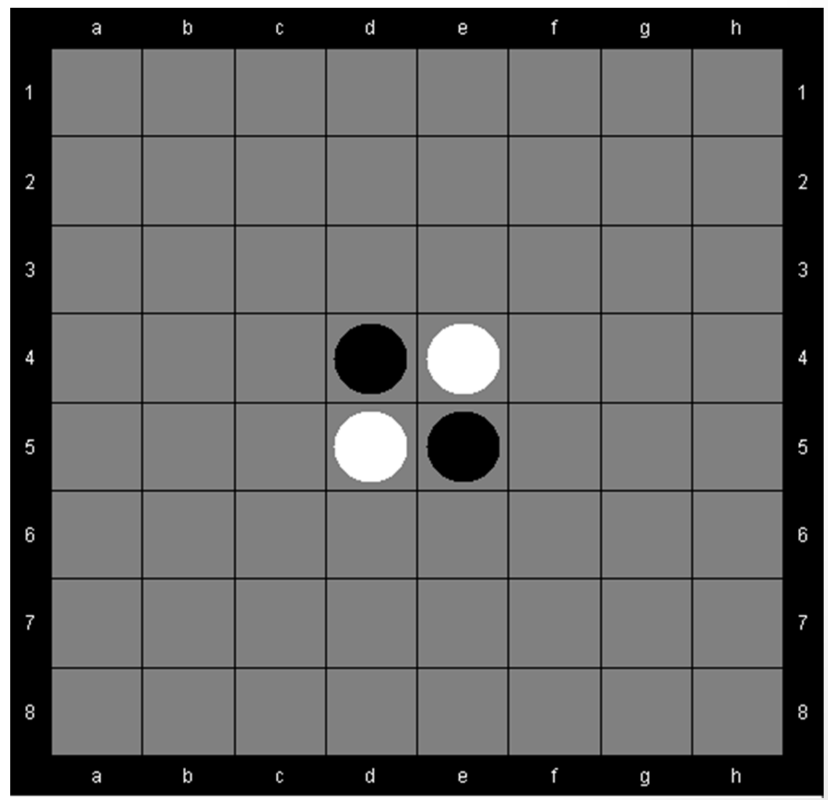
Othello/Reversi using Game Theory techniques

Parth Parekh
Urjit Singh Bhatia
Kushal Sukthankar

Othello

- Rules
  - Two Players (Black and White)
  - 8x8 board
  - Black plays first
  - Every move should ‘Flip’ over at least one opponent disk
  - Goal: Maximize ones disks
- Board (starting position)



Technical Description

- Two-player deterministic zero-sum game with perfect information.
- Game tree size is approximately 10^{56}.
- State space size (legal positions) is approximately 10^{28}.
- Branching factor is approximately 10.
- Max move length is 60.

Our AI players

- Random
- Absolute minimax
- Positional minimax
- Mobility minimax
- Boosting player
- Q-learning player

Heuristics-based players

- Minimax
  - The minimax algorithm with alpha-beta pruning was used to determine which move was optimal given the evaluation function.
- Three Heuristics based players created
  - Positional
  - Mobility
  - Absolute

Human Strategies

- Positional
  - Maximize its own valuable positions (such as corners and edges) while minimizing its opponent's valuable positions.
  - Evaluation Function :

$$w_{a1}v_{a1} + w_{a2}v_{a2} + \ldots + w_{a8}v_{a8} + \ldots + w_{h8}v_{h8}$$

w_i is +1, -1 or 0 if the square is occupied by player, opponent or empty

- Weights

	a	b	c	d	e	f	g	h
1	100	-20	10	5	5	10	-20	100
2	-20	-50	-2	-2	-2	-2	-50	-20
3	10	-2	-1	-1	-1	-1	-2	10
4	5	-2	-1	-1	-1	-1	-2	5
5	5	-2	-1	-1	-1	-1	-2	5
6	10	-2	-1	-1	-1	-1	-2	10
7	-20	-50	-2	-2	-2	-2	-50	-20
8	100	-20	10	5	5	10	-20	100

Human Strategies - 2

- Mobility
  - Number of Legal Moves a player can make in a particular position.
  - Maximize own mobility and minimize opponents mobility
  - Corner square are important in mobility
  - Evaluation Function :

$$10 * \left(c_{player} - c_{opponent}\right) + \left(\frac{m_{player} - m_{opponent}}{m_{player} + m_{opponent}}\right)$$

Where, c is corner squares,

m is the mobility

Human Strategies -3

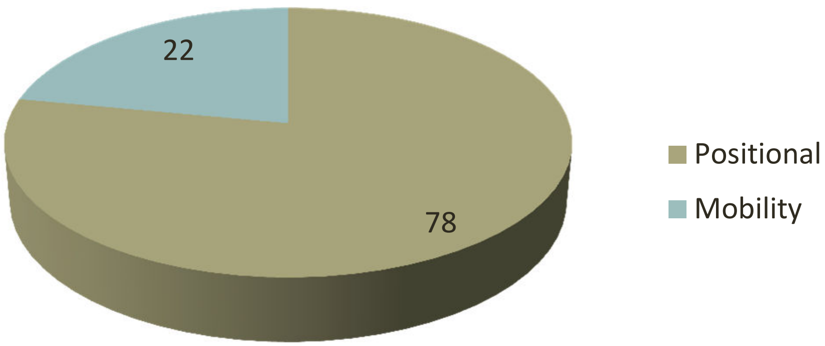
- Absolute
  - Maximize ones own disks
  - Evaluation Function:

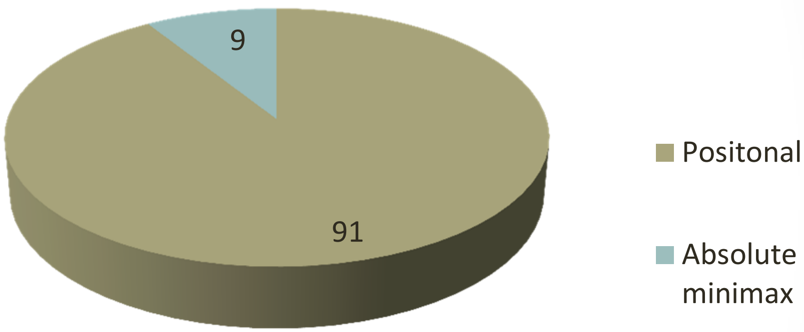
$$n_{player} - n_{opponent}$$

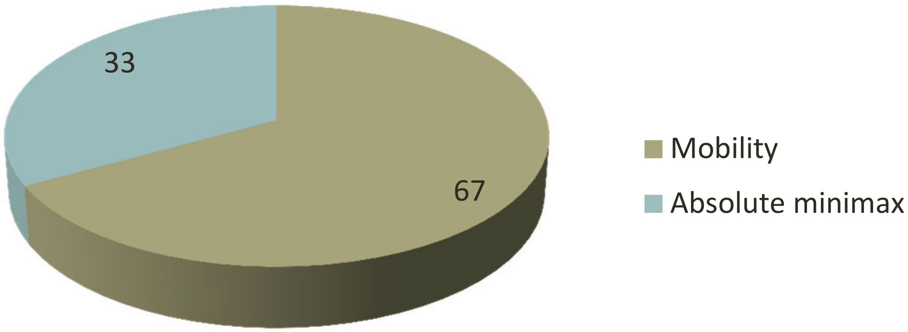
Game Phases

- An Othello game can be split into three phases where strategies can differ:
  - Beginning
    - First 20 to 25 moves
  - Middle
  - End
    - Last 10 to 16 moves
- Usually heuristic players use Positional/Mobility for beginning and middle phases. Then switch to Absolute for the end phase

Performance







Q-Learning

- The Q learning player is a reinforcement learning based player.
- Q learning tries to learn the function Q(s,a) to find the optimal policy.
- The Q function is defined as:
    - “The reward received upon executing action a from state s, plus the discounted value of rewards obtained by following an optimal policy thereafter”

Q-Learning

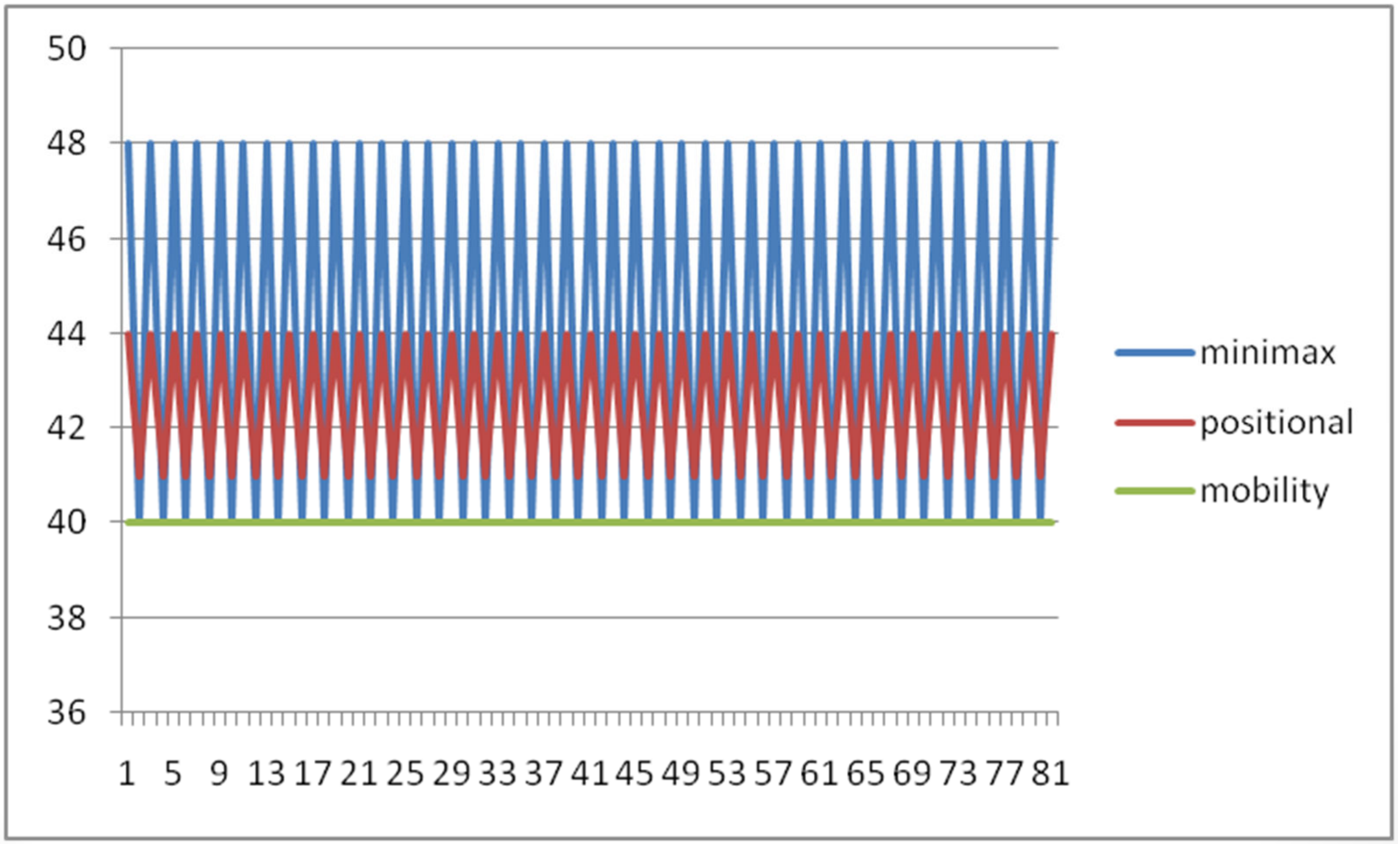
- In this system, rewards are defined as follows:
  - Wins gets 1 point, Draw gets 0 and Loss gets -1.
- We then save the learned Q information using a Neural Network since the state space is too large and we need a compact way of storing this data.

Q-Learning

- For all states s and all actions a: initialize Q(s, a) to an arbitrary value.
- Repeat (for each trial)
  - Initialize the current state s
  - Repeat (for each step of trial)
    - Observe the current state s
    - Select an action a using a policy
    - Execute action a
    - Receive an immediate reward r
    - Observe the resulting new state s’
    - Update Q(s, a)

Q-Learning

- Performance of Q-Learning against other simple AI – win margins graph



Q-Learning

- There is a problem – how to save the Q values learnt during a set of trials across sessions.
- Using a simple look-up table will be very time consuming and as more and more state-space is explored, there is a data explosion and it becomes impossible to store it.
- So we can simply get the Q value for the action which gets the maximum value to play.

Boosting

- General method of converting rough rules of thumb into highly accurate prediction rule
- Technically:
  - Assume given “weak” learning algorithm that can consistently find classifiers (“rules of thumb”) at least slightly better than random, say, accuracy 55% (in two-class setting)
  - Given sufficient data, a boosting algorithm can provably construct single classifier with very high accuracy, say, 99%

Weak Learners

- Frontier
  - The discs which have many empty neighboring squares are frontier. They increase opponents mobility.
- Parity
  - Playing last into each region gives best results.
- Edge (Stable)
  - A disc placed on a corner square cannot be flipped. Discs are stable if surrounding disks are stable.
- Absolute
  - Sometimes maximizing ones own disks gives best results

Weak Learners -2

- Evaporation
  - The fewer the disk the payer has, the greater his mobility
- Mobility
  - Reducing number of available moves for opponent increases chances he will make a bad move
- Positional
- Positional-2
  - Gain control of good squares and avoid bad.

AdaBoost Algorithm

D_t : Distribution for m Weak Learners

Initialize $D_1(i) = 1/m$
For every sample, given $D_t \& h$,
For every Weak Learner i,

$$D_{t+1}(i) = \frac{D_t(i)}{Z_i} * e^{-\alpha_t} \qquad \text{If move is correct}$$

$$= \frac{D_t(i)}{Z_i} * e^{\alpha_t} \qquad \text{If move is wrong}$$

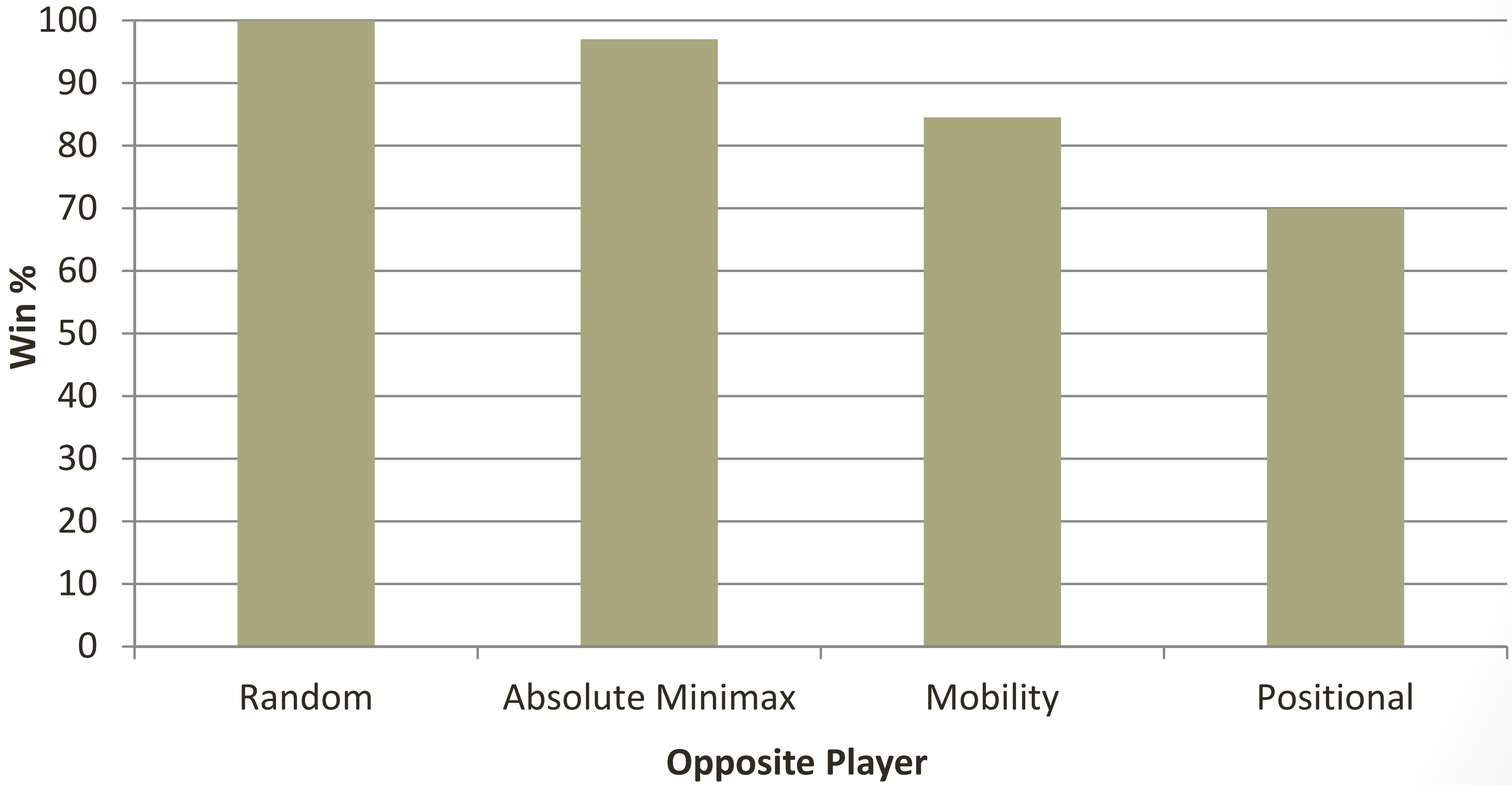
Where,
Z_i is the Normalization Constant.
and α_t is small and positive

Training

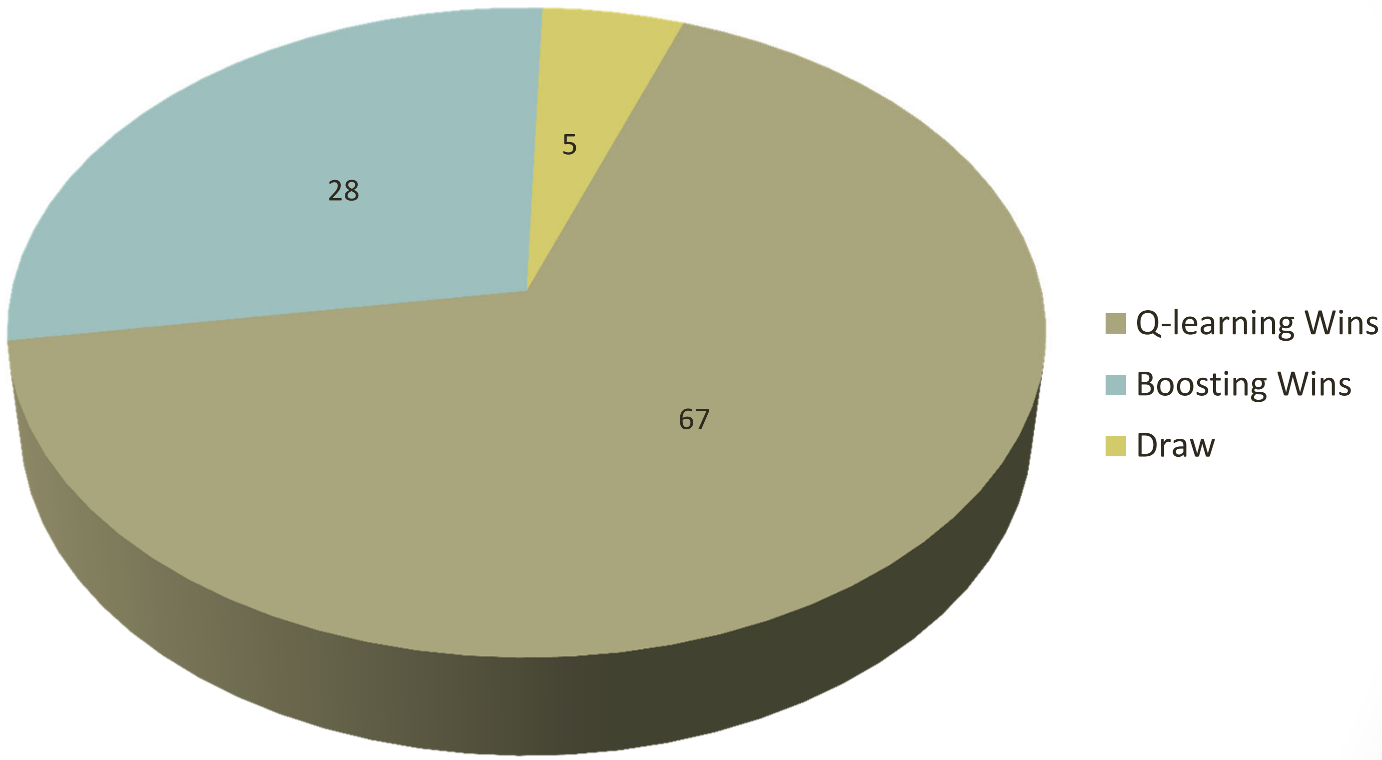
- The algorithm played against itself over multiple games
- Each phase of the game (Beginning, Middle and End) had a different distribution
- The winning distributions were saved and used for the next game

Distribution

Performance of Boosting



Boosting v/s Q-Learning



Papers

- *Reinforcement Learning and its Application to Othello*
  -- Nees Jan van Eck, Michiel van Wezel

- *Using AdaBoost to Implement Chinese Chess Evaluation Functions*
  -- Chuanqi Li

- *Using a Support Vector Machine to learn to play Othello*
  -- Daniel Karavolos