УДК 004

Реализация игры «Морской бой» с помощью библиотеки OpenGL

Вихляев Дмитрий Романович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
Данная статья содержит описание игры «морской бой». Для реализации игры используется графическая библиотека OpenGL, кроссплатформенная библиотека GLFW и язык программирования C++. В статье приводятся описание основных функций и процедур OpenGL, рисование простейшей графики для игры, способ отображения чисел чисто графическими средствами, рекурсивное построение и обнаружение кораблей, обработка нажатий мыши. Результат исследования - игра с подробным описанием её реализации.
Ключевые слова: C ++, OpenGL, GLFW, Visual Studio, Морской бой, Игра.

Implementation of the game "Sea battle" using OpenGL

Vikhlyaev Dmitry Romanovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article contains a description of the game "sea battle". To implement the game, the OpenGL graphics library, the GLFW cross-platform library and the C++programming language are used. The article describes the main functions and procedures of OpenGL, drawing the simplest graphics for the game, a way to display numbers using purely graphical means, recursive construction and detection of ships, processing mouse clicks. The result of the study is a game with a detailed description of its implementation.
Keywords: C ++, OpenGL, GLFW, Visual Studio, Sea Battle, Game.

1 Введение

1.1 Актуальность

OpenGL расшифровывается, как Open Graphics Library и представляет собой платформонезависимую библиотеку рисования 2D и 3D графики, а также сцен, с помощью простых примитивов. С точки зрения программирования OpenGL – это программный интерфейс, состоящий из набора функций, которые предоставляют возможность использование графики в приложениях. OpenGL имеет очень широкую сферу применения – от использования при создании компьютерных игр, до визуализации в научных исследованиях.

1.2Обзор исследований

В своей работе И. Ю. Просвирнина создала приложение «Морской бой», обладающая игровым искусственным интеллектом, в котором предусмотрен режим игры «Игрок против компьютера» [1]. В статье Н. А. Базеевой и Д. С. Лебедева описано начало игровой индустрии, а также рассмотрены особенности языков программирования для разработки игр [2]. В статье А. Д. Бирюковой и М. В. Новиковой рассматриваются все этапы создания компьютерной игры [3]. С.А. Суродин в своей статье представил сценарий углубленного изучения одного из лучших движков, существующих на данный момент, для создания красивых 2D и 3D игр[4]. Логическую игру «Буквы ареста» разработали М.Я.Арест и Е. В. Краснорукий [5].

1.3Цель исследования

Цель исследования – применяя библиотеку OpenGL подключенную к среде разработки Visual Studio и языка программирования C++, создать классическую игру «Морской бой» со всеми необходимыми для полноценной игры, механиками.

2 Материалы и методы

Для создания игры используется библиотека OpenGL, и дополнительная библиотека GLFW для открытия графического окна. Программа написана на языке программирования C++, а в качестве IDE используется Visual Studio.

3 Результаты и обсуждения

Для достижения цели в программе используется структура (struct) для описания типа поля, так как каждое поле должно содержать в себе следующую информацию: находится корабль в ячейке или нет (для двух игроков); закрыто или открыто поле; был ли выстрел в ячейку; был ли убит корабль полностью. Каждый элемент структуры влияет на конкретную ячейку карты. Хранить карту удобно в двумерном массиве, таким образом, каждая ячейка имеет свой индекс (рис. 1).

```
165  struct MyStruct
166  {
167      bool ship1;
168      bool ship2;
169      bool close;
170      bool aim;
171      bool kill;
172      bool naked;
173      bool open;
174  };
175  MyStruct map[map_w][map_h];
```

Рис. 1. Структура игрового поля

Палуба корабля имеет форму квадрата, помещённую в клетку. Чтобы начать рисование в библиотеке OpenGL используется функция glBegin().

Сама библиотека не имеет метода рисования прямоугольников, поэтому чтобы нарисовать квадрат используется метод GL_TRIANGLE_FAN. Данный метод рисует два треугольника по первым трём точкам и, соединяя две последних и первую точки, таким образом можно построить прямоугольник. Метод glColor3f() задаёт цвет точек в режиме RGB. Метод glVertex2f() задаёт местоположение точек по координатам на плоскости. Таким же способом задаются клетки карты (рис. 2).

```
void ship()
{
    glBegin(GL_TRIANGLE_FAN);
    glColor3f(0, 0, 0);  glVertex2f(0, 1);
    glColor3f(0, 0, 0);  glVertex2f(1, 1);
    glColor3f(0, 0, 0);  glVertex2f(1, 0);
    glColor3f(0, 0, 0);  glVertex2f(0, 0);
    glEnd();
}
```

Рис. 2. Рисование корабля

Далее представлена функция к описанию кода игры. До показа игры строится вражеская карта, и размещаются корабли. При каждом новом запуске игры корабли размещаются по-разному с помощью функции rand(), а те клетки, в которых находятся палубы, помечаются (рис. 3).

```
487  void game_start()
488  {
489      for (int i = 0; i < map_w; i++)
490      {
491          for (int j = 0; j < map_h; j++)
492          {
493              beat[i][j] = 1;
494          }
495      }
496      srand(time(NULL));
497      memset(map, 0, sizeof(map));
498      memset(SHIP1, 0, sizeof(SHIP1));
499      enemy_ships = 8;
500      int cnt = 8;
501      for (int i = 0; i < enemy_ships; i++)
502      {
503          if (cnt == 1) cnt = 2;
504
505          int x = rand() % map_w;
506          int y = rand() % map_h;
507          if (map[x][y].ship2) i--;
508          else if (map[x][y].close) i--;
509          else
510          {
511
512              map[x][y].ship2=true;
513              map[x][y].close = true;
514              SHIP2[di][0] = x;
515              SHIP2[di][1] = y;
516              di++;
517              building_ship(x, y,cnt/2);
518              cnt--;
519              for (int dx = -1; dx < 2; dx++)
520              {
521                  for (int dy = -1; dy < 2; dy++)
522                  {
523                      if (enemy_cell_map(x + dx+15, y + dy))
524                      {
525                          map[x + dx][y + dy].close = true;
526
527                      }
```

Рис. 3. Формирование вражеской карты

Построение кораблей последовательное, от большего количества палуб. Таким образом, у больших кораблей будет больше пространства, и они смогут располагаться в любую сторону. Для построения многопалубных кораблей используется рекурсия. Первая палуба ставится случайно, остальные подставляются в одну из четырёх сторон. Для интереса, корабли могут стоять углом или зигзагом (рис.4).

```
412  building_ship(int x, int y, int cnt)
413
414  f (cnt == 1) return;
415  nt dx;
416  nt dy;
417  nt stop = 0;
418  or (int i = 0; i < 1; i++)
419
420
421      dx = rand() % 3;
422      dy = rand() % 3;
423      if ((dx == 2) || (dy == 2)) {
424          dx = -1;
425          dy = -1;
426      }
427      if (enemy_cell_map(x + dx + 15, y + dy) && (!map[x + dx][y + dy].close) && ((dx == 0) || (dy == 0)))
428      {
429          map[x + dx][y + dy].ship2 = true;
430          map[x + dx][y + dy].close = true;
431          SHIP2[di][0] = x+dx;
432          SHIP2[di][1] = y+dy;
433          di++;
434          cnt--;
435          building_ship(x + dx, y + dy, cnt);
436          for (int i = -1; i < 2; i++)
437          {
438              for (int j = -1; j < 2; j++)
439              {
440                  if (enemy_cell_map(x + dx + i + 15, y + dy + j))
441                  {
442                      map[x + dx + i][y + dy + j].close = true;
443                  }
```

Рис. 4. Построение кораблей

Для построения морского поля строится клетка из прямых линий. С помощью функции glScaled(), уменьшается масштаб по двум осям относительно размера окна. Затем метод glTranslated() изменяет расположение клетки. Получается карта, состоящая из тесно соприкасающихся клеток, размером 10x10 для двух игроков (рис. 5).

```
535     for (int i = 0; i < map_w; i++)
536     {
537         for (int j = 0; j < map_h; j++)
538         {
539             glPushMatrix();
540             glScaled(L, L, 1);
541             glTranslated(i, j, 0);
542             if (map[i][j].ship1)
543             {
544                 ship();
545             }
546             map_building();
547             if ((beat[i][j]==0) && (map[i][j].ship1))
548             {
549                 xrest();
550             }
551             if (map[i][j].naked)
552             {
553                 point();
554             }
555             glPopMatrix();
```

Рис. 5. Создание игрового поля

Далее представлены функции, рисующие цифры по принципу семисигментного индикатора. То есть, если для данной цифры нужно нарисовать сегмент, то он рисуется по линиям. В качестве параметра в первую функцию передаются количество оставшихся кораблей. Определив принадлежность сегмента к данной цифре, вызывается вторая функция, в которую передаются координаты точек для обрисовки линии (рис.6).

```
125 void countshow(double x1, double y1, double x2, double y2, double alpha)
126 {
127     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
128     glEnable(GL_BLEND);
129     glLineWidth(2);
130     glBegin(GL_LINE_STRIP);
131     glColor4f(1, 1, 0, alpha);
132     glVertex2f(x1, y1); glVertex2f(x2, y2);
133     glEnd();
134 }
135 void count(int a)
136 {
137
138     if (a != 1 && a != 4) countshow(-0.27, 0.7, 0.27, 0.7, 1);
139     if (a != 1 && a != 4 && a != 7) countshow(-0.27, -0.7, 0.27, -0.7, 1);
140     if (a != 0 && a != 1 && a != 7) countshow(-0.27, 0, 0.27, 0, 1);
141     if (a != 5 && a != 6) countshow(0.3, 0.67, 0.3, 0.02, 1);
142     if (a != 2) countshow(0.3, -0.02, 0.3, -0.67, 1);
143     if (a != 1 && a != 2 && a != 3 && a != 7) countshow(-0.3, 0.67, -0.3, 0.02, 1);
144     if (a == 0 || a == 2 || a == 6 || a == 8) countshow(-0.3, -0.02, -0.3, -0.67, 1);
145
146 }
```

Рис. 6. Реализация процедуры рисования цифры

Для обрабатывания нажатий кнопок мыши будет использоваться метод glfwSetMouseButtonCallback(), библиотеки glfw. В качестве параметров используются переменная, в которой хранится информация об окне, и функция mouse_button_callback.

Функция принимает статические параметры о событии. Чтобы событие срабатывало при нажатии левой кнопкой мыши, в условии параметр button сравнивается с методом GLFW_MOUSE_BUTTON_LEFT, отвечающим за нажатие левой кнопки мыши. Далее переводятся координаты относительно размера окна, так чтобы совпадали с размером клетки и имели целое число. Сначала ширина экрана в пикселях становится равной единице, затем умножается на размер клетки (рис.7).

```
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
{
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS)
    {
        double x, y;
        glfwGetCursorPos(window, &x, &y);
        x = x / 2;
        x = x/hight_window*2;
        y = y/hight_window;
        x = x * 15;
        y = y * 15;
        y = y - 5;
        y = 10 - y;
        int dx = int(x), dy = int(y);
        std::cout <<dx << "\t" <<dy << std::endl;
```

Рис. 7. Обрабатывание событий при нажатии кнопки мыши

Теперь, когда события обрабатываются, можно ставить условия, с помощью которых игрок сможет сам создавать корабли и искать корабли соперника. Но для начала нужно убедиться, что игрок кликнул по собственной карте, что означает построение своих кораблей или по вражеской карте, открывая ячейки. Когда начинается игра, первый ход принадлежит игроку, если игрок промазал мимо корабля ход достаётся программе, а если попал то ход остаётся до тех пор, пока не промажет, точно также и в случае, когда ход делает программа (рис. 8).

```
if (my_cell_map(dx, dy))
{
    map[dx][dy].ship1 = true;
    SHIP1[dj][0] = dx;
    SHIP1[dj][1] = dy;
    dj++;
}
if (enemy_cell_map(dx, dy) && my_motion)
{

    if (map[dx - 15][dy].open) {
        my_motion = true;
    }
    else {
        map[dx - 15][dy].open = true;
        for (int i = 0; i < di; i++)
        {
            if ((SHIP2[i][0] == dx - 15) && (SHIP2[i][1] == dy))
            {
                map[dx - 15][dy].kill = true;
                SHIP2[i][0] = -1;
            }
        }
        if (!map[dx - 15][dy].ship2)
        {
            my_motion = false;
        }
```

Рис. 8. Условия при нажатии кнопки мыши

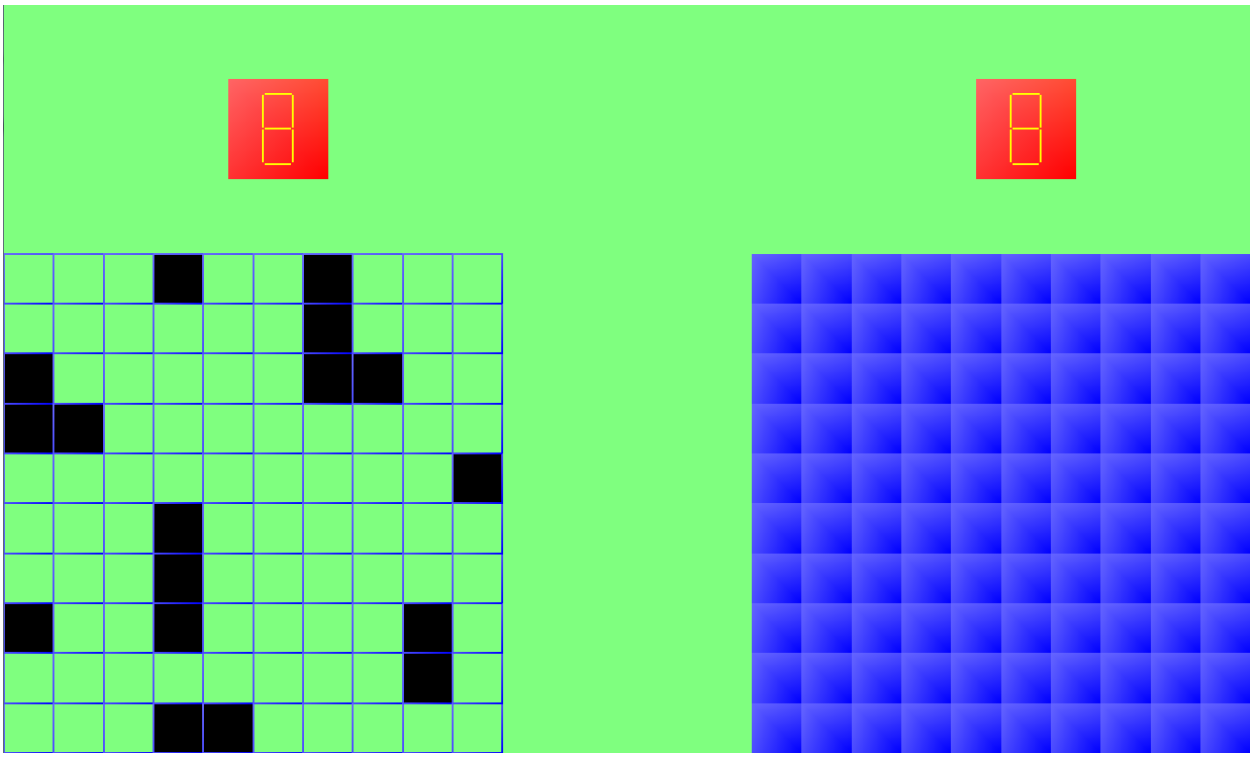

Рис. 9. Результат работы программы, начало игры

Чтобы программа могла играть с человеком, она должна не просто тыкать в любую клетку, а действительно каким-то образом быстро находить и убивать корабли. Однако для первого хода нет никаких алгоритмов,

корабль может быть в любом месте, поэтому первый ход случайный. У функции есть две части, и она принимает два параметра, координаты клетки. Координаты хранятся в глобальных переменных, в самом начале имеют значения -1. В таком случае программа понимает, что корабль не обнаружен и случайно стреляет по клеткам. Чтобы программа не попадала дважды в одну клетку, каждой клетке будет, присваивается булево значение ноль, а при попадании будет, манятся на один. Так как корабли не могут соприкасаться, ставится условие, о том, нет ли рядом уже убитого корабля. Если промах то ход передаётся игроку, а следующий ход снова случайный. Если попытка удачная, идёт проверка, убит ли корабль. Если корабль не убит, глобальным переменным передаётся координаты палубы и вызывается рекурсия, где происходит вторая часть функции (рис. 10).

```
int aim(int c_x, int c_y)
{
    srand(time(NULL));
    if ((c_x == -1) && (c_y == -1))
    {
        for (int i = 0; i < 1; i++)
        {
            bool znak = false;
            int x, y;
            x = rand() % map_w;
            y = rand() % map_h;
            for (int k = -1; k < 2; k++)
            {
                for (int j = -1; j < 2; j++)
                {
                    if (my_cell_map(x + k, y + j) && (map[x + k][y + j].aim))
                    {
                        znak = true;
                    }
                }
            }
            if (znak)
            {
                i--;
                std::cout << "znak" << std::endl;
            }
            else if (beat[x][y] == 0) i--;
            else
            {
                beat[x][y] *= 0;
                map[x][y].naked = true;
                if (map[x][y].ship1)
                {
                    map[x][y].aim = true;
                    for (int m = -1; m < 2; m++)
                    {
                        for (int n = -1; n < 2; n++)
                        {
                            if (my_cell_map(x + m, y + n) && ((m == 0 || n == 0)))
                            {
                                beat[x + m][y+n] *= 10;
                            }
                        }
                    }
                    for (int i = 0; i < dj; i++)
                    {
                        if ((SHIP1[i][0] == x) && (SHIP1[i][1] == y))
                        {
                            SHIP1[i][0] = -1;
```

Рис. 10. Реализация функции поиска кораблей

Чтобы полностью убить корабль, нужно стрелять вокруг той палубы, в которую случайно был сделан выстрел. При этом если случится неудачная попытка, нужно запомнить последнее попадание, чтобы от него отталкиваться, за это отвечают глобальные переменные. Суть алгоритма проста, если случилась удачная попытка и корабль не убит, вызывается рекурсия с координатами последнего попадания. Далее проверяются клетки с лева направо и сверху вниз. Но в таком случае при отстреливании многопалубного корабля, программа не сможет найти два края одного корабля. Поэтому каждой клетке присваивается целочисленное значение один, а при попадании в палубу корабля, значения клеток вокруг умножаются на десять, а при любом выстреле значение клетки умножается на ноль. Таким образом, корабль не потеряется. После того как система проверит четыре клетки вокруг и не найдёт палубу, при этом корабль не будет убит, программа начнёт искать клетки имеющие высокие значения. После того как корабль был полностью убит все высокие значения обнуляются. Значение глобальных переменных снова становится равным -1. (рис. 11).

```
for (int dx = -1; dx < 2; dx++)
{
    for (int dy = -1; dy < 2; dy++)
    {
        if (((c_x + dx) < 10) && ((c_x + dx) >= 0) && ((c_y + dy) < 10) && ((c_y + dy) >= 0) && (beat[c_x + dx][c_y + dy] != 0) && ((dx == 0) || (dy == 0)))
        {
            std::cout << "cikl" << std::endl;
            beat[c_x + dx][c_y + dy] = 0;
            map[c_x + dx][c_y + dy].naked = true;
            if (map[c_x + dx][c_y + dy].ship1)
            {
                map[c_x + dx][c_y + dy].aim = true;
                std::cout << "ship1" << std::endl;
                for (int i = 0; i < dj; i++)
                {
                    if ((SHIP1[i][0] == c_x + dx) && (SHIP1[i][1] == c_y + dy))
                    {
                        std::cout << "SHIP1[i][0]=-1" << std::endl;
                        SHIP1[i][0] = -1;
                    }
                }
                if (tempare > foo(my_ships)){
                    std::cout << "tempare" << std::endl;
                    tempare--;
                    cnt_x = -1;
                    cnt_y = -1;
                    for (int fx = -1; fx < 2; fx++)
                    {
                        for (int fy = -1; fy < 2; fy++)
                        {
                            if (my_cell_map(c_x + dx + fx, c_y + dy + fy))
                            {
                                std::cout << "8*0" << std::endl;
                                beat[c_x + dx + fx][c_y + dy + fy] *= 0;
                            }
                        }
                    }
                    for (int m = 0; m < map_w; m++)
                    {
                        for (int n = 0; n < map_h; n++)
                        {
                            if (beat[m][n] >= 10){
                                beat[m][n] *= 0;
                            }
                        }
                    }
                    return 0;
                }
                else {
                    std::cout << "4*10" << std::endl;
                    for (int m = -1; m < 2; m++)
                    {
                        for (int n = -1; n < 2; n++)
                        {
                            if ((my_cell_map(c_x + dx + m, c_y + dy + n)) && ((m==0 || n==0))){
                                beat[c_x + dx + m][c_y + dy + n] *= 10;
                            }
```

Рис. 11. Реализация функции поиска палуб

Если количество кораблей одного из игроков становится равным нулю, появляется табло о победе или проигрыше. Буквы были нарисованы из простых линий тем же способом что и цифры. Метод glLineWidth() задаёт ширину линии. Метод GL_LINE_STRIP рисует линии, соединяя две последние точки (рис 12,13,14).

```
glLineWidth(8);
glBegin(GL_LINE_STRIP);
glColor3f(1, 1, 0);
glVertex2f(0.1, 0.9);
glVertex2f(0.2, 0.1);
glVertex2f(0.3, 0.9);
glVertex2f(0.4, 0.1);
glVertex2f(0.5, 0.9);
glEnd();

glBegin(GL_LINE_STRIP);
glColor3f(1, 1, 0);
glVertex2f(0.6, 0.9);
glVertex2f(0.6, 0.1);
glEnd();

glBegin(GL_LINE_STRIP);
glColor3f(1, 1, 0);
glVertex2f(0.7, 0.1);
glVertex2f(0.7, 0.9);
glVertex2f(0.9, 0.1);
glVertex2f(0.9, 0.9);
glEnd();
```

Рис. 12. Рисование букв из простых линий

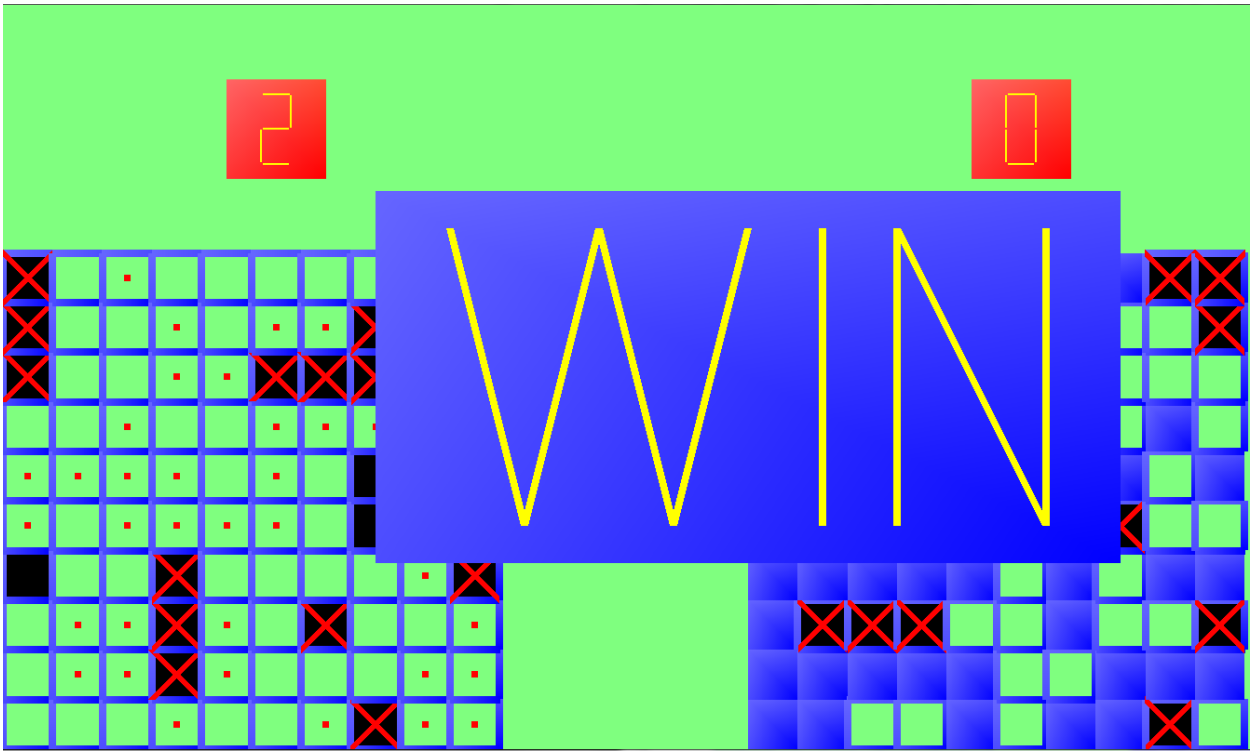

Рис. 13. Победа игрока

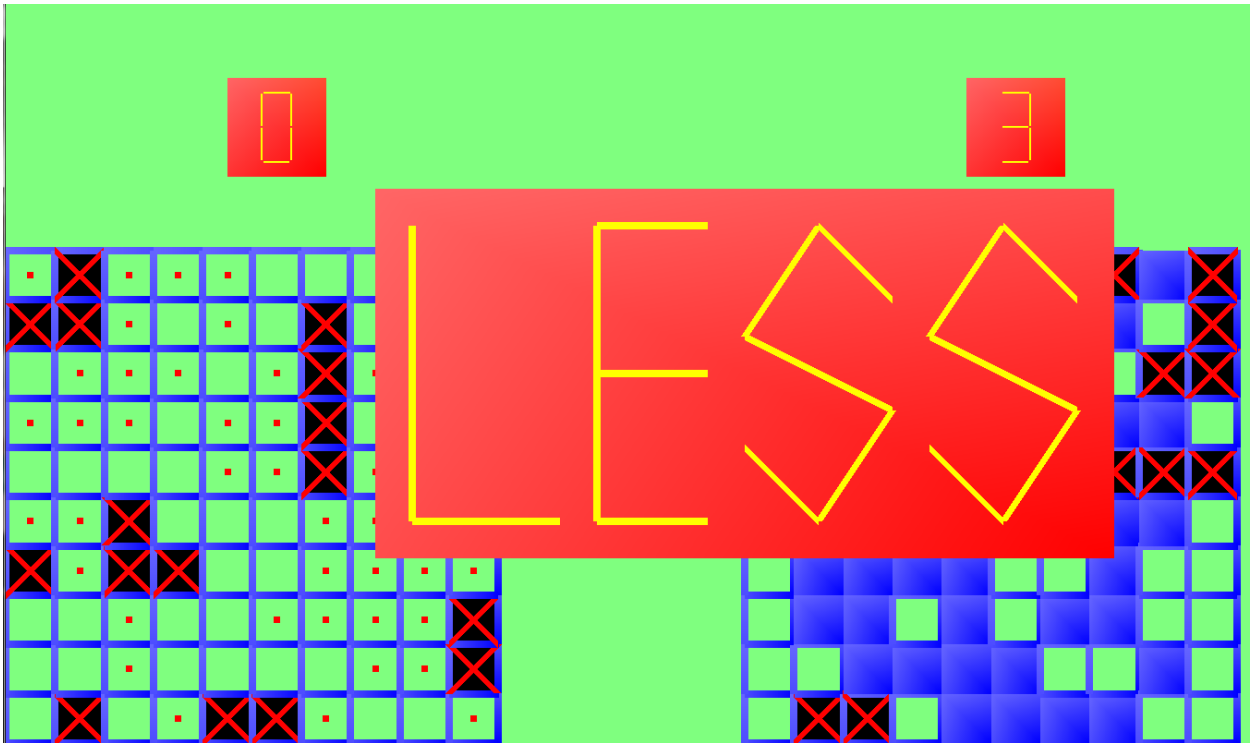

Рис. 14. Победа программы

Таким образом, была написана программа для создания классической игры «морской бой» с использованием чисто графических методов графической библиотеки OpenGL и языка программирования C++.

Библиографический список

1. Просвирнина И. Ю., Егунова А. И., Аббакумов А. А. Среда разработки Microsoft Visual Studio на примере создания игры "Морской бой" // Интеграционные процессы в науке в современных условиях. 2017. С. 123-125.
2. Базеева Н. А., Лебедев Д. С. Языки программирования для создания игр //E-Scio. 2019. №4. С. 31-39.
3. Бирюкова А. Д., Новикова М. В. Жизненный цикл разработки компьютерных игр // Вестник научных конференций. 2016. №12-4. С. 24-25.
4. Суродин С.А. Unity 3D. разработка сценария проектирования в среде Unity 3D// Информатика и вычислительная техника. 2015. №3. С. 504-511.
5. Арест М.Я., Красноруцкий Е.В. Логическая игра "Буквы ареста". Патент на изобретение RUS 2064316.
6. Смирнова М.В. Основной принцип построения графических объектов//Актуальные проблемы гуманитарных и естественных наук. 2016. № 2-2. С. 93-97.
7. Беляев В.В. Опыт классификации логических игр // Логико-философские штудии. 2008. №6. С. 102-118.
8. Гайнуллин Р.Ф., Захаров В.А., Аксенова Е.А. Создание 2d игры на Unity 3D 5.4 // Вестник современных исследований. 2018. №4. С. 78-82.
9. Худайберганов Т.Р., Адинаев Х.С. Библиотеки OpenGL и directx для программирования трехмерной графики // Современная техника и технологии. 2017. № 5 (69). С. 27.