

## Algorithms project- Task 6

Generated by Doxygen 1.9.8



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 ChessBoard Class Reference	5
3.1.1 Detailed Description	5
3.1.2 Constructor & Destructor Documentation	5
3.1.2.1 ChessBoard()	5
3.1.3 Member Function Documentation	6
3.1.3.1 getSquare() [1/2]	6
3.1.3.2 getSquare() [2/2]	6
3.2 Cost Class Reference	6
3.2.1 Detailed Description	7
3.3 Graph Class Reference	7
3.3.1 Detailed Description	7
3.3.2 Constructor & Destructor Documentation	7
3.3.2.1 Graph()	7
3.3.3 Member Function Documentation	9
3.3.3.1 solvePuzzle()	9
3.4 Move Class Reference	9
3.4.1 Detailed Description	10
3.4.2 Constructor & Destructor Documentation	10
3.4.2.1 Move()	10
3.4.3 Member Function Documentation	10
3.4.3.1 getknight()	10
3.4.3.2 getNewPos()	11
3.4.3.3 getOldPos()	11
3.5 Square Class Reference	11
3.5.1 Detailed Description	11
3.5.2 Member Function Documentation	12
3.5.2.1 getKnight()	12
3.5.2.2 getPosNo()	12
3.5.2.3 getPrevValue()	12
3.5.2.4 placeKnight()	12
3.5.2.5 removeKnight()	13
<b>4 File Documentation</b>	<b>15</b>
4.1 ChessBoard.cpp File Reference	15
4.1.1 Detailed Description	15
4.2 ChessBoard.h File Reference	15

4.2.1 Detailed Description . . . . .	16
4.3 ChessBoard.h . . . . .	16
4.4 Graph.cpp File Reference . . . . .	16
4.4.1 Detailed Description . . . . .	17
4.5 Graph.h File Reference . . . . .	17
4.5.1 Detailed Description . . . . .	17
4.6 Graph.h . . . . .	18
4.7 Move.h File Reference . . . . .	19
4.7.1 Detailed Description . . . . .	19
4.8 Move.h . . . . .	19
4.9 Square.cpp File Reference . . . . .	20
4.9.1 Detailed Description . . . . .	20
4.10 Square.h File Reference . . . . .	20
4.10.1 Detailed Description . . . . .	20
4.11 Square.h . . . . .	21
<b>Index</b>	<b>23</b>

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">ChessBoard</a>	Represents a chessboard . . . . .	<a href="#">5</a>
<a href="#">Cost</a>	Represents the cost of moving a knight from one square to another . . . . .	<a href="#">6</a>
<a href="#">Graph</a>	Represents a graph for solving the knight's puzzle on a chessboard . . . . .	<a href="#">7</a>
<a href="#">Move</a>	Represents a move made on a chessboard . . . . .	<a href="#">9</a>
<a href="#">Square</a>	Represents a square on the chessboard . . . . .	<a href="#">11</a>



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">ChessBoard.cpp</a>	Implementation of the <a href="#">ChessBoard</a> class methods . . . . .	15
<a href="#">ChessBoard.h</a>	Declaration of the <a href="#">ChessBoard</a> class . . . . .	15
<a href="#">Graph.cpp</a>	Implementation of the <a href="#">Graph</a> class methods . . . . .	16
<a href="#">Graph.h</a>	Declaration of the <a href="#">Graph</a> class and the <a href="#">Cost</a> class . . . . .	17
<a href="#">Move.h</a>	Declaration of the <a href="#">Move</a> class . . . . .	19
<a href="#">Square.cpp</a>	Implementation of the <a href="#">Square</a> class . . . . .	20
<a href="#">Square.h</a>	Declaration of the <a href="#">Square</a> class . . . . .	20





# Chapter 3

## Class Documentation

### 3.1 ChessBoard Class Reference

Represents a chessboard.

```
#include <ChessBoard.h>
```

#### Public Member Functions

- [ChessBoard](#) ()  
*Constructs a chessboard with squares and knights.*
- void [printBoard](#) ()  
*Prints the current state of the chessboard.*
- [Square](#) \* [getSquare](#) (int row, int col)  
*Gets the square at the specified row and column.*
- [Square](#) \* [getSquare](#) (int posNo)  
*Gets the square at the specified position number.*

#### 3.1.1 Detailed Description

Represents a chessboard.

This class provides functionalities for creating and manipulating a chessboard, as well as printing its current state.

#### 3.1.2 Constructor & Destructor Documentation

##### 3.1.2.1 ChessBoard()

```
ChessBoard::ChessBoard ( )
```

Constructs a chessboard with squares and knights.

Initializes a 4x3 grid representing the chessboard and populates it with squares. Initializes the knights vector with black and white knight values. Places white knights (W1, W2, W3) on the first row and black knights (B1, B2, B3) on the last row of the chessboard.

The layout of the chessboard is as follows: number(knight name), Em == empty 0 (W1) 1 (W2) 2 (W3) 3 (Em) 4 (Em) 5 (Em) 6 (Em) 7 (Em) 8 (Em) 9 (B1) 10 (B2) 11 (B3)

Note: each square represented by: number (piece name), E.g., W1 is placed on position 0 at beginning. Note: 'Em' represents an empty square.

### 3.1.3 Member Function Documentation

#### 3.1.3.1 `getSquare()` [1/2]

```
Square * ChessBoard::getSquare (
    int posNo )
```

Gets the square at the specified position number.

##### Parameters

<i>posNo</i>	The position number of the square.
--------------	------------------------------------

##### Returns

Pointer to the square at the specified position.

#### 3.1.3.2 `getSquare()` [2/2]

```
Square * ChessBoard::getSquare (
    int row,
    int col )
```

Gets the square at the specified row and column.

##### Parameters

<i>row</i>	The row index of the square.
<i>col</i>	The column index of the square.

##### Returns

Pointer to the square at the specified position.

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

- [ChessBoard.h](#)
- [ChessBoard.cpp](#)

## 3.2 Cost Class Reference

Represents the cost of moving a knight from one square to another.

```
#include <Graph.h>
```

##### Friends

- class **Graph**

### 3.2.1 Detailed Description

Represents the cost of moving a knight from one square to another.

The cost used to control the movement of knights to the correct squares.

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

- [Graph.h](#)

## 3.3 Graph Class Reference

Represents a graph for solving the knight's puzzle on a chessboard.

```
#include <Graph.h>
```

### Public Member Functions

- [Graph](#) ([ChessBoard](#) &board)  
*Constructs a [Graph](#) object with the given [ChessBoard](#).*
- void **printGraph** ()  
*Prints the graph (for debugging purposes).*
- vector< [Move](#) \* > \* **solvePuzzle** ()  
*Solves the knight's puzzle and returns a vector of [Move](#) objects representing the solution.*

### 3.3.1 Detailed Description

Represents a graph for solving the knight's puzzle on a chessboard.

The [Graph](#) class encapsulates the logic for solving the knight's tour puzzle using graph traversal algorithms.

### 3.3.2 Constructor & Destructor Documentation


#### 3.3.2.1 Graph()

```
Graph::Graph (  
    ChessBoard & board )
```

Constructs a [Graph](#) object with the given [ChessBoard](#).

The constructor initializes a [Graph](#) object with the provided [ChessBoard](#) by creating nodes and establishing connections between them based on the specified rules. Each square is connected to other squares based on specific knight moves allowed in chess.

TODO: add the graph with cost image.



image\_filename.png

**Figure 3.1** Caption text

## Parameters

<code>board</code>	The <a href="#">ChessBoard</a> object representing the chessboard.
--------------------	--

### 3.3.3 Member Function Documentation

#### 3.3.3.1 solvePuzzle()

```
vector< Move * > * Graph::solvePuzzle ( )
```

Solves the knight's puzzle and returns a vector of [Move](#) objects representing the solution.

This function implements a breadth-first search algorithm to solve the knight's puzzle.

- It starts from the given starting positions (9 and 11) and explores all possible moves using a modified queue-based approach.
- The algorithm iteratively explores all reachable squares considering the cost of each move and ensuring that the knight does not visit any square twice.
- If a square is empty, its neighboring squares are added to the queue for exploration.
- If the knight occupies a square, the algorithm attempts to move it to an empty neighboring square based on the cost associated with each move.
- After each move, the algorithm checks if the knight has reached its final destination, indicated by a square with only one edge.
- The algorithm continues until it completes all necessary swaps to solve the puzzle, i.e., until the knight has visited all squares exactly once.
- It maintains a record of all moves made during the puzzle solution, which are returned as a vector of [Move](#) objects.

The returned vector of [Move](#) objects represents the solution and is used by the GUI to display the sequence of knight movements on the chessboard.

## Returns

`vector<Move*>*` A pointer to a vector of [Move](#) objects representing the solution.

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

- [Graph.h](#)
- [Graph.cpp](#)

## 3.4 Move Class Reference

Represents a move made on a chessboard.

```
#include <Move.h>
```

## Public Member Functions

- `Move` (int oldPos, int newPos, string knight)  
*Constructor for the `Move` class.*
- string `getknight` ()  
*Retrieves the name of the knight.*
- int `getOldPos` ()  
*Retrieves the old position of the piece.*
- int `getNewPos` ()  
*Retrieves the new position of the piece.*
- void `printMove` ()  
*Prints the details of the move on console.*

### 3.4.1 Detailed Description

Represents a move made on a chessboard.

The `Move` class provides functionality to record and represent individual moves made by the knight, including the old and new positions of the knight.

Moves will be used by the GUI to display moves at each step of solving the puzzle.

### 3.4.2 Constructor & Destructor Documentation

#### 3.4.2.1 Move()

```
Move::Move (
    int oldPos,
    int newPos,
    string knight ) [inline]
```

Constructor for the `Move` class.

#### Parameters

<i>oldPos</i>	The old position of the piece.
<i>newPos</i>	The new position of the piece.
<i>knight</i>	The name of the knight making the move.

### 3.4.3 Member Function Documentation

#### 3.4.3.1 getknight()

```
string Move::getknight ( ) [inline]
```

Retrieves the name of the knight.

#### Returns

The name of the knight.

### 3.4.3.2 getNewPos()

```
int Move::getNewPos ( ) [inline]
```

Retrieves the new position of the piece.

#### Returns

The new position of the piece.

### 3.4.3.3 getOldPos()

```
int Move::getOldPos ( ) [inline]
```

Retrieves the old position of the piece.

#### Returns

The old position of the piece.

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

- [Move.h](#)

## 3.5 Square Class Reference

Represents a square on the chessboard.

```
#include <Square.h>
```

### Public Member Functions

- **Square ()**  
*Constructor for the [Square](#) class. Initialize the position number of the square as its declaration order.*
- bool [placeKnight](#) (const string \*knight)  
*Places a knight on the square.*
- const string \* [getKnight](#) () const  
*Retrieves the knight on the square.*
- const string \* [removeKnight](#) ()  
*Removes the knight from the square.*
- const string \* [getPrevValue](#) () const  
*Removes the knight from the square.*
- int [getPosNo](#) () const  
*Retrieves the position number of the square.*

### 3.5.1 Detailed Description

Represents a square on the chessboard.

## 3.5.2 Member Function Documentation

### 3.5.2.1 getKnight()

```
const string * Square::getKnight ( ) const
```

Retrieves the knight on the square.

#### Returns

Pointer to the knight on the square, or nullptr if no knight is present.

### 3.5.2.2 getPosNo()

```
int Square::getPosNo ( ) const
```

Retrieves the position number of the square.

#### Returns

The position number of the square.

### 3.5.2.3 getPrevValue()

```
const string * Square::getPrevValue ( ) const
```

Removes the knight from the square.

#### Returns

Pointer to the removed knight.

If a knight is present on the square, it is removed, and its value is stored as the previous value.

### 3.5.2.4 placeKnight()

```
bool Square::placeKnight (
    const string * knight )
```

Places a knight on the square.

#### Parameters

<i>knight</i>	Pointer to the knight to be placed.
---------------	-------------------------------------



**Returns**

True if the knight was successfully placed, false otherwise.

Returns false if the square already contains a knight. Check if the square already contains a knight no more than one piece in the square. no piece takes other piece.

**3.5.2.5 removeKnight()**

```
const string * Square::removeKnight ( )
```

Removes the knight from the square.

**Returns**

Pointer to the removed knight.

If a knight is present on the square, it is removed, and its value is stored as the previous value.

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

- [Square.h](#)
- [Square.cpp](#)



# Chapter 4

## File Documentation

### 4.1 ChessBoard.cpp File Reference

Implementation of the [ChessBoard](#) class methods.

```
#include "ChessBoard.h"
```

#### 4.1.1 Detailed Description

Implementation of the [ChessBoard](#) class methods.

Author

eslam

Date

March 2024

### 4.2 ChessBoard.h File Reference

Declaration of the [ChessBoard](#) class.

```
#include "Square.h"  
#include <vector>  
#include <iostream>
```

#### Classes

- class [ChessBoard](#)  
*Represents a chessboard.*

### 4.2.1 Detailed Description

Declaration of the [ChessBoard](#) class.

The [ChessBoard](#) class provides functionality to manage the chessboard state, including placing and moving knights on the board.

#### Author

eslam

#### Date

March 2024

## 4.3 ChessBoard.h

[Go to the documentation of this file.](#)

```
00001 /*****
00011 #pragma once
00012
00013 #ifndef CHESSBOARD_H
00014 #define CHESSBOARD_H
00015
00016 #include "Square.h"
00017 #include <vector>
00018 #include <iostream>
00019 using namespace std;
00020
00027 class ChessBoard
00028 {
00029 public:
00046     ChessBoard();
00047
00051     void printBoard();
00052
00059     Square* getSquare(int row, int col);
00060
00066     Square* getSquare(int posNo);
00067
00068 private:
00069     // 2D matrix to store the chessboard squares.
00070     vector<vector<Square>> squares;
00071
00072     // Values of knights.
00073     vector<string> knights;
00074
00075 }; // class ChessBoard
00076
00077 #endif // !CHESSBOARD_H
```

## 4.4 Graph.cpp File Reference

Implementation of the [Graph](#) class methods.

```
#include "Graph.h"
```

### 4.4.1 Detailed Description

Implementation of the [Graph](#) class methods.

Author

eslam

Date

March 2024

## 4.5 Graph.h File Reference

Declaration of the [Graph](#) class and the [Cost](#) class.

```
#include "ChessBoard.h"
#include "Move.h"
#include <utility>
#include <deque>
#include <iostream>
```

### Classes

- class [Cost](#)  
*Represents the cost of moving a knight from one square to another.*
- class [Graph](#)  
*Represents a graph for solving the knight's puzzle on a chessboard.*

### 4.5.1 Detailed Description

Declaration of the [Graph](#) class and the [Cost](#) class.

The [Graph](#) class encapsulates the logic for solving the knight's tour puzzle using graph traversal algorithms.

Author

eslam

Date

March 2024

## 4.6 Graph.h

[Go to the documentation of this file.](#)

```

00001  /*****
00011  #pragma once
00012  #ifndef GRAPH_H
00013  #define GRAPH_H
00014
00015  #include "ChessBoard.h"
00016  #include "Move.h"
00017  #include <utility>
00018  #include <deque>
00019  #include <iostream>
00020
00021
00022  using namespace std;
00023
00029  class Cost {
00030      friend class Graph;
00031  private:
00038      Cost(int whiteCost, int blackCost) : whiteCost(whiteCost), blackCost(blackCost) {}
00039
00046      int getCost(int color) {
00047          return color ? blackCost: whiteCost;
00048      }
00049
00050      // The cost of moving a white knight.
00051      int whiteCost;
00052
00053      // The cost of moving a black knight.
00054      int blackCost;
00055  }; // class Cost
00056
00057
00064  class Graph
00065  {
00066  public:
00080      Graph(ChessBoard& board);
00081
00085      void printGraph();
00086
00116      vector<Move*> solvePuzzle();
00117  private:
00118      // Represents the nodes (squares) of the graph, each containing a cost and a pointer to a Square
00119      // object.
00119      vector<
00120          vector
00121          <pair<Cost, Square*>
00122          >
00123      > nodes;
00124
00125      // ChessBoard to solve.
00126      ChessBoard* board;
00127
00128      // Helper methods
00129
00141      void addNeighborSquaresToQueue(vector<pair<Cost, Square*>*> neighborSquares, deque<int>&
00142      positionsToProcess);
00149      int getKnightColor(Square* square);
00150
00172      bool tryMovingKnight(Square* currentSquare, vector<pair<Cost, Square*>*> neighborSquares, int
00173      knightColor,
00173      int& completedSwaps, int& totalMoves, vector<Move*>*> allMoves, deque<int>&
00174      positionsToProcess);
00174
00189      bool canMoveToSquare(pair<Cost, Square*>*> neighborPair, Square* currentSquare, int knightColor);
00190
00201      void performMove(pair<Cost, Square*>*> neighborPair, Square* currentSquare);
00202
00218      void checkForFinalDestination(pair<Cost, Square*>*> neighborPair, bool& isFinalDestination, int&
00219      completedSwaps, int knightColor);
00219
00230      void storeMoveData(vector<Move*>*> allMoves, Square* currentSquare, pair<Cost, Square*>*>
00231      neighborPair);
00231
00243      void addNeighborsToQueue(vector<pair<Cost, Square*>*> neighborSquares, deque<int>&
00244      positionsToProcess, int destinationIndex);
00244
00253      void printMoveDetails(int& totalMoves, Square* currentSquare, vector<Move*>*> allMoves);
00254
00260      void printTotalMoves(int totalMoves);
00261  }; // class Graph
00262
00263  #endif // !GRAPH_H

```

## 4.7 Move.h File Reference

Declaration of the [Move](#) class.

```
#include <string>
#include <iostream>
```

### Classes

- class [Move](#)  
*Represents a move made on a chessboard.*

### 4.7.1 Detailed Description

Declaration of the [Move](#) class.

Author

eslam

Date

March 2024

## 4.8 Move.h

[Go to the documentation of this file.](#)

```
00001 /*****
00008 #pragma once
00009
00010 #ifndef MOVE_H
00011 #define MOVE_H
00012
00013 #include <string>
00014 #include <iostream>
00015 using namespace std;
00016
00025 class Move
00026 {
00027 public:
00034     Move(int oldPos, int newPos, string knight):
00035         oldPos(oldPos), newPos(newPos), knight(knight) {}
00036
00041     string getknight() {
00042         return this->knight;
00043     }
00044
00049     int getOldPos() {
00050         return this->oldPos;
00051     }
00052
00057     int getNewPos() {
00058         return this->newPos;
00059     }
00060
00064     void printMove() {
00065         cout << "Move: " << knight << " from " << oldPos << " to " << newPos << endl;
00066     }
00067 private:
00068     // The old position of the piece.
00069     int oldPos;
00070     // The new position of the piece.
00071     int newPos;
00072     // The name of the knight making the move.
00073     string knight;
00074 }; // class Move
00075
00076
00077
00078 #endif // !MOVE_H
00079
00080
```

## 4.9 Square.cpp File Reference

Implementation of the [Square](#) class.

```
#include "Square.h"
```

### 4.9.1 Detailed Description

Implementation of the [Square](#) class.

Author

eslam

Date

March 2024

## 4.10 Square.h File Reference

Declaration of the [Square](#) class.

```
#include <string>
```

### Classes

- class [Square](#)  
*Represents a square on the chessboard.*

### 4.10.1 Detailed Description

Declaration of the [Square](#) class.

Author

eslam

Date

March 2024



## 4.11 Square.h

[Go to the documentation of this file.](#)

```
00001 /*****  
00009 #pragma once  
00010 #ifndef SQUARE_H  
00011 #define SQUARE_H  
00012  
00013 #include <string>  
00014  
00015 using namespace std;  
00016  
00020 class Square  
00021 {  
00022 public:  
00027     Square() : posNo(counter++), knight(nullptr), prevValue(nullptr) {}  
00028  
00035     bool placeKnight(const string* knight);  
00036  
00041     const string* getKnight() const;  
00042  
00048     const string* removeKnight();  
00049  
00055     const string* getPrevValue() const;  
00056  
00061     int getPosNo() const;  
00062  
00063 private:  
00064     //Static counter for numbering the squares.  
00065     static int counter;  
00066     //The position number of the square.  
00067     int posNo;  
00078     const string* knight;  
00079  
00080     // The name of the last piece entered the square, used for certain conditions.  
00081     const string* prevValue;  
00082  
00083 }; // class Square  
00084 #endif // !SQUARE_H
```



# Index

- ChessBoard, [5](#)
  - ChessBoard, [5](#)
  - getSquare, [6](#)
- ChessBoard.cpp, [15](#)
- ChessBoard.h, [15](#), [16](#)
- Cost, [6](#)
- getKnight
  - Square, [12](#)
- getknight
  - Move, [10](#)
- getNewPos
  - Move, [10](#)
- getOldPos
  - Move, [11](#)
- getPosNo
  - Square, [12](#)
- getPrevValue
  - Square, [12](#)
- getSquare
  - ChessBoard, [6](#)
- Graph, [7](#)
  - Graph, [7](#)
  - solvePuzzle, [9](#)
- Graph.cpp, [16](#)
- Graph.h, [17](#), [18](#)
- Move, [9](#)
  - getknight, [10](#)
  - getNewPos, [10](#)
  - getOldPos, [11](#)
  - Move, [10](#)
- Move.h, [19](#)
- placeKnight
  - Square, [12](#)
- removeKnight
  - Square, [13](#)
- solvePuzzle
  - Graph, [9](#)
- Square, [11](#)
  - getKnight, [12](#)
  - getPosNo, [12](#)
  - getPrevValue, [12](#)
  - placeKnight, [12](#)
  - removeKnight, [13](#)
- Square.cpp, [20](#)
- Square.h, [20](#), [21](#)