SUDOKU

Generado por Doxygen 1.8.4

Índice general

Capítulo 1

Documentación de las clases

1.1. Referencia de la Clase Clock

`#include <clock.h>`

Diagrama de herencias de Clock

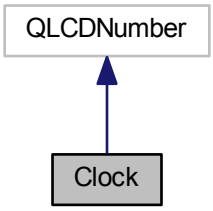
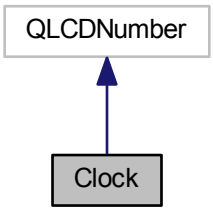


Diagrama de colaboración para Clock:



Slots públicos

- void **startTimer** ()

- void **stopTimer** ()
- int **getTimeScore** ()

Métodos públicos

- **Clock** (QWidget ∗parent=0)
- ∼**Clock** ()

Atributos públicos

- int **timeAdded**

Slots privados

- void **showTime** ()
- void **initGui** ()

Atributos privados

- Ui::Clock ∗ **ui**
- QTime **initialTime**
- QTimer ∗ **timer**

1.1.1. Documentación del constructor y destructor

1.1.1.1. Clock::Clock (QWidget ∗ *parent* = 0) [explicit]

```
6                               :
7     QLCDNumber(parent),
8     ui(new Ui::Clock)
9 {
10     ui->setupUi(this);
11     initGui();
12 }
```

1.1.1.2. Clock::∼Clock ( )

```
15 {
16     delete timer;
17     delete ui;
18 }
```

1.1.2. Documentación de las funciones miembro

1.1.2.1. int Clock::getTimeScore ( ) [slot]

```
54 {
55     QTime time = QTime::currentTime();
56     time = time.addSecs(-initialTime.hour() * 3600 - initialTime.minute() * 60 -
      initialTime.second());
57     return time.hour() * 3600 + time.minute() * 60 + time.second();
58 }
```

1.1.2.2. void Clock::initGui ( ) [private],[slot]

```
21 {
22     setSegmentStyle(Filled);
23     setWindowTitle(tr("Timer"));
24     resize(150, 60);
25 }
```

1.1.2.3. void Clock::showTime ( ) [private],[slot]

```
28 {
29     QTime time = QTime::currentTime();
30     time = time.addSecs(-initialTime.hour() * 3600 - initialTime.minute() * 60 -
      initialTime.second());
31     time = time.addSecs(timeAdded);
32     QString text = time.toString("hh:mm:ss");
33
34     display(text);
35 }
```

1.1.2.4. void Clock::startTimer ( ) [slot]

```
38 {
39     timer = new QTimer(this);
40     connect(timer, SIGNAL(timeout()), this, SLOT(showTime()));
41     timer->start(1000);
42
43     initialTime = QTime::currentTime();
44     display("00:00");
45     showTime();
46 }
```

1.1.2.5. void Clock::stopTimer ( ) [slot]

```
49 {
50     timer->stop();
51 }
```

1.1.3. Documentación de los datos miembro

1.1.3.1. QTime Clock::initialTime [private]

1.1.3.2. int Clock::timeAdded

1.1.3.3. QTimer* Clock::timer [private]

1.1.3.4. Ui::Clock* Clock::ui [private]

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- I:/Sudoku-master/Sudoku/Functional Version/**clock.h**
- I:/Sudoku-master/Sudoku/Functional Version/**clock.cpp**

1.2. Referencia de la Clase QPushButtonGrid

```
#include <qpushbuttongrid.h>
```

Diagrama de herencias de QPushButtonGrid

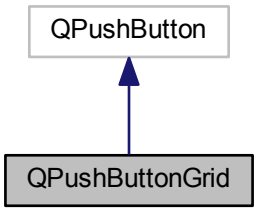
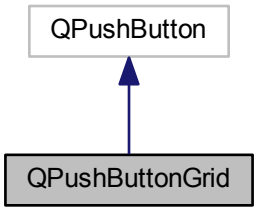


Diagrama de colaboración para QPushButtonGrid:



Métodos públicos

- **QPushButtonGrid** (QPushButton ∗parent=0)
- ∼**QPushButtonGrid** ()

Atributos públicos

- int **row**
- int **column**
- int **realNumber**
- int **assignedNumber**
- int **color**
- bool **isConstant**

Atributos privados

- Ui::QPushButtonGrid ∗ **ui**

1.2.1. Documentación del constructor y destructor

1.2.1.1. QPushButtonGrid::QPushButtonGrid (QPushButton * *parent* = 0) [explicit]

```
4                                                   :
5     QPushButton(parent),
6     ui(new Ui::QPushButtonGrid)
7 {
8     ui->setupUi(this);
9 }
```

1.2.1.2. QPushButtonGrid::~QPushButtonGrid ( )

```
12 {
13     delete ui;
14 }
```

1.2.2. Documentación de los datos miembro

1.2.2.1. int QPushButtonGrid::assignedNumber

1.2.2.2. int QPushButtonGrid::color

1.2.2.3. int QPushButtonGrid::column

1.2.2.4. bool QPushButtonGrid::isConstant

1.2.2.5. int QPushButtonGrid::realNumber

1.2.2.6. int QPushButtonGrid::row

1.2.2.7. Ui::QPushButtonGrid* QPushButtonGrid::ui [private]

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- I:/Sudoku-master/Sudoku/Functional Version/**qpushbuttongrid.h**
- I:/Sudoku-master/Sudoku/Functional Version/**qpushbuttongrid.cpp**

1.3. Referencia de la Clase Sudoku

`#include <sudoku.h>`

Diagrama de herencias de Sudoku

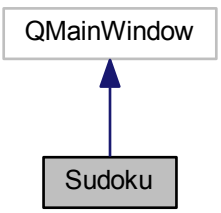
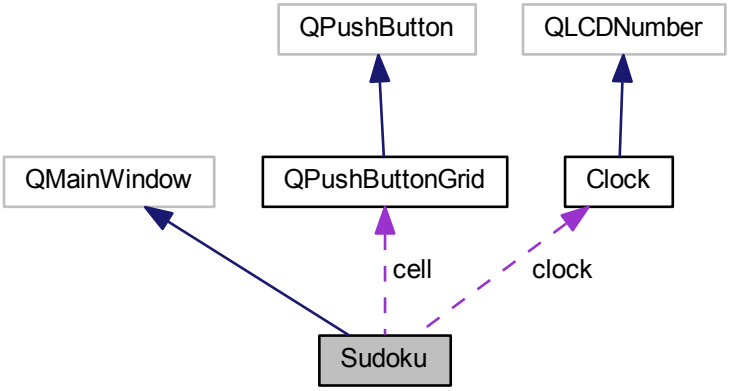


Diagrama de colaboración para Sudoku:



Métodos públicos

- **Sudoku** (QWidget ∗parent=0)
- ∼**Sudoku** ()

Slots privados

- void **initGui** ()
- bool **validateCellAtPointWithNumber** (int row, int column, int value, int counter)
- bool **numberAssigner** (int counter, int row, int column, int value)
- void **sudokuGenerator** (int row)
- bool **sudokuBlankChecker** ()
- void **sudokuErrorWiper** ()
- bool **isSudokuComplete** ()

  *Funcion Booleana **Sudoku** (p. 5) Completo.*
- void **number_clicked** ()
- void **cell_clicked** ()
- void **on_nullButton_clicked** ()
- void **validateCell** ()
- void **on_actionNew_triggered** ()
- void **on_actionOpen_triggered** ()
- void **on_actionSave_triggered** ()
- void **on_actionClose_triggered** ()
- void **on_actionQuit_triggered** ()
- void **validateWindow** ()
- void **invalidateWindow** ()
- void **on_pushButton_clicked** ()
- void **on_normalButton_clicked** ()

  Game Mode Normal (Modo de Juego)
- void **on_hintButton_clicked** ()

  Game Mode Hint (Modo de Juego Hint)
- void **on_clueButton_clicked** ()

Game Mode Clue(Modo de Juego)

- void **on_actionEasy_triggered** ()

  Nivel FACIL.
- void **on_actionMedium_triggered** ()

  Nivel DIFICIL.
- void **on_actionHard_triggered** ()

  Nivel DIFICIL.

Atributos privados

- Ui::Sudoku ∗ **ui**
- **QPushButtonGrid** ∗ **cell** [9][9]
- QPushButton ∗ **number** [9]
- QString **crypt**
- **Clock** ∗ **clock**
- int **coorX**
- int **coorY**
- bool **isCellSelected**

1.3.1. Documentación del constructor y destructor

1.3.1.1. Sudoku::Sudoku (QWidget ∗ *parent* = 0) [explicit]

```
19                                  :
20     QMainWindow(parent),
21     ui(new Ui::Sudoku)
22 {
23     srand((unsigned)time(NULL));
24     ui->setupUi(this);
25     initGui();
26 }
```

1.3.1.2. Sudoku::∼Sudoku ( )

```
29 {
30     delete ui;
31     for (int i = 0; i < 9; i++)
32     {
33         for(int j = 0; j < 9; j++)
34         {
35             delete cell[i][j];
36         }
37     }
38     for (int i = 0; i < 9; i++)
39     {
40         delete number[i];
41
42     }
43     delete clock;
44 }
```

1.3.2. Documentación de las funciones miembro

1.3.2.1. void Sudoku::cell_clicked ( ) [private], [slot]

```
364 {
365     if (!isCellSelected)
366     {
367         isCellSelected = true;
368     }
369     else
370     {
371         if(cell[coorX][coorY]->color == 0)
```

```
372         {
373             cell[coorX][coorY]->setStyleSheet("QPushButtonGrid { background-color: darkBlue; }");
374         }
375         else
376         {
377             cell[coorX][coorY]->setStyleSheet("QPushButtonGrid { background-color: magenta; }");
378         }
379     }
380     QPushButtonGrid *cell = (QPushButtonGrid *)sender();
381     coorX = cell->row;
382     coorY = cell->column;
383
384     if (cell->realNumber == cell->assignedNumber)
385     {
386         ui->radioButton->setChecked(true);
387     }
388     else
389     {
390         ui->radioButton->setChecked(false);
391     }
392
393     for (int i = 0; i < 9; i++)
394     {
395         number[i]->setEnabled(true);
396     }
397
398     if(ui->hintButton->isChecked())
399     {
400         validateCell();
401     }
402
403
404     cell->setStyleSheet("QPushButtonGrid { background-color: darkRed; }");
405 }
```

1.3.2.2. void Sudoku::initGui ( ) [private],[slot]

```
47 {
48     coorX = -1;
49     coorY = -1;
50     bool paintFlag = false;
51
52     this->setStyleSheet("background-color: white;");
53
54     for (int i = 0; i < 9; i++)
55     {
56         for (int j = 0; j < 9; j++)
57         {
58             cell[i][j] = new QPushButtonGrid();
59             cell[i][j]->row = i;
60             cell[i][j]->column = j;
61             cell[i][j]->realNumber = 0;
62             cell[i][j]->assignedNumber = 0;
63             cell[i][j]->isConstant = false;
64             if(paintFlag)
65             {
66                 cell[i][j]->color = 0;
67
68                 cell[i][j]->setStyleSheet("QPushButtonGrid { background-color: darkBlue; }");
69             }
70             else
71             {
72                 cell[i][j]->color = 1;
73                 cell[i][j]->setStyleSheet("QPushButtonGrid { background-color: magenta; }");
74             }
75
76             connect(cell[i][j], &QPushButtonGrid::clicked, this, &
    Sudoku::cell_clicked);
77             ui->gridLayout->addWidget(cell[i][j], i, j);
78             if (j % 3 == 2)
79             {
80                 paintFlag = !paintFlag;
81             }
82         }
83         if(i % 3 != 2)
84         {
85             paintFlag = !paintFlag;
86         }
87     }
88
89     for (int i = 0; i < 9; i++)
90     {
91         number[i] = new QPushButton(QString("%1").arg(i + 1));
92         connect(number[i], &QPushButton::clicked, this, &Sudoku::number_clicked);
```

```
93          ui->numberPad->addWidget(number[i], i / 3, i % 3);
94          number[i]->setStyleSheet("QPushButton { background-color: darkCyan; }");
95      }
96
97      ui->nullButton->setText("Erase");
98      ui->nullButton->setEnabled(false);
99      ui->nullButton->setStyleSheet("QPushButton { background-color: darkCyan; }");
100      isCellSelected = false;
101
102      invalidateWindow();
103
104
105      clock = new Clock();
106      ui->verticalLayout->addWidget(clock);
107      clock->setHidden(true);
108
109  }
```

1.3.2.3. void Sudoku::invalidateWindow ( ) [private],[slot]

```
791 {
792      coorX = -1;
793      coorY = -1;
794      for(int i = 0; i < 9; i++)
795      {
796          for(int j = 0; j < 9; j++)
797          {
798              cell[i][j]->setEnabled(false);
799
800              if(cell[i][j]->color == 0)
801              {
802                  cell[i][j]->setStyleSheet("QPushButtonGrid { background-color: darkBlue; }");
803              }
804              else
805              {
806                  cell[i][j]->setStyleSheet("QPushButtonGrid { background-color: magenta; }");
807              }
808          }
809          number[i]->setEnabled(false);
810      }
811      ui->nullButton->setEnabled(false);
812      ui->radioButton->setChecked(false);
813      isCellSelected = false;
814
815      ui->groupBox_2->setEnabled(false);
816
817  }
```

1.3.2.4. bool Sudoku::isSudokuComplete ( ) [private],[slot]

Funcion Booleana **Sudoku** (p. 5) Completo.

Esta funcion nos dice si ya todas las casillas del sudoku estan llenas @return true/false

```
839 {
840      for(int i = 0; i < 9; i++)
841      {
842          for(int j = 0; j < 9; j++)
843          {
844              if (cell[i][j]->realNumber != cell[i][j]->assignedNumber)
845              {
846                  return false;
847              }
848          }
849      }
850      return true;
851  }
```

1.3.2.5. void Sudoku::number_clicked ( ) [private],[slot]

```
257 {
258      if (isCellSelected) {
259          QPushButton *numberButton = (QPushButton *)sender();
260          number[cell[coorX][coorY]->assignedNumber - 1]->setEnabled(true);
```

```
261         cell[coorX][coorY]->setText(numberButton->text());
262         cell[coorX][coorY]->assignedNumber = numberButton->text().toInt();
263 
264         if (cell[coorX][coorY]->realNumber == cell[coorX][coorY]->assignedNumber)
265         {
266             ui->radioButton->setChecked(true);
267         }
268         else
269         {
270             ui->radioButton->setChecked(false);
271         }
272         if(ui->hintButton->isChecked())
273         {
274             number[cell[coorX][coorY]->assignedNumber - 1]->setEnabled(false);
275         }
276 
277         ui->nullButton->setEnabled(true);
278 
279         if(isSudokuComplete())
280         {
281             invalidateWindow();
282             //int timer = clock->getTimeScore(); //////////
283             clock->stopTimer();
284             QDialog saveName;
285 
286 
287             saveName.exec();
288         }
289     }
290     else if (ui->clueButton->isChecked())
291     {
292         invalidateWindow();
293         //validateWindow();
294         ui->groupBox_2->setEnabled(true);
295         QPushButton *numberButton = (QPushButton *)sender();
296         for (int k = 0; k < 9; k++)
297         {
298             for (int l = 0; l < 9; l++)
299             {
300                 cell[k][l]->setStyleSheet("background-color: yellow");
301             }
302 
303         }
304         for (int k = 0; k < 9; k++)
305         {
306             for (int l = 0; l < 9; l++)
307             {
308                 if(cell[k][l]->assignedNumber)
309                 {
310                     if(cell[k][l]->assignedNumber == numberButton->text().toInt())
311                     {
312                         int centerX = k / 3;
313                         int centerY = l / 3;
314 
315                         for (int i = 0; i < 9; i++)
316                         {
317                             for (int j = 0; j < 9; j++)
318                             {
319 
320                                 if  ((i == k || j == l || (i / 3 == centerX && j / 3 == centerY)))
321                                 {
322 
323                                         if (!cell[i][j]->assignedNumber)
324                                         {
325                                             if(cell[i][j]->color == 0)
326                                             {
327                                                 cell[i][j]->setStyleSheet("QPushButtonGrid {
    background-color: darkBlue; }");
328                                             }
329                                             else
330                                             {
331                                                 cell[i][j]->setStyleSheet("QPushButtonGrid {
    background-color: magenta; }");
332                                             }
333                                         }
334                                     }
335 
336                             }
337 
338                         }
339                     }
340                     if(cell[k][l]->color == 0)
341                     {
342                         cell[k][l]->setStyleSheet("QPushButtonGrid { background-color: darkBlue; }");
343                     }
344                     else
345                     {
```

```
346                         cell[k][l]->setStyleSheet("QPushButtonGrid { background-color: magenta; }");
347                     }
348                 }
349
350
351
352             }
353
354         }
355         for (int i = 0; i < 9; i++)
356         {
357             number[i]->setEnabled(true);
358         }
359     }
360 }
```

1.3.2.6. bool Sudoku::numberAssigner (int *counter,* int *row,* int *column,* int *value*) [private],[slot]

```
165 {
166     if (counter > column)
167     {
168         return false;
169     }
170     else {
171         if ((!counter && validateCellAtPointWithNumber(row, column, value, 0)) || (counter &&
      validateCellAtPointWithNumber(row, column, cell[row][column - counter]->realNumber, counter) &&
      validateCellAtPointWithNumber(row, column - counter, value, 0)))
172         {
173             if (!counter)
174             {
175                 cell[row][column]->realNumber = value;
176                 //cell[row][column]->setText(QString("%1").arg(value));
177             }
178             else
179             {
180                 cell[row][column]->realNumber = cell[row][column - counter]->
      realNumber;
181                 cell[row][column - counter]->realNumber = value;
182
183             }
184             return true;
185         }
186         else
187         {
188             counter++;
189             return numberAssigner(counter, row, column, value);
190         }
191     }
192 }
```

1.3.2.7. void Sudoku::on_actionClose_triggered ( ) [private],[slot]

```
821 {
822     sudokuErrorWiper();
823     invalidateWindow();
824     clock->setHidden(true);
825 }
```

1.3.2.8. void Sudoku::on_actionEasy_triggered ( ) [private],[slot]

Nivel FACIL.

Nos genera la tabla desabilitando 45 casillas de la tabla

```
917 {
918     invalidateWindow();
919     ui->groupBox_2->setEnabled(true);
920     for (int i = 0; i < 9; i++)
921     {
922         number[i]->setEnabled(true);
923     }
924         sudokuErrorWiper();
925         while(!sudokuBlankChecker())
926         {
```

```
927             sudokuErrorWiper();
928             sudokuGenerator(0);
929         }
930 
931         int counter;
932         int row;
933         int column;
934         for(int k = 0; k < 45; k++)
935         {
936             row = rand() % 9;
937             column = rand() % 9;
938 
939             counter = 0;
940 
941             for (int i = 0; i < 9; i++)
942             {
943                 for (int j = 0; j < 9; j++)
944                 {
945 
946                     if  (i == row || j == column || (i / 3 == row / 3 && j / 3 == column / 3))
947                     {
948                         if (cell[i][j]->isConstant)
949                         {
950                             counter ++;
951                         }
952                     }
953 
954                 }
955 
956             }
957             if (counter < 6)
958             {
959                 cell[row][column]->isConstant = true;
960                 cell[row][column]->setEnabled(false);
961                 cell[row][column]->setText(QString("%1").arg(cell[row][column]->realNumber));
962                 cell[row][column]->assignedNumber = cell[row][column]->
    realNumber;
963             }
964             else
965             {
966                 k--;
967             }
968 
969         }
970         invalidateWindow();
971         validateWindow();
972         clock->timeAdded = 0;
973         clock->startTimer();
974         clock->setHidden(false);
975 
976 }
```

1.3.2.9. void Sudoku::on_actionHard_triggered ( ) [private],[slot]

Nivel DIFICIL.

Nos genera la tabla desabilitando 30 casillas de la tabla

```
1049 {
1050     invalidateWindow();
1051     ui->groupBox_2->setEnabled(true);
1052     for (int i = 0; i < 9; i++)
1053     {
1054         number[i]->setEnabled(true);
1055     }
1056         sudokuErrorWiper();
1057         while(!sudokuBlankChecker())
1058         {
1059             sudokuErrorWiper();
1060             sudokuGenerator(0);
1061         }
1062 
1063         int counter;
1064         int row;
1065         int column;
1066         for(int k = 0; k < 30; k++)
1067         {
1068             row = rand() % 9;
1069             column = rand() % 9;
1070 
1071             counter = 0;
1072 
```

```
1073             for (int i = 0; i < 9; i++)
1074             {
1075                 for (int j = 0; j < 9; j++)
1076                 {
1077
1078                     if  (i == row || j == column || (i / 3 == row / 3 && j / 3 == column / 3))
1079                     {
1080                         if (cell[i][j]->isConstant)
1081                         {
1082                             counter ++;
1083                         }
1084                     }
1085
1086                 }
1087
1088             }
1089             if (counter < 3)
1090             {
1091                 cell[row][column]->isConstant = true;
1092                 cell[row][column]->setEnabled(false);
1093                 cell[row][column]->setText(QString("%1").arg(cell[row][column]->realNumber));
1094                 cell[row][column]->assignedNumber = cell[row][column]->
      realNumber;
1095             }
1096             else
1097             {
1098                 k--;
1099             }
1100         }
1101         invalidateWindow();
1102         validateWindow();
1103         clock->timeAdded = 0;
1104         clock->startTimer();
1105         clock->setHidden(false);
1106 }
```

1.3.2.10. void Sudoku::on_actionMedium_triggered ( ) [private],[slot]

Nivel DIFICIL.

Nos genera la tabla desabilitando 30 casillas de la tabla

```
984 {
985     invalidateWindow();
986     ui->groupBox_2->setEnabled(true);
987     for (int i = 0; i < 9; i++)
988     {
989         number[i]->setEnabled(true);
990     }
991         sudokuErrorWiper();
992         while(!sudokuBlankChecker())
993         {
994             sudokuErrorWiper();
995             sudokuGenerator(0);
996         }
997
998         int counter;
999         int row;
1000         int column;
1001         for(int k = 0; k < 36; k++)
1002         {
1003             row = rand() % 9;
1004             column = rand() % 9;
1005
1006             counter = 0;
1007
1008             for (int i = 0; i < 9; i++)
1009             {
1010                 for (int j = 0; j < 9; j++)
1011                 {
1012
1013                     if  (i == row || j == column || (i / 3 == row / 3 && j / 3 == column / 3))
1014                     {
1015                         if (cell[i][j]->isConstant)
1016                         {
1017                             counter ++;
1018                         }
1019                     }
1020
1021                 }
1022
1023             }
```

```
1024             if (counter < 5)
1025             {
1026                 cell[row][column]->isConstant = true;
1027                 cell[row][column]->setEnabled(false);
1028                 cell[row][column]->setText(QString("%1").arg(cell[row][column]->realNumber));
1029                 cell[row][column]->assignedNumber = cell[row][column]->
     realNumber;
1030             }
1031             else
1032             {
1033                 k--;
1034             }
1035
1036         }
1037         invalidateWindow();
1038         validateWindow();
1039         clock->timeAdded = 0;
1040         clock->startTimer();
1041         clock->setHidden(false);
1042 }
```

1.3.2.11. void Sudoku::on_actionNew_triggered ( ) [private],[slot]

```
456 {
457     invalidateWindow();
458     ui->groupBox_2->setEnabled(true);
459     for (int i = 0; i < 9; i++)
460     {
461         number[i]->setEnabled(true);
462     }
463     sudokuErrorWiper();
464     while(!sudokuBlankChecker())
465     {
466         sudokuErrorWiper();
467         sudokuGenerator(0);
468     }
469
470     int counter;
471     int row;
472     int column;
473     for(int k = 0; k < 30; k++)
474     {
475         row = rand() % 9;
476         column = rand() % 9;
477
478         counter = 0;
479
480         for (int i = 0; i < 9; i++)
481         {
482             for (int j = 0; j < 9; j++)
483             {
484
485                 if  (i == row || j == column || (i / 3 == row / 3 && j / 3 == column / 3))
486                 {
487                     if (cell[i][j]->isConstant)
488                     {
489                         counter ++;
490                     }
491                 }
492
493             }
494
495         }
496         if (counter < 5)
497         {
498             cell[row][column]->isConstant = true;
499             cell[row][column]->setEnabled(false);
500             cell[row][column]->setText(QString("%1").arg(cell[row][column]->realNumber));
501             cell[row][column]->assignedNumber = cell[row][column]->realNumber;
502         }
503         else
504         {
505             k--;
506         }
507
508     }
509     invalidateWindow();
510     validateWindow();
511     clock->timeAdded = 0;
512     clock->startTimer();
513     clock->setHidden(false);
514 }
```

1.3.2.12. void Sudoku::on_actionOpen_triggered ( ) [private],[slot]

```
518 {
519     QString code[9];
520     QString key[9];
521     QString hint[9];
522 
523     crypt = "";
524     int timeAdded;
525     int decimal = 0;
526     int digit = 0;
527     QString sign;
528     QString synt;
529 
530     sudokuErrorWiper();
531 
532     QFile inFile(QFileDialog::getOpenFileName(NULL, "Open", QDir::homePath(), "*.su"));
533     inFile.open(QIODevice::Text | QIODevice::ReadOnly);
534     QTextStream in(&inFile);
535 
536     int tracker = 0;
537 
538     while (!in.atEnd()) {
539             QString line = in.readLine();
540             if (line.length())
541             {
542                 line.replace(QString("\n"), QString(""));
543 
544                 if(tracker == 27)
545                 {
546                     timeAdded = line.toInt();
547                 }
548                 else if(tracker > 27)
549                 {
550                     cell[line.toInt() / 10][line.toInt() % 10]->isConstant = true;
551                 }
552                 else if(tracker % 3 == 0)
553                 {
554                     code[tracker / 3] = line;
555                 }
556                 else if(tracker % 3 == 1)
557                 {
558                     key[tracker / 3] = line;
559                 }
560                 else if(tracker % 3 == 2)
561                 {
562                     hint[tracker / 3] = line;
563                 }
564                 tracker++;
565             }
566 
567         }
568 
569     crypt = "";
570 
571     for (int i = 0; i < 9; i++)
572     {
573 
574         QString binary;
575 
576         for (int k = 0; k < 3; k++)
577         {
578 
579 
580             if(key[i].mid(k, 1) == "a")
581             {
582                 digit = 10;
583             }
584             else if(key[i].mid(k, 1) == "b")
585             {
586                 digit = 11;
587             }
588             else if(key[i].mid(k, 1) == "c")
589             {
590                 digit = 12;
591             }
592             else if(key[i].mid(k, 1) == "d")
593             {
594                 digit = 13;
595             }
596             else if(key[i].mid(k, 1) == "e")
597             {
598                 digit = 14;
599             }
600             else if(key[i].mid(k, 1) == "f")
601             {
602                 digit = 15;
```

```
603             }
604             else {
605                 digit = key[i].mid(k, 1).toInt();
606             }
607             decimal += digit * pow(16, 2 - k);
608         }
609 
610 
611 
612         binary.setNum(decimal, 2);
613 
614         for (int j = 0; j < 9; j++)
615         {
616 
617 
618             sign = binary.mid(j, 1);
619 
620             if(code[i].mid(j, 1) == "a")
621             {
622                 synt = "1";
623             }
624             else if(code[i].mid(j, 1) == "b")
625             {
626                 synt = "2";
627             }
628             else if(code[i].mid(j, 1) == "c")
629             {
630                 synt = "3";
631             }
632             else if(code[i].mid(j, 1) == "d")
633             {
634                 synt = "4";
635             }
636             else if(code[i].mid(j, 1) == "e")
637             {
638                 synt = "5";
639             }
640             else if(code[i].mid(j, 1) == "f")
641             {
642                 synt = "6";
643             }
644             else if(code[i].mid(j, 1) == "g")
645             {
646                 synt = "7";
647             }
648             else if(code[i].mid(j, 1) == "h")
649             {
650                 synt = "8";
651             }
652             else if(code[i].mid(j, 1) == "i")
653             {
654                 synt = "9";
655             }
656 
657             int deltaInverse = pow(-1, sign.toInt() + 1) * synt.toInt();
658 
659             if(!j)
660             {
661                 cell[i][j]->realNumber = deltaInverse;
662             }
663             else
664             {
665                 cell[i][j]->realNumber = cell[i][j - 1]->realNumber + deltaInverse;
666             }
667         }
668         decimal = 0;
669     }
670     invalidateWindow();
671     validateWindow();
672 
673     for (int i = 0; i < 9; i++)
674     {
675         for(int j = 0; j < 9; j++)
676         {
677             if (hint[i].mid(j, 1).toInt())
678             {
679                 cell[8 - i][8 - j]->setText(hint[i].mid(j, 1));
680             }
681             else
682             {
683                 cell[8 -i][8 - j]->setText("");
684             }
685             cell[8 -i][8 - j]->assignedNumber = hint[i].mid(j, 1).toInt();
686         }
687     }
688     clock->timeAdded = timeAdded;
689     clock->startTimer();
```

```
690     clock->setHidden(false);
691 }
```

1.3.2.13. void Sudoku::on_actionQuit_triggered ( ) [private],[slot]

```
907 {
908     qApp->quit();
909 }
```

1.3.2.14. void Sudoku::on_actionSave_triggered ( ) [private],[slot]

```
695 {
696     QString code[9];
697     QString key[9];
698     QString hint[9];
699     int binaryBuffer = 0;
700
701     crypt = "";
702     int delta = 0;
703
704     if (!sudokuBlankChecker())
705     {
706         return;
707     }
708     for (int i = 0; i < 9; i++)
709     {
710         for (int j = 0; j < 9; j++)
711         {
712
713             if(!j)
714             {
715                 delta = cell[i][j]->realNumber;
716                 code[i] = code[i].append(QString("%1").arg((char)(delta + 96)));
717
718             }
719             else
720             {
721                 delta = cell[i][j]->realNumber - cell[i][j - 1]->realNumber;
722                 code[i] = code[i].append(QString("%1").arg((char)(abs(delta) + 96)));
723
724
725             }
726             if(delta > 0)
727             {
728                 binaryBuffer += pow(2, 8 - j);
729
730             }
731
732             hint[i] = hint[i].append(QString("%1").arg(cell[8 - i][8 - j]->assignedNumber));
733
734         }
735         key[i].setNum(binaryBuffer, 16);
736
737         delta = 0;
738         binaryBuffer = 0;
739         crypt = crypt.append(QString("%1\n").arg(code[i]));
740         crypt = crypt.append(QString("%1\n").arg(key[i]));
741         crypt = crypt.append(QString("%1\n").arg(hint[i]));
742     }
743
744     crypt = crypt.append(QString("%1\n").arg(clock->getTimeScore() + clock->
    timeAdded));
745     for(int i = 0; i < 9; i++)
746     {
747         for(int j = 0; j < 9; j++)
748         {
749             if(cell[i][j]->isConstant)
750             {
751                 crypt = crypt.append(QString("%1%2\n").arg(i).arg(j));
752             }
753
754         }
755     }
756
757     QFileDialog dialog(NULL);
758     dialog.setFileMode(QFileDialog::AnyFile);
759     dialog.setDefaultSuffix("su");
760     QFile outFile(dialog.getSaveFileName(NULL, "Save", QDir::homePath()+".su","Any Files (*.su)" ));
761     outFile.open(QIODevice::Text | QIODevice::WriteOnly);
762     QTextStream out(&outFile);
```

```
763
764     out « crypt;
765     outFile.flush();
766     outFile.close();
767
768     crypt.clear();
769 }
```

1.3.2.15. void Sudoku::on_clueButton_clicked ( ) [private],[slot]

Game Mode Clue(Modo de Juego)

Esta opcion muestra en la tabla con color amarillo todos las casillas en donde se puede ubicar el numero seleccionado

```
897 {
898     invalidateWindow();
899     ui->groupBox_2->setEnabled(true);
900     for (int i = 0; i < 9; i++)
901     {
902         number[i]->setEnabled(true);
903     }
904 }
```

1.3.2.16. void Sudoku::on_hintButton_clicked ( ) [private],[slot]

Game Mode Hint (Modo de Juego Hint)

Esta opcion sirve para ayudar al usuario desactivando los numeros que no se pueden usar en la celda seleccionada

```
879 {
880     invalidateWindow();
881     validateWindow();
882     for (int i = 0; i < 9; i++)
883     {
884         number[i]->setEnabled(true);
885     }
886     ui->gridLayout->setEnabled(true);
887     if (isCellSelected){
888         validateCell();
889     }
890 }
```

1.3.2.17. void Sudoku::on_normalButton_clicked ( ) [private],[slot]

Game Mode Normal (Modo de Juego)

Esta opcion no da ayuda al usuario solo una retroalimentacion visual de donde se ubica en el tablero

```
858 {
859     invalidateWindow();
860     validateWindow();
861     for (int i = 0; i < 9; i++)
862     {
863         number[i]->setEnabled(true);
864     }
865     ui->gridLayout->setEnabled(true);
866     if (isCellSelected){
867         for (int i = 0; i < 9; i++)
868         {
869             number[i]->setEnabled(true);
870         }
871     }
872
873 }
```

1.3.2.18. void Sudoku::on_nullButton_clicked ( ) [private], [slot]

```
409 {
410     if (isCellSelected)
411     {
412         int value = cell[coorX][coorY]->assignedNumber;
413         number[value - 1]->setEnabled(true);
414         cell[coorX][coorY]->setText("");
415         cell[coorX][coorY]->assignedNumber = 0;
416         ui->nullButton->setEnabled(false);
417         ui->radioButton->setChecked(false);
418     }
419 }
```

1.3.2.19. void Sudoku::on_pushButton_clicked ( ) [private], [slot]

```
828 {
829     QDialog saveName;
830     saveName.exec();
831 }
```

1.3.2.20. bool Sudoku::sudokuBlankChecker ( ) [private], [slot]

```
224 {
225     for(int i = 0; i < 9; i++)
226     {
227         for(int j = 0; j < 9; j++)
228         {
229             if (!cell[i][j]->realNumber)
230             {
231                 return false;
232             }
233         }
234     }
235     return true;
236 }
```

1.3.2.21. void Sudoku::sudokuErrorWiper ( ) [private], [slot]

```
240 {
241     for(int i = 0; i < 9; i++)
242     {
243         for(int j = 0; j < 9; j++)
244         {
245             cell[i][j]->realNumber = 0;
246             cell[i][j]->assignedNumber = 0;
247             cell[i][j]->setText("");
248             cell[i][j]->isConstant = false;
249             cell[i][j]->setEnabled(true);
250         }
251     }
252 }
```

1.3.2.22. void Sudoku::sudokuGenerator (int *row*) [private], [slot]

```
113 {
114     if (row > 8)
115     {
116         return;
117     }
118
119     int range = 9;
120     int digitArray[9];
121     for (int i = 0; i < 9; i++)
122     {
123         digitArray[i] = i;
124     }
125     while (range > 0)
126     {
127         int random = rand() % range;
128         range--;
```

```
129
130         int temp = digitArray[range];
131         digitArray[range] = digitArray[random];
132         digitArray[random] = temp;
133
134     }
135     bool flag = false;
136     for (int i = 0; i < 9; i++)
137     {
138         for (int j = 0; j < 9; j++)
139         {
140             flag = false;
141             if (digitArray[j] >= 0)
142             {
143                 qDebug() « "Index: (" « row « ", " « i « ")\nValue: " « (digitArray[j] + 1) « "\n";
144                 if (numberAssigner(0, row, i, digitArray[j] + 1))
145                 {
146                     digitArray[j] = -1;
147                     flag = true;
148                 }
149
150             }
151             if (flag)
152             {
153                 break;
154             }
155         }
156     }
157
158
159
160     sudokuGenerator(row + 1);
161 }
```

1.3.2.23. void Sudoku::validateCell ( ) [private],[slot]

```
423 {
424     int centerX = coorX / 3;
425     int centerY = coorY / 3;
426
427     for (int i = 0; i < 9; i++)
428     {
429         for (int j = 0; j < 9; j++)
430         {
431
432             if  (i == coorX || j == coorY || (i / 3 == centerX && j / 3 == centerY))
433             {
434                 if (cell[i][j]->assignedNumber)
435                 {
436                     int value = cell[i][j]->assignedNumber;
437                     number[value - 1]->setEnabled(false);
438                 }
439             }
440
441         }
442
443     }
444     if (!cell[coorX][coorY]->assignedNumber)
445     {
446         ui->nullButton->setEnabled(false);
447     }
448     else
449     {
450         ui->nullButton->setEnabled(true);
451     }
452 }
```

1.3.2.24. bool Sudoku::validateCellAtPointWithNumber (int *row,* int *column,* int *value,* int *counter*) [private], [slot]

```
196 {
197     int centerX = row / 3;
198     int centerY = column / 3;
199
200     for (int i = 0; i < 9; i++)
201     {
202         for (int j = 0; j < 9; j++)
203         {
204
```

```
205             if  ((i == row || j == column || (i / 3 == centerX && j / 3 == centerY)))
206             {
207                 if (cell[i][j]->realNumber)
208                 {
209                     if (value == cell[i][j]->realNumber && !(i == row && j == column - counter))
210                     {
211                         return false;
212                     }
213                 }
214             }
215 
216         }
217 
218     }
219     return true;
220 }
```

1.3.2.25. void Sudoku::validateWindow ( ) [private], [slot]

```
773 {
774     for(int i = 0; i < 9; i++)
775     {
776         for(int j = 0; j < 9; j++)
777         {
778             if (!cell[i][j]->isConstant)
779             {
780                 cell[i][j]->setEnabled(true);
781             }
782         }
783     }
784 
785     ui->groupBox_2->setEnabled(true);
786     //ui->normalButton->setChecked(true);
787 }
```

1.3.3. Documentación de los datos miembro

1.3.3.1. QPushButtonGrid* Sudoku::cell[9][9] [private]

1.3.3.2. Clock* Sudoku::clock [private]

1.3.3.3. int Sudoku::coorX [private]

1.3.3.4. int Sudoku::coorY [private]

1.3.3.5. QString Sudoku::crypt [private]

1.3.3.6. bool Sudoku::isCellSelected [private]

1.3.3.7. QPushButton* Sudoku::number[9] [private]

1.3.3.8. Ui::Sudoku* Sudoku::ui [private]

La documentación para esta clase fue generada a partir de los siguientes ficheros:

- I:/Sudoku-master/Sudoku/Functional Version/**sudoku.h**
- I:/Sudoku-master/Sudoku/Functional Version/**sudoku.cpp**

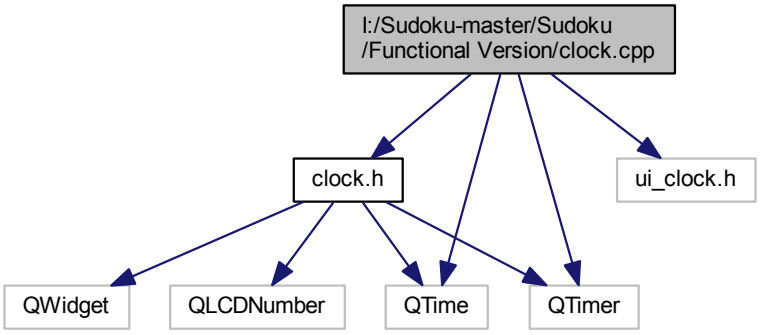
Capítulo 2

Documentación de archivos

2.1. Referencia del Archivo I:/Sudoku-master/Sudoku/Functional Version/clock.cpp

```
#include "clock.h"
#include "ui_clock.h"
#include <QTime>
#include <QTimer>
```

Dependencia gráfica adjunta para clock.cpp:



2.2. Referencia del Archivo I:/Sudoku-master/Sudoku/Functional Version/clock.h

```
#include <QWidget>
#include <QLCDNumber>
#include <QTime>
#include <QTimer>
```

Dependencia gráfica adjunta para clock.h:

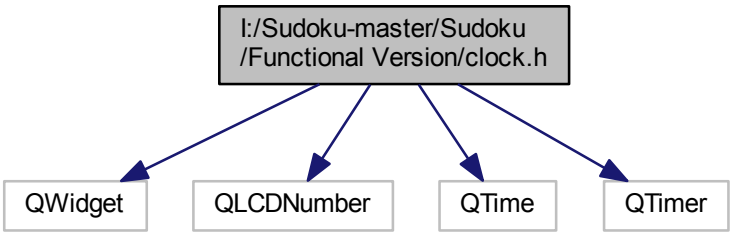
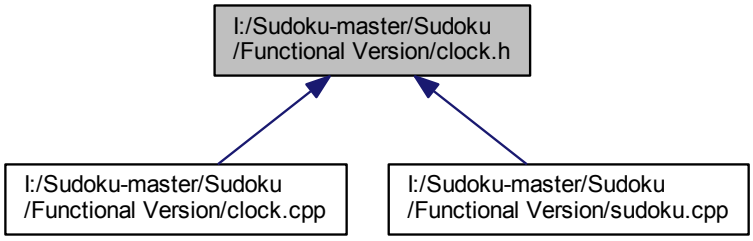


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class **Clock**

Namespaces

- **Ui**

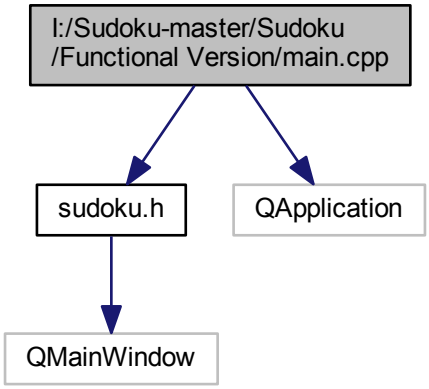
Constant Groups

- **Ui**

2.3. Referencia del Archivo I:/Sudoku-master/Sudoku/Functional Version/main.cpp

```
#include "sudoku.h"
#include <QApplication>
```

Dependencia gráfica adjunta para main.cpp:



Funciones

- int **main** (int argc, char ∗argv[])

2.3.1. Documentación de las funciones

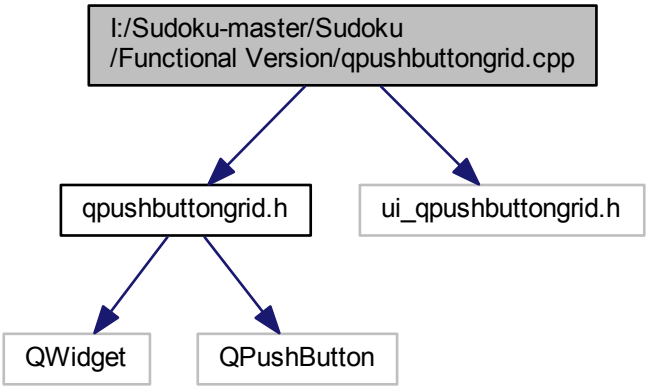
2.3.1.1. int main (int *argc*, char ∗ *argv[]*)

```
5 {
6     QApplication a(argc, argv);
7     Sudoku w;
8     w.show();
9
10     return a.exec();
11 }
```

2.4. Referencia del Archivo I:/Sudoku-master/Sudoku/Functional Version/qpushbuttongrid.cpp

```
#include "qpushbuttongrid.h"
#include "ui_qpushbuttongrid.h"
```

Dependencia gráfica adjunta para qpushbuttongrid.cpp:



2.5. Referencia del Archivo I:/Sudoku-master/Sudoku/Functional Version/qpushbuttongrid.h

```
#include <QWidget>
#include <QPushButton>
```

Dependencia gráfica adjunta para qpushbuttongrid.h:

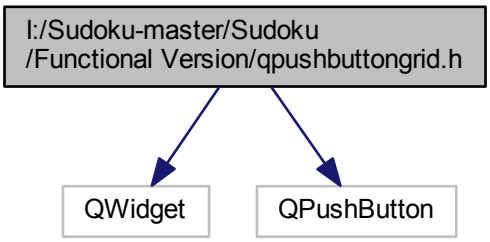
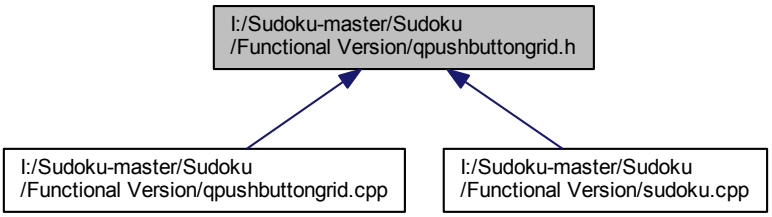


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class **QPushButtonGrid**

Namespaces

- **Ui**

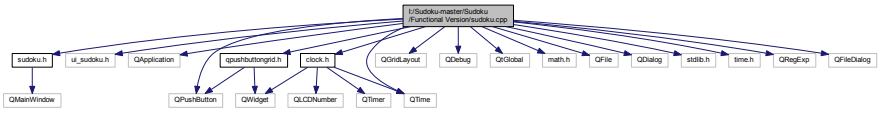
Constant Groups

- **Ui**

2.6. Referencia del Archivo I:/Sudoku-master/Sudoku/Functional Version/sudoku.cpp

```
#include "sudoku.h"
#include "ui_sudoku.h"
#include <QApplication>
#include <QPushButton>
#include <qpushbuttongrid.h>
#include <QGridLayout>
#include <QDebug>
#include <QTime>
#include <QtGlobal>
#include <math.h>
#include <QFile>
#include <QDialog>
#include <clock.h>
#include <stdlib.h>
#include "time.h"
#include <QRegExp>
#include <QFileDialog>
```

Dependencia gráfica adjunta para sudoku.cpp:



2.7. Referencia del Archivo I:/Sudoku-master/Sudoku/Functional Version/sudoku.h

```
#include <QMainWindow>
```

Dependencia gráfica adjunta para sudoku.h:

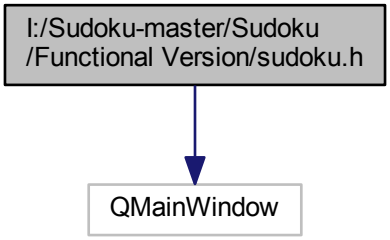
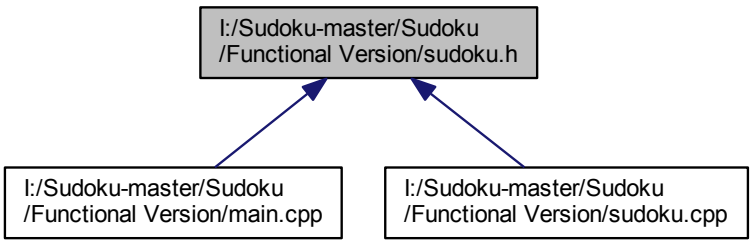


Gráfico de los archivos que directa o indirectamente incluyen a este archivo:



Clases

- class **Sudoku**

Namespaces

- **Ui**

Constant Groups

- **Ui**

Índice alfabético