1. Ministry of Education and Science of the Russian Federation
2. St. Petersburg State Polytechnic University
3. —
4. **Cybersecurity and Information Security department**

**REPORT**

**FOR Laboratory work № 2**

1. **«Creating UML diagram»**
2. subject «OOP»
3. Completed by
4. student from gr. 4851003/90801 A.G. Kuleeva

<*signature*>

1. Checked by
2. teacher’s assistant A.D. Dahnovich

<*signature*>

1. Saint – Petersburg

2021

The **goal** of this laboratory work is to familiarize with UML diagrams, create own UML and redesign old code in OOP style.

1. Task

The main tasks of this laboratory work are:

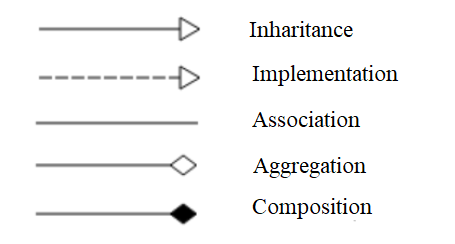
* study the proposed theoretical material UML.docx;
* build a UML class diagram: specify access modifiers, as well as the power of class relationships, justify the type of relationship used;
* develop a UML class diagram for an existing project implemented in C;
* implement a program in C++ that reflects the developed class diagram and preserves the functionality of the project under study in C.

2. Theory

UML (Unified Modeling Language) is a graphic language which helps to describe, visualize, create and document object-oriented systems. Main elements of UML are essences, relationships and diagrams. Essences are the key abstracts of language, relationships connect essences together, diagrams are graphic view of a number of essences and relationships. There are lots of types of diagrams for each case. Classes and interfaces are basic essences.

Attribute is a piece of information associated with class (fields and methods). They can be public, private and protected. Public means that any other class can see and change this attribute, in UML it defines as “+”. Private means that no other class can see or change this attribute, in UML it defines as “-”. Protected means that attribute can be seen or changed only by parent class or its’ children, in UML it defines as “#”.

Speaking about relationships there are 5 basic types (see Picture 1). Composition means that object consists of some details and can’t live without them (car can’t move without wheels). Aggregation means that objects have some connection but they are not in “part of” relationships. For example, either man or car can exist without each other. Association means that objects has the same rights.

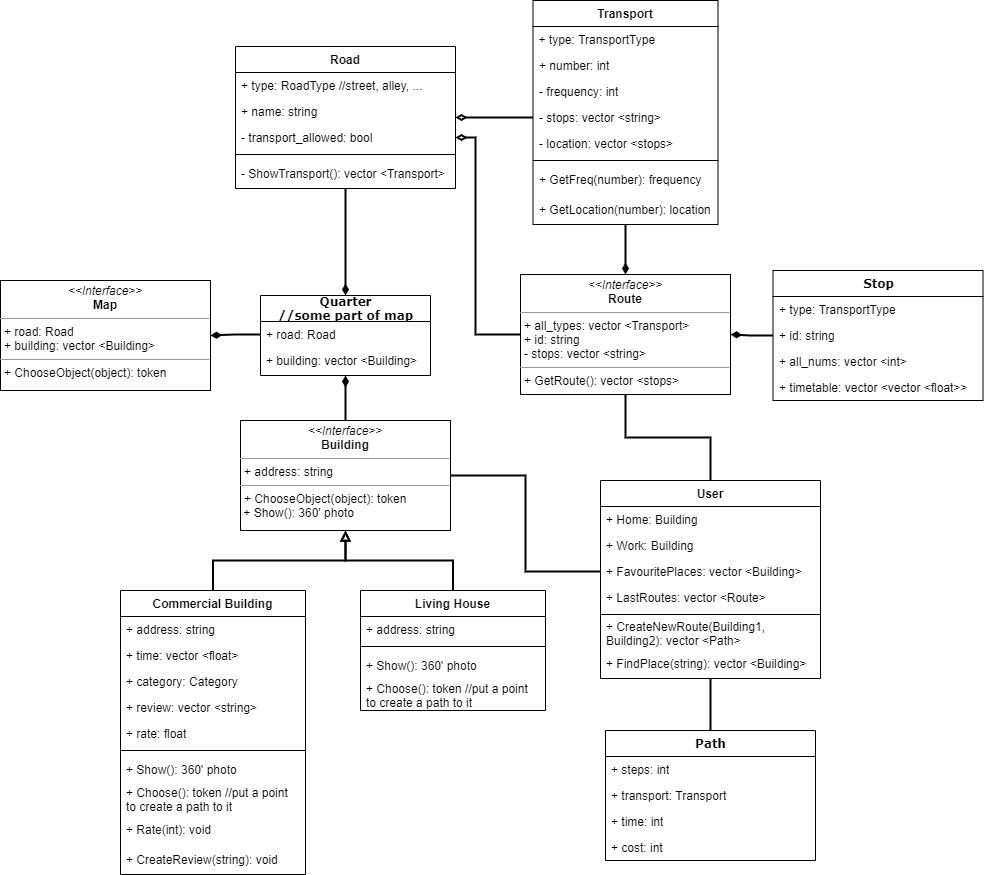


Picture 1 ― Relationships

3. Implementation

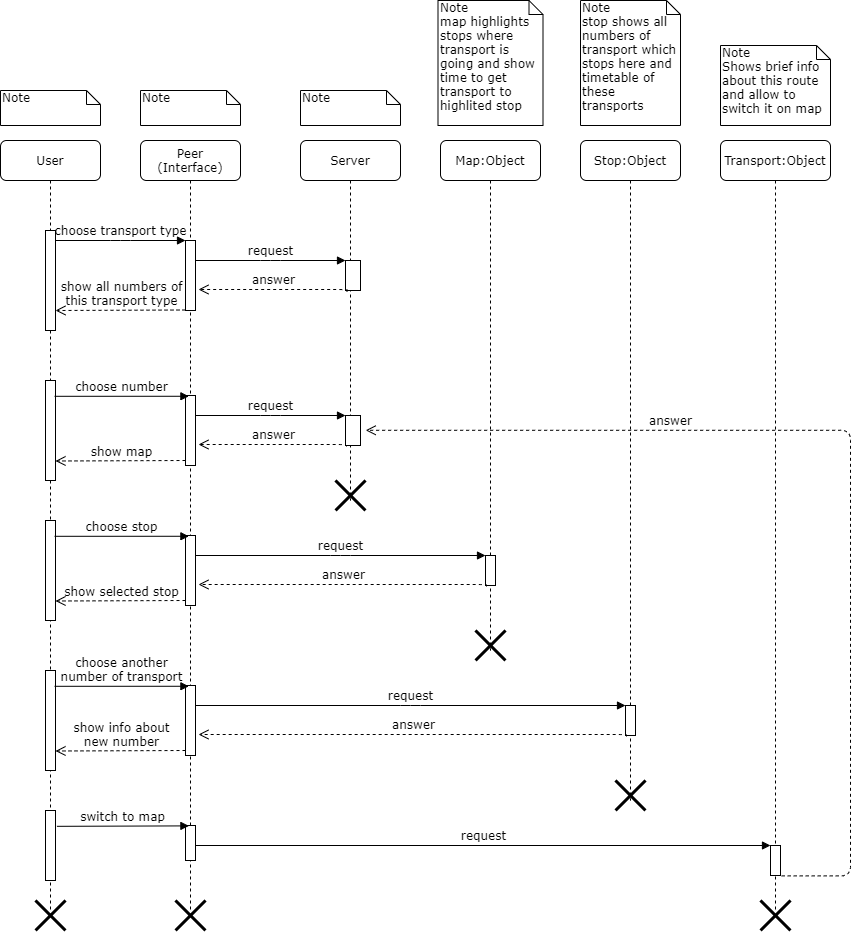
## UML

I had to implement sequence diagram of Yandex-maps with monitoring public transport. On Picture 2 you can see class diagram.



Picture 2 ― Class diagram

On Picture 3 there is sequence diagram.



Picture 3 ― Sequence diagram

Then below you can read use-case. Actor wants to go from point A to point B

1. Actor knows which transport he needs:
   1. Choose tram type
   2. Choose number 1

App opens map

* 1. App highlights next stop of tram and shows time to get to all stops
  2. Actor waits for his tram and then gets to point B

1. Actor don’t know which transport he needs:
   1. Open map
   2. Choose stop where he is now
   3. Choose point B

App shows all routes, e.g. bus, minibus, taxi, afoot

* 1. Choose most suitable variant, let it be bus 27
  2. App shows route of this bus, frequency, how much to wait it
  3. Actor waits for it and goes to point B

1. Actor searches for unexciting building or street (e.g. Leni street):
   1. “Nothing was found for this request”
   2. “Perhaps, you meant «Lenin street»” ― suggest alternative variant.
2. Some building is closed:
   1. In infobox about building write “work time: CLOSED”
   2. If user still wants to create a path to this place show extra massage
3. Some routes were updated or deleted
   1. Notify this info in route list
   2. Notify it on every stop of this route

## Project refactoring

I choose my course work from second semester to rewrite it in oop style. It has more than 800 lines of C code. I analyzed it and found these anti-patterns:

1. Bad naming.
   1. unsigned short check(UKKletka check1, UKKletka check2) ― check what?
   2. unsigned short SecondKing(UKKletka first) ― what does it mean?
   3. unsigned short NickiMinaj(UKKletka first, UKKletka second) ― this function do something with queen checker. Nicky Minaj is rap queen so we named this func in honor of her, but now I don’t understand what does it mean.
   4. unsigned short CheckSecond(UKKletka first, UKKletka second) ― what does it mean? Who is Second?
   5. void two\_people() ― this meant two players mode, but finally there is only one mode, so it’s better to rename it.
2. No comments or bad comments. It would be easier to understand these bad names if there were some comments what this function does.
3. Placing code in main but it could be separate function

Example:

int main() {

unsigned short players = 0;

inicialization();

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

//rules

printf("\n\t\t\tДобро пожаловать в игру \"Шашки\"!\n\n\tПравила:\n");

printf("\t1. Обозначения:\n\t\tw -- белая пешка\n\t\tb -- чёрная пешка\n\t\tW -- белая дамка\n\t\tB -- чёрная дамка\n");

printf("\t\t- -- поле, на котором нет шашки, но она может быть там\n\t\tНа пустые поля ходить нельзя!\n");

printf("\t2. На пустые поля ходить нельзя!\n");

printf("\t3. Чтобы сделать очередной ход, пользователь вводит с клавиатуры номер поля, указывающего на пешку,\n\t");

printf(" которую он хотел бы передвинуть, и номер поля, куда он хотел бы её передвинуть через пробел.\n");

printf("\t4. Номер поля состоит из буквы столбца и номера строки, записанных по очереди через пробел (например, А1 В2).\n");

printf("\t5. Обычные шашки не могут ходить назад.\n");

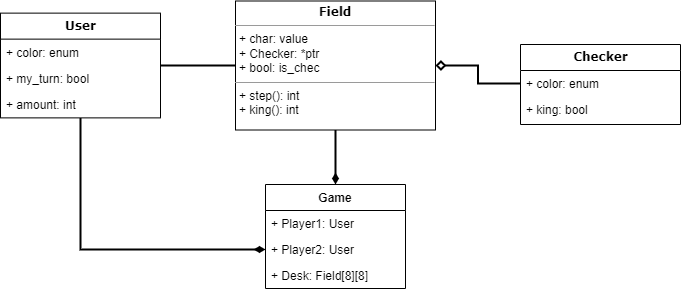
printf("\t6. Бить назад можно.\n");

printf("\t7. Дамки могут ходить в любом направлении на любое количество клеток, в том числе и назад. \n");

two\_people();

}

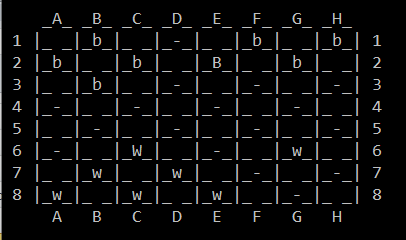
1. All code in one file (800+ strings of code). It’s better to separate algorithm, graphics, user massages, etc.
2. Also while rewriting code unused function was found.
3. Repeat code: if user has opportunity to do one more step we wrote one more “read coordinates” code instead of recursive call of function.



Picture 4 ― UML diagram of new project

After refactoring there are 562 strings: 66 for class implementation, 27 in main, 465 in play.cpp. Our previous work also was bad in data structures. Changing list to array also helped to shorten the code. For example there is no need to search for every field in list because we can call them by index in array. This also make program faster.

In init() function I initialize Users, Checkers and Field. Class Field has \*ptr to Checker. So in deinit() func we have to delete all pointers. In play() function we read coordinates of user’s command. Then check() function is called to check that coordinates are correct. If check() returned true we can do step. It can be simple step or killing enemy figure. Also in step() func there is checkup if simple figure became king. For kings there are different functions. There is extra checking for correct coordinates. While this checkup also we look through diagonal to find out if this simple step or killing. If it is killing check\_king returns coordinates of enemy figure. King’s step is similar to simple figure’s step.



Picture 5 ― Example of game interface

Appendix

enum Color{black, white};

class Checker {

public:

Color color;

bool king;

Checker();

Checker(const Checker &other);

//~Checker();

};

class Field {

public:

char value;

Checker \*ptr; //there is a fig

bool is\_chec; //is here some fig?

Field();

Field(const Field &other);

~Field();

int step(unsigned int i\_start, unsigned int j\_start, unsigned int i\_end, unsigned int j\_end);

int king(unsigned int i\_start, unsigned int j\_start, unsigned int i\_end, unsigned int j\_end);

};

class User {

public:

Color color;

bool my\_turn;

unsigned short amount;

};

typedef struct coord {

unsigned int i, j;

} coord;

Checker::Checker() {

this->king = false;

}

Checker::Checker(const Checker &other)

{

this->color = other.color;

this->king = other.king;

}

Field::Field(const Field &other)

{

this->value = other.value;

this->ptr = other.ptr;

this->is\_chec = other.is\_chec;

}

Field::Field() {

this->is\_chec = false;

}

Field::~Field() {

delete this;

}

void print\_msg() {

//rules

printf("\n\t\t\tДобро пожаловать в игру \"Шашки\"!\n\n\tПравила:\n");

printf("\t1. Обозначения:\n\t\tw -- белая пешка\n\t\tb -- чёрная пешка\n\t\tW -- белая дамка\n\t\tB -- чёрная дамка\n");

printf("\t\t- -- поле, на котором нет шашки, но она может быть там\n\t\tНа пустые поля ходить нельзя!\n");

printf("\t2. На пустые поля ходить нельзя!\n");

//printf("\t3. Чтобы выбрать режим игры (один или два игрока) необходимо ввести с клавиатуры число 1 или 2 соотвественно.\n");

printf("\t3. Чтобы сделать очередной ход, пользователь вводит с клавиатуры номер поля, указывающего на пешку,\n\t");

printf(" которую он хотел бы передвинуть, и номер поля, куда он хотел бы её передвинуть через пробел.\n");

printf("\t4. Номер поля состоит из буквы столбца и номера строки, записанных по очереди через пробел (например, А1 В2).\n");

printf("\t5. Обычные шашки не могут ходить назад.\n");

printf("\t6. Бить назад можно.\n");

printf("\t7. Дамки могут ходить в любом направлении на любое количество клеток, в том числе и назад. \n");

}

void main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

print\_msg(); //welcome message for players

init(); //initialize playground and all figures

play(); // go play!

deinit();

}

User user1, user2;

Checker chec[24];

Field\*\* field;

Field\*\* init() {

user1.color = white;

user1.my\_turn = true;

user1.amount = 12;

user2.color = black;

user2.my\_turn = false;

user2.amount = 12;

for (unsigned int i = 0; i < 12; i++) {

chec[i].color = black;

chec[23 - i].color = white;

}

field = new Field\*[8];

for (unsigned int i = 0; i < 8; i++)

field[i] = new Field[8];

char tmp; //to put colors

unsigned int index = 0;

for (unsigned int i = 0; i < 8; i++) {

if (i < 3)

tmp = 'b';

else if (i < 5)

tmp = '-';

else tmp = 'w';

for (unsigned int j = 0; j < 8; j++) {

if (i % 2 == 0 && j % 2 == 1 || i % 2 == 1 && j % 2 == 0) { //set figs like " i i"j = j;

field[i][j].value = tmp;

if (tmp != '-') {

field[i][j].is\_chec = true;

field[i][j].ptr = &chec[index];

index++;

if (tmp == 'b')

chec[index].color = black;

else

chec[index].color = white;

}

}

else {

field[i][j].value = ' ';

}

}

}

return field;

}

void deinit() {

for (unsigned int i = 0; i < 8; i++) {

for (unsigned int j = 0; j < 8; j++) {

if (field[i][j].is\_chec) {

field[i][j].ptr = nullptr;

field[i][j].is\_chec = false;

}

}

}

}

void print\_field() {

unsigned short x = 1;

printf("\n ");

for (unsigned short i = 'A'; i < 'A' + 8; i++)

printf("\_%c\_ ", i);

for (unsigned short i = 0; i < 8; i++) {

for (unsigned short j = 0; j < 8; j++) {

if (j == 0)

printf("\n %hu ", x);

printf("|\_%c\_", field[i][j].value);

if (j == 7) {

printf("| %hu ", x);

x++;

}

}

}

printf("\n ");

for (unsigned short i = 'A'; i < 'A' + 8; i++)

printf(" %c ", i);

}

void play() {

int count;

unsigned int i\_start, j\_start, i\_end, j\_end;

char ch;

while (user1.amount && user2.amount) {

count = -1;

print\_field();

if (user1.my\_turn) printf("\n\n\tХод белых.");

else printf("\n\n\tХод чёрных.");

printf(" Введите координаты своего хода: ");

j\_start = getchar() - 'A';

i\_start = getchar() - '0' - 1;

ch = getchar(); //пропускаем пробел

j\_end = getchar() - 'A';

i\_end = getchar() - '0' - 1;

ch = getchar();

if (ch == '\n') {

if (check(i\_start, j\_start, i\_end, j\_end)) {

if (field[i\_start][j\_start].ptr->king) count = field[i\_start][j\_start].king(i\_start, j\_start, i\_end, j\_end);

else count = field[i\_start][j\_start].step(i\_start, j\_start, i\_end, j\_end);

}

}

else {

printf("\tНеправильный формат ввода!\n");

while (ch != '\n') ch = getchar();

}

if (count == 0) {

if (user1.my\_turn) {

user1.my\_turn = false;

user2.my\_turn = true;

}

else {

user1.my\_turn = true;

user2.my\_turn = false;

}

system("cls");

}

else {

system("cls");

printf("\n\tДоступен еще один ход шашки.\n");

}

}

if (user1.amount) printf("\n\n\n\t\tЧёрные победили!");

else printf("\n\n\n\t\tБелые победили!");

}

bool check(unsigned int i\_start, unsigned int j\_start, unsigned int i\_end, unsigned int j\_end) {

if (i\_start > 8 || i\_end > 8 || j\_start > 8 || j\_end > 8) {

printf("\tОшибка! Неверные координаты!\n");

return false;

}

if (!field[i\_start][j\_start].is\_chec) { //is fig here?

printf("\tОшибка! Неверные координаты!\n");

return false;

}

if (field[i\_end][j\_end].is\_chec) { //is this field empty?

printf("\tОшибка! Неверные координаты!\n");

return false;

}

if (field[i\_end][j\_end].value != '-') { //is this field available for step?

printf("\tОшибка! Неверные координаты!\n");

return false;

}

if (field[i\_start][j\_start].ptr->king)

return true; // check\_king(i\_start, j\_start, i\_end, j\_end, field);

else { //no steps back for simple fig

if (field[i\_start][j\_start].value == 'b' && i\_start > i\_end && j\_start > j\_end) {

if (field[i\_start - 1][j\_start - 1].value != '-' && //gonna beat?

field[i\_start - 1][j\_start - 1].value != 'w' && //can't beat yourself

field[i\_start - 1][j\_start - 1].value != 'W') {

printf("\tОшибка! Неверные координаты!\n");

return false;

}

}

else

if (field[i\_start][j\_start].value == 'w' && i\_start < i\_end && j\_start < j\_end) {

if (field[i\_start - 1][j\_start - 1].value != '-' && //gonna beat?

field[i\_start - 1][j\_start - 1].value != 'b' && //can't beat yourself

field[i\_start - 1][j\_start - 1].value != 'B') {

printf("\tОшибка! Неверные координаты!\n");

return false;

}

}

}

return true;

}

int Field::step(unsigned int i\_start, unsigned int j\_start, unsigned int i\_end, unsigned int j\_end) {

char tmp;

//check for king

if (i\_end == 7 && user2.my\_turn) {

this->value = 'B';

this->ptr->king = true;

}

else if (i\_end == 0 && user1.my\_turn) {

this->value = 'W';

this->ptr->king = true;

}

if (user1.my\_turn && i\_start - 1 == i\_end || user2.my\_turn && i\_start == i\_end - 1) { //simple step

tmp = this->value;

field[i\_end][j\_end].value = tmp;

this->value = '-';

this->is\_chec = false;

field[i\_end][j\_end].is\_chec = true;

field[i\_end][j\_end].ptr = this->ptr;

this->ptr = NULL;

return 0;

} //beat

else if ((j\_start + 1 == j\_end - 1 || j\_start - 1 == j\_end + 1) && (i\_start + 1 == i\_end - 1 || i\_start - 1 == i\_end + 1)) {

//move fig

tmp = this->value;

field[i\_end][j\_end].value = tmp;

this->value = '-';

this->is\_chec = false;

field[i\_end][j\_end].is\_chec = true;

field[i\_end][j\_end].ptr = this->ptr;

this->ptr = NULL;

//del fig

unsigned int i\_mid, j\_mid;

if (i\_start + 1 == i\_end - 1) i\_mid = i\_start + 1;

else i\_mid = i\_start - 1;

if (j\_start + 1 == j\_end - 1) j\_mid = j\_start + 1;

else j\_mid = j\_start - 1;

field[i\_mid][j\_mid].value = '-';

field[i\_mid][j\_mid].is\_chec = false;

this->ptr = NULL;

if (user1.my\_turn) user2.amount--;

else user1.amount--;

return 1;

}

else {

printf("\tОшибка! Неверные координаты!\n");

return -1;

}

}

int Field::king(unsigned int i\_start, unsigned int j\_start, unsigned int i\_end, unsigned int j\_end) {

coord c = check\_king(i\_start, j\_start, i\_end, j\_end); //check this step is available

char tmp;

if (c.i == 0 && c.j == 0)

return -1;

else if (c.i == 10 && c.j == 10) { //simple step

tmp = this->value;

field[i\_end][j\_end].value = tmp;

this->value = '-';

this->is\_chec = false;

field[i\_end][j\_end].is\_chec = true;

field[i\_end][j\_end].ptr = this->ptr;

this->ptr = NULL;

return 0;

} //beat

else {

//move fig

tmp = this->value;

field[i\_end][j\_end].value = tmp;

this->value = '-';

this->is\_chec = false;

field[i\_end][j\_end].is\_chec = true;

field[i\_end][j\_end].ptr = this->ptr;

this->ptr = NULL;

//del fig

unsigned int i\_mid, j\_mid;

i\_mid = c.i;

j\_mid = c.j;

field[i\_mid][j\_mid].value = '-';

field[i\_mid][j\_mid].is\_chec = false;

this->ptr = NULL;

if (user1.my\_turn) user2.amount--;

else user1.amount--;

return 1;

}

}

coord check\_king(unsigned int i\_start, unsigned int j\_start, unsigned int i\_end, unsigned int j\_end) {

unsigned short i, j;

bool flag = false; //chek that we gonna beat only one fig

coord c;

c.i = 10;

c.j = 10;

if (i\_start > i\_end) { //влево

if (j\_start > j\_end) { //вверх

if (i\_start - i\_end == j\_start - j\_end) { //is this diagonal?

for (i = i\_start - 1, j = j\_start - 1; i > i\_end && j > j\_end; i--, j--) {

if (user1.my\_turn) {

if (field[i][j].value == 'W' || field[i][j].value == 'w') {

printf("\tОшибка! Нельзя бить свои фигуры!\n");

c.i = 0;

c.j = 0;

}

if (field[i][j].value != '-') { //if this is not step but kill

if (flag) { //can't beat two figs at once

c.i = 0;

c.j = 0;

}

else{

c.i = i;

c.j = j;

flag = true;

}

}

}

else {

if (field[i][j].value == 'B' || field[i][j].value == 'b') {

printf("\tОшибка! Нельзя бить свои фигуры!\n");

c.i = 0;

c.j = 0;

}

if (field[i][j].value != '-') { //if this is not step but kill

if (flag) { //can't beat two figs at once

c.i = 0;

c.j = 0;

}

else {

c.i = i;

c.j = j;

flag = true;

}

}

}

}

}

else {

printf("\tОшибка! Неверные координаты!\n");

c.i = 0;

c.j = 0;

}

}

else if (j\_start < j\_end) { //вниз

if (i\_start - i\_end == j\_end - j\_start) { //is this diagonal?

for (i = i\_start - 1, j = j\_start + 1; i > i\_end && j < j\_end; i--, j++) {

if (user1.my\_turn) {

if (field[i][j].value == 'W' || field[i][j].value == 'w') {

printf("\tОшибка! Нельзя бить свои фигуры!\n");

c.i = 0;

c.j = 0;

}

if (field[i][j].value != '-') { //if this is not step but kill

if (flag) { //can't beat two figs at once

c.i = 0;

c.j = 0;

}

else {

c.i = i;

c.j = j;

flag = true;

}

}

}

else {

if (field[i][j].value == 'B' || field[i][j].value == 'b') {

printf("\tОшибка! Нельзя бить свои фигуры!\n");

c.i = 0;

c.j = 0;

}

if (field[i][j].value != '-') { //if this is not step but kill

if (flag) { //can't beat two figs at once

c.i = 0;

c.j = 0;

}

else {

c.i = i;

c.j = j;

flag = true;

}

}

}

}

}

else {

printf("\tОшибка! Неверные координаты!\n");

c.i = 0;

c.j = 0;

}

}

}

else if (i\_start < i\_end) { //вправо

if (j\_start > j\_end) { //вверх

if (i\_end - i\_start == j\_start - j\_end) { //is this diagonal?

for (i = i\_start + 1, j = j\_start - 1; i < i\_end && j > j\_end; i++, j--) {

if (user1.my\_turn) {

if (field[i][j].value == 'W' || field[i][j].value == 'w') {

printf("\tОшибка! Нельзя бить свои фигуры!\n");

c.i = 0;

c.j = 0;

}

if (field[i][j].value != '-') { //if this is not step but kill

if (flag) { //can't beat two figs at once

c.i = 0;

c.j = 0;

}

else {

c.i = i;

c.j = j;

flag = true;

}

}

}

else {

if (field[i][j].value == 'B' || field[i][j].value == 'b') {

printf("\tОшибка! Нельзя бить свои фигуры!\n");

c.i = 0;

c.j = 0;

}

if (field[i][j].value != '-') { //if this is not step but kill

if (flag) { //can't beat two figs at once

c.i = 0;

c.j = 0;

}

else {

c.i = i;

c.j = j;

flag = true;

}

}

}

}

}

else {

printf("\tОшибка! Неверные координаты!\n");

c.i = 0;

c.j = 0;

}

}

else if (j\_start < j\_end) { //вниз

if (i\_end - i\_start == j\_end - j\_start) { //is this diagonal?

for (i = i\_start + 1, j = j\_start + 1; i < i\_end && j < j\_end; i++, j++) {

if (user1.my\_turn) {

if (field[i][j].value == 'W' || field[i][j].value == 'w') {

printf("\tОшибка! Нельзя бить свои фигуры!\n");

c.i = 0;

c.j = 0;

}

if (field[i][j].value != '-') { //if this is not step but kill

if (flag) { //can't beat two figs at once

c.i = 0;

c.j = 0;

}

else {

c.i = i;

c.j = j;

flag = true;

}

}

}

else {

if (field[i][j].value == 'B' || field[i][j].value == 'b') {

printf("\tОшибка! Нельзя бить свои фигуры!\n");

c.i = 0;

c.j = 0;

}

if (field[i][j].value != '-') { //if this is not step but kill

if (flag) { //can't beat two figs at once

c.i = 0;

c.j = 0;

}

else {

c.i = i;

c.j = j;

flag = true;

}

}

}

}

}

else {

printf("\tОшибка! Неверные координаты!\n");

c.i = 0;

c.j = 0;

}

}

}

return c;

}