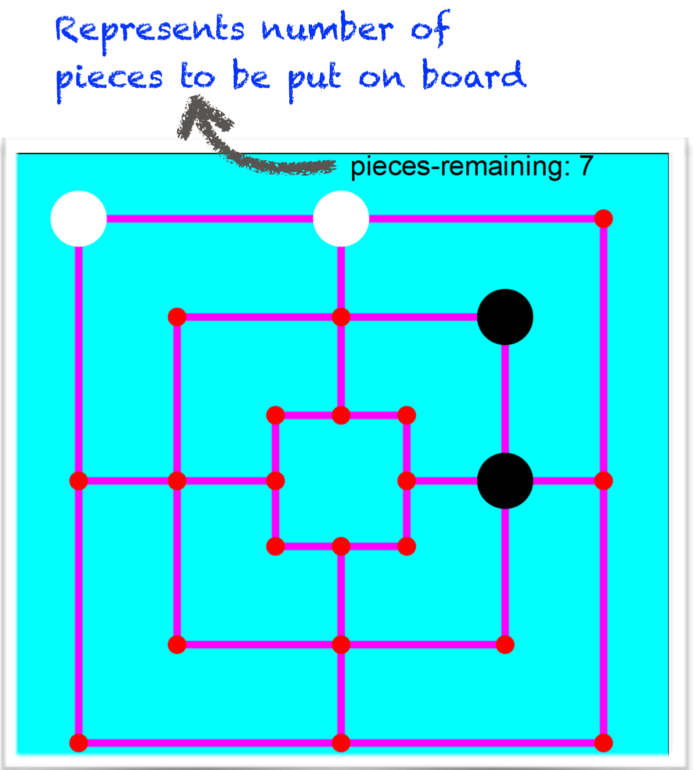
9 Men's Morris

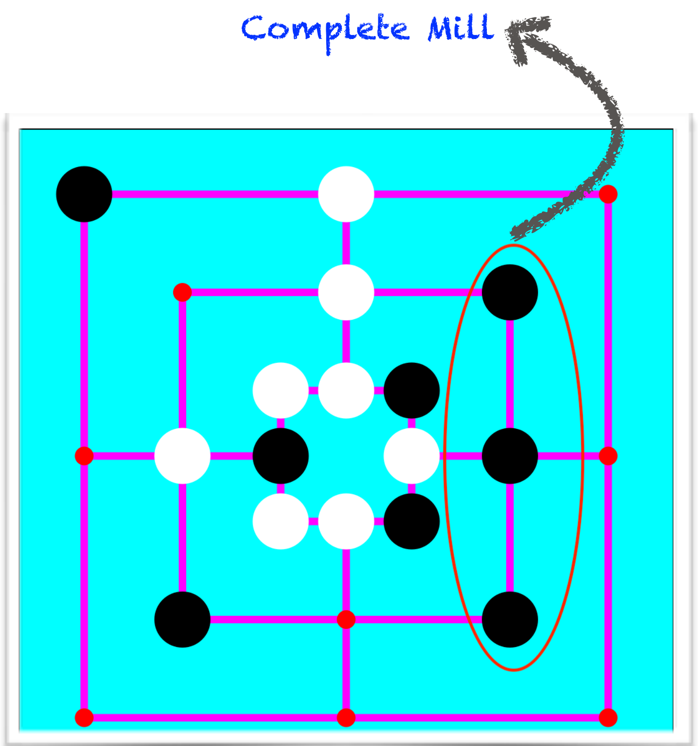
Shreya Pathak: 180050100
Pawan Goyal:180050072
Mohammad Ali Rehan: 180050061

Description Of The Problem

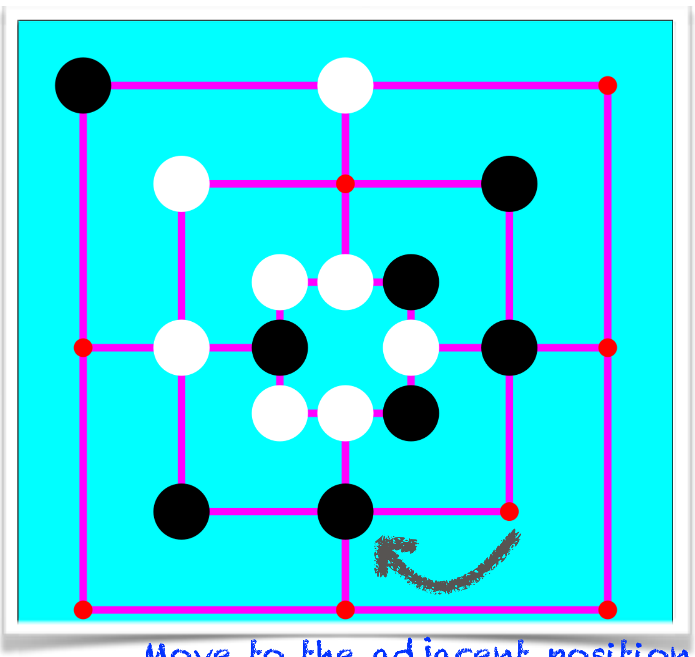
The aim is to design an interactive GUI software that allows the user to play 9 men's Morris. The software offers two modes – multiplayer or single player. For the single player, there needs to be a game engine, i.e. an AI which plays against the user.
The game is played in three phases. The starting game phase involves the players taking turns placing their pieces on the board. In the mid-game phase, the players can move their pieces to the adjacent empty places (if any) and is reached after all 9 pieces of both players have been placed. The final phase is reached when the player has only three pieces left. In this, the player can move his/her piece to any empty position on the board. If any 3 pieces of a player fall in a straight line (in any phase) they are said to be in a closed mill. In such a situation the player can remove a piece of his/her opponent which is not itself in a closed mill. The game ends when a player has less than 3 pieces on board or has no further moves (i.e. all pieces are blocked).



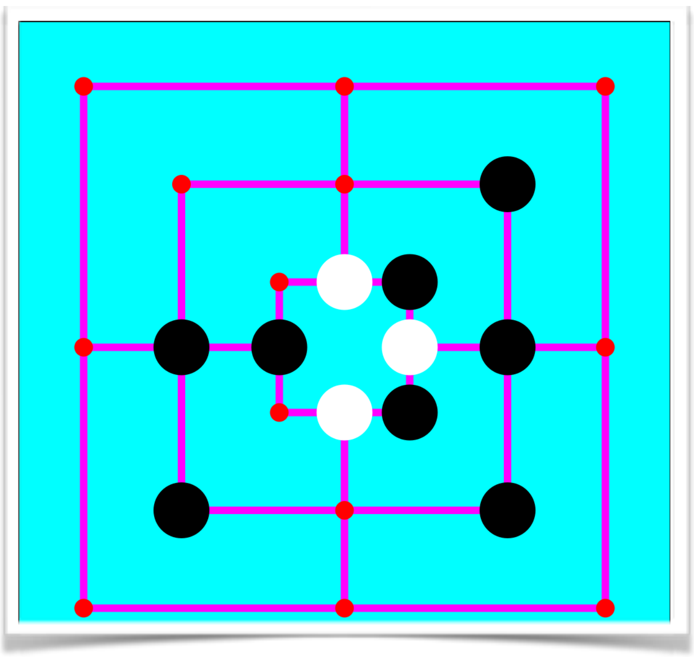
Start Phase



Mid-Game Phase(a)



Mid-Game Phase(b)



End Phase

Design Of The Program

The files in our program are:

- **final.rkt**: Its the final file which integrates all the rest and once run, initiates the game. It ,after compiling, shows the initial home screen and calls upon corresponding mouse handlers depending on the mode of game play chosen by the user.
- **main_gui.rkt**:- This file has functions for all running the interface seen by the user in the single-player mode. GUI is implemented using the 2htdp/universe, and big-bang.
- **Board_gui.rkt**: Here we have used 2htdp/image to design the board's graphics, the pieces etc. Classes(an example of OOP and thus abstraction) have been used to implement white pieces, black pieces and the red circles(empty slots)
- **Board.rkt**: all the primitive functions to change the state of the board are implemented here. We have used a 2d vector to represent the board. Positions are represented in terms of s, r, c coordinates on the board. S being the square. R being the row in it. C being the column In it. The squares, rows and columns are numbered 0 to 2, the squares going from the outermost one to the innermost, the rows from top to bottom and the columns from left to right.
- **2player.rkt** : This implements the 2 player game, and in turn calls on corresponding mouse handler and drawing functions to allow the game to play. It is very similar to main_gui.rkt.

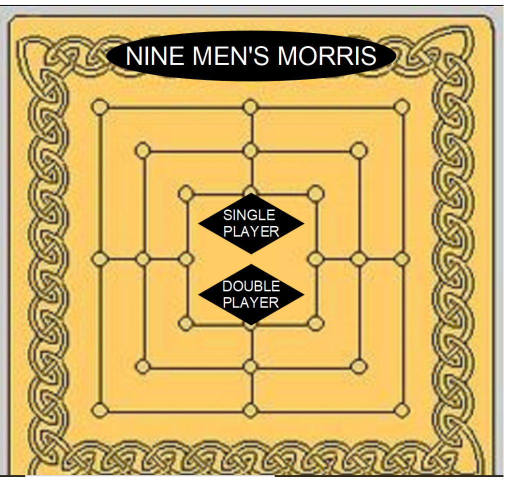
- **eval.rkt**: This file includes the evaluation function required to give a value to each possible board which is then used by our AI to choose the best possible move using minimax algorithm.
- **minmax.rkt**: This file implements the alpha-beta pruning algorithm to find the leaf with max value. It is a search algorithm that assumes two players: one which tries to maximise the value and other which want to minimise it. Alpha beta pruning cuts off branches in the game tree which need not be searched because there already exists a better move available.
- **lplay.rkt**: As the name suggests, it enables the user to play by providing the move by computer. This included the AI of the game which, after generating a tree and evaluating its leaf with eval function, gives the most optimal move. This file combines functions written in eval, board and minmax and forms the heart of our game.

Abstractions and Objects used

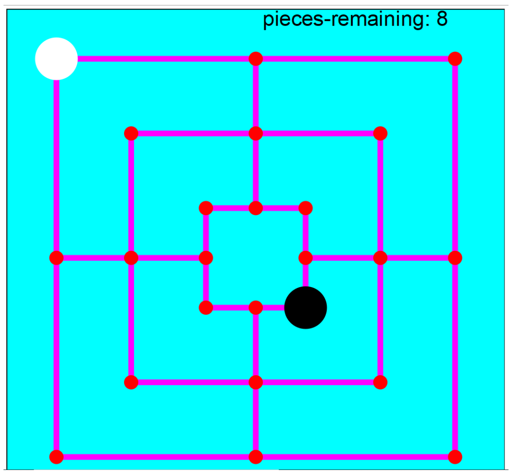
- A struct gnode is used to represent the tree of boards used to predict the next optimal move.
- Board in lplay.rkt is represented using a class which has a member field remove which is true when next move on the given board is to remove a piece and false if the next move is anything other than remove. This greatly simplifies the work as before there was no hand waving way to say whether a piece needs to be removed or not by just looking at the board.
- Classes(an example of OOP and thus abstraction) have been used to implement white pieces, black pieces and the red circles(empty slots). Button, and game pieces inherit from an abstract class of piece.
- Macros (especially lc) have been extensively used to reduce the size of the code and simplify the problem.
- The board has been implemented using 2D vectors, but we cannot access all positions in the vector. The functions new-piece, remove-piece, etc. only allow us to access a few indices of the 2d vector (which have meaningful s r c coordinates), hence 2D vector has been abstracted to give us the data structure board, by keeping only relevant positions accessible.

SAMPLE INPUT AND OUTPUT

The user will interact via mouse clicks on the screen. The input will be a click on the piece to be moved/ position to place the piece in and output will be displayed on the screen in the form of a move made of the user and consequently, of the computer.



The Home Screen



Board With Moves

When The Game Ends

Limitations and Bugs

- The AI may not make the best move possible in all situations due to depth limitations
- The AI may take unexpectedly long time to find the next move especially when both players have 3 pieces on board.
- The evaluation function could use more evaluators to give a more accurate value to the boards so that better moves are predicted.
- We tried to add a networking feature to the game using which players could play the game across lan using the features provided by the universe library of racket but could not complete due to paucity of time.