Продолжим создание нашего проекта. Перейдем к классу game в файл game.java (см. рис. 1).

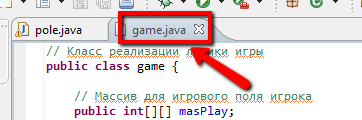


Рис. 1

Добавим необходимые свойства (переменные) класса:

// Класс реализации логики игры

public class game {

// Массив для игрового поля игрока

public int[][] masPlay;

// Массив для игрового поля компьютера

public int[][] masComp;

//Признак хода компьютера (false - ходит игрок)

public boolean compHod;

// Признак конца игры

// (0-игра идет, 1-победил игрок,2-победил компьютер)

public int endg;

Мы добавили массив для игрового поля компьютера masComp, переменную compHod для определения очередности хода и признак конца игры endg.

В конструкторе класса добавим создание массива для игрового поля компьютера:

// Конструктор класса

public game() {

//Создаем массив 10x10 - игровое поле игрока

masPlay = new int[10][10];

//Создаем массив 10x10 - игровое поле компьютера

masComp = new int[10][10];

}

В методе start() добавим обнуление массива компьютера, начальные значения переменных compHod, endg. А также метод rasstanovka(),который вызывается в конструкторе и выполняет расстановку всех кораблей:

//Запуск игры - начало игры

public void start() {

//Очищаем игровое поле игрока и компьютера

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

masPlay[i][j] = 0;

masComp[i][j] = 0;

}

}

//Обнуляем признак чьей-то победы

endg = 0;

//Передаем первый ход игроку

compHod = false;

//Расставляем корабли игрока

rasstanovkaKorabley(masPlay);

//Расставляем корабли компьютера

rasstanovkaKorabley(masComp);

}

// Расстановка кораблей

private void rasstanovkaKorabley(int[][] mas) {

//Создаем один четырехпалубный корабль

make4P(mas, 4);

//Создаем два трехпалубных корабля

for (int i = 1; i <= 2; i++)

make4P(mas, 3);

//Создаем три двухпалубных корабля

for (int i = 1; i <= 3; i++)

make4P(mas, 2);

//Создаем четыре однопалубных корабля

make1P(mas);

}

Обратите внимание, что расстановку кораблей будем выполнять,начиная с самого большого, и идти по уменьшению размера! Метод make1P() уже создан для расстановки однопалубных кораблей. Метод make4P() мы доработаем так, чтобы он мог создавать корабль из нескольких палуб, любого размера. При вызове, второй параметр метода будет указывать количество палуб. Заменим метод make4P() на усовершенствованный вариант:

// Создание корабля с несколькими палубами от 2-х до 4-х

private void make4P(int[][] mas, int kolPaluba) {

//Глухой цикл

while (true) {

boolean flag = false;

// Координаты головы корабля

int i = 0, j = 0;

// Создание первой палубы - головы корабля

// Получение случайной строки

i = (int) (Math.random() \* 10);

// Получение случайной колонки

j = (int) (Math.random() \* 10);

// Выбираем случайное направление построения корабля

// 0 - вверх, 1 -вправо, 2 - вниз, 3 - влево

int napr = (int) (Math.random() \* 4);

if (testNewPaluba(mas, i, j) == true) {

if (napr == 0) // вверх

{

// Если можно расположить палубу

if (testNewPaluba(mas, i -(kolPaluba - 1), j) == true)

flag = true;

}

else if (napr == 1) // вправо

{

// Если можно расположить палубу

if (testNewPaluba(mas, i, j + (kolPaluba - 1)) == true)

flag = true;

}

else if (napr == 2) // вниз

{

// Если можно расположить палубу

if (testNewPaluba(mas, i + (kolPaluba - 1), j) == true)

flag = true;

}

else if (napr == 3) // влево

{

//Если можно расположить палубу

if (testNewPaluba(mas, i, j -(kolPaluba - 1)) == true)

flag = true;

}

}

if (flag == true) {

//Помещаем в ячейку число палуб

mas[i][j] = kolPaluba;

// Окружаем минус двойками

okrBegin(mas, i, j, -2);

if (napr == 0) // вверх

{

for (int k = kolPaluba - 1; k >= 1; k--) {

//Помещаем в ячейку число палуб

mas[i -k][j] = kolPaluba;

//Окружаем минус двойками

okrBegin(mas, i - k, j, -2);

}

}

else if (napr == 1) // вправо

{

for (int k = kolPaluba - 1; k >= 1; k--) {

//Помещаем в ячейку число палуб

mas[i][j + k] = kolPaluba;

//Окружаем минус двойками

okrBegin(mas, i, j + k, -2);

}

}

else if (napr == 2) // вниз

{

for (int k = kolPaluba - 1; k >= 1; k--) {

//Помещаем в ячейку число палуб

mas[i + k][j] = kolPaluba;

//Окружаем минус двойками

okrBegin(mas, i + k, j, -2);

}

}

else if (napr == 3) // влево

{

for (int k = kolPaluba - 1; k >= 1; k--) {

//Помещаем в ячейку число палуб

mas[i][j -k] = kolPaluba;

//Окружаем минус двойками

okrBegin(mas, i, j - k, -2);

}

}

break;

}

}

//Конечное окружение

okrEnd(mas);

}

Построение двухпалубных и трехпалубных кораблей происходит по такой же схеме, как и четырехпалубного. Находим свободную ячейку на игровом поле, выбираем одно из четырех направлений случайным образом.Проверяем, что в этом направлении есть свободное место. Если есть, то размещаем остальные палубы корабля, если нет, то ищем новую свободную ячейку. И так до тех пор,пока не удастся построить корабль.

Добавим метод для выстрела игрока, метод будет выполняться в тот момент, когда игрок выберет ячейку для выстрела и нажмет левую кнопку мыши:

// Выстрел игрока

public void vistrelPlay(int i, int j)

{

}

Методу будут передаваться координаты ячейки, которую выбрал пользователь. Тело этого метода мы создадим несколько позднее, а сейчас перейдем в класс pole в файл pole.java (см. рис.2).

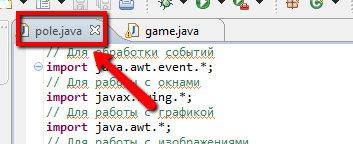


Рис. 2

Управление игрой будет происходить только при помощи мыши, без использования клавиатуры. Поэтому необходимо добавить обработчики событий для мыши. Но в первую очередь, добавим новые переменные класса:

//Класс панели игрового поля

public class pole extends JPanel {

// Таймер отрисовки

private Timer tmDraw;

// Изображения, используемые в игре

private Image fon, paluba, ubit, ranen, end1, end2, bomba;

// Две кнопки

private JButton btn1, btn2;

// Переменная для реализации логики игры

private game myGame;

// Координаты курсора мыши

private int mX, mY;

Мы добавили переменные mX и mY, которые будут хранить координаты курсора мыши. Под блоком объявления переменных добавим классы для работы с мышью, эти два класса получатся вложенными в класс pole:

public class myMouse1 implements MouseListener {

public void mouseClicked(MouseEvent e) {}

// При нажатии кнопки мыши

public void mousePressed(MouseEvent e) {

// Если сделано одиночное нажатие левой клавишей мыши

if ((e.getButton() == 1) && (e.getClickCount() == 1)) {

// Получаем текущие координаты курсора мыши

mX = e.getX();

mY = e.getY();

// Если курсор мыши внутри игрового поля компьютера

if ((mX > 100) && (mY > 100) && (mX < 400) && (mY < 400)) {

// Если не конец игры и ход игрока

if ((myGame.endg == 0) && (myGame.compHod == false)) {

// Вычисляем номер строки в массиве

int i = (mY - 100) / 30;

// Вычисляем номер элемента в строке в массиве

int j = (mX - 100) / 30;

// Если ячейка подходит для выстрела

if (myGame.masComp[i][j] <= 4)

// Производим выстрел

myGame.vistrelPlay(i, j);

}

}

}

}

public void mouseReleased(MouseEvent e) { }

public void mouseEntered(MouseEvent e) { }

public void mouseExited(MouseEvent e) { }

}

public class myMouse2 implements MouseMotionListener {

public void mouseDragged(MouseEvent e) { }

// При перемещении курсора мыши

public void mouseMoved(MouseEvent e) {

// Получаем координаты курсора

mX = e.getX();

mY = e.getY();

// Если курсор в области поля игрока

if ((mX> = 100) && (mY >= 100) && (mX <= 400) && (mY< = 400))

setCursor(new Cursor(Cursor.CROSSHAIR\_CURSOR));

else

setCursor(new Cursor(Cursor.DEFAULT\_CURSOR));

}

}

С этими классами мы уже работали при создании приложения для рисования мышью. В данном случае мы отслеживаем две ситуации: перемещение курсора мыши mouseMoved() и нажатие кнопки мыши mousePressed(). При перемещении курсора мы запоминаем координаты курсора в переменных:

// Получаем координаты курсора

mX = e.getX();

mY = e.getY();

А также изменяем вид курсора на крестик Cursor.CROSSHAIR\_CURSOR

при вхождении курсора в область игрового поля компьютера и меняем его на обычный курсор Cursor.DEFAULT\_CURSOR при выходе из этой области.

При нажатии на кнопку мыши мы делаем проверки: нажата левая клавиша, сделано одно нажатие, курсор находится внутри игрового поля компьютера,игра не закончена, ход принадлежит игроку.После этого вычисляем позицию в массиве:

//Вычисляем номер строки в массиве

int i = (mY- 100) / 30;

//Вычисляем номер элемента в строке в массиве

int j = (mX- 100) / 30;

Проверяем, что в выбранную ячейку можно выполнить выстрел и вызываем метод для выстрела игроком:

if (myGame.masComp[i][j] <= 4)

// Производим выстрел

myGame.vistrelPlay(i, j);

Теперь необходимо подключить обработчики события к панели.Для этого добавим строки в самом начале конструктора класса:

//Конструктор класса

public pole() {

//Подключаем обработчики события для мыши к панели

addMouseListener(new myMouse1());

addMouseMotionListener(new myMouse2());

setFocusable(true); // Передаем фокус панели

Выполним запуск приложения и проверим, что курсор мыши изменяет свой вид при входе на игровое поле компьютера (см. рис. 3).



Рис. 3

При наведении курсора мыши на ячейку для выстрела необходимо выделять ее красным цветом. Чтобы создать этот эффект - перейдем в метод paintComponent(), перед рисованием сетки игрового поля добавим фрагмент кода:

gr.setColor(Color.RED); // Красный цвет

// Если курсор мыши внутри игрового поля компьютера

if ((mX > 100) && (mY > 100) && (mX < 400) && (mY < 400)) {

// Если не конец игры и ход игрока

if ((myGame.endg == 0) && (myGame.compHod == false)) {

// Вычисляем номер строки в массиве

int i = (mY - 100) / 30;

// Вычисляем номер элемента в строке в массиве

int j = (mX - 100) / 30;

// Если ячейка подходит для выстрела

if (myGame.masComp[i][j] <= 4)

// Рисуем квадрат с заливкой

gr.fillRect(100 + j \* 30, 100 + i \* 30, 30, 30);

}

}

Вначале осуществляются проверки, часть из них совпадает с проверками при нажатии клавиши мыши при выстреле. Затем рисуется закрашенный красным цветом квадрат. Выполним запуск приложения и проверим (см.рис. 4).

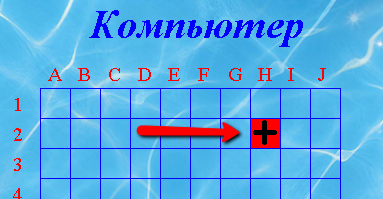


Рис. 4

В самом конце метода paintComponent() добавим программный код для вывода сообщения об окончании игры:

// Вывод изображения конца игры - при окончании игры

if (myGame.endg == 1) // Если победил Игрок

{

gr.drawImage(end1, 300, 200, 300, 100, null);

}

else if (myGame.endg == 2) // Если победил Компьютер

{

gr.drawImage(end2, 300, 200, 300, 100, null);

}

В зависимости от ситуации в конце игры будут выводиться два варианта сообщений (см. рис. 5, 6).



Рис. 5

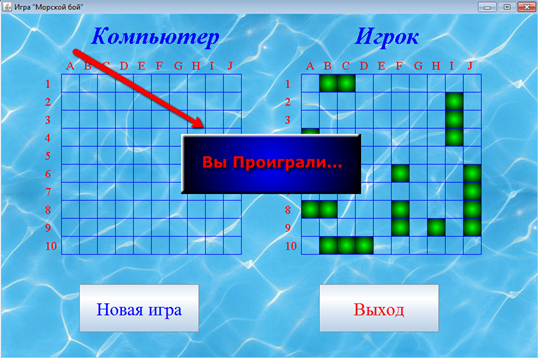


Рис. 6

Осталось сделать последнее в классе pole–добавить вывод палуб кораблей в различных состояниях, а также отрисовку ячеек с выстрелом - пометить бомбой. Для этого необходимо заменить цикл под выведением надписей Компьютер, Игрок на следующий программный код:

// Отрисовка игровых полей Компьютера и Игрока на основании массивов

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

// Игровое поле компьютера

if (myGame.masComp[i][j] != 0) {

// Если это подбитая палуба корабля

if ((myGame.masComp[i][j] >= 8) && (myGame.masComp[i][j] <= 11)) {

gr.drawImage(ranen, 100 + j \* 30, 100 + i \* 30, 30, 30,null);

}

//Если это палуба полностью подбитого корабля

else if (myGame.masComp[i][j] >= 15) {

gr.drawImage(ubit, 100 + j \* 30, 100 + i \* 30, 30, 30,null);

}

// Если был выстрел

if (myGame.masComp[i][j] >= 5) {

gr.drawImage(bomba, 100 + j \* 30, 100 + i \* 30, 30, 30,null);

}

}

// Игровое поле игрока

if (myGame.masPlay[i][j] != 0) {

// Если это палуба корабля

if ((myGame.masPlay[i][j] >= 1) && (myGame.masPlay[i][j] <= 4)) {

gr.drawImage(paluba, 500 + j \* 30, 100 + i \* 30, 30, 30, null);

}

// Если это подбитая палуба корабля

else if ((myGame.masPlay[i][j] >= 8) && (myGame.masPlay[i][j] <= 11)) {

gr.drawImage(ranen, 500 + j \* 30, 100 + i \* 30, 30, 30,null);

}

// Если это палуба полностью подбитого корабля

else if (myGame.masPlay[i][j] >= 15) {

gr.drawImage(ubit, 500 + j \* 30, 100 + i \* 30, 30, 30,null);

}

// Если был выстрел

if (myGame.masPlay[i][j] >= 5) {

gr.drawImage(bomba, 500 + j \* 30, 100 + i \* 30, 30, 30,null);

}

}

}

}

Ранее этот блок рисовал только палубы кораблей на поле игрока.Теперь в него добавлено рисование поля компьютера и выведение всех возможных состояний ячейки игрового поля. Циклы for работают, как и прежде.

Генерация кораблей реализована в классе game, при запуске игры можно видеть сгенерированные корабли на поле игрока, корабли компьютера также генерируются, но не показываются (см. рис. 7). При каждом нажатии на кнопку Новая игра–корабли выстраиваются случайным образом по другой схеме.



Рис. 7

Класс pole для третьего уровня сложности полностью закончен. Все оставшиеся доработки будут в классе game. Полный программный код класса pole выглядит так:

поля

public class pole extends JPanel {

// Таймер отрисовки

private Timer tmDraw;

// Изображения, используемые в игре

private Image fon, paluba, ubit, ranen, end1, end2, bomba;

// Две кнопки

private JButton btn1, btn2;

// Переменная для реализации логики игры

private game myGame;// Для обработки событий

import java.awt.event.\*;

// Для работы с окнами

import javax.swing.\*;

// Для работы с графикой

import java.awt.\*;

// Для работы с изображениями

import javax.imageio.\*;

// Для работы с файлами

import java.io.\*;

//Класс панели игрового

//Координаты курсора мыши

private int mX, mY;

public class myMouse1 implements MouseListener {

public void mouseClicked(MouseEvent e) {}

// При нажатии кнопки мыши

public void mousePressed(MouseEvent e) {

// Если сделано одиночное нажатие левой клавишей мыши

if ((e.getButton() == 1) && (e.getClickCount() == 1)) {

// Получаем текущие координаты курсора мыши

mX = e.getX();

mY = e.getY();

// Если курсор мыши внутри игрового поля компьютера

if ((mX > 100)&& (mY > 100) && (mX < 400) && (mY < 400)) {

// Если не конец игры и ход игрока

if ((myGame.endg == 0) && (myGame.compHod == false)) {

// Вычисляем номер строки в массиве

int i = (mY - 100) / 30;

//Вычисляем номер элемента в строке в массиве

int j = (mX - 100) / 30;

// Если ячейка подходит для выстрела

if (myGame.masComp[i][j] <= 4)

// Производим выстрел

myGame.vistrelPlay(i, j);

}

}

}

}

public void mouseReleased(MouseEvent e) { }

public void mouseEntered(MouseEvent e) { }

public void mouseExited(MouseEvent e) { }

}

public class myMouse2 implements MouseMotionListener {

public void mouseDragged(MouseEvent e) { }

// При перемещении курсора мыши

public void mouseMoved(MouseEvent e) {

// Получаем координаты курсора

mX = e.getX();

mY = e.getY();

// Если курсор в области поля игрока

if ((mX >= 100) && (mY >= 100) && (mX <= 400) && (mY <= 400))

setCursor(new Cursor(Cursor.CROSSHAIR\_CURSOR));

else

setCursor(new Cursor(Cursor.DEFAULT\_CURSOR));

}

}

// Конструктор класса

public pole() {

//Подключаем обработчики события для мыши к панели

addMouseListener(new myMouse1());

addMouseMotionListener(new myMouse2());

setFocusable(true); // Передаем фокус панели

//Создаем объект новой игры

myGame = new game();

//Запускаем игру

myGame.start();

//Попытка загрузки всех изображений для игры

try {

fon = ImageIO.read(new File("c:\\fon.png"));

paluba = ImageIO.read(new File("c:\\paluba.png"));

ranen = ImageIO.read(new File("c:\\ranen.png"));

ubit = ImageIO.read(new File("c:\\ubit.png"));

end1 = ImageIO.read(new File("c:\\end1.png"));

end2 = ImageIO.read(new File("c:\\end2.png"));

bomba = ImageIO.read(new File("c:\\bomba.png"));

} catch (Exception ex) {

}

//Создаем, настраиваем и запускаем таймер

// для отрисовки игрового поля

tmDraw = new Timer(50, new ActionListener() {

@Override

public void actionPerformed(ActionEvent arg0) {

// Вызываем перерисовку -paintComponent()

repaint();

}

});

tmDraw.start();

//Включаем возможность произвольного размещения

//элементов интерфейса на панели

setLayout(null);

//Создаем кнопку Новая игра

btn1 = new JButton();

btn1.setText("Новая игра");

btn1.setForeground(Color.BLUE);

btn1.setFont(new Font("serif", 0, 30));

btn1.setBounds(130, 450, 200, 80);

btn1.addActionListener(new ActionListener() {

// Обработчик события при нажатии на кнопку Новая игра

public void actionPerformed(ActionEvent arg0) {

// Запуск - начало игры

myGame.start();

}

});

add(btn1);

//Создаем кнопку Выход

btn2 = new JButton();

btn2.setText("Выход");

btn2.setForeground(Color.RED);

btn2.setFont(new Font("serif", 0, 30));

btn2.setBounds(530, 450, 200, 80);

btn2.addActionListener(new ActionListener() {

// Обработчик события при нажатии на кнопку Новая игра

public void actionPerformed(ActionEvent arg0) {

// Выход их игры -завершение работы приложения

System.exit(0);

}

});

add(btn2);

}

// Метод отрисовки

public void paintComponent(Graphics gr) {

//Очищение игрового поля

super.paintComponent(gr);

//Отрисовка фона

gr.drawImage(fon, 0, 0, 900, 600, null);

//Установка шрифта

gr.setFont(new Font("serif", 3, 40));

//Установка цвета

gr.setColor(Color.BLUE);

//Выведение надписей

gr.drawString("Компьютер", 150, 50);

gr.drawString("Игрок", 590, 50);

//Отрисовка игровых полей Компьютера

// и Игрока на основании массивов

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

// Игровое поле компьютера

if (myGame.masComp[i][j] != 0) {

// Если это подбитая палуба корабля

if ((myGame.masComp[i][j] >= 8) && (myGame.masComp[i][j] <= 11)) {

gr.drawImage(ranen, 100 + j \* 30, 100 + i \* 30, 30, 30,null);

}

// Если это палуба полностью подбитого корабля

else if (myGame.masComp[i][j] >= 15) {

gr.drawImage(ubit, 100 + j \* 30, 100 + i \* 30, 30, 30,null);

}

// Если был выстрел

if (myGame.masComp[i][j] >= 5) {

gr.drawImage(bomba, 100 + j \* 30, 100 + i \* 30, 30, 30,null);

}

}

// Игровое поле игрока

if (myGame.masPlay[i][j] != 0) {

// Если это палуба корабля

if ((myGame.masPlay[i][j] >= 1) && (myGame.masPlay[i][j] <= 4)) {

gr.drawImage(paluba, 500 + j \* 30, 100 + i \* 30, 30, 30, null);

}

// Если это подбитая палуба корабля

else if ((myGame.masPlay[i][j] >= 8) && (myGame.masPlay[i][j] <= 11)) {

gr.drawImage(ranen, 500 + j \* 30, 100 + i \* 30, 30, 30,null);

}

// Если это палуба полностью подбитого корабля

else if (myGame.masPlay[i][j] >= 15) {

gr.drawImage(ubit, 500 + j \* 30, 100 + i \* 30, 30, 30,null);

}

// Если был выстрел

if (myGame.masPlay[i][j] >= 5) {

gr.drawImage(bomba, 500 + j \* 30, 100 + i \* 30, 30, 30,null);

}

}

}

}

gr.setColor(Color.RED); // Красный цвет

// Если курсор мыши внутри игрового поля компьютера

if ((mX > 100) && (mY > 100) && (mX < 400) && (mY < 400)) {

// Если не конец игры и ход игрока

if ((myGame.endg == 0) && (myGame.compHod == false)) {

// Вычисляем номер строки в массиве

int i = (mY - 100) / 30;

// Вычисляем номер элемента в строке в массиве

int j = (mX - 100) / 30;

// Если ячйека подходит для выстрела

if (myGame.masComp[i][j] <= 4)

// Рисуем квадрат с заливкой

gr.fillRect(100 + j \* 30, 100 + i \* 30, 30, 30);

}

}

//Отрисовка сетки игрового поля из синих линий

gr.setColor(Color.BLUE);

for (int i = 0; i <= 10; i++) {

// Рисование линий сетки игрового поля Компьютера

gr.drawLine(100 + i \* 30, 100, 100 + i \* 30, 400);

gr.drawLine(100, 100 + i \* 30, 400, 100 + i \* 30);

// Рисование линий сетки игрового поля Человека

gr.drawLine(500 + i \* 30, 100, 500 + i \* 30, 400);

gr.drawLine(500, 100 + i \* 30, 800, 100 + i \* 30);

}

//Установка шрифта

gr.setFont(new Font("serif", 0, 20));

//Установка цвета

gr.setColor(Color.RED);

//Введение цифр и букв слева и сверху от игровых полей

for (int i = 1; i <= 10; i++) {

// Вывод цифр

gr.drawString("" + i, 73, 93 + i \* 30);

gr.drawString("" + i, 473, 93 + i \* 30);

// Вывод букв

gr.drawString("" + (char) ('A' + i - 1), 78 + i \* 30, 93);

gr.drawString("" + (char) ('A' + i - 1), 478 + i \* 30, 93);

}

//Вывод изображения конца игры - при окончании игры

if (myGame.endg == 1) // Если победил Игрок

{

gr.drawImage(end1, 300, 200, 300, 100, null);

}

else if (myGame.endg == 2) // Если победил Компьютер

{

gr.drawImage(end2, 300, 200, 300, 100, null);

}

}

}

Перейдем в класс gameв файл game.java (см. рис. 8).

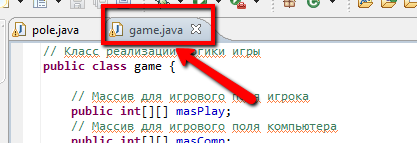


Рис. 8

Все, что касается графики уже выполнено. Также запрограммировано управление игрой при помощи мыши. Это все выполняется в классеpole. Но еще много работы нужно произвести, чтобы игра была полностью закончена. И эти работы будут производиться в классе gameв отношении двумерных массивов.

Давайте рассмотрим числовые обозначения, которые будут использоваться в процессе игры между компьютером и игроком.

· Число 1:палуба однопалубного корабля

· Число 2:палуба двухпалубного корабля

· Число 3:палуба трехпалубного корабля

· Число 4:палуба четырехпалубного корабля

· Число -1:пространство вокруг корабля

· При выстреле в ячейку будем прибавлять к ее значению число 7

· При полном уничтожении корабля будем прибавлять к значению каждой палубы еще одно число 7

Таким образом, после выстрела в любую ячейку ее значение может быть в пределах от 6 до 11 включительно. От 8до 11включительно –это палубы раненых кораблей. Число 6–попадание в пространство рядом с кораблем, число 7–попадание в пустую ячейку (там где изначально стоял нуль).

Когда корабль будет полностью "убит" - к значениям его раненых палуб будет прибавляться еще число 7, поэтому их значения станут от 15 до18включительно.

После того, как корабль будет полностью "убит" –значение ячеек вокруг мы будем уменьшать на единицу.

Добавим метод проверки окончания игры, после каждого хода компьютера и игрока. Мы будем вызывать этот метод для проверки окончания игры:

// Проверка окончания игры

private void testEndGame()

{

//Тестовое число = 15\*4+16\*2\*3+17\*3\*2+18\*4

//Ситуация, когда все корабли убиты

int testNumber = 330;

int kolComp=0; // Сумма убитых палуб компьютера

int kolPlay=0; // Сумма убитых палуб игрока

//Перебираем все элементы сразу двух массивов

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

// Суммируем подбитые палубы игрока

if (masPlay[i][j] >= 15) kolPlay += masPlay[i][j];

// Суммируем подбитые палубы компьютера

if (masComp[i][j] >= 15) kolComp += masComp[i][j];

}

}

if (kolPlay == testNumber) endg=2; // Если победил игрок

else if (kolComp == testNumber) endg=1; // Если победил компьютер

}

Значения палуб убитых кораблей будут находиться в пределах от15до 18включительно. Это значит, что при подсчете суммы всех значений в этом интервале мы можем определить –закончилась игра или еще нет. Когда сумма станет равна 330–игра закончилась. Необходимо подсчитывать сразу две суммы –у компьютера и у игрока. Кто первый наберет эту сумму, тот и выиграл. Например:палуба убитого четырехпалубного корабля имеет значение 18, всего на игровом поле их четыре. Для всех кораблей сумма будет вычисляться так:15\*4+16\*2\*3+17\*3\*2+18\*4 = 330.

После того как корабль"убит" –значение ячеек вокруг будем уменьшать на единицу, чтобы отличать это пространство от еще не убитых кораблей. Рассмотрим пример этого пространства до уменьшения на единицу (см. рис. 9).

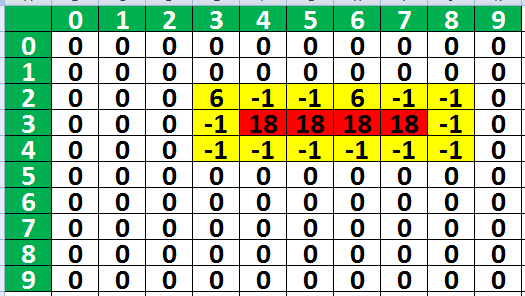


Рис. 9

Числом 18отмечены палубы четырехпалубного полностью убитого корабля.

Числом - минус единица-1 отмечено пространство вокруг корабля, как это и было с момента генерации кораблей. Числом6 отмечено пространство, в которое попало выстрелом,при выстреле мы прибавляем число 7 (-1+7=6).

Теперь посмотрим, что мы должны получить после уменьшения на единицу

(см. рис. 10). В результате все минус единицы-1 стали минус двойками, а все числа шесть 6превратились в числа пять 5. Добавим для начала метод который будет выполнять такое изменение в одной ячейке:

// Установить один элемент окружения подбитого корабля

private void setOkrPodbit(int[][] mas, int i, int j) {

// Если не происходит выход за пределы массива

// и в ячейке нулевое значение

if (testMasPoz(i, j)==true)

{

//Устанавливаем необходимое значение

if ((mas[i][j]==-1)||(mas[i][j]==6)) mas[i][j]--;

}

}

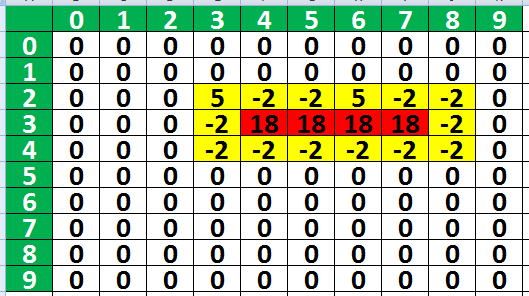


Рис. 10

Теперь добавим метод,который выполнит окружение по описанной схеме вокруг:

// Окружение одной ячейки подбитого вокруг

private void okrPodbit(int[][] mas, int i, int j) {

setOkrPodbit(mas, i - 1, j - 1); // сверху слева

setOkrPodbit(mas, i - 1, j); // сверху

setOkrPodbit(mas, i - 1, j + 1); // сверху справа

setOkrPodbit(mas, i, j + 1); // справа

setOkrPodbit(mas, i + 1, j + 1); // снизу справа

setOkrPodbit(mas, i + 1, j); // снизу

setOkrPodbit(mas, i + 1, j - 1); // снизу слева

setOkrPodbit(mas, i, j - 1); // слева

}

При программировании ходов основная сложность заключается в ходах компьютера. Компьютер хранит данные игрового поля соперника, но "не подглядывает" в них–играет честно! Игрок сам выбирает свой ход и сам производит анализ.

Схема ходов компьютера определяется программистом. Можно разрабатывать различные схемы. Мы будем использовать схему, которая дает некоторое преимущество игроку. Если создать игру, в которую всегда будет выигрывать компьютер,то вряд ли такая игра будет интересна игроку с компьютером. Если игрок будет выигрывать несколько чаще, чем компьютер –такой вариант порадует игрока. Побеждать всегда приятно! С другой стороны игрок должен понимать, что компьютер также прилагает усилия. И если расслабиться, то можно и проиграть.

1. Первый ход будет всегда принадлежать игроку.

2. Если на поле игрока нет раненых палуб, то компьютер выбирает любую случайную ячейку и делает выстрел в нее.При выстреле компьютер не делает выстрелов в пространство рядом с кораблем. Так как известно, что там не могут располагаться палубы других кораблей.

3. Если на поле игрока есть раненые палубы, то компьютер "смотрит" четыре ячейки вокруг этой палубы и выбирает одну из них, в которую можно сделать выстрел. При ранении корабля компьютер стреляет в него до полного уничтожения. Компьютер не отслеживает направление расположения корабля –тем самым, иногда, делает лишние ходы. Это и есть небольшое "подыгрывание"сопернику.

При выборе случайной ячейки для выстрела мы будем выполнять 100 попыток поиска такой ячейки случайным образом. Если за 100 попыток не удается найти, то начинаем пробегать массив от начала до конца и берем первую подходящую ячейку для выстрела. После каждого выстрела мы будем проверять убит корабль или еще нет. Добавим метод, который выполняет выстрел компьютера:

//Выстрел компьютера -

// возвращает истину - если попал

private boolean compHodit()

{

//Признак попадания в цель

boolean rez = false;

//Признак выстрела в раненый

//корабль

boolean flag = false;

\_for1:

//Пробегаем все игровое поле игрока

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

// Если находим раненую палубу

if ((masPlay[i][j]>=9)&&(masPlay[i][j]<=11))

{

flag = true;

// ячейка сверху

// Проверяем, что можно сделать выстрел

if (testMasPoz(i-1, j)&&(masPlay[i-1][j]<=4)&&(masPlay[i-1][j] !=-2))

{

//делаем выстрел

masPlay[i-1][j] += 7;

//проверяем, что убит

testUbit(masPlay, i-1, j);

// если произошло попадание

if (masPlay[i-1][j]>=8) rez = true;

//прерываем сразу все циклы

break \_for1;

}

// ячейка снизу

// Проверяем, что можно сделать выстрел

else if (testMasPoz(i+1, j)&&(masPlay[i+1][j]<=4)&&(masPlay[i+1][j] !=-2))

{

//делаем выстрел

masPlay[i+1][j] += 7;

//проверяем, что убит

testUbit(masPlay, i+1, j);

// если произошло попадание

if (masPlay[i+1][j]>=8) rez = true;

//прерываем сразу все циклы

break \_for1;

}

// ячейка слева

// Проверяем, что можно сделать выстрел

if (testMasPoz(i, j-1)&&(masPlay[i][j-1]<=4)&&(masPlay[i][j-1] !=-2))

{

//делаем выстрел

masPlay[i][j-1] += 7;

//проверяем, что убит

testUbit(masPlay, i, j-1);

// если произошло попадание

if (masPlay[i][j-1]>=8) rez = true;

//прерываем сразу все циклы

break \_for1;

}

// ячейка справа

// Проверяем, что можно сделать выстрел

else if (testMasPoz(i, j+1)&&(masPlay[i][j+1]<=4)&&(masPlay[i][j+1] !=-2))

{

//делаем выстрел

masPlay[i][j+1] += 7;

//проверяем, что убит

testUbit(masPlay, i, j+1);

// если произошло попадание

if (masPlay[i][j+1]>=8) rez = true;

//прерываем сразу все циклы

break \_for1;

}

}

}

}

// если не было выстрела в раненую палубу

if (flag == false) {

// делаем 100случайных попыток выстрела

// в случайное место

for (int l = 1; l <= 100; l++) {

// Находим случайную позицию на игровом поле

int i = (int) (Math.random() \* 10);

int j = (int) (Math.random() \* 10);

// Проверяем, что можно сделать выстрел

if ((masPlay[i][j] <= 4) && (masPlay[i][j] != -2)) {

// делаем выстрел

masPlay[i][j] += 7;

// проверяем, что убит

testUbit(masPlay, i, j);

// если произошло попадание

if (masPlay[i][j] >= 8)

rez = true;

// выстрел произошел

flag = true;

// прерываем цикл

break;

}

}

// если выстрела еще не было

if (flag == false) {

//начинаем пробегать весь массив от начала до конца

\_for2: for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

// Проверяем, что можно сделать выстрел

if ((masPlay[i][j] <= 4)&& (masPlay[i][j] != -2)) {

// делаем выстрел

masPlay[i][j] += 7;

// проверяем, что убит

testUbit(masPlay, i, j);

// если произошло попадание

if (masPlay[i][j] >= 8)

rez = true;

// прерываем сразу все циклы

break \_for2;

}

}

}

}

}

//проверяем конец игры

testEndGame();

//возвращаем результат

return rez;

В классе game имеется метод для выстрела игрока,добавим его программный код:

//Выстрел игрока

public void vistrelPlay(int i, int j)

{

// При выстреле прибавляем число 7

masComp[i][j] += 7;

//Проверяем убит ли корабль

testUbit(masComp, i, j);

//Проверяем конец игры

testEndGame();

// Если был промах - передаем ход компьютеру

if (masComp[i][j]<8)

{

compHod=true; // передаем ход компьютеру

// Ходит компьютер - пока попадает в цель

while (compHod==true) compHod = compHodit();

}

}

В методах для ходов игрока и компьютера был использован метод проверки после выстрела на признак полностью подбитого корабля. Дальше нам необходимо реализовать этот метод:

// Проверка убит ли корабль

private void testUbit(int[][] mas, int i, int j)

{

//Если однопалубный

if (mas[i][j]==8)

{

// делаем выстрел

mas[i][j] += 7;

// окружаем убитый корабль

okrPodbit(mas, i, j);

}

// Если двухпалубный

else if (mas[i][j]==9) analizUbit(mas, i, j, 2);

// Если трехпалубный

else if (mas[i][j]==10) analizUbit(mas, i, j, 3);

// Если четырехпалубный

else if (mas[i][j]==11) analizUbit(mas, i, j, 4);

}

Метод testUbit() вызывает методanalizUbit() для разных видов кораблей.

Осталось добавить код последнего метода analizUbit():

// Анализ убитого корабля

private void analizUbit(int[][] mas, int i, int j, int kolPalub)

{

//Количество раненых палуб

int kolRanen=0;

//Выполняем подсчет раненых палуб

for (int k=i-(kolPalub-1);k<=i+(kolPalub-1);k++)

{

for (int g=j-(kolPalub-1);g<=j+(kolPalub-1);g++)

{

// Если это палуба раненого корабля

if (testMasPoz(k, g)&&(mas[k][g]==kolPalub+7)) kolRanen++;

}

}

// Если количество раненых палуб совпадает с количеством палуб

//корабля, то он убит - прибавляем число7

if (kolRanen==kolPalub)

{

for (int k=i-(kolPalub-1);k<=i+(kolPalub-1);k++)

{

for (int g=j-(kolPalub-1);g<=j+(kolPalub-1);g++)

{

// Если это палуба раненого корабля

if (testMasPoz(k, g)&&(mas[k][g]==kolPalub+7))

{

// помечаем палубой убитого корабля

mas[k][g]+=7;

// окружаем палубу убитого корабля

okrPodbit(mas, k, g);

}

}

}

}

}

Рассмотрим алгоритм тестирования убитого корабля. После того как происходит выстрел в ячейку с попаданием –нам нужно проверять –убит корабль или еще нет. Если убит,то прибавлять число семь 7 ко всем его палубам, чтобы отметить как убитый, а также отметить пространство вокруг него.

Допустим, был сделан успешный выстрел в палубу четырехпалубного корабля (см. рис. 11). Значит, в этой ячейке будет число 11. Мы начинаем просматривать все ячейки вокруг и подсчитывать количество подбитых палуб четырехпалубного корабля. Если мы насчитываем их четыре –это значит, что корабль убит. Если корабль убит, то мы еще раз проходим пространство вокруг и ко всем ячейкам, которые содержат число 11,прибавляем число 7. В результате во всех этих ячейках получится число 18 (см. рис. 12). Если корабль еще не убит, то мы ничего не делаем.

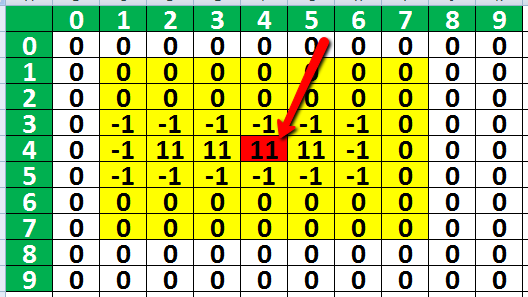


Рис. 11

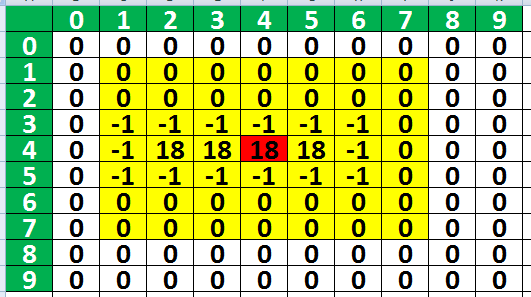


Рис. 12

После отметки палуб корабля как убитого, мы вычитаем единицу вокруг него (см. рис. 13). Для этих целей был создан специальный метод okrPodbit(). Когда происходит обход области вокруг проверяемой палубы, то она выбирается по следующему принципу. Мы берем отступ вверх,вниз, вправо, влево по три ячейки для четырех палубного корабля (см. рис. 14). Так как другие палубы одного корабля не могут быть дальше. Такой же подход используется для двухпалубных и трехпалубных кораблей.Может возникнуть вопрос: а не сможем ли мы случайно захватить палубы соседних кораблей? Ответ на этот вопрос – нет, не можем. Потому что компьютер стреляет в корабль до тех пор, пока полностью не подобьет его.Другими словами, на игровом поле всегда один корабль с ранеными палубами.

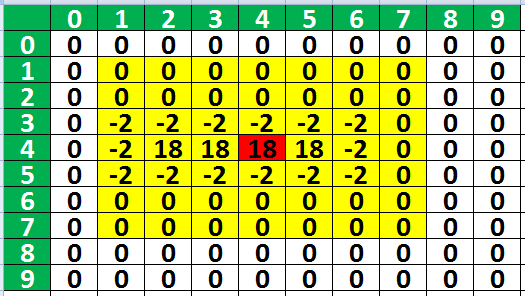


Рис. 13

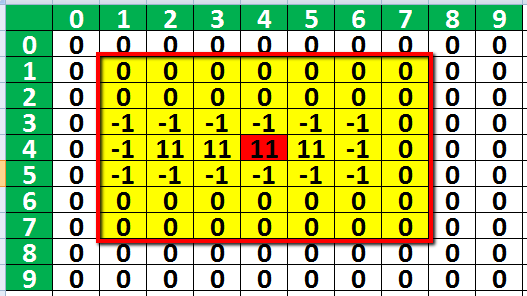


Рис. 14

Теперь наша игра для третьего уровня сложности закончена, а значит, завершена полностью. Сохраним проект в JAR-архив. Запустим наше приложение и поиграем! (см. рис. 15).

Внимание!!!JAR-архив игры Морской бой создается стандартным способом. Пример создания JAR-архива был рассмотрен при рассмотрении игры Змейка.



Рис. 15

Полный программный код класса game выглядит так:

// Класс реализации логики игры

public class game {

// Массив для игрового поля игрока

public int[][] masPlay;

// Массив для игрового поля компьютера

public int[][] masComp;

// Признак хода компьютера (false - ходит игрок)

public boolean compHod;

// Признак конца игры

// (0-игра идет, 1-победил игрок,2-победил компьютер)

public int endg;

// Конструктор класса

public game() {

//Создаем массив 10x10 - игровое поле игрока

masPlay = new int[10][10];

//Создаем массив 10x10 - игровое поле компьютера

masComp = new int[10][10];

}

// Запуск игры - начало игры

public void start() {

//Очищаем игровое поле игрока и компьютера

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

masPlay[i][j] = 0;

masComp[i][j] = 0;

}

}

//Обнуляем признак чьей-то победы

endg = 0;

//Передаем первый ход игроку

compHod = false;

//Расставляем корабли игрока

rasstanovkaKorabley(masPlay);

//Расставляем корабли компьютера

rasstanovkaKorabley(masComp);

}

// Анализ убитого корабля

private void analizUbit(int[][] mas, int i, int j, int kolPalub)

{

//Количество раненых палуб

int kolRanen=0;

//Выполняем подсчет раненых палуб

for (int k=i-(kolPalub-1);k<=i+(kolPalub-1);k++)

{

for (int g=j-(kolPalub-1);g<=j+(kolPalub-1);g++)

{

// Если это палуба раненого корабля

if (testMasPoz(k, g)&&(mas[k][g]==kolPalub+7)) kolRanen++;

}

}

// Если количество раненых палуб совпадает с количеством палуб

// корабля, то он убит -прибавляем число 7

if (kolRanen==kolPalub)

{

for (int k=i-(kolPalub-1);k<=i+(kolPalub-1);k++)

{

for (int g=j-(kolPalub-1);g<=j+(kolPalub-1);g++)

{

// Если это палуба раненого корабля

if (testMasPoz(k, g)&&(mas[k][g]==kolPalub+7))

{

// помечаем палубой убитого корабля

mas[k][g]+=7;

// окружаем палубу убитого корабля

okrPodbit(mas, k, g);

}

}

}

}

}

// Проверка убит ли корабль

private void testUbit(int[][] mas, int i, int j)

{

// Если однопалубный

if (mas[i][j]==8)

{

// делаем выстрел

mas[i][j] += 7;

// окружаем убитый корабль

okrPodbit(mas, i, j);

}

// Если двухпалубный

else if (mas[i][j]==9) analizUbit(mas, i, j, 2);

// Если трехпалубный

else if (mas[i][j]==10) analizUbit(mas, i, j, 3);

// Если четырехпалубный

else if (mas[i][j]==11) analizUbit(mas, i, j, 4);

}

// Расстановка кораблей

private void rasstanovkaKorabley(int[][] mas) {

//Создаем один четырехпалубный корабль

make4P(mas, 4);

//Создаем два трехпалубных корабля

for (int i = 1; i <= 2; i++)

make4P(mas, 3);

//Создаем три двухпалубных корабля

for (int i = 1; i <= 3; i++)

make4P(mas, 2);

//Создаем четыре однопалубных корабля

make1P(mas);

}

// Выстрел игрока

public void vistrelPlay(int i, int j) {

// При выстреле прибавляем число 7

masComp[i][j] += 7;

//Проверяем убит ли корабль

testUbit(masComp, i, j);

//Проверяем конец игры

testEndGame();

// Если был промах - передаем ход компьютеру

if (masComp[i][j] < 8) {

compHod = true; //передаем ход компьютеру

// Ходит компьютер- пока попадает в цель

while (compHod == true)

compHod = compHodit();

}

}

// Выстрел компьютера -

//возвращает истину - если попал

private boolean compHodit()

{

// Признак попадания в цель

boolean rez = false;

// Признак выстрела в раненый

// корабль

boolean flag = false;

\_for1:

// Пробегаем все игровое поле игрока

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

// Если находим раненую палубу

if ((masPlay[i][j]>=9)&&(masPlay[i][j]<=11))

{

flag = true;

//ячейка сверху

// Проверяем, что можно сделать выстрел

if (testMasPoz(i-1, j)&&(masPlay[i-1][j]<=4)&&(masPlay[i-1][j] !=-2))

{

// делаем выстрел

masPlay[i-1][j] += 7;

// проверяем, что убит

testUbit(masPlay, i-1, j);

// если произошло попадание

if (masPlay[i-1][j]>=8) rez = true;

// прерываем сразу все циклы

break \_for1;

}

//ячейка снизу

//Проверяем, что можно сделать выстрел

else if (testMasPoz(i+1, j)&&(masPlay[i+1][j]<=4)&&(masPlay[i+1][j] !=-2))

{

// делаем выстрел

masPlay[i+1][j] += 7;

// проверяем, что убит

testUbit(masPlay, i+1, j);

// если произошло попадание

if (masPlay[i+1][j]>=8) rez = true;

// прерываем сразу все циклы

break \_for1;

}

// ячейка слева

// Проверяем, что можно сделать выстрел

if (testMasPoz(i, j-1)&&(masPlay[i][j-1]<=4)&&(masPlay[i][j-1] !=-2))

{

// делаем выстрел

masPlay[i][j-1] += 7;

// проверяем, что убит

testUbit(masPlay, i, j-1);

// если произошло попадание

if (masPlay[i][j-1]>=8) rez = true;

// прерываем сразу все циклы

break \_for1;

}

// ячейка справа

// Проверяем, что можно сделать выстрел

else if (testMasPoz(i, j+1)&&(masPlay[i][j+1]<=4)&&(masPlay[i][j+1] !=-2))

{

//делаем выстрел

masPlay[i][j+1] += 7;

//проверяем, что убит

testUbit(masPlay, i, j+1);

// если произошло попадание

if (masPlay[i][j+1]>=8) rez = true;

//прерываем сразу все циклы

break \_for1;

}

}

}

}

// если не было выстрела в раненую палубу

if (flag == false) {

// делаем 100 случайных попыток выстрела

// в случайное место

for (int l = 1; l <= 100; l++) {

// Находим случайную позицию на игровом поле

int i = (int) (Math.random() \* 10);

int j = (int) (Math.random() \* 10);

//Проверяем, что можно сделать выстрел

if ((masPlay[i][j] <= 4) && (masPlay[i][j] != -2)) {

//делаем выстрел

masPlay[i][j] += 7;

//проверяем, что убит

testUbit(masPlay, i, j);

// если произошло попадание

if (masPlay[i][j] >= 8)

rez = true;

//выстрел произошел

flag = true;

//прерываем цикл

break;

}

}

// если выстрела еще не было

if (flag == false) {

// начинаем пробегать весь массив от начала до конца

\_for2: for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

// Проверяем, что можно сделать выстрел

if ((masPlay[i][j] <= 4) && (masPlay[i][j] != -2)) {

//делаем выстрел

masPlay[i][j] += 7;

//проверяем, что убит

testUbit(masPlay, i, j);

// если произошло попадание

if (masPlay[i][j] >= 8)

rez = true;

//прерываем сразу все циклы

break \_for2;

}

}

}

}

}

// проверяем конец игры

testEndGame();

// возвращаем результат

return rez;

}

// Установить один элемент окружения подбитого корабля

private void setOkrPodbit(int[][] mas, int i, int j) {

// Если не происходит выход за пределы массива

// и в ячейке нулевое значение

if (testMasPoz(i, j)==true)

{

//Устанавливаем необходимое значение

if ((mas[i][j]==-1)||(mas[i][j]==6)) mas[i][j]--;

}

}

// Окружение одной ячейки подбитого вокруг

private void okrPodbit(int[][] mas, int i, int j) {

setOkrPodbit(mas, i - 1, j - 1); // сверху слева

setOkrPodbit(mas, i - 1, j); // сверху

setOkrPodbit(mas, i - 1, j + 1); // сверху справа

setOkrPodbit(mas, i, j + 1); // справа

setOkrPodbit(mas, i + 1, j + 1); // снизу справа

setOkrPodbit(mas, i + 1, j); // снизу

setOkrPodbit(mas, i + 1, j - 1); // снизу слева

setOkrPodbit(mas, i, j - 1); // слева

}

// Проверка окончания игры

private void testEndGame()

{

//Тестовое число = 15\*4+16\*2\*3+17\*3\*2+18\*4

//Ситуация, когда все корабли убиты

int testNumber = 330;

int kolComp=0; // Сумма убитых палуб компьютера

int kolPlay=0; // Сумма убитых палуб игрока

//Перебираем все элементы сразу двух массивов

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

// Суммируем подбитые палубы игрока

if (masPlay[i][j] >= 15) kolPlay += masPlay[i][j];

// Суммируем подбитые палубы компьютера

if (masComp[i][j] >= 15) kolComp += masComp[i][j];

}

}

if (kolPlay == testNumber) endg=2; // Если победил игрок

else if (kolComp == testNumber) endg=1; // Если победил компьютер

}

// Проверка не выхода за границы массива

private boolean testMasPoz(int i, int j) {

if (((i> = 0) && (i <= 9)) && ((j >= 0) && (j <= 9))) {

return true;

} else

return false;

}

// Запись значения в массив с проверкой границ массива

private void setMasValue(int[][] mas, int i, int j, int val) {

// Если не происходит выход за границы массива

if (testMasPoz(i, j) == true) {

// Записываем значение в массив

mas[i][j] = val;

}

}

// Установить один элемент окружения

private void setOkr(int[][] mas, int i, int j, int val) {

// Если не происходит выход за пределы массива

// и в ячейке нулевое значение

if (testMasPoz(i, j) && (mas[i][j] == 0))

// Устанавливаем необходимое значение

setMasValue(mas, i, j, val);

}

// Окружение одной ячейки вокруг

private void okrBegin(int[][] mas, int i, int j, int val) {

setOkr(mas, i - 1, j - 1, val); // сверху слева

setOkr(mas, i - 1, j, val); // сверху

setOkr(mas, i - 1, j + 1, val); // сверху справа

setOkr(mas, i, j + 1, val); // справа

setOkr(mas, i + 1, j + 1, val); // снизу справа

setOkr(mas, i + 1, j, val); // снизу

setOkr(mas, i + 1, j - 1, val); // снизу слева

setOkr(mas, i, j - 1, val); // слева

}

// Конечное окружение

private void okrEnd(int[][] mas) {

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

// Если значение элемента массива -2, то заменяем его на -1

if (mas[i][j] == -2)

mas[i][j] = -1;

}

}

}

// Создание четырех однопалубных кораблей

private void make1P(int[][] mas) {

// Циклfor делает четыре шага - для четырех кораблей

for (int k = 1; k <= 4; k++) {

// Глухой циклwhile

while (true) {

// Находим случайную позицию на игровом поле

int i = (int) (Math.random() \* 10);

int j = (int) (Math.random() \* 10);

//Проверяем, что там ничего нет и можно разместить корабль

if (mas[i][j] == 0) {

// Размещаем однопалубный корабль

mas[i][j] = 1;

// Выполняем окружение

okrBegin(mas, i, j, -1);

// Прерываем цикл

break;

}

}

}

}

// Проверка ячейки для возможности размещения в ней палубы корабля

private boolean testNewPaluba(int[][] mas, int i, int j) {

// Если выход за границы массива

if (testMasPoz(i, j) == false)

return false;

// Если в этой ячейке 0 или -2, то она нам подходит

if ((mas[i][j] == 0) || (mas[i][j] == -2))

return true;

return false;

}

// Создание корабля с несколькими палубами от 2-х до 4-х

private void make4P(int[][] mas, int kolPaluba) {

//Глухой цикл

while (true) {

boolean flag = false;

// Координаты головы корабля

int i = 0, j = 0;

// Создание первой палубы - головы корабля

// Получение случайной строки

i = (int) (Math.random() \* 10);

// Получение случайной колонки

j = (int) (Math.random() \* 10);

// Выбираем случайное направление построения корабля

// 0 - вверх, 1 -вправо, 2 - вниз, 3 - влево

int napr = (int) (Math.random() \* 4);

if (testNewPaluba(mas, i, j) == true) {

if (napr == 0) // вверх

{

// Если можно расположить палубу

if (testNewPaluba(mas, i -(kolPaluba - 1), j) == true)

flag = true;

}

else if (napr == 1) // вправо

{

// Если можно расположить палубу

if (testNewPaluba(mas, i, j + (kolPaluba - 1)) == true)

flag = true;

}

else if (napr == 2) // вниз

{

// Если можно расположить палубу

if (testNewPaluba(mas, i + (kolPaluba - 1), j) == true)

flag = true;

}

else if (napr == 3) // влево

{

// Если можно расположить палубу

if (testNewPaluba(mas, i, j -(kolPaluba - 1)) == true)

flag = true;

}

}

if (flag == true) {

// Помещаем в ячейку число палуб

mas[i][j] = kolPaluba;

// Окружаем минус двойками

okrBegin(mas, i, j, -2);

if (napr == 0) // вверх

{

for (int k = kolPaluba - 1; k >= 1; k--) {

//Помещаем в ячейку число палуб

mas[i -k][j] = kolPaluba;

//Окружаем минус двойками

okrBegin(mas, i - k, j, -2);

}

}

else if (napr == 1) // вправо

{

for (int k = kolPaluba - 1; k >= 1; k--) {

//Помещаем в ячейку число палуб

mas[i][j + k] = kolPaluba;

//Окружаем минус двойками

okrBegin(mas, i, j + k, -2);

}

}

else if (napr == 2) // вниз

{

for (int k = kolPaluba - 1; k >= 1; k--) {

//Помещаем в ячейку число палуб

mas[i + k][j] = kolPaluba;

//Окружаем минус двойками

okrBegin(mas, i + k, j, -2);

}

}

else if (napr == 3) // влево

{

for (int k = kolPaluba - 1; k >= 1; k--) {

//Помещаем в ячейку число палуб

mas[i][j -k] = kolPaluba;

//Окружаем минус двойками

okrBegin(mas, i, j - k, -2);

}

}

break;

}

}

//Конечное окружение

okrEnd(mas);

}

}