МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

КУРСОВАЯ РАБОТА

по дисциплине «Сети ЭВМ»

Тема: реализация межсетевой игры «Pixel Battle»

Студент гр. 9302 _____ Ширнин К.В.

Преподаватель _____ Горячев А.В.

Санкт-Петербург

2022

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Ширнин К.В.

Группа 9302

Тема работы: реализация межсетевой игры «Pixel Battle»

Исходные данные:

Требуется написать игру в стиле «Pixel Battle» с межсетевым взаимодействием клиентов и сервера.

Дата выдачи задания: 05.02.2022

Дата сдачи реферата: 03.06.2022

Студент _____ Ширнин К.В.

Преподаватель _____ Горячев А.В.

АННОТАЦИЯ

В данной работе представлены программа-сервер и программа-клиент, разработанные на языке C++ под фреймворком Qt, которые содержат в себе решение поставленной задачи. Клиентская часть программы взаимодействует с серверной посредством TCP соединения.

SUMMARY

This paper presents a server program and a client program developed in C ++ under the Qt framework, which contain the solution to the problem. The client part of the program interacts with the server part through a TCP connection.

СОДЕРЖАНИЕ

Оглавление

Введение

Цель работы: обобщить знания и практические навыки по межсетевому взаимодействию ЭВМ, полученные за семестр.

Основные задачи: реализовать игру в стиле «Pixel Battle».

Методы решения: для решения поставленной задачи потребовалось изучить основы TCP соединения между ЭВМ, а также получить базовые знания по построению “байт” пакетов и их оптимизации. Для единообразного отображения информации у клиентов потребовалось реализовать сервер, с которым будет происходить взаимодействие. Для реализации серверной части потребовалось изучить основы настройки маршрутизатора, для проброса портов на локальный компьютер.

1. Суть игры

1 апреля 2017 сайт Reddit запустил социальный эксперимент «Place». На протяжении трёх суток на пустом холсте размером 1000 на 1000 пикселей каждый участник сообщества мог закрасить один пиксель раз в пять минут. Изначально событие представлялось в качестве «первоапрельской шутки», но спустя неделю он вошел в историю.

Выделяется полотно с определённым количеством пикселей. Один участник может закрасить один любой пиксель раз в 1-5 минут (ограничение в реализованной игре установлено на 15 секунд). Таким образом, это заставляет объединяться людей в группы, чтобы нанести на холст свои пиксельные рисунки общими усилиями. Так же для игроков будет проблемой то, что обязательно найдутся те, кто захочет занять их место на холсте и будут перерисовывать уже готовый рисунок.

2. Ход работы

Для решения поставленной задачи была создана серверная часть приложения и клиентская. Межсетевое взаимодействие было принято сделать посредством TCP протокола, потому для точного отображения состояния поля требуется гарантированная передача данных. Каждое из приложений обладает реализованными сервисами (TCPServer и TCPClient), которые взаимодействуют между собой. Эти сервисы, в процессе работы программы, передают полученные данные посредством сигналов (инструментарий QtFramework), а также при помощи слотов (инструментарий QtFramework) могут принимать пакеты для последующей их передачи между друг другом. Для своевременного отображения состояния поля сервер передает данные клиентам с частотой 500мс. Так же добавлена заданная задержка между ходами одного и того же пользователя в 900мс. Время задержки хода клиента, а также возможность совершения хода высчитывается на сервере, в целях безопасности. Реализована защита от спама игрового поля с одного и

того же IP адреса, путем открытия нескольких клиентов на одном и том же компьютере – хранение данных осуществлено путем сохранения IP адреса, а не сокета отправителя. Но, тем не менее, играть можно с двух разных клиентов, запущенных на одном IP, но таймер хода у обоих будет один и тот же.

Для передачи данных формируются оптимизированные пакеты. Информация для передачи была урезана до возможного минимума, в целях оптимизации скорости передачи данных. Было реализовано два типа пакетов: “TIME” – для передачи клиенту времени, которое ему следует ожидать до следующего хода и “DATA” – для передачи состояния поля. Строение пакета для передачи данных о состоянии поля от сервера клиентам представлено на рисунке ниже (Рисунок 1)

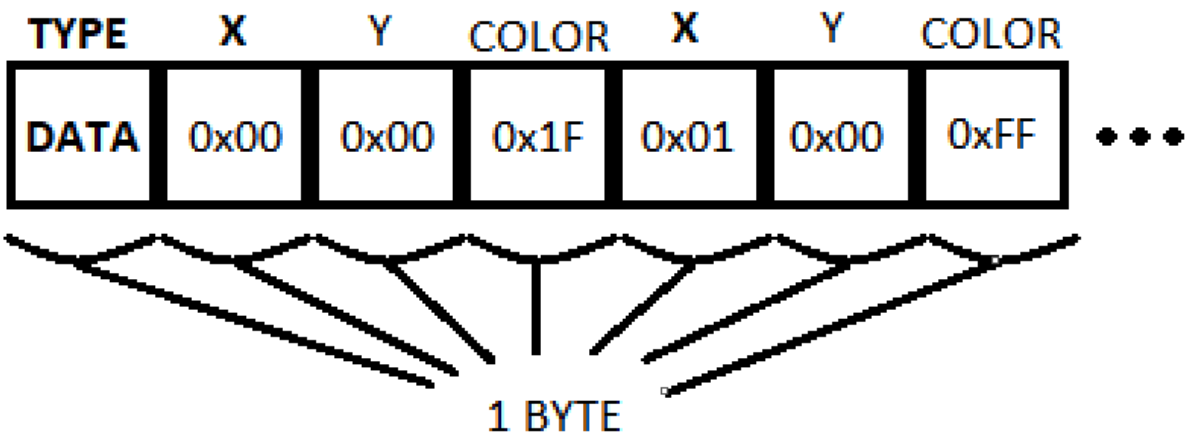

Рисунок 1

Строение пакета для передачи данных об оставшемся времени ожидания от сервера клиенту представлено на рисунке ниже (Рисунок 2)

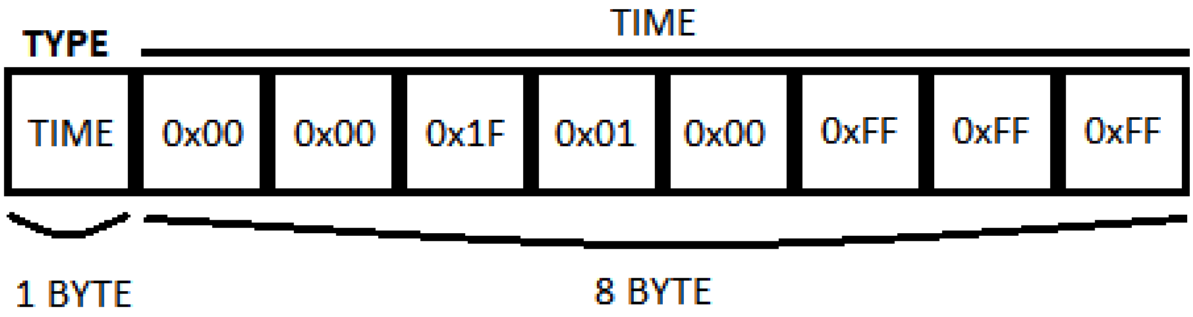

Рисунок 2

Строение пакета для передачи данных о совершении хода пользователем (передача от клиента серверу) представлено на рисунке ниже (Рисунок 3)

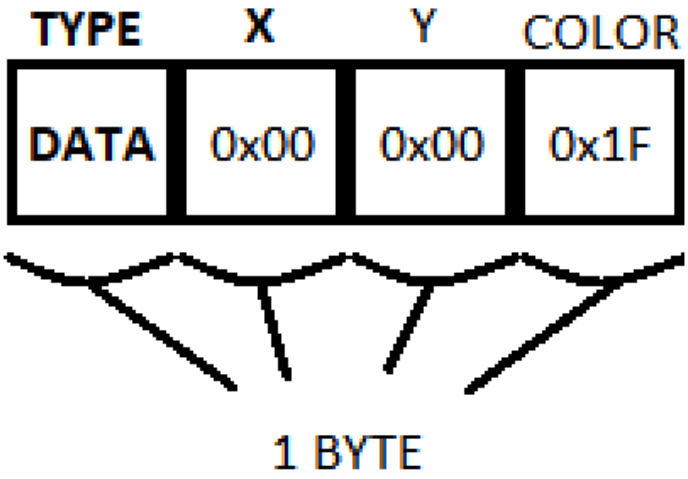

Рисунок 3

Пакет для передачи данных, содержащий данные о клетке, изначально содержал в себе 5 байт (X, Y, R_color, G_color, B_color). В целях оптимизации пакета, было решено изменить структуру хранения цвета у ячейки в размер 1 байт. В программе заданы 16 базовых цветов, которые легко можно представить в виде 1 байта (1 байт даже много).

3. Разбиение основной задачи на подзадачи

В обеих программах имеется общий модуль Cell – представляет из себя класс, унаследованный от QLabel. Предназначен для отображения одного игрового пикселя

CLIENT PROGRAM

Имя модуля	Описание
BrashBattleClient	Класс клиента игры. Содержит в себе реализацию GUI – игровое поле, палитру цветов для выбора кисти, индикатор состояния подключения, таймер отсчета
TCPClient	Класс TCP клиента. Взаимодействует с BrashBattleClient, отсылая получаемые данные через сигналы и получая данные через слоты для последующей их передачи в TCPServer

SERVER PROGRAM

Имя модуля	Описание
BrashBattleServer	Класс сервера игры. Содержит в себе реализацию GUI – игровое поле для отображения его текущего состояния
TCPServer	Класс TCP сервера. Взаимодействует с BrashBattleServer, отсылая получаемые данные через сигналы и получая данные через слоты для последующей их передачи в TCPClient

4. Диаграмма классов

Диаграмма классов для клиентской части приложения представлена на рисунке ниже (Рисунок 4)

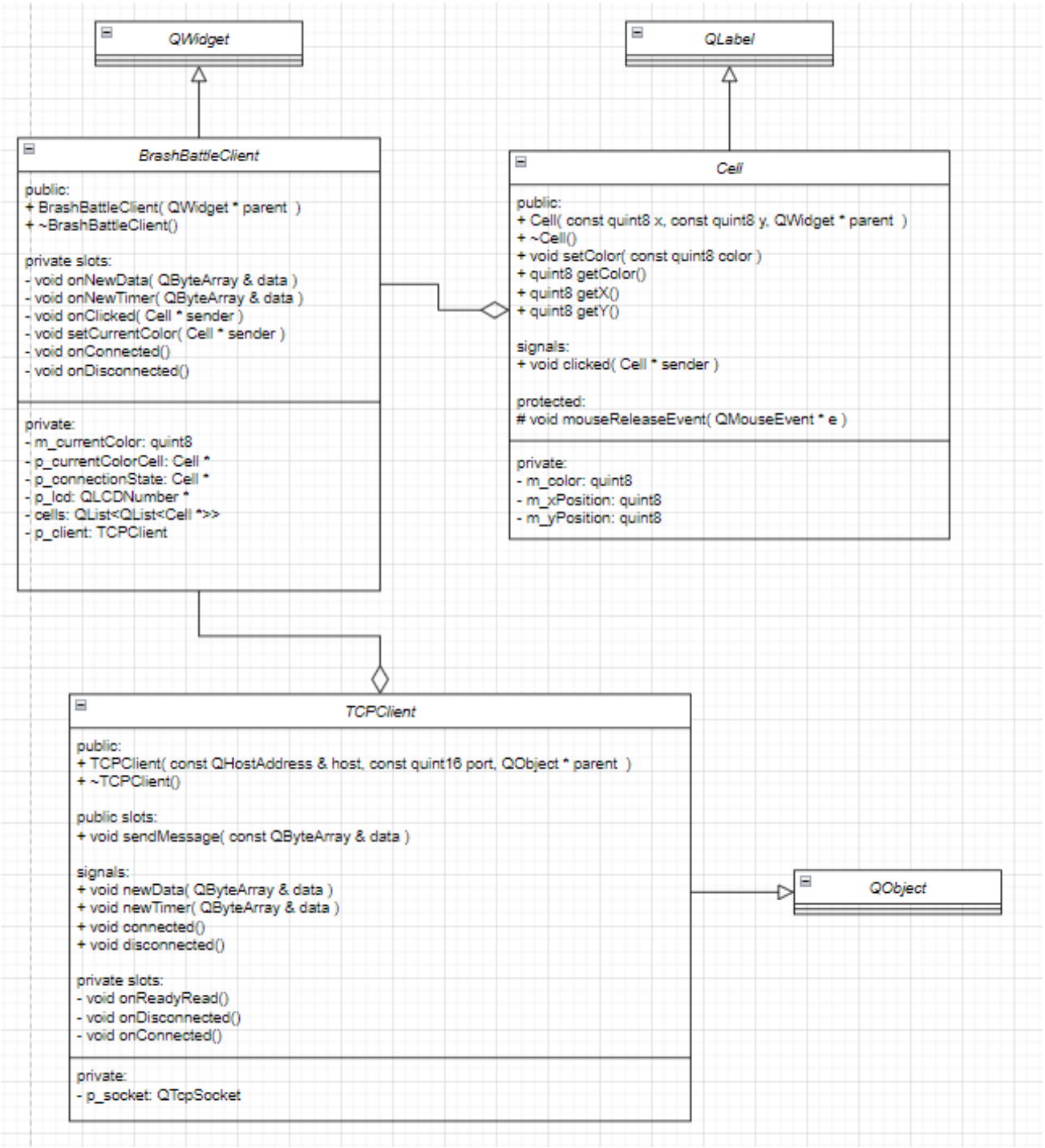

Рисунок 4

Диаграмма классов для серверной части приложения представлена на рисунке ниже (Рисунок 5)

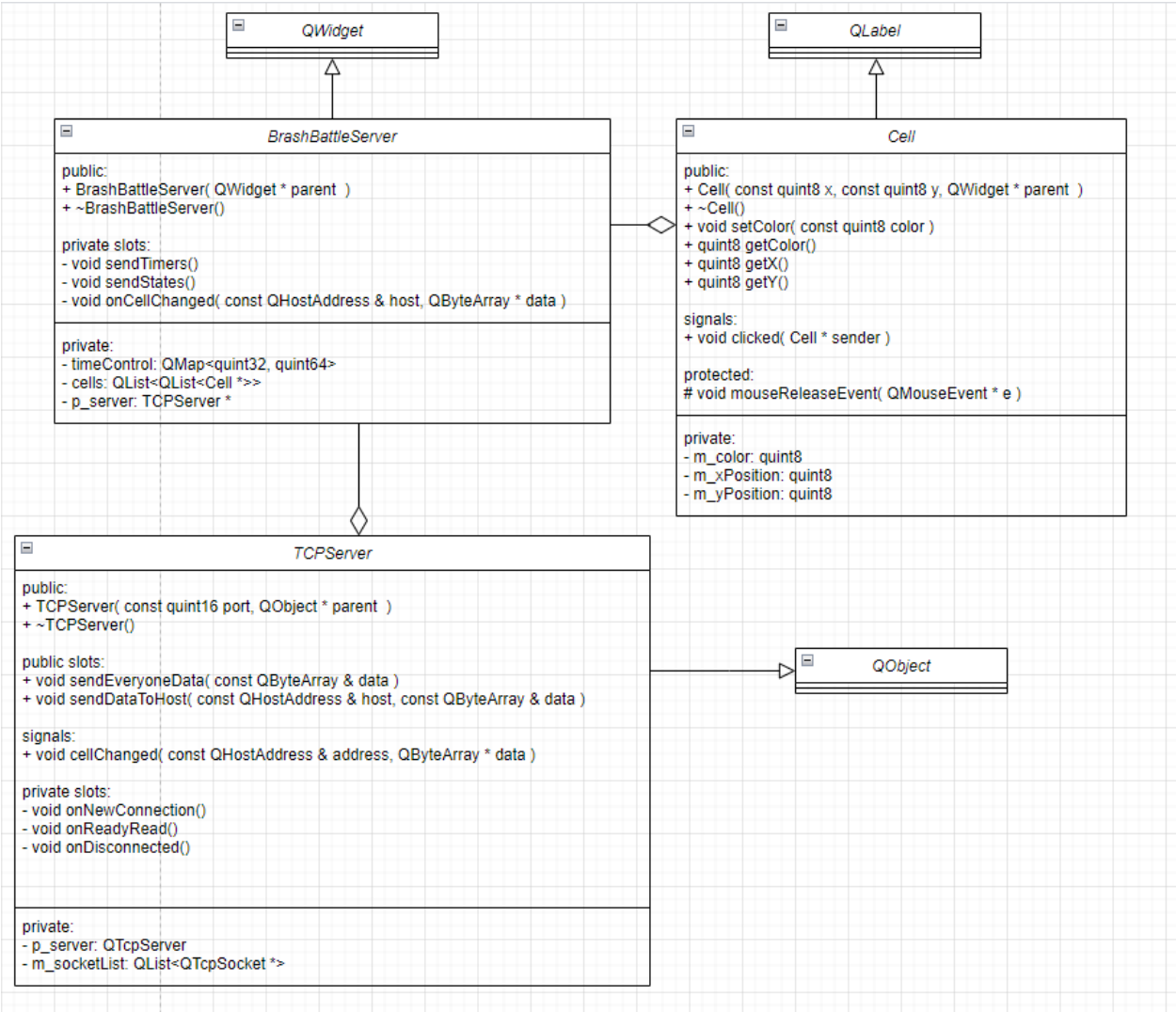

Рисунок 5

5. Код программы

Cell.h

```
#pragma once

#define TIME_DELAY 15    //!< Задержка между возможной сменой цвета одним пользователем

#include <QWidget>
#include <QLabel>
#include <QColor>
#include <QObject>
```

```
#include <QMouseEvent>
#include <QByteArray>
#include <QDataStream>


//!
//! \brief The COLOR_TYPE enum Перечисление цветов пикселя
//!
enum COLOR_TYPE
{
    BLACK,
    GRAY,
    SILVER,
    WHITE,
    FUCHSIA,
    PURPLE,
    RED,
    MAROON,
    YELLOW,
    OLIVE,
    LIME,
    GREEN,
    AQUA,
    TEAL,
    BLUE,
    NAVY
};

//!
//! \brief The PACKET_TYPE enum Тип отправляемого TCP пакета
//!
enum PACKET_TYPE
{
    DATA,    //!< Данные о пикселях
    TIME,    //!< Время
    CMD      //!< Резерв
};

//!
//! \brief The Cell class Класс одного пикселя
//!
class Cell : public QLabel
{
    Q_OBJECT
public:
    //!
    //! \brief Cell Конструктор
    //! \param x Координата по X
    //! \param y Координата по Y
    //! \param parent Родительский виджет
    //!
    Cell( const quint8 x, const quint8 y, QWidget * parent = nullptr );
    //!
    //! \brief ~Cell Деструктор
    //!
    virtual ~Cell();
    //!
    //! \brief setColor Устанавливает цвет пикселя
    //! \param color Цвет
    //!
    void setColor( const quint8 color );
    //!
    //! \brief getColor Возвращает цвет пикселя
```

```
    //! \return Цвет пикселя
    //!
    quint8 getColor();
    //!
    //! \brief getX Возвращает координату X
    //! \return Координата X
    //!
    quint8 getX();
    //!
    //! \brief getY Возвращает координату Y
    //! \return Координата Y
    //!
    quint8 getY();

signals:
    //!
    //! \brief clicked Сигнал, испускающийся при клике на пиксель
    //! \param sender Собственный объект
    //!
    void clicked( Cell * sender );

protected:
    //!
    //! \brief mouseReleaseEvent Событие клика. Реализован для сигнала
    //! \param e Событие
    //!
    void mouseReleaseEvent( QMouseEvent * e );

private:
    quint8 m_color;         //!< Цвет
    quint8 m_xPosition;     //!< Координата по X
    quint8 m_yPosition;     //!< Координата по Y
};
```

Cell.cpp

```
#include "cell.h"

#include <QDebug>

Cell::Cell( const quint8 x, const quint8 y, QWidget * parent ) : QLabel(
parent )
{
    m_xPosition = x;
    m_yPosition = y;
    m_color = WHITE;
    setFixedSize(24, 24);
    setStyleSheet( "QLabel{ border: 1px solid black; }" );
}

Cell::~Cell()
{
}

void Cell::setColor( const quint8 color )
{
    m_color = color;
    switch ( m_color )
    {
    case BLACK:
```

```
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(0,
0, 0); }" );
        break;
    case GRAY:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(128,
128, 128); }" );
        break;
    case SILVER:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(192,
192, 192); }" );
        break;
    case WHITE:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(255,
255, 255); }" );
        break;
    case FUCHSIA:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(255,
0, 255); }" );
        break;
    case PURPLE:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(128,
0, 128); }" );
        break;
    case RED:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(255,
0, 0); }" );
        break;
    case MAROON:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(128,
0, 0); }" );
        break;
    case YELLOW:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(255,
255, 0); }" );
        break;
    case OLIVE:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(128,
128, 0); }" );
        break;
    case LIME:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(0,
255, 0); }" );
        break;
    case GREEN:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(0,
128, 0); }" );
        break;
    case AQUA:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(0,
255 , 255); }" );
        break;
    case TEAL:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(0,
128, 128); }" );
        break;
    case BLUE:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(0,
0, 255); }" );
        break;
    case NAVY:
        setStyleSheet( "QLabel{ border: 1px solid black; background: rgb(0,
0, 128); }" );
        break;
```

```
    default:
        qDebug() << "$$$ Incorrect color $$$";
        break;
    }
}

quint8 Cell::getColor()
{
    return m_color;
}

quint8 Cell::getX()
{
    return m_xPosition;
}

quint8 Cell::getY()
{
    return m_yPosition;
}

void Cell::mouseReleaseEvent( QMouseEvent * e )
{
    if ( e->button() == Qt::LeftButton )
        emit clicked( this );
}
```

brashbattleclient.h

```
#pragma once

#include "../cell.h"
#include "tcpclient.h"
#include <QWidget>
#include <QByteArray>
#include <QLCDNumber>

#define HOST "178.162.92.161"    //!< IP сервера
#define PORT 2673                //!< Порт сервера

//!
//! \brief The BrashBattleClient class Класс клиента игры
//!
class BrashBattleClient : public QWidget
{
    Q_OBJECT

public:
    //!
    //! \brief BrashBattleClient Конструктор
    //! \param parent Родительский виджет
    //!
    BrashBattleClient( QWidget * parent = nullptr );
    //!
    //! \brief ~BrashBattleClient Деструктор
    //!
    ~BrashBattleClient();

private slots:
    //!
    //! \brief onNewData Реакция на получение новых данных об игровом поле
    //! \param data Данные
```

```
    //!
    void onNewData( QByteArray & data );
    //!
    //! \brief onNewTimer Реакция на получение данных об окончании задержки
между ходами
    //! \param data Данные
    //!
    void onNewTimer( QByteArray & data );
    //!
    //! \brief onClicked Реакция на нажатие на один из пикселей
    //! \param sender Указатель на пиксель
    //!
    void onClicked( Cell * sender );
    //!
    //! \brief setCurrentColor Установить текущий цвет для рисования
    //! \param sender Указатель на пиксель с выбранным цветом
    //!
    void setCurrentColor( Cell * sender );
    //!
    //! \brief onConnected Реакция на подключение к серверу
    //!
    void onConnected();
    //!
    //! \brief onDisconnected Реакция на отключение от сервера
    //!
    void onDisconnected();

private:
    quint8                      m_currentColor;     //!< Выбранный цвет;
    Cell                    *   p_currentColorCell; //!< Указатель на пискель
с выбранным цветом
    Cell                    *   p_connectionState;  //!< Указатель на
индикатор состояния подключения к серверу
    QLCDNumber              *   p_lcd;              //!< Указатель на счетчик
времени
    QList<QList<Cell *> >       cells;              //!< Массив пикселей
    TCPManager::TCPClient   *   p_client;           //!< Указатель на TCP
клиент
};
```

brashbattleclient.cpp

```
#include "brashbattleclient.h"

#include <QByteArray>
#include <QDataStream>
#include <QHostAddress>
#include <QGridLayout>
#include <QDateTime>
#include <QHBoxLayout>
#include <QSpacerItem>

#define SIZE 16

BrashBattleClient::BrashBattleClient( QWidget * parent )
    : QWidget( parent )
{
    p_client = new TCPManager::TCPClient( QHostAddress( HOST ), PORT );

    p_lcd = new QLCDNumber( 2 );
    p_lcd->setSegmentStyle( QLCDNumber::Flat );
```

```
    p_lcd->setMinimumHeight( 70 );

    connect( p_client, SIGNAL( newData( QByteArray & ) ), this, SLOT(
onNewData( QByteArray & ) ) );
    connect( p_client, SIGNAL( newTimer( QByteArray & ) ), this, SLOT(
onNewTimer( QByteArray & ) ) );
    connect( p_client, SIGNAL( connected() ), this, SLOT( onConnected() ) );
    connect( p_client, SIGNAL( disconnected() ), this, SLOT( onDisconnected()
) );

    QGridLayout * grd = new QGridLayout ();
    grd->setSpacing( 0 );

    for ( int i = 0; i < SIZE; ++i )
    {
        cells.append( QList<Cell *>() );
        for ( int j = 0; j < SIZE; ++j )
        {
            Cell * cell = new Cell( i, j, this );
            connect( cell, SIGNAL( clicked( Cell * ) ), this, SLOT(
onClicked( Cell * ) ) );
            cells[i].append( cell );
            grd->addWidget( cells[i][j], i, j, 1, 1 );
        }
    }

    grd->addWidget( p_lcd, SIZE, 0, 3, SIZE );

    QGridLayout * colorLayout   = new QGridLayout();
    QHBoxLayout * mainLayout    = new QHBoxLayout( this );

    for ( int i = 0; i < 16; ++i )
    {
        Cell * cell = new Cell( 0, 0 );
        cell->setColor( i );
        colorLayout->addWidget( cell, (i % 8) + 1, i / 8, 1, 1 );
        connect( cell, SIGNAL( clicked( Cell * ) ), this, SLOT(
setCurrentColor( Cell * ) ) );
    }

    p_currentColorCell = new Cell( 0, 0 );
    p_currentColorCell->setFixedSize( 48, 48 );
    colorLayout->addWidget( p_currentColorCell, 9, 0, 2, 2 );
    p_currentColorCell->setColor( WHITE );

    p_connectionState = new Cell( 0, 0 );
    p_connectionState->setFixedSize( 16, 16 );
    colorLayout->addWidget( p_connectionState, 10, 0, 1, 1 );
    p_connectionState->setColor( RED );

    colorLayout->setSpacing( 1 );
    colorLayout->setContentsMargins( 0, 0, 0, 0 );
    //colorLayout->addItem( new QSpacerItem( 30, 100 ), 10, 0 );

    mainLayout->addStretch( 1 );
    mainLayout->addLayout( colorLayout );
    mainLayout->addLayout( grd );

    m_currentColor = BLACK;
}

BrashBattleClient::~BrashBattleClient()
{
```

```
    delete p_client;
}

void BrashBattleClient::onNewData( QByteArray & data )
{
    QDataStream in( &data, QIODevice::ReadOnly );

    char type;
    char xPosition;
    char yPosition;
    char color;

    in >> type;

    for ( int i = 0; i < SIZE; ++i )
        for ( int j = 0; j < SIZE; ++j )
        {
            in >> xPosition >> yPosition >> color;
            cells[xPosition][yPosition]->setColor( color );
        }
}

void BrashBattleClient::onNewTimer( QByteArray & data )
{
    QDataStream in( &data, QIODevice::ReadOnly );

    char type;
    quint64 seconds;

    in >> type;
    in >> seconds;

    QString string;

    qDebug() << QDateTime::currentSecsSinceEpoch() - seconds << seconds;

    string = QString::number( QDateTime::currentSecsSinceEpoch() - seconds >=
TIME_DELAY ? 0
                                                : ( TIME_DELAY -
(QDateTime::currentSecsSinceEpoch() - seconds) ) );

    p_lcd->display( string );
}

void BrashBattleClient::onClicked( Cell * sender )
{
    QByteArray buffer;
    QDataStream out( &buffer, QIODevice::WriteOnly );

    //sender->setColor( 1 );

    out << (char)DATA << sender->getX() << sender->getY() <<
(char)m_currentColor; //sender->getColor();
    p_client->sendMessage( buffer );
}

void BrashBattleClient::setCurrentColor( Cell * sender )
{
    m_currentColor = sender->getColor();
    p_currentColorCell->setColor( sender->getColor() );
}

void BrashBattleClient::onConnected()
```

```
{
    p_connectionState->setColor( LIME );
}

void BrashBattleClient::onDisconnected()
{
    p_connectionState->setColor( RED );
}
```

tcpclient.h

```
#pragma once

#include <QObject>
#include <QByteArray>

class QTcpSocket;
class QHostAddress;

namespace TCPManager
{
    //!
    //! \brief The TCPClient class Класс TCP клиента
    //!
    class TCPClient : public QObject
    {
        Q_OBJECT
    public:
        //!
        //! \brief TCPClient Конструктор
        //! \param host Хост
        //! \param port Порт
        //! \param parent Родительский объект
        //!
        TCPClient( const QHostAddress & host, const quint16 port, QObject *
parent = nullptr );
        ~TCPClient();

    public slots:
        //!
        //! \brief sendMessage Отправить данные
        //! \param data Данные
        //!
        void sendMessage( const QByteArray & data );

    signals:
        //!
        //! \brief newData Сигнал, испускающийся при получении новых данных
об игровом поле
        //! \param data Данные
        //!
        void newData( QByteArray & data );
        //!
        //! \brief newTimer Сигнал, испускающийся при получении новых данных
о времени задержки
        //! \param data
        //!
        void newTimer( QByteArray & data );
        //!
        //! \brief connected Сигнал, испускаемый при подключении к серверу
        //!
        void connected();
```

```
        //!
        //! \brief disconnected Сигнал, испускаемый при отключении от сервера
        //!
        void disconnected();

    private slots:
        //!
        //! \brief onReadyRead Реакция на получение TCP пакета
        //!
        void onReadyRead();
        //!
        //! \brief onDisconnected Реакция на отключение от сервера
        //!
        void onDisconnected();
        //!
        //! \brief onConnected Реакция на подключение к серверу
        //!
        void onConnected();

    private:
        //!
        //! \brief p_socket Указатель на TCP сокет
        //!
        QTcpSocket  * p_socket;
    };
}
```

tcpclient.cpp

```
#include "../cell.h"
#include "tcpclient.h"

#include <QByteArray>
#include <QDebug>
#include <QHostAddress>
#include <QTcpSocket>

namespace TCPManager
{
    TCPClient::TCPClient( const QHostAddress & host, const quint16 port,
QObject * parent )
        : QObject( parent )
    {
        p_socket = new QTcpSocket( this );
        qDebug() << "--- Connect to Host ---";

        p_socket->connectToHost( host, port );

        connect( p_socket, SIGNAL( connected() ), this, SLOT( onConnected() )
);

        connect( p_socket, SIGNAL( readyRead() ), this, SLOT( onReadyRead() )
);

        connect( p_socket, SIGNAL( disconnected() ), this, SLOT(
onDisconnected() ) );
    }

    TCPClient::~TCPClient()
    {
        p_socket->close();
        p_socket->deleteLater();
```

```
    }

    void TCPClient::sendMessage( const QByteArray & data )
    {

        qDebug() << "--- Sending Message ---";
        p_socket->write( data );
    }

    void TCPClient::onReadyRead()
    {
        qDebug() << "--- Read Message ---";

        QByteArray outArray;
        if( p_socket->bytesAvailable() )
        {
            outArray = p_socket->readAll();
        }


        QDataStream in( &outArray, QIODevice::ReadOnly );
        char type;
        in >> type;

        if ( !outArray.isEmpty() )
        {
            if ( type == (char)TIME )
                emit newTimer( outArray );
            else if ( type == (char)DATA )
                emit newData( outArray );
            else
                qDebug() << "wtf";
        }
    }

    void TCPClient::onDisconnected()
    {
        qDebug() << "--- Connection Ended ---";
        emit disconnected();
    }

    void TCPClient::onConnected()
    {
        qDebug() << "--- Successful connection ---";
        emit connected();
    }
}
```

brashbattleserver.h

```
#pragma once

#include "../cell.h"

#include "tcpserver.h"
#include <QWidget>
#include <QHostAddress>
#include <QDateTime>

#define PORT 2673   //!< Порт подключения к серверу
```

```
//!
//! \brief The BrashBattleServer class Класс сервера игры
//!
class BrashBattleServer : public QWidget
{
    Q_OBJECT

public:
    //!
    //! \brief BrashBattleServer Конструктор
    //! \param parent Родительский виджет
    //!
    BrashBattleServer(QWidget *parent = nullptr);
    ~BrashBattleServer();

private slots:
    //!
    //! \brief sendTimers Реакция на отправку времени задержки
    //!
    void sendTimers();
    //!
    //! \brief sendStates Реакция на отправку состояния поля
    //!
    void sendStates();
    //!
    //! \brief onCellChanged Реакция на изменение цвета поля
    //! \param host Хост
    //! \param data Данные
    //!
    void onCellChanged( const QHostAddress & host, QByteArray * data );

private:
    QMap<quint32, quint64>          timeControl;    //!< Мапа для контроля
времени хода на каждый IP
    QList<QList<Cell *> >           cells;          //!< Игровое поле
    TCPManager::TCPServer       *   p_server;       //!< Указатель на TCP
сервер
};
```

brashbattleserver.cpp

```
#include "brashbattleserver.h"

#include <QTimer>
#include <QByteArray>
#include <QDataStream>
#include <QGridLayout>
#include <QDebug>

#define SIZE 16

BrashBattleServer::BrashBattleServer( QWidget * parent )
    : QWidget( parent )
{
    p_server = new TCPManager::TCPServer( PORT );

    connect( p_server, SIGNAL( cellChanged( const QHostAddress &, QByteArray
* ) ),
             this, SLOT( onCellChanged( const QHostAddress &, QByteArray * )
) );

    QTimer * timer1 = new QTimer();
```

```
    QTimer * timer2 = new QTimer();

    connect( timer1, SIGNAL( timeout() ), this, SLOT( sendStates() ) );
    connect( timer2, SIGNAL( timeout() ), this, SLOT( sendTimers() ) );

    timer1->start( 500 );
    timer2->start( 900 );

    QGridLayout * grd = new QGridLayout( this );
    grd->setSpacing( 0 );

    for ( int i = 0; i < SIZE; ++i )
    {
        cells.append( QList<Cell *>() );
        for ( int j = 0; j < SIZE; ++j )
        {
            cells[i].append( new Cell( i, j, this ) );
            grd->addWidget( cells[i][j], i, j, 1, 1 );
        }
    }
}

BrashBattleServer::~BrashBattleServer()
{
}

void BrashBattleServer::sendTimers()
{
    //qDebug() << "--- SENDING TIMERS ---";

    foreach ( quint32 host, timeControl.keys() ) {
        qDebug() << "--- SENDING TIMER TO ---" << QHostAddress( host );
        QByteArray buffer;
        QDataStream out( &buffer, QIODevice::WriteOnly );
        out << (char)TIME;
        out << timeControl[host];

        p_server->sendDataToHost( QHostAddress( host ), buffer );
    }
}

void BrashBattleServer::sendStates()
{
    //qDebug() << "--- SENDING STATES ---";
    QByteArray buffer;
    QDataStream out( &buffer, QIODevice::WriteOnly );

    out << (char)DATA;

    for ( int i = 0; i < SIZE; ++i )
        for ( int j = 0; j < SIZE; ++j )
            out << cells[i][j]->getX() << cells[i][j]->getY() << cells[i][j]-
>getColor();


    p_server->sendEveryoneData( buffer );
}

void BrashBattleServer::onCellChanged( const QHostAddress & host, QByteArray
* data )
{
    QDataStream in( data, QIODevice::ReadOnly );
```

```
    char type;
    char xPosition;
    char yPosition;
    char color;

    in >> type >> xPosition >> yPosition >> color;

    if ( cells[xPosition][yPosition]->getColor() == color )
        return;

    if ( timeControl.find( host.toIPv4Address() ) == timeControl.end() )
    {
        timeControl.insert( host.toIPv4Address(),
QDateTime::currentSecsSinceEpoch() );
        cells[xPosition][yPosition]->setColor( color );
    }
    else
    {
        if ( QDateTime::currentSecsSinceEpoch() -
timeControl[host.toIPv4Address()] > TIME_DELAY )
        {
                cells[xPosition][yPosition]->setColor( color );
                timeControl[host.toIPv4Address()] =
QDateTime::currentSecsSinceEpoch();
        }
        else
            return;
    }
}
```

tcpserver.h

```
#pragma once

#include <QObject>
#include <QByteArray>
#include <QHostAddress>

class QTcpServer;
class QTcpSocket;

namespace TCPManager
{
    //!
    //! \brief The TCPServer class Класс TCP сервера
    //!
    class TCPServer : public QObject
    {
        Q_OBJECT
    public:
        //!
        //! \brief TCPServer Конструктор
        //! \param port Порт подключения
        //! \param parent Родительский объект
        //!
        TCPServer( const quint16 port, QObject * parent = nullptr );
        ~TCPServer();

    public slots:
        //!
        //! \brief sendEveryoneData Отправить данные всем подключенным
сокетам
        //! \param data Данные
```

```
        //!
        void sendEveryoneData( const QByteArray & data );
        //!
        //! \brief sendDataToHost Отправить данные только по одному IP
        //! \param host IP
        //! \param data Данные
        //!
        void sendDataToHost( const QHostAddress & host, const QByteArray &
data );

    signals:
        //!
        //! \brief cellChanged Сигнал об изменении состояния пикселя
        //! \param address Хост
        //! \param data Данные
        //!
        void cellChanged( const QHostAddress & address, QByteArray * data );

    private slots:
        //!
        //! \brief onNewConnection Реакция на подключение нового пользователя
        //!
        void onNewConnection();
        //!
        //! \brief onReadyRead Реакция на получение данных
        //!
        void onReadyRead();
        //!
        //! \brief onDisconnected Реакция на отключение
        //!
        void onDisconnected();

    private:
        QTcpServer              * p_server;       //!< Указатель на TCP сервер
        QList<QTcpSocket *>     m_socketList;     //!< Список клиентских
сокетов
    };
}
```

tcpserver.cpp

```
#include "../cell.h"
#include "tcpserver.h"

#include <QByteArray>
#include <QDebug>
#include <QHostAddress>
#include <QTcpSocket>
#include <QTcpServer>

namespace TCPManager
{
    TCPServer::TCPServer( const quint16 port, QObject * parent )
        : QObject( parent )
    {
        p_server = new QTcpServer;

        if ( p_server->listen( QHostAddress::Any, port ) )
            qDebug() << "--- Linsting to Port ---";
        else
            qDebug() << "*** FAIL LISTING ***";
```

```
        connect( p_server, SIGNAL( newConnection() ), this, SLOT(
onNewConnection() ) );
    }

    TCPServer::~TCPServer()
    {
        p_server->close();
        p_server->deleteLater();

        foreach ( QTcpSocket * socket, m_socketList )
        {
            socket->close();
            socket->deleteLater();
        }

        qDebug() << "--- Server closed the connection ---";
    }

    void TCPServer::sendEveryoneData( const QByteArray & data )
    {
        foreach( QTcpSocket * socket, m_socketList )
        {
            socket->write( data );
        }
    }

    void TCPServer::sendDataToHost( const QHostAddress & host, const
QByteArray & data )
    {
        foreach( QTcpSocket * socket, m_socketList )
        {
            if ( socket->peerAddress().toIPv4Address() ==
host.toIPv4Address() )
            {
                socket->write( data );
            }
        }
    }

    void TCPServer::onNewConnection()
    {

        QTcpSocket * newSocket;
        newSocket = p_server->nextPendingConnection();
        m_socketList.append( newSocket );

        connect( newSocket, SIGNAL( readyRead() ), this, SLOT( onReadyRead()
) );
        connect( newSocket, SIGNAL( disconnected() ), this, SLOT(
onDisconnected() ) );

        qDebug() << "--- Accept Connection ---" << newSocket->peerAddress()
<< newSocket->peerPort();
    }

    void TCPServer::onReadyRead()
    {
        qDebug() << "--- Read Message ---";

        QTcpSocket * senderSocket = ( QTcpSocket * )sender();

        QByteArray outArray;
```

```
        if( senderSocket->bytesAvailable() )
        {
            outArray = senderSocket->readAll();
        }

        QDataStream in( &outArray, QIODevice::ReadOnly );
        char type;
        in >> type;

        if ( !outArray.isEmpty() )
        {
            if ( type == (char)DATA )
                emit cellChanged( senderSocket->peerAddress(), &outArray );
        }

        qDebug() << outArray.data() << senderSocket->peerAddress() <<
senderSocket->peerPort();
    }

    void TCPServer::onDisconnected()
    {
        qDebug() << "--- Disconnection ---";
        QTcpSocket * socket = ( QTcpSocket * )sender();
        if ( !m_socketList.removeOne( socket ) )
            qDebug() << "--- Delete socket ERROR ---";
        socket->deleteLater();
    }
}
```

6. Описание интерфейса пользователя

Расположение элементов пользовательского взаимодействия представлено на рисунке ниже (Рисунок 6)

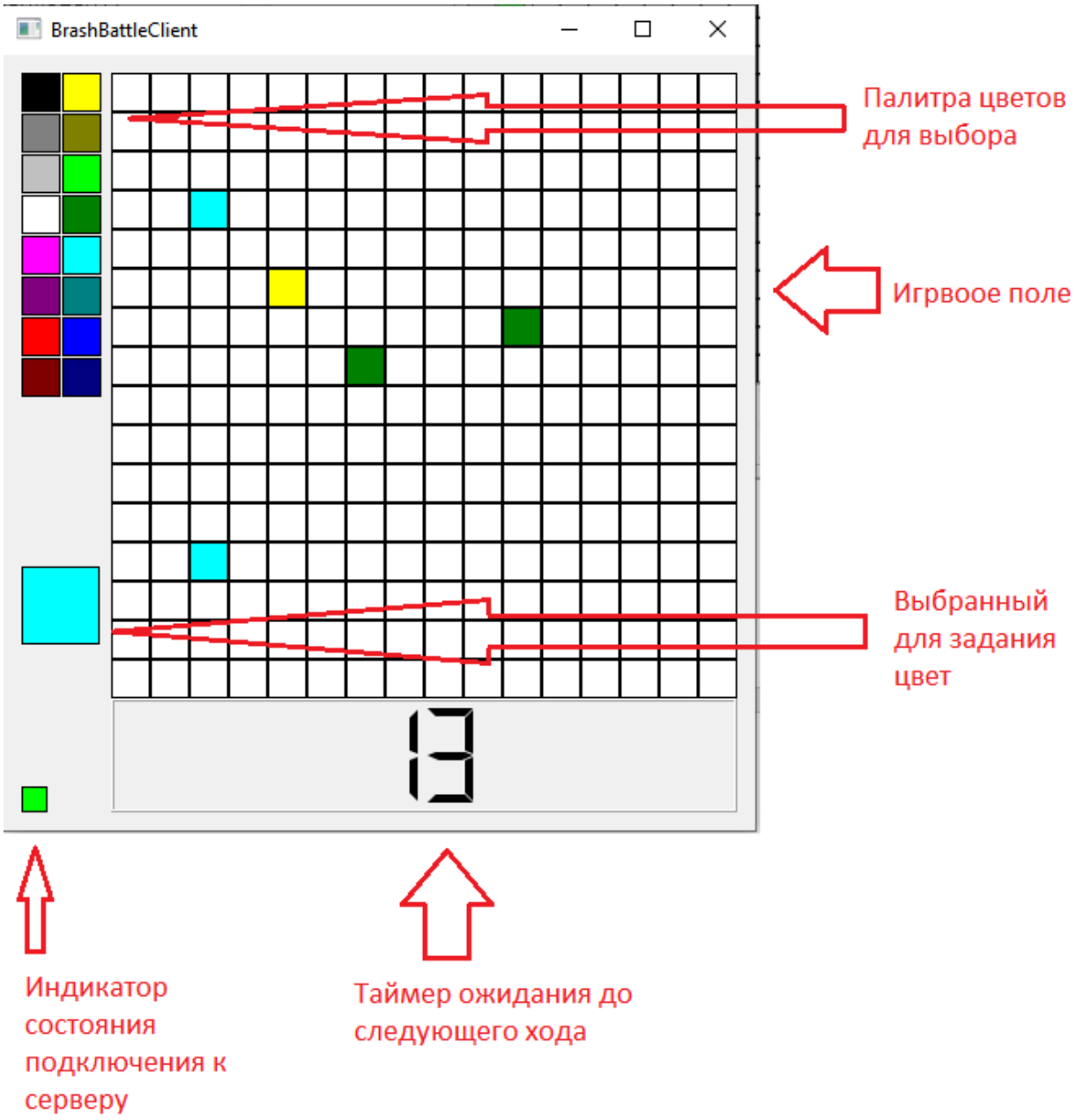

Рисунок 6

7. Пример работы программы

Пример возможного состояния игры представлено на рисунке ниже (Рисунок 7)

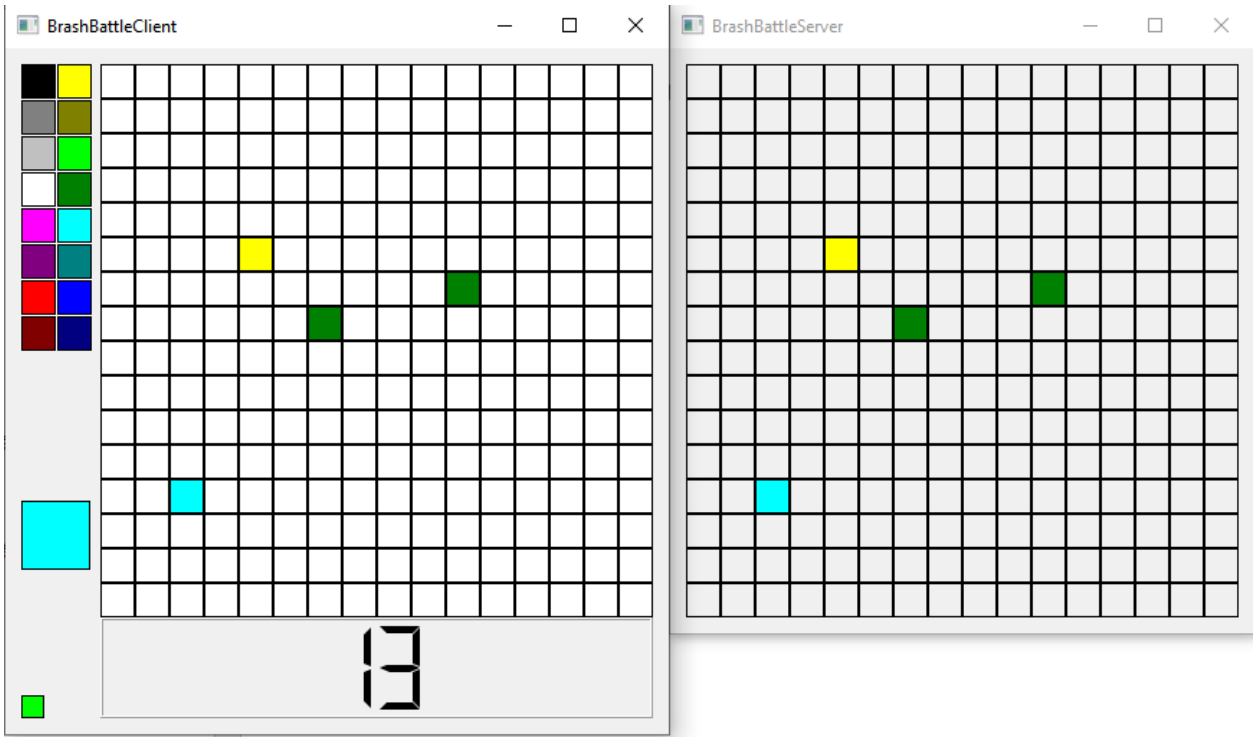

Рисунок 7

ЗАКЛЮЧЕНИЕ

Для написания данной курсовой работы потребовались знания, полученные за семестр. Клиентская часть программы взаимно успешно взаимодействует с сервером. В результате удалось решить поставленную задачу и реализовать игру в стиле «Pixel Battle» с глобальным межсетевым взаимодействием. В целях улучшения программы можно изменить способ отображения игровых пикселей, потому сейчас это реализовано через тяжелый объект QLabel, большое количество которых приведет к подтормаживанию системы. Следует работать не с QLabel, а напрямую с растровым изображением, обработку которого следует положить на сервер. Это в разы оптимизирует программный продукт.