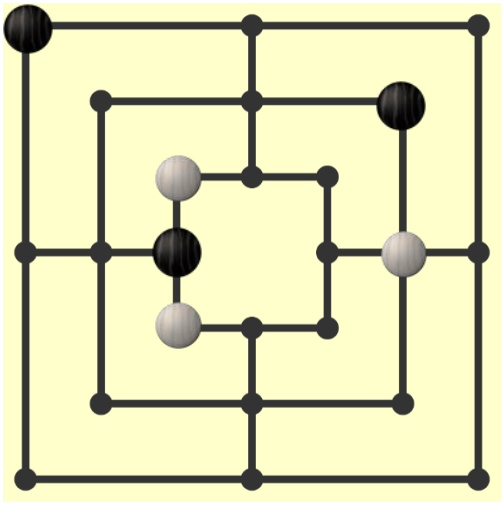
Programmer's Documentation

Mill



Markéta Sauerová
Univerzita Karlova MFF
letní semestr 2018/2019
Programování II.

Contents

1. Brief Description

Mill is an implementation of the board game "Nine Men's Morris". Other names for this game are: Mills, Merrills, Morris, or Mühle in German. Nine Men's Morris, being probably 2000-3000 years old, appears to be one of the oldest board games, much older than chess.

This implementation offers the classic game, where the board consists of a grid with twenty-four points. Each player has nine pieces, or "men", coloured black and white.

Players try to form 'mills'—three of their own men lined horizontally or vertically—allowing a player to remove an opponent's man from the game. A player wins by reducing the opponent to two pieces.

The game proceeds in three phases according to rules:

1. Placing men on vacant points
2. Moving men to adjacent points
3. (optional phase) Moving men to any vacant point when the player has been reduced to three men

Furthermore, player can choose whether he wants to play with another player or take turns with artificial intelligence.

Project is written as a windows form application in C#.

2. Project Structure

In the Mill solution are situated three folders*, AI, bin* and *BoardEvaluation*. The *bin* folder contains *images* folder with all embedded bitmaps (Properties.Resources), used in the solution.

On the contrary *AI* and *BoardEvaluation* represent extension to the Mill solution. Both have to be aded to the using block of the code before individual .cs files are accesible through code.

AI consists of AlfaBetaAI, MinimaxAI and TakeStoneHeuristic classes, for implementation of the artifical intelligence, based on a basic search problem involving game tree with each node representing a game position and each edge representing a move made by a player. This problem is solved by the minimax algorithm to the defined depth, because of the huge size of Nine Men's Morris's game tree. To further improved performance there is a slightly modified version of the recursive minimax algorithm implemented, called recursive minimax with alpha-beta pruning.

BoardEvaluation contains of interface IboardEvaluationHeuristic located in the BoardEvaluationInterface.cs file, which implements two classes PiecesCountEvaluation and MillsCountGameEvaluation (Heuristics.cs) reresenting two heuristics for evaluating a board state.

Next are the individual files with classes taking care of the smooth continuation of the game. Namely, Program with the main function, entry point of the whole application. Board

initializing game board and containing methods for maintenance of the board as well as obtaining useful information about the board. Linking, static class used for linking code logic with form logic and making the code more readable and understandable. Move and Mill classes creating structured objects as needed.

Lastly, the solution also contains the Form1.cs file with partial classes Form1 and Form1.Designer. Main thread is maintained in the Form1 class, so that the application would run witout difficulties. On the other hand, design of the application is mostly generated in the Form1.Designer class with adjustments, for instance labels for the specified points on the board are created by for loop.

This documentation is organized according to the given structure above.

2.1. Artificial Intelligence algorithms

As I mentioned before, intelligence is based on the search game tree problem, solving with recursive minimax algorithm for specified depth, because of the number of possible moves in the Mill game. For improved performance recursive minimax with alpha-beta pruning is also implemented. When AI formes a mill, the heuristic for taking oponent‘s stone is constructed as a random picker from the oponent’s almost mill formation (two stones in row or column), which can easily become mill. For evaluation of the board state, there are currently two simple heuristics.

2.1.1. MinimaxAI.cs

- `public MinimaxAI(IBoardEvaluationHeuristic boardEvaluationHeuristic)`
  Constructor of the class, taking interface value, set by user, for current evaluation heuristic.
- `public Move AIMinimaxMove(Board board, bool blackIsPlaying)`
  Calls minimax algorithm and defines move result of type Move, obtained with minimax, through global variables nextMinMove and nextMaxMove . Min if black, Max if white.
- `private int Minimax(Board board, int depth, bool blackIsPlaying, Board.PlaceOnBoardIs[,] currentBoard)`
  Recursive minimax algorithm. Creates game tree to the const depth 5. Returns evaluation of every position and sets global nextMaxMove and nextMinMove variables.
- `private Board.PlaceOnBoardIs[,] TakeStone(Board.PlaceOnBoardIs[,] board, bool blackIsPlaying)`
  Takes stone from current board. Used in minimax algorithm, when mill formation occurs.

2.1.2. AlfaBetaAI.cs

- `public AlfaBetaAI(IBoardEvaluationHeuristic boardEvaluationHeuristic)`
  Constructor of the class, taking interface value, set by user, for current evaluation heuristic.

- `public Move AIAlfaBetaMove(Board board, bool blackIsPlaying)`
  Calls minimax algorithm with alpha-beta pruning and defines move result of type Move, obtained with alpha-beta, through global variables nextMinMove and nextMaxMove. Min if black, Max if white.
- `private int AlfaBeta(Board board, int depth, int alpha, int beta, bool blackIsPlaying, Board.PlaceOnBoardIs[,] currentBoard)`
  Recursive minimax algorithm with alpha-beta pruning. Creates game tree to the const depth 5. Returns evaluation of every position and sets global nextMaxMove and nextMinMove variables.
- `private Board.PlaceOnBoardIs[,] TakeStone(Board.PlaceOnBoardIs[,] board, bool blackIsPlaying)`
  Takes stone from current board. Used in minimax algorithm with alpha-beta pruning, when mill formation occurs.

2.1.3. TakeStoneHeuristic

- `public Tuple<int, int> ChooseWhichStone(bool blackIsPlaying, Board.PlaceOnBoardIs[,] board)`
  Uses randomly generated numbers to determine which stone should be taken from the board. The stone has to be from an array of almost mill formations (two pieces of the same color vertically or horizontally, easily creates a mill). Returns tuple with indexes of the chosen stone. Uses AlmostMill and GetOccupiedPositions methods.

2.1.4. BoardEvaluationHeuristic

- `public class PiecesCountGameEvaluation: IBoardEvaluationHeuristic`
  Implements interface for evaluating board state, used in minimax and alpha-beta algorithms. Evaluates by counting number of black and white stones on the board and subtracting them. Uses NumberOfStonesCurrentlyOnBoard method.
- `public class MillsCountGameEvaluation: IBoardEvaluationHeuristic`
  Implements interface for evaluating board state, used in minimax and alpha-beta algorithms. Evaluates by counting of possible mills. Uses CountMills method.

2.2. Game Logic

The Program class as the entry point of the program contains solely main function, from where the Form is initialized. After that, main game logic is mostly maintained in the Form1 partial class, so that the application runs smoothly.

Linking class provides needed linkage between program board (2D array of PlaceOnBoardIs enum) and form board, controled mainly through labels.

2.2.1. Board

Representation of the playing board is by an 2D array of enum type :

`PlaceOnBoardIs {free, whiteOccupied, blackOccupied, movable, notMovable}`

Created with CreateBoard method, after initialization of form components.

In GetAdjacentVacantPositions method is searching for adjacent vacant positions done by going in 4 directions (left, right, up, down) from the current place, until free, whiteOccupied, blackOccupied, notMovable or the edge occurs.

Tuple<int,int> is usually used for indexes of 2D array.

In form, board is represented by background image and 24 clickable labels as places for stones.

2.2.2. Mills

Mills are represented as structures in individual class, consisting of 3 tuples holding two integers, each one for indexes of one stone in the created mill, and boolean, decisive if the mill is white or black. They are stored in global arrays whiteMills and blackMills.

Their type is for reading only, editable only whole with class constructor. Can be compared with implemented Equal method.

- `public bool IsMill(PlaceOnBoardIs[,] board, Mill[] whiteMills, Mill[] blackMills)`
  Method for detecting mill occurrences. Returns true if Mill was created, false otherwise. Uses ReportMill method for detecting mills on the board and MillLogic for not detecting already existed mills.
- `private void DeleteAllMillsWithStone(int x, int y, Mill[] mills)`
  Deletes all mills with given stone (indexes) from given mill array.

2.2.3. Moves

Moves are represented as structures in individual class, consisting of two tuples of indexes to positions From and To on board and two booleans whether the move is played by black or white and human or AI player.

Only allowed property for editing is From tuple. Else have to be modified through constructor.

2.2.4. Form design

User interface in Mill project is only the initialized form, with strip menu for choosing game option, such as human vs human, AI vs human, human vs AI or AI vs AI. Textbox for keeping the player updated on continuation of the game. Two buttons for playing again (restarts the application) and next move, used when the AI player is on turn. And finally, 24 labels used for obtaining played positions from human player.

2.2.5. Game

When playing the current GamePhase is always known as type enum of { opening, midPhase, finishing } options.

Color of the current player is maintained by blackIsPlaying boolean, just like human and AI player takes turns by AIturn boolean.

The game itself is then simulated by various methods and intern logic, e.g. OpeningPhase, MidGamePhase, PutStoneOnBoard, TakeStoneFromBoard, CanStoneBeSlided, SlideStone etc...

Game can be won by reducing opponent's stones to two, when he can no longer create mill. Then the GameWon method disenables all the labels and overwrites Textbox with winning message.

3. User's Guide

Abstract

Mill is an implementation of the board game called Nine Men's Morris in C#, using windows forms. Due to its simple rules it's suitable for players of all ages. Program offers you option for playing against an artificial intelligent opponent.

3.1. How to play

3.1.1. Choosing player vs player option

The options can be chosen form strip menu at the top of the application, they have to be done before playing starts. By default, human vs human is set. If you want change player option but already started game, just hit the play again button for restarting the application.

When AI vs human or human vs AI is chosen, then player will be asked to press Next Move button on the top of the application every time an AI player should play.

AI vs AI option is automated.

3.1.2. Phase 1: Placing pieces

The game begins with an empty board. The player determines who plays first, then take turns placing their men one per play on empty points. First stone placed at board is white and second is black. If a player is able to place three of their pieces on contiguous points in a straight line, vertically or horizontally, they have formed a mill and may remove one of their opponent's pieces from the board. After all men have been placed, phase two begins

3.1.3. Phase 2: Moving pieces

Players continue to alternate moves, this time moving a man to an adjacent point. A stone may not "jump" another piece. When the stone is chosen for moving and the choice is valid, it§s also ultimate and then no other stone can be chosen. Players continue to try to form mills and remove their opponent's stones as in phase one. A player can "break" a mill by

moving one of his stone out of an existing mill, then moving it back to form the same mill a second time (or any number of times), each time removing one of his opponent's men. When one player has been reduced to three men, phase three begins.

3.1.4. Phase 3: "Flying"

When a player is reduced to three stones, there is no longer a limitation on that player of moving to only adjacent points: The player's men may "fly" from any point to any vacant point. Game is won when one of the players has less than three stones.