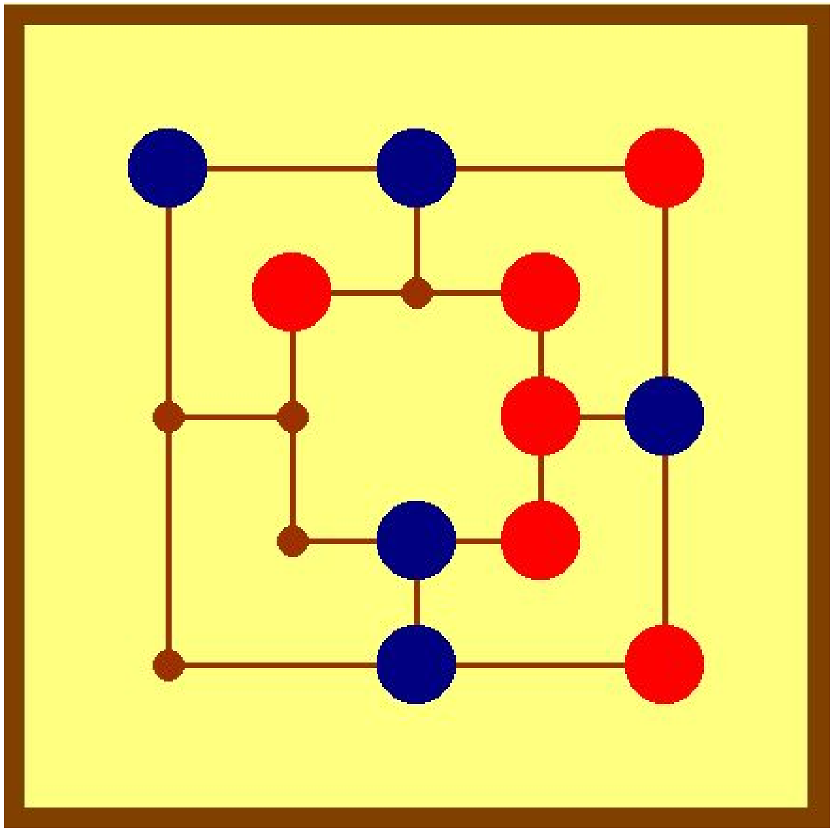
SIX MEN'S MORRIS

Patricia Fong and Kevin Liu
Gamescrafters

December 6, 2006



Contents

1 Game Rules

Six Men's Morris is similar to Nine Men's Morris except that each player has 6 pieces each rather than 9 pieces and the board is smaller.

The objective of the game is the have the opponent left with 2 pieces.

The game starts with an empty board. Each player is equipped with 6 pieces and takes turns placing them down anywhere on the board. This stage is called stage 1. When the player places a piece that creates a three pieces in a row from the same player, the player is allowed to remove an opponent's piece that is not in a three in a row with pieces from the opponent. If all pieces are in a three in a row with pieces from the opponent, then the player is allowed to take any piece. When each player has no more pieces to place on the board, stage 2 begins.

Stage 2 only allows pieces to move to adjacent empty slots. When the player moves to a slot that creates a three pieces in a row from the same player, the player is allowed to remove an opponent's piece that is not in a three in a row with pieces from the opponent. If all pieces are in a three in a row with pieces from the opponent, then the player is allowed to take any piece.

When a player is reduced to three pieces, those pieces may fly from any intersection to any intersection. This appears to be a powerful resource for an underdog, but in fact rarely changes the outcome of a game.

The game ends when a player only has 2 pieces on the board.

2 Design Decisions

We first wrote a program independent of GAMESMAN to test a player to player game of 9 men's morris. After that, we reorganized the code to fit the GAMESMAN's template. We also reused some functions from the previous authors to help with the generateMoves function. There are some variables that have to be remembered and are stored in the tier number.

3 Functions

3.1 updatepieces()

```
POSITION updatepieces(char* board,char turn,int piecesLeft,int numx,int
numo,MOVE move, POSITION position)
```

This function updates piecesLeft, numx, and numo and hashes the board.

- `board` represents the board in character form

- `turn` is whose turn it is represented by a 'X' or 'O'
- `piecesLeft` is the number of pieces left to place on the board of both players combined
- `numx` is the number of 'X' [player one] pieces on the board
- `numo` is the number of 'O' [player two] pieces on the board
- `move` is the action done on the board represented in a numerical form
- `position` represents a board that has been hashed

3.2 can_be_taken()

`BOOLEAN can_be_taken(POSITION position, int slot)`

this function returns TRUE when the slot given is a piece that the player can take when the player makes a 3 in a row.

- `position` represents a board that has been hashed
- `slot` represents the location on the board

3.3 all_mills()

`BOOLEAN all_mills(char *board, int slot)`

This function returns TRUE when all the player's pieces are part of a 3 in a row.

- `board` represents the board in character form
- `slot` represents the location on the board

3.4 find_piece()

`int find_pieces(char *board, char piece, int *pieces)`

This function finds the number of pieces that a given player has on the board and returns this number and an array containing the position of the pieces on the board.

- `board` represents the board in character form
- `(`piece) is the type of piece ('X' or 'O') we want to find the piece of
- `pieces` is an array of positions of the pieces on the board

3.5 closes_mill()

`BOOLEAN closes_mill(POSITION position, int raw_move)`

This function returns TRUE if the move will create a three in a row.

- `position` represents a board that has been hashed
- `move` is the action done on the board represented in a numerical form

3.6 check_mill()

`BOOLEAN check_mill(char *board, int slot, char turn)`

This function checks and returns TRUE if a slot will create a three in a row by providing the adjacent pieces to the slot and calling three_in_a_row.

- `board` represents the board in character form
- `slot` represents the location on the board
- `turn` is whose turn it is represented by a 'X' or 'O'

3.7 three_in_a_row()

`BOOLEAN three_in_a_row(char *board, int slot1, int slot2, char turn)`

This function checks and returns TRUE if all three slots are filled with a piece of the given player.

-
- `board` represents the board in character form
- `slot1` represents the location on the board
- `slot2` represents the location on the board
- `turn` is whose turn it is represented by a 'X' or 'O'

3.8 find_adjacent()

`int find_adjacent(int slot, int *slots)`

This function returns the number and array of adjacent pieces that a piece can move to.

- `slot` represents the location on the board
- `(`slots) represents an array of locations on the board

3.9 EvalMove()

`POSITION EvalMove(char* board,char turn,int piecesLeft,int numx,int numo,MOVE move, POSITION position)`

This function is similar to doMove but is used more efficiently since it does not have to call unhash.

- `board` represents the board in character form
- `turn` is whose turn it is represented by a ’X’ or ’O’
- `piecesLeft` is the number of pieces left to place on the board of both players combined
- `numx` is the number of ’X’ [player one] pieces on the board
- `numo` is the number of ’O’ [player two] pieces on the board
- `move` is the action done on the board represented in a numerical form
- `position` represents a board that has been hashed

4 Move Semantic

For stage one:

- If only placing piece:
  *MOVE = slot_to_place_piece * BOARDSIZE * BOARDSIZE + slot_to_place_piece * BOARDSIZE + slot_to_place_piece*
- If placing piece that makes a 3-in-a-row and removing an opponent’s piece:
  *MOVE = slot_to_place_piece * BOARDSIZE * BOARDSIZE + remove_piece * BOARDSIZE + slot_to_place_piece*

For stage two:

- If only moving a piece:
  *MOVE = slot_with_piece * BOARDSIZE*BOARDSIZE + slot_to_move_to * BOARDSIZE + piece_to_move*
- If moving a piece that makes a 3-in-a-row and removing an opponent’s piece:
  *MOVE = slot_with_piece * BOARDSIZE * BOARDSIZE + slot_to_move_to * BOARDSIZE + remove_piece*

-BOARDSIZE is equal to the number of positions on the board, which is 16.

5 Tier Semantic

Given a number in the form: **XXYZ**
I.E.: 1200, 534, 66

XX - Number of pieces left to place on the board of both players combined. The maximum is 12, since each player initially gets 8 pieces to place onto the board.
Y - Number of pieces Player One has on the board. The maximum is 6.
Z - Number of pieces Player Two has on the board. The maximum is 6.

6 Options Supported

- MoveToString
- Tier

7 Typical Execution Sequence

- **GenerateMove()**
  Calls:
  - find_adjacent()
  - closes_mill()
  - can_be_taken()
  - CreateMovelistNode()
- **DoMove()**
  Calls:
  - unhash()
  - updatepieces()
- **Primitive()**
  Calls:
  - unhash()
  - GenerateMoves()
- **PrintPosition()**
  Calls:
  - unhash()
- **GetAndPrintPlayersMove()**
  Calls:
  - unhash()

  - HandleDefaultTextInput()

- **updatepieces()**
  Calls:
  - unhash()
- **can_be_taken()**
  Calls:
  - unhash()
  - all_mills()
  - check_mill()
- **all_mills()**
  Calls:
  - find_pieces()
  - check_mill()
- **closes_mill()**
  Calls:
  - unhash()
  - EvalMove()
  - check_mill()
- **EvalMove()**
  Calls:
  - updatepieces()
- **check_mill()**
  Calls:
  - three_in_a_row()