МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

ИИТУС
БГТУ им. В.Г. Шухова

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

КУРСОВОЙ ПРОЕКТ

по дисциплине: Объектно-ориентированное программирование
тема: «Программа моделирования игры Морской Бой»

Автор работы ____________ Пахомов Владислав Андреевич ПВ-223
(подпись)

Руководитель проекта ____________ Черников Сергей Викторович
(подпись)

Оценка ____

Белгород 2024 г.

Оглавление

Название, цель, постановка задачи

Название: Программа моделирования игры Морской Бой

Цель: Написать программу для моделирования игры Морской Бой

Постановка задачи: Программа для игры в морской бой. Программа должна обеспечивать возможность игры человека с компьютером. На экране отображаются два игровых поля: поле для расстановки кораблей человеком и поле для отметки наносимых ударов по кораблям противника.

Должна быть предусмотрена возможность использования кораблей различного типа: одно-, двух-, трех- и четырехпалубных. Пользователь в режиме расстановки кораблей должен иметь возможность размещения кораблей на игровом поле, перемещения, поворота кораблей.

В режиме игры пользователь в наглядном виде должен информироваться о том, достиг ли цели его выстрел и выстрел противника.

1 Описание функциональных требований к разрабатываемой системе

Функциональные требования:

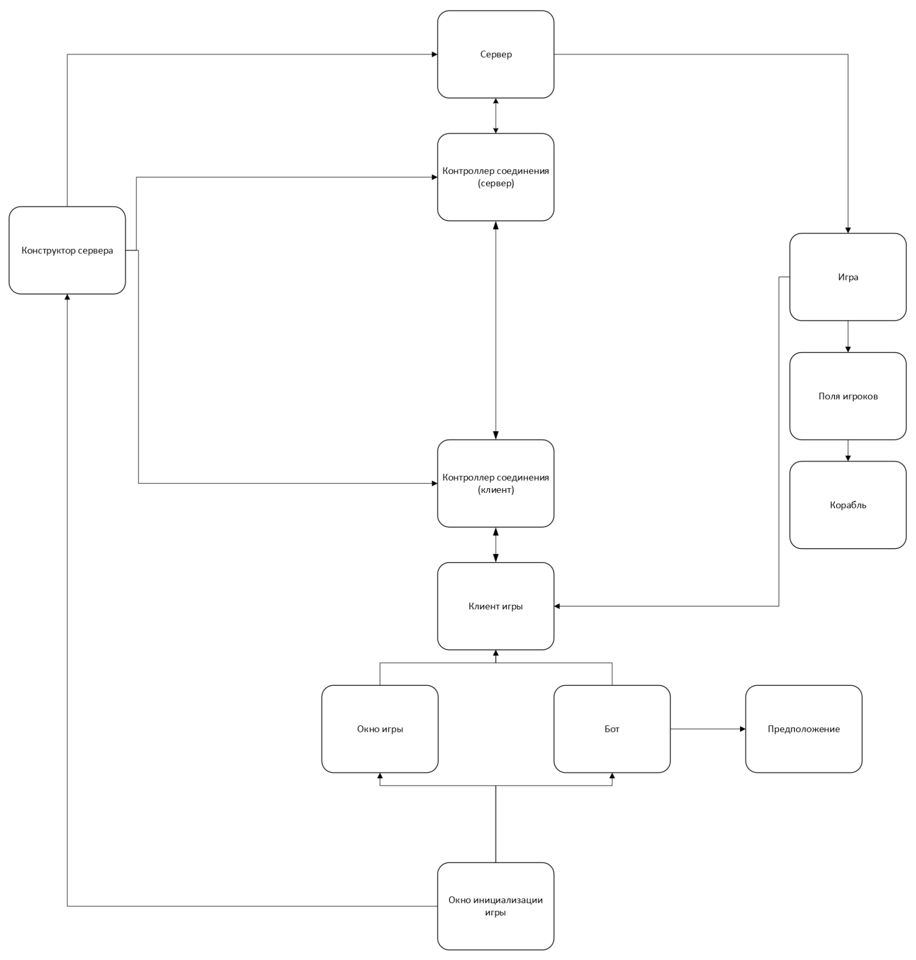
- Программа должна обеспечивать возможность игры человека с компьютером
- На экране должно отображаться два поля: собственное поле и поле для отметки ударов по противнику
- Должна быть предусмотрена возможность расставить (переместить, повернуть) корабли с различным кол-вом палуб
- В режиме игры игрок должен быть проинформирован о ходе игры

Планируется использовать следующие паттерны:

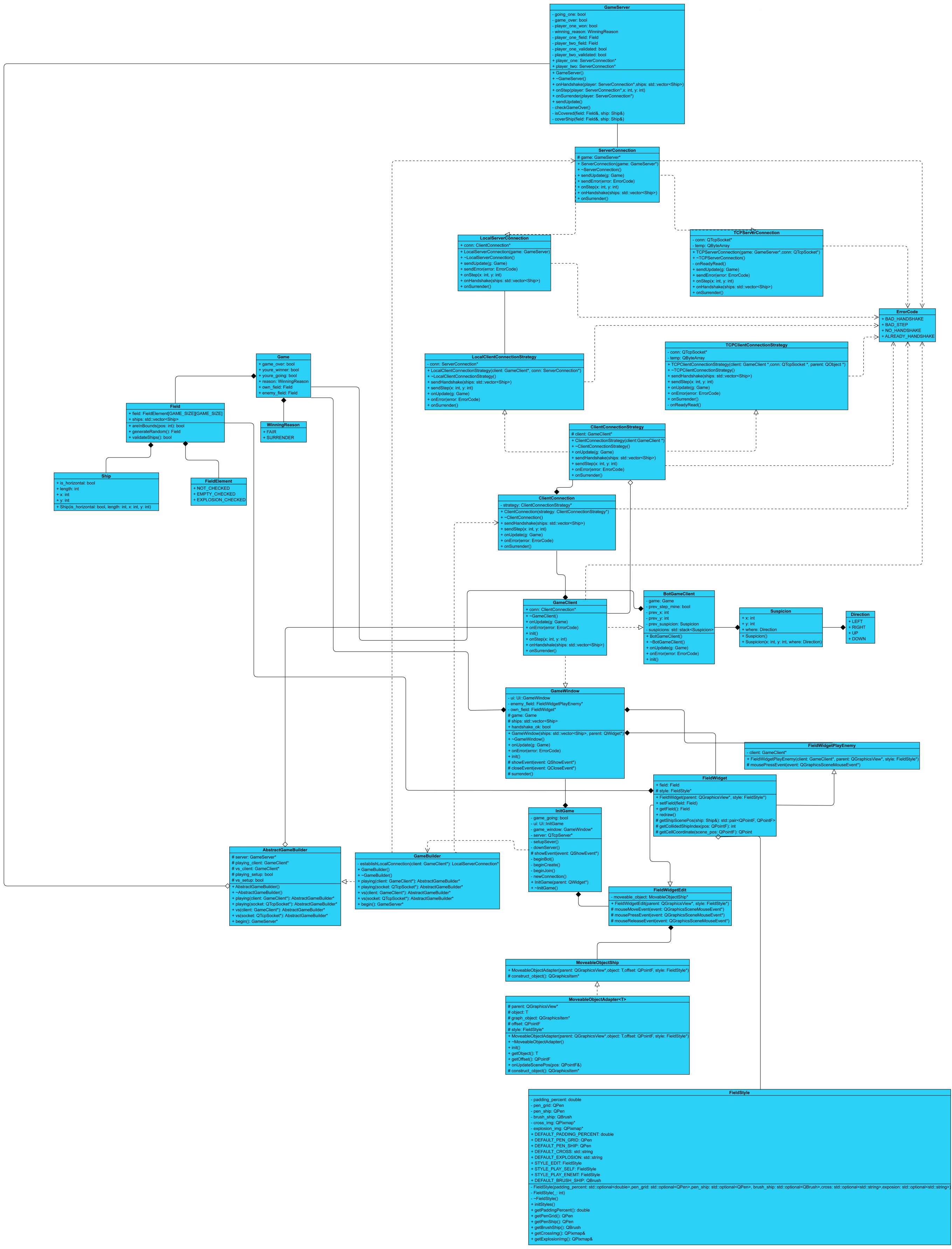
- Фабрика: для генерации игрового сервера, установки соединения сервер-игрок и игрок-сервер
- Стратегия: для создания различного поведения клиентского контроллера соединения в зависимости от вида соединения: программного или TCP-протокол
- Адаптер: для получения возможности отрисовывать на QGraphicsScene объекты, которые можно захватить и передвинуть при помощи мышки

2 Объектная декомпозиция (диаграмма объектов, классов)

2.1 Диаграмма объектов



2.2 UML-диаграмма


GameServer
- going_one: bool
- game_over: bool
- player_one_won: bool
- winning_reason: WinningReason
- player_one_field: Field
- player_two_field: Field
- player_one_validated: bool
- player_two_validated: bool
+ player_one: ServerConnection*
+ player_two: ServerConnection*
+ GameServer()
+ ~GameServer()
+ onHandshake(player: ServerConnection*,ships: std::vector<Ship>)
+ onStep(player: ServerConnection*,x: int, y: int)
+ onSurrender(player: ServerConnection*)
+ sendUpdate()
- checkGameOver()
- isCovered(field: Field&, ship: Ship&)
- coverShip(field: Field&, ship: Ship&)
ServerConnection
# game: GameServer*
+ ServerConnection(game: GameServer*)
+ ~ServerConnection()
+ sendUpdate(g: Game)
+ sendError(error: ErrorCode)
+ onStep(x: int, y: int)
+ onHandshake(ships: std::vector<Ship>)
+ onSurrender()
LocalServerConnection
+ conn: ClientConnection*
+ LocalServerConnection(game: GameServer)
+ ~LocalServerConnection()
+ sendUpdate(g: Game)
+ sendError(error: ErrorCode)
+ onStep(x: int, y: int)
+ onHandshake(ships: std::vector<Ship>)
+ onSurrender()
TCPServerConnection
- conn: QTcpSocket*
- temp: QByteArray
+ TCPServerConnection(game: GameServer*,conn: QTcpSocket*)
+ ~TCPServerConnection()
- onReadyRead()
+ sendUpdate(g: Game)
+ sendError(error: ErrorCode)
+ onStep(x: int, y: int)
+ onHandshake(ships: std::vector<Ship>)
+ onSurrender()
ErrorCode
+ BAD_HANDSHAKE
+ BAD_STEP
+ NO_HANDSHAKE
+ ALREADY_HANDSHAKE
Game
+ game_over: bool
+ youre_winner: bool
+ youre_going: bool
+ reason: WinningReason
+ own_field: Field
+ enemy_field: Field
Field
+ field: FieldElement[GAME_SIZE][GAME_SIZE]
+ ships: std::vector<Ship>
+ areInBounds(pos: int): bool
+ generateRandom(): Field
+ validateShips(): bool
WinningReason
+ FAIR
+ SURRENDER
Ship
+ is_horizontal: bool
+ length: int
+ x: int
+ y: int
+ Ship(is_horizontal: bool, length: int, x: int, y: int)
FieldElement
+ NOT_CHECKED
+ EMPTY_CHECKED
+ EXPLOSION_CHECKED
LocalClientConnectionStrategy
- conn: ServerConnection*
+ LocalClientConnectionStrategy(client: GameClient*, conn: ServerConnection*)
+ ~LocalClientConnectionStrategy()
+ sendHandshake(ships: std::vector<Ship>)
+ sendStep(x: int, y: int)
+ onUpdate(g: Game)
+ onError(error: ErrorCode)
+ onSurrender()
TCPClientConnectionStrategy
- conn: QTcpSocket*
- temp: QByteArray
+ TCPClientConnectionStrategy(client: GameClient *,conn: QTcpSocket *, parent: QObject *)
+ ~TCPClientConnectionStrategy()
+ sendHandshake(ships: std::vector<Ship>)
+ sendStep(x: int, y: int)
+ onUpdate(g: Game)
+ onError(error: ErrorCode)
+ onSurrender()
- onReadyRead()
ClientConnectionStrategy
# client: GameClient*
+ ClientConnectionStrategy(client:GameClient *)
+ ~ClientConnectionStrategy()
+ onUpdate(g: Game)
+ sendHandshake(ships: std::vector<Ship>)
+ sendStep(x: int, y: int)
+ onError(error: ErrorCode)
+ onSurrender()
ClientConnection
- strategy: ClientConnectionStrategy*
+ ClientConnection(strategy: ClientConnectionStrategy*)
+ ~ClientConnection()
+ sendHandshake(ships: std::vector<Ship>)
+ sendStep(x: int, y: int)
+ onUpdate(g: Game)
+ onError(error: ErrorCode)
+ onSurrender()
GameClient
+ conn: ClientConnection*
+ ~GameClient()
+ onUpdate(g: Game)
+ onError(error: ErrorCode)
+ init()
+ onStep(x: int, y: int)
+ onHandshale(ships: std::vector<Ship>)
+ onSurrender()
BotGameClient
- game: Game
- prev_step_mine: bool
- prev_x: int
- prev_y: int
- prev_suspicion: Suspicion
- suspicions: std::stack<Suspicion>
+ BotGameClient()
+ ~BotGameClient()
+ onUpdate(g: Game)
+ onError(error: ErrorCode)
+ init()
Suspicion
+ x: int
+ y: int
+ where: Direction
+ Suspicion()
+ Suspicion(x: int, y: int, where: Direction)
Direction
+ LEFT
+ RIGHT
+ UP
+ DOWN
GameWindow
- ui: Ui::GameWindow
- enemy_field: FieldWidgetPlayEnemy*
- own_field: FieldWidget*
# game: Game
# ships: std::vector<Ship>
+ handshake_ok: bool
+ GameWindow(ships: std::vector<Ship>, parent: QWidget*)
+ ~GameWindow()
+ onUpdate(g: Game)
+ onError(error: ErrorCode)
+ init()
# showEvent(event: QShowEvent*)
# closeEvent(event: QCloseEvent*)
# surrender()
FieldWidgetPlayEnemy
- client: GameClient*
+ FieldWidgetPlayEnemy(client: GameClient*, parent: QGraphicsView*, style: FieldStyle*)
# mousePressEvent(event: QGraphicsSceneMouseEvent*)
FieldWidget
+ field: Field
# style: FieldStyle*
+ FieldWidget(parent: QGraphicsView*, style: FieldStyle*)
+ setField(field: Field)
+ getField(): Field
+ redraw()
# getShipScenePos(ship: Ship&): std::pair<QPointF, QPointF>
# getCollidedShipIndex(pos: QPointF): int
# getCellCoordinate(scene_pos: QPointF): QPoint
InitGame
- game_going: bool
- ui: Ui::InitGame
- game_window: GameWindow*
- server: QTcpServer*
- setupSever()
- downServer()
# showEvent(event: QShowEvent*)
- beginBot()
- beginCreate()
- beginJoin()
- newConnection()
+ InitGame(parent: QWidget*)
+ ~InitGame()
AbstractGameBuilder
# server: GameServer*
# playing_client: GameClient*
# vs_client: GameClient*
# playing_setup: bool
# vs_setup: bool
+ AbstractGameBuilder()
+ ~AbstractGameBuilder()
+ playing(client: GameClient*): AbstractGameBuilder*
+ playing(socket: QTcpSocket*): AbstractGameBuilder*
+ vs(client: GameClient*): AbstractGameBuilder*
+ vs(socket: QTcpSocket*): AbstractGameBuilder*
+ begin(): GameServer*
GameBuilder
- establishLocalConnection(client: GameClient*): LocalServerConnection*
+ GameBuilder()
+ ~GameBuilder()
+ playing(client: GameClient*): AbstractGameBuilder*
+ playing(socket: QTcpSocket*): AbstractGameBuilder*
+ vs(client: GameClient*): AbstractGameBuilder*
+ vs(socket: QTcpSocket*): AbstractGameBuilder*
+ begin(): GameServer*
FieldWidgetEdit
- moveable_object: MovableObjectShip*
+ FieldWidgetEdit(parent: QGraphicsView*, style: FieldStyle*)
# mouseMoveEvent(event: QGraphicsSceneMouseEvent*)
# mousePressEvent(event: QGraphicsSceneMouseEvent*)
# mouseReleaseEvent(event: QGraphicsSceneMouseEvent*)
MoveableObjectShip
+ MoveableObjectAdapter(parent: QGraphicsView*,object: T,offset: QPointF, style: FieldStyle*)
# construct_object(): QGraphicsItem*
MoveableObjectAdapter<T>
# parent: QGraphicsView*
# object: T
# graph_object: QGraphicsItem*
# offset: QPointF
# style: FieldStyle*
+ MoveableObjectAdapter(parent: QGraphicsView*,object: T,offset: QPointF, style: FieldStyle*)
+ ~MoveableObjectAdapter()
+ init()
+ getObject(): T
+ getOffset(): QPointF
+ onUpdateScenePos(pos: QPointF&)
# construct_object(): QGraphicsItem*
FieldStyle
- padding_percent: double
- pen_grid: QPen
- pen_ship: QPen
- brush_ship: QBrush
- cross_img: QPixmap*
- explosion_img: QPixmap*
+ DEFAULT_PADDING_PERCENT: double
+ DEFAULT_PEN_GRID: QPen
+ DEFAULT_PEN_SHIP: QPen
+ DEFAULT_CROSS: std::string
+ DEFAULT_EXPLOSION: std::string
+ STYLE_EDIT: FieldStyle
+ STYLE_PLAY_SELF: FieldStyle
+ STYLE_PLAY_ENEMT: FieldStyle
+ DEFAULT_BRUSH_SHIP: QBrush
- FieldStyle(padding_percent: std::optional<double>,pen_grid: std::optional<QPen>,pen_ship: std::optional<QPen>, brush_ship: std::optional<QBrush>,cross: std::optional<std::string>,exposion: std::optional<std::string>)
- FieldStyle(_: int)
- ~FieldStyle()
+ initStyles()
+ getPaddingPercent(): double
+ getPenGrid(): QPen
+ getPenShip(): QPen
+ getBrushShip(): QBrush
+ getCrossImg(): QPixmap&
+ getExplosionImg(): QPixmap&

3 Листинг программы

cbattle.pro

```
QT       += core gui network

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++17

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \
    src/game/abstractgamebuilder.cpp \
    src/game/client/botclient.cpp \
    src/game/client/client.cpp \
    src/game/client/connection.cpp \
    src/game/client/connectionstrategy.cpp \
    src/game/client/localconnectionstrategy.cpp \
    src/game/client/tcpconnectionstrategy.cpp \
    src/game/server/conneciton.cpp \
    src/game/server/localconnection.cpp \
    src/game/server/server.cpp \
    src/game/gamebuilder.cpp \
    src/game/server/tcpconnection.cpp \
    src/models/field.cpp \
    src/models/game.cpp \
    src/models/ship.cpp \
    src/utils/moveableobjectship.cpp \
    src/widgets/fieldstyle.cpp \
    src/widgets/fieldwidget.cpp \
    src/widgets/fieldwidgetedit.cpp \
    src/widgets/fieldwidgetplayenemy.cpp \
    src/widgets/gamewindow.cpp \
    src/widgets/initgame.cpp \
    src/main.cpp

HEADERS += \
    include/game/abstractgamebuilder.h \
    include/game/client/botclient.h \
    include/game/client/client.h \
    include/game/client/connection.h \
    include/game/client/connectionstrategy.h \
    include/game/client/localconnectionstrategy.h \
    include/game/client/tcpconnectionstrategy.h \
    include/game/server/connection.h \
    include/game/server/localconnection.h \
    include/game/server/server.h \
    include/game/gamebuilder.h \
    include/game/server/tcpconnection.h \
    include/models/field.h \
    include/models/game.h \
    include/models/ship.h \
    include/utils/moveableobjectadapter.hpp \
    include/utils/moveableobjectship.h \
    include/widgets/fieldstyle.h \
    include/widgets/fieldwidget.h \
    include/widgets/fieldwidgetedit.h \
    include/widgets/fieldwidgetplayenemy.h \
    include/widgets/gamewindow.h \
    include/widgets/initgame.h

FORMS += \
    src/ui/game.ui \
    src/ui/initgame.ui \

INCLUDEPATH += include

RC_ICONS += src/assets/icon.ico

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

RESOURCES += \
    resources.qrc

DISTFILES += \
    model.qmodel
```

include \game \abstractgamebuilder.h

```
#pragma once

#include <QTcpSocket>

#include <game/client/client.h>
#include <game/server/localconnection.h>

class AbstractGameBuilder {
protected:
    GameServer *server = nullptr;
    GameClient *playing_client = nullptr;
    GameClient *vs_client = nullptr;
    bool playing_setup = false;
    bool vs_setup = false;
public:
    AbstractGameBuilder();
    virtual ~AbstractGameBuilder();
    virtual AbstractGameBuilder* playing(GameClient *client) = 0;
    virtual AbstractGameBuilder* playing(QTcpSocket *socket) = 0;
    virtual AbstractGameBuilder* vs(GameClient *client) = 0;
    virtual AbstractGameBuilder* vs(QTcpSocket *socket) = 0;
    virtual GameServer* begin() = 0;
};
```

include \game \client \botclient.h

```
#pragma once

#include <game/client/client.h>
#include <models/game.h>
#include <stack>

enum Direction { LEFT, RIGHT, UP, DOWN, UNKNOWN };

class Suspicion {
public:
  int x = 0;
  int y = 0;
  Direction where = UNKNOWN;

  Suspicion();
  Suspicion(int x, int y, Direction where);
};

class BotGameClient : public GameClient {
public:
  BotGameClient();
  virtual ~BotGameClient();

  void onUpdate(Game g);
  void onError(ErrorCode error);
  void init();

private:
  Game game;
  bool prev_step_mine = false;
  int prev_x = 0, prev_y = 0;
  Suspicion prev_suspicion = Suspicion();
  std::stack<Suspicion> suspicions;
};
```

include \game \client \client.h

```
#pragma once

#include <game/client/connection.h>
#include <game/server/server.h>
#include <models/game.h>

class GameClient {
public:
  ClientConnection *conn;

  virtual ~GameClient();

  virtual void onUpdate(Game g) = 0;
  virtual void onError(ErrorCode error) = 0;
  virtual void init() = 0;
  virtual void onStep(int x, int y);
  virtual void onHandshake(std::vector<Ship> ships);
  virtual void onSurrender();
};
```

include \game \client \connection.h

```
#pragma once

#include <game/client/connectionstrategy.h>
#include <game/server/server.h>
#include <models/field.h>
#include <models/game.h>

class ClientConnection {
  ClientConnectionStrategy *strategy;
public:
  ClientConnection(ClientConnectionStrategy *strategy);
  ~ClientConnection();

  void sendHandshake(std::vector<Ship> ships);
  void sendStep(int x, int y);
  void onUpdate(Game g);
  void onError(ErrorCode error);
  void onSurrender();
};
```

include \game \client \connectionstrategy.h

```
#pragma once

#include <game/server/server.h>
#include <models/field.h>
#include <models/game.h>

class GameClient;

class ClientConnectionStrategy {
protected:
  GameClient *client;
public:
  ClientConnectionStrategy(GameClient *client);
  virtual ~ClientConnectionStrategy();

  virtual void sendHandshake(std::vector<Ship> ships) = 0;
  virtual void sendStep(int x, int y) = 0;
  virtual void onUpdate(Game g) = 0;
  virtual void onError(ErrorCode error) = 0;
  virtual void onSurrender() = 0;
};
```

include \game \client \localconnectionstrategy.h

```
#pragma once

#include <models/field.h>
#include <models/game.h>

#include <game/client/connectionstrategy.h>

class ServerConnection;

class LocalClientConnectionStrategy : public ClientConnectionStrategy {
  ServerConnection *conn;
public:
  LocalClientConnectionStrategy(GameClient *client, ServerConnection *conn);
  ~LocalClientConnectionStrategy();
  void sendHandshake(std::vector<Ship> ships);
  void sendStep(int x, int y);
  void onUpdate(Game g);
  void onError(ErrorCode error);
  void onSurrender();
};
```

include \game \client \tcpconnectionstrategy.h

```
#pragma once

#include <QTcpSocket>

#include <game/client/connectionstrategy.h>
#include <models/field.h>
#include <models/game.h>

class TCPClientConnectionStrategy : public QObject,
                                    public ClientConnectionStrategy {
  Q_OBJECT

  QTcpSocket *conn;
  QByteArray temp;
  void onReadyRead();
public:
  TCPClientConnectionStrategy(GameClient *client, QTcpSocket *conn,
                              QObject *parent);
  ~TCPClientConnectionStrategy();
  void sendHandshake(std::vector<Ship> ships);
  void sendStep(int x, int y);
  void onUpdate(Game g);
  void onError(ErrorCode error);
  void onSurrender();
};
```

include \game \gamebuilder.h

```
#pragma once

#include <QTcpServer>

#include <game/client/client.h>
#include <game/server/localconnection.h>
#include <game/server/server.h>
#include <game/abstractgamebuilder.h>

class GameBuilder : public AbstractGameBuilder {
  LocalServerConnection *establishLocalConnection(GameClient *client);

public:
  GameBuilder();
  ~GameBuilder();
  AbstractGameBuilder* playing(GameClient *client);
  AbstractGameBuilder* playing(QTcpSocket *socket);
  AbstractGameBuilder* vs(GameClient *client);
  AbstractGameBuilder* vs(QTcpSocket *socket);
  GameServer* begin();
};
```

include \game \server \connection.h

```
#pragma once

#include <game/server/server.h>
#include <models/game.h>

class ServerConnection {
protected:
  GameServer *game;
public:
  ServerConnection(GameServer *game);
  virtual ~ServerConnection();

  virtual void sendUpdate(Game g) = 0;
  virtual void sendError(ErrorCode error) = 0;
  virtual void onStep(int x, int y) = 0;
  virtual void onHandshake(std::vector<Ship> ships) = 0;
  virtual void onSurrender() = 0;
};
```

include \game \server \localconnection.h

```
#pragma once

#include <game/client/connection.h>
#include <game/server/connection.h>
#include <game/server/localconnection.h>
#include <game/server/server.h>

class LocalServerConnection : public ServerConnection {
public:
  ClientConnection *conn;

  LocalServerConnection(GameServer *game);
  virtual ~LocalServerConnection();

  void sendUpdate(Game g);
  void sendError(ErrorCode error);
  void onStep(int x, int y);
  void onHandshake(std::vector<Ship> ships);
  void onSurrender();
};
```

include \game \server \server.h

```
#pragma once

#include <models/field.h>

class ServerConnection;

enum ErrorCode {
  BAD_HANDSHAKE = 0,
  BAD_STEP = 1,
  NO_HANDSHAKE = 2,
  ALREADY_HANDSHAKE = 3
};

enum WinningReason { FAIR = 0, SURRENDER = 1 };

class GameServer {
  bool going_one = true;
  bool game_over = false;
  bool player_one_won = false;
  WinningReason winning_reason = FAIR;
  Field player_one_field;
  bool player_one_validated = false;
  Field player_two_field;
  bool player_two_validated = false;
  static bool isCovered(Field &field, Ship &ship);
  static void coverShip(Field &field, Ship &ship);
  void checkGameOver();

public:
  GameServer();
  ~GameServer();

  ServerConnection *player_one;
  ServerConnection *player_two;

  void onHandshake(ServerConnection *player, std::vector<Ship> ships);
  void onStep(ServerConnection *player, int x, int y);
  void onSurrender(ServerConnection *player);
  void sendUpdate();
};
```

include \game \server \tcpconnection.h

```
#pragma once

#include <QObject>
#include <QTcpSocket>

#include <game/server/connection.h>
#include <game/server/server.h>

class TCPServerConnection : public QObject, public ServerConnection {
  Q_OBJECT

  QTcpSocket *conn;
  QByteArray temp;
  void onReadyRead();
public:
  TCPServerConnection(GameServer *game, QTcpSocket *conn);
  virtual ~TCPServerConnection();

  void sendUpdate(Game g);
  void sendError(ErrorCode error);
  void onStep(int x, int y);
  void onHandshake(std::vector<Ship> ships);
  void onSurrender();
};
```

include \models \field.h

```
#pragma once

#include <vector>

#include <models/ship.h>

#define GAME_SIZE 10
#define ONE_SHIP 4
#define TWO_SHIP 3
#define THREE_SHIP 2
#define FOUR_SHIP 1

enum FieldElement { NOT_CHECKED = 0, EMPTY_CHECKED = 1, EXPOLSION_CHECKED = 2 };

class Field {
public:
  FieldElement field[GAME_SIZE][GAME_SIZE] = {};
  std::vector<Ship> ships;
  static bool areInBounds(int pos);
  static Field generateRandom();
  bool validateShips();
};
```

include \models \game.h

```
#pragma once

#include <game/server/server.h>
#include <models/field.h>

class Game {
public:
  bool game_over = false;
  bool youre_winner;
  bool youre_going;
  WinningReason reason = FAIR;
  Field own_field;
  Field enemy_field;
};
```

include \models \ship.h

```
#pragma once

class Ship {
public:
  bool is_horizontal = true;
  int length = 1;
  int x = 0;
  int y = 0;

  Ship(bool is_horizontal, int length, int x, int y);
};
```

include \utils \moveableobjectadapter.hpp

```
#pragma once

#include <QGraphicsItem>
#include <QGraphicsView>
```

```
#include <widgets/fieldstyle.h>

template <typename T> class MoveableObjectAdapter {
protected:
  QGraphicsView *parent;
  T object;
  QGraphicsItem *graph_object;
  QPointF offset;
  FieldStyle *style;

  virtual QGraphicsItem *construct_object() = 0;

public:
  void onUpdateScenePos(QPointF &pos) {
    graph_object->setPos(
        QPointF(pos.x() + this->offset.x(), pos.y() + this->offset.y()));
  }

  MoveableObjectAdapter(QGraphicsView *parent, T object, QPointF offset,
                        FieldStyle *style)
      : parent(parent), object(object), offset(offset), style(style) {}

  virtual ~MoveableObjectAdapter() {
    this->parent->scene()->removeItem(graph_object);
    delete graph_object;
  }

  void init() {
    this->graph_object = construct_object();

    this->parent->scene()->addItem(graph_object);
  }

  T getObject() { return object; }

  QPointF getOffset() { return offset; }
};
```

include \utils \moveableobjectship.h

```
#pragma once

#include <QGraphicsItem>
#include <QGraphicsScene>

#include <models/ship.h>
#include <utils/moveableobjectadapter.hpp>
```

```
class MoveableObjectShip : public MoveableObjectAdapter<Ship> {
protected:
  QGraphicsItem *construct_object();

public:
  MoveableObjectShip(QGraphicsView *parent, Ship object, QPointF offset,
                     FieldStyle *style);
};
```

include \widgets \fieldstyle.h

```
#pragma once

#include <QBrush>
#include <QPen>

class FieldStyle {
  double padding_percent;
  QPen pen_grid;
  QPen pen_ship;
  QBrush brush_ship;
  QPixmap *cross_img;
  QPixmap *explosion_img;

  FieldStyle(std::optional<double> padding_percent = std::nullopt,
             std::optional<QPen> pen_grid = std::nullopt,
             std::optional<QPen> pen_ship = std::nullopt,
             std::optional<QBrush> brush_ship = std::nullopt,
             std::optional<std::string> cross = std::nullopt,
             std::optional<std::string> exposion = std::nullopt);

  FieldStyle(int _);

  ~FieldStyle();

public:
  static double DEFAULT_PADDING_PERCENT;
  static QPen DEFAULT_PEN_GRID;
  static QPen DEFAULT_PEN_SHIP;
  static QBrush DEFAULT_BRUSH_SHIP;
  static std::string DEFAULT_CROSS;
  static std::string DEFAULT_EXPLOSION;

  static FieldStyle STYLE_EDIT;
  static FieldStyle STYLE_PLAY_SELF;
  static FieldStyle STYLE_PLAY_ENEMY;

  static void initStyles();
  double getPaddingPercent();
  QPen getPenGrid();
  QPen getPenShip();
  QBrush getBrushShip();
  QPixmap &getCrossImg();
  QPixmap &getExplosionImg();
};
```

include \widgets \fieldwidget.h

```
#pragma once

#include <QColor>
#include <QGraphicsItem>
#include <QGraphicsScene>
#include <QGraphicsSceneMouseEvent>
#include <QGraphicsView>
#include <QPen>

#include <models/field.h>
#include <models/ship.h>
#include <widgets/fieldstyle.h>

class FieldWidget : public QGraphicsScene {
  Q_OBJECT
public:
  Field field;
  FieldWidget(QGraphicsView *parent = nullptr,
              FieldStyle *style = &FieldStyle::STYLE_PLAY_SELF);

  void setField(Field field);
  Field getField();
  void redraw();
protected:
  FieldStyle *style;
  std::pair<QPointF, QPointF> getShipScenePos(Ship &ship);
  int getCollidedShipIndex(QPointF pos);
  QPoint getCellCoordinate(QPointF scene_pos);
};
```

include \widgets \fieldwidgetedit.h

```
#pragma once

#include <utils/moveableobjectship.h>
#include <widgets/fieldwidget.h>

class FieldWidgetEdit : public FieldWidget {
  Q_OBJECT

  MoveableObjectShip *moveable_object = nullptr;

public:
  FieldWidgetEdit(QGraphicsView *parent = nullptr,
                  FieldStyle *style = &FieldStyle::STYLE_EDIT);

protected:
  void mouseMoveEvent(QGraphicsSceneMouseEvent *event);
  void mousePressEvent(QGraphicsSceneMouseEvent *event);
  void mouseReleaseEvent(QGraphicsSceneMouseEvent *event);
};
```

include \widgets \fieldwidgetplayenemy.h

```
#pragma once

#include <game/client/client.h>
#include <widgets/fieldwidget.h>

class FieldWidgetPlayEnemy : public FieldWidget {
  Q_OBJECT
  GameClient *client;

public:
  FieldWidgetPlayEnemy(GameClient *client, QGraphicsView *parent = nullptr,
                       FieldStyle *style = &FieldStyle::STYLE_PLAY_ENEMY);

protected:
  void mousePressEvent(QGraphicsSceneMouseEvent *event);
};
```

include \widgets \gamewindow.h

```
#pragma once

#include <QDialog>
#include <QTcpServer>
```

```cpp
#include <game/client/client.h>
#include <widgets/fieldwidgetplayenemy.h>

namespace Ui {
class GameWindow;
}

class GameWindow : public QDialog, public GameClient {
  Q_OBJECT

  Ui::GameWindow *ui;
  FieldWidgetPlayEnemy *enemy_field = nullptr;
  FieldWidget *own_field = nullptr;
public:
  bool handshake_ok = true;

  explicit GameWindow(std::vector<Ship> ships, QWidget *parent = nullptr);
  ~GameWindow();
  void onUpdate(Game g);
  void onError(ErrorCode error);
  void init();

protected:
  Game game;
  std::vector<Ship> ships;
  void showEvent(QShowEvent *event);
  void closeEvent(QCloseEvent *event);
  void surrender();
};
```

include \widgets \initgame.h

```cpp
#pragma once

#include <QMainWindow>

#include <models/field.h>
#include <models/ship.h>
#include <widgets/fieldwidgetedit.h>
#include <widgets/gamewindow.h>

namespace Ui {
class InitGame;
}

class InitGame : public QMainWindow {
  Q_OBJECT
  bool game_going = false;
  Ui::InitGame *ui;
  GameWindow *game_window;
  QTcpServer *server = nullptr;
  void setupServer();
  void downServer();
  void beginBot();
  void beginCreate();
  void beginJoin();
  void newConnection();

public:
  explicit InitGame(QWidget *parent = nullptr);
  ~InitGame();

protected:
  void showEvent(QShowEvent *event);
};
```

src \game \abstractgamebuilder.cpp

```
#include <game/abstractgamebuilder.h>

AbstractGameBuilder::AbstractGameBuilder() {}

AbstractGameBuilder::~AbstractGameBuilder() {}
```

src \game \client \botclient.cpp

```
#include <game/client/botclient.h>

#include <QDebug>
#include <random>

Suspicion::Suspicion() {}
Suspicion::Suspicion(int x, int y, Direction where)
    : x(x), y(y), where(where) {}

BotGameClient::BotGameClient() : GameClient() {}

BotGameClient::~BotGameClient() {
}

void BotGameClient::onUpdate(Game g) {
```

```
this->game = g;
if (g.game_over)
  return;
if (!g.youre_going) {
  prev_step_mine = false;
  return;
}

if (prev_step_mine && prev_suspicion.where != UNKNOWN) {
  Suspicion sus;
  sus.where = prev_suspicion.where;
  sus.x = prev_suspicion.x + (sus.where == LEFT    ? -1
                            : sus.where == RIGHT ? 1
                                                 : 0);
  sus.y = prev_suspicion.y + (sus.where == UP     ? -1
                            : sus.where == DOWN ? 1
                                                : 0);

  this->suspicions.push(sus);
} else if (prev_step_mine) {
  std::vector<Suspicion> suses = {Suspicion(prev_x - 1, prev_y, LEFT),
                                  Suspicion(prev_x + 1, prev_y, RIGHT),
                                  Suspicion(prev_x, prev_y - 1, UP),
                                  Suspicion(prev_x, prev_y + 1, DOWN)};
  std::random_shuffle(suses.begin(), suses.end());
  for (auto &sus : suses) {
    this->suspicions.push(sus);
  }
}

if (!this->suspicions.empty()) {
  auto sus = suspicions.top();
  suspicions.pop();
  prev_suspicion = sus;
  prev_x = sus.x;
  prev_y = sus.y;

  prev_step_mine = true;
  this->onStep(sus.x, sus.y);

  return;
}

while (true) {
  static auto dev = std::random_device();
  static auto gen = std::mt19937{dev()};
  static auto dist = std::uniform_int_distribution(0, GAME_SIZE - 1);

  int y = dist(gen), x = dist(gen);
  if (g.enemy_field.field[y][x] == FieldElement::NOT_CHECKED) {
      prev_step_mine = true;
      prev_x = x;
      prev_y = y;
      prev_suspicion = Suspicion();

      this->onStep(x, y);

      return;
    }
  }
}

void BotGameClient::onError(ErrorCode error) {
  switch (error) {
  case BAD_HANDSHAKE:
    break;
  case BAD_STEP:
    prev_step_mine = false;
    onUpdate(this->game);
    break;
  case NO_HANDSHAKE:
    init();
    break;
  case ALREADY_HANDSHAKE:
    break;
  }
}

void BotGameClient::init() {
  auto data = Field::generateRandom();
  this->onHandshake(data.ships);
}
```

src \game \client \client.cpp

```
#include <game/client/client.h>
#include <game/client/connection.h>

GameClient::~GameClient() {
    delete this->conn;
}

void GameClient::onStep(int x, int y) { return conn->sendStep(x, y); }

void GameClient::onHandshake(std::vector<Ship> ships) {
  return conn->sendHandshake(ships);
}
```

```cpp
void GameClient::onSurrender() { return conn->onSurrender(); }
```

src \game \client \connection.cpp

```cpp
#include <game/client/connection.h>

ClientConnection::ClientConnection(ClientConnectionStrategy *strategy)
    : strategy(strategy) {}

ClientConnection::~ClientConnection() { delete strategy; }

void ClientConnection::sendHandshake(std::vector<Ship> ships) {
  return strategy->sendHandshake(ships);
}

void ClientConnection::sendStep(int x, int y) {
  return strategy->sendStep(x, y);
}

void ClientConnection::onUpdate(Game g) { return strategy->onUpdate(g); }

void ClientConnection::onError(ErrorCode error) {
  return strategy->onError(error);
}

void ClientConnection::onSurrender() { return strategy->onSurrender(); }
```

src \game \client \connectionstrategy.cpp

```cpp
#include <game/client/connectionstrategy.h>

ClientConnectionStrategy::ClientConnectionStrategy(GameClient *client)
    : client(client) {}

ClientConnectionStrategy::~ClientConnectionStrategy() {}
```

src \game \client \localconnectionstrategy.cpp

```cpp
#include <game/client/client.h>
#include <game/client/localconnectionstrategy.h>
#include <game/server/connection.h>

LocalClientConnectionStrategy::LocalClientConnectionStrategy(
    GameClient *client, ServerConnection *conn)
    : ClientConnectionStrategy(client), conn(conn) {}

LocalClientConnectionStrategy::~LocalClientConnectionStrategy() {}

void LocalClientConnectionStrategy::sendHandshake(std::vector<Ship> ships) {
  conn->onHandshake(ships);
}

void LocalClientConnectionStrategy::sendStep(int x, int y) {
  conn->onStep(x, y);
}

void LocalClientConnectionStrategy::onUpdate(Game g) {
  this->client->onUpdate(g);
}

void LocalClientConnectionStrategy::onError(ErrorCode error) {
  this->client->onError(error);
}

void LocalClientConnectionStrategy::onSurrender() { conn->onSurrender(); }
```

src \game \client \tcpconnectionstrategy.cpp

```
#include <game/client/client.h>
#include <game/client/tcpconnectionstrategy.h>
#include <game/server/connection.h>

#include <sstream>

TCPClientConnectionStrategy::TCPClientConnectionStrategy(GameClient *client,
                                                         QTcpSocket *conn,
                                                         QObject *parent)
    : QObject(parent), ClientConnectionStrategy(client), conn(conn) {
  connect(conn, &QTcpSocket::readyRead, this,
          &TCPClientConnectionStrategy::onReadyRead);
}

TCPClientConnectionStrategy::~TCPClientConnectionStrategy() {
  conn->flush();
  conn->close();
  delete conn;
}

void TCPClientConnectionStrategy::onReadyRead() {
  QTcpSocket *clientSocket = qobject_cast<QTcpSocket *>(sender());
  if (!clientSocket)
    return;

  QByteArray data = clientSocket->readAll();
  for (auto &b : data) {
    if (b != '\n') {
      temp.append(b);
      continue;
    }

    if (temp.toStdString().rfind("error: ", 0) == 0) {
      std::stringstream input(temp.toStdString().substr(7));
      int code;
      input >> code;
      onError(static_cast<ErrorCode>(code));
    } else if (temp.toStdString().rfind("update: ", 0) == 0) {
      std::stringstream input(temp.toStdString().substr(8));
      Game g;
      int dat;
      input >> dat;
      g.game_over = dat == 1;
      input >> dat;
      g.youre_winner = dat == 1;
      input >> dat;
      g.youre_going = dat == 1;
      input >> dat;
      g.reason = static_cast<WinningReason>(dat);

      for (int i = 0; i < GAME_SIZE; i++) {
        for (int j = 0; j < GAME_SIZE; j++) {
          input >> dat;
          g.own_field.field[i][j] = static_cast<FieldElement>(dat);
        }
      }
      int sh_amount;
      input >> sh_amount;
      for (int i = 0; i < sh_amount; i++) {
        input >> dat;
        bool is_horizontal = dat == 1;
        int length;
        input >> length;
        int x;
        input >> x;
        int y;
        input >> y;
        g.own_field.ships.push_back(Ship(is_horizontal, length, x, y));
      }

      for (int i = 0; i < GAME_SIZE; i++) {
        for (int j = 0; j < GAME_SIZE; j++) {
          input >> dat;
```

```
          g.enemy_field.field[i][j] = static_cast<FieldElement>(dat);
        }
      }
      input >> sh_amount;
      for (int i = 0; i < sh_amount; i++) {
        input >> dat;
        bool is_horizontal = dat == 1;
        int length;
        input >> length;
        int x;
        input >> x;
        int y;
        input >> y;
        g.enemy_field.ships.push_back(Ship(is_horizontal, length, x, y));
      }

      onUpdate(g);
    }

    temp.clear();
  }

  // Parse data from server here:
}

void TCPClientConnectionStrategy::sendHandshake(std::vector<Ship> ships) {
  std::stringstream output;
  output << "handshake: " << ships.size() << " ";
  for (auto &ship : ships) {
    output << (ship.is_horizontal ? 1 : 0) << " " << ship.length << " "
           << ship.x << " " << ship.y << " ";
  }
  output << "\n";

  conn->write(output.str().c_str());
  conn->flush();
}

void TCPClientConnectionStrategy::sendStep(int x, int y) {
  std::stringstream output;
  output << "step: " << x << " " << y << "\n";
  conn->write(output.str().c_str());
  conn->flush();
}

void TCPClientConnectionStrategy::onSurrender() {
  std::stringstream output;
  output << "surrender: \n";
  conn->write(output.str().c_str());
  conn->flush();
```

```
}

void TCPClientConnectionStrategy::onUpdate(Game g) {
  this->client->onUpdate(g);
}

void TCPClientConnectionStrategy::onError(ErrorCode error) {
  this->client->onError(error);
}
```

src \game \gamebuilder.cpp

```
#include <game/gamebuilder.h>

#include <QTcpSocket>
#include <game/client/connection.h>
#include <game/client/localconnectionstrategy.h>
#include <game/server/server.h>
#include <game/server/tcpconnection.h>

LocalServerConnection *
GameBuilder::establishLocalConnection(GameClient *client) {
  auto server_conn = new LocalServerConnection(server);

  auto host_client_connection = new ClientConnection(
      new LocalClientConnectionStrategy(client, server_conn));
  client->conn = host_client_connection;
  server_conn->conn = host_client_connection;

  return server_conn;
}

GameBuilder::GameBuilder() : AbstractGameBuilder() { this->server = new GameServer(); };

GameBuilder::~GameBuilder() {}

AbstractGameBuilder* GameBuilder::playing(GameClient *client) {
  server->player_one = establishLocalConnection(client);
  playing_client = client;
  playing_setup = true;

  return this;
}

AbstractGameBuilder* GameBuilder::playing(QTcpSocket *socket) {
  server->player_one = new TCPServerConnection(this->server, socket);
  playing_setup = true;
  return this;
}

AbstractGameBuilder* GameBuilder::vs(GameClient *client) {
  server->player_two = establishLocalConnection(client);
  vs_client = client;
  vs_setup = true;

  return this;
}

AbstractGameBuilder* GameBuilder::vs(QTcpSocket *socket) {
  server->player_two = new TCPServerConnection(this->server, socket);
  vs_setup = true;

  return this;
}

GameServer* GameBuilder::begin() {
  if (!playing_setup) throw std::runtime_error("Player is not set up, call playing... method in
  ↪  builder");
  if (!vs_setup) throw std::runtime_error("Player is not set up, call vs... method in builder");
  if (playing_client)
    playing_client->init();
  if (vs_client)
    vs_client->init();

  return this->server;
}
```

src \game \server \conneciton.cpp

```
#include <game/server/connection.h>

ServerConnection::ServerConnection(GameServer *game) : game(game){};
ServerConnection::~ServerConnection() {}
```

src \game \server \localconnection.cpp

```
#include <game/server/localconnection.h>

LocalServerConnection::LocalServerConnection(GameServer *game)
    : ServerConnection(game) {}
LocalServerConnection::~LocalServerConnection() {}
```

```
void LocalServerConnection::sendUpdate(Game g) { this->conn->onUpdate(g); }

void LocalServerConnection::sendError(ErrorCode error) {
  this->conn->onError(error);
}

void LocalServerConnection::onStep(int x, int y) {
  this->game->onStep(this, x, y);
}

void LocalServerConnection::onHandshake(std::vector<Ship> ships) {
  this->game->onHandshake(this, ships);
}

void LocalServerConnection::onSurrender() { this->game->onSurrender(this); }
```

src \game \server \server.cpp

```
#include <algorithm>
#include <game/server/connection.h>
#include <game/server/server.h>

#include <random>

#include <models/game.h>

GameServer::GameServer() {
  static auto dev = std::random_device();
  static auto gen = std::mt19937{dev()};
  static auto dist = std::uniform_int_distribution(0, 1);
  this->going_one = dist(gen) == 0;
}

GameServer::~GameServer() {
  delete this->player_one;
  delete this->player_two;
}

void GameServer::onHandshake(ServerConnection *player,
                             std::vector<Ship> ships) {
  bool was = this->player_one_validated && this->player_two_validated;
  if (was)
    return player->sendError(ALREADY_HANDSHAKE);

  if (this->player_one == player) {
    if (this->player_one_validated)
      return player->sendError(ALREADY_HANDSHAKE);
    this->player_one_field.ships = ships;
    if (this->player_one_field.validateShips())
      this->player_one_validated = true;
    else
      return player->sendError(BAD_HANDSHAKE);
  }

  if (this->player_two == player) {
    if (this->player_two_validated)
      return player->sendError(ALREADY_HANDSHAKE);

    this->player_two_field.ships = ships;
    if (this->player_two_field.validateShips())
      this->player_two_validated = true;
    else
      return player->sendError(BAD_HANDSHAKE);
  }

  if (!was && this->player_one_validated && this->player_two_validated) {
    sendUpdate();
  }
}

void GameServer::onStep(ServerConnection *player, int xx, int yy) {
  if (game_over)
    return;
  if (player == player_one && !this->player_one_validated)
    return player->sendError(ErrorCode::NO_HANDSHAKE);
  if (player == player_two && !this->player_two_validated)
    return player->sendError(ErrorCode::NO_HANDSHAKE);

  if (!Field::areInBounds(xx) || !Field::areInBounds(yy))
    return player->sendError(ErrorCode::BAD_STEP);
  Field *selected_field = &player_one_field;
  bool selected = false;

  if (player == player_one && going_one) {
    selected_field = &player_two_field;
    selected = true;
  } else if (player == player_two && !going_one) {
    selected = true;
  }

  if (!selected)
    return player->sendError(ErrorCode::BAD_STEP);

  if (selected_field->field[yy][xx] != FieldElement::NOT_CHECKED)
    return player->sendError(ErrorCode::BAD_STEP);

  for (auto &ship : selected_field->ships) {
    if (ship.is_horizontal) {
      for (int x = ship.x; x < ship.x + ship.length; x++) {
        if (ship.y != yy || x != xx)
          continue;

        selected_field->field[yy][xx] = FieldElement::EXPOLSION_CHECKED;
        if (isCovered(*selected_field, ship))
          coverShip(*selected_field, ship);

        checkGameOver();
        sendUpdate();
        return;
      }
    } else {
      for (int y = ship.y; y < ship.y + ship.length; y++) {
        if (y != yy || ship.x != xx)
          continue;

        selected_field->field[yy][xx] = FieldElement::EXPOLSION_CHECKED;
        if (isCovered(*selected_field, ship))
          coverShip(*selected_field, ship);

        checkGameOver();
        sendUpdate();
        return;
      }
    }
  }

  selected_field->field[yy][xx] = FieldElement::EMPTY_CHECKED;
  going_one = !going_one;
  sendUpdate();
}

void GameServer::onSurrender(ServerConnection *player) {
  if (game_over)
    return;
  if (player == player_one && !this->player_one_validated)
    return player->sendError(ErrorCode::NO_HANDSHAKE);
  if (player == player_two && !this->player_two_validated)
    return player->sendError(ErrorCode::NO_HANDSHAKE);

  if (player == player_one) {
    this->game_over = true;
    this->winning_reason = SURRENDER;
    this->player_one_won = false;

    sendUpdate();
  } else if (player == player_two) {
    this->game_over = true;
    this->winning_reason = SURRENDER;
    this->player_one_won = true;

    sendUpdate();
  }
}

void GameServer::checkGameOver() {
  if (game_over || !this->player_one_validated || !this->player_two_validated)
    return;

  bool all_player_one = true;
  for (auto &ship : this->player_one_field.ships) {
    if (!isCovered(this->player_one_field, ship)) {
      all_player_one = false;
      break;
    }
  }

  if (all_player_one) {
    this->game_over = true;
    this->player_one_won = false;
    this->winning_reason = FAIR;

    return;
  }

  bool all_player_two = true;
  for (auto &ship : this->player_two_field.ships) {
    if (!isCovered(this->player_two_field, ship)) {
      all_player_two = false;
      break;
    }
  }

  if (all_player_two) {
    this->game_over = true;
    this->player_one_won = true;
    this->winning_reason = FAIR;
  }
}

void GameServer::sendUpdate() {
  Game p1_game;
  p1_game.reason = this->winning_reason;
  p1_game.youre_going = going_one;
  p1_game.own_field = player_one_field;
  p1_game.enemy_field = player_two_field;
  p1_game.enemy_field.ships.clear();
  std::copy_if(player_two_field.ships.begin(), player_two_field.ships.end(),
             std::back_inserter(p1_game.enemy_field.ships),
             [&](Ship sh) { return isCovered(player_two_field, sh); });
  p1_game.game_over = this->game_over;
  p1_game.youre_winner = this->player_one_won;

  player_one->sendUpdate(p1_game);

  Game p2_game;
  p2_game.reason = this->winning_reason;
  p2_game.youre_going = !going_one;
  p2_game.own_field = player_two_field;
  p2_game.enemy_field = player_one_field;
  p2_game.enemy_field.ships.clear();
  std::copy_if(player_one_field.ships.begin(), player_one_field.ships.end(),
             std::back_inserter(p2_game.enemy_field.ships),
             [&](Ship sh) { return isCovered(player_one_field, sh); });
  p2_game.game_over = this->game_over;
  p2_game.youre_winner = !this->player_one_won;

  player_two->sendUpdate(p2_game);
}

bool GameServer::isCovered(Field &field, Ship &ship) {
  if (ship.is_horizontal) {
    for (int x = ship.x; x < ship.x + ship.length; x++)
      if (field.field[ship.y][x] != FieldElement::EXPOLSION_CHECKED)
        return false;
  } else {
    for (int y = ship.y; y < ship.y + ship.length; y++)
      if (field.field[y][ship.x] != FieldElement::EXPOLSION_CHECKED)
        return false;
  }

  return true;
}

void GameServer::coverShip(Field &field, Ship &ship) {
  if (ship.is_horizontal) {
    for (int x = ship.x; x < ship.x + ship.length; x++) {
      if (Field::areInBounds(ship.y - 1) && Field::areInBounds(x))
        field.field[ship.y - 1][x] = FieldElement::EMPTY_CHECKED;
      if (Field::areInBounds(ship.y + 1) && Field::areInBounds(x))
        field.field[ship.y + 1][x] = FieldElement::EMPTY_CHECKED;
      if (Field::areInBounds(ship.y - 1) && Field::areInBounds(x - 1))
        field.field[ship.y - 1][x - 1] = FieldElement::EMPTY_CHECKED;
      if (Field::areInBounds(ship.y - 1) && Field::areInBounds(x + 1))
        field.field[ship.y - 1][x + 1] = FieldElement::EMPTY_CHECKED;
      if (Field::areInBounds(ship.y + 1) && Field::areInBounds(x + 1))
        field.field[ship.y + 1][x + 1] = FieldElement::EMPTY_CHECKED;
      if (Field::areInBounds(ship.y + 1) && Field::areInBounds(x - 1))
```

```
        field.field[ship.y + 1][x - 1] = FieldElement::EMPTY_CHECKED;
    }

    if (Field::areInBounds(ship.y) && Field::areInBounds(ship.x - 1))
      field.field[ship.y][ship.x - 1] = FieldElement::EMPTY_CHECKED;
    if (Field::areInBounds(ship.y) && Field::areInBounds(ship.x + ship.length))
      field.field[ship.y][ship.x + ship.length] = FieldElement::EMPTY_CHECKED;
  } else {
    for (int y = ship.y; y < ship.y + ship.length; y++) {
      if (Field::areInBounds(y) && Field::areInBounds(ship.x + 1))
        field.field[y][ship.x + 1] = FieldElement::EMPTY_CHECKED;
      if (Field::areInBounds(y) && Field::areInBounds(ship.x - 1))
        field.field[y][ship.x - 1] = FieldElement::EMPTY_CHECKED;
      if (Field::areInBounds(y - 1) && Field::areInBounds(ship.x - 1))
        field.field[y - 1][ship.x - 1] = FieldElement::EMPTY_CHECKED;
      if (Field::areInBounds(y - 1) && Field::areInBounds(ship.x + 1))
        field.field[y - 1][ship.x + 1] = FieldElement::EMPTY_CHECKED;
      if (Field::areInBounds(y + 1) && Field::areInBounds(ship.x + 1))
        field.field[y + 1][ship.x + 1] = FieldElement::EMPTY_CHECKED;
      if (Field::areInBounds(y + 1) && Field::areInBounds(ship.x - 1))
        field.field[y + 1][ship.x - 1] = FieldElement::EMPTY_CHECKED;
    }

    if (Field::areInBounds(ship.y - 1) && Field::areInBounds(ship.x))
      field.field[ship.y - 1][ship.x] = FieldElement::EMPTY_CHECKED;
    if (Field::areInBounds(ship.y + ship.length) && Field::areInBounds(ship.x))
      field.field[ship.y + ship.length][ship.x] = FieldElement::EMPTY_CHECKED;
  }
}
```

src \game \server \tcpconnection.cpp

```
#include <game/server/tcpconnection.h>

#include <sstream>

TCPServerConnection::TCPServerConnection(GameServer *game, QTcpSocket *conn)
    : QObject(nullptr), ServerConnection(game), conn(conn) {
  this->conn = conn;
  connect(conn, &QTcpSocket::readyRead, this,
          &TCPServerConnection::onReadyRead);
  connect(conn, &QTcpSocket::disconnected, this,
          &TCPServerConnection::onSurrender);
}

TCPServerConnection::~TCPServerConnection() {
  conn->flush();
  conn->close();
  delete conn;
}

void TCPServerConnection::onReadyRead() {
  QTcpSocket *clientSocket = qobject_cast<QTcpSocket *>(sender());
  if (!clientSocket)
    return;

  QByteArray data = clientSocket->readAll();
  for (auto &b : data) {
    if (b != '\n') {
      temp.append(b);
      continue;
    }

    if (temp.toStdString().rfind("handshake: ", 0) == 0) {
      std::stringstream input(temp.toStdString().substr(11));
      int sh_amount;
      input >> sh_amount;
      std::vector<Ship> ships;
      for (int i = 0; i < sh_amount; i++) {
        int dat;
        input >> dat;
        bool is_horizontal = dat == 1;
        int length;
        input >> length;
        int x;
        input >> x;
        int y;
        input >> y;
        ships.push_back(Ship(is_horizontal, length, x, y));
      }

      onHandshake(ships);
    } else if (temp.toStdString().rfind("step: ", 0) == 0) {
      std::stringstream input(temp.toStdString().substr(6));
      int x, y;
      input >> x >> y;

      onStep(x, y);
    } else if (temp.toStdString().rfind("surrender: ", 0) == 0) {
      onSurrender();
    }

    temp.clear();
  }
}

void TCPServerConnection::sendUpdate(Game g) {
  std::stringstream output;
  output << "update: " << (g.game_over ? 1 : 0) << " "
         << (g.youre_winner ? 1 : 0) << " " << (g.youre_going ? 1 : 0) << " "
         << static_cast<int>(g.reason) << " ";

  for (int i = 0; i < GAME_SIZE; i++) {
    for (int j = 0; j < GAME_SIZE; j++) {
      output << static_cast<int>(g.own_field.field[i][j]) << " ";
    }
  }
  output << g.own_field.ships.size() << " ";
  for (auto &ship : g.own_field.ships) {
    output << (ship.is_horizontal ? 1 : 0) << " " << ship.length << " "
           << ship.x << " " << ship.y << " ";
  }

  for (int i = 0; i < GAME_SIZE; i++) {
    for (int j = 0; j < GAME_SIZE; j++) {
      output << static_cast<int>(g.enemy_field.field[i][j]) << " ";
    }
  }
  output << g.enemy_field.ships.size() << " ";
  for (auto &ship : g.enemy_field.ships) {
    output << (ship.is_horizontal ? 1 : 0) << " " << ship.length << " "
           << ship.x << " " << ship.y << " ";
  }

  output << "\n";

  conn->write(output.str().c_str());
  conn->flush();
}

void TCPServerConnection::sendError(ErrorCode error) {
  std::stringstream output;
  output << "error: " << static_cast<int>(error) << "\n";

  conn->write(output.str().c_str());
  conn->flush();
}

void TCPServerConnection::onStep(int x, int y) {
  this->game->onStep(this, x, y);
}

void TCPServerConnection::onHandshake(std::vector<Ship> ships) {
  this->game->onHandshake(this, ships);
}

void TCPServerConnection::onSurrender() { this->game->onSurrender(this); }
```

src \main.cpp

```
#include <QApplication>

#include <QTcpSocket>
#include <widgets/initgame.h>

int main(int argc, char *argv[]) {
  QApplication a(argc, argv);

  FieldStyle::initStyles();
  InitGame init_game;
  init_game.show();

  return a.exec();
}
```

src \models \field.cpp

```
#include <models/field.h>

#include <random>

#include <models/ship.h>

Field Field::generateRandom() {
  std::random_device dev;
  std::mt19937 rng(dev());
  std::uniform_int_distribution<std::mt19937::result_type> dist_game_size(
      0, GAME_SIZE - 1);
  std::uniform_int_distribution<std::mt19937::result_type> dist_bool(0, 1);
  Field result;

  while (1) {
    bool should_keep_going = true;
    result.ships.clear();
    for (int i = 0; i < ONE_SHIP && should_keep_going; i++) {
      result.ships.push_back(Ship(dist_bool(rng) == 1 ? true : false, 1,
                                  dist_game_size(rng), dist_game_size(rng)));

      if (!result.validateShips())
        should_keep_going = false;
      ;
    }
    for (int i = 0; i < TWO_SHIP && should_keep_going; i++) {
      result.ships.push_back(Ship(dist_bool(rng) == 1 ? true : false, 2,
                                  dist_game_size(rng), dist_game_size(rng)));

      if (!result.validateShips())
        should_keep_going = false;
    }

    for (int i = 0; i < THREE_SHIP && should_keep_going; i++) {
      result.ships.push_back(Ship(dist_bool(rng) == 1 ? true : false, 3,
                                  dist_game_size(rng), dist_game_size(rng)));

      if (!result.validateShips())
        should_keep_going = false;
    }

    for (int i = 0; i < FOUR_SHIP && should_keep_going; i++) {
      result.ships.push_back(Ship(dist_bool(rng) == 1 ? true : false, 4,
                                  dist_game_size(rng), dist_game_size(rng)));

      if (!result.validateShips())
        should_keep_going = false;
    }

    if (should_keep_going)
      break;
  }

  return result;
}

bool Field::validateShips() {
  std::vector<std::vector<int>> test_field(GAME_SIZE,
                                           std::vector(GAME_SIZE, 0));

  for (auto &ship : ships) {
    if (ship.is_horizontal) {
      if (!areInBounds(ship.y))
        return false;

      for (int x = ship.x; x < ship.x + ship.length; x++) {
        if (!areInBounds(x))
          return false;

        if (test_field[ship.y][x] != 0)
          return false;
        test_field[ship.y][x] = 2;

        if (areInBounds(ship.y - 1) && areInBounds(x))
          test_field[ship.y - 1][x] = 1;
        if (areInBounds(ship.y + 1) && areInBounds(x))
          test_field[ship.y + 1][x] = 1;
        if (areInBounds(ship.y - 1) && areInBounds(x - 1))
          test_field[ship.y - 1][x - 1] = 1;
        if (areInBounds(ship.y - 1) && areInBounds(x + 1))
          test_field[ship.y - 1][x + 1] = 1;
        if (areInBounds(ship.y + 1) && areInBounds(x + 1))
          test_field[ship.y + 1][x + 1] = 1;
        if (areInBounds(ship.y + 1) && areInBounds(x - 1))
          test_field[ship.y + 1][x - 1] = 1;
      }

      if (areInBounds(ship.y) && areInBounds(ship.x - 1))
        test_field[ship.y][ship.x - 1] = 1;
      if (areInBounds(ship.y) && areInBounds(ship.x + ship.length))
        test_field[ship.y][ship.x + ship.length] = 1;
    } else {
      if (!areInBounds(ship.x))
        return false;

      for (int y = ship.y; y < ship.y + ship.length; y++) {
        if (!areInBounds(y))
          return false;

        if (test_field[y][ship.x] != 0)
          return false;
        test_field[y][ship.x] = 2;

        if (areInBounds(y) && areInBounds(ship.x + 1))
          test_field[y][ship.x + 1] = 1;
        if (areInBounds(y) && areInBounds(ship.x - 1))
          test_field[y][ship.x - 1] = 1;
        if (areInBounds(y - 1) && areInBounds(ship.x - 1))
          test_field[y - 1][ship.x - 1] = 1;
        if (areInBounds(y - 1) && areInBounds(ship.x + 1))
          test_field[y - 1][ship.x + 1] = 1;
        if (areInBounds(y + 1) && areInBounds(ship.x + 1))
          test_field[y + 1][ship.x + 1] = 1;
        if (areInBounds(y + 1) && areInBounds(ship.x - 1))
          test_field[y + 1][ship.x - 1] = 1;
      }

      if (areInBounds(ship.y - 1) && areInBounds(ship.x))
        test_field[ship.y - 1][ship.x] = 1;
      if (areInBounds(ship.y + ship.length) && areInBounds(ship.x))
        test_field[ship.y + ship.length][ship.x] = 1;
    }
  }
```

```
  return true;
}

bool Field::areInBounds(int pos) { return pos >= 0 && pos < GAME_SIZE; }
```

src \models \game.cpp

```
#include <models/game.h>
```

src \models \ship.cpp

```
#include <models/ship.h>

Ship::Ship(bool is_horizontal, int length, int x, int y)
    : is_horizontal(is_horizontal), length(length), x(x), y(y) {}
```

src \ui \game.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>GameWindow</class>
 <widget class="QDialog" name="GameWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>950</width>
    <height>600</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Морской бой</string>
  </property>
  <property name="windowIcon">
   <iconset resource="../../resources.qrc">
    <normaloff>:/src/assets/icon.png</normaloff>:/src/assets/icon.png</iconset>
  </property>
  <widget class="QWidget" name="centralwidget" native="true">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>950</width>
     <height>600</height>
    </rect>
   </property>
   <widget class="QWidget" name="verticalLayoutWidget">
    <property name="geometry">
     <rect>
      <x>25</x>
      <y>25</y>
      <width>900</width>
      <height>550</height>
     </rect>
    </property>
    <layout class="QVBoxLayout" name="verticalLayout">
     <item>
      <layout class="QHBoxLayout" name="horizontalLayout">
       <item>
        <layout class="QVBoxLayout" name="verticalLayout_2">
         <item>
          <widget class="QLabel" name="label_2">
           <property name="styleSheet">
            <string notr="true">font: 700 14pt &quot;Segoe UI&quot;; color: rgb(0, 0, 0);</string>
           </property>
           <property name="text">
            <string>Поле соперника:</string>
           </property>
          </widget>
         </item>
         <item>
          <widget class="QGraphicsView" name="enemyfield">
           <property name="sizePolicy">
            <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
             <horstretch>0</horstretch>
             <verstretch>0</verstretch>
            </sizepolicy>
           </property>
           <property name="minimumSize">
            <size>
             <width>400</width>
             <height>400</height>
            </size>
           </property>
           <property name="maximumSize">
            <size>
             <width>400</width>
             <height>400</height>
            </size>
           </property>
           <property name="verticalScrollBarPolicy">
            <enum>Qt::ScrollBarAlwaysOff</enum>
           </property>
           <property name="horizontalScrollBarPolicy">
```

```
            <enum>Qt::ScrollBarAlwaysOff</enum>
           </property>
          </widget>
         </item>
        </layout>
       </item>
       <item>
        <spacer name="horizontalSpacer">
         <property name="orientation">
          <enum>Qt::Horizontal</enum>
         </property>
         <property name="sizeHint" stdset="0">
          <size>
           <width>40</width>
           <height>20</height>
          </size>
         </property>
        </spacer>
       </item>
       <item>
        <layout class="QVBoxLayout" name="verticalLayout_3">
         <item>
          <widget class="QLabel" name="label_3">
           <property name="styleSheet">
            <string notr="true">font: 700 14pt &quot;Segoe UI&quot;; color: rgb(0, 0, 0);</string>
           </property>
           <property name="text">
            <string>Ваше поле:</string>
           </property>
          </widget>
         </item>
         <item>
          <widget class="QGraphicsView" name="yourfield">
           <property name="sizePolicy">
            <sizepolicy hsizetype="Preferred" vsizetype="Preferred">
             <horstretch>0</horstretch>
             <verstretch>0</verstretch>
            </sizepolicy>
           </property>
           <property name="minimumSize">
            <size>
             <width>400</width>
             <height>400</height>
            </size>
           </property>
           <property name="maximumSize">
            <size>
             <width>400</width>
             <height>400</height>
            </size>
           </property>
           <property name="verticalScrollBarPolicy">
            <enum>Qt::ScrollBarAlwaysOff</enum>
           </property>
           <property name="horizontalScrollBarPolicy">
            <enum>Qt::ScrollBarAlwaysOff</enum>
           </property>
          </widget>
         </item>
        </layout>
       </item>
      </layout>
     </item>
     <item>
      <widget class="QLabel" name="whose_turn">
       <property name="styleSheet">
        <string notr="true">font: 700 16pt &quot;Segoe UI&quot;; color: rgb(0, 0, 0);</string>
       </property>
       <property name="text">
        <string>Ожидаем противника...</string>
       </property>
       <property name="alignment">
        <set>Qt::AlignCenter</set>
       </property>
      </widget>
     </item>
     <item>
      <spacer name="verticalSpacer">
       <property name="orientation">
        <enum>Qt::Vertical</enum>
       </property>
       <property name="sizeHint" stdset="0">
        <size>
         <width>20</width>
         <height>40</height>
        </size>
       </property>
      </spacer>
     </item>
     <item>
      <layout class="QHBoxLayout" name="horizontalLayout_2">
       <item>
        <spacer name="horizontalSpacer_2">
         <property name="orientation">
          <enum>Qt::Horizontal</enum>
         </property>
         <property name="sizeHint" stdset="0">
          <size>
           <width>40</width>
           <height>20</height>
          </size>
         </property>
        </spacer>
       </item>
       <item>
        <widget class="QPushButton" name="surrender">
         <property name="sizePolicy">
          <sizepolicy hsizetype="Maximum" vsizetype="Fixed">
           <horstretch>0</horstretch>
           <verstretch>0</verstretch>
          </sizepolicy>
         </property>
         <property name="text">
          <string>Выйти из игры</string>
         </property>
        </widget>
       </item>
      </layout>
     </item>
    </layout>
   </widget>
  </widget>
 </widget>
 <resources>
  <include location="../../resources.qrc"/>
 </resources>
 <connections/>
</ui>
```

src \ui \initgame.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>InitGame</class>
 <widget class="QMainWindow" name="InitGame">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>650</width>
    <height>450</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Морской Бой</string>
  </property>
  <property name="windowIcon">
   <iconset resource="../../resources.qrc">
```

```
    <normaloff>:/src/assets/icon.png</normaloff>:/src/assets/icon.png</iconset>
  </property>
  <widget class="QWidget" name="centralwidget">
   <widget class="QWidget" name="horizontalLayoutWidget">
    <property name="geometry">
     <rect>
      <x>25</x>
      <y>25</y>
      <width>600</width>
      <height>400</height>
     </rect>
    </property>
    <layout class="QHBoxLayout" name="horizontalLayout" stretch="2,1">
     <item>
      <widget class="QGraphicsView" name="field">
       <property name="sizePolicy">
        <sizepolicy hsizetype="Maximum" vsizetype="Maximum">
         <horstretch>0</horstretch>
         <verstretch>0</verstretch>
        </sizepolicy>
       </property>
       <property name="maximumSize">
        <size>
         <width>600</width>
         <height>400</height>
        </size>
       </property>
       <property name="verticalScrollBarPolicy">
        <enum>Qt::ScrollBarAlwaysOff</enum>
       </property>
       <property name="horizontalScrollBarPolicy">
        <enum>Qt::ScrollBarAlwaysOff</enum>
       </property>
      </widget>
     </item>
     <item>
      <layout class="QVBoxLayout" name="verticalLayout">
       <item>
        <spacer name="verticalSpacer">
         <property name="orientation">
          <enum>Qt::Vertical</enum>
         </property>
         <property name="sizeHint" stdset="0">
          <size>
           <width>20</width>
           <height>40</height>
          </size>
         </property>
        </spacer>
       </item>
       <item alignment="Qt::AlignHCenter">
        <widget class="QPushButton" name="begin_bot">
         <property name="sizePolicy">
          <sizepolicy hsizetype="Maximum" vsizetype="Fixed">
           <horstretch>0</horstretch>
           <verstretch>0</verstretch>
          </sizepolicy>
         </property>
         <property name="layoutDirection">
          <enum>Qt::LeftToRight</enum>
         </property>
         <property name="text">
          <string>Начать игру с ботом</string>
         </property>
        </widget>
       </item>
       <item>
        <spacer name="verticalSpacer_3">
         <property name="orientation">
          <enum>Qt::Vertical</enum>
         </property>
         <property name="sizeHint" stdset="0">
          <size>
           <width>20</width>
           <height>40</height>
          </size>
         </property>
        </spacer>
       </item>
       <item>
        <layout class="QHBoxLayout" name="horizontalLayout_2" stretch="2,1">
         <item>
          <widget class="QTextEdit" name="join_ip">
           <property name="sizePolicy">
            <sizepolicy hsizetype="Preferred" vsizetype="Maximum">
             <horstretch>0</horstretch>
             <verstretch>0</verstretch>
            </sizepolicy>
           </property>
           <property name="maximumSize">
            <size>
             <width>16777215</width>
             <height>30</height>
            </size>
           </property>
           <property name="placeholderText">
            <string>IP-адрес</string>
           </property>
          </widget>
         </item>
         <item>
          <widget class="QTextEdit" name="join_port">
           <property name="sizePolicy">
            <sizepolicy hsizetype="Preferred" vsizetype="Maximum">
             <horstretch>0</horstretch>
             <verstretch>0</verstretch>
            </sizepolicy>
           </property>
           <property name="maximumSize">
            <size>
             <width>16777215</width>
             <height>30</height>
            </size>
           </property>
           <property name="placeholderText">
            <string>Порт</string>
           </property>
          </widget>
         </item>
        </layout>
       </item>
       <item alignment="Qt::AlignHCenter">
        <widget class="QPushButton" name="begin_join">
         <property name="sizePolicy">
          <sizepolicy hsizetype="Maximum" vsizetype="Fixed">
           <horstretch>0</horstretch>
           <verstretch>0</verstretch>
          </sizepolicy>
         </property>
         <property name="text">
          <string>Присоединиться к игре</string>
         </property>
        </widget>
       </item>
       <item>
        <spacer name="verticalSpacer_4">
         <property name="orientation">
          <enum>Qt::Vertical</enum>
         </property>
         <property name="sizeHint" stdset="0">
          <size>
           <width>20</width>
           <height>40</height>
          </size>
         </property>
        </spacer>
       </item>
       <item alignment="Qt::AlignHCenter">
        <widget class="QPushButton" name="begin_create">
         <property name="sizePolicy">
```

```
          <sizepolicy hsizetype="Maximum" vsizetype="Fixed">
           <horstretch>0</horstretch>
           <verstretch>0</verstretch>
          </sizepolicy>
         </property>
         <property name="text">
          <string>Создать игру</string>
         </property>
        </widget>
       </item>
       <item>
        <spacer name="verticalSpacer_2">
         <property name="orientation">
          <enum>Qt::Vertical</enum>
         </property>
         <property name="sizeHint" stdset="0">
          <size>
           <width>20</width>
           <height>40</height>
          </size>
         </property>
        </spacer>
       </item>
      </layout>
     </item>
    </layout>
   </widget>
  </widget>
 </widget>
 <resources>
  <include location="../../resources.qrc"/>
 </resources>
 <connections/>
</ui>
```

src \utils \moveableobjectship.cpp

```
#include <utils/moveableobjectship.h>

#include <widgets/fieldwidget.h>

MoveableObjectShip::MoveableObjectShip(QGraphicsView *parent, Ship object,
                                       QPointF offset, FieldStyle *style)
    : MoveableObjectAdapter<Ship>(parent, object, offset, style) {
  init();
}

QGraphicsItem *MoveableObjectShip::construct_object() {
  double height = parent->height() - style->getPenGrid().width();
  double width = parent->width() - style->getPenGrid().width();
  double tile_size_height = height / GAME_SIZE;
  double tile_size_width = width / GAME_SIZE;

  double beg_x = style->getPaddingPercent() * tile_size_width;
  double end_x = (((this->object.is_horizontal ? this->object.length : 1)) -
                  style->getPaddingPercent()) *
                 tile_size_width;
  double beg_y = style->getPaddingPercent() * tile_size_height;
  double end_y = (((this->object.is_horizontal ? 1 : this->object.length)) -
                  style->getPaddingPercent()) *
                 tile_size_height;

  auto res = new QGraphicsRectItem(beg_x, beg_y, end_x, end_y);
  res->setPen(style->getPenShip());
  res->setBrush(style->getBrushShip());

  return res;
}
```

src \widgets \fieldstyle.cpp

```
#include <QImage>
#include <widgets/fieldstyle.h>

double FieldStyle::DEFAULT_PADDING_PERCENT = 0.15;
QPen FieldStyle::DEFAULT_PEN_GRID = QPen(Qt::gray, 3);
QPen FieldStyle::DEFAULT_PEN_SHIP = QPen(Qt::blue, 3);
QBrush FieldStyle::DEFAULT_BRUSH_SHIP = QColor(0, 0, 255, 30);
std::string FieldStyle::DEFAULT_CROSS = ":/src/assets/cross_self.png";
std::string FieldStyle::DEFAULT_EXPLOSION = ":/src/assets/boom.png";

FieldStyle FieldStyle::STYLE_EDIT = FieldStyle(1);
FieldStyle FieldStyle::STYLE_PLAY_SELF = FieldStyle(1);
FieldStyle FieldStyle::STYLE_PLAY_ENEMY = FieldStyle(1);

FieldStyle::FieldStyle(std::optional<double> padding_percent,
                       std::optional<QPen> pen_grid,
                       std::optional<QPen> pen_ship,
                       std::optional<QBrush> brush_ship,
                       std::optional<std::string> cross,
                       std::optional<std::string> explosion)
    : padding_percent(
          padding_percent.value_or(FieldStyle::DEFAULT_PADDING_PERCENT)),
      pen_grid(pen_grid.value_or(FieldStyle::DEFAULT_PEN_GRID)),
      pen_ship(pen_ship.value_or(FieldStyle::DEFAULT_PEN_SHIP)),
      brush_ship(brush_ship.value_or(FieldStyle::DEFAULT_BRUSH_SHIP)) {
```

```
  std::string cross_path = cross.value_or(FieldStyle::DEFAULT_CROSS);
  std::string explosion_path =
      explosion.value_or(FieldStyle::DEFAULT_EXPLOSION);
  this->cross_img = new QPixmap(QString::fromStdString(cross_path));
  this->explosion_img = new QPixmap(QString::fromStdString(explosion_path));
}

FieldStyle::FieldStyle(int _) {
  delete this->cross_img;
  delete this->explosion_img;
}

void FieldStyle::initStyles() {
  FieldStyle::STYLE_EDIT = FieldStyle();
  FieldStyle::STYLE_PLAY_SELF = FieldStyle();
  FieldStyle::STYLE_PLAY_ENEMY =
      FieldStyle(std::nullopt, std::nullopt,
                 QPen(Qt::red, FieldStyle::DEFAULT_PEN_SHIP.widthF()),
                 QColor(255, 0, 0, 30), ":/src/assets/cross_enemy.png",
                 ":/src/assets/boom.png");
}

FieldStyle::~FieldStyle() {}

double FieldStyle::getPaddingPercent() { return this->padding_percent; }
QPen FieldStyle::getPenGrid() { return this->pen_grid; }
QPen FieldStyle::getPenShip() { return this->pen_ship; }
QBrush FieldStyle::getBrushShip() { return this->brush_ship; }
QPixmap &FieldStyle::getCrossImg() { return *this->cross_img; }
QPixmap &FieldStyle::getExplosionImg() { return *this->explosion_img; }
```

src \widgets \fieldwidget.cpp

```
#include <widgets/fieldwidget.h>

#include <QMouseEvent>
#include <QPixmap>

FieldWidget::FieldWidget(QGraphicsView *parent, FieldStyle *style)
      : QGraphicsScene(parent), style(style) {}

void FieldWidget::setField(Field field) {
  this->field = field;

  redraw();
}

Field FieldWidget::getField() { return this->field; }
```

```
void FieldWidget::redraw() {
  this->clear();
  auto parent = reinterpret_cast<QGraphicsView *>(this->parent());
  double height = parent->height() - style->getPenGrid().width();
  double width = parent->width() - style->getPenGrid().width();
  double tile_size_height = height / GAME_SIZE;
  double tile_size_width = width / GAME_SIZE;

  for (int i = 0; i < GAME_SIZE + 1; i++) {
    this->addLine(0, i * tile_size_height, parent->width(),
                  i * tile_size_height, style->getPenGrid());
    this->addLine(i * tile_size_width, 0, i * tile_size_width, parent->height(),
                  style->getPenGrid());
  }

  for (auto &ship : field.ships) {
    auto ship_data = getShipScenePos(ship);

    this->addRect(ship_data.first.x(), ship_data.first.y(),
                  ship_data.second.x() - ship_data.first.x(),
                  ship_data.second.y() - ship_data.first.y(),
                  style->getPenShip(), style->getBrushShip());
  }

  for (int y = 0; y < GAME_SIZE; y++) {
    for (int x = 0; x < GAME_SIZE; x++) {
      if (field.field[y][x] == FieldElement::EMPTY_CHECKED) {
        auto p = new QGraphicsPixmapItem(style->getCrossImg().scaled(
            tile_size_width * (1 - style->getPaddingPercent()),
            tile_size_height * (1 - style->getPaddingPercent())));
        p->setPos((x + style->getPaddingPercent()) * tile_size_width,
                  (y + style->getPaddingPercent()) * tile_size_height);
        this->addItem(p);
      } else if (field.field[y][x] == FieldElement::EXPOLSION_CHECKED) {
        auto p = new QGraphicsPixmapItem(style->getExplosionImg().scaled(
            tile_size_width * (1 - style->getPaddingPercent()),
            tile_size_height * (1 - style->getPaddingPercent())));
        p->setPos((x + style->getPaddingPercent()) * tile_size_width,
                  (y + style->getPaddingPercent()) * tile_size_height);
        this->addItem(p);
      }
    }
  }
}

int FieldWidget::getCollidedShipIndex(QPointF pos) {
  for (int i = 0; i < this->field.ships.size(); i++) {
    auto ship_data = getShipScenePos(this->field.ships[i]);
    if (pos.x() >= ship_data.first.x() && pos.x() <= ship_data.second.x() &&
        pos.y() >= ship_data.first.y() && pos.y() <= ship_data.second.y())
      return i;
  }

  return -1;
}

std::pair<QPointF, QPointF> FieldWidget::getShipScenePos(Ship &ship) {
  auto parent = reinterpret_cast<QGraphicsView *>(this->parent());
  double height = parent->height() - style->getPenGrid().width();
  double width = parent->width() - style->getPenGrid().width();
  double tile_size_height = height / GAME_SIZE;
  double tile_size_width = width / GAME_SIZE;

  double beg_x = (ship.x + style->getPaddingPercent()) * tile_size_width;
  double end_x = ((ship.x + (ship.is_horizontal ? ship.length : 1)) -
                  style->getPaddingPercent()) *
                 tile_size_width;
  double beg_y = (ship.y + style->getPaddingPercent()) * tile_size_height;
  double end_y = ((ship.y + (ship.is_horizontal ? 1 : ship.length)) -
                  style->getPaddingPercent()) *
                 tile_size_height;

  return {QPointF(beg_x, beg_y), QPointF(end_x, end_y)};
}

QPoint FieldWidget::getCellCoordinate(QPointF scene_pos) {
  auto parent = reinterpret_cast<QGraphicsView *>(this->parent());
  double height = parent->height() - style->getPenGrid().width();
  double width = parent->width() - style->getPenGrid().width();
  double tile_size_height = height / GAME_SIZE;
  double tile_size_width = width / GAME_SIZE;

  int x = scene_pos.x() / tile_size_width;
  int y = scene_pos.y() / tile_size_height;

  return {x, y};
}
```

src \widgets \fieldwidgetedit.cpp

```
#include <widgets/fieldwidgetedit.h>

FieldWidgetEdit::FieldWidgetEdit(QGraphicsView *parent, FieldStyle *style)
    : FieldWidget(parent, style) {}

void FieldWidgetEdit::mouseMoveEvent(QGraphicsSceneMouseEvent *event) {
  if (moveable_object != nullptr) {
    QPointF newPos = event->scenePos();
    moveable_object->onUpdateScenePos(newPos);
  }
}

void FieldWidgetEdit::mousePressEvent(QGraphicsSceneMouseEvent *event) {
  QPointF newPos = event->scenePos();
  if (event->button() == Qt::LeftButton) {
    auto index = getCollidedShipIndex(event->scenePos());
    if (index == -1)
      return;

    Ship ship = this->field.ships[index];
    auto ship_data = getShipScenePos(ship);
    this->field.ships.erase(this->field.ships.begin() + index);
    this->redraw();

    if (moveable_object != nullptr)
      delete moveable_object;
    this->moveable_object = new MoveableObjectShip(
        reinterpret_cast<QGraphicsView *>(this->parent()), ship,
        QPointF(ship_data.first.x() - event->scenePos().x(),
                ship_data.first.y() - event->scenePos().y()),
        style);
    this->moveable_object->onUpdateScenePos(newPos);
  } else if (event->button() == Qt::RightButton && this->moveable_object) {
    auto ship = this->moveable_object->getObject();
    ship.is_horizontal = !ship.is_horizontal;
    auto offset = this->moveable_object->getOffset();

    delete this->moveable_object;
    this->moveable_object = new MoveableObjectShip(
        reinterpret_cast<QGraphicsView *>(this->parent()), ship, offset, style);
    this->moveable_object->onUpdateScenePos(newPos);
  }
}

void FieldWidgetEdit::mouseReleaseEvent(QGraphicsSceneMouseEvent *event) {
  if (event->button() == Qt::LeftButton && moveable_object) {
    auto parent = reinterpret_cast<QGraphicsView *>(this->parent());
    double height = parent->height() - style->getPenGrid().width();
    double width = parent->width() - style->getPenGrid().width();
    double tile_size_height = height / GAME_SIZE;
    double tile_size_width = width / GAME_SIZE;
    auto ship = moveable_object->getObject();
    auto cell = getCellCoordinate(
        {event->scenePos().x() + moveable_object->getOffset().x() +
             0.5 * tile_size_width, // moveable_object->getOffset().x(),
         event->scenePos().y() + moveable_object->getOffset().y() +
              0.5 * tile_size_height}); //- moveable_object->getOffset().y()});
    ship.x = cell.x();
    ship.y = cell.y();

    if (ship.x < 0)
      ship.x = 0;
    if (ship.y < 0)
      ship.y = 0;
    if (ship.x + (ship.is_horizontal ? ship.length : 1) > GAME_SIZE)
      ship.x = GAME_SIZE - (ship.is_horizontal ? ship.length : 1);
    if (ship.y + (!ship.is_horizontal ? ship.length : 1) > GAME_SIZE)
      ship.y = GAME_SIZE - (!ship.is_horizontal ? ship.length : 1);

    delete this->moveable_object;
    moveable_object = nullptr;

    this->field.ships.push_back(ship);
    this->redraw();
  }
}
```

src \widgets \fieldwidgetplayenemy.cpp

```
#include <widgets/fieldwidgetplayenemy.h>

FieldWidgetPlayEnemy::FieldWidgetPlayEnemy(GameClient *client,
                                           QGraphicsView *parent,
                                           FieldStyle *style)
    : FieldWidget(parent, style), client(client) {}

void FieldWidgetPlayEnemy::mousePressEvent(QGraphicsSceneMouseEvent *event) {
  if (event->button() == Qt::LeftButton) {
    auto cell =
        getCellCoordinate({event->scenePos().x(), event->scenePos().y()});

    this->client->onStep(cell.x(), cell.y());
  }
}
```

src \widgets \gamewindow.cpp

```
#include <QPen>
#include <QTcpServer>
#include <game/client/connection.h>
#include <widgets/fieldwidgetplayenemy.h>
```

```cpp
#include <widgets/gamewindow.h>

#include "ui_game.h"

GameWindow::GameWindow(std::vector<Ship> ships, QWidget *parent)
    : QDialog(parent), ui(new Ui::GameWindow) {
  ui->setupUi(this);
  this->ships = ships;

  ui->enemyfield->setScene(nullptr);
  ui->yourfield->setScene(nullptr);

  this->connect(ui->surrender, &QPushButton::clicked, this,
                &GameWindow::surrender);
}

GameWindow::~GameWindow() {
  delete enemy_field;
  delete ui;
}

void GameWindow::showEvent(QShowEvent *event) {
  QDialog::showEvent(event);

  if (this->ui->enemyfield->scene() == nullptr) {
    this->enemy_field = new FieldWidgetPlayEnemy(this, ui->enemyfield,
                                                 &FieldStyle::STYLE_PLAY_ENEMY);
    this->ui->enemyfield->setScene(this->enemy_field);
    this->enemy_field->field = this->game.enemy_field;
    this->enemy_field->redraw();
  }
  if (this->ui->yourfield->scene() == nullptr) {
    this->own_field =
        new FieldWidget(ui->yourfield, &FieldStyle::STYLE_PLAY_SELF);
    this->ui->yourfield->setScene(this->own_field);
    this->own_field->field = this->game.own_field;
    this->own_field->redraw();
  }
}

void GameWindow::onUpdate(Game g) {
  this->game = g;

  if (this->enemy_field) {
    this->enemy_field->field = this->game.enemy_field;
    this->enemy_field->redraw();
  }
  if (this->own_field) {
    this->own_field->field = this->game.own_field;
    this->own_field->redraw();
  }

  if (!this->game.game_over) {
    if (this->game.youre_going) {
      this->ui->whose_turn->setText("Ваш ход");
    } else {
      this->ui->whose_turn->setText("Ход противника");
    }
  } else {
    if (this->game.youre_winner) {
      if (this->game.reason == WinningReason::FAIR)
        this->ui->whose_turn->setText("Вы победили");
      else if (this->game.reason == WinningReason::SURRENDER)
        this->ui->whose_turn->setText("Противник сдался");
    } else {
      this->ui->whose_turn->setText("Вы проиграли");
    }
  }
}

void GameWindow::onError(ErrorCode error) {
  switch (error) {
  case NO_HANDSHAKE:
  case BAD_HANDSHAKE:
    handshake_ok = false;
    this->close();
    break;
  default:
    break;
  }
}

void GameWindow::init() {
  this->handshake_ok = true;
  this->conn->sendHandshake(this->ships);
}

void GameWindow::closeEvent(QCloseEvent *event) { this->onSurrender(); }

void GameWindow::surrender() { this->close(); }
```

src \widgets \initgame.cpp

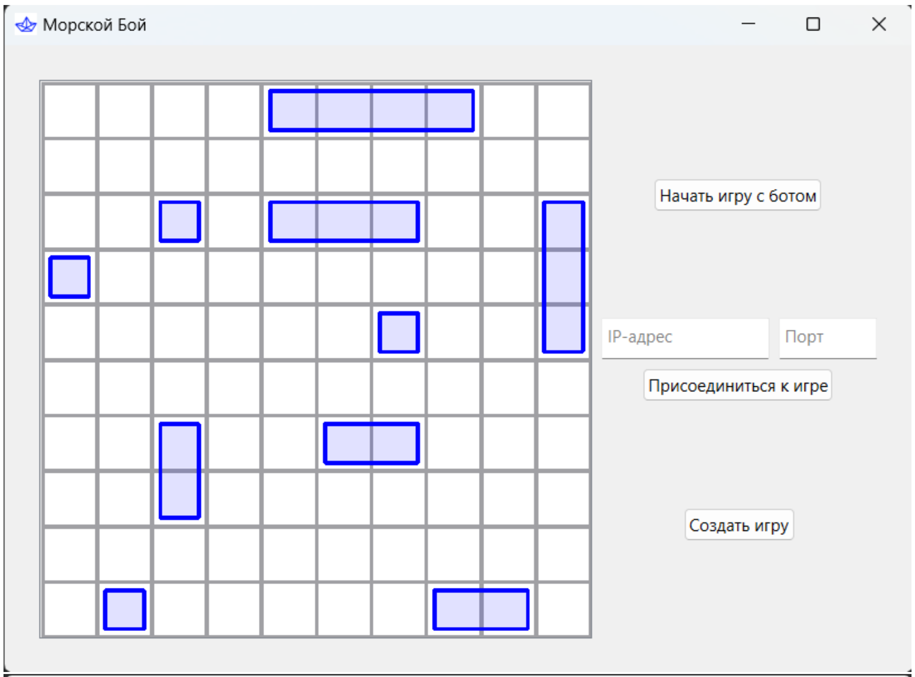
```
#include <widgets/initgame.h>

#include <game/client/botclient.h>
#include <game/client/tcpconnectionstrategy.h>
#include <game/gamebuilder.h>
```

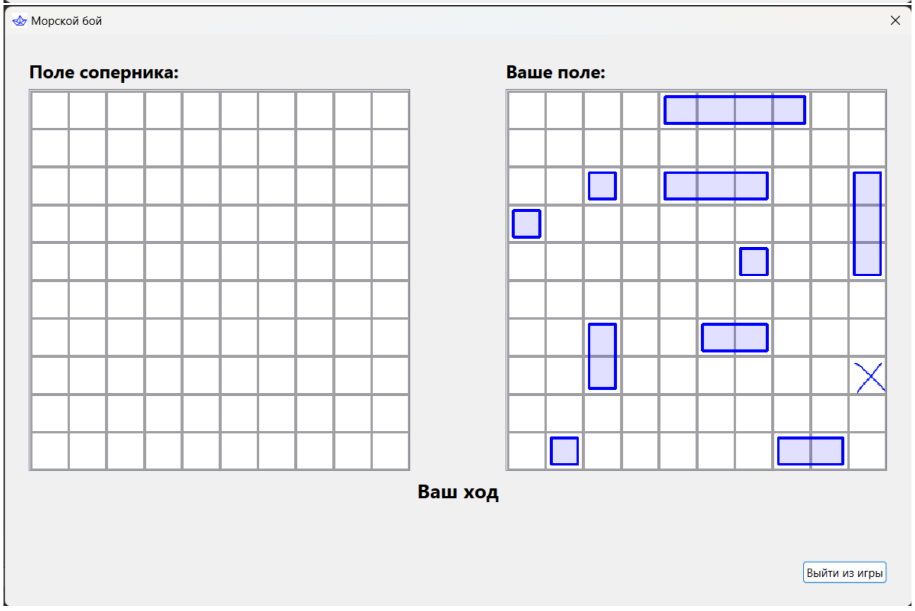
```
#include <widgets/gamewindow.h>

#include <QNetworkInterface>
#include <QTcpSocket>
#include <sstream>

#include "ui_initgame.h"

InitGame::InitGame(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::InitGame) {
  ui->setupUi(this);

  ui->field->setMouseTracking(true);
  ui->field->setScene(nullptr);

  this->connect(ui->begin_bot, &QPushButton::clicked, this,
                &InitGame::beginBot);
  this->connect(ui->begin_create, &QPushButton::clicked, this,
                &InitGame::beginCreate);
  this->connect(ui->begin_join, &QPushButton::clicked, this,
                &InitGame::beginJoin);
}

void InitGame::beginJoin() {
  if (game_going)
    return;
  if (this->server)
    return;
  auto ip = ui->join_ip->toPlainText().toStdString();
  auto port = ui->join_port->toPlainText().toInt();

  QTcpSocket *socket = new QTcpSocket(this);
  socket->connectToHost(QHostAddress(QString::fromStdString(ip)), port);

  if (socket->waitForConnected()) {
    game_window = new GameWindow(
        reinterpret_cast<FieldWidgetEdit *>(ui->field->scene())->field.ships);
    game_window->conn = new ClientConnection(
        new TCPClientConnectionStrategy(game_window, socket, this));

    this->hide();
    game_going = true;
    game_window->init();
    game_window->exec();
    this->show();

    delete game_window;

    game_going = false;
  }
```

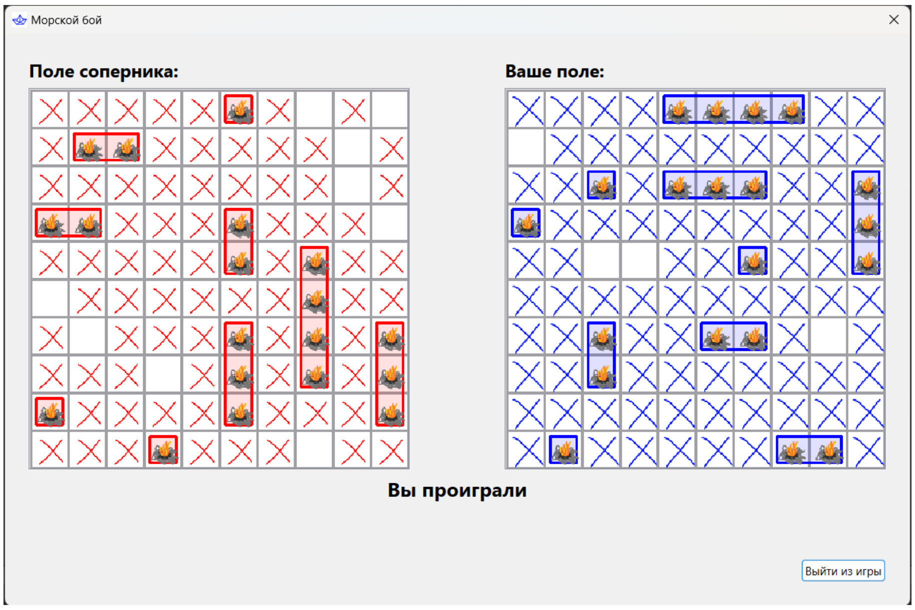
```
}

void InitGame::newConnection() {
  QTcpSocket *socket = this->server->nextPendingConnection();
  if (game_going) {
    socket->close();
    return;
  }

  game_window = new GameWindow(
      reinterpret_cast<FieldWidgetEdit *>(ui->field->scene())->field.ships);
  auto game_server = GameBuilder().playing(game_window)->vs(socket)->begin();

  if (game_window->handshake_ok) {
    this->hide();
    game_going = true;
    game_window->exec();
  }
  this->show();

  delete game_window;
  delete game_server;
  game_going = false;
}

void InitGame::downServer() {
  if (!this->server)
    return;
  this->server->close();
  delete this->server;
  this->server = nullptr;
  this->ui->begin_bot->setEnabled(true);
  this->ui->begin_join->setEnabled(true);

  this->ui->begin_create->setText("Создать игру");
}

void InitGame::setupServer() {
  if (this->server)
    return;
  std::stringstream btntext;
  btntext << "Порт для подключения: ";
  this->server = new QTcpServer(this);
  this->server->listen(QHostAddress::LocalHost);
  btntext << this->server->serverPort();
  this->ui->begin_create->setText(QString::fromStdString(btntext.str()));
  this->ui->begin_bot->setEnabled(false);
  this->ui->begin_join->setEnabled(false);

  this->connect(this->server, &QTcpServer::newConnection, this,
                &InitGame::newConnection);
}

void InitGame::beginCreate() {
  if (this->server) {
    downServer();
  } else {
    setupServer();
  }
}

InitGame::~InitGame() { delete ui; }

void InitGame::showEvent(QShowEvent *event) {
  QMainWindow::showEvent(event);

  if (ui->field->scene())
    return;
  auto scene = new FieldWidgetEdit(ui->field);
  scene->setField(Field::generateRandom());

  ui->field->setScene(scene);
}

void InitGame::beginBot() {
  if (game_going)
    return;
  if (this->server)
    return;
  game_window = new GameWindow(
      reinterpret_cast<FieldWidgetEdit *>(ui->field->scene())->field.ships);
  auto bot_view = new BotGameClient();
  auto game_server = GameBuilder().playing(game_window)->vs(bot_view)->begin();

  if (game_window->handshake_ok) {
    this->hide();
    game_going = true;
    game_window->exec();
  }
  this->show();

  delete game_window;
  delete bot_view;
  delete game_server;
  game_going = false;
}
```

Ссылка на репозиторий: https://github.com/IAmProgrammist/cbattle

4 Результат работы, на предложенных тестовых данных







5 Вывод о проделанной работе

В ходе работы обобщены и закреплены знания о паттернах, используемых в объектно-ориентированном программировании, техниках рефакторинга, SOLID, создания графического интерфейса программы при использовании инструментов C++ и QT, проектирования системы клиент-сервер. В результате работы была разработана программа, моделирующая игру Морской Бой, позволяющая играть в неё как с компьютером, так и с другим игроком при помощи протокола TCP.

6 Список источников и литературы

1. QT Documentation [Электронный ресурс]
   Режим доступа: https://doc.qt.io/

2. Паттерны проектирования на C++ [Электронный ресурс]
   Режим доступа: https://refactoringguru.cn/ru/design-patterns/cpp

3. C++ Reference [Электронный ресурс]
   Режим доступа: https://en.cppreference.com/w/