Connect-Toe Program User Manual

Martha Mercaldi, Katarzyna Wilamowska

November 10, 2004

Contents

1 Introduction

This is a short guide on your way through using the Connect-Toe Program. The Connect-Toe Program works only in Unix/Linux based systems. There is an executable included in the code folder. The program is called **Main**. No private member functions or variables are listed in this user guide.

2 Getting started

2.1 Compilation

In command line type **make** to compile all code. **make debug** turns on extensive debugging output.

2.2 Command line attributes

Unless otherwise specified, values passed through the command line are integers. Since there are **no defaults**, all attributes must be set for proper program functionality.

-dimensions *sizeD1 sizeD2 sizeD3 . . . sizeDn*

-openFaces *faceId1 faceId2 . . . faceIdn*

-cutoff *searchDepth*

-winLength *lengthRequiredToWin*

-playerA *heuristic*

-playerB *heuristic*

Where *heuristic* is either O, P, U, or R.

2.2.1 Example use - Traditional Tic Tac Toe

```
Main -dimensions 3 3 1 -openFaces 2 -cutoff 3 -winLength 3 -playerA O -playerB U
```

2.2.2 Example use - Traditional Connect Four

```
Main -dimensions 6 7 1 -openFaces 1 -cutoff 2 -winLength 4 -playerA O -playerB U
```

3 Class definitions

3.1 Move

3.1.1 SquareId

A vector of integers that uniquely identifies each square in a restricted set. The least restricted of these sets is the board set, where SquareId holds all the squares on the board. A more restrictive example is using SquareId to pass a single column on the board.

3.1.2 Constructors

The Move class has three constructors and one copy constructor.

Move()
: dummy constructor

Move(char, int, SquareId)
: character to specify player
integer to specify the open face into which the token is placed
SquareId to specify where in the open face the token goes

Move(char, int, SquareId, int)
: character to specify player
integer to specify the open face into which the token is placed
SquareId to specify where in the open face the token goes
integer representing the board score with the move

Move(Move)
: copy constructor

3.1.3 Helper functions

GetFace()
: Returns the face from which a token is dropped.

GetPlayer()
: Returns player executing the move.

GetScore()
: Returns the board score of the move. The score of the board is a value calculated externally to this class.

GetColumn()
: Returns a SquareId identifying the position in the open face.

setPlayer(char)
: Sets character given to player.

setScore(int)
: Sets integer value to the score of the move.

3.1.4 Print

Print()
: Prints a move. Example output:

```
[Player: A, Face: 2 (0,2,2) SCORE = -1]
```

This output is a print of a move by player A, token dropped from face 2, landing in position 0, 2, 2 (3D), with this move leading to A losing the game.

3.2 Board

3.2.1 possibleMoves

A vector containing all possible moves on the board. This vector contains all moves without discriminating between legal and illegal moves.

3.2.2 Constructors

Board()
: dummy constructor

Board(int)
: integer defines the number of dimensions that board will have

Board(Board)
: copy constructor

3.2.3 Helper functions

void SetDimensionSize(int, int)
: Creates a dimension given which dimension it is and how large it must be.

int GetStoppingIndex(Move)
: Finds out where in the "falling" dimension will the player's piece rest.

void MakeMove(Move)
: Assumes the move is legal and then applies the move to the board.

bool HasRun(int, bool)
: Checks if there is a run on the board for either player in the length given by the integer of the function. Depending on the boolean value being true or false (typically used for debug purposes), the function prints out the player with the run, the starting square and the direction of the run.

bool IsWin(int, char)
: Checks if a specified player has a run. If the player has a run, then return true, else false.

bool IsDraw()
: Checks if there is a full board where neither player has won.

int numPiecesA()
: Returns the number of pieces owned by player A on the board.

int numPiecesB()
: Returns the number of pieces owned by player B on the board.

bool fullBoard()
: Returns true if there are no more empty spots on the board.

void SetOpenFace(int)
: Sets an face specified by the integer value to be open for moves.

bool IsOpenFace(int)
: Returns true if specified face is open, false otherwise.

bool IsLegal(Move)
Returns true is move is legal, false otherwise.

int getNumSquares()
Returns the number of squares in the board.

3.2.4 Print

void Print()
Prints the board status. Output example given below is a Tic Tac Toe board.

```
-   B   A
-   A   B
A   A   -
```

3.3 Game

3.3.1 Constructors

Game(int, int)
constructs a game with a specified number of dimensions and winning length

3.3.2 Helper functions

Move Random(char, Board)
Returns a randomly chosen move for a given player and board.

Move MiniMax(char, Board, int, char)
Returns a calculated move for a given player, board, depth boundary and heuristic.

int minimaxHelper(char, Board, Move, bool, int, char)
This is the recursive function for MiniMax taking in the player, board, move, whether we are in a min or max state, depth boundary and heuristic.

Move playToLoss(char, Board, int)
Returns a calculated move for a given player, board, depth boundary with miniMax, but then multiplying the values by negative one.

vector<Move> getLegalMoves(char, Board)
Returns the set of all legal moves for a player and a given board.

Board newBoardWithMove(Board, Move)
Returns a new board with the new move in it.

int optimisticBoardScore(Board, char)
Utility function that returns an optimistic view of the board.

int pessimisticBoardScore(Board, char)
Utility function that returns a pessimistic view of the board.

int uncertainBoardScore(Board, char)
Utility function that returns a undecided/don't care view of the board.