```
/**
  @brief declaration of Board class that represents the mined board and works
with the main logic of the game
  @author Angelica Patsko
  @date 6/5/2016
*/

#ifndef FIELD_H
#define FIELD_H
#include <QWidget>
#include "cell.h"
#include "clock.h"


/**
 * @brief The Board class represents the mined board
 */
class Board :public QWidget {

  Q_OBJECT

public:

  Board(QWidget *parent = 0, int m_n = 8, int m_mineNumber = 10);

protected:

  void openAll();
  bool isValidCoord(int i, int j);
  void openCell(int i, int j);
  bool is_win();
  int m_n;
  int m_mineNumber;
  Cell** m_board;
  Clock* m_clock;
  bool m_isGameActive;

protected slots:
  void on_CellClickedLeft();
  void on_CellClickedRight();
};

#endif // FIELD_H
```

```
/**
  @brief declaration of Cell class that represents one cell on the board
  @author Angelica Patsko
  @date 6/5/2016
*/

#ifndef DOT_H
#define DOT_H
#include <QPushButton>

/**
 * @brief The Cell class represents one Cell on the board
 */
class Cell : public QPushButton {
  Q_OBJECT

public:
    Cell(int i, int j, QWidget *parent = 0);
    int i();
    int j();
    bool isMine();
    void setMine();
    void incValue();
    int value();
    void setOpen();
    bool isOpened();
    void swapFlag();

signals:
    void clicked_left();
    void clicked_right();

protected:
    virtual void paintEvent(QPaintEvent *event);
    virtual void mouseReleaseEvent(QMouseEvent *e);
    //row- and column- coordinates
    int m_i, m_j;
    //value containing in a square = number of mines around
    int m_value;
    //checks whether the square is mined
    bool m_isMine;
    bool m_isOpen;
    bool m_isFlag;
};

#endif // DOT_H
```

```
/**
  @brief declaration of CLock class derived from QLSDNumber that works with
the clock on the board
  @author Angelica Patsko
  @date 6/5/2016
*/

#ifndef CLOCK_H
# define CLOCK_H
# include <QLCDNumber>
# include <QTime>

/**
 * @brief The Clock class for displaying time on the board
 */
class Clock: public QLCDNumber {
    Q_OBJECT

public:
  Clock(QWidget *parent = 0);

private slots:
  void on_tick();
  void stop();

protected:
  //current time
  QTime m_time;

  //timer
  QTimer *m_timer;
};

#endif // CLOCK_H
```

```
/**
  @brief declaration of MainWindow class that handles the first window you
see when you launch the program
  @author Angelica Patsko
  @date 6/5/2016
*/


#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "minesweeper.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public slots:
    void easy_game_begin();
    void medium_game_begin();
    void hard_game_begin();
    void game_over();

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
    Minesweeper* board;

    QMenuBar* menubar;
    QMenu* gameMenu;
    QMenu* helpMenu;
};

#endif // MAINWINDOW_H
```

```
/**
  @brief declaration of MinesweeperBoard class that works with interface
  @author Angelica Patsko
  @date 6/5/2016
*/



#ifndef MINESWEEPERBOARD_H
#define MINESWEEPERBOARD_H

#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QGridLayout>
#include <QLabel>
#include <QMenuBar>
#include <QMainWindow>
#include <QMessageBox>
#include "clock.h"
#include "cell.h"
#include "board.h"



namespace Ui{
class MinesweeperBoard;
}

class Minesweeper : public QWidget
{
    Q_OBJECT

public:
    explicit Minesweeper(QWidget *parent = 0, size_t boardSize = 8, size_t
minesNumber = 10);
    ~Minesweeper();

public slots:
   void info();
   void about();

signals:
   void game_over();
   void easy_game_begin();
   void medium_game_begin();
   void hard_game_begin();

private:
    Ui::MinesweeperBoard *ui;

    //parameters we need to create a new game depending on the level
    size_t boardSize;
    size_t minesNumber;

    //layouts
    QVBoxLayout* v_layout;
    QHBoxLayout* h_layout;

    //topmenu items
    QMenuBar* menubar;
    QMenu* gameMenu;
    QMenu* helpMenu;

    //the top part
    QWidget* top;
    QLabel* labelBombs;
    QLabel* labelTime;
    QPushButton* startOverButton;

    //the actual gameboard
    QWidget* board;
    QPushButton** cells;

    Board* m_field;
    Clock* m_clock;
};

#endif // MINESWEEPERBOARD_H
```

```
/**
  @brief implementation of Board class that represents the mined board and
works with the main logic of the game
  @author Angelica Patsko
  @date 6/5/2016
*/

#include "board.h"
#include "cell.h"
#include <QGridLayout>
#include <QMessageBox>

/**
 * @brief Board::Board constructor, sets up the board
 * @param parent used for memory management purposes
 * @param m_n number of rows and columns on the board
 * @param m_mineNumber number of mines
 */
Board::Board(QWidget *parent, int m_n, int m_mineNumber ): QWidget(parent) {

    m_isGameActive = true;
    this->m_n = m_n;
    this->m_mineNumber = m_mineNumber;
    QGridLayout *layout = new QGridLayout(this);

    m_board = new Cell*[m_n*m_n];

    //adds cells to the board
    for (int i = 0; i < m_n; ++i) {
        for (int j = 0; j < m_n; ++j) {
        m_board[i*m_n+j] = new Cell(i, j, this);
        layout->addWidget(m_board[i*m_n+j], i, j, 1, 1);
        connect(m_board[i*m_n+j], SIGNAL(clicked_left()), this,
SLOT(on_CellClickedLeft()));
        connect(m_board[i*m_n+j], SIGNAL(clicked_right()), this,
SLOT(on_CellClickedRight()));
        }
    }

    //adding mines to the board
    for (int i = 0; i < m_mineNumber;) {
        int rand_i = qrand() % m_n;
        int rand_j = qrand() % m_n;
        Cell *p = m_board[rand_i*m_n+rand_j];
        if (p->isMine())
            continue;
        else {
            p->setMine();
            ++i;
        }
    }

    //setting the mines number by checking the surrounding cells
    for (int i = 0; i < m_n; ++i) {
        for (int j = 0; j < m_n; ++j) {
            if (m_board[i*m_n+j]->isMine()) {
                if (isValidCoord(i - 1, j - 1)) m_board[(i - 1)*m_n + j - 1]-
>incValue();
                if (isValidCoord(i - 1, j)) m_board[(i - 1)*m_n+j]->incValue();
                if (isValidCoord(i - 1, j + 1)) m_board[(i - 1)*m_n+j + 1]-
>incValue();
                if (isValidCoord(i, j - 1)) m_board[i* m_n + j - 1]->incValue();
                if (isValidCoord(i, j + 1)) m_board[i* m_n + j + 1]->incValue();
                if (isValidCoord(i + 1, j - 1)) m_board[(i + 1)*m_n + j - 1]-
>incValue();
                if (isValidCoord(i + 1, j)) m_board[(i + 1)*m_n+j]->incValue();
                if (isValidCoord(i + 1, j + 1)) m_board[(i + 1)*m_n+j + 1]-
>incValue();
            }
        }
    }
}



/**
 * @brief Board::isValidCoord checks whether the coordinanes fall in this
range [0; m_n -1]
 * @param i row-coordinate
 * @param j column
 * @return true if the coordinates are in the allowed range
 */
bool Board::isValidCoord(int i, int j) {
    return i >= 0 && j >= 0 && i < m_n && j < m_n;
}


/**
 * @brief Board::is_win checks if a player opened all the unmined cells
 * @return true if he/she did
 */
bool Board::is_win() {
    int n = m_n * m_n - m_mineNumber;
    for (int i = 0; i < m_n; ++i)
        for (int j = 0; j < m_n; ++j)
            n -= m_board[i*m_n+j]->isOpened();
    return 0 == n;
}

/**
 * @brief Board::openCell opens the cell with (i,j) coordinates, if zero,
opens suuronding o-cells as well
 * @param i row-coordinate
 * @param j column-coordinate
 */
void Board::openCell(int i, int j) {
    if (!isValidCoord(i, j)) return;
    Cell *p = m_board[i*m_n+j];
    if (p->isOpened())
        return;
    p->setOpen();
    if (p->value()) return;
    //open all other cells with 0 value
    openCell(i - 1, j);
    openCell(i + 1, j);
    openCell(i, j - 1);
    openCell(i, j + 1);
}

/**
 * @brief Board::on_CellClickedRight processes right clicks on closed cells
 */
void Board::on_CellClickedRight() {
    if (!m_isGameActive) return;
    Cell *t = reinterpret_cast<Cell*>(sender());
    if (t->isOpened()) return;
    t->swapFlag();
}

/**
 * @brief Board::openAll reveals all the cells
 */
void Board::openAll() {
    for (int i = 0; i < m_n; ++i)
        for (int j = 0; j < m_n; ++j)
            m_board[i*m_n+j]->setOpen();
}

/**
 * @brief Board::on_CellClickedLeft processes right clicks on closed cells,
if click on mined cell - lose
 */
void Board::on_CellClickedLeft() {
    if (!m_isGameActive)
        return;
    Cell* t = reinterpret_cast<Cell*>(sender());
    if (t->isOpened())
        return;
    //clicking on mined cell
    if (t->isMine()) {
        m_isGameActive = false;
        QMessageBox::information(0, "Information", "Sorry, you lost");
        openAll();
        return;
    }
    openCell(t->i(), t->j());
    //if opened the last cell
    if (is_win()) {
        m_isGameActive = false;
        QMessageBox::information(0, "Information", "Congratulations! You
won!");
        openAll();
    }
}
```

```
/**
  @brief implementation of Cell class that that represents one cell on the
board
  @author Angelica Patsko
  @date 6/5/2016
*/

#include "cell.h"
#include <QPainter>
#include <QMouseEvent>
/**
 * @brief  constructor
 * @param i - row-coordinate
 * @param j - column-coordinate
 * @param parent - passed for memory management purposes
 */
Cell::Cell(int i, int j, QWidget *parent): QPushButton(parent), m_i(i),
m_j(j), m_value(0), m_isMine(false), m_isOpen(false), m_isFlag(false) {
    setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);
    setFixedSize(30, 30);
}

/**
 * @brief Cell::i - getter of i
 * @return the row-coordinate
 */
int Cell::i() { return m_i; }

/**
 * @brief Cell::j - getter of j
 * @return the column-coordinate
 */
int Cell::j() { return m_j; }

/**
 * @brief Cell::value getter of m_value
 * @return m_value - number of mines around
 */
int Cell::value() { return m_value; }

/**
 * @brief Cell::incValue increments the number of mines around the square
 */
void Cell::incValue() { ++m_value; }

/**
 * @brief Cell::isMine checks whether the square is mined
 * @return true if it is
 */
bool Cell::isMine() { return m_isMine; }

/**
 * @brief Cell::setMine sets the Mine on the square
 */
void Cell::setMine() { m_isMine = true; repaint(); }

/**
```

```
 * @brief Cell::isOpen checks whethe the square is opened
 * @return true if it is
 */
bool Cell::isOpened() { return m_isOpen; }

/**
 * @brief Cell::setOpen sets the m_isOpen property
 */
void Cell::setOpen() { m_isOpen = true; repaint(); }

/**
 * @brief Cell::swapFlag sets/removes the flag on the square
 */
void Cell::swapFlag() { m_isFlag = !m_isFlag; repaint(); }

/**
 * @brief Cell::paintEvent - paints circles on the cells, red for mine, blue
for flag
 * @param event - paint event
 */
void Cell::paintEvent(QPaintEvent *event) {
    if (isOpened() && isMine() == false && 0 == m_value)
        return;
    QPushButton::paintEvent(event);
    QPainter p(this);
    if (isOpened()) {
        if (isMine()) {
            p.setBrush(QBrush(Qt::red, Qt::SolidPattern));
            p.drawEllipse(2, 2, width() - 4, height() - 4);
            return;
        }
        setText(QString::number(m_value));
        return;
    }
    if (m_isFlag) {
    p.setBrush(QBrush(Qt::blue, Qt::SolidPattern));
    p.drawEllipse(2, 2, width() - 4, height() - 4);
  }
}

/**
** * @brief Cell::mouseReleaseEvent processes left- and right- mouse clicks
** * @param e - mouse event
** */
void Cell::mouseReleaseEvent(QMouseEvent *e) {
    if (e->button() == Qt::LeftButton) emit clicked_left();
    if (e->button() == Qt::RightButton) emit clicked_right();
}
```

```cpp
/**
  @brief implementation of Clock class that works with the clock on the board
  @author Angelica Patsko
  @date 6/5/2016
*/

#include "clock.h"
#include <QTimer>

/**
 * @brief Clock::Clock constructor
 * @param parent - used for memory management (Clock will be added to param's
cildren list and will be deleted with param)
 */
Clock::Clock(QWidget *parent): QLCDNumber(parent), m_time(0, 0, 0) {
    m_timer = new QTimer(this);
    connect(m_timer, SIGNAL(timeout()), this, SLOT(on_tick()));
    m_timer->start(1000);
    display(m_time.toString("hh:mm:ss"));
    setFixedSize(100, 30);
}

/**
 * @brief Clock::on_tick - slot for processing timer's signal
 */
void Clock::on_tick() {
    m_time = m_time.addSecs(1);
    display(m_time.toString("hh:mm:ss"));
}


/**
 * @brief Clock::stop - slot for stopping the timer
 */
void Clock::stop() {
    m_timer->stop();
}
```

```
/**
  @brief main function the launches the application
  @author Angelica Patsko
  @date 6/5/2016
*/
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

```
/**
  @brief implementation of MainWindow class that handles the first window you
see when you launch the program
  @author Angelica Patsko
  @date 6/5/2016
*/

#include "mainwindow.h"
#include "minesweeper.h"
#include "ui_mainwindow.h"

/**
 * @brief MainWindow::MainWindow class to display the main window where you
choose the difficulty level you'd like to play
 * @param parent - helps with memory management (QWidget uses RAII)
 */
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    //connecting the buttons to start a game (choosing the difficulty level)
    QObject::connect(ui->pushButtonBeginner,SIGNAL(clicked()),this,
SLOT(easy_game_begin()));
    QObject::connect(ui->pushButtonIntermediate,SIGNAL(clicked()),this,
SLOT(medium_game_begin()));
    QObject::connect(ui->pushButtonExpert,SIGNAL(clicked()),this,
SLOT(hard_game_begin()));
}

/**
 * @brief MainWindow::easy_game_begin starts a game in a beginner mode
 */

void MainWindow::easy_game_begin() {
    board = new Minesweeper(this,8,10);
    this->setCentralWidget(board);
}

/**
 * @brief MainWindow::medium_game_begin starts a game in an intermediate mode
 */

void MainWindow::medium_game_begin() {
    board = new Minesweeper(this,16,30);
    this->setCentralWidget(board);
}

/**
 * @brief MainWindow::hard_game_begin starts a game in an expert mode
 */
void MainWindow::hard_game_begin() {
    board = new Minesweeper(this,20,60);
    this->setCentralWidget(board);
}
```

```
/**
 * @brief MainWindow::game_over returns to the initial window
 */
void MainWindow::game_over() {
    QWidget* wid = this->centralWidget();
    this->setFixedSize(420,330);
    wid->setParent(nullptr);
    ui->setupUi(this);
    QObject::connect(ui->pushButtonBeginner,SIGNAL(clicked()),this,
SLOT(easy_game_begin()));
    QObject::connect(ui->pushButtonIntermediate,SIGNAL(clicked()),this,
SLOT(medium_game_begin()));
    QObject::connect(ui->pushButtonExpert,SIGNAL(clicked()),this,
SLOT(hard_game_begin()));
}


/**
 * @brief MainWindow::~MainWindow deallocates memory for Ui::MainWindow
 */

MainWindow::~MainWindow()
{
    delete ui;
}
```

```
/**
  @brief implementation of MinesweeperBoard class that directly works with
the game logic and interface
  @author Angelica Patsko
  @date 6/5/2016
*/

#include "minesweeper.h"
#include "mainwindow.h"
#include "ui_minesweeperboard.h"

/**
 * @brief MinesweeperBoard::MinesweeperBoard
 * @param parent type QWidget, helps with the memory management
 * @param boardSz size of the gameboard created
 * @param minesNum number of bombs on the gameboard
 */
Minesweeper::Minesweeper(QWidget *parent, size_t boardSz, size_t minesNum) :
    QWidget(parent), ui(new Ui::MinesweeperBoard), boardSize(boardSz),
minesNumber(minesNum)
{
    ui->setupUi(this);

    //resizing the window
    switch (boardSz){
    case 8:
        parent->setFixedSize(425,370);
        break;
    case 16:
        parent->setFixedSize(490,570);
        break;
    case 20:

        parent->setFixedSize(620,670);
        break;
    default: break;
    }

    //creating and organizing menubar with two menus Game and Help
    menubar  = new QMenuBar(this);

    gameMenu = menubar->addMenu("Game");

    QAction* newAct = new QAction("New",this);
    gameMenu->addAction(newAct);
    QObject::connect(newAct, SIGNAL(triggered()), parent, SLOT(game_over()));

    QAction* beginnerAct = new QAction("Beginner",this);
    gameMenu->addAction(beginnerAct);
    QObject::connect(beginnerAct, SIGNAL(triggered()), parent,
SLOT(easy_game_begin()));

    QAction* intermediateAct = new QAction("Intermediate",this);
    gameMenu->addAction(intermediateAct);
    QObject::connect(intermediateAct, SIGNAL(triggered()), parent,
SLOT(medium_game_begin()));
```

```
    QAction* expertAct = new QAction("Expert",this);
    gameMenu->addAction(expertAct);
    QObject::connect(expertAct, SIGNAL(triggered()), parent,
SLOT(hard_game_begin()));

    helpMenu = menubar->addMenu("Help");

    QAction* instrAct = new QAction("Instructions",this);
    helpMenu->addAction(instrAct);
    QObject::connect(instrAct, SIGNAL(triggered()), this, SLOT(info()));

    QAction* aboutAct = new QAction("About",this);
    helpMenu->addAction(aboutAct);
    QObject::connect(aboutAct, SIGNAL(triggered()), this, SLOT(about()));

    //working with the top part of the window where labels and pushbutton are
    top = new QWidget;
    labelBombs = new QLabel;
    labelBombs->setText("Bombs in the Game: " +
QString::number(minesNumber));
    labelBombs->setAlignment(Qt::AlignCenter);

    startOverButton = new QPushButton;
    startOverButton->setText("Start over!");
    QObject::connect(startOverButton, SIGNAL(clicked()), parent,
SLOT(game_over()));


    Clock* m_time = new Clock(this);


    //adding the label, the button, and the clock to the layout
    h_layout = new QHBoxLayout(top);
    h_layout->addWidget(labelBombs);
    h_layout->addWidget(startOverButton);
    h_layout->addWidget(m_time);

    m_field = new Board(this, boardSize, minesNumber);

    //putting everything together
    v_layout = new QVBoxLayout;
    v_layout->addWidget(top, 0, Qt::AlignCenter);
    v_layout->addWidget(m_field,0, Qt::AlignCenter);
    this->setLayout(v_layout);
}

/**
 * @brief MinesweeperBoard::~MinesweeperBoard deallocates memory for
Ui::MinesweeperBoard and QPushButton**
 */
Minesweeper::~Minesweeper(){
    delete ui;
}

/**
 * @brief Minesweeper::info is the slot that shows a QMessageBox with the
instructions to the game
```

```
 */
void Minesweeper::info(){
    QMessageBox::information(0, "Instructions", "The object of the game is to
find the empty squares while avoiding the mines. If you uncover a mine, the
game ends; if you uncover an empty square, you keep playing. Uncover a
number, and it tells you how many mines lay hidden in the eight surrounding
squares — the information you use to deduce which nearby squares are safe to
click. To uncover a square, right-click it; to identify a potential mine,
left-click the square. Good luck!");
}

/**
 * @brief Minesweeper::about is the slot that shows a QMessageBox with the
the information about the author
 */
void Minesweeper::about(){
    QMessageBox::information(0,"About", "The game was created by Angelica
Patsko. All rights reserved.");
}
```