Othello

Generated by Doxygen 1.8.11

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

Class Documentation

3.1 AI Class Reference

The AI class implements the alpha-beta pruning algorithm to find the next move for the computer player. Thereby three different heuristics (disc count, legal moves count, square rating), implemented in the evaluation funciton, are used.

`#include <ai.h>`

Public Member Functions

- **AI** (Board *board)
- void **makeRandomMove** ()
- double max (int depth, double alpha, double beta)

  max maximizing funciton for the black player of the the alpha-beta pruning algorithm.
- double min (int depth, double alpha, double beta)

  min minimizing function for the white player of the alpha-beta pruning algorithm.
- QPair< int, int > savedMove ()

  savedMove is used to get the best move, found by the alpha-beta pruning algorithm, inside the GameEngine class.

Public Attributes

- int m_startingDepth

  m_startingDepth maximum search depth defined by the user.

3.1.1 Detailed Description

The AI class implements the alpha-beta pruning algorithm to find the next move for the computer player. Thereby three different heuristics (disc count, legal moves count, square rating), implemented in the evaluation funciton, are used.

3.1.2 Member Function Documentation

3.1.2.1 double AI::max (int *depth*, double *alpha*, double *beta*)

max maximizing funciton for the black player of the the alpha-beta pruning algorithm.

Parameters

depth	user defined search depth in the beginning of the recursion.
alpha	maximum score that the maximizing player (black) is assured of.
beta	minimum score that the minimizing player (white) is assured of.

Returns

value of the evaluation function at depth zero or at a terminal node (board where current player has no more moves availabel).

3.1.2.2 double AI::min (int *depth*, double *alpha*, double *beta*)

min minimizing function for the white player of the alpha-beta pruning algorithm.

Parameters

depth	depth user defined search depth in the beginning of the recursion.
alpha	alpha maximum score that the maximizing player (black) is assured of.
beta	beta minimum score that the minimizing player (white) is assured of.

Returns

value of the evaluation function at depth zero or at a terminal node (board where current player has no more moves availabel).

3.1.2.3 QPair< int, int > AI::savedMove ( )

savedMove is used to get the best move, found by the alpha-beta pruning algorithm, inside the GameEngine class.

Returns

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

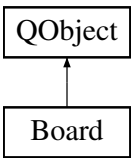
- ai.h
- ai.cpp

3.2 Board Class Reference

The Board class holds the board matrix and stores every board on a stack with the current and opponent player. It is used to find all allowed moves for the current player and actually execute them.

`#include <board.h>`

Inheritance diagram for Board:



Public Slots

- bool undoMove ()

  undoMove get the last board from the board stack.

Signals

- void signalBoardChanged ()

  signalBoardChanged is used to inform the GameEngine about changes on the board in order to evoke a redraw.
- void signalUpdateInfo (QString)

  signalUpdateInfo informs the GameEngine (InfoList) about a pass with the information about the player who passed.

Public Member Functions

- Board (QObject *parent=0, int numberOfHumans=2)

  Board initialize the member variables and create a new board matrix by calling newBoard(numberOfHumans).
- Board (const Board &board)

  Board copy constructor is used to store the boards on the stack. The storeBoardOnStack() function, which uses the copy constructor, is called after every move and pass.
- State getState (int x, int y) const

  getState returns the current State of the board at position x and y.
- void setAllowed (QMap< QPair< int, int >, QVector< QPair< int, int > > > legalMoves)

  setAllowed updates the board matrix with the allowed moves for the current player.
- void newBoard (int numberOfHumans)

  newBoard initializes the board with the number of set human players (and computer). Updates the board matrix with the 2x2 diagonal beginning and sets the allowed moves for the black player using setAllowed().
- void storeBoardOnStack ()

  storeBoardOnStack uses the copy constructor to store the current board on the "global" stack which is used for alpha-beta pruning algorithm and undoing a move. This funciton is called after a move and a pass.
- void togglePlayer ()

  togglePlayer adjusts the pointers of the current and opponent player. Furthermore it sets the allowed moves and evokes a redraw of the board using signalBoardChanged().
- void makePass ()

  makePass used if there are no legal moves left for the current player This functions calls storeBoardOnStack() to register the pass.
- void countDisks (void)

  countDisks count number of black, white and total disks and store it in the corresponding member variables.
- int countPlayerDisks (void)

  countPlayerDisks counts the disk of the current player of the board. This function is used for the evaluation function inside the AI class.
- bool legalMove (int x, int y)

  legalMove check if the move at position (x,y) is legal.
- bool legalMovesAvailable ()

  legalMovesAvailable returns true if the current player has legal moves available.
- QMap< QPair< int, int >, QVector< QPair< int, int > > > getLegalMoves ()

  getLegalMoves stores the legal moves for the current player and the therefore happening disk flips inside a map. The keys are the available legal moves. Its corresponding values are QVectors with the disks that will be flipped (taking a key(=move)).
- void makeMove (int x, int y)

  makeMove execute a move, at position (x,y), for the current player that is legal and switches to the opponent user by calling togglePlayer().

- State whosTurn ()

  whosTurn returns the color of the current player stored in m_currentPlayer.
- Type whosTurnType ()

  whosTurnType returns the type of the current player stored in m_currentPlayer.
- bool onBoard (int x, int y)

  onBoard checks if a position or move is on the board.
- State getOtherPlayer (Player *player)

  getOtherPlayer used to get the color of the opponent

Public Attributes

- int **m_numberOfBlackDisks**
- int **m_numberOfWhiteDisks**
- int **m_numberOfDisks**
- bool **m_gameOver**
- bool **m_movesAvailable**
- QStack< Board * > * m_boardStack

  m_boardStack stores the boards according to the made moves. Mainly used for the AI to unwind the recursion.
- int **m_numberOfActualMoves**
- int **m_numberOfTotalMoves**

Static Public Attributes

- static const int m_direction [8][2] = {{1, 0}, {1, 1}, {0, 1}, {-1, 1}, {-1, 0}, {-1, -1}, {0, -1}, {1, -1}}

  m_direction is a member that is used to check all directions for legal moves.

3.2.1 Detailed Description

The Board class holds the board matrix and stores every board on a stack with the current and opponent player. It is used to find all allowed moves for the current player and actually execute them.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Board::Board (QObject * *parent* = 0, int *numberOfHumans* = 2)

Board initialize the member variables and create a new board matrix by calling newBoard(numberOfHumans).

Parameters

parent	class that created the Board.
numberOfHumans	is the set number of humans inside the GUI.

3.2.2.2 Board::Board (const Board & *board*)

Board copy constructor is used to store the boards on the stack. The storeBoardOnStack() function, which uses the copy constructor, is called after every move and pass.

Parameters

board	the current board to store on the stack.

3.2.3 Member Function Documentation

3.2.3.1 int Board::countPlayerDisks (void)

countPlayerDisks counts the disk of the current player of the board. This function is used for the evaluation function inside the AI class.

Returns

3.2.3.2 QMap< QPair< int, int >, QVector< QPair< int, int > > > Board::getLegalMoves ( )

getLegalMoves stores the legal moves for the current player and the therefore happening disk flips inside a map. The keys are the available legal moves. Its corresponding values are QVectors with the disks that will be flipped (taking a key(=move)).

Returns

3.2.3.3 State Board::getOtherPlayer (Player * *player*)

getOtherPlayer used to get the color of the opponent

Parameters

pointer	to a player (current or opponent).

Returns

the color of the other player depending on the passed player.

3.2.3.4 State Board::getState (int *x*, int *y*) const

getState returns the current State of the board at position x and y.

Parameters

x	column of the board matrix.
y	row of the board matrix.

Returns

state of the chosen position (x,y).

3.2.3.5 bool Board::legalMove (int *x*, int *y*)

legalMove check if the move at position (x,y) is legal.

Parameters

x	column of the board
y	row of the board

Returns

true if move at position (x,y) is legal.

3.2.3.6 bool Board::legalMovesAvailable ( )

legalMovesAvailable returns true if the current player has legal moves available.

Returns

3.2.3.7 void Board::makeMove (int *x*, int *y*)

makeMove execute a move, at position (x,y), for the current player that is legal and switches to the opponent user by calling togglePlayer().

Parameters

x	column of the move.
y	row of the move.

3.2.3.8 void Board::newBoard (int *numberOfHumans*)

newBoard initializes the board with the number of set human players (and computer). Updates the board matrix with the 2x2 diagonal beginning and sets the allowed moves for the black player using setAllowed().

Parameters

numberOfHumans	

3.2.3.9 bool Board::onBoard (int *x*, int *y*)

onBoard checks if a position or move is on the board.

Parameters

x	column of the position.
y	row of the position.

Returns

ture if position (x,y) is on the board, false if outisde.

3.2.3.10 void Board::setAllowed (QMap< QPair< int, int >, QVector< QPair< int, int > > > *legalMoves*)

setAllowed updates the board matrix with the allowed moves for the current player.

Parameters

legalMoves	the list with allowed moves for the current player. As parameter getLegalMoves() is usually used.

3.2.3.11 bool Board::undoMove ( ) [slot]

undoMove get the last board from the board stack.

Returns

true if undo was possible.

3.2.3.12 State Board::whosTurn ( )

whosTurn returns the color of the current player stored in m_currentPlayer.

Returns

Color of current player.

3.2.3.13 Type Board::whosTurnType ( )

whosTurnType returns the type of the current player stored in m_currentPlayer.

Returns

Type of current player.

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

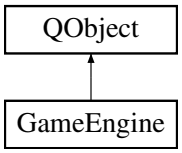
- board/board.h
- board/board.cpp

3.3 GameEngine Class Reference

The GameEngine class controls the course of the game by processing user input, executing moves from humans and the computer and informing the user about events using lists from the GUI. Furthermore it has an interface to the ui class UIGameScene to evoke redraws of the board.

`#include <gameengine.h>`

Inheritance diagram for GameEngine:



Public Slots

- void mouseReleased (QPointF point)

  mouseReleased converts the position where the user clicked inside the GraphicsScene (UIGameScene), to a row and column inside the board. The event is only forwarded to eventHandling() if it is a human move.
- void updateUISquare (int x, int y, State currentPlayer)

  updateUISquare DEPRECATED! this funciton was used to update single squres of the board on the gui side.
- void updateUIGameScene ()

  updateUIGameScene evokes a redraw of the whole board using the m_board member.
- void updateInfoTextPass (QString string)

  updateInfoTextPass informs the user about the current player and game result.

Public Member Functions

- GameEngine (QObject ∗parent, UIGameScene ∗uiGameScene, QTextEdit ∗eventList, QTextEdit ∗infoList)

  GameEngine initializes the passed pointers from the main window gui elements (textEdits). Furthermore the pointer to GameEngine gets assigned to the corresponding member variable.
- void startGame (int numberOfHumans=1, double timeLimit=10)

  startGame reads the user settings, like number of humans, search depth (m_timeLimit), connects the most of the signal and slot mechanism for redrawing the gui and updating the textEdits and creates the classes important classes (AI, Board).

Public Attributes

- Board ∗ m_board

  m_board points to the Board class that stores the board. This representation is used to update the gui with the current states of the signle squares of the board.

3.3.1 Detailed Description

The GameEngine class controls the course of the game by processing user input, executing moves from humans and the computer and informing the user about events using lists from the GUI. Furthermore it has an interface to the ui class UIGameScene to evoke redraws of the board.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 GameEngine::GameEngine (QObject * *parent*, UIGameScene * *uiGameScene*, QTextEdit * *eventList*, QTextEdit * *infoList*)

GameEngine initializes the passed pointers from the main window gui elements (textEdits). Furthermore the pointer to GameEngine gets assigned to the corresponding member variable.

Parameters

parent	creating class.
uiGameScene	pointer to UIGameScene class, used to update the graphical representaiton of the board.
eventList	pointer to the textEdit that informs the user about the moves.
infoList	pointer to the textEdit that informs the user about the current player and the game results.

3.3.3 Member Function Documentation

3.3.3.1 void GameEngine::mouseReleased (QPointF *point*) [slot]

mouseReleased converts the position where the user clicked inside the GraphicsScene (UIGameScene), to a row and column inside the board. The event is only forwarded to eventHandling() if it is a human move.

Parameters

point	received mouse position from the GraphicsScene (UIGameScene).

3.3.3.2 void GameEngine::startGame (int *numberOfHumans* = 1, double *timeLimit* = 10)

startGame reads the user settings, like number of humans, search depth (m_timeLimit), connects the most of the signal and slot mechanism for redrawing the gui and updating the textEdits and creates the classes important classes (AI, Board).

Parameters

numberOfHumans	number of human players. One: Human vs Computer, Two: Human vs Human.
timeLimit	is the search depth of the used alpha-beta pruning algorithm.

3.3.3.3 void GameEngine::updateInfoTextPass (QString *string*) [slot]

updateInfoTextPass informs the user about the current player and game result.

Parameters

string	passed informations which is shown to the user inside the textEdit.

3.3.3.4 void GameEngine::updateUISquare (int *x*, int *y*, State *currentPlayer*) [slot]

updateUISquare DEPRECATED! this funciton was used to update single squres of the board on the gui side.

Parameters

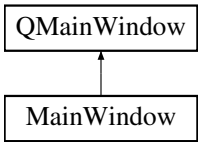
x	
y	
currentPlayer	

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

- gamelogic/gameengine.h
- gamelogic/gameengine.cpp

3.4 MainWindow Class Reference

Inheritance diagram for MainWindow:



Public Slots

- void startNewGame ()

  startNewGame creates a new UIGameScene class that gets passed to the newly created GameEngine class. Also the gui textEdits (ui->textEditEvents ui->textEditInfo) to display user informations are passed to GameEngine. Further more, the undo push button is registered with the GameEngine and the user mouse events are forwarded too. Finally the number of humans from the ui->comboBoxNumberOfHumans is passed to the GameEngine.

Public Member Functions

- MainWindow (QWidget *parent=0)

  MainWindow constrains the window elements by fixing their size. Improves rendering of the board UIGameScene inside the GraphicsView. Sets a validator to the user input for the search depth to allow only integer numbers (0 to 10). Sets the connects for the signal and slot mechanism for the newGame push button.

3.4.1 Constructor & Destructor Documentation

3.4.1.1 MainWindow::MainWindow (QWidget * *parent* = 0) [explicit]

MainWindow constrains the window elements by fixing their size. Improves rendering of the board UIGameScene inside the GraphicsView. Sets a validator to the user input for the search depth to allow only integer numbers (0 to 10). Sets the connects for the signal and slot mechanism for the newGame push button.

Parameters

parent	

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

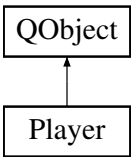
- main/mainwindow.h
- main/mainwindow.cpp

3.5 Player Class Reference

The Player class defines the current and the opponent player. Each player has its own color, BLACK or WHITE, and is of type HUMAN or COMPUTER.

`#include <player.h>`

Inheritance diagram for Player:



Public Member Functions

- Player (QObject *parent=0, State color=NONE, Type type=UNKNOWN)

  Player initializes the instance with its color and type.

Public Attributes

- Type m_type

  m_type stores the player type. Either HUMAN or COMPUTER.
- State m_color

  m_color stores the color of the player. Either Black or White.

3.5.1 Detailed Description

The Player class defines the current and the opponent player. Each player has its own color, BLACK or WHITE, and is of type HUMAN or COMPUTER.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 Player::Player (QObject * *parent* = 0, State *color* = NONE, Type *type* = UNKNOWN) [explicit]

Player initializes the instance with its color and type.

Parameters

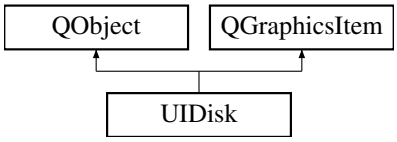
parent	
color	
type	

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

- player/player.h
- player/player.cpp

3.6 UIDisk Class Reference

Inheritance diagram for UIDisk:



Public Types

- enum **State** {
  BOARD, **BLACK**, **WHITE**, **ALLOWED**,
  SUGGESTED }

Public Member Functions

- **UIDisk** (const double height, const double width)
- **UIDisk** (const UIDisk &disk)
- QRectF **boundingRect** () const
- void **paint** (QPainter ∗painter, const QStyleOptionGraphicsItem ∗option, QWidget ∗widget)
- void **setState** (const State state)
- State **getState** () const
- void **setPosition** (const double positionX, const double positionY)
- void **setSize** (const double height, const double width)

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

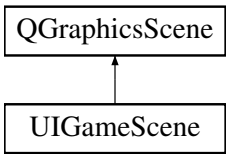
- ui/uidisk.h
- ui/uidisk.cpp

3.7 UIGameScene Class Reference

The UIGameScene class stores the graphical representation of the board using UISquare class. The GameEngine uses this class to redraw the board matrix from the Board class. This class also forwards mouse release events to the GameEngine made by the user.

`#include <uigamescene.h>`

Inheritance diagram for UIGameScene:



Public Slots

- virtual void mouseReleaseEvent (QGraphicsSceneMouseEvent ∗mouseEvent)

  mouseReleaseEvent this event happens when the user releases the mouse inside this UIGameScene. These events are used to forward the current mouse position to the GameEngine using the signal newMouseEvent(). The connect of the Signal and Slot is done inside the MainWindow class.

Signals

- void newMouseEvent (QPointF position)

  newMouseEvent is used to inform the GameEngine where the user pressed inside this UIGameScene. The cursor position is processed inside slot mouseReleased(QPoint position) of the GameEngine class.

Public Member Functions

- UIGameScene (QObject ∗parent)

  UIGameScene.
- void setSquareState (int x, int y, State state)

  setSquareState
- void redrawBoard (Board ∗board)

  redrawBoard redraws every single square with the current state of the board matrix from the passed Board class instance.

Public Attributes

- double m_sizeSceneRect

  m_sizeSceneRect size (width and height) of the whole board.
- double m_squareSize

  m_squareSize size (width and height) of a single square.

3.7.1 Detailed Description

The UIGameScene class stores the graphical representation of the board using UISquare class. The GameEngine uses this class to redraw the board matrix from the Board class. This class also forwards mouse release events to the GameEngine made by the user.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 UIGameScene::UIGameScene (QObject * *parent*)

UIGameScene.

Parameters

parent	

3.7.3 Member Function Documentation

3.7.3.1 void UIGameScene::mouseReleaseEvent (QGraphicsSceneMouseEvent * *mouseEvent*) [virtual], [slot]

mouseReleaseEvent this event happens when the user releases the mouse inside this UIGameScene. These events are used to forward the current mouse position to the GameEngine using the signal newMouseEvent(). The connect of the Signal and Slot is done inside the MainWindow class.

Parameters

mouseEvent	defines which mouse button was released.

3.7.3.2 void UIGameScene::newMouseEvent (QPointF *position*) [signal]

newMouseEvent is used to inform the GameEngine where the user pressed inside this UIGameScene. The cursor position is processed inside slot mouseReleased(QPoint position) of the GameEngine class.

Parameters

position	

3.7.3.3 void UIGameScene::redrawBoard (Board * *board*)

redrawBoard redraws every single square with the current state of the board matrix from the passed Board class instance.

Parameters

board	Board class that holds the current board matrix with its states.

3.7.3.4 void UIGameScene::setSquareState (int *x*, int *y*, State *state*)

setSquareState

Parameters

x	column of the board.
y	row of the board.
state	new state of the square at position (x,y).

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

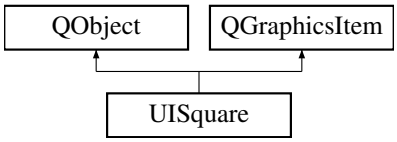
- ui/uigamescene.h
- ui/uigamescene.cpp

3.8 UISquare Class Reference

The UISquare class represents a single square inside the QGraphicsScene (UIGameScene). It is used to show the different states of a square (BOARD, BLACK, WHITE, ALLOWED) to the user. The board matrix 2x2 QVector of the UIGameScene class consists of these UISquares.

```
#include <uisquare.h>
```

Inheritance diagram for UISquare:



Public Member Functions

- UISquare (int x, int y, State state=NONE)

  UISquare main constructor. Position on the board is used for debugging.
- **UISquare** (const double height, const double width)
- **UISquare** (const UISquare &square)
- QRectF **boundingRect** () const
- void **paint** (QPainter ∗painter, const QStyleOptionGraphicsItem ∗option, QWidget ∗widget)
- void setUISquareState (const State state)

  setState update the state of the square according to passed enum State {NONE, BOARD, BLACK, WHITE, ALLO↩ WED, SUGGESTED}
- State **getUISquareState** () const
- void **setPosition** (const double boardPositionX, const double boardPositionY)
- void **setSize** (const double size)

3.8.1 Detailed Description

The UISquare class represents a single square inside the QGraphicsScene (UIGameScene). It is used to show the different states of a square (BOARD, BLACK, WHITE, ALLOWED) to the user. The board matrix 2x2 QVector of the UIGameScene class consists of these UISquares.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 UISquare::UISquare (int *x*, int *y*, State *state* = NONE)

UISquare main constructor. Position on the board is used for debugging.

Parameters

x	
y	

3.8.3 Member Function Documentation

3.8.3.1 void UISquare::setUISquareState (const State *state*)

setState update the state of the square according to passed enum State {NONE, BOARD, BLACK, WHITE, ALL↩ OWED, SUGGESTED}

Parameters

state	

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

- ui/uisquare.h
- ui/uisquare.cpp

Index