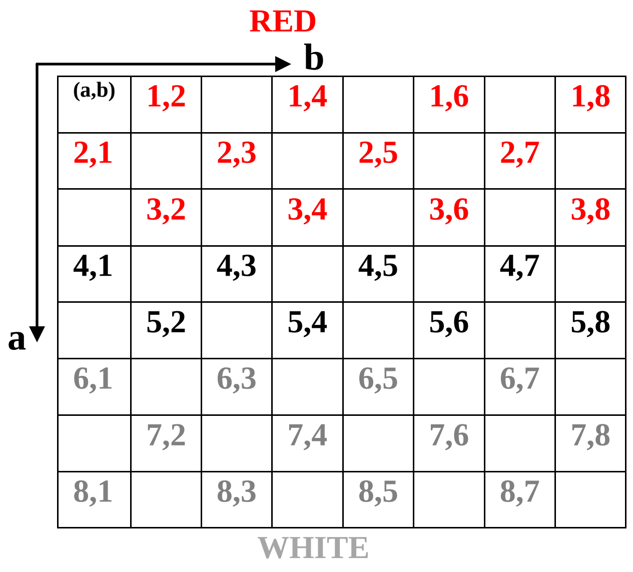
Homework 2-Checkers, Strategies, Evaluation functions

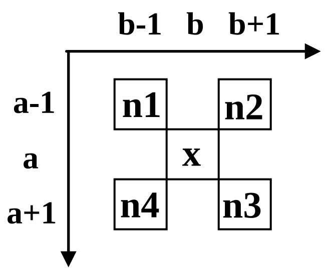
Professor Davi Geiger
Due February 13th.

/* Checkers Board and Coordinates */



(a,b)	1,2		1,4		1,6		1,8
2,1		2,3		2,5		2,7	
	3,2		3,4		3,6		3,8
4,1		4,3		4,5		4,7	
	5,2		5,4		5,6		5,8
6,1		6,3		6,5		6,7	
	7,2		7,4		7,6		7,8
8,1		8,3		8,5		8,7	

/* Neighbors of x */



/* States
S(a,b**)=**0,1,2,3,4**;**
/* respectively empty, red, white, kingred, kingwhite***/**

1. *Develop min-max search strategy using depth first search.* You can try to reach depth 6 or 10 or 14, depending on your computer RAM. Here you assume you do have an evaluation function (to be developed later, below). So every move (or action) will have an associated value “v”, defined by the evaluation function.

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

2. *Add the following Cut-off depth criteria of the search:* If the final state where you apply evaluation allow for follow up jumps, then “don’t stop the evaluation there, do more search steps in this path, until no jumps are left for either player.

3. *Write two evaluation function that makes good selection of moves among the possible moves.* Describe your two evaluation functions precisely. Simulate a game where red uses one strategy and white uses the other one. The first move should be a random choice (there are 7 possible moves).
   Below are suggestions of some components for constructing an evaluation function.

Suggested functions, but they must be combined with some weights and you don’t have to use all of them and you can choose others.
/* 1: sum total number of your pieces (subtract from adversary) */

/* 2: sum total number of your king pieces (subtract from opponent king pieces) */

/* 3: Offense: count how advanced are your pieces (how close to become a king) (subtract from opponents advanced position). How to count this? For example, count how much is the sum of all your “a” component. Note that depending if you are red or white, high sum or low sum may be preferred.*/

/* 4: Defense: how well defended are the pieces? sum total number of red previous neighbors (n3,n4) to a red piece and subtract from total number of white previous neighbors (n1, n2) to a white piece. */

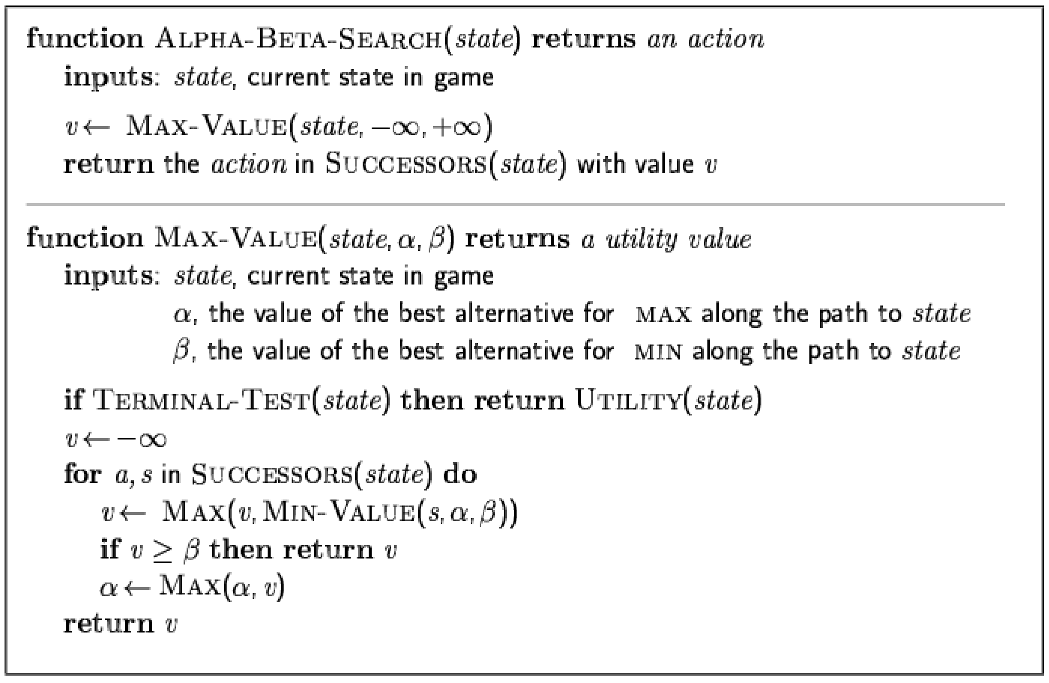
/* 5: Defense against kings: sum total number of red post neighbors (n1,n2) to a red piece and subtract from total number of white post neighbors (n1, n2) to a white piece. Combine with 4. */

/* 6: Defense on sides: see if pieces are on the sides of the board, so that they are defended. This criteria should be combined with 4,5.*/

/*7: Defense: Kings on corners are better defended. This is should be combined with 4, 5 and 6 */

/* 8: Dynamic position. Count number of possible moves by pieces. Count number of possible moves by kings. Compare with opponents. */

4. *Develop α–β pruning strategy to cut computations*. Report on your improved depth and the RAM of your computer (how much depth you get with the same memory).



```
function ALPHA-BETA-SEARCH(state) returns an action
  inputs: state, current state in game
  $v \leftarrow$ MAX-VALUE(state, $-\infty$, $+\infty$)
  return the action in SUCCESSORS(state) with value $v$

function MAX-VALUE(state, $\alpha$, $\beta$) returns a utility value
  inputs: state, current state in game
          $\alpha$, the value of the best alternative for MAX along the path to state
          $\beta$, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow -\infty$
  for $a, s$ in SUCCESSORS(state) do
    $v \leftarrow$ MAX($v$, MIN-VALUE($s, \alpha, \beta$))
    if $v \geq \beta$ then return $v$
    $\alpha \leftarrow$ MAX($\alpha$, $v$)
  return $v$
```

```
function MIN-VALUE($state, \alpha, \beta$) returns a utility value
    inputs: $state$, current state in game
            $\alpha$, the value of the best alternative for MAX along the path to $state$
            $\beta$, the value of the best alternative for MIN along the path to $state$
    if TERMINAL-TEST($state$) then return UTILITY($state$)
    $v \leftarrow +\infty$
    for $a, s$ in SUCCESSORS($state$) do
        $v \leftarrow$ MIN($v$, MAX-VALUE($s, \alpha, \beta$))
        if $v \leq \alpha$ then return $v$
        $\beta \leftarrow$ MIN($\beta, v$)
    return $v$
```