Sudoku Solver

1.00

Generated by Doxygen 1.10.0

Chapter 1

sudoku-solver

Sudoku game and algorithmic solver made in C.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

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

Chapter 4

Data Structure Documentation

4.1 Puzzle Struct Reference

Structure to store data of a single Sudoku puzzle.

`#include <puzzle.h>`

Data Fields

- int **grid** [**GRID_SIZE**][**GRID_SIZE**]
  The unmodified version of the puzzle.
- int **userGrid** [**GRID_SIZE**][**GRID_SIZE**]
  The user modified version of the grid.
- int **map** [**GRID_SIZE**][**GRID_SIZE**]
  Bitmap used to determine which squares are modifiable by user.

4.1.1 Detailed Description

Structure to store data of a single Sudoku puzzle.

The grid is always fixed, except for when generating a new sudoku

4.1.2 Field Documentation

4.1.2.1 map

`int map[ GRID_SIZE][ GRID_SIZE]`

Bitmap used to determine which squares are modifiable by user.

Note

If 1 == unmodifiable

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

- /home/zakajus/Documents/Code/sudoku-solver/ **puzzle.h**

4.2 translation Struct Reference

Key-value (dictionary) pair to store display strings.

`#include <ui.h>`

Data Fields

- char * **key**

  *Key to be converted by **translate()** (p. 34)*
- char **value** [**LINE_MAX**]

  Value to be output to display.

4.2.1 Detailed Description

Key-value (dictionary) pair to store display strings.

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

- /home/zakajus/Documents/Code/sudoku-solver/ **ui.h**

Chapter 5

File Documentation

5.1 /home/zakajus/Documents/Code/sudoku-solver/dependencies.h File Reference

Includes standard library dependencies, defines macros.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>
```

Macros

- #define **BIN_SAVE_FILENAME** "save.bin"

  Binary save file name.
- #define **LOG_FILENAME** "log.txt"

  Log text file name.
- #define **GRID_SIZE** 9

  Grid size of Sudoku puzzle (9x9 is standard)
- #define **SUBGRID_SIZE** 3

  Subgrids the Sudoku grid is divided into.
- #define **LINE_MAX** 80

  Maximum length of displeyed strings.
- #define **BUFFER_SIZE** 128

  Buffer size for user input.
- #define **ANSI_COLOR_MAGENTA** "\x1b[35m"

  ASCII break code to make text magenta.
- #define **ANSI_COLOR_GREEN** "\x1b[32m"

  ASCII break code to make text green.
- #define **ANSI_COLOR_RED** "\x1b[31m"

  ASCII break code to make text red.
- #define **ANSI_COLOR_RESET** "\x1b[0m"

  ASCII break code to reset text color.

5.1.1 Detailed Description

Includes standard library dependencies, defines macros.

Author

Kajus Zakaras (kajus.z@tuta.io)

Version

1.00

Date

2024-01-25

Copyright

5.2 /home/zakajus/Documents/Code/sudoku-solver/dependencies.h

Go to the documentation of this file.

```
00001
00011 #ifndef DEPENDENCIES_H
00012 #define DEPENDENCIES_H
00013
00014 #include <stdio.h>
00015 #include <stdlib.h>
00016 #include <string.h>
00017 #include <stdbool.h>
00018 #include <time.h>
00019
00020
00022 #define BIN_SAVE_FILENAME "save.bin"
00024 #define LOG_FILENAME "log.txt"
00025
00026
00028 #define GRID_SIZE 9
00030 #define SUBGRID_SIZE 3
00032 #define LINE_MAX 80
00034 #define BUFFER_SIZE 128
00035
00036
00038 #define ANSI_COLOR_MAGENTA "\x1b[35m"
00040 #define ANSI_COLOR_GREEN   "\x1b[32m"
00042 #define ANSI_COLOR_RED     "\x1b[31m"
00044 #define ANSI_COLOR_RESET   "\x1b[0m"
00045
00046 #endif
```

5.3 /home/zakajus/Documents/Code/sudoku-solver/files.c File Reference

Runtime and launch logging, saving / loading binary save data.

```
#include "./dependencies.h"
#include "./puzzle.h"
#include "./ui.h"
```

Functions

- long double **findCurrentRuntime** ()

  Finds the CPU runtime of the current launch.
- void **logRuntime** ()

  Calculates and appends CPU runtime during this launch to log file.
- void **logLaunch** ()

  Appends launch date and time to log file.
- long double **readTotalRuntime** ()

  Calculates total CPU runtime from log and the current launch.
- int **readLaunchCount** ()

  Reads total launch count from log file.
- void **saveDataToFile** (**Puzzle** ∗puzzleArray, int puzzleCount)

  Saves puzzle array and size to .bin file.
- void **loadDataFromFile** (**PuzzleArray** ∗puzzleArray, int ∗puzzleCount)

  Loads puzzle array and size from .bin file, allocates memory.
- void **initDataIfNoBinary** (**Puzzle** ∗puzzleArray, int puzzleCount)

  Initializes dynamic puzzle array if there is no .bin save file.

Variables

- clock_t **startTime**

  Global variable to track CPU runtime.

5.3.1 Detailed Description

Runtime and launch logging, saving / loading binary save data.

Header file for **files.h** (p. 13).

Author

Kajus Zakaras (kajus.z@tuta.io)

Version

1.00

Date

2024-01-25

Copyright

5.3.2 Function Documentation

5.3.2.1 findCurrentRuntime()

```
long double findCurrentRuntime ( )
```

Finds the CPU runtime of the current launch.

Returns

Runtime in seconds

5.3.2.2 initDataIfNoBinary()

```
void initDataIfNoBinary (
            Puzzle * puzzleArray,
           int puzzleCount )
```

Initializes dynamic puzzle array if there is no .bin save file.

Parameters

puzzleArray	Array of default puzzles to initialize
puzzleCount	Array size

Change binary file name using **BIN_SAVE_FILENAME** (p. 9) macro.

5.3.2.3 loadDataFromFile()

```
void loadDataFromFile (
            PuzzleArray * puzzleArray,
           int * puzzleCount )
```

Loads puzzle array and size from .bin file, allocates memory.

Parameters

puzzleArray	Array to load to
puzzleCount	Where to save array size

Change binary file name using **BIN_SAVE_FILENAME** (p. 9) macro. Loads array size before array.

5.3.2.4 readLaunchCount()

```
int readLaunchCount ( )
```

Reads total launch count from log file.

Returns

Count of total launches by program

5.3.2.5 readTotalRuntime()

```
long double readTotalRuntime ( )
```

Calculates total CPU runtime from log and the current launch.

Returns

Total CPU runtime in seconds

5.3.2.6 saveDataToFile()

```
void saveDataToFile (
              Puzzle * puzzleArray,
             int puzzleCount )
```

Saves puzzle array and size to .bin file.

Parameters

puzzleArray	Array to save
puzzleCount	Array size to save

Change binary file name using **BIN_SAVE_FILENAME** (p. 9) macro

Note

Saves array size before array!

5.4 /home/zakajus/Documents/Code/sudoku-solver/files.h

```
00001
00012 #ifndef FILES_H
00013 #define FILES_H
00014
00015
00016 #include "puzzle.h"
00017 #include "./dependencies.h"
00018
00019
00020 extern clock_t startTime;
00021
00022
00023 long double findCurrentRuntime();
00024 void logRuntime();
00025 void logLaunch();
00026 long double readTotalRuntime();
00027 int readLaunchCount();
00028
00029
00030 void saveDataToFile(Puzzle *puzzleArray, int puzzleCount);
00031 void loadDataFromFile(PuzzleArray *puzzleArray, int *puzzleCount);
00032 void initDataIfNoBinary(Puzzle *puzzleArray, int puzzleCount);
00033
00034
00035 #endif
```

5.5 /home/zakajus/Documents/Code/sudoku-solver/main.c File Reference

Handles launching the program, defining default puzzles.

```
#include "./dependencies.h"
#include "./puzzle.h"
#include "./files.h"
#include "./ui.h"
```

Functions

- int **main** ()

5.5.1 Detailed Description

Handles launching the program, defining default puzzles.

Author

Kajus Zakaras (kajus.z@tuta.io)

Version

1.00

Date

2024-01-25

Copyright

5.6 /home/zakajus/Documents/Code/sudoku-solver/puzzle.c File Reference

Sudoku puzzle logic, dynamic array management.

```
#include "./dependencies.h"
#include "./puzzle.h"
#include "./ui.h"
```

Functions

- void **generateBitmap** (**Puzzle** ∗puzzle)
  Generates bitmap in puzzle from its grid array values.
- void **generateUserGrid** (**Puzzle** ∗puzzle)
  Replaces user grid with an unmodified copy.
- int **changeValue** (**Puzzle** ∗puzzle, int x, int y, int value)
  Change user grid square if bitmap allows.
- void **addPuzzle** (**Puzzle** puzzle, **PuzzleArray** ∗puzzleArray, int ∗puzzleCount)
  Dynamically appends puzzle to array and handles memory allocation.
- int **checkRow** (**Puzzle** puzzle, int row)
  Checks if puzzle user grid row includes 1-9 exactly once.
- int **checkColumn** (**Puzzle** puzzle, int col)
  Checks if puzzle user grid column includes 1-9 exactly once.
- int **checkBox** (**Puzzle** puzzle, int startRow, int startCol)
  Checks if puzzle puzzle 3x3 subgrid includes 1-9 exactly once.
- int **isSudokuSolved** (**Puzzle** puzzle)
  Checks if puzzle is fully solved.
- int **isSquareSafe** (**Puzzle** ∗puzzle, int row, int col, int num)
  Checks if num would appear once in row, column, subgrid.
- int **solveSudokuUserGrid** (**Puzzle** ∗puzzle, int row, int col)
  Solves user grid of puzzle.
- int **countSolvedSudokus** (int puzzleArrayCount, **PuzzleArray** puzzleArray)
  Calculates the number of solved puzzles in array.
- void **fillUserGridDiagonal** (**Puzzle** ∗puzzle)
  Generates valid random values for subgrids across the primary diagonal.
- void **setNCluesInUserGrid** (**Puzzle** ∗puzzle, int n)
  Clears user grid squares until n clues remain.
- void **copyUserGridtoGrid** (**Puzzle** ∗puzzle)
  Copies values from user grid to grid in puzzle.
- void **generatePuzzle** (**PuzzleArray** ∗puzzleArrayPtr, int ∗puzzleCountPtr, int clues)
  Generates and appends a valid puzzle with specified number of clues to array.
- void **deleteNthPuzzle** (**PuzzleArray** ∗puzzleArrayPtr, int ∗puzzleCountPtr, int n)
  Deletes nth puzzle from array and reallocates memory.

5.6.1 Detailed Description

Sudoku puzzle logic, dynamic array management.

Author

Kajus Zakaras (`kajus.z@tuta.io`)

Version

1.00

Date

2024-01-25

Copyright

5.6.2 Function Documentation

5.6.2.1 addPuzzle()

```
void addPuzzle (
              Puzzle puzzle,
              PuzzleArray * puzzleArray,
             int * puzzleCount )
```

Dynamically appends puzzle to array and handles memory allocation.

Parameters

puzzle	**Puzzle** (p. 7) to append
puzzleArray	Array to append to
puzzleCount	Array size, increments +1 after automatically

5.6.2.2 changeValue()

```
int changeValue (
              Puzzle * puzzle,
             int x,
             int y,
             int value )
```

Change user grid square if bitmap allows.

Parameters

puzzle	The puzzle to modify
x	Cartesian x coordinate of square
y	Cartesian y coordinate of square
value	Value to write

Returns

-1: unchanged due to bitmap; 0: changed successfully

5.6.2.3 checkBox()

```
int checkBox (
              Puzzle puzzle,
             int startRow,
             int startCol )
```

Checks if puzzle puzzle 3x3 subgrid includes 1-9 exactly once.

Parameters

puzzle	**Puzzle** (p. 7) to check
startRow	Subgrid start row
startCol	Subgrid start column

Returns

1: Proper subgrid; 0: Improper subgrid

5.6.2.4 checkColumn()

```
int checkColumn (
              Puzzle puzzle,
             int col )
```

Checks if puzzle user grid column includes 1-9 exactly once.

Parameters

puzzle	**Puzzle** (p. 7) to check
col	Column to check

Returns

1: Proper column; 0: Improper column

5.6.2.5 checkRow()

```
int checkRow (
              Puzzle puzzle,
             int row )
```

Checks if puzzle user grid row includes 1-9 exactly once.

Parameters

puzzle	**Puzzle** (p. 7) to check
row	Row to check

Returns

1: Proper row; 0: Improper row

5.6.2.6 copyUserGridtoGrid()

```
void copyUserGridtoGrid (
              Puzzle * puzzle )
```

Copies values from user grid to grid in puzzle.

Parameters

puzzle	**Puzzle** (p. 7) to modify

5.6.2.7 countSolvedSudokus()

```
int countSolvedSudokus (
            int puzzleArrayCount,
             PuzzleArray puzzleArray )
```

Calculates the number of solved puzzles in array.

Parameters

puzzleArrayCount	Array size
puzzleArray	Array to check

Returns

Number of solved puzzles in array

Note

Can be optimized in the future by storing .solved value in **Puzzle** (p. 7) structure

5.6.2.8 deleteNthPuzzle()

```
void deleteNthPuzzle (
             PuzzleArray * puzzleArrayPtr,
            int * puzzleCountPtr,
            int n )
```

Deletes nth puzzle from array and reallocates memory.

Parameters

puzzleArrayPtr	Array to delete from
puzzleCountPtr	Array size, increments -1 after automatically
n	nth puzzle to delete,

5.6.2.9 fillUserGridDiagonal()

```
void fillUserGridDiagonal (
             Puzzle * puzzle )
```

Generates valid random values for subgrids across the primary diagonal.

Parameters

puzzle	**Puzzle** (p. 7) to fill

Shuffled using Fisher-Yates algorithm, relies on <time.h> to seed on launch

5.6.2.10 generateBitmap()

```
void generateBitmap (
            Puzzle * puzzle )
```

Generates bitmap in puzzle from its grid array values.

Parameters

puzzle	**Puzzle** (p. 7) for which to generate the bitmap

5.6.2.11 generatePuzzle()

```
void generatePuzzle (
             PuzzleArray * puzzleArrayPtr,
            int * puzzleCountPtr,
            int clues )
```

Generates and appends a valid puzzle with specified number of clues to array.

Parameters

puzzleArrayPtr	Array to append to
puzzleCountPtr	Array size
clues	Number of clues puzzle will have (n>=17)

Note

Make sure new puzzle is properly allocated in stack - current method works based on testing

Dependant on helper functions: \ **generateUserGrid()** (p. 19), **fillUserGridDiagonal()** (p. 18), **solveSudokuUser↩ Grid()** (p. 21), **setNCluesInUserGrid()** (p. 20), \ **copyUserGridtoGrid()** (p. 17), **generateBitmap()** (p. 19), **add↩ Puzzle()** (p. 16)

5.6.2.12 generateUserGrid()

```
void generateUserGrid (
            Puzzle * puzzle )
```

Replaces user grid with an unmodified copy.

Parameters

puzzle	**Puzzle** (p. 7) for which to reset the user grid

5.6.2.13 isSquareSafe()

```
int isSquareSafe (
              Puzzle * puzzle,
             int row,
             int col,
             int num )
```

Checks if num would appear once in row, column, subgrid.

Parameters

puzzle	**Puzzle** (p. 7) to check
row	Row to check
col	Column to check
num	Number that would be written

Returns

1: Safe to write; 0: Unsafe to write

Note

Functions **checkColumn()** (p. 17), **checkRow()** (p. 17), and **checkBox()** (p. 16) do not work here since they check every number

5.6.2.14 isSudokuSolved()

```
int isSudokuSolved (
              Puzzle puzzle )
```

Checks if puzzle is fully solved.

Parameters

puzzle	**Puzzle** (p. 7) to check

Returns

1: Solved; 0: Unsolved

5.6.2.15 setNCluesInUserGrid()

```
void setNCluesInUserGrid (
```

```
 Puzzle * puzzle,
int n )
```

Clears user grid squares until n clues remain.

Parameters

puzzle	**Puzzle** (p. 7) to modify
n	How many clues (non-empty squares) should remain

5.6.2.16 solveSudokuUserGrid()

```
int solveSudokuUserGrid (
 Puzzle * puzzle,
int row,
int col )
```

Solves user grid of puzzle.

Parameters

puzzle	**Puzzle** (p. 7) to solve
row	Call with 0, used by recursive calls
col	Call with 0, used by recursive calls

Returns

0: unsolvable if returned by initial call; used to trigger backtrack in recursion

Uses a backtracking (brute-force) algorithm. More optimized algorithms (e.g. stochastic search) are outside the scope of this project.

5.7 /home/zakajus/Documents/Code/sudoku-solver/puzzle.h File Reference

Header file for **puzzle.h** (p. 21).

```
#include "dependencies.h"
```

Data Structures

- struct **Puzzle**

  Structure to store data of a single Sudoku puzzle.

Typedefs

- typedef struct Puzzle **Puzzle**

  Structure to store data of a single Sudoku puzzle.
- typedef **Puzzle** ∗ **PuzzleArray**

  *Alias for an array of **Puzzle** (p. 7).*

Functions

- void **generateBitmap** (**Puzzle** ∗puzzle)

  Generates bitmap in puzzle from its grid array values.
- void **generateUserGrid** (**Puzzle** ∗puzzle)

  Replaces user grid with an unmodified copy.
- int **changeValue** (**Puzzle** ∗puzzle, int x, int y, int value)

  Change user grid square if bitmap allows.
- void **addPuzzle** (**Puzzle** puzzle, **PuzzleArray** ∗puzzleArray, int ∗puzzleCount)

  Dynamically appends puzzle to array and handles memory allocation.
- int **checkRow** (**Puzzle** puzzle, int row)

  Checks if puzzle user grid row includes 1-9 exactly once.
- int **checkColumn** (**Puzzle** puzzle, int col)

  Checks if puzzle user grid column includes 1-9 exactly once.
- int **checkBox** (**Puzzle** puzzle, int startRow, int startCol)

  Checks if puzzle puzzle 3x3 subgrid includes 1-9 exactly once.
- int **isSudokuSolved** (**Puzzle** puzzle)

  Checks if puzzle is fully solved.
- int **isSquareSafe** (**Puzzle** ∗puzzle, int row, int col, int num)

  Checks if num would appear once in row, column, subgrid.
- int **solveSudokuUserGrid** (**Puzzle** ∗puzzle, int row, int col)

  Solves user grid of puzzle.
- int **countSolvedSudokus** (int puzzleArrayCount, **PuzzleArray** puzzleArray)

  Calculates the number of solved puzzles in array.
- void **fillUserGridDiagonal** (**Puzzle** ∗puzzle)

  Generates valid random values for subgrids across the primary diagonal.
- void **setNCluesInUserGrid** (**Puzzle** ∗puzzle, int n)

  Clears user grid squares until n clues remain.
- void **copyUserGridtoGrid** (**Puzzle** ∗puzzle)

  Copies values from user grid to grid in puzzle.
- void **generatePuzzle** (**PuzzleArray** ∗puzzleArrayPtr, int ∗puzzleCountPtr, int clues)

  Generates and appends a valid puzzle with specified number of clues to array.
- void **deleteNthPuzzle** (**PuzzleArray** ∗puzzleArrayPtr, int ∗puzzleCountPtr, int n)

  Deletes nth puzzle from array and reallocates memory.

5.7.1 Detailed Description

Header file for **puzzle.h** (p. 21).

Author

Kajus Zakaras (kajus.z@tuta.io)

Version

1.00

Date

2024-01-25

Copyright

5.7.2 Typedef Documentation

5.7.2.1 Puzzle

```
typedef struct Puzzle Puzzle
```

Structure to store data of a single Sudoku puzzle.

The grid is always fixed, except for when generating a new sudoku

5.7.3 Function Documentation

5.7.3.1 addPuzzle()

```
void addPuzzle (
              Puzzle puzzle,
              PuzzleArray * puzzleArray,
             int * puzzleCount )
```

Dynamically appends puzzle to array and handles memory allocation.

Parameters

puzzle	**Puzzle** (p. 7) to append
puzzleArray	Array to append to
puzzleCount	Array size, increments +1 after automatically

5.7.3.2 changeValue()

```
int changeValue (
              Puzzle * puzzle,
             int x,
             int y,
             int value )
```

Change user grid square if bitmap allows.

Parameters

puzzle	The puzzle to modify
x	Cartesian x coordinate of square
y	Cartesian y coordinate of square
value	Value to write

Returns

-1: unchanged due to bitmap; 0: changed successfully

5.7.3.3 checkBox()

```
int checkBox (
              Puzzle puzzle,
             int startRow,
             int startCol )
```

Checks if puzzle puzzle 3x3 subgrid includes 1-9 exactly once.

Parameters

puzzle	**Puzzle** (p. 7) to check
startRow	Subgrid start row
startCol	Subgrid start column

Returns

1: Proper subgrid; 0: Improper subgrid

5.7.3.4 checkColumn()

```
int checkColumn (
              Puzzle puzzle,
             int col )
```

Checks if puzzle user grid column includes 1-9 exactly once.

Parameters

puzzle	**Puzzle** (p. 7) to check
col	Column to check

Returns

1: Proper column; 0: Improper column

5.7.3.5 checkRow()

```
int checkRow (
              Puzzle puzzle,
             int row )
```

Checks if puzzle user grid row includes 1-9 exactly once.

Parameters

puzzle	**Puzzle** (p. 7) to check
row	Row to check

Returns

1: Proper row; 0: Improper row

5.7.3.6 copyUserGridtoGrid()

```
void copyUserGridtoGrid (
              Puzzle * puzzle )
```

Copies values from user grid to grid in puzzle.

Parameters

puzzle	**Puzzle** (p. 7) to modify

5.7.3.7 countSolvedSudokus()

```
int countSolvedSudokus (
             int puzzleArrayCount,
              PuzzleArray puzzleArray )
```

Calculates the number of solved puzzles in array.

Parameters

puzzleArrayCount	Array size
puzzleArray	Array to check

Returns

Number of solved puzzles in array

Note

Can be optimized in the future by storing .solved value in **Puzzle** (p. 7) structure

5.7.3.8 deleteNthPuzzle()

```
void deleteNthPuzzle (
              PuzzleArray * puzzleArrayPtr,
             int * puzzleCountPtr,
             int n )
```

Deletes nth puzzle from array and reallocates memory.

Parameters

puzzleArrayPtr	Array to delete from
puzzleCountPtr	Array size, increments -1 after automatically
n	nth puzzle to delete,

5.7.3.9 fillUserGridDiagonal()

```
void fillUserGridDiagonal (
              Puzzle * puzzle )
```

Generates valid random values for subgrids across the primary diagonal.

Parameters

puzzle	**Puzzle** (p. 7) to fill

Shuffled using Fisher-Yates algorithm, relies on <time.h> to seed on launch

5.7.3.10 generateBitmap()

```
void generateBitmap (
              Puzzle * puzzle )
```

Generates bitmap in puzzle from its grid array values.

Parameters

puzzle	**Puzzle** (p. 7) for which to generate the bitmap

5.7.3.11 generatePuzzle()

```
void generatePuzzle (
              PuzzleArray * puzzleArrayPtr,
             int * puzzleCountPtr,
             int clues )
```

Generates and appends a valid puzzle with specified number of clues to array.

Parameters

puzzleArrayPtr	Array to append to
puzzleCountPtr	Array size
clues	Number of clues puzzle will have (n>=17)

Note

Make sure new puzzle is properly allocated in stack - current method works based on testing

Dependant on helper functions: \ **generateUserGrid()** (p. 19), **fillUserGridDiagonal()** (p. 18), **solveSudokuUser↩ Grid()** (p. 21), **setNCluesInUserGrid()** (p. 20), \ **copyUserGridtoGrid()** (p. 17), **generateBitmap()** (p. 19), **add↩ Puzzle()** (p. 16)

5.7.3.12 generateUserGrid()

```
void generateUserGrid (
              Puzzle * puzzle )
```

Replaces user grid with an unmodified copy.

Parameters

puzzle	**Puzzle** (p. 7) for which to reset the user grid

5.7.3.13 isSquareSafe()

```
int isSquareSafe (
              Puzzle * puzzle,
             int row,
             int col,
             int num )
```

Checks if num would appear once in row, column, subgrid.

Parameters

puzzle	**Puzzle** (p. 7) to check
row	Row to check
col	Column to check
num	Number that would be written

Returns

1: Safe to write; 0: Unsafe to write

Note

Functions **checkColumn()** (p. 17), **checkRow()** (p. 17), and **checkBox()** (p. 16) do not work here since they check every number

5.7.3.14 isSudokuSolved()

```
int isSudokuSolved (
            Puzzle puzzle )
```

Checks if puzzle is fully solved.

Parameters

puzzle	**Puzzle** (p. 7) to check

Returns

1: Solved; 0: Unsolved

5.7.3.15 setNCluesInUserGrid()

```
void setNCluesInUserGrid (
            Puzzle * puzzle,
           int n )
```

Clears user grid squares until n clues remain.

Parameters

puzzle	**Puzzle** (p. 7) to modify
n	How many clues (non-empty squares) should remain

5.7.3.16 solveSudokuUserGrid()

```
int solveSudokuUserGrid (
            Puzzle * puzzle,
           int row,
           int col )
```

Solves user grid of puzzle.

Parameters

puzzle	**Puzzle** (p. 7) to solve
row	Call with 0, used by recursive calls
col	Call with 0, used by recursive calls

Returns

0: unsolvable if returned by initial call; used to trigger backtrack in recursion

Uses a backtracking (brute-force) algorithm. More optimized algorithms (e.g. stochastic search) are outside the scope of this project.

5.8 /home/zakajus/Documents/Code/sudoku-solver/puzzle.h

Go to the documentation of this file.

```
00001
00012 #ifndef PUZZLE_H
00013 #define PUZZLE_H
00014
00015 #include "dependencies.h"
00016
00017
00020 typedef struct Puzzle {
00022     int grid[GRID_SIZE][GRID_SIZE];
00024     int userGrid[GRID_SIZE][GRID_SIZE];
00027     int map[GRID_SIZE][GRID_SIZE];
00028 } Puzzle;
00029
00030
00032 typedef Puzzle* PuzzleArray;
00033
00034
00035 void generateBitmap(Puzzle *puzzle);
00036 void generateUserGrid(Puzzle *puzzle);
00037 int changeValue(Puzzle *puzzle, int x, int y, int value);
00038 void addPuzzle(Puzzle puzzle, PuzzleArray *puzzleArray, int *puzzleCount);
00039 int checkRow(Puzzle puzzle, int row);
00040 int checkColumn(Puzzle puzzle, int col);
00041 int checkBox(Puzzle puzzle, int startRow, int startCol);
00042 int isSudokuSolved(Puzzle puzzle);
00043 int isSquareSafe(Puzzle *puzzle, int row, int col, int num);
00044 int solveSudokuUserGrid(Puzzle *puzzle, int row, int col);
00045 int countSolvedSudokus(int puzzleArrayCount, PuzzleArray puzzleArray);
00046 void fillUserGridDiagonal(Puzzle* puzzle);
00047 void setNCluesInUserGrid(Puzzle* puzzle, int n);
00048 void copyUserGridtoGrid(Puzzle *puzzle);
00049 void generatePuzzle(PuzzleArray *puzzleArrayPtr, int *puzzleCountPtr, int clues);
00050 void deleteNthPuzzle(PuzzleArray *puzzleArrayPtr, int *puzzleCountPtr, int n);
00051
00052
00053 #endif
```

5.9 /home/zakajus/Documents/Code/sudoku-solver/ui.c File Reference

Displaying CLI interfaces, localized string translation.

```
#include "./dependencies.h"
#include "./ui.h"
#include "./puzzle.h"
#include "./files.h"
```

Functions

- char * **translate** (char *key)

  Used to translate a display string key into string value from a localized dictionary.
- void **clearDisplay** ()

  Clears CLI display.
- void **displayBanner** ()

  Display Sudoku banner.
- void **displayPuzzleGrid** (**Puzzle** puzzle)

  Displays grid of puzzle in CLI.

- void **displayPuzzleUserGrid** (**Puzzle** puzzle)

  Displays user grid of puzzle in CLI.
- void **menuPlay** (**Puzzle** ∗puzzle)

  Opens CLI for playing a puzzle.
- void **menuChoosePuzzle** (**PuzzleArray** ∗puzzleArray, int puzzleCount)

  Opens CLI to choose puzzle to play.
- void **menuDelete** (**PuzzleArray** ∗puzzleArrayPtr, int ∗puzzleCountPtr)

  Opens CLI to delete a puzzle from array.
- void **menuSolver** (**PuzzleArray** ∗puzzleArrayPtr, int puzzleCount)

  Opens CLI for algorithmic solver.
- void **menuStats** (**PuzzleArray** ∗puzzleArray, int puzzleCount)

  Opens CLI to show statistics.
- void **menuGenerate** (**PuzzleArray** ∗puzzleArrayPtr, int ∗puzzleCountPtr)

  Opens CLI to generate a new puzzle.
- void **menuManager** (**PuzzleArray** ∗puzzleArrayPtr, int ∗puzzleCountPtr, **PuzzleArray** defaultPuzzleArray, int defaultPuzzleCount)

  Opens CLI to reset data, navigate to menus to generate or delete puzzles.
- void **menuMain** (**PuzzleArray** ∗puzzleArray, int ∗puzzleCountPtr, **PuzzleArray** defaultPuzzles, int default↩ PuzzleCount)

  Opens main CLI menu when program is launched, used to navigate to all other menus.

Variables

- **translation dictionaryEN** [ ]

  English key-value dictionary for display strings.

5.9.1 Detailed Description

Displaying CLI interfaces, localized string translation.

Author

Kajus Zakaras (`kajus.z@tuta.io`)

Version

1.00

Date

2024-01-25

Copyright

5.9.2 Function Documentation

5.9.2.1 displayBanner()

```
void displayBanner ( )
```

Display Sudoku banner.

Note

Improper tab indentation is intentional - otherwise banner deforms

5.9.2.2 displayPuzzleGrid()

```
void displayPuzzleGrid (
            Puzzle puzzle )
```

Displays grid of puzzle in CLI.

Parameters

puzzle	The puzzle to display

Note

As of version 1.00, never used in production, however very useful for debugging

5.9.2.3 displayPuzzleUserGrid()

```
void displayPuzzleUserGrid (
            Puzzle puzzle )
```

Displays user grid of puzzle in CLI.

Parameters

puzzle	The puzzle to display

5.9.2.4 menuChoosePuzzle()

```
void menuChoosePuzzle (
            PuzzleArray * puzzleArray,
           int puzzleCount )
```

Opens CLI to choose puzzle to play.

Parameters

puzzleArray	Array to choose puzzle from
puzzleCount	Array size

Launched by **menuMain()** (p. 32), launches **menuPlay()** (p. 33) after puzzle is chosen

5.9.2.5 menuDelete()

```
void menuDelete (
             PuzzleArray * puzzleArrayPtr,
            int * puzzleCountPtr )
```

Opens CLI to delete a puzzle from array.

Parameters

puzzleArrayPtr	Array to delete from
puzzleCountPtr	Array size

Launched by **menuManager()** (p. 33)

5.9.2.6 menuGenerate()

```
void menuGenerate (
             PuzzleArray * puzzleArrayPtr,
            int * puzzleCountPtr )
```

Opens CLI to generate a new puzzle.

Parameters

puzzleArrayPtr	Array to generate puzzle to
puzzleCountPtr	Array size

Launched by **menuManager()** (p. 33)

5.9.2.7 menuMain()

```
void menuMain (
             PuzzleArray * puzzleArray,
            int * puzzleCountPtr,
             PuzzleArray defaultPuzzles,
            int defaultPuzzleCount )
```

Opens main CLI menu when program is launched, used to navigate to all other menus.

Parameters

puzzleArray	**Puzzle** (p. 7) array to play from
puzzleCountPtr	**Puzzle** (p. 7) array size
defaultPuzzles	Default puzzle array to reset save to
defaultPuzzleCount	Default puzzle array size

Launched by main()

5.9.2.8 menuManager()

```
void menuManager (
             PuzzleArray * puzzleArrayPtr,
            int * puzzleCountPtr,
             PuzzleArray defaultPuzzleArray,
            int defaultPuzzleCount )
```

Opens CLI to reset data, navigate to menus to generate or delete puzzles.

Parameters

puzzleArrayPtr	**Puzzle** (p. 7) array to play from
puzzleCountPtr	**Puzzle** (p. 7) array size
defaultPuzzleArray	Default puzzle array to reset save to
defaultPuzzleCount	Default puzzle array size

Launched by **menuManager()** (p. 33), can launch **menuGenerate()** (p. 32), **menuDelete()** (p. 32)

5.9.2.9 menuPlay()

```
void menuPlay (
             Puzzle * puzzle )
```

Opens CLI for playing a puzzle.

Parameters

puzzle	**Puzzle** (p. 7) to play

Launched by **menuChoosePuzzle()** (p. 31)

5.9.2.10 menuSolver()

```
void menuSolver (
             PuzzleArray * puzzleArrayPtr,
            int puzzleCount )
```

Opens CLI for algorithmic solver.

Parameters

puzzleArrayPtr	Array to solve from
puzzleCount	Array size

Launched by **menuMain()** (p. 32)

5.9.2.11 menuStats()

```
void menuStats (
            PuzzleArray * puzzleArray,
            int puzzleCount )
```

Opens CLI to show statistics.

Parameters

puzzleArray	Array to show statistic from
puzzleCount	Array size

Launched by **menuMain()** (p. 32)

5.9.2.12 translate()

```
char * translate (
            char * key )
```

Used to translate a display string key into string value from a localized dictionary.

Note

English dictionary (dictionaryEN) is hard-coded for now, however replacing it can be easily done using a macro

Parameters

key	Display string key to translate

Returns

Localized string for display output

5.9.3 Variable Documentation

5.9.3.1 dictionaryEN

```
translation dictionaryEN[]
```

English key-value dictionary for display strings.

Note

ERROR prefix strings are used in stderr stream

5.10 /home/zakajus/Documents/Code/sudoku-solver/ui.h File Reference

Header file for **ui.c** (p. 29).

```
#include "./dependencies.h"
#include "./puzzle.h"
```

Data Structures

- struct **translation**

  Key-value (dictionary) pair to store display strings.

Functions

- char * **translate** (char *key)

  Used to translate a display string key into string value from a localized dictionary.
- void **clearDisplay** ()

  Clears CLI display.
- void **displayBanner** ()

  Display Sudoku banner.
- void **displayPuzzleGrid** (**Puzzle** puzzle)

  Displays grid of puzzle in CLI.
- void **displayPuzzleUserGrid** (**Puzzle** puzzle)

  Displays user grid of puzzle in CLI.
- void **menuPlay** (**Puzzle** *puzzle)

  Opens CLI for playing a puzzle.
- void **menuChoosePuzzle** (**PuzzleArray** *puzzleArray, int puzzleCount)

  Opens CLI to choose puzzle to play.
- void **menuDelete** (**PuzzleArray** *puzzleArrayPtr, int *puzzleCountPtr)

  Opens CLI to delete a puzzle from array.
- void **menuSolver** (**PuzzleArray** *puzzleArrayPtr, int puzzleCount)

  Opens CLI for algorithmic solver.
- void **menuStats** (**PuzzleArray** *puzzleArray, int puzzleCount)

  Opens CLI to show statistics.
- void **menuGenerate** (**PuzzleArray** *puzzleArrayPtr, int *puzzleCountPtr)

  Opens CLI to generate a new puzzle.
- void **menuManager** (**PuzzleArray** *puzzleArrayPtr, int *puzzleCountPtr, **PuzzleArray** defaultPuzzleArray, int defaultPuzzleCount)

  Opens CLI to reset data, navigate to menus to generate or delete puzzles.
- void **menuMain** (**PuzzleArray** *puzzleArray, int *puzzleCountPtr, **PuzzleArray** defaultPuzzles, int defaultPuzzleCount)

  Opens main CLI menu when program is launched, used to navigate to all other menus.

5.10.1 Detailed Description

Header file for **ui.c** (p. 29).

Author

Kajus Zakaras (kajus.z@tuta.io)

Version

1.00

Date

2024-01-25

Copyright

Copyright (c) 2024

5.10.2 Function Documentation

5.10.2.1 displayBanner()

```
void displayBanner ( )
```

Display Sudoku banner.

Note

Improper tab indentation is intentional - otherwise banner deforms

5.10.2.2 displayPuzzleGrid()

```
void displayPuzzleGrid (
            Puzzle puzzle )
```

Displays grid of puzzle in CLI.

Parameters

puzzle	The puzzle to display

Note

As of version 1.00, never used in production, however very useful for debugging

5.10.2.3 displayPuzzleUserGrid()

```
void displayPuzzleUserGrid (
            Puzzle puzzle )
```

Displays user grid of puzzle in CLI.

Parameters

puzzle	The puzzle to display

5.10.2.4 menuChoosePuzzle()

```
void menuChoosePuzzle (
            PuzzleArray * puzzleArray,
            int puzzleCount )
```

Opens CLI to choose puzzle to play.

Parameters

puzzleArray	Array to choose puzzle from
puzzleCount	Array size

Launched by **menuMain()** (p. 32), launches **menuPlay()** (p. 33) after puzzle is chosen

5.10.2.5 menuDelete()

```
void menuDelete (
            PuzzleArray * puzzleArrayPtr,
            int * puzzleCountPtr )
```

Opens CLI to delete a puzzle from array.

Parameters

puzzleArrayPtr	Array to delete from
puzzleCountPtr	Array size

Launched by **menuManager()** (p. 33)

5.10.2.6 menuGenerate()

```
void menuGenerate (
            PuzzleArray * puzzleArrayPtr,
            int * puzzleCountPtr )
```

Opens CLI to generate a new puzzle.

Parameters

puzzleArrayPtr	Array to generate puzzle to
puzzleCountPtr	Array size

Launched by **menuManager()** (p. 33)

5.10.2.7 menuMain()

```
void menuMain (
              PuzzleArray * puzzleArray,
             int * puzzleCountPtr,
              PuzzleArray defaultPuzzles,
             int defaultPuzzleCount )
```

Opens main CLI menu when program is launched, used to navigate to all other menus.

Parameters

puzzleArray	**Puzzle** (p. 7) array to play from
puzzleCountPtr	**Puzzle** (p. 7) array size
defaultPuzzles	Default puzzle array to reset save to
defaultPuzzleCount	Default puzzle array size

Launched by main()

5.10.2.8 menuManager()

```
void menuManager (
              PuzzleArray * puzzleArrayPtr,
             int * puzzleCountPtr,
              PuzzleArray defaultPuzzleArray,
             int defaultPuzzleCount )
```

Opens CLI to reset data, navigate to menus to generate or delete puzzles.

Parameters

puzzleArrayPtr	**Puzzle** (p. 7) array to play from
puzzleCountPtr	**Puzzle** (p. 7) array size
defaultPuzzleArray	Default puzzle array to reset save to
defaultPuzzleCount	Default puzzle array size

Launched by **menuManager()** (p. 33), can launch **menuGenerate()** (p. 32), **menuDelete()** (p. 32)

5.10.2.9 menuPlay()

```
void menuPlay (
              Puzzle * puzzle )
```

Opens CLI for playing a puzzle.

Parameters

puzzle	**Puzzle** (p. 7) to play

Launched by **menuChoosePuzzle()** (p. 31)

5.10.2.10 menuSolver()

```
void menuSolver (
             PuzzleArray * puzzleArrayPtr,
            int puzzleCount )
```

Opens CLI for algorithmic solver.

Parameters

puzzleArrayPtr	Array to solve from
puzzleCount	Array size

Launched by **menuMain()** (p. 32)

5.10.2.11 menuStats()

```
void menuStats (
             PuzzleArray * puzzleArray,
            int puzzleCount )
```

Opens CLI to show statistics.

Parameters

puzzleArray	Array to show statistic from
puzzleCount	Array size

Launched by **menuMain()** (p. 32)

5.10.2.12 translate()

```
char * translate (
            char * key )
```

Used to translate a display string key into string value from a localized dictionary.

Note

English dictionary (dictionaryEN) is hard-coded for now, however replacing it can be easily done using a macro

Parameters

key	Display string key to translate

Returns

Localized string for display output

5.11 /home/zakajus/Documents/Code/sudoku-solver/ui.h

Go to the documentation of this file.

```
00001
00012 #ifndef UI_I
00013 #define UI_I
00014
00015
00016 #include "./dependencies.h"
00017 #include "./puzzle.h"
00018
00019
00021 typedef struct {
00023     char* key;
00025     char value[LINE_MAX];
00026 } translation;
00027
00028
00029 char* translate(char* key);
00030 void clearDisplay();
00031 void displayBanner();
00032 void displayPuzzleGrid(Puzzle puzzle);
00033 void displayPuzzleUserGrid(Puzzle puzzle);
00034
00035
00036 void menuPlay(Puzzle *puzzle);
00037 void menuChoosePuzzle(PuzzleArray *puzzleArray, int puzzleCount);
00038 void menuDelete(PuzzleArray *puzzleArrayPtr, int *puzzleCountPtr);
00039 void menuSolver(PuzzleArray *puzzleArrayPtr, int puzzleCount);
00040 void menuStats(PuzzleArray *puzzleArray, int puzzleCount);
00041 void menuGenerate(PuzzleArray *puzzleArrayPtr, int *puzzleCountPtr);
00042 void menuManager(PuzzleArray *puzzleArrayPtr, int *puzzleCountPtr, PuzzleArray defaultPuzzleArray, int
      defaultPuzzleCount);
00043 void menuMain(PuzzleArray *puzzleArray, int *puzzleCountPtr, PuzzleArray defaultPuzzles, int
      defaultPuzzleCount);
00044
00045
00046 #endif
```

Index