Файл Game.h:

```
#ifndef GAME_H
#define GAME_H
#include "Subject.h"
#include "Board.h"
class Board;
class Game : public Subject {
public:
    Game(int dimension, int target);
    Board* getGameBoard() const;
    void restart();
    void playerMove(DIRECTION dir);
    int getScore() const;
    int getBest() const;
    bool isGameOver() const;
    bool isFinished() const;

private:
    Board *board;
    bool gameOver;
    string line;
    int finish;
    int score;
    int best;

};

#endif // GAME_H
```

Файл Game.cpp:

```
#include "Headerfile.h"
```

```
#include "Game.h"
#include "Tile.h"

Game::Game(int dimension, int target) {
    finish = target;
    score = 0;
    ifstream fin(":/output/best", ios::in);
    fin >> best;
    cout << best << endl;
    board = new Board(dimension);
    restart();
}

Board* Game::getGameBoard() const {
    return board;
}

void Game::playerMove(DIRECTION dir) {
    board->move(dir);
    if (board->isCollision())
        score += board->getPoints();
    if (!board->isMovePossible())
        gameOver = true;
    notifyObs();
}

void Game::restart() {
    board->reset();
    gameOver = false;
    score = 0;
}

int Game::getScore() const {
    return score;
}

int Game::getBest() const {
    return best;
}

bool Game::isGameOver() const {
    if (!board->isMovePossible())
        return true;
    else
        return false;
}

bool Game::isFinished() const {
    for (int i = 0; i < board->getDimension(); i++) {
        for (int j = 0; j < board->getDimension(); j++) {
```

```
                if (board->getTile(i, j) != NULL && board->getTile(i,
j)->getNumber() == finish)
                    return true;
            }
        }
        return false;
    }
```

Файл Board.h:

```
    #ifndef BOARD_H
    #define BOARD_H

    #include "QHeaderfile.h"
    #include "Subject.h"

    class Tile;

    enum DIRECTION {
        UP, DOWN, LEFT, RIGHT
    };

    class Board : public Subject {
    public:
        Board(int dimension);
        Board(const Board &brd);
        ~Board();

        Tile* getTile(int i, int j);
        void move(DIRECTION direction);
        void reset();
        int getDimension() const;
        int getPoints() const;
        bool isFull() const;
        bool isCollision() const;
        bool isMovePossible() const;

    private:
        QVector<int> freePosition();
        void initialize();
        void moveHorizontal(int i, int j, DIRECTION dir);
        void moveVertical(int i, int j, DIRECTION dir);
        bool isChanged(Board &brd) const;
        bool isInBounds(int i, int j);

        QVector< QVector<Tile*> > board;
        int dimension;
        int lastPoints;
        bool lastCollision;
```

```
};

#endif // BOARD_H
```

Файл Board.cpp:

```
#include "QHeaderfile.h"
#include "Headerfile.h"
#include "Board.h"
#include "Tile.h"

using namespace std;

Board::Board(int dimension) {
    lastPoints = 0;
    lastCollision = false;
    this->dimension = dimension;
    initialize();
}

Board::Board(const Board &brd) {
    dimension = brd.dimension;
    initialize();
    for (int i = 0; i < dimension; i++) {
        for (int j = 0; j < dimension; j++) {
            if (brd.board[i][j] == NULL)
                board[i][j] = NULL;
            else
                board[i][j] = new Tile(*(brd.board[i][j]));
        }
    }
}

Board::~Board() {
    for (int i = 0 ; i < dimension; i++) {
        for (int j = 0; j < dimension; j++)
            delete board[i][j];
    }
}

Tile* Board::getTile(int i, int j) {
    return board[i][j];
}

void Board::initialize() {
    board.resize(dimension);
    for (int i = 0; i < dimension; i++)
        board[i].resize(dimension);
```

```
    for (int i = 0; i < dimension; i++) {
        for (int j = 0; j < dimension; j++)
            board[i][j] = NULL;
    }
}

void Board::move(DIRECTION direction) {
    Board pre_move_board(*this);
    lastCollision = false;
    lastPoints = 0;

    switch (direction) {
        case UP:
            for (int i = 0; i < dimension; i++) {
                for (int j = 0; j < dimension; j++)
                    moveVertical(i, j, UP);
            }
            break;
        case DOWN:
            for (int i = dimension-1; i >= 0; i--) {
                for (int j = 0; j < dimension; j++)
                    moveVertical(i, j, DOWN);
            }
            break;
        case LEFT:
            for (int i = 0; i < dimension; i++) {
                for (int j = 0; j < dimension; j++)
                    moveHorizontal(i, j, LEFT);
            }
            break;
        case RIGHT:
            for (int i = 0; i < dimension; i++) {
                for (int j = dimension-1; j >= 0; j--)
                    moveHorizontal(i, j, RIGHT);
            }
            break;
    }

    if (isChanged(pre_move_board)) {
        QVector<int> newpos = freePosition();
        board[newpos[0]][newpos[1]] = new Tile();
    }

    notifyObs();
}

void Board::reset() {
    lastPoints = 0;
    lastCollision = false;
    for (int i = 0; i < dimension ; i++) {
```

```
        for (int j = 0; j < dimension; j++) {
            delete board[i][j];
            board[i][j] = NULL;
        }
    }

    QVector<int> start = freePosition();
    board[start[0]][start[1]] = new Tile();

    start = freePosition();
    board[start[0]][start[1]] = new Tile();

}

int Board::getDimension() const {
    return dimension;
}

int Board::getPoints() const {
    return lastPoints;
}

QVector<int> Board::freePosition() {
    QVector<int> position;
    if (isFull()) {
        position.append(-1);
        position.append(-1);
    }
    else {
        int i, j;

        do {
            i = rand() % dimension;
            j = rand() % dimension;
        } while (board[i][j] != NULL);

        position.append(i);
        position.append(j);
    }

    return position;
}

void Board::moveHorizontal(int i, int j, DIRECTION dir) {
    if (board[i][j] != NULL) {
        bool isCollision = false;
        int newj;

        if (dir == RIGHT)
            newj = j + 1;
```

```
            else
                newj = j - 1;

            while (isInBounds(i,newj) && board[i][newj] == NULL) {
                if (dir == RIGHT)
                    newj++;
                else
                    newj--;
            }


            if (!isInBounds(i, newj)) {
                if (dir == RIGHT)
                    board[i][dimension-1] = board[i][j];
                else
                    board[i][0] = board[i][j];
            }
            else {
                if    (board[i][newj]->getNumber()    ==    board[i][j]-
>getNumber()) {
                    board[i][newj]->upgrade();
                    isCollision = true;
                    lastPoints += board[i][newj]->getNumber();
                }
                else {
                    if (dir == RIGHT)
                        board[i][newj-1] = board[i][j];
                    else
                        board[i][newj+1] = board[i][j];
                }
            }

            if ((dir == RIGHT && newj-1 != j) || (dir == LEFT && newj+1
!= j) || isCollision)
                board[i][j] = NULL;

            if (isCollision)
                lastCollision = true;
        }
    }

    void Board::moveVertical(int i, int j, DIRECTION dir) {
        if (board[i][j] != NULL) {
            bool isCollision = false;
            int newi;

            if (dir == UP)
                newi = i - 1;
            else
                newi = i + 1;
```

```
            while (isInBounds(newi, j) && board[newi][j] == NULL) {
                if (dir == UP)
                    newi--;
                else
                    newi++;
            }


            if (!isInBounds(newi, j)) {
                if (dir == UP)
                    board[0][j] = board[i][j];
                else
                    board[dimension-1][j] = board[i][j];
            }
            else {
                if    (board[newi][j]->getNumber()    ==    board[i][j]-
>getNumber()) {
                    board[newi][j]->upgrade();
                    isCollision = true;
                    lastPoints += board[newi][j]->getNumber();
                }
                else {
                    if (dir == UP)
                        board[newi+1][j] = board[i][j];
                    else
                        board[newi-1][j] = board[i][j];
                }
            }

            if ((dir == UP && newi+1 != i) || (dir == DOWN && newi-1 !=
i) || isCollision)
                board[i][j] = NULL;

            if (isCollision)
                lastCollision = true;
        }
    }

    bool Board::isChanged(Board &brd) const {
        if (dimension != brd.dimension)
            return false;

        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                if (((board[i][j] == NULL && brd.board[i][j] != NULL) ||
                    (board[i][j] != NULL && brd.board[i][j] == NULL)) ||
                     ((board[i][j] != NULL && brd.board[i][j] != NULL)
&&
```

```
                    board[i][j]->getNumber()     !=     brd.board[i][j]-
>getNumber()))
                    return true;
            }
        }

        return false;
    }

    bool Board::isMovePossible() const {
        if (isFull()) {
            Board newBoard(*this);

            newBoard.move(UP);
            if (isChanged(newBoard))
                return true;

            newBoard.move(DOWN);
            if (isChanged(newBoard))
                return true;

            newBoard.move(LEFT);
            if (isChanged(newBoard))
                return true;

            newBoard.move(RIGHT);
            if (isChanged(newBoard))
                return true;

            return false;
        }
        else
            return true;
    }

    bool Board::isFull() const {
        bool flag = true;

        for (int i = 0; i < dimension; i++) {
            for (int j = 0; j < dimension; j++) {
                if (board[i][j] == NULL)
                    flag = false;
            }
        }

        return flag;
    }

    bool Board::isCollision() const {
        return lastCollision;
```

```
}

bool Board::isInBounds(int i, int j) {
    return (i >= 0 && j >= 0 && i < dimension && j < dimension);
}
```

Файл Headerfile.h:

```
#ifndef HEADERFILE_H
#define HEADERFILE_H

#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <cstring>
#include <vector>
#include <string>

using namespace std;

#endif // HEADERFILE_H
```

Файл main.cpp:

```
#include "QHeaderfile.h"
#include "Headerfile.h"
#include "QGame.h"

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    app.setWindowIcon(QIcon(":/img/num2048.ico"));

    srand(time(NULL));
    QGame *game = new QGame();
    game->show();
    game->drawMenu();

    return app.exec();
}
```

Файл Observer.h:

```
#ifndef OBSERVER_H
#define OBSERVER_H

class Observer {
public:
```

```
    Observer();

    virtual void notify() = 0;

};

#endif // OBSERVER_H
```

Файл Observer.cpp:

```
#include "Observer.h"

Observer::Observer() {

}
```

Файл QBest.h:

```
#ifndef QBEST_H
#define QBEST_H

#include "QHeaderfile.h"
#include "Headerfile.h"
#include "Game.h"

class Game;

class QBest : public QWidget {
    Q_OBJECT

public:
    QBest(QWidget *parent = 0);
    QString getBest();
    void setBest(QString count);

private:
    Game *game;
    QVBoxLayout *mainLayout;
    QLabel *title, *count;
    string line;
    int curr;
};


#endif // QBEST_H
```

Файл QBest.cpp:

```
#include "QBest.h"

QBest::QBest(QWidget *parent) : QWidget(parent) {
    mainLayout = new QVBoxLayout;
    setLayout(mainLayout);

    title = new QLabel(QString("        BEST        "));
    title->setAlignment(Qt::AlignCenter);
    QFont font1("Ubuntu", 16, QFont::Bold);
    title->setFont(font1);

    ifstream fin(":/output/best", ios::in);
    getline(fin, line);
    curr = (int)line[0];

    count = new QLabel(QString("%1").fromStdString(line));
    count->setAlignment(Qt::AlignCenter);
    QFont font2("Ubuntu", 24, QFont::Bold);
    count->setFont(font2);

    title->setStyleSheet("QLabel   {   background:   rgb(139,115,85);
color: rgb(255,255,255); }");
    count->setStyleSheet("QLabel   {   background:   rgb(139,115,85);
color: rgb(255,255,255); }");

    setStyleSheet("QBest { border-radius: 10px; }");
    mainLayout->addWidget(title);
    mainLayout->addWidget(count);
    mainLayout->setSpacing(0);
}

QString QBest::getBest() {
    return count->text();
}

void QBest::setBest(QString record) {
    int curr = record.toInt();
    ofstream fout(":/output/best", ios::out);
    fout << curr;
    count->setText(record);
}
```

Файл QBoard.h:

```
#ifndef QBOARD_H
#define QBOARD_H
```

```
#include "QHeaderfile.h"
#include "Headerfile.h"
#include "Observer.h"
#include "QHead.h"
#include "QHint.h"
#include "QBest.h"
#include "QScore.h"
#include "QNewbutton.h"
#include "QGameover.h"
#include "QWinning.h"

class QTile;
class QHint;
class QBest;
class QScore;
class QGameOver;
class QWinning;

class QBoard : public QWidget, public Observer {
    Q_OBJECT

public:
    QBoard(int version, int dimension, int target, QWidget *parent =
0);
    void notify();
    bool getFinished();

    Game *game;

private:
    QVector< QVector<QTile*> > playerBoard;
    QGridLayout *mainLayout;
    QGridLayout *board;
    QLabel *gameTitle;
    QHead *head;
    QHint *hint;
    QScore *score;
    QBest *best;
    QNewButton *reset;
    QGameOver gameOver;
    QWinning winning;
    int currVersion;
    int currDimension;
    bool finished;

    void drawBoard(int version, int dimension);

protected:
    void keyPressEvent(QKeyEvent *event);
```

```
    signals:

    private slots:
        void newGame();
        void resetGame();

    };

    #endif // QGAMEBOARD_H
```

Файл QBoard.cpp:

```
    #include "QHeaderfile.h"
    #include "QBoard.h"
    #include "QTile.h"
    #include "QButton.h"
    #include "QNewbutton.h"
    #include "QResetbutton.h"
    #include "Board.h"
    #include "Tile.h"
    #include "Game.h"

    QBoard::QBoard(int version, int dimension, int target, QWidget
*parent) : QWidget(parent) {
        setFixedSize(750, 900);

        currVersion = version;
        currDimension = dimension;

        mainLayout = new QGridLayout();
        setLayout(mainLayout);
        setSizePolicy(QSizePolicy(QSizePolicy::Fixed,
QSizePolicy::Fixed));

        board = NULL;

        // create the game and register as observer
        game = new Game(dimension, target);
        game->registerObs(this);

        // create the gui board and draw it
        playerBoard.resize(game->getGameBoard()->getDimension());

        for (int i = 0; i < game->getGameBoard()->getDimension(); i++)
            playerBoard[i].resize(game->getGameBoard()->getDimension());

        for (int i = 0; i < game->getGameBoard()->getDimension(); i++) {
            for (int j = 0; j < game->getGameBoard()->getDimension();
j++)
```

```
                playerBoard[i][j] = NULL;
            }

            head = new QHead(version, target);
            mainLayout->addWidget(head, 0, 0, 1, 1);

            best = new QBest();
            mainLayout->addWidget(best, 0, 1, 1, 1);

            score = new QScore();
            mainLayout->addWidget(score, 0, 2, 1, 1);

            hint = new QHint(target);
            mainLayout->addWidget(hint, 1, 0, 1, 1);

            reset = new QNewButton();
            mainLayout->addWidget(reset, 1, 2, 1, 1);

            drawBoard(version, dimension);

            // style sheet of the board
            setStyleSheet("QBoard { background-color: rgb(187,173,160); }");

            connect(reset, SIGNAL(clicked()), this, SLOT(resetGame()));
            connect(gameOver.getResetBtn(),     SIGNAL(clicked()),     this,
SLOT(resetGame()));
            connect(winning.getResetBtn(),     SIGNAL(clicked()),     this,
SLOT(resetGame()));
        }

        void QBoard::keyPressEvent(QKeyEvent *event) {
            switch (event->key()) {
                case Qt::Key_Up:
                    game->playerMove(UP);
                    break;
                case Qt::Key_Left:
                    game->playerMove(LEFT);
                    break;
                case Qt::Key_Right:
                    game->playerMove(RIGHT);
                    break;
                case Qt::Key_Down:
                    game->playerMove(DOWN);
                    break;
            }
        }

        void QBoard::notify() {
            if (game->isGameOver())
                gameOver.show();
```

```
        if (game->isFinished())
            winning.show();

        QString getBest = best->getBest();
        int temp = getBest.toInt();
        if (game->isFinished()) {
            QString getCurr = score->getScore();
            int curr = getCurr.toInt();
            if (curr > temp)
                best->setBest(QString("%1").arg(curr));
            score->setScore(QString("%1").arg(game->getScore()));
        }
        else {
            best->setBest(QString("%1").arg(temp));
            score->setScore(QString("%1").arg(game->getScore()));
        }

        drawBoard(currVersion, currDimension);
    }

    void QBoard::drawBoard(int version, int dimension) {
        delete board;
        board = new QGridLayout();
        board->setSizeConstraint(QLayout::SetFixedSize);
        for (int i = 0; i < game->getGameBoard()->getDimension(); i++) {
            for (int j = 0; j < game->getGameBoard()->getDimension();
j++) {
                delete playerBoard[i][j];
                playerBoard[i][j] = new QTile(game->getGameBoard()-
>getTile(i, j));
                board->addWidget(playerBoard[i][j], i, j);
                playerBoard[i][j]->drawTile(version, dimension);
            }
        }
        mainLayout->addLayout(board, 2, 0, 1, 3);
    }

    bool QBoard::getFinished() {
        return game->isFinished();
    }

    void QBoard::newGame() {
        game->restart();
        best->setBest(QString("%1").arg(game->getBest()));
        score->setScore(QString("%1").arg(game->getScore()));
        drawBoard(currVersion, currDimension);
        gameOver.hide();
    }

    void QBoard::resetGame() {
```

```
        game->restart();
        best->setBest(QString("%1").arg(game->getBest()));
        score->setScore(QString("%1").arg(game->getScore()));
        drawBoard(currVersion, currDimension);
        gameOver.hide();
    }
```

Файл QButton.h:

```
    #ifndef QBUTTON_H
    #define QBUTTON_H

    #include "QHeaderfile.h"

    class QButton : public QObject, public QGraphicsRectItem {
        Q_OBJECT

    public:
        QButton(QString name, QGraphicsItem *parent = 0);

        void mousePressEvent(QGraphicsSceneMouseEvent *event);
        void hoverEnterEvent(QGraphicsSceneHoverEvent *event);
        void hoverLeaveEvent(QGraphicsSceneHoverEvent *event);

    private:
        QGraphicsTextItem *text;

    signals:
        void clicked();

    public slots:

    };

    #endif // QBUTTON_H
```

Файл QButton.cpp:

```
    #include "QHeaderfile.h"
    #include "QButton.h"

    QButton::QButton(QString    name,    QGraphicsItem    *parent)    :
QGraphicsRectItem(parent) {
        setRect(0, 0, 200, 50);

        QBrush brush;
        brush.setStyle(Qt::SolidPattern);
```

```
    brush.setColor(Qt::darkGreen);
    setBrush(brush);

    text = new QGraphicsTextItem(name, this);
    QFont font("Ubuntu", 20, QFont::Bold);
    text->setFont(font);
    int pos_x = rect().width()/2 - text->boundingRect().width()/2;
    int pos_y = rect().height()/2 - text->boundingRect().height()/2;
    text->setPos(pos_x, pos_y);

    setAcceptHoverEvents(true);
}

void QButton::mousePressEvent(QGraphicsSceneMouseEvent *event) {
    emit clicked();
}

void QButton::hoverEnterEvent(QGraphicsSceneHoverEvent *event) {
    QBrush brush;
    brush.setStyle(Qt::SolidPattern);
    brush.setColor(Qt::red);
    setBrush(brush);
}

void QButton::hoverLeaveEvent(QGraphicsSceneHoverEvent *event) {
    QBrush brush;
    brush.setStyle(Qt::SolidPattern);
    brush.setColor(Qt::darkGreen);
    setBrush(brush);
}
```

Файл QGame.h:

```
#ifndef QGAME_H
#define QGAME_H

#include "QHeaderfile.h"
#include "Headerfile.h"
#include "QBoard.h"
#include "QGameover.h"
#include "QScore.h"
#include "QBest.h"
#include "Game.h"
#include "Observer.h"

class QBoard;
class QScore;
class QBest;
```

```
class QGameOver;
class Game;

class QGame : public QGraphicsView {
    Q_OBJECT

public:
    QGame(QWidget *parent = 0);
    void drawMenu();
    void drawOption();
    void drawGameOver();
    void setFinished();
    bool getFinished();

    QGraphicsScene *scene;

private:
    Game *game;
    QComboBox *combobox1, *combobox2, *combobox3;
    QGameOver gameOver;
    QBoard *board;
    QScore *score;
    QBest *best;
    int version;
    int dimension;
    int target;

public slots:
    void start();
    void play();
    void changeVersion(int version);
    void changeDimension(int dimension);
    void changeTarget(int target);
    void restart();

};

#endif // QGAME_H
```

Файл QGame.cpp:

```
#include "QHeaderfile.h"
#include "QButton.h"
#include "QBoard.h"
#include "QGame.h"

QGame::QGame(QWidget *parent) : QGraphicsView(parent) {
    setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
    setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
```

```
        setFixedSize(800, 950);

        scene = new QGraphicsScene();
        scene->setSceneRect(0, 0, 750, 900);
        setScene(scene);
        setStyleSheet("QGame { background: rgb(187,173,160); }");
    }

    void QGame::drawMenu() {
        QGraphicsTextItem            *title            =            new
QGraphicsTextItem(QString("2048"));
        QFont font1("Ubuntu", 100, QFont::Bold);
        title->setFont(font1);
        int pos_x = this->width()/2 - title->boundingRect().width()/2;
        int pos_y = 50;
        title->setPos(pos_x, pos_y);
        scene->addItem(title);

        QGraphicsTextItem *hint1 = new QGraphicsTextItem(QString("HOW TO
PLAY:"));
        QFont font2("Ubuntu", 22, QFont::Bold);
        hint1->setFont(font2);
        pos_x = 150;
        pos_y = 250;
        hint1->setPos(pos_x, pos_y);
        scene->addItem(hint1);

        QGraphicsTextItem *hint2 = new QGraphicsTextItem(QString("- Use
your arrow keys to move the tiles."));
        QFont font3("Ubuntu", 18, QFont::Bold);
        hint2->setFont(font3);
        pos_x = 150;
        pos_y = 300;
        hint2->setPos(pos_x, pos_y);
        scene->addItem(hint2);

        QGraphicsTextItem *hint3 = new QGraphicsTextItem(QString("- When
two tiles with the same number touch,\n   they merge into one!"));
        hint3->setFont(font3);
        pos_x = 150;
        pos_y = 350;
        hint3->setPos(pos_x, pos_y);
        scene->addItem(hint3);

        QButton *play = new QButton(QString("Start !"));
        int btn1_x = this->width()/2 - play->boundingRect().width()/2;
        int btn1_y = 475;
        play->setPos(btn1_x, btn1_y);
        connect(play, SIGNAL(clicked()), this, SLOT(start()));
        scene->addItem(play);
```

```
        QButton *quit = new QButton(QString("Quit !"));
        int btn2_x = this->width()/2 - quit->boundingRect().width()/2;
        int btn2_y = 550;
        quit->setPos(btn2_x, btn2_y);
        connect(quit, SIGNAL(clicked()), this, SLOT(close()));
        scene->addItem(quit);
    }

    void QGame::drawOption() {
        QGraphicsTextItem            *title            =            new
QGraphicsTextItem(QString("Option"));
        QFont font1("Ubuntu", 100, QFont::Bold);
        title->setFont(font1);
        int pos_x = this->width()/2 - title->boundingRect().width()/2;
        int pos_y = 50;
        title->setPos(pos_x, pos_y);
        scene->addItem(title);

        QGraphicsTextItem           *option1           =            new
QGraphicsTextItem(QString("Version: "));
        QFont font2("Ubuntu", 20, QFont::Bold);
        option1->setFont(font2);
        pos_x = 225;
        pos_y = 300;
        option1->setPos(pos_x, pos_y);
        scene->addItem(option1);

        combobox1 = new QComboBox();
        QFont font3("Ubuntu", 16, QFont::Bold);
        combobox1->setFont(font3);
        combobox1->setGeometry(390, 300, 200, 40);
        combobox1->setEditable(false);
        combobox1->insertItem(0, "Numbers");
        combobox1->insertItem(1, "Letters");
        scene->addWidget(combobox1);

        QGraphicsTextItem           *option3           =            new
QGraphicsTextItem(QString("Target: "));
        option3->setFont(font2);
        pos_x = 225;
        pos_y = 500;
        option3->setPos(pos_x, pos_y);
        scene->addItem(option3);

        combobox3 = new QComboBox();
        combobox3->setFont(font3);
        combobox3->setGeometry(390, 500, 200, 40);
        combobox3->setEditable(false);
        combobox3->insertItem(0, "2048");
```

```
        combobox3->insertItem(1, "4096");
        scene->addWidget(combobox3);

        QGraphicsTextItem           *option2           =           new
QGraphicsTextItem(QString("Dimension: "));
        option2->setFont(font2);
        pos_x = 225;
        pos_y = 400;
        option2->setPos(pos_x, pos_y);
        scene->addItem(option2);

        combobox2 = new QComboBox();
        combobox2->setFont(font3);
        combobox2->setGeometry(390, 400, 200, 40);
        combobox2->setEditable(false);
        combobox2->insertItem(0, "3");
        combobox2->insertItem(1, "4");
        combobox2->insertItem(2, "5");
        combobox2->insertItem(3, "6");
        combobox2->insertItem(4, "7");
        combobox2->insertItem(5, "8");
        scene->addWidget(combobox2);

        QButton *play = new QButton(QString("Play !"));
        int btn1_x = this->width()/2 - play->boundingRect().width()/2;
        int btn1_y = 600;
        play->setPos(btn1_x, btn1_y);
        connect(play, SIGNAL(clicked()), this, SLOT(play()));
        scene->addItem(play);

        connect(combobox1,         SIGNAL(activated(int)),         this,
SLOT(changeVersion(int)));
        connect(combobox2,         SIGNAL(activated(int)),         this,
SLOT(changeDimension(int)));
        connect(combobox3,         SIGNAL(activated(int)),         this,
SLOT(changeTarget(int)));
    }

    void QGame::drawGameOver() {
        QGraphicsTextItem *title = new QGraphicsTextItem(QString("Game
over!"));
        QFont font1("Ubuntu", 100, QFont::Bold);
        title->setFont(font1);
        title->setPos(400, 350);
        scene->addItem(title);

        score = new QScore();
        score->setGeometry(400, 400, 100, 150);
        scene->addWidget(score);
```

```
    best = new QBest();
    best->setGeometry(600, 400, 100, 150);
    scene->addWidget(best);

    QButton *again = new QButton(QString("Try again !"));
    int btn1_x = this->width()/2 - again->boundingRect().width()/2;
    int btn1_y = 475;
    again->setPos(btn1_x, btn1_y);
    connect(again, SIGNAL(clicked()), this, SLOT(restart()));
    scene->addItem(again);

    QButton *quit = new QButton(QString("Quit !"));
    int btn2_x = this->width()/2 - quit->boundingRect().width()/2;
    int btn2_y = 550;
    quit->setPos(btn2_x, btn2_y);
    connect(quit, SIGNAL(clicked()), this, SLOT(close()));
    scene->addItem(quit);
}

void QGame::start() {
    scene->clear();
    drawOption();
}

void QGame::play() {
    scene->clear();

    if (version > 1 || version < 0)
        version = 0;

    if (dimension == 0 || dimension > 8)
        dimension = 3;

    if (target > 4096 || target < 4096)
        target = 2048;

    board = new QBoard(version, dimension, target);
    scene->addWidget(board);
}

void QGame::changeVersion(int v) {
    version = v;
}

void QGame::changeDimension(int d) {
    dimension = d + 3;
}

void QGame::changeTarget(int t) {
    target = (t + 1) * 2048;
```

```
}

void QGame::restart() {
    scene->clear();
    start();
}
```

Файл QGameover.h:

```
#ifndef QGAMEOVER_H
#define QGAMEOVER_H

#include "QHeaderfile.h"

class QResetButton;

class QGameOver : public QWidget {
    Q_OBJECT

public:
    QGameOver(QWidget *parent = 0);
    QResetButton* getResetBtn() const;

private:
    QResetButton *reset;

signals:

public slots:

};

#endif // QGAMEOVER_H
```

Файл QGameover.cpp:

```
#include "QHeaderfile.h"
#include "QGameover.h"
#include "QResetbutton.h"

QGameOver::QGameOver(QWidget *parent) : QWidget(parent) {
    setStyleSheet("QGameOverWindow { background: rgb(237,224,200);
}");
    setFixedSize(425, 205);

    QVBoxLayout *layout = new QVBoxLayout();
```

```
        QLabel *gameover = new QLabel("Game Over!", this);
        gameover->setStyleSheet("QLabel { color: rgb(119,110,101); font:
40pt; font: bold; } ");
        gameover->setGeometry(60, 30, 300, 50);

        reset = new QResetButton(this);
        reset->setFixedHeight(50);
        reset->setFixedWidth(200);
        reset->setGeometry(120, 120, 50, 100);

        layout->insertWidget(0, gameover, 0, Qt::AlignCenter);
        layout->insertWidget(1, reset, 0, Qt::AlignCenter);
    }

    QResetButton* QGameOver::getResetBtn() const {
        return reset;
    }
```

Файл QHead.h:

```
#ifndef QHEAD_H
#define QHEAD_H

#include "Headerfile.h"
#include "QHeaderfile.h"
#include "Game.h"

class Game;

class QHead : public QWidget {
    Q_OBJECT

public:
    explicit QHead(int version, int target, QWidget *parent = 0);

private:
    Game *game;
    QHBoxLayout *mainLayout;
    QLabel *gameTitle;
    string head;

};

#endif // QHEAD_H
```

Файл QHead.cpp:

```
#include "QHeaderfile.h"
#include "Headerfile.h"
#include "QHead.h"

QHead::QHead(int version, int target, QWidget *parent) :
QWidget(parent) {
        // create the main layout
        mainLayout = new QHBoxLayout();
        mainLayout->setSizeConstraint(QLayout::SetFixedSize);
        setLayout(mainLayout);

        // Gametitle
        if (target == 2048) {
            if (version == 0)
                head = "Game 2048";
            else
                head = "Game 2048";
        }
        else {
            if (version == 0)
                head = "Game 4096";
            else
                head = "Game 4096";
        }

        gameTitle = new QLabel(QString::fromStdString(head));
        QFont font("Ubuntu", 50, QFont::Bold);
        gameTitle->setFont(font);
        gameTitle->setStyleSheet("QLabel { color: rgb(48,48,48); }");
        mainLayout->addWidget(gameTitle);
    }
```

Файл QHeaderfile.h:

```
#ifndef QHEADERFILE_H
#define QHEADERFILE_H

#include <QWidget>
#include <QMainWindow>
#include <QApplication>

#include <QFrame>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QGridLayout>

#include <QPushButton>
#include <QLabel>
```

```
#include <QLineEdit>
#include <QComboBox>
#include <QIcon>
#include <QPixmap>
#include <QFont>

#include <QKeyEvent>
#include <QMouseEvent>
#include <QResizeEvent>
#include <QGraphicsSceneMouseEvent>

#include <QString>
#include <QVector>

#include <QBrush>
#include <QGraphicsView>
#include <QGraphicsScene>
#include <QGraphicsTextItem>
#include <QGraphicsRectItem>
#include <QGraphicsDropShadowEffect>

#include <QDebug>

#endif // QHEADERFILE_H
```

Файл QHint.h:

```
#ifndef QHINT_H
#define QHINT_H

#include "QHeaderfile.h"
#include "QBoard.h"

class QHint : public QWidget {
    Q_OBJECT

public:
    QHint(int target, QWidget *parent = 0);

private:
    QHBoxLayout *mainLayout;
    QLabel *hint;

signals:

public slots:

};
```

```
#endif // QHINT_H
```

Файл QHint.cpp:

```
#include "QHeaderfile.h"
#include "QHint.h"

QHint::QHint(int target, QWidget *parent) : QWidget(parent) {
    // create the main layout
    mainLayout = new QHBoxLayout();
    mainLayout->setSizeConstraint(QLayout::SetFixedSize);
    setLayout(mainLayout);

    // Hint
    if (target == 2048)
        hint = new QLabel("Join the numbers and get to the 2048
tile!    ");
    else
        hint = new QLabel("Join the numbers and get to the 4096
tile!    ");

    QFont font1("Ubuntu", 18, QFont::Bold);
    hint->setFont(font1);
    hint->setStyleSheet("QLabel { color: rgb(119,110,101); border-
radius: 5px; }");

    mainLayout->addWidget(hint);
}
```

Файл QNewbutton.h:

```
#ifndef QNEWBUTTON_H
#define QNEWBUTTON_H

#include "QHeaderfile.h"

class QNewButton : public QLabel, public QGraphicsRectItem {
    Q_OBJECT

public:
    QNewButton(QWidget *parent = 0);

signals:
    void clicked();
```

```
public slots:

protected:
    void mousePressEvent(QMouseEvent *event);
    void hoverEnterEvent(QHoverEvent *event);
    void hoverLeaveEvent(QHoverEvent *event);

};

#endif // QNEWBUTTON_H
```

Файл QNewbutton.cpp:

```
#include "QNewbutton.h"

QNewButton::QNewButton(QWidget* parent) : QLabel(parent) {
    setText("New Game");

    QFont font("Ubuntu", 20, QFont::Bold);
    setFont(font);

    setFixedHeight(50);
    setFixedWidth(150);
    setAlignment(Qt::AlignCenter);

    setStyleSheet("QNewButton { background: rgb(92,64,51); color:
rgb(255,255,255); border-radius: 5px; }");
}

void QNewButton::mousePressEvent(QMouseEvent *event) {
    emit clicked();
}

void QNewButton::hoverEnterEvent(QHoverEvent *event) {
    QBrush brush;
    brush.setStyle(Qt::SolidPattern);
    brush.setColor(Qt::cyan);
    setBrush(brush);
}

void QNewButton::hoverLeaveEvent(QHoverEvent *event) {
    QBrush brush;
    brush.setStyle(Qt::SolidPattern);
    brush.setColor(Qt::darkCyan);
    setBrush(brush);
}
```

Файл QResetbutton.h:

```
#ifndef QRESETBUTTON_H
#define QRESETBUTTON_H

#include "QHeaderfile.h"

class QResetButton : public QLabel {
    Q_OBJECT

public:
    QResetButton(QWidget *parent = 0);

signals:
    void clicked();

public slots:

protected:
    void mousePressEvent(QMouseEvent *event);

};

#endif // QRESETBUTTON_H
```

Файл QResetbutton.cpp:

```
#include "QResetbutton.h"

QResetButton::QResetButton(QWidget* parent) : QLabel(parent) {
    setText("New Game");

    QFont font("Ubuntu", 20, QFont::Bold);
    setFont(font);

    setAlignment(Qt::AlignCenter);

    setStyleSheet("QResetButton { background: rgb(92,64,51); color:
rgb(255,255,255); border-radius: 5px; }");
}

void QResetButton::mousePressEvent(QMouseEvent* event) {
    emit clicked();
}
```

Файл QScore.h:

```
#ifndef QSCORE_H
#define QSCORE_H

#include "QHeaderfile.h"

class QScore : public QWidget {
    Q_OBJECT

public:
    QScore(QWidget *parent = 0);
    QString getScore();
    void setScore(QString count);

private:
    QVBoxLayout *mainLayout;
    QLabel *title, *count;

};

#endif // QSCORE_H
```

Файл QScore.cpp:

```
#include "QScore.h"

QScore::QScore(QWidget *parent) : QWidget(parent) {
    mainLayout = new QVBoxLayout;
    setLayout(mainLayout);

    title = new QLabel(QString("      SCORE      "));
    title->setAlignment(Qt::AlignCenter);
    QFont font1("Ubuntu", 16, QFont::Bold);
    title->setFont(font1);

    count = new QLabel("0");
    count->setAlignment(Qt::AlignCenter);
    QFont font2("Ubuntu", 24, QFont::Bold);
    count->setFont(font2);

    title->setStyleSheet("QLabel   {   background:   rgb(139,115,85);
color: rgb(255,255,255); }");
    count->setStyleSheet("QLabel   {   background:   rgb(139,115,85);
color: rgb(255,255,255); }");

    setStyleSheet("QBest { border-radius: 10px; }");
    mainLayout->addWidget(title);
    mainLayout->addWidget(count);
    mainLayout->setSpacing(0);
}
```

```
QString QScore::getScore() {
    return count->text();
}

void QScore::setScore(QString record) {
    count->setText(record);
}
```

Файл QTile.h:

```
#ifndef QTILE_H
#define QTILE_H

#include "QHeaderfile.h"
#include "Headerfile.h"

class Tile;

class QTile : public QLabel {
    Q_OBJECT

public:
    QTile(const QString &text);
    QTile(Tile *tile);

    void drawTile(int version, int dimension);

private:
    Tile *tile;
    qreal scale;

protected:

signals:

public slots:

};

#endif // QTILE_H
```

Файл QTile.cpp:

```
#include "QHeaderfile.h"
#include "Headerfile.h"
#include "QTile.h"
```

```
#include "Tile.h"

QTile::QTile(Tile *tile) {
    setAlignment(Qt::AlignCenter);
    this->tile = tile;
}

void QTile::drawTile(int version, int dimension) {
    if (tile == NULL) {
        setText("");
        setStyleSheet("QTile { background: rgb(204,192,179); border-
radius: 10px; }");
    }
    else {
        if (version)
            setText(QString::fromStdString(tile->getWord(tile-
>getNumber())));
        else
            setText(QString::number(tile->getNumber()));

        switch (dimension) {
        case 3: {
            QFont font1("Ubuntu", 60);
            setFont(font1);
            break;
        }
        case 4: {
            QFont font2("Ubuntu", 55);
            setFont(font2);
            break;
        }
        case 5: {
            QFont font3("Ubuntu", 45);
            setFont(font3);
            break;
        }
        case 6: {
            QFont font4("Ubuntu", 40);
            setFont(font4);
            break;
        }
        case 7: {
            QFont font5("Ubuntu", 30);
            setFont(font5);
            break;
        }
        case 8: {
            QFont font6("Ubuntu", 25);
            setFont(font6);
            break;
```

```
            }
            }

            switch (tile->getNumber()) {
            case 2:
                setStyleSheet("QTile  {  background:  rgb(238,228,218);
color: rgb(119,110,101); border-radius: 10px; }");
                break;
            case 4:
                setStyleSheet("QTile  {  background:  rgb(237,224,200);
color: rgb(119,110,101); border-radius: 10px; }");
                break;
            case 8:
                setStyleSheet("QTile  {  background:  rgb(242,177,121);
color: rgb(255,255,255); border-radius: 10px; }");
                break;
            case 16:
                setStyleSheet("QTile  {  background:  rgb(245,150,100);
color: rgb(255,255,255); border-radius: 10px; }");
                break;
            case 32:
                setStyleSheet("QTile  {  background:  rgb(245,125,95);
color: rgb(255,255,255); border-radius: 10px; }");
                break;
            case 64:
                setStyleSheet("QTile  {  background:  rgb(245,95,60);
color: rgb(255,255,255); border-radius: 10px; }");
                break;
            case 128:
                setStyleSheet("QTile  {  background:  rgb(237,207,114);
color: rgb(255,255,255); border-radius: 10px; }");
                break;
            case 256:
                setStyleSheet("QTile  {  background:  rgb(237,99,97);
color: rgb(255,255,255); border-radius: 10px; }");
                break;
            case 512:
                setStyleSheet("QTile  {  background:  rgb(237,97,130);
color: rgb(255,255,255); border-radius: 10px; }");
                break;
            case 1024:
                setStyleSheet("QTile  {  background:  rgb(204,97,237);
color: rgb(255,255,255); border-radius: 10px; }");
                break;
            case 2048:
                setStyleSheet("QTile  {  background:  rgb(134,97,237);
color: rgb(255,255,255); border-radius: 10px; }");
                break;
            case 4096:
```

```
                setStyleSheet("QTile   {   background:   rgb(99,97,237);
color: rgb(255,255,255); border-radius: 10px; }");
                break;
            default:
                setStyleSheet("QTile  {  background:  rgb(238,228,218);
color: rgb(119,110,101); border-radius: 10px; }");
                break;
            }
        }
    }
```

Файл QWinning.h:

```
#ifndef QWINNING_H
#define QWINNING_H

#include "QHeaderfile.h"
#include "QBest.h"
#include "QScore.h"

class QResetButton;
class QBest;
class QScore;

class QWinning : public QWidget {
    Q_OBJECT

public:
    QWinning(QWidget *parent = 0);
    QResetButton* getResetBtn() const;

private:
    QResetButton *reset;
    QBest *best;
    QScore * score;

signals:

public slots:

};

#endif // QWINNING_H
```

Файл QWinning.cpp:

```
#include "QHeaderfile.h"
```

```
#include "QWinning.h"
#include "QResetbutton.h"

QWinning::QWinning(QWidget *parent) : QWidget(parent) {
    setStyleSheet("QGameOverWindow  {  background:  rgb(237,224,200);
}");
    setFixedSize(425, 205);

    QGridLayout *layout = new QGridLayout();

    QLabel *gameover = new QLabel("Winning !", this);
    gameover->setStyleSheet("QLabel { color: rgb(119,110,101); font:
40pt; font: bold; } ");
    gameover->setGeometry(100, 30, 300, 60);

    reset = new QResetButton(this);
    reset->setFixedHeight(50);
    reset->setFixedWidth(200);
    reset->setGeometry(120, 120, 50, 100);

    layout->addWidget(gameover, 0, 0);
    layout->addWidget(reset, 2, 0, 1, 2);
}

QResetButton* QWinning::getResetBtn() const {
    return reset;
}
```

Файл Subject.h:

```
#ifndef SUBJECT_H
#define SUBJECT_H

#include "Headerfile.h"

using namespace std;

class Observer;

class Subject {
public:
    Subject();

    void notifyObs();
    void registerObs(Observer *observer);

private:
    vector<Observer*> observers;
};
```

```
#endif // SUBJECT_H
```

Файл Subject.cpp:

```
#include "Subject.h"
#include "Observer.h"

Subject::Subject() {

}

void Subject::notifyObs() {
    for (Observer *obs : observers)
        obs->notify();
}

void Subject::registerObs(Observer *obs) {
    observers.push_back(obs);
}
```

Файл Tile.h:

```
#ifndef TILE_H
#define TILE_H

#include <Headerfile.h>

using namespace std;

struct VALUE {
    int number;
    string word;
};

class Tile {
public:
    Tile();
    Tile(const Tile &tile);
    Tile(int number);
    int getNumber();
    string getWord(int number);
    string match(int number);
    void upgrade();

private:
    struct VALUE value;

};
```

```
#endif // TILE_H
```

Файл Tile.cpp:

```
#include "Tile.h"

#define MULTIPLIER 2

Tile::Tile() {
    value.number = 2;
}

Tile::Tile(const Tile &other) {
    this->value.number = other.value.number;
}

Tile::Tile(int number) {
    this->value.number = number;
}

int Tile::getNumber() {
    return value.number;
}

string Tile::getWord(int number) {
    return match(number);
}

string Tile::match(int number) {
    switch(number) {
    case 2:
        value.number = 2;
        value.word = "Two";
        break;
    case 4:
        value.number = 4;
        value.word = "Four";
        break;
    case 8:
        value.number = 8;
        value.word = "eight ";
        break;
    case 16:
        value.number = 16;
        value.word = "sixteen";
        break;
    case 32:
        value.number = 32;
        value.word = "thirty two";
        break;
```

```
    case 64:
        value.number = 64;
        value.word = "sixty four";
        break;
    case 128:
        value.number = 128;
        value.word = "one hundred and twenty eight";
        break;
    case 256:
        value.number = 256;
        value.word = "two hundred fifty six";
        break;
    case 512:
        value.number = 512;
        value.word = "five hundred twelve";
        break;
    case 1024:
        value.number = 1024;
        value.word = "one thousand twenty four";
        break;
    case 2048:
        value.number = 2048;
        value.word = "two thousand forty-eight";
        break;
    case 4096:
        value.number = 4096;
        value.word = "four thousand ninety-six";
        break;
    }

    return value.word;
}

void Tile::upgrade() {
    value.number *= MULTIPLIER;
}
```