Файл battleShipCore.cpp:

```
#include "battleShipCore.h"
BattleShipCore::BattleShipCore(QObject *parent)
    : QObject (parent),
      m_change(false),
      m_turn(false) {
    m_pPlayerHuman = new Player(Settings::inst().playerName(),
this);
    m_pPlayerBot = new Bot(tr("COMPUTER"), this);
    m_pGameMapEditor = QSharedPointer<GameMapEditor>::create(510,
330);
    m_pTurnIndicator = new TurnIndicator(60, 60);
    m_pGameMapEditor->setShips(standartShips());
    QSharedPointer<GameMap>                mapHuman                =
QSharedPointer<GameMap>::create(330, 330, true);
    QSharedPointer<GameMap>                 mapBot                 =
QSharedPointer<GameMap>::create(330, 330);
    m_pPlayerHuman->setMap(mapHuman);
    m_pPlayerHuman->setShips(standartShips());
    m_pPlayerBot->setMap(mapBot);
    m_pPlayerBot->setShips(standartShips());
    m_timer.setInterval(Settings::inst().animationDelay() + 100);
    connect(mapBot.data(),         &GameMap::clicked,         this,
&BattleShipCore::turnHuman);
    connect(&m_timer,          &QTimer::timeout,          &m_loop,
&QEventLoop::quit); }
void BattleShipCore::setShipsFromEditorToPlayer() {
    m_pGameMapEditor->moveShipsOnGameMap(m_pPlayerHuman-
>gameMap(),
                                         m_pPlayerHuman->ships());
}
QVector<QSharedPointer<Ship> > BattleShipCore::standartShips() {
    QVector<int> shipsLength = {4, 3, 3, 2, 2, 2, 1, 1, 1, 1};
    QVector<QSharedPointer<Ship>> result(shipsLength.size());
    for (int i = 0; i < shipsLength.size(); ++i)
        result[i] = QSharedPointer<Ship>::create(shipsLength[i]);
    return result; }
bool BattleShipCore::isChange() const {
    return m_change; }
void BattleShipCore::setTurnInterval(int msec) {
    m_timer.setInterval(msec); }
void BattleShipCore::resetGame() {
    m_change = false;
    m_turn = false;
    m_pPlayerBot->reset();
    m_pPlayerHuman->reset();
    m_pTurnIndicator->reset(); }
QSharedPointer<GameMap> BattleShipCore::playerHumanMap() const {
    return m_pPlayerHuman->gameMap(); }
QSharedPointer<GameMap> BattleShipCore::playerBotMap() const {
    return  m_pPlayerBot->gameMap(); }
QSharedPointer<GameMapEditor> BattleShipCore::gameMapEditor() {
    return m_pGameMapEditor; }
TurnIndicator *BattleShipCore::turnIndicator() const {
    return m_pTurnIndicator; }
void   BattleShipCore::setRandShip(QSharedPointer<GameMap>   &map,
QVector<QSharedPointer<Ship> > &ships) {
    int x = 0, y = 0, dir = 0;
    for (int i = 0; i < 10; i++) {
```

```
        do {
            x = generateRandomNumber(0, g_MAP_SIZE - 1);
            y = generateRandomNumber(0, g_MAP_SIZE - 1);
            dir = generateRandomNumber(0, 1);
        } while (!setShip(map, ships[i], x, y, dir)); } }
bool     BattleShipCore::setShip(QSharedPointer<GameMap>     &map,
QSharedPointer<Ship> &ship, int x, int y, bool isHor) {
    bool correct = true;
    if (isHor ? x + ship->length() >= g_MAP_SIZE : y + ship->length()
>= g_MAP_SIZE) {
        return false;
    } else {
        for (int i = 0, j = 0; i < ship->length() && j < ship-
>length(); isHor ? i++ : j++) {
            if (map->cellStatus(x + i, y + j) != e_Status::Empty) {
                correct = false;
                break; } } }
    if (correct) {
        ship->setOrientation(isHor     ?     Ship::Horizontal     :
Ship::Vertical);
        for (int a = 0, b = 0; a < ship->length() && b < ship-
>length(); isHor ? a++ : b++) {
            for (int i = -1; i < 2; i++) {
                for (int j = -1; j < 2; j++) {
                    if (i == 0 && j == 0) {
                        map->setCellStatus(x   +   a,   y   +   b,
e_Status::Life);
                        map->setCellShip(x + a, y + b, ship);
                        ship->setCellCoord(a + b, x + a, y + b);
                    } else if (x + i + a < g_MAP_SIZE && x + i + a
>= 0 &&
                               y + j + b < g_MAP_SIZE && y + j + b >=
0 &&
                               map->cellStatus(x + i + a, y + j + b)
!= e_Status::Life) {
                        map->setCellStatus(x + i + a, y + j + b,
e_Status::NearbyShip); } } } }
        return true; }
    return false; }
void BattleShipCore::turnHuman(int x, int y) {
    if (!m_turn) {
        m_turn = true;
        m_change = true;
        switch(m_pPlayerBot->gameMap()->shot(x, y)) {
        case e_Status::Impossible:
        case e_Status::Hit:
            m_turn = false;
            return;
        case e_Status::Destroyed:
            m_pPlayerBot->gameMap()->setDestroyedArea(x, y);
            winnerChecker();
            m_turn = false;
            return;
        default:
            break; }
        m_pTurnIndicator->change(180, Qt::red);
        while (true) {
            switch (m_pPlayerBot->turn(m_pPlayerHuman)) {
            case e_Status::Miss:
```

```
                m_timer.start();
                m_loop.exec();
                m_pTurnIndicator->change(180, Qt::green);
                m_turn = false;
                return;
            case e_Status::Hit:
                m_timer.start();
                m_loop.exec();
                break;
            case e_Status::Destroyed:
                m_pPlayerHuman->gameMap()-
>setDestroyedArea(m_pPlayerBot->botX(),

m_pPlayerBot->botY());
                if (winnerChecker())
                    return;
                m_timer.start();
                m_loop.exec();
                break;
            default:
                break; } } } }
void BattleShipCore::generateBotMap() {
    m_pPlayerBot->reset();
    setRandShip(m_pPlayerBot->gameMap(), m_pPlayerBot->ships()); }
void          BattleShipCore::changeBotDifficulty(Bot::e_Difficulty
difficulty) {
    m_pPlayerBot->setDifficulty(difficulty); }
bool BattleShipCore::winnerChecker() {
    if (m_pPlayerHuman->isDead()) {
        m_change = false;
        emit endGame(m_pPlayerBot->name());
        return true; }
    if (m_pPlayerBot->isDead()) {
        m_change = false;
        emit endGame(m_pPlayerHuman->name());
        return true; }
    return false; }
```

Файл battleShipCore.h:

```
#ifndef BATTLESHIPCORE_H
#define BATTLESHIPCORE_H
#include <QtWidgets>
#include "utilities.h"
#include "settings.h"
#include "player.h"
#include "bot.h"
#include "gameMap.h"
#include "gameMapEditor.h"
#include "turnIndicator.h"
class BattleShipCore : public QObject {
    Q_OBJECT
public:
    BattleShipCore(QObject *parent = nullptr);
    void setShipsFromEditorToPlayer();
    QVector<QSharedPointer<Ship>> standartShips();
    bool isChange() const;
    void setTurnInterval(int msec);
    void resetGame();
    QSharedPointer<GameMap> playerHumanMap() const;
```

```
    QSharedPointer<GameMap> playerBotMap() const;
    QSharedPointer<GameMapEditor> gameMapEditor();
    TurnIndicator *turnIndicator() const;
    void         setRandShip(QSharedPointer<GameMap>         &map,
QVector<QSharedPointer<Ship>> &ships);
    bool setShip(QSharedPointer<GameMap> &map, QSharedPointer<Ship>
&ship, int x, int y, bool isHor);
signals:
    void endGame(QString winner);
public slots:
    void generateBotMap();
    void changeBotDifficulty(Bot::e_Difficulty difficulty);
    void turnHuman(int x, int y);
private:
    bool winnerChecker();
private:
    bool m_change;
    bool m_turn;
    QEventLoop m_loop;
    QTimer m_timer;
    Player *m_pPlayerHuman;
    Bot *m_pPlayerBot;
    TurnIndicator *m_pTurnIndicator;
    QSharedPointer<GameMapEditor> m_pGameMapEditor;
};
#endif // BATTLESHIPCORE_H
```

Файл battleShipView.cpp:

```
#include "battleShipView.h"
BattleShipView::BattleShipView(QWidget *parent)
    : QGraphicsView(parent) {
    m_pScene = new QGraphicsScene(Settings::inst().sceneRect());
    m_pBattleShipCore = new BattleShipCore(this);
    m_pScene->addItem(initMainMenu());
    m_pScene->addItem(initSinglePlayerMenu());
    m_pScene->addItem(initGameMenu());
    m_pScene-
>setBackgroundBrush(QImage(QStringLiteral(":/image/bg-image")));
    setScene(m_pScene);
    setMinimumSize(800, 600);
    setRenderHint(QPainter::Antialiasing, true);
    setCacheMode(QGraphicsView::CacheBackground);

setViewportUpdateMode(QGraphicsView::BoundingRectViewportUpdate);
    mainMenu.m_pMainMenu->setVisible(true);
    connect(m_pBattleShipCore,   &BattleShipCore::endGame,   this,
&BattleShipView::playerWins); }
BattleShipCore *BattleShipView::core() const {
    return m_pBattleShipCore; }
void BattleShipView::drawMainMenu(QGraphicsItem *clickedItem) {
    clickedItem->parentItem()->setVisible(false);
    mainMenu.m_pMainMenu->setVisible(true); }
void            BattleShipView::drawSinglePlayerMenu(QGraphicsItem
*clickedItem) {
    clickedItem->parentItem()->setVisible(false);
    singlePlayerMenu.m_pSinglePlayerMenu->setVisible(true);
    clickedSetRandomMode(); }
void BattleShipView::drawGame(QGraphicsItem *clickedItem) {
    if (!singlePlayerMenu.m_pGameMapEditor->isReady()) {
```

```
        QMessageBox::information(this, tr("Information"), tr("Not
all ships are set up!"));
        return; }
    clickedItem->parentItem()->setVisible(false);
    gameMenu.m_pGameMenu->setVisible(true);
    m_pBattleShipCore->setShipsFromEditorToPlayer();
    m_pBattleShipCore->generateBotMap(); }
void BattleShipView::clickedSetClearMode() {
    singlePlayerMenu.m_pSinglePlayerMenu->setX(-90);
    singlePlayerMenu.m_pSinglePlayerTitle->setX(90);
    singlePlayerMenu.m_pGameMapEditor->setClearMapMode();
    singlePlayerMenu. m_pGameMapEditor->update(); }
void BattleShipView::clickedSetRandomMode() {
    singlePlayerMenu.m_pSinglePlayerMenu->setX(0);
    singlePlayerMenu.m_pSinglePlayerTitle->setX(0);
    singlePlayerMenu.m_pGameMapEditor->setGenerateRandomMode();
    singlePlayerMenu.m_pGameMapEditor->update(); }
void BattleShipView::exitFromGameToMenu(QGraphicsItem *clickedItem)
{
    if (m_pBattleShipCore->isChange()) {
        auto answer = QMessageBox::question(this, tr("Exit to the
main menu"),
                                            tr("Are you sure you
want to go out?",
                                               "Go to the main
menu"));
        if (answer == QMessageBox::StandardButton::No)
            return; }
    m_pBattleShipCore->resetGame();
    drawMainMenu(clickedItem); }
void BattleShipView::playerWins(QString winnerName) {
    QMessageBox::information(this, tr("Win!"), tr("Player ") +
winnerName + tr(" is winner!"));
    exitFromGameToMenu(gameMenu.m_pGameMenu-
>childItems().first()); }
void BattleShipView::changeEvent(QEvent *event) {
      QWidget::changeEvent(event); }
bool BattleShipView::event(QEvent *event) {
    if (event->type() == SettingsChangeEvent::typeEvent()) {
        m_pBattleShipCore-
>setTurnInterval(Settings::inst().animationDelay());
        m_pBattleShipCore->turnIndicator()-
>setAnimationDelay(Settings::inst().animationDelay());
        return true; }
    return QGraphicsView::event(event); }
QGraphicsRectItem *BattleShipView::initMainMenu() {
    auto countElements = 4;
    auto buttonWidth = m_pScene->width() / 2;
    auto buttonHeigth = m_pScene->height() / 8;
    auto width = m_pScene->width() / 2 - buttonWidth / 2;
    auto height = m_pScene->height() / 2 - ((countElements + 2) *
buttonHeigth) / 2;
    mainMenu.m_pMainMenu = new QGraphicsRectItem();
    mainMenu.m_pMainMenu->setVisible(false);
    mainMenu.m_pMainMenuTitle                =                new
TextLabel(QApplication::applicationDisplayName());
    mainMenu.m_pMainMenuTitle->setSize(m_pScene->width(),
buttonHeigth);
    mainMenu.m_pMainMenuTitle->setPos(0, height - 50);
```

```
    mainMenu.m_pMainMenuTitle-
>setParentItem(mainMenu.m_pMainMenu);
    mainMenu.m_pButtonSinglePlayer   =   new   MenuButton(tr("SINGLE
PLAYER"));
    mainMenu.m_pButtonSinglePlayer->setSize(buttonWidth,
buttonHeigth);
    mainMenu.m_pButtonSinglePlayer->setPos(width,     height     +=
buttonHeigth);
    mainMenu.m_pButtonSinglePlayer-
>setParentItem(mainMenu.m_pMainMenu);
    mainMenu.m_pButtonMultiPlayer                =                new
MenuDisableButton(tr("MULTIPLAYER (SOON)"));
    mainMenu.m_pButtonMultiPlayer->setSize(buttonWidth,
buttonHeigth);
    mainMenu.m_pButtonMultiPlayer->setPos(width,     height     +=
buttonHeigth);
    mainMenu.m_pButtonMultiPlayer-
>setParentItem(mainMenu.m_pMainMenu);
    mainMenu.m_pButtonExit = new MenuButton(tr("EXIT"));
    mainMenu.m_pButtonExit->setSize(buttonWidth, buttonHeigth);
    mainMenu.m_pButtonExit->setPos(width, height += buttonHeigth);
    mainMenu.m_pButtonExit->setParentItem(mainMenu.m_pMainMenu);
    connect(mainMenu.m_pButtonSinglePlayer,   &MenuButton::clicked,
this, &BattleShipView::drawSinglePlayerMenu);
    connect(mainMenu.m_pButtonExit,   &MenuButton::clicked,   this-
>parentWidget(), &QWidget::close);
    return mainMenu.m_pMainMenu; }
QGraphicsRectItem *BattleShipView::initSinglePlayerMenu() {
    auto countElements = 5;
    auto buttonWidth = m_pScene->width() / 3;
    auto buttonHeigth = m_pScene->height() / 10;
    auto width = m_pScene->width() / 4 - buttonWidth / 2 + 20;
    auto height = m_pScene->height() / 2 - ((countElements + 2) *
buttonHeigth) / 2;
    singlePlayerMenu.m_pSinglePlayerMenu = new QGraphicsRectItem();
    singlePlayerMenu.m_pSinglePlayerMenu->setVisible(false);
    singlePlayerMenu.m_pSinglePlayerTitle          =          new
TextLabel(tr("SINGLE PALYER"));
    singlePlayerMenu.m_pSinglePlayerTitle->setSize(m_pScene-
>width(), buttonHeigth);
    singlePlayerMenu.m_pSinglePlayerTitle->setPos(0, height - 50);
    singlePlayerMenu.m_pSinglePlayerTitle-
>setParentItem(singlePlayerMenu.m_pSinglePlayerMenu);
    singlePlayerMenu.m_pButtonStartGame = new MenuButton(tr("START
GAME"));
    singlePlayerMenu.m_pButtonStartGame->setSize(buttonWidth,
buttonHeigth);
    singlePlayerMenu.m_pButtonStartGame->setPos(width,  height  +=
buttonHeigth);
    singlePlayerMenu.m_pButtonStartGame-
>setParentItem(singlePlayerMenu.m_pSinglePlayerMenu);
    singlePlayerMenu.m_pButtonSelectDifficulty        =        new
MenuSelectedButton(buttonWidth, buttonHeigth);
    singlePlayerMenu.m_pButtonSelectDifficulty-
>setPrefix(tr("DIFFICULTY"));
    singlePlayerMenu.m_pButtonSelectDifficulty-
>addOption(tr("EASY"), Bot::Easy);
    singlePlayerMenu.m_pButtonSelectDifficulty-
>addOption(tr("MEDIUM"), Bot::Medium);
```

```
    singlePlayerMenu.m_pButtonSelectDifficulty->setPos(width,
height += buttonHeigth);
    singlePlayerMenu.m_pButtonSelectDifficulty-
>setParentItem(singlePlayerMenu.m_pSinglePlayerMenu);
    singlePlayerMenu.m_pButtonClearMap = new MenuButton(tr("CLEAR
MAP"));
    singlePlayerMenu.m_pButtonClearMap->setSize(buttonWidth,
buttonHeigth);
    singlePlayerMenu.m_pButtonClearMap->setPos(width,  height  +=
buttonHeigth);
    singlePlayerMenu.m_pButtonClearMap-
>setParentItem(singlePlayerMenu.m_pSinglePlayerMenu);
    singlePlayerMenu.m_pButtonGenerateNewMap        =        new
MenuButton(tr("GENERATE NEW MAP"));
    singlePlayerMenu.m_pButtonGenerateNewMap->setSize(buttonWidth,
buttonHeigth);
    singlePlayerMenu.m_pButtonGenerateNewMap->setPos(width,  height
+= buttonHeigth);
    singlePlayerMenu.m_pButtonGenerateNewMap-
>setParentItem(singlePlayerMenu.m_pSinglePlayerMenu);
    singlePlayerMenu.m_pButtonSinglePlayerBack       =       new
MenuButton(tr("BACK"));
    singlePlayerMenu.m_pButtonSinglePlayerBack-
>setSize(buttonWidth, buttonHeigth);
    singlePlayerMenu.m_pButtonSinglePlayerBack->setPos(width,
height += buttonHeigth);
    singlePlayerMenu.m_pButtonSinglePlayerBack-
>setParentItem(singlePlayerMenu.m_pSinglePlayerMenu);
    singlePlayerMenu.m_pGameMapEditor    =    m_pBattleShipCore-
>gameMapEditor();
    singlePlayerMenu.m_pGameMapEditor-
>setLabelText(tr("ABCDEFGHIJ"));
    singlePlayerMenu.m_pGameMapEditor->setPos(m_pScene->width() / 2
- 20, m_pScene->height() / 2 - 165);
    singlePlayerMenu.m_pGameMapEditor-
>setParentItem(singlePlayerMenu.m_pSinglePlayerMenu);
    connect(singlePlayerMenu.m_pButtonStartGame,
&MenuButton::clicked,
            this, &BattleShipView::drawGame);
    connect(singlePlayerMenu.m_pButtonSelectDifficulty,
&MenuSelectedButton::changeValue,
            m_pBattleShipCore,
&BattleShipCore::changeBotDifficulty);
    connect(singlePlayerMenu.m_pButtonClearMap,
&MenuButton::clicked,
            this, &BattleShipView::clickedSetClearMode);
    connect(singlePlayerMenu.m_pButtonGenerateNewMap,
&MenuButton::clicked,
            this, &BattleShipView::clickedSetRandomMode);
    connect(singlePlayerMenu.m_pButtonSinglePlayerBack,
&MenuButton::clicked,
            this, &BattleShipView::drawMainMenu);
    return singlePlayerMenu.m_pSinglePlayerMenu; }
QGraphicsRectItem *BattleShipView::initGameMenu() {
    auto width = m_pScene->width() / 2;
    auto height = m_pScene->height() / 2;
    gameMenu.m_pGameMenu = new QGraphicsRectItem();
    gameMenu.m_pGameMenu->setVisible(false);
    gameMenu.m_pHumanMap = m_pBattleShipCore->playerHumanMap();
```

```
    gameMenu.m_pHumanMap->setLabelText(tr("ABCDEFGHIJ"));
    gameMenu.m_pHumanMap->setPos(m_pScene->width()  /  2  -  380,
m_pScene->height() / 2 - 180);
    gameMenu.m_pHumanMap->setParentItem(gameMenu.m_pGameMenu);
    gameMenu.m_pBotMap = m_pBattleShipCore->playerBotMap();
    gameMenu.m_pBotMap->setLabelText(tr("ABCDEFGHIJ"));
    gameMenu.m_pBotMap->setPos(m_pScene->width() / 2 + 50, m_pScene-
>height() / 2 - 180);
    gameMenu.m_pBotMap->setParentItem(gameMenu.m_pGameMenu);
    auto indicator = m_pBattleShipCore->turnIndicator();
    indicator->setPos(width - 30, height - 30);
    indicator->setParentItem(gameMenu.m_pGameMenu);
    gameMenu.m_pButtonGameMenuBack = new MenuButton(tr("BACK"));
    gameMenu.m_pButtonGameMenuBack->setSize(100, 60);
    gameMenu.m_pButtonGameMenuBack->setPos(m_pScene->width()  /  2,
m_pScene->height() - 60);
    gameMenu.m_pButtonGameMenuBack-
>setParentItem(gameMenu.m_pGameMenu);
    connect(gameMenu.m_pButtonGameMenuBack,   &MenuButton::clicked,
this, &BattleShipView::exitFromGameToMenu);
    return gameMenu.m_pGameMenu; }
```

Файл battleShipView.h:

```
#ifndef BATTLESHIP_H
#define BATTLESHIP_H
#include <QtWidgets>
#include "utilities.h"
#include "settings.h"
#include "battleShipCore.h"
#include "turnIndicator.h"
#include "menuButton.h"
#include "menuSelectedButton.h"
#include "menuDisableButton.h"
#include "gameMap.h"
#include "gameMapEditor.h"
class BattleShipView : public QGraphicsView {
    Q_OBJECT
public:
    BattleShipView(QWidget *parent = nullptr);
    BattleShipCore *core() const;
private slots:
    void drawMainMenu(QGraphicsItem *clickedItem);
    void drawSinglePlayerMenu(QGraphicsItem *clickedItem);
    void drawGame(QGraphicsItem *clickedItem);
    void clickedSetClearMode();
    void clickedSetRandomMode();
    void exitFromGameToMenu(QGraphicsItem *clickedItem);
    void playerWins(QString winnerName);
protected:
    void changeEvent(QEvent *event) override;
    bool event(QEvent *event) override;
private:
    QGraphicsRectItem *initMainMenu();
    QGraphicsRectItem *initSinglePlayerMenu();
    QGraphicsRectItem *initGameMenu();
private:
    QGraphicsScene *m_pScene;
    struct {
        QGraphicsRectItem *m_pMainMenu;
```

```
        TextLabel *m_pMainMenuTitle;
        MenuButton *m_pButtonSinglePlayer;
        MenuDisableButton *m_pButtonMultiPlayer;
        MenuDisableButton *m_pButtonStatistics;
        MenuButton *m_pButtonExit;
    } mainMenu;
    struct {
        QGraphicsRectItem *m_pSinglePlayerMenu;
        TextLabel *m_pSinglePlayerTitle;
        MenuButton *m_pButtonStartGame;
        MenuSelectedButton *m_pButtonSelectDifficulty;
        MenuButton *m_pButtonClearMap;
        MenuButton *m_pButtonGenerateNewMap;
        MenuButton *m_pButtonSinglePlayerBack;
        QSharedPointer<GameMapEditor> m_pGameMapEditor;
    } singlePlayerMenu;
    struct {
        QGraphicsRectItem *m_pGameMenu;
        MenuButton *m_pButtonGameMenuBack;
        QSharedPointer<GameMap> m_pHumanMap;
        QSharedPointer<GameMap> m_pBotMap;
    } gameMenu;
    BattleShipCore *m_pBattleShipCore;
};
#endif // BATTLESHIP_H
```

Файл battleShipWindow.cpp:

```
#include "battleShipWindow.h"
BattleShipWindow::BattleShipWindow(QWidget *parent)
    : QMainWindow (parent) {
    QApplication::setApplicationDisplayName(tr("SEA BATTLE"));
    // initialization
    m_pView = new BattleShipView(this);
    // setting
    setWindowTitle(QApplication::applicationDisplayName());
    setMinimumSize(840, 630);
    resize(Settings::inst().windowSize());
    m_pMenuGame = new QMenu(tr("&GAME"));
    m_pSettingsMenuGame  =  m_pMenuGame->addAction(tr("SETTINGS"),
this,

&BattleShipWindow::settingWindow);
    m_pAboutMenuGame = m_pMenuGame->addAction(tr("ABOUT"), this,

&BattleShipWindow::aboutBattleShip);
    m_pExitMenuGame = m_pMenuGame->addAction(tr("EXIT"), this,

&BattleShipWindow::close, QKeySequence("CTRL+Q"));
    menuBar()->addMenu(m_pMenuGame);
    setCentralWidget(m_pView); }
void BattleShipWindow::closeEvent(QCloseEvent *event) {
    if (m_pView->core()->isChange()) {
        auto  answer  =  QMessageBox::question(this,  tr("Exit  from
program"),
                                               tr("Are  you  sure  you
want to go out?",
                                                  "From  the  program
during the game"));
        if (answer == QMessageBox::No) {
```

```
            event->ignore();
            return; } }
    Settings::inst().setWindowSize(this->size());
    QWidget::closeEvent(event); }
void BattleShipWindow::changeEvent(QEvent *event) {
    if (event->type() == QEvent::LanguageChange) {
        QApplication::setApplicationDisplayName(tr("BATTLE"));
        setWindowTitle(QApplication::applicationDisplayName());
        m_pMenuGame->setTitle(tr("&GAME"));
        m_pSettingsMenuGame->setText(tr("SETTINGS"));
        m_pExitMenuGame->setText(tr("EXIT"));
    } else {
        QWidget::changeEvent(event); } }
bool BattleShipWindow::event(QEvent *event) {
    if (event->type() == SettingsChangeEvent::typeEvent()) {
        return QApplication::sendEvent(m_pView, event); }
    return QMainWindow::event(event); }
void BattleShipWindow::aboutBattleShip() {
    QMessageBox::information(this, tr("INFO"),
                             tr("CREATED BY:\n"
                                "IVAN MATSUR\n"
                                "BSUIR, 250502\n"
                                "LAST UPD: 29.11.2023") +
                             " NEW UPDATE: SOON"); }
void BattleShipWindow::settingWindow() {
    SettingsWindow window(this);
    window.exec(); }
```

Файл battleShipWindow.h:

```
#ifndef BATTLESHIPWINDOW_H
#define BATTLESHIPWINDOW_H
#include <QtWidgets>
#include "settingsWindow.h"
#include "settings.h"
#include "battleShipView.h"
class BattleShipWindow : public QMainWindow {
    Q_OBJECT
public:
    BattleShipWindow(QWidget *parent = nullptr);
protected:
    void closeEvent(QCloseEvent *event) override;
    void changeEvent(QEvent *event) override;
    bool event(QEvent *event) override;
private slots:
    void aboutBattleShip();
    void settingWindow();
private:
    QMenu *m_pMenuGame;
    QAction *m_pSettingsMenuGame;
    QAction *m_pAboutMenuGame;
    QAction *m_pExitMenuGame;
    BattleShipView *m_pView;
};
#endif // BATTLESHIPWINDOW_H
```

Файл bot.cpp:

```
#include "bot.h"
Bot::Bot(QString name, QObject *parent)
    : Player (name, parent),
```

```
      m_expertMode(false),
      m_changeShotDirection(false),
      m_botX(0), m_botY(0),
      m_primaryX(0), m_primaryY(0),
      m_shotDirection(e_Direction::None),
      m_difficulty(e_Difficulty::Easy) { }
e_Status Bot::turn(Player *otherPlayer) {
    switch (m_difficulty) {
    case e_Difficulty::Easy:
        return easyDifficulty(otherPlayer);
    case e_Difficulty::Medium:
        return mediumDifficulty(otherPlayer); }
    return e_Status::Impossible; }
void Bot::setDifficulty(e_Difficulty difficulty) {
    m_difficulty = difficulty; }
int Bot::botX() const {
    return m_botX; }
int Bot::botY() const {
    return m_botY; }
e_Status Bot::easyDifficulty(Player *otherPlayer) {
    bool selectCoords = false;
    do {
        m_botX = generateRandomNumber(0, g_MAP_SIZE - 1);
        m_botY = generateRandomNumber(0, g_MAP_SIZE - 1);
        if (otherPlayer->gameMap()->isEmptyCell(m_botX, m_botY))
            selectCoords = true;
    } while (!selectCoords);
    return otherPlayer->gameMap()->shot(m_botX, m_botY); }
e_Status Bot::mediumDifficulty(Player *otherPlayer) {
    if (!m_expertMode) {
        bool selectCoords = false;
        do {
            m_botX = generateRandomNumber(0, g_MAP_SIZE - 1);
            m_botY = generateRandomNumber(0, g_MAP_SIZE - 1);
            if         (otherPlayer->gameMap()->isEmptyCell(m_botX,
m_botY))
                selectCoords = true;
        } while (!selectCoords);
    } else {
        do {
            m_changeShotDirection = false;
            switch (m_shotDirection) {
            case e_Direction::Left:
                if (m_botX > 0) {
                    m_botX--;
                    if                    (!otherPlayer->gameMap()-
>isEmptyCell(m_botX, m_botY))
                        m_changeShotDirection = true;
                } else
                    m_changeShotDirection = true;
                break;
            case e_Direction::Right:
                if (m_botX < g_MAP_SIZE - 1) {
                    m_botX++;
                    if                    (!otherPlayer->gameMap()-
>isEmptyCell(m_botX, m_botY))
                        m_changeShotDirection = true;
                } else
                    m_changeShotDirection = true;
```

```
                break;
            case e_Direction::Down:
                if (m_botY > 0) {
                    m_botY--;
                    if                    (!otherPlayer->gameMap()-
>isEmptyCell(m_botX, m_botY))
                        m_changeShotDirection = true;
                } else
                    m_changeShotDirection = true;
                break;
            case e_Direction::Up:
                if (m_botY < g_MAP_SIZE - 1) {
                    m_botY++;
                    if                    (!otherPlayer->gameMap()-
>isEmptyCell(m_botX, m_botY))
                        m_changeShotDirection = true;
                } else
                    m_changeShotDirection = true;
                break;
            default:
                m_expertMode = false;
                break; }
            // change the direction of fire
            if (m_changeShotDirection) {
                m_shotDirection                                    =
static_cast<e_Direction>((static_cast<int>(m_shotDirection) + 1) %
4);
                m_botX = m_primaryX;
                m_botY = m_primaryY;
                qDebug()   <<   "Bot:   change   direction   "   <<
static_cast<int>(m_shotDirection) << ": " << m_botX << ", " <<
m_botY; }
        } while (m_changeShotDirection); }
    auto   resultShooting   =   otherPlayer->gameMap()->shot(m_botX,
m_botY);
    switch (resultShooting) {
    case e_Status::Hit:
        if (!m_expertMode) {
            m_primaryX = m_botX;
            m_primaryY = m_botY;
            m_expertMode = true;
            m_shotDirection = static_cast<e_Direction>(rand() % 4);
}
        break;
    case e_Status::Miss:
        if (m_expertMode) {
            m_botX = m_primaryX;
            m_botY = m_primaryY;
            switch (m_shotDirection) {
            case e_Direction::Left:
                m_shotDirection = e_Direction::Right;
                break;
            case e_Direction::Right:
                m_shotDirection = e_Direction::Left;
                break;
            case e_Direction::Down:
                m_shotDirection = e_Direction::Up;
                break;
            case e_Direction::Up:
```

```
                m_shotDirection = e_Direction::Down;
                break;
            default:
                break; } }
        break;
    case e_Status::Destroyed:
        m_expertMode = false;
        break;
    default:
        break; }
    return resultShooting; }
```

Файл bot.h:

```
#ifndef BOT_H
#define BOT_H
#include <QtWidgets>
#include "utilities.h"
#include "player.h"
class Bot : public Player {
public:
    enum e_Difficulty {
        Easy,
        Medium
    };
    Bot(QString name, QObject *parent = nullptr);
    e_Status turn(Player *otherPlayer);
    void setDifficulty(e_Difficulty difficulty);
    int botX() const;
    int botY() const;
private:
    e_Status easyDifficulty(Player *otherPlayer);
    e_Status mediumDifficulty(Player *otherPlayer);
private:
    bool m_expertMode;
    bool m_changeShotDirection;
    int m_botX;
    int m_botY;
    int m_primaryX;
    int m_primaryY;
    e_Direction m_shotDirection;
    e_Difficulty m_difficulty;
};
#endif // BOT_H
```

Файл gameMap.cpp:

```
#include "gameMap.h"
GameMap::GameMap(int width, int height, bool disable, const QString
&textLayout)
    : QGraphicsObject (), m_width(width), m_height(height) {
    for (int i = 0; i < g_MAP_SIZE; ++i)

m_mesh.push_back(QVector<QSharedPointer<Cell>>(g_MAP_SIZE));
    m_textLayout.resize(g_MAP_SIZE);
    const int dx = m_width / (g_MAP_SIZE + 1);
    const int dy = m_height / (g_MAP_SIZE + 1);
    for (int i = 0; i < g_MAP_SIZE + 1; ++i) {
        for (int j = 0; j < g_MAP_SIZE + 1; ++j) {
            if (i == 0 && j == 0) {
                continue;
```

```
            } else if (i == 0) {
                auto temp = new TextLabel(dx * j, dy * i, dx, dy,
textLayout.at(j - 1));
                temp->setParentItem(this);
                m_textLayout[j - 1].reset(temp);
            } else if (j == 0) {
                TextLabel *temp = new TextLabel(dx * j, dy * i, dx,
dy, QString::number(i));
                temp->setParentItem(this);
            } else {
                auto temp = new Cell(dx, dy, i - 1, j - 1);
                temp->setPos(dx * j, dy * i);
                temp->setParentItem(this);
                if (!disable) {
                    temp->setAcceptHoverEvents(true);
                    temp->setShowShip(false); }
                m_mesh[i - 1][j - 1].reset(temp); } } } }
e_Status GameMap::shot(int x, int y) {
    switch (m_mesh[x][y]->status()) {
    case e_Status::Empty:
    case e_Status::NearbyShip:
        this->setCellStatus(x, y, e_Status::Miss);
        return e_Status::Miss;
    case e_Status::Life:
        switch(m_mesh[x][y]->ship()->shot(x, y)) {
        case e_Status::Hit:
            this->setCellStatus(x, y, e_Status::Hit);
            return  e_Status::Hit;
        case e_Status::Destroyed:
            this->setCellStatus(x, y, e_Status::Destroyed);
            return  e_Status::Destroyed;
        default:
            return e_Status::Impossible; }
    default:
        return e_Status::Impossible; } }
void GameMap::setCellShip(int x, int y, QSharedPointer<Ship> &ship)
{
    m_mesh[x][y]->setShip(ship); }
void GameMap::setCellStatus(int x, int y, e_Status st) {
    m_mesh[x][y]->setStatus(st);
    m_mesh[x][y]->update(); }
e_Status GameMap::cellStatus(int x, int y) const {
    return m_mesh[x][y]->status(); }
bool GameMap::isEmptyCell(int x, int y) const {
    switch (m_mesh[x][y]->status()) {
    case e_Status::Miss:
    case e_Status::Hit:
    case e_Status::Destroyed:
        return false;
    default:
        return true; } }
void GameMap::resetStatusMesh() {
    for (int i = 0; i < g_MAP_SIZE; ++i)
        for (int j = 0; j < g_MAP_SIZE; ++j)
            m_mesh[i][j]->reset(); }
void GameMap::setDestroyedArea(int x, int y) {
    e_Direction dir = e_Direction::None;
    int tempX = x, tempY = y;
    bool find = false;
```

```
    while (true) {
        setDestroyedAreaImpl(x, y, dir);
        find = false;
        while (!find) {
            x = tempX;
            y = tempY;
            dir = static_cast<e_Direction>((static_cast<int>(dir) +
1) % 5);
            switch (dir) {
            case e_Direction::Right:
                x++;
                break;
            case e_Direction::Left:
                x--;
                break;
            case e_Direction::Up:
                y--;
                break;
            case e_Direction::Down:
                y++;
                break;
            default:
                return; }
            if (x >= g_MAP_SIZE || y >= g_MAP_SIZE || x < 0 || y <
0)
                continue;
            if (cellStatus(x, y) == e_Status::Destroyed)
                find = true; } } }
void GameMap::setLabelText(const QString &textLayout) {
    for (int i = 0; i < m_textLayout.size(); ++i) {
        m_textLayout[i]->setText(textLayout.at(i)); } }
QRectF GameMap::boundingRect() const {
    return QRectF(0, 0, m_width, m_height); }
void        GameMap::paint(QPainter        *painter,        const
QStyleOptionGraphicsItem */*option*/, QWidget */*widget*/) {
    painter->setPen(Qt::NoPen);
    painter->drawRect(this->boundingRect()); }
void GameMap::setDestroyedAreaImpl(int x, int y, e_Direction dir) {
    bool find;
    do {
        find = false;
        for (int i = -1; i < 2; i++) {
            for (int j = -1; j < 2; j++) {
                if ((i + x) < g_MAP_SIZE && (i + x) >= 0 &&
                    (j + y) < g_MAP_SIZE && (j + y) >= 0) {
                    if   (cellStatus(x   +   i,   y   +   j)   ==
e_Status::NearbyShip)
                        setCellStatus(x + i, y + j, e_Status::Miss);
                    if (cellStatus(x + i, y + j) == e_Status::Hit)
                        setCellStatus(x    +    i,    y    +    j,
e_Status::Destroyed); } } }
        switch (dir) {
        case e_Direction::Right:
            x++;
            break;
        case e_Direction::Left:
            x--;
            break;
        case e_Direction::Up:
```

```
            y--;
            break;
        case e_Direction::Down:
            y++;
            break;
        default:
            return; }
        if (x >= g_MAP_SIZE || y >= g_MAP_SIZE || x < 0 || y < 0)
            continue;
        if (cellStatus(x, y) == e_Status::Destroyed)
            find = true;
    } while (find); }
GameMap::Cell::Cell(int width, int heigth, int idX, int idY)
    : QGraphicsObject (),
      m_status(e_Status::Empty),
      m_width(width),
      m_height(heigth),
      m_idX(idX),
      m_idY(idY),
      m_hover(false),
      m_showShip(true) { }
void GameMap::Cell::setShowShip(bool flag) {
    m_showShip = flag; }
void GameMap::Cell::reset() {
    m_status = e_Status::Empty;
    m_pShip.reset(); }
QSharedPointer<Ship> &GameMap::Cell::ship() {
    return m_pShip; }
void GameMap::Cell::setShip(const QSharedPointer<Ship> &ship) {
    m_pShip = ship; }
e_Status GameMap::Cell::status() const {
    return m_status; }
void GameMap::Cell::setStatus(const e_Status &status) {
    m_status = status; }
QRectF GameMap::Cell::boundingRect() const {
    return  QRectF(0, 0, m_width, m_height); }
void      GameMap::Cell::paint(QPainter      *painter,      const
QStyleOptionGraphicsItem */*option*/, QWidget */*widget*/) {
    switch (m_status) {
    case e_Status::Miss:
        painter->setBrush(Qt::cyan);
        painter->drawRect(0, 0, m_width, m_height);
        painter->setPen(QPen(Qt::red, 2));
        painter->drawLine(0, 0, m_width, m_height);
        painter->drawLine(0, m_height, m_width, 0);
        break;
    case e_Status::Destroyed:
        painter->setBrush(Qt::red);
        painter->drawRect(0, 0, m_width, m_height);
        break;
    case e_Status::Hit:
        painter->setBrush(Qt::yellow);
        painter->drawRect(0, 0, m_width, m_height);
        break;
    case e_Status::Life:
    case e_Status::NearbyShip:
    case e_Status::Empty:
        if (m_showShip && m_status == e_Status::Life) {
            painter->setBrush(Qt::blue);
```

```
            painter->drawRect(0, 0, m_width, m_height);
            return; }
        painter->setBrush(Qt::cyan);
        painter->drawRect(0, 0, m_width, m_height);
        if (m_hover) {
            painter->drawLine(0, 0, m_width, m_height);
            painter->drawLine(0, m_height, m_width, 0); }
        break;
    default:
        break; } }
void GameMap::Cell::hoverMoveEvent(QGraphicsSceneHoverEvent *event)
{
    m_hover = true;
    update();
    QGraphicsObject::hoverMoveEvent(event); }
void        GameMap::Cell::hoverLeaveEvent(QGraphicsSceneHoverEvent
*event) {
    m_hover = false;
    update();
    QGraphicsObject::hoverLeaveEvent(event); }
void GameMap::Cell::mousePressEvent(QGraphicsSceneMouseEvent *) { }
void      GameMap::Cell::mouseReleaseEvent(QGraphicsSceneMouseEvent
*event) {
    if (!m_showShip)
        emit     qobject_cast<GameMap     *>(this->parentObject())-
>clicked(m_idX, m_idY);
    QGraphicsObject::mouseReleaseEvent(event); }
```

Файл gameMap.h:

```
#ifndef GAMEMAP_H
#define GAMEMAP_H
#include <QtWidgets>
#include "textLabel.h"
#include "ship.h"
class GameMap : public QGraphicsObject {
    Q_OBJECT
public:
    GameMap(int  width,  int  height,  bool  disable  =  false,  const
QString &textLayout = "ABCDEFGHIJ");
    e_Status shot(int x, int y);
    void setCellShip(int x, int y, QSharedPointer<Ship> &ship);
    void setCellStatus(int x, int y, e_Status st);
    e_Status cellStatus(int x, int y) const;
    bool isEmptyCell(int x, int y) const;
    void resetStatusMesh();
    void setDestroyedArea(int x, int y);
    void setLabelText(const QString &textLayout);
    QRectF boundingRect() const override;
    void  paint(QPainter  *painter,  const  QStyleOptionGraphicsItem
*option, QWidget *widget) override;
signals:
    void clicked(int x, int y);
private:
    void setDestroyedAreaImpl(int x, int y, e_Direction dir);
private:
    class Cell : public QGraphicsObject {
    public:
        Cell(int width, int heigth, int idX, int idY);
        void setShowShip(bool flag);
```

```
        void reset();
        QRectF boundingRect() const override;
        void         paint(QPainter         *painter,         const
QStyleOptionGraphicsItem *option, QWidget *widget) override;
        e_Status status() const;
        void setStatus(const e_Status &status);
        QSharedPointer<Ship> &ship();
        void setShip(const QSharedPointer<Ship> &ship);
    private:
        void    hoverMoveEvent(QGraphicsSceneHoverEvent    *event)
override;
        void    hoverLeaveEvent(QGraphicsSceneHoverEvent    *event)
override;
        void    mousePressEvent(QGraphicsSceneMouseEvent    *event)
override;
        void   mouseReleaseEvent(QGraphicsSceneMouseEvent   *event)
override;
    private:
        e_Status m_status;
        QSharedPointer<Ship> m_pShip;
        int m_width;
        int m_height;
        int m_idX;
        int m_idY;
        bool m_hover;
        bool m_showShip;
    };
    int m_width;
    int m_height;
    QVector<QSharedPointer<TextLabel>> m_textLayout;
    QVector<QVector<QSharedPointer<Cell>>> m_mesh;
};
#endif // GAMEMAP_H
```

Файл gameMapEditor.cpp:

```
#include "gameMapEditor.h"
GameMapEditor::GameMapEditor(int width, int height, const QString
&textLayout)
    : QGraphicsObject (),
      m_RIGHT_BORDER(g_MAP_SIZE + 5),
      m_width(width),
      m_height(height),
      m_isDropAllowed(false) {
    setAcceptDrops(true);
    for (int i = 0; i < g_MAP_SIZE; ++i)

m_mesh.append(QVector<QSharedPointer<CellDragAndDrop>>(m_RIGHT_BOR
DER));
    m_textLayout.resize(g_MAP_SIZE);
    const int dx = m_width / (m_RIGHT_BORDER + 2);
    const int dy = m_height / (g_MAP_SIZE + 1);
    for (int i = 0; i < g_MAP_SIZE + 1; ++i) {
        for (int j = 0; j < g_MAP_SIZE + 1; ++j) {
            if (i == 0 && j == 0) {
                continue;
            } else if (i == 0) {
                auto temp = new TextLabel(dx * j, dy * i, dx, dy,
textLayout.at(j - 1));
                temp->setParentItem(this);
```

```
                m_textLayout[j - 1].reset(temp);
            } else if (j == 0) {
                TextLabel *temp = new TextLabel(dx * j, dy * i, dx,
dy, QString::number(i));
                temp->setParentItem(this);
            } else {
                auto temp = new CellDragAndDrop(dx, dy, i - 1, j -
1);
                temp->setPos(dx * j, dy * i);
                temp->setParent(this);
                temp->setParentItem(this);
                m_mesh[i - 1][j - 1].reset(temp); } } }
    m_pClearPlaceMode = new QGraphicsRectItem(this);
    m_pClearPlaceMode->setVisible(false);
    for (int i = 1; i < g_MAP_SIZE + 1; ++i) {
        for (int j = g_MAP_SIZE + 2; j < m_RIGHT_BORDER + 2; ++j) {
            auto temp = new CellDragAndDrop(dx, dy, i - 1, j - 2);
            temp->setPos(dx * j, dy * i);
            temp->setParent(this);
            temp->setParentItem(m_pClearPlaceMode);
            m_mesh[i - 1][j - 2].reset(temp); } } }
void GameMapEditor::setLabelText(const QString &textLayout) {
    for (int i = 0; i < m_textLayout.size(); ++i) {
        m_textLayout[i]->setText(textLayout.at(i)); } }
void      GameMapEditor::moveShipsOnGameMap(QSharedPointer<GameMap>
&map, QVector<QSharedPointer<Ship>> &ships) {
    for (int i = 0; i < g_MAP_SIZE; ++i) {
        for (int j = 0; j < g_MAP_SIZE; ++j) {
            map->setCellShip(i, j, m_mesh[i][j]->ship());
            map->setCellStatus(i, j, m_mesh[i][j]->status()); } }
    ships.swap(m_ships);
    reset(); }
bool GameMapEditor::isReady() {
    if (!m_pClearPlaceMode->isVisible())
        return true;
    for (int i = 0; i < g_MAP_SIZE; ++i) {
        for (int j = g_MAP_SIZE; j < m_RIGHT_BORDER; ++j) {
            if (m_mesh[i][j]->status() == e_Status::Life)
                return false; } }
    return true; }
void GameMapEditor::setClearMapMode() {
    reset();
    m_pClearPlaceMode->setVisible(true);
    setShipsOnClearMap(); }
void GameMapEditor::setGenerateRandomMode() {
    reset();
    m_pClearPlaceMode->setVisible(false);
    setRandomShips(); }
void GameMapEditor::reset() {
    for (int i = 0; i < g_MAP_SIZE; ++i)
        for (int j = 0; j < m_RIGHT_BORDER; ++j)
            m_mesh[i][j]->reset();
    for (auto &ship : m_ships)
        ship->reset(); }
QVector<QSharedPointer<Ship>> GameMapEditor::ships() const {
    return m_ships; }
void  GameMapEditor::setShips(const  QVector<QSharedPointer<Ship>>
&ships) {
    m_ships = ships; }
```

```
QRectF GameMapEditor::boundingRect() const {
    return QRectF(0, 0, m_width, m_height); }
void     GameMapEditor::paint(QPainter     */*painter*/,     const
QStyleOptionGraphicsItem */*option*/, QWidget */*widget*/) { }
void      GameMapEditor::dragEnterEvent(QGraphicsSceneDragDropEvent
*event) {
    if (event->mimeData()->hasFormat(MimeDataOfShip::mimeType())) {
        event->acceptProposedAction(); } }
void       GameMapEditor::dragMoveEvent(QGraphicsSceneDragDropEvent
*event) {
    int x = static_cast<int>(event->pos().y() * (g_MAP_SIZE + 1) /
m_height) - 1;
    int  y  =  static_cast<int>(event->pos().x()  *  (m_RIGHT_BORDER  +
2) / m_width) - 1;
    clearDropEffect(x, y);
    m_isDropAllowed = false;
    if (x >= 0 && y >= 0) {
        if (y == g_MAP_SIZE)
            return;
        if (y >= g_MAP_SIZE) {
            if (!m_pClearPlaceMode->isVisible()) {
                return; }
            --y; }
        auto  data  =  dynamic_cast<const  MimeDataOfShip  *>(event-
>mimeData());
        int index = -1;
        if  ((index  =  data->ship->numberCell(data->dragX,  data-
>dragY)) == -1) {
            return; }
        int dx = x - data->ship->body().at(index).x;
        int dy = y - data->ship->body().at(index).y;
        bool isLeftMap = false, isRightMap = false;
        for (const auto &el : data->ship->body()) {
            if (el.y + dy < g_MAP_SIZE)
                isLeftMap = true;
            if (el.y + dy >= g_MAP_SIZE)
                isRightMap = true;
            if (isLeftMap && isRightMap) {
                return; } }
        for (const auto &el : data->ship->body()) {
            if (!isCorrectPlaceForShip(el.x + dx, el.y + dy, data-
>ship, isLeftMap)) {
                return; } }
        for (const auto &el : data->ship->body()) {
            m_mesh[el.x + dx][el.y + dy]->setDropEffect(true);
            m_mesh[el.x + dx][el.y + dy]->update(); }
        m_isDropAllowed = true; } }
void GameMapEditor::dropEvent(QGraphicsSceneDragDropEvent *event) {
    if (m_isDropAllowed) {
        int  x  =  static_cast<int>(event->pos().y()  *  (g_MAP_SIZE  +
1) / m_height) - 1;
        int y = static_cast<int>(event->pos().x() * (m_RIGHT_BORDER
+ 2) / m_width) - 1;
        if (y >= g_MAP_SIZE)
            --y;
        auto  data  =  dynamic_cast<const  MimeDataOfShip  *>(event-
>mimeData());
        clearDropEffect(x, y);
```

```
        int index = data->ship->numberCell(data->dragX, data->dragY);
        int dx = x - data->ship->body().at(index).x;
        int dy = y - data->ship->body().at(index).y;
        for (auto &el : data->ship->body()) {
            removeNearbyShipStatus(el.x, el.y, data->ship);
            m_mesh[el.x][el.y]->reset(); }
        index = 0;
        for (auto &el : data->ship->body()) {
            m_mesh[el.x + dx][el.y + dy]->setStatus(e_Status::Life);
            m_mesh[el.x + dx][el.y + dy]->setShip(data->ship);
            setNearbyShipStatus(el.x + dx, el.y + dy);
            data->ship->setCellCoord(index, el.x + dx, el.y + dy);
            ++index; }
        update(); } }
void GameMapEditor::setRandomShips() {
    int x = 0, y = 0, dir = 0;
    for (int i = 0; i < 10; i++) {
        do {
            x = generateRandomNumber(0, g_MAP_SIZE - 1);
            y = generateRandomNumber(0, g_MAP_SIZE - 1);
            dir = generateRandomNumber(0, 1);
        } while (!setShip(m_ships[i], x, y, dir)); } }
bool GameMapEditor::setShip(QSharedPointer<Ship> &ship, int x, int y, bool isHor) {
    bool correct = true;
    if (isHor ? x - ship->length() < 0 : y + ship->length() >= g_MAP_SIZE) {
        return false;
    } else {
        for (int i = 0, j = 0; i < ship->length() && j < ship->length(); isHor ? i++ : j++) {
            if (m_mesh[x - i][y + j]->status() != e_Status::Empty) {
                correct = false;
                break; } } }
    if (correct) {
        ship->setOrientation(!isHor ? Ship::Horizontal : Ship::Vertical);
        for (int a = 0, b = 0; a < ship->length() && b < ship->length(); isHor ? a++ : b++) {
            for (int i = -1; i < 2; i++) {
                for (int j = -1; j < 2; j++) {
                    if (i == 0 && j == 0) {
                        m_mesh[x - a][y + b]->setStatus(e_Status::Life);
                        m_mesh[x - a][y + b]->setShip(ship);
                        ship->setCellCoord(a + b, x - a, y + b);
                    } else if (x + i - a < g_MAP_SIZE && x + i - a >= 0 &&
                               y + j + b < g_MAP_SIZE && y + j + b >= 0 &&
                               m_mesh[x + i - a][y + j + b]->status() != e_Status::Life) {
                        m_mesh[x + i - a][y + j + b]->setStatus(e_Status::NearbyShip); } } } }
        return true; }
    return false; }
```

```
void GameMapEditor::setShipsOnClearMap() {
    int x = -1;
    int y = g_MAP_SIZE;
    for (int i = 0; i < m_ships.size(); ++i) {
        if (x + m_ships[i]->length() < g_MAP_SIZE) {
            x += 1;
        } else {
            x = 0;
            y += 2; }
        m_ships[i]->setOrientation(Ship::Vertical);
        for (int j = 0; j < m_ships[i]->length(); ++j) {
            m_ships[i]->setCellCoord(m_ships[i]->length() - j - 1,
x, y);
            m_mesh[x][y]->setStatus(e_Status::Life);
            m_mesh[x][y]->setShip(m_ships[i]);
            setNearbyShipStatus(x, y);
            ++x; } } }
void GameMapEditor::setNearbyShipStatus(int x, int y) {
    for (int i = -1; i < 2; ++i)
        for (int j = -1; j < 2; ++j)
            if (i != 0 || j != 0)
                if (isCorrectCoords(x + i, y + j, isLeftMap(y)))
                    if   (m_mesh[x   +   i][y   +   j]->status()   ==
e_Status::Empty)
                        m_mesh[x      +      i][y      +      j]-
>setStatus(e_Status::NearbyShip); }
void  GameMapEditor::removeNearbyShipStatus(int  x,  int  y,  const
QSharedPointer<Ship> &ship) {
    for (int i = -1; i < 2; ++i) {
        for (int j = -1; j < 2; ++j) {
            if (i != 0 || j != 0) {
                if (isCorrectCoords(x + i, y + j, isLeftMap(y))) {
                    bool correct = true;
                    for (int q = -1; q < 2; ++q) {
                        for (int p = -1; p < 2; ++p)  {
                            if (isCorrectCoords(x + i + q, y + j +
p, isLeftMap(y))) {
                                if (m_mesh[x + i + q][y + j + p]-
>status() == e_Status::Life &&
                                        ship->numberCell(x + i + q,
y + j + p) == -1) {
                                    correct = false;
                                    break; } } }
                        if (!correct)
                            break; }
                    if (correct)
                        m_mesh[x      +      i][y      +      j]-
>setStatus(e_Status::Empty); } } } } }
bool  GameMapEditor::isCorrectPlaceForShip(int  x,  int  y,  const
QSharedPointer<Ship> &ship, bool isLeftPartMap) const {
    if (isCorrectCoords(x, y, isLeftPartMap)) {
        if  (m_mesh[x][y]->status()  ==  e_Status::Life   &&  ship-
>numberCell(x, y) == -1) {
            return false; }
        if (m_mesh[x][y]->status() == e_Status::NearbyShip) {
            for (int i = -1; i < 2; ++i) {
                for (int j = -1; j < 2; ++j) {
                    if   (isCorrectCoords(x   +   i,   y   +   j,
isLeftPartMap)) {
```

```
                        if (m_mesh[x + i][y + j]->status() ==
e_Status::Life &&
                                ship->numberCell(x + i, y + j) == -
1) {
                            return false; } } } } }
    } else {
        return false; }
    return true; }
bool GameMapEditor::isCorrectCoords(int x, int y, bool
isLeftPartMap) const {
    if (x < g_MAP_SIZE && x >= 0)
        if (y < (isLeftPartMap ? g_MAP_SIZE : m_RIGHT_BORDER) &&
                y >= (isLeftPartMap ? 0 : g_MAP_SIZE))
            return true;
    return false; }
bool GameMapEditor::isLeftMap(int y) {
    return y < g_MAP_SIZE; }
void GameMapEditor::clearDropEffect(int x, int y) {
    for (int i = x - 5; i < x + 5; ++i) {
        for (int j = y - 5; j < y + 5; ++j) {
            if (i >= 0 && j >= 0 && i < g_MAP_SIZE && j <
m_RIGHT_BORDER) {
                m_mesh[i][j]->setDropEffect(false);
                m_mesh[i][j]->update(); } } } }
void GameMapEditor::rotateShip(int x, int y, QSharedPointer<Ship>
ship) {
    int rotatePos = ship->numberCell(x, y);
    int rotateInvert = ship->orientation() == Ship::Vertical ? -1 :
1;
    bool isLeftMap = true;
    if (y >= g_MAP_SIZE) {
        --y;
        isLeftMap = false; }
    for (auto &el : ship->body()) {
        int rotateValue = ship->numberCell(el.x, el.y) - rotatePos;
        if (!isCorrectPlaceForShip(el.x - rotateValue *
rotateInvert,
                                   el.y - rotateValue *
rotateInvert,
                                   ship, isLeftMap)) {
            return; } }
    for (auto &el : ship->body()) { // remove ship from old coords
        removeNearbyShipStatus(el.x, el.y, ship);
        m_mesh[el.x][el.y]->reset(); }
    for (auto &el : ship->body()) { // set ship to the new coords
        int rotateValue = ship->numberCell(el.x, el.y) - rotatePos;
        el.x -= rotateValue * rotateInvert;
        el.y -= rotateValue * rotateInvert;
        m_mesh[el.x][el.y]->setStatus(e_Status::Life);
        m_mesh[el.x][el.y]->setShip(ship);
        setNearbyShipStatus(el.x, el.y); }
    ship->setOrientation(ship->orientation() == Ship::Horizontal ?
Ship::Vertical : Ship::Horizontal);
    update(); }
GameMapEditor::MimeDataOfShip::MimeDataOfShip()
    : QMimeData () { }
QString GameMapEditor::MimeDataOfShip::mimeType() {
    return "app/mimeDataOfShip"; }
```

```
GameMapEditor::CellDragAndDrop::CellDragAndDrop(int width, int
heigth, int idX, int idY)
    : QGraphicsObject(),
      m_status(e_Status::Empty),
      m_pShip(nullptr),
      m_dropEffect(false),
      m_width(width),
      m_height(heigth),
      m_idX(idX),
      m_idY(idY) { }
void GameMapEditor::CellDragAndDrop::reset() {
    m_status = e_Status::Empty;
    m_pShip.reset(); }
bool GameMapEditor::CellDragAndDrop::dropEffect() const {
    return m_dropEffect; }
void GameMapEditor::CellDragAndDrop::setDropEffect(bool dropEffect)
{
    m_dropEffect = dropEffect; }
QSharedPointer<Ship> &GameMapEditor::CellDragAndDrop::ship() {
    return m_pShip; }
void                  GameMapEditor::CellDragAndDrop::setShip(const
QSharedPointer<Ship> &ship) {
    m_pShip = ship; }
e_Status GameMapEditor::CellDragAndDrop::status() const {
    return m_status; }
void    GameMapEditor::CellDragAndDrop::setStatus(const    e_Status
&status) {
    m_status = status; }
QRectF GameMapEditor::CellDragAndDrop::boundingRect() const {
    return QRectF(0, 0, m_width, m_height); }
void GameMapEditor::CellDragAndDrop::paint(QPainter *painter,
                                           const
QStyleOptionGraphicsItem *, QWidget *) {
    switch (m_status) {
    case e_Status::Life:
        painter->setBrush(Qt::blue);
        painter->drawRect(0, 0, m_width, m_height);
        break;
    default:
        if (m_dropEffect)
            painter->setBrush(Qt::green);
        else
            painter->setBrush(Qt::cyan);
        painter->drawRect(0, 0, m_width, m_height);
        break; } }
void
GameMapEditor::CellDragAndDrop::mousePressEvent(QGraphicsSceneMous
eEvent *event) {
    if   (m_status   ==   e_Status::Life   &&   event->button()   ==
Qt::LeftButton)
        m_dragPos = event->pos(); }
void
GameMapEditor::CellDragAndDrop::mouseMoveEvent(QGraphicsSceneMouse
Event *event) {
    if   (m_status   ==   e_Status::Life   &&   (event->buttons()   &
Qt::LeftButton)) {
        auto        distance        =        (event->pos()        -
m_dragPos).manhattanLength();
        if (distance > QApplication::startDragDistance()) {
```

```
            this->startDrag(); } }
    QGraphicsObject::mouseMoveEvent(event); }
void
GameMapEditor::CellDragAndDrop::mouseDoubleClickEvent(QGraphicsSce
neMouseEvent *event) {
    if (m_pShip->length() > 1 && m_status == e_Status::Life) {
        dynamic_cast<GameMapEditor            *>(this->parent())-
>rotateShip(m_idX, m_idY, m_pShip); }
    QGraphicsObject::mouseDoubleClickEvent(event); }
void GameMapEditor::CellDragAndDrop::startDrag() {
    MimeDataOfShip *mimeData = new MimeDataOfShip;
    mimeData->ship = m_pShip;
    mimeData->dragX = m_idX;
    mimeData->dragY = m_idY;
    QDrag *drag = new QDrag(this);
    drag->setMimeData(mimeData);
    QPixmap  dragIcon(":/image/ship"  +  QString::number(m_pShip-
>length()));
    dragIcon  =  dragIcon.scaled(m_width  *  m_pShip->length(),
m_height);
    QImage                                image(dragIcon.size(),
QImage::Format_ARGB32_Premultiplied);
    image.fill(Qt::transparent);
    QPainter painter(&image);
    painter.setOpacity(0.5);
    painter.drawPixmap(0, 0, dragIcon);
    painter.end();
    dragIcon = QPixmap::fromImage(image);
    drag->setPixmap(dragIcon);
    if (m_pShip->orientation() == Ship::Vertical)
        drag->setHotSpot(QPoint(static_cast<int>(m_width * 0.5),
                                static_cast<int>(m_height        *
(m_pShip->length() - m_pShip->numberCell(m_idX, m_idY) - 0.5))));
    else
        drag->setHotSpot(QPoint(static_cast<int>(m_width         *
(m_pShip->numberCell(m_idX, m_idY) + 0.5)),
                                static_cast<int>(m_height * 0.5)));
    drag->exec(); }
```

Файл gameMapEditor.h:

```
#ifndef GAMEMAPDEDITOR_H
#define GAMEMAPDEDITOR_H
#include <QtWidgets>
#include "gameMap.h"
#include "textLabel.h"
#include "utilities.h"
class GameMapEditor : public QGraphicsObject {
public:
    GameMapEditor(int width, int height, const QString &textLayout
= "ABCDEFGHIJ");
    void setLabelText(const QString &text);
    void  moveShipsOnGameMap(QSharedPointer<GameMap>  &playerMap,
QVector<QSharedPointer<Ship>> &playerShips);
    bool isReady();
    void setClearMapMode();
    void setGenerateRandomMode();
    void reset();
    QVector<QSharedPointer<Ship>> ships() const;
    void setShips(const QVector<QSharedPointer<Ship>> &ships);
```

```
    QRectF boundingRect() const override;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget) override;
protected:
    void dragEnterEvent(QGraphicsSceneDragDropEvent *event)
override;
    void dragMoveEvent(QGraphicsSceneDragDropEvent *event)
override;
    void dropEvent(QGraphicsSceneDragDropEvent *event) override;
private:
    void setShipsOnClearMap();
    void setRandomShips();
    bool setShip(QSharedPointer<Ship> &ship, int x, int y, bool
isHor);
    void setNearbyShipStatus(int x, int y);
    void removeNearbyShipStatus(int x, int y, const
QSharedPointer<Ship> &ship);
    bool isCorrectPlaceForShip(int x, int y, const
QSharedPointer<Ship> &ship, bool isLeftPartMap) const;
    bool isCorrectCoords(int x, int y, bool isLeftPartMap) const;
    bool isLeftMap(int y);
    void clearDropEffect(int x, int y);
    void rotateShip(int x, int y, QSharedPointer<Ship> ship);
private:
    struct MimeDataOfShip : public QMimeData {
        MimeDataOfShip();
        static QString mimeType();
        QSharedPointer<Ship> ship;
        int dragX;
        int dragY;
    };
    class CellDragAndDrop : public QGraphicsObject {
    public:
        CellDragAndDrop(int width, int heigth, int idX, int idY);
        void reset();
        QRectF boundingRect() const override;
        void paint(QPainter *painter, const
QStyleOptionGraphicsItem *option, QWidget *widget) override;
        e_Status status() const;
        void setStatus(const e_Status &status);
        QSharedPointer<Ship> &ship();
        void setShip(const QSharedPointer<Ship> &ship);
        bool dropEffect() const;
        void setDropEffect(bool dropEffect);
    protected:
        void mousePressEvent(QGraphicsSceneMouseEvent *event)
override;
        void mouseMoveEvent(QGraphicsSceneMouseEvent *event)
override;
        void mouseDoubleClickEvent(QGraphicsSceneMouseEvent *event)
override;
    private:
        void startDrag();
    private:
        e_Status m_status;
        QSharedPointer<Ship> m_pShip;
        bool m_dropEffect;
        int m_width;
        int m_height;
```

```
        int m_idX;
        int m_idY;
        QPointF m_dragPos;
    };
    const int m_RIGHT_BORDER;
    int m_width;
    int m_height;
    bool m_isDropAllowed;
    QGraphicsRectItem *m_pClearPlaceMode;
    QVector<QSharedPointer<TextLabel>> m_textLayout;
    QVector<QVector<QSharedPointer<CellDragAndDrop>>> m_mesh;
    QVector<QSharedPointer<Ship>> m_ships;
};
#endif // GAMEMAPDEDITOR_H
```

Файл main.cpp:

```
#include <QApplication>
#include "battleShipWindow.h"
#include "settings.h"
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    BattleShipWindow wgt;
    wgt.show();
    return app.exec(); }
```

Файл menuButton.cpp:

```
#include "menuButton.h"
MenuButton::MenuButton(const QString &title)
    : QGraphicsObject (),
      m_title(title),
      m_width(0),
      m_height(0),
      m_hover(false) {
    setAcceptHoverEvents(true);
    m_font = QApplication::font(); }
MenuButton::MenuButton(const QString &title, double width, double
height)
    : QGraphicsObject (),
      m_title(title),
      m_width(width),
      m_height(height),
      m_hover(false) {
    setAcceptHoverEvents(true);
    m_font      =      computeFontForText(boundingRect().toRect(),
Qt::AlignCenter | Qt::TextWordWrap, m_title); }
void MenuButton::setSize(double width, double height) {
    m_width = width;
    m_height = height;
    m_font      =      computeFontForText(boundingRect().toRect(),
Qt::AlignCenter | Qt::TextWordWrap, m_title); }
void MenuButton::setText(const QString &title) {
    m_title = title;
    m_font      =      computeFontForText(boundingRect().toRect(),
Qt::AlignCenter | Qt::TextWordWrap, m_title); }
QRectF MenuButton::boundingRect() const {
    return QRectF(0, 0, m_width, m_height); }
void       MenuButton::paint(QPainter       *painter,       const
QStyleOptionGraphicsItem */*option*/, QWidget */*widget*/) {
    if (m_hover)
```

```
        painter->setBrush(Qt::blue);
    else
        painter->setBrush(Qt::cyan);
    painter->setPen(Qt::black);
    painter->setFont(m_font);
    painter->drawRect(this->boundingRect());
    painter->drawText(this->boundingRect(),    Qt::AlignCenter    |
Qt::TextWordWrap, m_title); }
void          MenuButton::mousePressEvent(QGraphicsSceneMouseEvent
*/*event*/) {
//    QGraphicsObject::mousePressEvent(event); }
void MenuButton::mouseReleaseEvent(QGraphicsSceneMouseEvent *event)
{
    emit clicked(this);
    m_hover = false;
    QGraphicsObject::mouseReleaseEvent(event); }
void MenuButton::hoverEnterEvent(QGraphicsSceneHoverEvent *event) {
    m_hover = true;
    update();
    QGraphicsObject::hoverEnterEvent(event); }
void MenuButton::hoverLeaveEvent(QGraphicsSceneHoverEvent *event) {
    m_hover = false;
    update();
    QGraphicsObject::hoverLeaveEvent(event); }
```

Файл menuButton.h:

```
#ifndef MENUBUTTON_H
#define MENUBUTTON_H
#include <QtWidgets>
#include "utilities.h"
class MenuButton : public QGraphicsObject {
    Q_OBJECT
public:
    MenuButton(const QString &title);
    MenuButton(const QString &title, double width, double height);
    void setSize(double width, double height);
    void setText(const QString &title);
    QRectF boundingRect() const override;
    void  paint(QPainter  *painter,  const  QStyleOptionGraphicsItem
*option, QWidget *widget) override;
signals:
    void clicked(QGraphicsItem *item);
protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *event) override;
    void     mouseReleaseEvent(QGraphicsSceneMouseEvent     *event)
override;
    void hoverEnterEvent(QGraphicsSceneHoverEvent *event) override;
    void hoverLeaveEvent(QGraphicsSceneHoverEvent *event) override;
private:
    QString m_title;
    double m_width;
    double m_height;
    bool m_hover;
    QFont m_font;
};
#endif // MENUBUTTON_H
```

Файл menuDisableButton.cpp:

```
#include "menuDisableButton.h"
```

```
MenuDisableButton::MenuDisableButton(const QString &title)
    : QGraphicsItem (),
      m_title(title),
      m_width(0),
      m_height(0) {
    m_font = QApplication::font(); }
MenuDisableButton::MenuDisableButton(const QString &title, double
width, double heigth)
    : QGraphicsItem (),
      m_title(title),
      m_width(width),
      m_height(heigth) {
    m_font      =      computeFontForText(boundingRect().toRect(),
Qt::AlignCenter | Qt::TextWordWrap, m_title); }
void MenuDisableButton::setSize(double width, double heigth) {
    m_width = width;
    m_height = heigth;
    m_font      =      computeFontForText(boundingRect().toRect(),
Qt::AlignCenter | Qt::TextWordWrap, m_title); }
void MenuDisableButton::setText(const QString &title) {
    m_title = title;
    m_font      =      computeFontForText(boundingRect().toRect(),
Qt::AlignCenter | Qt::TextWordWrap, m_title); }
QRectF MenuDisableButton::boundingRect() const {
    return QRectF(0, 0, m_width, m_height); }
void     MenuDisableButton::paint(QPainter     *painter,     const
QStyleOptionGraphicsItem */*option*/, QWidget */*widget*/) {
    QLinearGradient gradient(0, 0, m_width, m_height);
    for (int i = 0; i < 50; ++i) {
        if (i % 2 == 0)
            gradient.setColorAt(i / 50.0, Qt::gray);
        else
            gradient.setColorAt(i / 50.0, Qt::white); }
    painter->setBrush(gradient);
    painter->setPen(Qt::black);
    painter->setFont(m_font);
    painter->drawRect(this->boundingRect());
    painter->drawText(this->boundingRect(),    Qt::AlignCenter    |
Qt::TextWordWrap, m_title); }
```

Файл menuDisableButton.h:

```
#ifndef MENUDISABLEBUTTON_H
#define MENUDISABLEBUTTON_H
#include <QtWidgets>
#include "utilities.h"
class MenuDisableButton : public QGraphicsItem {
public:
    MenuDisableButton(const QString &title);
    MenuDisableButton(const QString &title, double width, double
heigth);
    void setSize(double width, double heigth);
    void setText(const QString &title);
    QRectF boundingRect() const override;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget) override;
private:
    QString m_title;
    double m_width;
    double m_height;
```

```
    QFont m_font;
};
#endif // MENUDISABLEBUTTON_H
```

Файл menuSelectedButton.cpp:

```
#include "menuSelectedButton.h"
MenuSelectedButton::MenuSelectedButton()
    : QGraphicsObject (),
      m_width(0),
      m_height(0),
      m_hover(false),
      m_correctValue(0),
      m_prefix() {
    setAcceptHoverEvents(true); }
MenuSelectedButton::MenuSelectedButton(double width, double height)
    : QGraphicsObject (),
      m_width(width),
      m_height(height),
      m_hover(false),
      m_correctValue(0) {
    setAcceptHoverEvents(true); }
void MenuSelectedButton::setSize(double width, double height) {
    m_width = width;
    m_height = height; }
void    MenuSelectedButton::addOption(const    QString    &name,
Bot::e_Difficulty value) {
    m_options.append({name, value}); }
void  MenuSelectedButton::updateTranslate(const  QString  &name,
Bot::e_Difficulty value) {
    for (auto &el : m_options) {
        if (el.second == value) {
            el.first = name;
            break; } } }
void MenuSelectedButton::setPrefix(const QString &prefix) {
    m_prefix = prefix; }
QRectF MenuSelectedButton::boundingRect() const {
    return QRectF(0, 0, m_width, m_height); }
void    MenuSelectedButton::paint(QPainter    *painter,    const
QStyleOptionGraphicsItem */*option*/, QWidget */*widget*/) {
    if (m_hover)
        painter->setBrush(Qt::blue);
    else
        painter->setBrush(Qt::cyan);
    painter->setPen(Qt::black);
    QFont correctFont = QApplication::font();
    correctFont.setPixelSize(static_cast<int>(m_height * 0.35));
    painter->setFont(correctFont);
    painter->drawRect(this->boundingRect());
    auto dx = m_width / (m_options.size() + 1);
    for (int i = 0; i < m_options.size(); ++i) {
        painter->setBrush(Qt::darkYellow);
        if (i == m_correctValue) {
            painter->setFont(computeFontForText(QRectF(0,       0,
m_width, m_height * 0.6).toRect(),
                                                Qt::AlignCenter  |
Qt::TextWordWrap,
                                                m_prefix + ": " +
m_options[i].first));
            painter->drawText(QRectF(0, 0, m_width, m_height * 0.6),
```

```
                                Qt::AlignCenter | Qt::TextWordWrap,
                                m_prefix + ": " + m_options[i].first);
            painter->setBrush(Qt::magenta); }
        painter->drawEllipse(QPointF((i + 1) * dx, m_height * 0.8),
m_height * 0.1, m_height * 0.1); } }
void   MenuSelectedButton::hoverEnterEvent(QGraphicsSceneHoverEvent
*event) {
    m_hover = true;
    update();
    QGraphicsObject::hoverEnterEvent(event); }
void   MenuSelectedButton::hoverLeaveEvent(QGraphicsSceneHoverEvent
*event) {
    m_hover = false;
    update();
    QGraphicsObject::hoverLeaveEvent(event); }
void   MenuSelectedButton::mousePressEvent(QGraphicsSceneMouseEvent
*/*event*/) {
//    QGraphicsObject::mousePressEvent(event); }
void MenuSelectedButton::mouseReleaseEvent(QGraphicsSceneMouseEvent
*event) {
    m_correctValue = (m_correctValue + 1) % m_options.size();
    update();
    emit changeValue(m_options[m_correctValue].second);
    QGraphicsObject::mouseReleaseEvent(event); }
```

Файл menuSelectedButton.h:

```
#ifndef MENUSELECTEDBUTTON_H
#define MENUSELECTEDBUTTON_H
#include <QtWidgets>
#include "bot.h"
class MenuSelectedButton : public QGraphicsObject {
    Q_OBJECT
public:
    MenuSelectedButton();
    MenuSelectedButton(double width, double height);
    void setSize(double width, double height);
    void addOption(const QString &name, Bot::e_Difficulty value);
    void  updateTranslate(const  QString  &name,  Bot::e_Difficulty
value);
    void setPrefix(const QString &prefix);
    QRectF boundingRect() const override;
    void  paint(QPainter  *painter,  const  QStyleOptionGraphicsItem
*option, QWidget *widget) override;
signals:
    void changeValue(Bot::e_Difficulty value);
protected:
    void hoverEnterEvent(QGraphicsSceneHoverEvent *event) override;
    void hoverLeaveEvent(QGraphicsSceneHoverEvent *event) override;
    void mousePressEvent(QGraphicsSceneMouseEvent *event) override;
    void     mouseReleaseEvent(QGraphicsSceneMouseEvent     *event)
override;
private:
    double m_width;
    double m_height;
    bool m_hover;
    int m_correctValue;
    QString m_prefix;
    QVector<QPair<QString, Bot::e_Difficulty>> m_options;
};
```

```
#endif // MENUSELECTEDBUTTON_H
```

Файл player.cpp:

```
#include "player.h"
Player::Player(QString name, QObject *parent)
    : QObject(parent),
      m_playerName(name),
      m_pGameMapPlayer(nullptr) { }
void Player::setMap(QSharedPointer<GameMap> &map) {
    m_pGameMapPlayer = map; }
void Player::setShips(QVector<QSharedPointer<Ship>> ships) {
    m_shipsPlayer = ships; }
bool Player::isDead() const {
    for (auto &ship : m_shipsPlayer)
        if (ship->isLife())
            return false;
    return true; }
void Player::reset() {
    m_pGameMapPlayer->resetStatusMesh();
    for (int i = 0; i < g_MAP_SIZE; ++i)
        m_shipsPlayer[i]->reset(); }
QString Player::name() const {
    return m_playerName; }
QSharedPointer<GameMap> &Player::gameMap() {
    return m_pGameMapPlayer; }
QVector<QSharedPointer<Ship> > &Player::ships() {
    return  m_shipsPlayer; }
```

Файл player.h:

```
#ifndef PLAYER_H
#define PLAYER_H
#include <QtWidgets>
#include "gameMap.h"
#include "ship.h"
class Player : public QObject {
    Q_OBJECT
public:
    Player(QString name, QObject *parent = nullptr);
    void setMap(QSharedPointer<GameMap> &map);
    void setShips(QVector<QSharedPointer<Ship>> ships);
    bool isDead() const;
    void reset();
    QString name() const;
    QSharedPointer<GameMap> &gameMap();
    QVector<QSharedPointer<Ship>> &ships();
signals:
    void mapClicked(int x, int y);
protected:
    QString m_playerName;
    QSharedPointer<GameMap> m_pGameMapPlayer;
    QVector<QSharedPointer<Ship>> m_shipsPlayer;
};
#endif // PLAYER_H
```

Файл settings.cpp:

```
#include "settings.h"
Settings &Settings::inst() {
    static Settings single;
    return single; }
```

```
int Settings::animationDelay() {
    return m_animationDelay; }
Settings::Settings() {
    load(); }
Settings::~Settings() {
    save(); }
void Settings::load() {
    QSettings setting("config.ini", QSettings::IniFormat);
    m_windowSize    =    setting.value("Window-Size",    QSize(800,
600)).toSize();
    m_playerName           =           setting.value("Player-Name",
"player").toString();
    m_animationDelay       =       setting.value("Animation-Delay",
500).toInt(); }
void Settings::save() {
    QSettings setting("config.ini", QSettings::IniFormat);
    setting.setValue("Window-Size", m_windowSize);
    setting.setValue("Player-Name", m_playerName);
    setting.setValue("Animation-Delay", m_animationDelay); }
QString Settings::playerName() const {
    return m_playerName; }
void Settings::setPlayerName(const QString &playerName) {
    m_playerName = playerName; }
QSize Settings::windowSize() const {
    return m_windowSize; }
QRect Settings::sceneRect() const {
    return QRect(0, 0, 800, 600); }
void Settings::setWindowSize(const QSize &windowSize) {
    m_windowSize = windowSize; }
void Settings::setAnimationDelay(int animationDelay) {
    m_animationDelay = animationDelay; }
SettingsChangeEvent::SettingsChangeEvent()
    : QEvent(static_cast<QEvent::Type>(QEvent::User + 200)) { }
QEvent::Type SettingsChangeEvent::typeEvent() {
    return static_cast<QEvent::Type>(QEvent::User + 200); }
```

Файл settings.h:

```
#ifndef SETTINGS_H
#define SETTINGS_H
#include <QtWidgets>
#include "ship.h"
class Settings {
public:
    static Settings &inst();
    int animationDelay();
    void setAnimationDelay(int animationDelay);
    QSize windowSize() const;
    QRect sceneRect() const;
    void setWindowSize(const QSize &windowSize);
    QString playerName() const;
    void setPlayerName(const QString &playerName);
private:
    Settings();
    ~Settings();
    Settings(const Settings &other) = delete;
    Settings& operator=(const Settings &other) = delete;
    void load();
    void save();
private:
```

```
    QSize m_windowSize;
    QString m_playerName;
    int m_animationDelay;
};
class SettingsChangeEvent : public QEvent {
public:
    SettingsChangeEvent();
    static QEvent::Type typeEvent();
};
#endif // SETTINGS_H
```

Файл settingsWindow.cpp:

```
#include "settingsWindow.h"
SettingsWindow::SettingsWindow(QWidget *parent)
    : QDialog (parent) {
    // initialization
    m_pPlayerNameLabel = new QLabel(tr("Player name:"));
    m_pPlayerNameLineEdit = new QLineEdit();
    m_pAnimationSpeedLabel = new QLabel(tr("Animation speed:"));
    m_pAnimationSpeedSlider = new QSlider(Qt::Horizontal);
    m_pAnimationSpeedSpinBox = new QSpinBox();
    m_pButtonOk = new QPushButton(tr("OK"));
    m_pButtonApply = new QPushButton(tr("Apply"));
    m_pButtonCancel =  new QPushButton(tr("Cancel"));
    // settings
    setMinimumSize(400, 300);
    setWindowFlags(this->windowFlags()                                    &
~Qt::WindowContextHelpButtonHint);
    setWindowTitle(tr("Settings"));
    m_pPlayerNameLineEdit->setText(Settings::inst().playerName());
    m_pAnimationSpeedSlider->setRange(200, 2000);
    m_pAnimationSpeedSpinBox->setRange(200, 2000);
    m_pAnimationSpeedSlider-
>setValue(Settings::inst().animationDelay());
    m_pAnimationSpeedSpinBox-
>setValue(Settings::inst().animationDelay());
    m_pAnimationSpeedSpinBox->setSuffix(tr("ms"));
    // layout setup
    QHBoxLayout *layoutPlayerName = new QHBoxLayout;
    layoutPlayerName->addWidget(m_pPlayerNameLabel);
    layoutPlayerName->addWidget(m_pPlayerNameLineEdit);
    QVBoxLayout *layoutGeneralSetting = new QVBoxLayout;
    layoutGeneralSetting->addLayout(layoutPlayerName);
    m_pGroupGeneralSetting = new QGroupBox(tr("General settings"));
    m_pGroupGeneralSetting->setLayout(layoutGeneralSetting);
    QHBoxLayout *layoutAnimationSpeed = new QHBoxLayout;
    layoutAnimationSpeed->addWidget(m_pAnimationSpeedLabel);
    layoutAnimationSpeed->addWidget(m_pAnimationSpeedSlider);
    layoutAnimationSpeed->addWidget(m_pAnimationSpeedSpinBox);
    QVBoxLayout *layoutGameSetting = new QVBoxLayout;
    layoutGameSetting->addLayout(layoutAnimationSpeed);
    m_pGroupGameSetting = new QGroupBox(tr("Game settings"));
    m_pGroupGameSetting->setLayout(layoutGameSetting);
    QHBoxLayout *layoutButton = new QHBoxLayout;
    layoutButton->addStretch(1);
    layoutButton->addWidget(m_pButtonOk);
    layoutButton->addWidget(m_pButtonApply);
    layoutButton->addWidget(m_pButtonCancel);
    QVBoxLayout *layout = new QVBoxLayout;
```

```
    layout->addWidget(m_pGroupGeneralSetting);
    layout->addWidget(m_pGroupGameSetting);
    layout->addStretch(1);
    layout->addLayout(layoutButton);
    setLayout(layout);
    // connenctions
    connect(m_pButtonOk,        &QPushButton::clicked,        this,
&SettingsWindow::clickedButtonOk);
    connect(m_pButtonApply,       &QPushButton::clicked,       this,
&SettingsWindow::clickedButtonApply);
    connect(m_pButtonCancel,      &QPushButton::clicked,      this,
&SettingsWindow::clickedButtonCancel);
    connect(m_pAnimationSpeedSlider,        &QSlider::valueChanged,
m_pAnimationSpeedSpinBox, &QSpinBox::setValue);
    connect(m_pAnimationSpeedSpinBox,    SIGNAL(valueChanged(int)),
m_pAnimationSpeedSlider, SLOT(setValue(int))); }
void SettingsWindow::changeEvent(QEvent *event) {
    if (event->type() == QEvent::LanguageChange) {
        setWindowTitle(tr("SETTINGS"));
        m_pGroupGeneralSetting->setTitle(tr("GENERAL SETTINGS"));
        m_pLocaleListLabel->setText(tr("LOCALE:"));
        m_pPlayerNameLabel->setText(tr("PLAYER NAME:"));
        m_pButtonOk->setText(tr("OK"));
        m_pButtonApply->setText(tr("Apply"));
        m_pButtonCancel->setText(tr("Cancel"));
    } else {
        QWidget::changeEvent(event); } }
void SettingsWindow::clickedButtonOk() {
    clickedButtonApply();
    close(); }
void SettingsWindow::clickedButtonApply() {
    bool isChange = false;
    if               (m_pPlayerNameLineEdit->text()               !=
Settings::inst().playerName()) {
        Settings::inst().setPlayerName(m_pPlayerNameLineEdit-
>text());
        isChange = true; }
    if             (m_pAnimationSpeedSlider->value()             !=
Settings::inst().animationDelay()) {

Settings::inst().setAnimationDelay(m_pAnimationSpeedSlider-
>value());
        isChange = true; }
    if (isChange) {
        SettingsChangeEvent *event = new SettingsChangeEvent();
        QApplication::postEvent(this->parent(), event); } }
void SettingsWindow::clickedButtonCancel() {
    if               (m_pPlayerNameLineEdit->text()               !=
Settings::inst().playerName() ||
           m_pAnimationSpeedSlider->value()                       !=
Settings::inst().animationDelay()) {
        auto  answer  =  QMessageBox::question(this,  tr("Exit  from
settings"),
                                               tr("Are  you  sure  you
want to go out? "
                                                  "Any  settings  you
have changed will not be saved",
                                                  "Exit           from
settings"));
```

```
        if (answer == QMessageBox::No)
            return; }
    close(); }
```

Файл settingsWindow.h:

```
#ifndef SETTINGSWINDOW_H
#define SETTINGSWINDOW_H
#include <QtWidgets>
#include "settings.h"
class SettingsWindow : public QDialog {
    Q_OBJECT
public:
    SettingsWindow(QWidget *parent = nullptr);
protected:
    void changeEvent(QEvent *event) override;
private slots:
    void clickedButtonOk();
    void clickedButtonApply();
    void clickedButtonCancel();
private:
    QGroupBox *m_pGroupGeneralSetting;
    QLabel *m_pLocaleListLabel;
    QComboBox *m_pLocaleListComboBox;
    QLabel *m_pPlayerNameLabel;
    QLineEdit *m_pPlayerNameLineEdit;
    QGroupBox *m_pGroupGameSetting;
    QLabel *m_pAnimationSpeedLabel;
    QSlider *m_pAnimationSpeedSlider;
    QSpinBox *m_pAnimationSpeedSpinBox;
    QPushButton *m_pButtonOk;
    QPushButton *m_pButtonApply;
    QPushButton *m_pButtonCancel;
};
#endif // SETTINGSWINDOW_H
```

Файл ship.cpp:

```
#include "ship.h"
Ship::Ship(int length, Orientation orient)
    : m_length(length),
      m_orientation(orient) {
    m_body.resize(length); }
e_Status Ship::shot(int x, int y) {
    for (auto &cell : m_body)
        if (cell.x == x && cell.y == y) {
            cell.status = e_Status::Destroyed;
            if (isLife())
                return e_Status::Hit;
            else
                return e_Status::Destroyed; }
    return e_Status::Life; }
void Ship::setCellCoord(int numberCell, int x, int y) {
    m_body[numberCell] = {x, y, e_Status::Life}; }
int Ship::numberCell(int x, int y) const {
    for (int i = 0; i < m_body.size(); ++i) {
        if (m_body[i].x == x && m_body[i].y == y)
            return i; }
    return -1; }
QVector<Ship::CellShip> &Ship::body() {
    return m_body; }
```

```
void Ship::reset() {
    for (auto &el : m_body)
        el = {0, 0, e_Status::Life}; }
bool Ship::isLife() const {
    for (const auto &cell : m_body)
        if (cell.status == e_Status::Life)
            return true;
    return false; }
int Ship::length() const {
    return m_length; }
Ship::Orientation Ship::orientation() const {
    return m_orientation; }
void Ship::setOrientation(const Orientation &orientation) {
    m_orientation = orientation; }
```

Файл ship.h:

```
#ifndef SHIP_H
#define SHIP_H
#include <QtWidgets>
#include "utilities.h"
class Ship {
public:
    struct CellShip {
        int x;
        int y;
        e_Status status;
    };
    enum Orientation {
        Vertical,
        Horizontal,
        None
    };
    Ship(int length, Orientation orient = Orientation::None);
    e_Status shot(int x, int y);
    Orientation orientation() const;
    void setOrientation(const Orientation &orientation);
    void setCellCoord(int numberCell, int x, int y);
    int numberCell(int x, int y) const;
    QVector<CellShip> &body();
    void reset();
    bool isLife() const;
    int length() const;
private:
    int m_length;
    Orientation m_orientation;
    QVector<CellShip> m_body;
};
#endif // SHIP_H
```

Файл textLabel.cpp:

```
#include "textLabel.h"
TextLabel::TextLabel(const QString &text)
    : m_width(0),
      m_height(0),
      m_text(text) {
    m_font = QApplication::font(); }
TextLabel::TextLabel(double x, double y, double w, double h, const
QString &text)
    : m_width(w),
```

```
      m_height(h),
      m_text(text) {
    setPos(x, y);
    m_font      =      computeFontForText(boundingRect().toRect(),
Qt::AlignCenter, m_text, 0.95); }
void TextLabel::setSize(double width, double height) {
    m_width = width;
    m_height = height;
    m_font      =      computeFontForText(boundingRect().toRect(),
Qt::AlignCenter, m_text, 0.95); }
void TextLabel::setText(const QString &text) {
    m_text = text;
    m_font      =      computeFontForText(boundingRect().toRect(),
Qt::AlignCenter, m_text, 0.95);
    update(); }
QRectF TextLabel::boundingRect() const {
    return QRectF(0, 0, m_width, m_height); }
void        TextLabel::paint(QPainter        *painter,        const
QStyleOptionGraphicsItem */*option*/, QWidget */*widget*/) {
    painter->setFont(m_font);
    painter->drawText(boundingRect(), Qt::AlignCenter, m_text); }
```

Файл textLabel.h:

```
#ifndef TEXTLABEL_H
#define TEXTLABEL_H
#include <QtWidgets>
#include "utilities.h"
class TextLabel : public QGraphicsItem {
public:
    TextLabel(const QString &text);
    TextLabel(double x, double y, double w, double h, const QString
&text);
    void setSize(double width, double height);
    void setText(const QString &text);
    QRectF boundingRect() const override;
    void  paint(QPainter  *painter,  const  QStyleOptionGraphicsItem
*option, QWidget *widget) override;
private:
    double m_width;
    double m_height;
    QString m_text;
    QFont m_font;
};
#endif // TEXTLABEL_H
```

Файл turnIndicator.cpp:

```
#include "turnIndicator.h"
TurnIndicator::TurnIndicator(int width, int height)
    : QGraphicsObject (),
      m_width(width),
      m_height(height),
      m_angle(0),
      m_color(Qt::green) {
    setTransformOriginPoint(m_width / 2, m_height / 2);
    m_colorAnimation.setTargetObject(this);
    m_colorAnimation.setPropertyName("color");
    m_colorAnimation.setDuration(Settings::inst().animationDelay()
/ 2);
```

```
m_directionAnimation.setDuration(Settings::inst().animationDelay()
/ 2);
    m_points = {
        {0, m_height / 3.0},
        {0, m_height * 2 / 3.0},
        {m_width * 2 / 3.0, m_height * 2 / 3.0},
        {m_width * 2 / 3.0, m_height / 1.0},
        {m_width / 1.0, m_height / 2.0},
        {m_width * 2 / 3.0, 0.0},
        {m_width * 2 / 3.0, m_height / 3.0},
    };
    connect(&m_directionAnimation,
&QVariantAnimation::valueChanged, [this](const QVariant &value){
        this->setRotation(value.toReal());
    }); }
void TurnIndicator::setAnimationDelay(int msec) {
    m_colorAnimation.setDuration(msec / 2);
    m_directionAnimation.setDuration(msec / 2); }
void TurnIndicator::reset() {
    m_angle = 0;
    setRotation(m_angle);
    m_color = Qt::green;
    update(); }
QRectF TurnIndicator::boundingRect() const {
    return QRectF(0, 0, m_width, m_height); }
void      TurnIndicator::paint(QPainter      *painter,      const
QStyleOptionGraphicsItem */*option*/, QWidget */*widget*/) {
    painter->setPen(Qt::black);
    painter->setBrush(m_color);
    painter->drawPolygon(m_points.constData(), m_points.size()); }
QColor TurnIndicator::color() const {
    return m_color; }
void TurnIndicator::change(double angle, QColor color) {
    m_colorAnimation.setStartValue(m_color);
    m_directionAnimation.setStartValue(m_angle);
    m_colorAnimation.setEndValue(QColor(color));
    m_directionAnimation.setEndValue(m_angle + angle);
    m_angle = static_cast<int>(m_angle + angle) % 360;
    m_color = color;
    m_colorAnimation.start();
    m_directionAnimation.start(); }
void TurnIndicator::setColor(QColor color) {
    if (m_color == color)
        return;
    m_color = color;
    emit colorChanged(m_color); }
```

Файл turnIndicator.h:

```
#ifndef TURNINDICATOR_H
#define TURNINDICATOR_H
#include <QtWidgets>
#include "settings.h"
class TurnIndicator : public QGraphicsObject {
    Q_OBJECT
    Q_PROPERTY(QColor  color  READ  color  WRITE  setColor  NOTIFY
colorChanged)
public:
    TurnIndicator(int width, int height);
```

```
    void setAnimationDelay(int msec);
    void reset();
    QColor color() const;
    QRectF boundingRect() const override;
    void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget) override;
public slots:
    void change(double angle, QColor color);
    void setColor(QColor color);
signals:
    void colorChanged(QColor color);
private:
    QVector<QPointF> m_points;
    int m_width;
    int m_height;
    qreal m_angle;
    QColor m_color;
    QVariantAnimation m_directionAnimation;
    QPropertyAnimation m_colorAnimation;
};
#endif // TURNINDICATOR_H
```

Файл utilities.cpp:

```
#include "utilities.h"
QFont computeFontForText(QRect rect, int flags, const QString &text,
double scale) {
    QFont correctFont = QApplication::font();
    rect.setWidth(static_cast<int>(rect.width() * scale));
    rect.setHeight(static_cast<int>(rect.height() * scale));
    for (int i = 1; i < 1000; ++i) {
        correctFont.setPixelSize(i);
        auto tempRect =
QFontMetrics(correctFont).boundingRect(rect, flags, text);
        if (tempRect.height() > rect.height() || tempRect.width() >
rect.width()) {
            correctFont.setPixelSize(i - 1);
            return correctFont; } }
    return correctFont; }
int generateRandomNumber(int from, int to) {
    static std::random_device rd;
    static std::mt19937 gen(rd());
    std::uniform_int_distribution<> uid(from, to);
    return uid(gen); }
```

Файл utilities.h:

```
#ifndef APPNAMESPACE_H
#define APPNAMESPACE_H
#include <QtWidgets>
#include <random>
const int g_MAP_SIZE = 10;
enum class e_Status {
    Empty,
    NearbyShip,
    Life,
    Hit,
    Destroyed,
    Miss,
    Impossible
};
```

```
enum class e_Direction {
    Down,
    Up,
    Left,
    Right,
    None
};
QFont computeFontForText(QRect rect, int flags, const QString &text,
double scale = 0.65);
int generateRandomNumber(int from, int to);
#endif // APPNAMESPACE_H
```