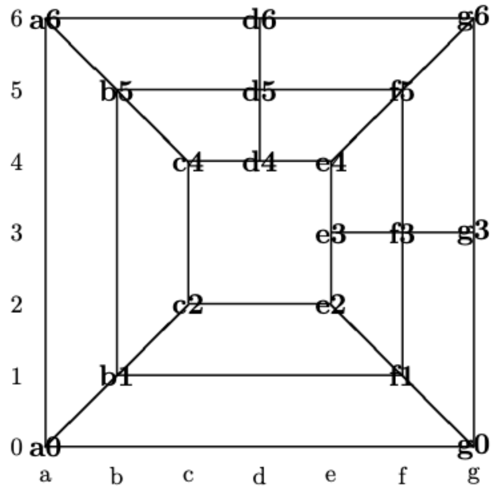
MORRIS GAME

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a0	g0	b1	f1	c2	e2	e3	f3	g3	c4	d4	e4	b5	d5	f5	a6	d6	g6



This variant of Morris Game has 18 positions which are denoted by the array shown above.
There are 8 moves for white and black pieces in the game. The pieces with less than 3 on the board loses the game.

The positions are estimated using **MiniMax Algorithm** and **α-β Pruning Algorithm**

- The MiniMax program written for opening phase of the game is **MiniMaxOpening.java**

  This program is recursive defined MiniMax algorithm in the opening phase of the game with a depth value of the tree where the leaf values are calculated using the static estimation function in the opening phase.

  The **input** tested:
  xWxxxWxxBxxxxxBxxx
  The **output**:
  Board Position: xWxWxWxxxxxxxxBxxx
  Positions evaluated by static estimation: 197
  MINIMAX estimate: 1
  depth: 2

  The **input** tested:
  xxxxxxWxxxxxxBxxxx
  The **output**:
  Board Position: WxxxxxWxxxxxxBxxxx
  Positions evaluated by static estimation: 240
  MINIMAX estimate: 0
  depth: 2

- The MiniMax program written for mid / ending phase of the game is **MiniMaxGame.java**

  This program is recursive defined MiniMax algorithm in the mid/endgame phase of the game with a depth value of the tree where the leaf values are calculated using the static estimation function in the midgame / endgame phase.

  The **input** tested:
  WWBBBBxWxxxxWBWxWx
  The **output**:
  Board Position: WWBBBBWxxxxxWBWxWx
  Positions evaluated by static estimation: 38
  MINIMAX estimate: 994
  depth: 2

  The **input** tested:
  xBBxxxWxxWxxBxBxxW
  The **output**:
  Board Position: xBBxxxxxxWxxBWBxxW
  Positions evaluated by static estimation: 442
  MINIMAX estimate: -1011
  depth: 2

- The MiniMax program written for opening phase of the game for black is **MiniMaxOpeningBlack.java**

  This program is recursive defined MiniMax algorithm for the black moves in the opening phase of the game with a depth value of the tree where the leaf values are calculated using the static estimation function in the opening phase.

  The **input** tested:
  xWxxxWxxBxxxxxBxxx
  The **output**:
  Board Position: xWxBxWxxBxxxxxBxxx
  Positions evaluated by static estimation: 208
  MINIMAX estimate: 0
  depth: 2

  The **input** tested:
  The **output**:
  Board Position: BxxxxxWxxxxxxBxxxx
  Positions evaluated by static estimation: 240
  MINIMAX estimate: 0
  depth: 2

- The MiniMax program written for mid / ending phase of the game is **MiniMaxGameBlack.java**

  This program is recursive defined MiniMax algorithm for the black moves in the mid/endgame phase of the game with a depth value of the tree where the leaf values are calculated using the static estimation function in the midgame / endgame phase.

  The **input** tested:
  WWBBBBxWxxxxWBWxWx
  The **output**:
  Board Position: WWBBxBxWxBxxWBWxWx
  Positions evaluated by static estimation: 35

MINIMAX estimate: -1021
depth: 2

The **input** tested:
xBBxxxWxxWxxBxBxxW
The **output**:
Board Position: xBxxxxxxxWxxBBBxxW
Positions evaluated by static estimation: 377
MINIMAX estimate: 10000
depth: 2

- The α-β Pruning program written for opening phase of the game is **ABOpening.java**

  This program is recursive defined α-β Pruning algorithm in the opening phase of the game with a depth value of the tree where the leaf values are calculated using the static estimation function in the opening phase.

  The **input** tested:
  xWxxxWxxBxxxxxBxxx
  The **output**:
  Positions evaluated by static estimation: 40
  αβ estimate: 1
  depth: 2

  The same input when done with MiniMax had to evaluate 197 positions which is 40 for α-β. This clearly shows that not all the leaf nodes are evaluated.

  The **input** tested:
  xxxxxxWxxxxxxBxxxx
  The **output**:
  Board Position: WxxxxxWxxxxxxBxxxx
  Positions evaluated by static estimation: 30
  αβ estimate: 0
  depth: 2

- The α-β Pruning program written for mid / ending phase of the game is **ABGame.java**

  This program is recursive defined α-β Pruning algorithm in the mid/endgame phase of the game with a depth value of the tree where the leaf values are calculated using the static estimation function in the midgame / endgame phase.

  The **input** tested:
  WWBBBBxWxxxxWBWxWx
  The **output**:
  Board Position: WWBBBBWxxxxxWBWxWx
  Positions evaluated by static estimation: 15
  αβ estimate: 994
  depth: 2

  The same input when done with MiniMax had to evaluate 38 positions which is 15 for α-β. This clearly shows that not all the leaf nodes are evaluated.

  The **input** tested:
  xBBxxxWxxWxxBxBxxW
  The **output**:
  Board Position: xBBxxxxxxWxxBWBxxW
  Positions evaluated by static estimation: 102

αβ estimate: -1011
depth: 2

The same input when done with MiniMax had to evaluate 442 positions which is 102 for α-β. This clearly shows that not all the leaf nodes are evaluated.

- The MiniMax program written for opening phase of the game with improved static estimation function is **MiniMaxOpeningImproved.java**

  This program is recursive defined MiniMax algorithm in the opening phase of the game with a depth value of the tree where the leaf values are calculated using the improved static estimation function in the opening phase.

  The improved static estimation function included the possible Mill counts for white pieces on the board which are potential win positions in the future this increases the value of the leaf nodes in MiniMax calculation and possibly a better winning move for white.

  The **input** tested:
  xWxxxWxxBxxxxxBxxx
  The **output**:
  Board Position: xWxWxWxxxxxxxxBxxx
  Positions evaluated by static estimation: 197
  MINIMAX estimate: 4
  depth: 2

  The MiniMax estimate for the same input was 1 compared to 4 when used with improved static estimation function

  The **input** tested:
  xxxxxxWxxxxxxBxxxx
  The **output**:
  Board Position: WxxxxxWxxxxxxBxxxx
  Positions evaluated by static estimation: 240
  MINIMAX estimate: 1
  depth: 2

  The MiniMax estimate for the same input was 0 compared to 1 when used with improved static estimation function

- The MiniMax program written for mid/ending phase of the game with improved static estimation function is **MiniMaxGameImproved.java**

  This program is recursive defined MiniMax algorithm in the mid/endgame phase of the game with a depth value of the tree where the leaf values are calculated using the improved static estimation function in the midgame / endgame phase.

  The improved static estimation function included the possible Mill counts for white pieces on the board which are potential win positions in the future this increases the value of the leaf nodes in MiniMax calculation and possibly a better winning move for white.

  The **input** tested:
  WWBBBBxWxxxxWBWxWx
  The **output**:
  Board Position: WWBBBBxWxWxxxBWxWx
  Positions evaluated by static estimation: 38
  MINIMAX estimate: 2994
  depth: 2

The MiniMax estimate for the same input was 994 compared to 2994 when used with improved static estimation function. The board position generated by MinMax was **WWBBBBWxxxxxWBWxWx** the position generated with improved MiniMax is **WWBBBBxWxWxxxBWxWx** which is clearly a better move when mill positions are considered.

The **input** tested:
xBBxxxWxxWxxBxBxxW
The **output**:
Board Position: xBBxxxxxxWxxBWBxxW
Positions evaluated by static estimation: 442
MINIMAX estimate: -11
depth: 2

The MiniMax estimate for the same input was -1011 compared to -11 when used with improved static estimation function.