

Reversi Reference Manual

0.1

Author: Jan Bartipan

Generated by Doxygen 1.3.7

Fri May 14 02:44:24 2004

Contents

1	Reversi Hierarchical Index	1
2	Reversi Class Index	1
3	Reversi Class Documentation	2

1 Reversi Hierarchical Index

1.1 Reversi Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AITree	4
Board	5
Game	11
Grid	12
Player	17
AIPlayer	2
HumanPlayer	15
TreeNode	19

2 Reversi Class Index

2.1 Reversi Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

AIPlayer (AI player class)	2
AITree (Testing class to draw solution treee)	4
Board (Board class representing state of game)	5
Game (Game class)	11
Grid (Class for board visualisation)	12
HumanPlayer (Human player class)	15
Player (Abstract class for player)	17
TreeNode (This class represents the node of solution tree)	19

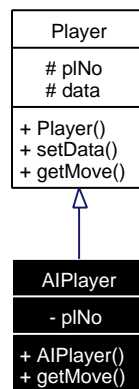
3 Reversi Class Documentation

3.1 AIPlayer Class Reference

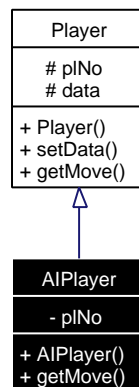
AI player class.

```
#include <player.h>
```

Inheritance diagram for AIPlayer:



Collaboration diagram for AIPlayer:



Public Member Functions

- **AIPlayer** (unsigned char no)
Standard constructor for AIPlayer class.
- virtual void **getMove** (int *x, int *y, **Grid** &grid)
Returns computer move.

3.1.1 Detailed Description

AI player class.

This class represents interface for computer player input.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 AIPlayer::AIPlayer (unsigned char *no*)

Standard construcotr for AIPlayer class.

Parameters:

no The nubmer of player.

Here is the call graph for this function:



3.1.3 Member Function Documentation

3.1.3.1 void AIPlayer::getMove (int * *x*, int * *y*, Grid & *grid*) [virtual]

Returns computer move.

Parameters:

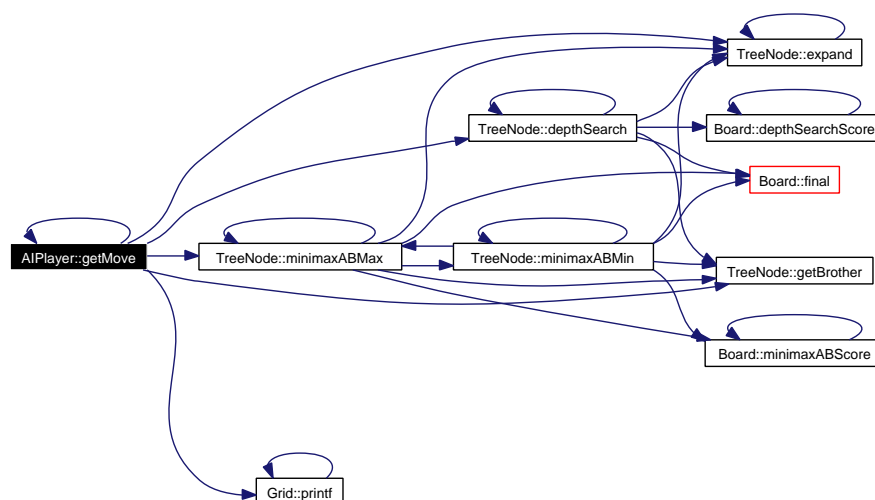
x The x position of player move.

y The y position of player move.

grid The grid data.

Implements **Player** (p.18).

Here is the call graph for this function:



The documentation for this class was generated from the following files:

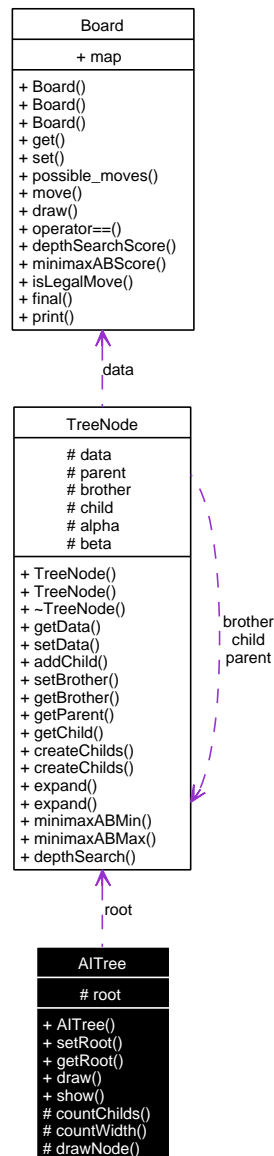
- player.h
- player.cpp

3.2 AITree Class Reference

Testing class to draw solution tree.

```
#include <tree.h>
```

Collaboration diagram for AITree:



Public Member Functions

- void **setRoot** (**TreeNode** *new_root)
- **TreeNode** * **getRoot** ()
- void **draw** (BITMAP *b, int field_width, int grid_space)

Static Public Member Functions

- void **show** (**Board** data, unsigned char plNo)

Protected Member Functions

- int **countChilds** (**TreeNode** *node)
- int **countWidth** (**TreeNode** *node, int grid_space)
- int **drawNode** (BITMAP *b, **TreeNode** *node, int off_x, int off_y, int field_width, int grid_space)

Protected Attributes

- **TreeNode** * **root**

3.2.1 Detailed Description

Testing class to draw solution treee.

The documentation for this class was generated from the following files:

- tree.h
- tree.cpp

3.3 Board Class Reference

Board class representing state of game.

```
#include <board.h>
```

Public Member Functions

- **Board** ()
Standard constructor.
- **Board** (int *map)
Standard constructor with initialization.
- **Board** (int *map, int x, int y, int n)
Standard constructor with initialization and setting position value.
- int **get** (int x, int y)
Gets value of position.

- void **set** (int x, int y, int n)
Sets value of position.
- vector< **Board** > **possible_moves** (int pl_no)
Returns vector of boards of possible moves for player pl_no.
- void **move** (int pl_no)
Sets new move mark to player value.
- void **draw** (BITMAP *b, int off_x, int off_y, int field_width)
Draws the board on screen (bitmap).
- bool **operator==** (**Board** &b)
Compares a board with itself.
- int **depthSearchScore** (unsigned char pl, int moveAt)
Score of board for depth search.
- int **minimaxABScore** (unsigned char pl)
Score of board for minimax search.
- bool **isLegalMove** (int px, int py, int pl_no)
Tests if is legal move.
- bool **final** ()
Tests if this board has no possible moves for any player.
- void **print** ()
Prints debug information.

Public Attributes

- int **map** [boardWidth *boardHeight]
Map of board.

3.3.1 Detailed Description

Board class representing state of game.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Board::Board ()

Standard constructor.

Initializes the map array to zeros.

3.3.2.2 Board::Board (int * *map*)

Standard constructor with initialization.

Construct object and initialize the map attribute with new data.

Parameters:

map Data to initialize from.

3.3.2.3 Board::Board (int * *map*, int *x*, int *y*, int *n*)

Standard constructor with initialization and setting position value.

Initializes map attribute with data. And then set value of member at x,y position.

Parameters:

map Data to initialize from.

x The x coordinate of member to be set.

y The y coordinate of member to be set.

n The value of member to be set.

Here is the call graph for this function:

**3.3.3 Member Function Documentation****3.3.3.1 int Board::depthSearchScore (unsigned char *pl*, int *moveAt*)**

Score of board for depth search.

This returns a score for the board, for player and his move.

Parameters:

pl The player number.

moveAt The last move of player.

Returns:

The score of board.

Here is the call graph for this function:

**3.3.3.2 void Board::draw (BITMAP * *b*, int *off_x*, int *off_y*, int *field_width*)**

Draws the board on screen (bitmap).

This draws board in bitmap.

Parameters:

- b* The bitmap into will be drawn.
- off_x* The x offset in bitmap.
- off_y* The y offset in bitmap.
- field_width* The width of field (in pixels)

Here is the call graph for this function:

**3.3.3.3 bool Board::final ()**

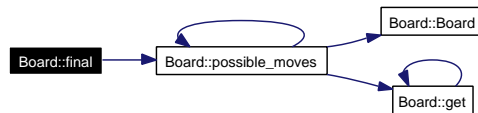
Tests if this board has no possible moves for any player.

This function determinate if can player one or player two move on this board. In other words this check if this is final state of game.

Returns:

True if player one and player two can't move.

Here is the call graph for this function:

**3.3.3.4 int Board::get (int x, int y)**

Gets value of position.

Returns the map attribute value of x, y position.

Parameters:

- x* The x coordinate of member to get.
- y* The y coordinate of member to get.

Returns:

The value of map attribute at position.

Here is the call graph for this function:



3.3.3.5 bool Board::isLegalMove (int *px*, int *py*, int *pl_no*)

Tests if is legal move.

Tests if player number *pl* can legally put stone on *px*,*py* position of the board/

Parameters:

px The x position of player move.

py The y position of player move.

pl_no The nubmer of player.

Returns:

legallity of move.

Here is the call graph for this function:

**3.3.3.6 int Board::minimaxABScore (unsigned char *pl*)**

Score of board for minimax search.

This returns a score of the board, for player. It's used for minimax search.

Parameters:

pl The player number.

Returns:

The score of board.

Here is the call graph for this function:

**3.3.3.7 void Board::move (int *pl_no*)**

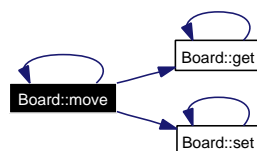
Sets new move mark to player value.

This search the map attribute until it finds the mark of new player move (= -1). Than this possiton sets to value, that represents the occupation of this filed for player *pl_no*.

Parameters:

pl_no The number of player the move belongs.

Here is the call graph for this function:



3.3.3.8 bool Board::operator== (Board & b)

Compares a board with itself.

Parameters:

b The board to be compared with.

Here is the call graph for this function:

**3.3.3.9 vector< Board > Board::possible_moves (int pl_no)**

Returns vector of boards of possible moves for player pl_no.

This returns a STL vector of boards. This vector represents possible moves for player with number pl_no.

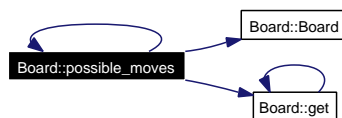
Parameters:

pl_no The nubmer of player for generating possible moves.

Returns:

A STL vector of possible_moves (boards) for player pl_no.

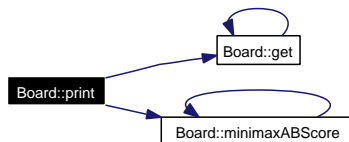
Here is the call graph for this function:

**3.3.3.10 void Board::print ()**

Prints debug information.

This prints debug information of state of this board on standard output.

Here is the call graph for this function:

**3.3.3.11 void Board::set (int x, int y, int n)**

Sets value of position.

Sets the map attribute value of x, y position.

Parameters:

- x The x coordinate of memeber to set.
- y The y coordinate of memeber to set.
- n The value of member to set.

Here is the call graph for this function:

**3.3.4 Member Data Documentation****3.3.4.1 int Board::map[boardWidth *boardHeight]**

Map of board.

This is array of int. Every member represents if this position is occupied by player one (=1), by player two (=2), or if it's free (=0). There's a special mark (= -1) that stands for position of next move of player.

The documentation for this class was generated from the following files:

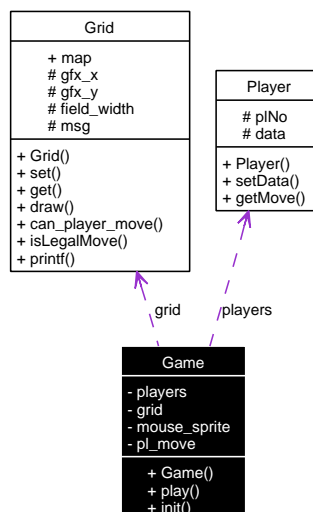
- board.h
- board.cpp

3.4 Game Class Reference

Game class.

```
#include <game.h>
```

Collaboration diagram for Game:



Public Member Functions

- **Game** ()
Standard constructor.
- void **play** ()
Runs game.

Static Public Member Functions

- void **init** ()
Initialization of game.

3.4.1 Detailed Description

Game class.

The documentation for this class was generated from the following files:

- game.h
- game.cpp

3.5 Grid Class Reference

Class for board visualisation.

```
#include <grid.h>
```

Public Member Functions

- **Grid** ()
Standard constructor.
- void **set** (int x, int y, char pl)
Sets board data of grid.
- char **get** (int x, int y)
Returns board data at position x,y.
- void **draw** ()
Draws grid on screen.
- bool **can_player_move** (int pl_no)
Tests if player can move.
- bool **isLegalMove** (int px, int py, int plNo)
Checks if player's move at x,y is legal.

- void **printf** (const char *fmt,...)
Sets up message.

Public Attributes

- int **map** [boardSize]
The board data of grid.

Protected Attributes

- int **gfx_x**
The x screen offset of grid.
- int **gfx_y**
The y screen offset of grid.
- int **field_width**
The grid field width.
- string **msg**
The message string to be displayed.

3.5.1 Detailed Description

Class for board visualisation.

This class is used to draw board on screen.

3.5.2 Member Function Documentation

3.5.2.1 bool Grid::can_player_move (int *pl_no*)

Tests if player can move.

Parameters:

pl_no The number of player to test.

Returns:

True if player can move, else otherwise.

Here is the call graph for this function:



3.5.2.2 char Grid::get (int *x*, int *y*)

Returns board data at position *x*,*y*.

Parameters:

x The x field position.

y The y field position.

Returns:

The value of grid at *x*,*y* position.

Here is the call graph for this function:

**3.5.2.3 bool Grid::isLegalMove (int *px*, int *py*, int *plNo*)**

Checks if player's move at *x*,*y* is legal.

Parameters:

px The player move x coordinate.

py The player move y coordinate.

plNo The nubmer of player.

Returns:

Legallity of player move.

Here is the call graph for this function:

**3.5.2.4 void Grid::printf (const char * *fmt*, ...)**

Sets up message.

This method sets message string. It's simillar like C printf function.

Parameters:

fmt The formating string.

Here is the call graph for this function:



3.5.2.5 void Grid::set (int *x*, int *y*, char *pl*)

Sets board data of grid.

Parameters:

- x* The x field position of move.
- y* The y field position of move.
- pl* The value of field.

Here is the call graph for this function:



The documentation for this class was generated from the following files:

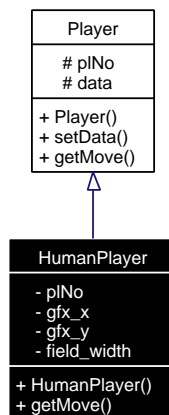
- `grid.h`
- `grid.cpp`

3.6 HumanPlayer Class Reference

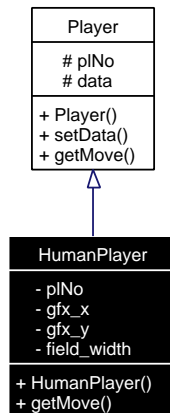
Human player class.

```
#include <player.h>
```

Inheritance diagram for HumanPlayer:



Collaboration diagram for HumanPlayer:



Public Member Functions

- **HumanPlayer** (int off_x, int off_y, int field_width, unsigned char no)
Standard constructor for HumanPlayer class.
- virtual void **getMove** (int *x, int *y, **Grid** &grid)
Returns user input.

3.6.1 Detailed Description

Human player class.

This class represents interface for player input.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 HumanPlayer::HumanPlayer (int off_x, int off_y, int field_width, unsigned char no)

Standard constructor for HumanPlayer class.

Creates new HumanPlayer object. This class interact with user to get user move.

Parameters:

- off_x* The x screen offset of grid.
- off_y* The y screen offset of grid.
- field_width* The width of grid field.
- no* The player nubmer.

Here is the call graph for this function:



3.6.3 Member Function Documentation

3.6.3.1 void HumanPlayer::getMove (int * *x*, int * *y*, Grid & *grid*) [virtual]

Returns user input.

Parameters:

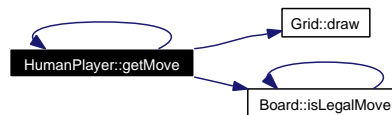
x The x position of player move. If player moves illegally, this is -1.

y The y position of player move. If player moves illegally, this is -1.

grid The grid data.

Implements **Player** (p.18).

Here is the call graph for this function:



The documentation for this class was generated from the following files:

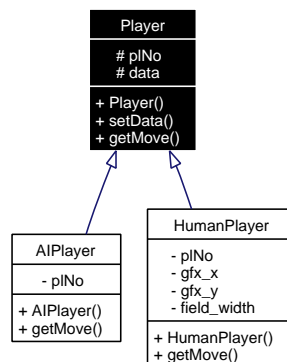
- player.h
- player.cpp

3.7 Player Class Reference

Abstract class for player.

```
#include <player.h>
```

Inheritance diagram for Player:



Public Member Functions

- **Player** (unsigned char no)
Standard constructor.

- void **setData** (int ***data**)
Sets data before getMove method call.
- virtual void **getMove** (int ***x**, int ***y**, **Grid** &**grid**)=0
Gets player move.

Protected Attributes

- unsigned char **plNo**
The number of a player.
- int **data** [boardSize]
Board(p. 5) *data.*

3.7.1 Detailed Description

Abstract class for player.

This class defines interface for player classes.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 **Player::Player** (unsigned char *no*) [inline]

Standard constructor.

Parameters:

no The number of player.

3.7.3 Member Function Documentation

3.7.3.1 **virtual void Player::getMove** (int * *x*, int * *y*, **Grid** & *grid*) [pure virtual]

Gets player move.

Parameters:

x The x position of player move.

y The y position of player move.

grid The grid data.

Implemented in **HumanPlayer** (p. 17), and **AIPlayer** (p. 3).

3.7.3.2 **void Player::setData** (int * *data*) [inline]

Sets data before getMove method call.

Parameters:

data **Board**(p. 5) data.

Here is the call graph for this function:



The documentation for this class was generated from the following file:

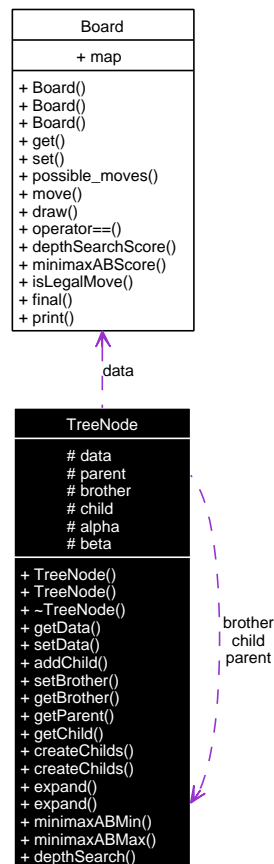
- player.h

3.8 TreeNode Class Reference

This class represents the node of solution tree.

```
#include <tree.h>
```

Collaboration diagram for TreeNode:



Public Member Functions

- **TreeNode (Board data)**
Standard constructor.

- **TreeNode (Board data, TreeNode *parent)**
Standard constructor to build children nodes.
- **virtual ~TreeNode ()**
Standard destructor.
- **Board getData ()**
Return state of node.
- **void setData (Board data)**
Sets stata data of node.
- **void addChild (Board data)**
Creates child with data initialization.
- **void setBrother (TreeNode *bro)**
Sets brother node.
- **TreeNode * getBrother ()**
Gets brother pointer.
- **TreeNode * getParent ()**
Gets parent pointer.
- **TreeNode * getChild ()**
Gets children pointer.
- **void createChilds (int count, Board *data)**
Creates node childrens with intialization.
- **void createChilds (vector< Board > data)**
Creates node childrens with intialization.
- **TreeNode * expand (int no)**
This expands the node for player number no.
- **TreeNode * expand (int no, vector< int > &moveAt)**
This expands the node for player number no.
- **float minimaxABMin (float a, float b, int no, int level)**
The minimax search method for minimal level.
- **float minimaxABMax (float a, float b, int no, int level)**
The minimax search method for maximal level.
- **float depthSearch (int level, int stopLevel, int no, int moveAt)**
The depth search method.

Protected Attributes

- **Board data**
State of game.
- **TreeNode * parent**
Pointer to parent node.
- **TreeNode * brother**
Pointer to brother node.
- **TreeNode * child**
Pointer to first child node.
- **float alpha**
The alpha value of node.
- **float beta**
The beta value of node.

3.8.1 Detailed Description

This class represents the node of solution tree.

This class encapsulate information of game state and other information to able to navigate through tree.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 `TreeNode::TreeNode (Board data)`

Standard constructor.

This constructor is used to create root node. It also initialize the state of node.

Parameters:

data The state of node.

Here is the call graph for this function:



3.8.2.2 `TreeNode::TreeNode (Board data, TreeNode * parent)`

Standard constructor to build children nodes.

This constructor is used to create children node. It also initialize the state of node.

Parameters:

data The state of node.

parent The point to parent node.

Here is the call graph for this function:



3.8.2.3 `TreeNode::~~TreeNode ()` [virtual]

Standard destructor.

This deallocates root node, all brothers and childrens.

3.8.3 Member Function Documentation

3.8.3.1 `void TreeNode::addChild (Board data)`

Creates child with data initialization.

This adds children node fot this node.

Parameters:

data The state data of children node.

Here is the call graph for this function:



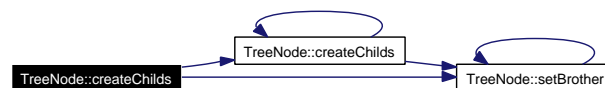
3.8.3.2 `void TreeNode::createChilds (vector< Board > data)`

Creates node childrens with intialization.

Parameters:

data The STL vector of children state datas.

Here is the call graph for this function:



3.8.3.3 `void TreeNode::createChilds (int count, Board * data)`

Creates node childrens with intialization.

Parameters:

count The count of childrens.

data The array of children state datas.

Here is the call graph for this function:



3.8.3.4 float TreeNode::depthSearch (int *level*, int *stopLevel*, int *no*, int *moveAt*)

The depth search method.

Parameters:

level Current level of node in tree.

stopLevel The number of level to stop expanding.

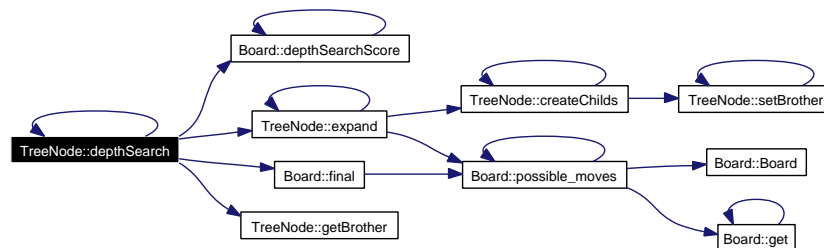
no The nubmer of player.

moveAt The position of last player move/

Returns:

Value of node.

Here is the call graph for this function:



3.8.3.5 TreeNode * TreeNode::expand (int *no*, vector< int > & *moveAt*)

This expands the node for player number no.

Parameters:

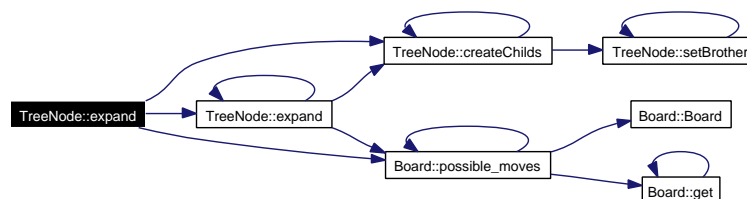
no The number of player.

moveAt The vector to be filled with moveAt position for each children.

Returns:

The pointer to first children node.

Here is the call graph for this function:



3.8.3.6 `TreeNode * TreeNode::expand (int no)`

This expands the node for player number no.

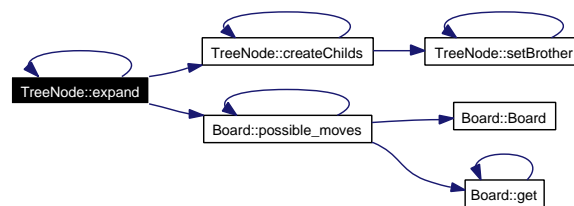
Parameters:

no The number of player.

Returns:

The pointer to first children node.

Here is the call graph for this function:

**3.8.3.7** `TreeNode * TreeNode::getBrother ()`

Gets brother pointer.

Returns:

Pointer to brother node.

3.8.3.8 `TreeNode * TreeNode::getChild ()`

Gets children pointer.

Returns:

Pointer to first children node.

3.8.3.9 `Board TreeNode::getData ()`

Return state of node.

Returns:

The state (board) of this node.

3.8.3.10 `TreeNode * TreeNode::getParent ()`

Gets parent pointer.

Returns:

Pointer to parent node.

3.8.3.11 float TreeNode::minimaxABMax (float *a*, float *b*, int *no*, int *level*)

The minimax search method for maximal level.

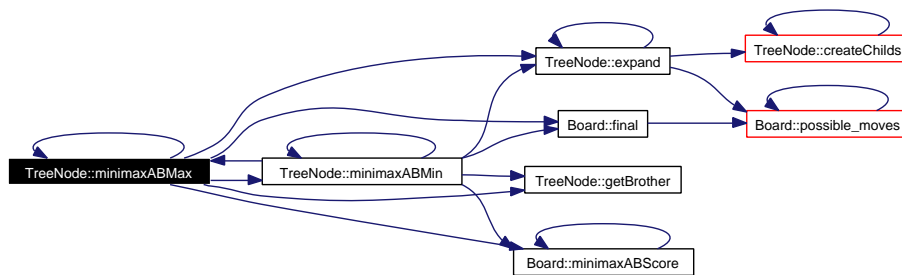
Parameters:

- a* Initial alpha value.
- b* initial beta value.
- no* The number of player the moves are generated for.
- level* Actual level of node in tree.

Returns:

Value of node.

Here is the call graph for this function:

**3.8.3.12 float TreeNode::minimaxABMin (float *a*, float *b*, int *no*, int *level*)**

The minimax search method for minimal level.

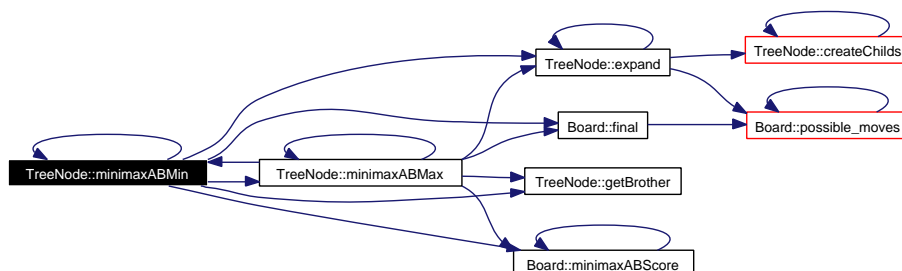
Parameters:

- a* Initial alpha value.
- b* initial beta value.
- no* The number of player the moves are generated for.
- level* Actual level of node in tree.

Returns:

Value of node.

Here is the call graph for this function:



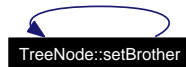
3.8.3.13 void TreeNode::setBrother (TreeNode * *bro*) [inline]

Sets brother node.

Parameters:

bro The pointer to new brother.

Here is the call graph for this function:

**3.8.3.14 void TreeNode::setData (Board *data*)**

Sets stata data of node.

Parameters:

data The new state data.

Here is the call graph for this function:



The documentation for this class was generated from the following files:

- `tree.h`
- `tree.cpp`

Index

- ~TreeNode
 - TreeNode, 22
- addChild
 - TreeNode, 22
- AIPlayer, 2
 - AIPlayer, 3
 - getMove, 3
- AITree, 4
- Board, 5
 - Board, 6, 7
 - depthSearchScore, 7
 - draw, 7
 - final, 8
 - get, 8
 - isLegalMove, 8
 - map, 11
 - minimaxABScore, 9
 - move, 9
 - operator==, 10
 - possible_moves, 10
 - print, 10
 - set, 10
- can_player_move
 - Grid, 13
- createChilds
 - TreeNode, 22
- depthSearch
 - TreeNode, 23
- depthSearchScore
 - Board, 7
- draw
 - Board, 7
- expand
 - TreeNode, 23, 24
- final
 - Board, 8
- Game, 11
- get
 - Board, 8
 - Grid, 13
- getBrother
 - TreeNode, 24
- getChild
 - TreeNode, 24
- getData
 - TreeNode, 24
- getMove
 - AIPlayer, 3
 - HumanPlayer, 17
 - Player, 18
- getParent
 - TreeNode, 24
- Grid, 12
 - can_player_move, 13
 - get, 13
 - isLegalMove, 14
 - printf, 14
 - set, 14
- HumanPlayer, 15
 - HumanPlayer, 16
- HumanPlayer
 - getMove, 17
 - HumanPlayer, 16
- isLegalMove
 - Board, 8
 - Grid, 14
- map
 - Board, 11
- minimaxABMax
 - TreeNode, 24
- minimaxABMin
 - TreeNode, 25
- minimaxABScore
 - Board, 9
- move
 - Board, 9
- operator==
 - Board, 10
- Player, 17
 - getMove, 18
 - Player, 18
 - setData, 18
- possible_moves
 - Board, 10
- print
 - Board, 10
- printf
 - Grid, 14
- set
 - Board, 10
 - Grid, 14

setBrother
 TreeNode, 25
setData
 Player, 18
 TreeNode, 26

TreeNode, 19
 TreeNode, 21
TreeNode
 ~TreeNode, 22
 addChild, 22
 createChilds, 22
 depthSearch, 23
 expand, 23, 24
 getBrother, 24
 getChild, 24
 getData, 24
 getParent, 24
 minimaxABMax, 24
 minimaxABMin, 25
 setBrother, 25
 setData, 26
 TreeNode, 21