SUDOKU

1.0.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 sScore Struct Reference

`#include <score_service.h>`

Data Fields

- char name [10]
- int userId
- int time
- int difficulty
- struct sScore * next

3.1.1 Field Documentation

3.1.1.1 difficulty

`int sScore::difficulty`

3.1.1.2 name

`char sScore::name[10]`

3.1.1.3 next

`struct sScore* sScore::next`

3.1.1.4 time

```
int sScore::time
```

3.1.1.5 userId

```
int sScore::userId
```

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

- headers/services/score_service.h

Chapter 4

File Documentation

4.1 headers/core/game.h File Reference

```
#include "../shared/shared.h"
```

Functions

- void fillNotesForCell (int iX_coordinate, int iY_coordinate)
- int isElementInArray (int iArray[ ], int iNumber, int iSize)
- void resetArray (int iArray[ ], int iSize)
- void generateGameData (int iGameData[ ][9])
- void deleteCells (int iArray[ ][9], int iDifficulty)
- void navigateTo (int iPos)
- void makeNote (int iX, int iY, int iSuggestion)
- int isElementInBox (int iArr[ ][9], int iBox_start_row, int iBox_start_col, int iEle)
- int generateRandomNumber ()
- int solveGame (int iGameData[ ][9])
- void resetGameData (int iGmaeData[ ][9])
- int generateNumberByInterval (int x, int y)
- int getGridStatus (int array[ ][9])
- void solveCell (int iGameData[ ][9], int iX, int iY)
- int getGameStatus (int iArray[ ][9])
- void solveAll (int iGameData[ ][9], int iDeletedCells[ ][9])
- int timer (int iAction)
- void timeToString (int iUserTime, char cStringTime[ ])
- void checkGameState ()

Variables

- int iMarks [9][9][MAX_MARKS]

  Array wo die Notizen gespeicht werden.
- int iX_coordinate

  Cursor X-Koordinate.
- int iY_coordinate

  Cursor Y-Koordinate.
- int iDifficulty

  Enthält die aktuelle Schwierigkeinsstufe.
- char cGameMessage [200]

  Hier werden Spiel Mitteilungen gespeichert. Sie wird nach jeder Iteration zurückgesetzt.
- int iIsGameActive

  Schalter ob das Spiel weiter oder beendet werden soll.
- int iCurrentPosition

  Aktiver Spiel-Screen.
- char cUusername [8]

  Hier wird der Name des gerade angemeldeten Users gespeichert.
- int * piIsUserLoggedIn

  Hier wird nach jedem Login versuch den Status abgespeichert. 1 => Erfolgreich, 0 => Fehlgeschlagen.
- int iExitTheGame
- int isSolvedAutomatic
- int * piUserID
- int iGameData [9][9]

  array which holds the game data.
- int iDeletedCells [9][9]

  array which holds deleted cells to keep track of them.
- int iUserCells [9][9]

  array which holds the coordinates of user cells (cells to solve) to keep track of theme.
- int iAnzahlDerTipps
- int iAnzahlDerHilfe
- int iErlaubteAnzahlDerHilfe
- int iErlaubteAnzahlDerTipps

4.1.1 Function Documentation

4.1.1.1 checkGameState()

```
void checkGameState ( )
```

Leitet nötige Schritte zur Überprüfung, ob das Sudoku vollständig und richtig gelöst wurde, ein.

4.1.1.2 deleteCells()

```
void deleteCells (
            int iArray[][9],
            int iDifficulty )
```

Löscht je nach Schwierigkeitsgrad mehr oder weniger zufällige Zellen aus dem sichtbaren Spielfeld und speichert die gelöschten Zellen im Array iDeletedCells;

1. Parameter: Spielfeld
2. Parameter: Schwierigkeitsgrad

4.1.1.3 fillNotesForCell()

```
void fillNotesForCell (
            int iX_coordinate,
            int iY_coordinate )
```

Wird aufgerufen, wenn der Spieler einen Tipp haben möchte. Befüllt die Notiz in der entsprechenden Zelle mit drei Zahlen, von denen eine die richtige im Kontext des Sudokus darstellt.

1. Parameter: x-Koordinate, an der sich der Spieler befindet
2. Parameter: y-Koordinate, an der sich der Spieler befindet

4.1.1.4 generateGameData()

```
void generateGameData (
            int iGameData[][9] )
```

Generiert vor jedem neuen Spiel ein neues gelöstet Spielfeld. Sorgt damit dafür, dass das Sudoku lösbar ist. Der Vorgang wird nach zwei Sekunden neu gestartet, fall die Funktion zu keiner Lösung gekommen ist, um eine Endlossschleife zu vermeiden.

1. Parameter: leeres Spielfeld

4.1.1.5 generateNumberByInterval()

```
int generateNumberByInterval (
            int x,
            int y )
```

Generiert eine Zufallszahl in einem definierbaren Intervall.

1. Parameter: Intervall-Start
2. Parameter: Intervall-Ende Rückgabewert: generierte Zufallszahl

4.1.1.6 generateRandomNumber()

```
int generateRandomNumber ( )
```

Generiert eine Zufallszahl zwischen 1 und 9. Rückgabewert: generierte Zufallszahl

4.1.1.7 getGameStatus()

```
int getGameStatus (
            int iArray[][9] )
```

Überprüft, ob alle Felder des sichtbaren Spielfeldes aufgefüllt wurden.

1. Parameter: sichtbares Spielfeld Rückgabewert: Vollständig (FILLED) oder unvollständig (NOT_FILLED)

4.1.1.8 getGridStatus()

```
int getGridStatus (
            int array[][9] )
```

Parameters

array[][]	which holds the game data.

Returns

Returns a integer which indicates if the grid if filled complete.

Checks if the grid if filled complete.

4.1.1.9 isElementInArray()

```
int isElementInArray (
            int iArray[],
            int iNumber,
            int iSize )
```

Überpüft, ob sich ein bestimmtes Element in einen bestimmten Array befindet.

1. Parameter: zu überprüfender Array
2. Parameter: Element, nach dem gesucht werden soll
3. Parameter: Länge des Arrays Rückgabewert: Stelle, an der das Element gefunden wurde oder -1, falls es nicht im Array ist

4.1.1.10 isElementInBox()

```
int isElementInBox (
            int iArr[][9],
            int iBox_start_row,
            int iBox_start_col,
            int iEle )
```

Dient zur Überprüfung, ob eine bestimmte Zahl in einer bestimmten der neun Unterquadrate des Sudokus vorhanden ist.

1. Parameter: aktuelles Spielfeld
2. Parameter: y-Koordinate der oberen, linken Ecke der Box
3. Parameter: x-Koordinate der obenen, linken Ecke der Box
4. Parameter: Element, nach dem gesucht werden soll Rückgabewert: Gefunden? 1 -> Ja, -1 -> Nein

4.1.1.11 makeNote()

```
void makeNote (
            int iX,
            int iY,
            int iSuggestion )
```

Wird nur aufgerufen, wenn sich der Spieler im "Makiere-Modus" befindet. Es können bis zu drei Zahlen in eine Notiz eingetragen werden. Wenn die Notiz voll ist und eine weitere Zahl eingetragen wird, wird die Notiz gelöscht und die weitere Zahl eingetragen.

1. Parameter: x-Koordinate, an der der Spieler sich befindet
2. Parameter: y-Koordinate, an der der Spieler sich befindet
3. Parameter: Zahl, die der Spiele in die Notiz schreiben möchte

4.1.1.12 navigateTo()

```
void navigateTo (
            int iPos )
```

Parameters

iPos	of where to navigate

Moves the cursor in the desired direction

4.1.1.13 resetArray()

```
void resetArray (
            int iArray[],
            int iSize )
```

Setzt einen Array auf den Ursprungszustand zurück, indem alle Werte auf Null gesetzt werden.

1. Parameter: der zurückzusetzende Array
2. Parameter: Länge des Arrays

4.1.1.14 resetGameData()

```
void resetGameData (
            int iGmaeData[][9] )
```

Setzt das sichtbare Spielfeld in seinen Ursprungszustand zurück, indem alle Werte auf Null gesetzt werden.

1. Parameter: aktuelles Spielfeld

4.1.1.15 solveAll()

```
void solveAll (
            int iGameData[][9],
            int iDeletedCells[][9] )
```

Wird aufgerufen, wenn der Spieler aufgibt und das Sudoku auflösen lassen möchte. Alle Felder, die durch den Nutzer hätten gefüllt werden sollen, werden von dem Array der gelöschten Zellen in das Spielfeld über- tragen.

1. Parameter: aktuelles Spielfeld
2. Parameter: Array mit den gelöschten Zellen (Lösung)

4.1.1.16 solveCell()

```
void solveCell (
            int iGameData[][9],
            int iX,
            int iY )
```

Wird aufgerufen, wenn der Spieler die Hilfefunktion nutzt. Aus dem zweidimensionalen Array, in dem die gelöschten Zellen gespeichert sind, wird der Lösungswert in das aktuelle Spielfeld eingetragen.

1. Parameter: aktuelles Spielfeld
2. Parameter: x-Koordinate, an der der Spieler sich befindet
3. Parameter: y-Koordinate, an der der Spieler sich befindet

4.1.1.17 solveGame()

```
int solveGame (
            int iGameData[][9] )
```

Wird aufgerufen, wenn die keine Zellen mehr ausgefüllt werden müssen. Überprüft, ob der Spieler das Spielfeld richtig gelöst hat.

1. Parameter: aktuelles Spielfeld Rückgabewert: Richtig gelöst (1) oder es existiert mindestens ein Fehler (0)

4.1.1.18 timer()

```
int timer (
            int iAction )
```

Stoppuhr für das Spiel. Wird zur Messung der benötigten Zeit zum Lösen des Sudokus verwendet. Stoppuhr kann (neu-)gestartet, ge- stopped und pausiert werden.

1. Parameter: Aktion für die Stoppuhr (Zeit abfragen, Zurück- setzen, Starten, Stoppen)

4.1.1.19 timeToString()

```
void timeToString (
            int iUserTime,
            char cStringTime[] )
```

Wandelt eine Zeitangabe in Sekunden in das "00:00"-Format um.

1. Parameter: Zeitangabe in Sekunden
2. Parameter: Char-Array, in den die formatierte Zeit geschrieben werden soll

4.1.2 Variable Documentation

4.1.2.1 cGameMessage

```
char cGameMessage[200]
```

Hier werden Spiel Mitteilungen gespeichert. Sie wird nach jeder Iteration zurückgesetzt.

4.1.2.2 cUusername

```
char cUusername[8]
```

Hier wird der Name des gerade angemeldeten Users gespeichert.

4.1.2.3 iAnzahlDerHilfe

```
int iAnzahlDerHilfe
```

4.1.2.4 iAnzahlDerTipps

```
int iAnzahlDerTipps
```

4.1.2.5 iCurrentPosition

```
int iCurrentPosition
```

Aktiver Spiel-Screen.

4.1.2.6 iDeletedCells

```
int iDeletedCells[9][9]
```

array which holds deleted cells to keep track of them.

4.1.2.7 iDifficulty

```
int iDifficulty
```

Enthält die aktuelle Schwierigkeinsstufe.

4.1.2.8 iErlaubteAnzahlDerHilfe

```
int iErlaubteAnzahlDerHilfe
```

4.1.2.9 iErlaubteAnzahlDerTipps

```
int iErlaubteAnzahlDerTipps
```

4.1.2.10 iExitTheGame

```
int iExitTheGame
```

4.1.2.11 iGameData

```
int iGameData[9][9]
```

array which holds the game data.

4.1.2.12 iIsGameActive

```
int iIsGameActive
```

Schalter ob das Spiel weiter oder beendet werden soll.

4.1.2.13 iMarks

```
int iMarks[9][9][MAX_MARKS]
```

Array wo die Notizen gespeicht werden.

4.1.2.14 isSolvedAutomatic

```
int isSolvedAutomatic
```

4.1.2.15 iUserCells

```
int iUserCells[9][9]
```

array which holds the coordinates of user cells (cells to solve) to keep track of theme.

4.1.2.16 iX_coordinate

```
int iX_coordinate
```

Cursor X-Koordinate.

4.1.2.17 iY_coordinate

```
int iY_coordinate
```

Cursor Y-Koordinate.

4.1.2.18 piIsUserLoggedIn

```
int* piIsUserLoggedIn
```

Hier wird nach jedem Login versuch den Status abgespeichert. 1 => Erfolgreich, 0 => Fehlgeschlagen.

4.1.2.19 piUserID

```
int* piUserID
```

4.2 headers/core/inputHandler.h File Reference

```
#include "../../headers/shared/shared.h"
```

Functions

- void handleUserNameInput ()
- void handleSetPasswordInput ()
- void handleEnterPasswordInput ()
- void handleDifficultyDialogInput (int iUserInput)
- void handleMenuInput (int iUserInput)
- void handleInGameInput (int iUserInput)
- void handleSolvedGameInput (int iUserInput)
- void handleSetMarkInput (int iUserInput)
- void handleDetailsInput (int iUserInput)
- void handleDetailsDialogInput (int iUserInput)
- void handleHelpInput (int iUserInput)
- void setConfig ()

4.2.1 Function Documentation

4.2.1.1 handleDetailsDialogInput()

```
void handleDetailsDialogInput (
            int iUserInput )
```

Wenn der Spieler eine Eingabe im Dialog, in dem er den Schwierigkeitsgrad für die Bestenliste auswählen soll, tätigt, wird diese Funktion aufgerufen und seine Eingabe wird verarbeitet.

1. Parameter: Eingabe des Nutzers

4.2.1.2 handleDetailsInput()

```
void handleDetailsInput (
            int iUserInput )
```

Wird aufgerufen, sobald der Nutzer versucht etwas einzugeben, während er sich in der Bestenliste befindet. "z" bringt den Spieler zurück zum Menü.

1. Parameter: Eingabe des Nutzers

4.2.1.3 handleDifficultyDialogInput()

```
void handleDifficultyDialogInput (
            int iUserInput )
```

Wird aufgerufen, wenn sich ein bereits registrierter Nutzer anmelden möchte. Dieser kann nun sein Passwort eingeben. Das eingegebene Passwort wird dem in der Datenbank hinterlegtem Passwort verglichen und auf Übereinstimmung geprüft. Es werden einzelne Buchstaben eingelesen, um den Dialog dynamisch zu gestalten und das nutzen der Löschen-Taste zu ermöglichen, um einzelne Buchstaben löschen zu können. So wird ein möglichst nutzerfreundliches Erlebnis garantiert.

4.2.1.4 handleEnterPasswordInput()

```
void handleEnterPasswordInput ( )
```

Wird aufgerufen, wenn sich ein bereits registrierter Nutzer anmelden möchte. Dieser kann nun sein Passwort eingeben. Das eingegebene Passwort wird dem in der Datenbank hinterlegtem Passwort verglichen und auf Übereinstimmung geprüft. Es werden einzelne Buchstaben eingelesen, um den Dialog dynamisch zu gestalten und das nutzen der Löschen-Taste zu ermöglichen, um einzelne Buchstaben löschen zu können. So wird ein möglichst nutzerfreundliches Erlebnis garantiert.

4.2.1.5 handleHelpInput()

```
void handleHelpInput (
            int iUserInput )
```

Wird aufgerufen, sobald der Nutzer versucht etwas einzugeben, während er sich in den Spielregeln befindet. "z" bringt den Spieler zurück zum Menü oder zum laufenden Spiel; ausgehend davon, ob es noch ein ungelöstes Sudokufeld gibt.

1. Parameter: Eingabe des Nutzers

4.2.1.6 handleInGameInput()

```
void handleInGameInput (
            int iUserInput )
```

Wird aufgerufen, wenn der Nutzer eine Eingabe tätigt, wenn er sich gerade im laufenden Spiel befindet. Ruft nötige Funktionen auf, um die Eingabe des Nutzers der Legende entsprechend zu verarbeiten.

1. Parameter: Eingabe des Nutzers

4.2.1.7 handleMenuInput()

```
void handleMenuInput (
            int iUserInput )
```

Wird aufgerufen, wenn der Spieler eine Eingabe im Hauptmenü tätigt. Verarbeiet die Eingabe des Nutzers und ruft die nötigen Funtkionen auf, um den Nutzer zu den nächsten "Screens" weiterzuleiten.

1. Parameter: Eingabe des Nutzers

4.2.1.8 handleSetMarkInput()

```
void handleSetMarkInput (
            int iUserInput )
```

4.2.1.9 handleSetPasswordInput()

```
void handleSetPasswordInput ( )
```

Wird aufgerufen, wenn sich ein neuer Nutzer anmeldet und ein Passwort setzen möchte. Dieses wird nach einer Bestätigung durch das Drücken der Entertaste mit dem Nutzernamen und der UserID in der Datenbank hinterlegt (bzw. die Funktion, die dies erledigt wird aufgerufen). Es werden einzelne Buchstaben eingelesen, um den Dialog dynamisch zu gestalten und das nutzen der Löschen-Taste zu ermöglichen, um einzelne Buchstaben löschen zu können. So wird ein möglichst nutzerfreundliches Erlebnis garantiert.

4.2.1.10 handleSolvedGameInput()

```
void handleSolvedGameInput (
            int iUserInput )
```

Wird aufgerufen, wenn der Nutzer eine Eingabe im "Winscreen" tätigt. Eingabe wird sachgerecht behandelt.

1. Parameter: Eingabe des Nutzers

4.2.1.11 handleUserNameInput()

```
void handleUserNameInput ( )
```

Wird aufgerufen, wenn der Nutzer sich im Dialog befindet, in dem er seinen Namen angeben soll. Ermöglicht zudem das Überspringen dieses Schrittes. Der Spieler ist dann unter "anonym" angemeldet. Es werden einzelne Buchstaben eingelesen, um den Dialog dynamisch zu gestalten und das nutzen der Löschen-Taste zu ermöglichen, um einzelne Buchstaben löschen zu können. So wird ein möglichst nutzerfreundliches Erlebnis garantiert.

4.2.1.12 setConfig()

```
void setConfig ( )
```

Wird aufgerufen, sobald der Nutzer versucht etwas einzugeben, während er sich in den Spielregeln befindet. "z" bringt den Spieler zurück zum Menü oder zum laufenden Spiel; ausgehend davon, ob es noch ein ungelöstes Sudokufeld gibt.

1. Parameter: Eingabe des Nutzers Setzt bzw. initialisiert die erlaubte Anzahl, wie oft der Spieler die Tipp- und Hilfefunktion nutzen darf. Setzt ausßerdem die bereits genutzten Funktionen zurück.

4.3 headers/core/view.h File Reference

```
#include "../shared/shared.h"
```

Functions

- void renderMarkModeMessage ()
- void renderUsernameDialog (char *pcUsername)
- void renderMenu ()
- void renderDBestScoreDialog ()
- void renderDetails (struct sScore *scores, int difficulty)
- void renderCourt (int gameData[ ][9], int userCells[ ][9], int x_coordinate, int y_coordinate)
- void renderInfoBox (char *username, int *score, int iCurrentUserBestScore, int _difficulty, int remaining)
- void renderGameMenu ()
- void renderSolvedGame (int solvedAutomatic, int anzahlDerTipps, int anzahlDerHilfe)
- void renderDifficultyDialog ()
- void renderHelpDialog ()
- int lenHelper (int input)

  Function to get the length of integers (aka strlen for strings)
- int getRemainingCells (int iArray[ ][9])
- void print_list (struct sScore *head, int iDifficulty)
- void initColors ()
- void printGameMessage ()
- void printColoredString (char text[ ], char color[ ], int newLine)
- void printColoredNumber (int number, char *color, int newLine)
- void setPrintingColor (char *color)
- void printStartOfLine ()
- void printEndOfLine ()
- void printTableLine ()
- void printEndOfTable ()
- void printEmptyTableLine ()
- void renderLoadingBar (int x)
- void clear_output ()
- void renderNotesBox (int x, int y)
- void renderSetPassword ()
- void renderEnterPassword ()

4.3.1 Function Documentation

4.3.1.1 clear_output()

```
void clear_output ( )
```

Löscht den aktuellen Konsoleninhalt. Überprüft zunächst das laufende Betriebssystem, um einen ordnungsgemäßen Ablauf zu gewährleisten.

4.3.1.2 getRemainingCells()

```
int getRemainingCells (
            int iArray[][9] )
```

Ermittelt die Anzahl der Zellen, die noch befüllt werden müssen.

1. Parameter: sichtbares Spielfeld

4.3.1.3 initColors()

```
void initColors ( )
```

4.3.1.4 lenHelper()

```
int lenHelper (
            int input )
```

Function to get the length of integers (aka strlen for strings)

Parameters

input	an integer to get length of

Returns

the length of the given integer

It return the length of integer number (for optimal printing)

4.3.1.5 print_list()

```
void print_list (
            struct sScore * head,
            int iDifficulty )
```

Gibt die Bestenliste (TOP 10) in Abhängigkeit von der Zeit und dem ausgewählten Schwierigkeitsgrad aus.

1. Parameter: Zeiger auf Struktur, in der die Bestscores gespeichert sind
2. Parameter: ausgewählter Schwierigkeitsgrad

4.3.1.6 printColoredNumber()

```
void printColoredNumber (
            int number,
            char * color,
            int newLine )
```

Dient zur formatierten Ausgabe einer farbigen Zahl im Spielfeld

1. Parameter: Zahl, die gedruckt werden soll

1. Parameter: Zahl, die gedruckt werden soll
2. Parameter: Farbe, in der die Zahl gedruckt werden soll

1. Parameter: Zeiger auf die Variable mit dem Wert für die Farbe, in der die Zahl gedruckt werden soll
2. Parameter: Soll danach eine neue Zeile folgen? (1 = Ja, 0 = Nein)

1. Parameter: Soll danach eine neue Zeile folgen? (1 = Ja, 0 = Nein)

4.3.1.7 printColoredString()

```
void printColoredString (
            char text[],
            char color[],
            int newLine )
```

Dient zur formatierten Ausgabe eines vorgegeben Strings

1. Parameter: String, der gedruckt werden soll
2. Parameter: Farbe, in der der String gedruckt werden soll
3. Parameter: Soll danach eine neue Zeile folgen? (1 = Ja, 0 = Nein)

4.3.1.8 printEmptyTableLine()

```
void printEmptyTableLine ( )
```

4.3.1.9 printEndOfLine()

```
void printEndOfLine ( )
```

Ausgabe des Endes einer Zeile, die zu einer Box mit Informationen gehört (Beispiel: Spielregeln).

4.3.1.10 printEndOfTable()

```
void printEndOfTable ( )
```

Ausgabe der Zeile, die das Ende einer Box mit Informationen darstellt (Beispiel: Spielregeln).

4.3.1.11 printGameMessage()

```
void printGameMessage ( )
```

Gibt die kontextbezogene Nachricht an den Spieler aus (z.B. "Passwort falsch")

4.3.1.12 printStartOfLine()

```
void printStartOfLine ( )
```

Ermittelt die Anzahl der Zellen, die noch befüllt werden müssen.

1. Parameter: sichtbares Spielfeld

4.3.1.13 printTableLine()

```
void printTableLine ( )
```

Ausgabe einer Zeile, die zu einer Box mit Informationen gehört (Beispiel: Spielregeln).

1. Parameter: Text, der gedruckt werden sollen

4.3.1.14 renderCourt()

```
void renderCourt (
            int gameData[][9],
            int userCells[][9],
            int x_coordinate,
            int y_coordinate )
```

Gibt das "sichtbare" Spielfeld in der Konsole aus.

1. Parameter: sichtbares Spielfeld
2. Parameter: Array, in dem gespeichert ist, welche Zellen vom Nutzer befüllt werden müssen
3. Parameter: x-Koordinate, an der sich der Cursor des Spielers befindet
4. Parameter: y-Koordinate, an der sich der Cursor des Spielers befindet

4.3.1.15 renderDBestScoreDialog()

```
void renderDBestScoreDialog ( )
```

Stellt das Grundgerüst für die Ausgabe der Bestenliste bereit bzw. initialisiert diese Ausgabe initialisiert diese Ausgabe

1. Parameter: Struktur, in der die Bestscores gespeichert sind

1. Parameter: Zeiger auf Struktur, in der die Bestscores gespeichert sind
2. Parameter: ausgewählter Schwierigkeitsgrad

1. Parameter: ausgewählter Schwierigkeitsgrad

4.3.1.16 renderDetails()

```
void renderDetails (
            struct sScore * scores,
            int difficulty )
```

Stellt das Grundgerüst für die Ausgabe der Bestenliste bereit bzw. initialisiert diese Ausgabe

1. Parameter: Struktur, in der die Bestscores gespeichert sind
2. Parameter: ausgewählter Schwierigkeitsgrad

4.3.1.17 renderDifficultyDialog()

```
void renderDifficultyDialog ( )
```

Ausgabe des Dialoges, in dem der Spieler den Schwierigkeitsgrad für sein Spiel wählen kann.

4.3.1.18 renderEnterPassword()

```
void renderEnterPassword ( )
```

Gibt den Dialog aus, in dem ein bereits registrierter Spieler sein Passwort eingeben kann.

4.3.1.19 renderGameMenu()

```
void renderGameMenu ( )
```

Gibt die Legende aus, die dem Spieler aufzeigt, welche Tasten er drücken kann, um mit dem Spiel zu interagieren.

4.3.1.20 renderHelpDialog()

```
void renderHelpDialog ( )
```

Gibt die Spielregeln in der Konsole aus.

4.3.1.21 renderInfoBox()

```
void renderInfoBox (
            char * username,
            int * score,
            int iCurrentUserBestScore,
            int _difficulty,
            int remaining )
```

Gibt die Infobox aus, die sich über dem Spielfeld zur Spielzeit befindet und alle wichtigen Informationen, wie die aktuell schon benötigte Zeit, und alle wichtigen Informationen, wie die aktuell schon benötigte Zeit, enthält. enthält.

1. Parameter: Nutzernamen, des aktuell spielenden Spielers

1. Parameter: Zeiger auf die Variable, die den Nutzernamen des aktuell spielenden Spielers enthält
2. Parameter: Highscore im aktuellen Schwierigkeitsgrad

1. Parameter: Zeiger auf die Variable, die den Highscore im aktuellen Schwierigkeitsgrad enthält
2. Parameter: aktuell ausgewählter Schwierigkeitsgrad

1. Parameter: aktuell ausgewählter Schwierigkeitsgrad
2. Parameter: Anzahl der Zellen, die der Spieler noch ausfüllen muss

1. Parameter: Anzahl der Zellen, die der Spieler noch ausfüllen muss

4.3.1.22 renderLoadingBar()

```
void renderLoadingBar (
            int x )
```

4.3.1.23 renderMarkModeMessage()

```
void renderMarkModeMessage ( )
```

Ausgabe des Hinweises auf den "Markieren-Modus"

4.3.1.24 renderMenu()

```
void renderMenu ( )
```

Gibt die Bestenliste (TOP 10) in Abhängigkeit von der Zeit und dem ausgewählten Schwierigkeitsgrad aus. aus.

1. Parameter: Struktur, in der die Bestscores gespeichert sind

1. Parameter: Zeiger auf Struktur, in der die Bestscores gespeichert sind
2. Parameter: ausgewählter Schwierigkeitsgrad

1. Parameter: ausgewählter Schwierigkeitsgrad

4.3.1.25 renderNotesBox()

```
void renderNotesBox (
            int x,
            int y )
```

Augabe - sofern vorhanden - der Notizen des Spielers (bzw. Tipps).

4.3.1.26 renderSetPassword()

```
void renderSetPassword ( )
```

Gibt den Dialog aus, in dem ein neuer Spieler sein Passwort setzen kann.

4.3.1.27 renderSolvedGame()

```
void renderSolvedGame (
            int solvedAutomatic,
            int anzahlDerTipps,
            int anzahlDerHilfe )
```

Gibt den "Winscreen" in der Konsole aus, wenn das Sudoku vollständig gelöst wurde.

1. Parameter: Wurde das Spiel aufgelöst oder hat der Spieler es selbst gelöst (1 = es wurde aufglöst, 0 = Spieler hat es gelöst)
2. Parameter: Anzahl der benutzten Tipps
3. Parameter: Anzahl der benutzten Zelllösugen

4.3.1.28 renderUsernameDialog()

```
void renderUsernameDialog (
            char * pcUsername )
```

Gibt den Dialog, in dem der Spieler nach seinem Namen gefragt wird aus. aus.

1. Parameter: bisheriger Benutzername (Zeichen werden einzeln

1. Parameter: Zeiger auf die Variable mit dem Wert des bisherigern entgegengenommen, um eine nutzerfreundliche Obfläche / Eingabe Benutzernamens (Zeichen werden einzeln entgegengenommen, um eine nutzerfreundliche Obfläche / Eingabe und das Nutzen der Löschen-Taste zu ermöglichen) und das Nutzen der Löschen-Taste zu ermöglichen)

4.3.1.29 setPrintingColor()

```
void setPrintingColor (
            char * color )
```

Setzt die Farbe, in der ab sofort in die Konsole geschrieben werden soll

1. Parameter: Farbe

1. Parameter: Zeiger auf die Variable mit dem Wert für die Farbe

4.4 headers/getch.h File Reference

Functions

- int cbreak (int fd)
- int getch (void)

4.4.1 Function Documentation

4.4.1.1 cbreak()

```
int cbreak (
            int fd )
```

4.4.1.2 getch()

```
int getch (
            void  )
```

4.5 headers/services/connection.h File Reference

```
#include "../../libs/sqlite3.h"
```

Variables

- sqlite3 * psqlConnection
  Verbindung zur Datenbank.

4.5.1 Variable Documentation

4.5.1.1 psqlConnection

```
sqlite3* psqlConnection
```

Verbindung zur Datenbank.

4.6 headers/services/score_service.h File Reference

```
#include "../../libs/sqlite3.h"
```

Data Structures

- struct sScore

Typedefs

- typedef struct sScore score

Functions

- int insertScore (int *piUserID, int iScore, int iDifficulty)
  Inserts score in the database.
- void getScores (struct sScore *scores)
- int getBestScoreByUserID (int userID)
- int bestScoreCallback (void *pvBestScore, int iArgc, char **ppcArgv, char **ppcAzColName)
- int getBestScore (int *piBestScore, int iDifficulty)

4.6.1 Typedef Documentation

4.6.1.1 score

```
typedef struct sScore score
```

4.6.2 Function Documentation

4.6.2.1 bestScoreCallback()

```
int bestScoreCallback (
            void * pvBestScore,
            int iArgc,
            char ** ppcArgv,
            char ** ppcAzColName )
```

4.6.2.2 getBestScore()

```
int getBestScore (
            int * piBestScore,
            int iDifficulty )
```

Erfragt den Highscore aller Spieler im aktuellen Schwierigkeitsgrad aus der Datenbank.

1. Parameter: Zeiger auf die Variable, in die der Highscore geschrieben werden soll
2. Parameter: aktueller Schwierigkeitsgrad

4.6.2.3 getBestScoreByUserID()

```
int getBestScoreByUserID (
            int userID )
```

Erfragt den Highscore des aktuellen Nutzers aus der Datenbank.

1. Parameter: UserID des aktuellen Nutzers

4.6.2.4 getScores()

```
void getScores (
            struct sScore * scores )
```

Abfrage der Highscores inklusive der zugehörigen Spielernamen / UserIDs

1. Parameter: Zeiger auf die Struktur, in der die abgefragten Daten ge- speichert werden sollen

4.6.2.5 insertScore()

```
int insertScore (
            int * piUserID,
            int iScore,
            int iDifficulty )
```

Inserts score in the database.

Parameters

username[]	
iScore	
_time	
iDifficulty	

Returns

Returns the id of the last user if insered, otherwise -1.

4.7 headers/services/user_service.h File Reference

```
#include "../../libs/sqlite3.h"
```

Functions

- int registerUser (char cUsername[ ], char cPassword[ ], int ∗piNewUserId)
- void loginUser (char cUsername[ ], char cPassword[ ], int ∗piId)
- int createUserTable ()
- int createScoreTable ()
- int getUserIdCallback (void ∗userID, int argc, char ∗∗argv, char ∗∗azColName)
- void getUserID (char cUsername[8], int ∗piUserID)

4.7.1 Function Documentation

4.7.1.1 createScoreTable()

```
int createScoreTable ( )
```

Erstellt die Tabelle für die Daten des Nutzers (NutzerID, Name und Passwort).

4.7.1.2 createUserTable()

```
int createUserTable ( )
```

Erstellt die Tabelle für die Daten des Scores (id, time, userId, difficulty).

4.7.1.3 getUserID()

```
void getUserID (
            char cUsername[8],
            int * piUserID )
```

Erfragt die zugehörige UserID zu einem bestimmten Nutzernamen aus der Datenbank.

1. Parameter: Nutzername des Spielers, dessen UserID erfragt werden soll
2. Parameter: Zeiger auf die zu befüllende Variable UserID

4.7.1.4 getUserIdCallback()

```
int getUserIdCallback (
            void * userID,
            int argc,
            char ** argv,
            char ** azColName )
```

4.7.1.5 loginUser()

```
void loginUser (
            char cUsername[],
            char cPassword[],
            int * piId )
```

Überprüft mit Hilfe der Daten aus der Datenbank, ob das eingegebene Passwort richtig ist.

1. Parameter: eingegebener Nutzername
2. Parameter: eingegebenes Passwort
3. Parameter: Zeiger auf die zu befüllende Variable UserID

4.7.1.6 registerUser()

```
int registerUser (
            char cUsername[],
            char cPassword[],
            int * piNewUserId )
```

Registriert einen neuen Nutzer mit seinem von ihm gesetzten Passwort in der Datenbank.

1. Parameter: eingegebener Nutzername
2. Parameter: eingegebenes Passwort
3. Parameter: Zeiger auf die zu befüllende Variable UserID

4.8 headers/shared/shared.h File Reference

Macros

- #define MAX_MARKS 3

Enumerations

- enum DIFFICULTY { EASY = 5, MEDIUM = 7, HARD = 8 }
  Difficulty enum.
- enum ACTIONS {
  TIMER_STATE = 0, TIMER_START = 1, TIMER_PAUSE = 2, TIPP_USED = 3,
  HELP_USED, RESET_TIMER }
- enum POSITIONS {
  MENU = 0, IN_GAME = 1, DIFFICULTY_DIALOG = 2, DETAILS = 3,
  HELP = 4, USER_NAME = 5, DETAILS_DIALOG = 6, SOLVED_GAME = 7,
  SET_MARK = 8, SET_PASSWORD = 9, ENTER_PASSWORD = 10 }
  Positions enum.
- enum GRID_STATUS { FILLED = 1, NOT_FILLED = 0 }
  Grid-Status enum.
- enum ARROWS {
  UP = 72, DOWN = 80, LEFT = 75, RIGHT = 77,
  UP_LINUX = 65, DOWN_LINUX = 66, RIGHT_LINUX = 67, LEFT_LINUX = 68 }
  Arrows enum.

4.8.1 Macro Definition Documentation

4.8.1.1 MAX_MARKS

```
#define MAX_MARKS 3
```

4.8.2 Enumeration Type Documentation

4.8.2.1 ACTIONS

```
enum ACTIONS
```

Enumerator

TIMER_STATE	
TIMER_START	///< Aktion um Timer Status zu holen Aktion um Timer zu starten
TIMER_PAUSE	Aktion um Timer zu pausieren.
TIPP_USED	Aktion wenn Tipp benutzt wird.
HELP_USED	Aktion wenn Hilfe benutzt wird.
RESET_TIMER	Aktion um den Timmer zurückzusetzen.

4.8.2.2 ARROWS

`enum ARROWS`

Arrows enum.

Enumerator

UP	UP.
DOWN	DOWN.
LEFT	LEFT.
RIGHT	RIGHT.
UP_LINUX	RIGHT ON UNIX.
DOWN_LINUX	DOWN ON UNIX.
RIGHT_LINUX	DOWN ON UNIX.
LEFT_LINUX	DOWN ON UNIX.

4.8.2.3 DIFFICULTY

`enum DIFFICULTY`

Difficulty enum.

< max. deleted cells per box

Enumerator

EASY	Repräsentiert Schwierigkeitsstufe einfach.
MEDIUM	Repräsentiert Schwierigkeitsstufe mittel.
HARD	Repräsentiert Schwierigkeitsstufe schwer.

4.8.2.4 GRID_STATUS

`enum GRID_STATUS`

Grid-Status enum.

Enumerator

FILLED	Beschreibt den Staus von Sudoku.
NOT_FILLED	Beschreibt den Staus von Sudoku.

4.8.2.5 POSITIONS

`enum POSITIONS`

Positions enum.

Enumerator

MENU	MENU.
IN_GAME	IN_GAME.
DIFFICULTY_DIALOG	DIFFICULTY_DIALOG.
DETAILS	DETAILS.
HELP	HELP.
USER_NAME	USER_NAME.
DETAILS_DIALOG	DETAILS_DIALOG.
SOLVED_GAME	SOLVED_GAME.
SET_MARK	SET_MARK.
SET_PASSWORD	SET_PASSWORD.
ENTER_PASSWORD	ENTER_PASSWORD.