При реализации первого уровня сложности мы не писали программный код для класса game. Начнем разработку класса game. Теперь при запуске игры или нажатии на кнопку Новая игра на игровом поле игрока должны случайным образом генерироваться четыре однопалубных и один четырехпалубный корабль (см.рис. 1).



Рис. 1

При повторной генерации расположение кораблей должно меняться и быть случайным и непредсказуемым.

Перейдем на закладку game.java (см. рис. 2)к классу game.

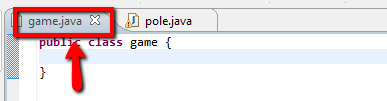


Рис. 2

Класс game, реализующий логику игры будет работать с двумерными массивами. Данные игровых полей компьютера и игрока будут храниться в двумерных массивах. Таймер отрисовки будет постоянно перерисовывать игровое поле на основании данных этих массивов. Этот подход ничем не отличается от использованного при создании игры "Змейка".

Прежде всего, добавим свойство –переменную класса game. Нам предстоит генерировать и отрисовывать данные поля игрока. Поэтому добавим двумерный массив целых чисел для этих целей:

// Класс реализации логики игры

public class game {

// Массив для игрового поля игрока

public int[][] masPlay;

}

Далее добавим конструктор класса:

// Класс реализации логики игры

public class game {

// Массив для игрового поля игрока

public int[][] masPlay;

// Конструктор класса

public game() {

//Создаем массив 10x10 - игровое поле игрока

masPlay = new int[10][10];

}

}

Конструктор автоматически запускается при создании объекта. Данный конструктор создает двумерный массив целых чисел размером 10x10, (десять строчек и десять колонок). После конструктора добавим метод start(), его задача выполнить запуск игры.

// Класс реализации логики игры

public class game {

// Массив для игрового поля игрока

public int[][] masPlay;

// Конструктор класса

public game() {

//Создаем массив 10x10 - игровое поле игрока

masPlay = new int[10][10];

}

// Запуск игры - начало игры

public void start() {

//Очищаем игровое поле игрока

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

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

masPlay[i][j] = 0;

}

}

}

}

При создании игры мы будем считать, что пустая ячейка игрового поля содержит нулевое значение. Поэтому,перед началом новой игры, сначала необходимо полностью очистить игровое поле. Именно это и происходит в методе start() c помощью двойного (вложенного) цикла for. Верхний цикл:

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

перебирает строки двумерного массива, а нижний (внутренний)цикл перебирает элементы массива в каждой строке:

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

Оба цикла выполняют по десять шагов и устанавливают нулевое значение во все 100 элементов двумерного массива.

При обращении к элементу массива:

masPlay[i][j] = 0;

в первых квадратных скобках указывается номер строки, во вторых квадратных скобках указывается номер элемента в строке. Вспомним, что нумерация начинается с нуля, поэтому максимальный номер –девять 9.

После обнуления массива получится двумерный массив, заполненный нулями (см. рис. 3). На рисунке (см. рис. 3)слева от массива находятся номера строк массива, а сверху номера позиций в строке. Такое расположение номеров будем использовать каждый раз при рассмотрении двумерного массива.Нумерация строк лежит в пределах от 0 до 9, нумерация позиций в строке также лежит в пределах от 0 до 9.

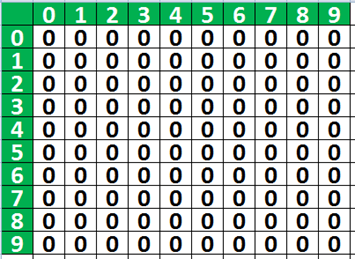


Рис. 3

Если имеется запись:

masPlay[1][3] = 0;

То такая запись помещает нулевое значение в элемент, находящийся в строке с номером один 1 и номер элемента в строке -три 3 (см. рис. 4).

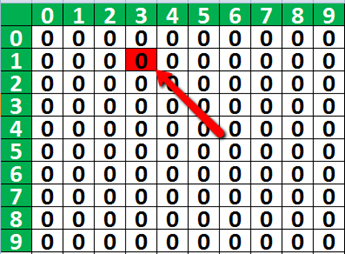


Рис. 4

Перейдем в класс pole, на закладку pole.java (см. рис. 5).

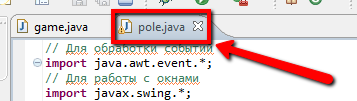


Рис. 5

В классе pole выполним отрисовку игрового поля игрока на основании данных двумерного массива. Прежде всего, добавим переменную myGame с типом game для класса pole:

//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;

Через эту переменную мы сможем получить доступ к двумерному массиву после создания объекта на основании класса game. В конструкторе класса pole выполним создание объекта и вызовем метод start()для запуска игры. Добавим строки в самом начале конструктора перед загрузкой изображений:

//Конструктор класса

public pole()

{

// Создаем объект новой игры

myGame = new game();

// Запускаем игру

myGame.start();

//Попытка загрузки всех изображений для игры

try

{

Теперь через переменную myGame можно обратиться к данным двумерного массива и вызывать методы класса game. Обработчик события кнопки Новая игра пока не выполняет никаких действий.Добавим в него вызов метода start():

//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);

Теперь каждый раз при нажатии на кнопку Новая игра будет происходить перезапуск игры. Осталось выполнить отрисовку игрового поля игрока на основании данных двумерного массива. Для этого перейдем в метод paintComponent()и после выведения надписей Компьютер, Игрок добавим программный код:

// Выведение надписей

gr.drawString("Компьютер", 150, 50);

gr.drawString("Игрок", 590, 50);

// Отрисовка игрового поля Игрока на основании массива

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

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

// 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);

}

}

}

// Отрисовка сетки игрового поля из синих линий

gr.setColor(Color.BLUE);

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

{

Два цикла for перебирают массив от начала до конца. Верхний цикл for перебирает строки, а внутренний (нижний) цикл for перебирает элементы в каждой строке. Если значение в массиве в пределах от 1 до 4, значит - это палуба корабля и мы выводим изображение палубы корабля. Самое сложное здесь - подобрать параметры в методе drawImage():

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

Первый параметр –это изображение paluba.png, которое загружено в игру.

Четвертый параметр–это ширина картинки в пикселях – 30.

Пятый параметр –это высота картинки в пикселях –30.

Сложнее всего дело обстоит со вторым и третьим параметром–это координаты верхнего левого уголка картинки по отношению к левому верхнему углу панели (игрового поля). Отступ слева для каждой ячейки будет: 500+j\*30, каждая следующая картинка должна быть правее соседней на 30 пикселей. Значение 500 – это левая граница игрового поля игрока. Отступ сверху для каждой ячейки будет:100+i\*30, каждая следующая картинка должна быть ниже соседней на 30 пикселей. Значение 100 – это верхняя граница игрового поля игрока.

Чтобы полностью отладить работу этих циклов –необходимо устанавливать различные значения в двумерном массиве в классе game. Затем запускать,смотреть и отлаживать.

На этом программный код класса pole для второго уровня сложности закончен.Остальная работа предстоит в классе game. Отрисовка в классе pole является зеркальным отражением двумерного массива на игровом поле.

Полный программный код класса pole для второго уровня сложности, выглядит так:

// 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 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;

// Конструктор класса

public pole()

{

// 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 игрового поля Игрока на основании массива

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

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

// 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);

}

}

}

//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);

}

}

}

Перейдем на закладку game.java (см. рис. 6)к классу game.

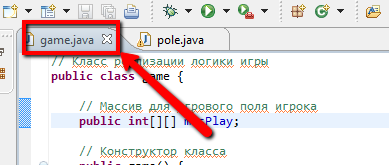


Рис. 6

Если сейчас выполнить запуск игры, то не увидим каких-либо изменений, потому что двумерный массив заполнен нулями. Перейдем в метод start()

после обнуления массива запишем значение один "1", например, в верхний левый элемент массива:

// Запуск игры - начало игры

public void start() {

//Очищаем игровое поле игрока

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

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

masPlay[i][j] = 0;

}

}

//Запишем единицу в верхний левый элемент массива

masPlay[0][0] = 1;

}

В результате наш массив будет выглядеть так (см. рис. 7).

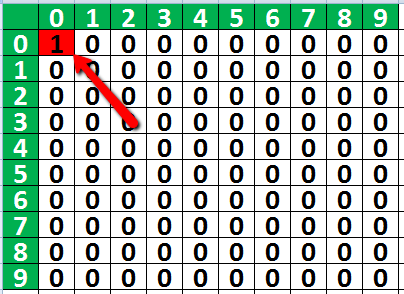


Рис. 7

Мы установили однопалубный корабль в верхний левый элемент массива.

Запустим наше приложение (см. рис. 8). В результате однопалубный корабль появился на игровом поле в указанном месте.Путем подобных экспериментов можно отлаживать программный код, который отвечает за отрисовку. Подставляя нужные значения в различные элементы массива –нужно смотреть на правильность отображения корабля на игровом поле.Удалим строчку:

//Запишем единицу в верхний левый элемент массива

masPlay[0][0] = 1;

Эта строка была добавлена для отладочных целей!



Рис. 8

Двумерный массив,который хранит данные игрового поля имеет размеры 10x10 (10 строчек, 10 элементов в каждой строке). Нумерация строк и элементов в строке лежит в пределах от 0 до 9. Номер не может быть меньше нуля или больше девяти! Если попытаться в программном коде выполнить следующие строки, то произойдет ошибка:

//Примеры выхода за границы массива

masPlay[-1][4] = 1;

masPlay[10][4] = 1;

Нельзя допускать выход за границы массива! В классе game нам предстоит очень много работать с двумерным массивом и нужно будет проверять ситуацию выхода за пределы массива. Добавим метод для тестирования данной ситуации:

// 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;

}

Внимание!!!Методы класса можно размещать в произвольном порядке внутри класса! Таким образом, метод testMasPoz() можно разместить под конструктором класса или над ним.

Методу testMasPoz(int i, int j) передаются два параметра –номер строки i-

и номер позиции в строке j. Метод выполняет проверку что i, j лежат в пределах от 0 до 9. Если все в порядке–метод возвращает true (истина), если происходит выход за пределы массива, то false(ложь).

Важно!!!Любой фрагмент программного кода, который будет неоднократно использоваться в программе, необходимо реализовать в виде отдельного метода. А готовый метод вызывать в нужных местах программного кода.

При записи значения в элемент массива – мы должны быть уверены, что не выходим за границы массива. Для этих целей добавим еще один метод:

// 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;

}

}

Метод setMasValue() запишет значение в элемент массива только в том случае, если не будет выхода за пределы массива. Этот метод гарантирует нам, что мы случайно не нарушим границы массива.

На игровом поле будут располагаться четыре однопалубных корабля. В двумерном массиве однопалубный корабль будет отмечен единицей "1". Корабли не могут располагаться рядом, вокруг каждого должны быть свободные ячейки. Свободные ячейки вокруг корабля в двумерном массиве будут отмечены минус единицей "-1". Значение минус единица "-1"будет признаком того, что в эту ячейку уже нельзя поместить палубу корабля. При каждой генерации кораблей –их расположение должно выбираться случайным образом. После генерации четырех однопалубных кораблей в двумерном массиве получится следующее (см. рис. 9).

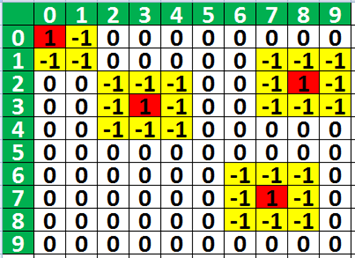


Рис. 9

Расположение кораблей каждый раз должно выбираться случайно.

Схема генерации одного однопалубного корабля будет заключаться в следующем:

1. Выбираем случайную ячейку

2. Если ячейка содержит нулевое значение, то размещаем в ней корабль

3. Если ячейка занята, то выбираем следующую случайную ячейку

4. Так продолжаем до тех пор, пока не найдем ячейку с нулевым значением

5. В выбранную ячейку помещаем единицу"1"

6. Пространство вокруг окружаем минус единицами "-1"

Создадим отдельный метод для установки одного значения при окружении:

// 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);

}

Метод setOkr() помещает одно значение для окружения в элемент массива.Прежде всего, происходит проверка, что нет выхода за пределы массива testMasPoz(i,j). И вторая проверка, что в ячейке находится нулевое значение.

При выполнении двух условий происходит установка нового значения.

Чтобы выполнить полное окружение необходимо установить значение вокруг элемента, а значит в восьми местах.Для этого добавим метод:

//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

}

Метод okrBegin() перебирает все ячейки вокруг и устанавливает в них нужное значение. Если происходит выход за границы массива –эта ситуация контролируется в методе setOkr().

Теперь можно добавить метод для генерации четырех однопалубных кораблей:

//Создание четырех однопалубных кораблей

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

// Цикл for делает четыре шага - для четырех кораблей

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

// Глухой цикл while

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;

}

}

}

}

Метод make1P() создает четыре однопалубных корабля. Верхний цикл for

выполняет четыре шага. Внутренний "глухой" цикл while выполняется до тех пор, пока не будет найдена свободная случайная ячейка с нулевым значением.После нахождения подходящей ячейки в нее записывается значение единица, происходит окружение минус единицами вокруг и прерывание цикла.

Теперь необходимо перейти в метод start() и вызвать метод генерации однопалубных кораблей:

//Запуск игры - начало игры

public void start() {

//Очищаем игровое поле игрока

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

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

masPlay[i][j] = 0;

}

}

//Генерация однопалубных кораблей

// для игрового поля Игрока

make1P(masPlay);

}

Запустим приложение и проверим результат (см. рис.10). При повторных нажатиях на кнопку Новая игра–корабли будут размещаться уже по-другому.



Рис. 10

Четырехпалубный корабль будет один на игровом поле и он будет самым большим. Найти случайное место для однопалубных кораблей проще всего.

Четырехпалубный корабль будем создавать самым первым, пока игровое поле еще не заполнено другими кораблями. Палубы такого корабля будут отмечаться в двумерном массиве числом четыре 4. Вокруг такой корабль будет также окружаться значениями минус единица -1. После генерации четырех палубного корабля в двумерном массиве получится следующее (см. рис. 11).

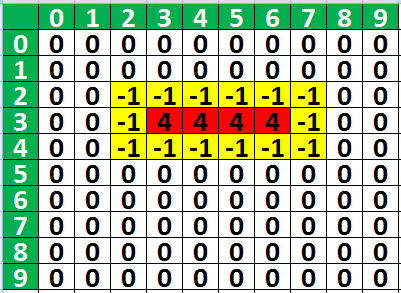


Рис. 11

Расположение корабля каждый раз должно выбираться случайно.

Схема генерации одного четырех палубного корабля будет заключаться в следующем:

1. Выбираем случайную ячейку –это будет голова корабля - одна из четырех палуб (см. рис. 12).

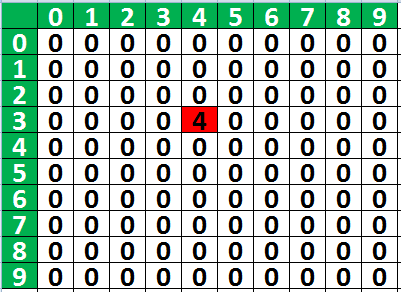


Рис. 12

2. На пустом поле эта ячейка содержит нулевое значение. Устанавливаем в нее число четыре 4 (см. рис. 12).

3. Окружаем ячейку значением минус два-2 (см. рис. 13). При построении корабля мы будем окружать каждую палубу значением минус два -2. После завершения построения заменим минус два -2 на минус один -1.

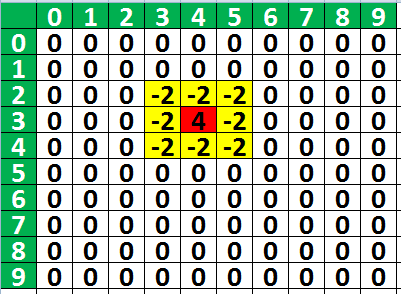


Рис. 13

4. Далее выбираем направление построения корабля –вверх, вправо, вниз, влево.Направление должно выбираться случайным образом одно из четырех.

5. Проверяем возможность построения дополнительных трех палуб корабля в выбранном направлении. Если происходит выход за границы массива –меняем направление на противоположное.

6. Добавляем три палубы. После добавления каждой палубы выполняем ее окружение значением минус два -2 (см. рис. 14, 15, 16).

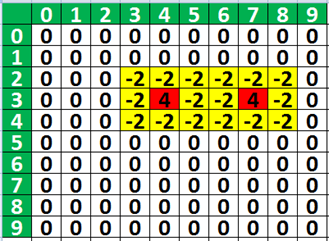


Рис. 14

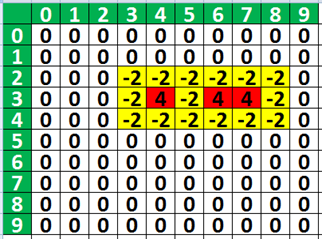


Рис. 15

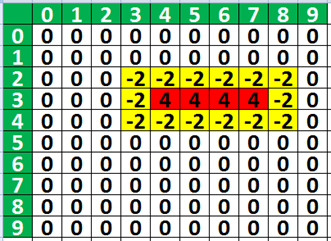


Рис. 16

7. После завершения построения корабля заменяем окружение значением минус два -2на значение минус один -1 (см. рис. 17).

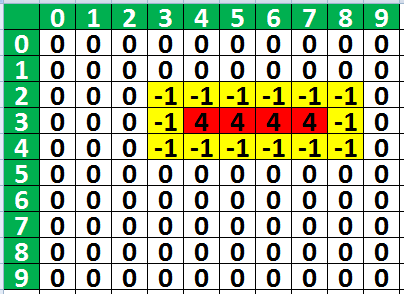


Рис. 17

Значение минус два-2 в двумерном массиве будет обозначать, что происходит построение корабля, и оно еще не завершено.Это необходимо, чтобы отличать окружение в виде минус единиц -1 соседних кораблей.

Добавим метод для конечного окружения корабля. Метод,который заменяет все значения минус два-2 в массиве на минус один -1:

//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;

}

}

}

Также нам понадобится метод для тестирования возможности расположения палубы корабля:

// 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;

}

Этот метод выполняет проверку выхода за границы массива и значение в ячейке, в которой мы хотим разместить новую палубу. Если это значения 0 или -2, то мы можем разместить там новую палубу.

Осталось добавить метод генерации четырех палубного корабля. Этот метод реализует схему построения четырех палубного корабля, рассмотренную выше:

//Создание одного четырех палубного корабля

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

//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);

//Помещаем в ячейку число четыре 4

mas[i][j] = 4;

//Окружаем минус двойками

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

//Выбираем случайное направление построения корабля

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

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

if (napr == 0) // вверх

{

// Если выход за границы массива

if (testNewPaluba(mas, i - 3, j) == false)

napr = 2; // меняем на вниз

}

else if (napr == 1) // вправо

{

// Если выход за границы массива

if (testNewPaluba(mas, i, j + 3) == false)

napr = 3; // меняем на влево

}

else if (napr == 2) // вниз

{

// Если выход за границы массива

if (testNewPaluba(mas, i + 3, j) == false)

napr = 0; // меняем на вверх

}

else if (napr == 3) // влево

{

// Если выход за границы массива

if (testNewPaluba(mas, i, j - 3) == false)

napr = 1; // меняем на вправо

}

if (napr == 0) // вверх

{

// Помещаем в ячейку число четыре 4

mas[i - 3][j] = 4;

// Окружаем минус двойками

okrBegin(mas, i - 3, j,-2);

// Помещаем в ячейку число четыре 4

mas[i - 2][j] = 4;

// Окружаем минус двойками

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

// Помещаем в ячейку число четыре 4

mas[i - 1][j] = 4;

// Окружаем минус двойками

okrBegin(mas, i - 1, j,-2);

}

else if (napr == 1) // вправо

{

// Помещаем в ячейку число четыре 4

mas[i][j + 3] = 4;

// Окружаем минус двойками

okrBegin(mas, i, j + 3,-2);

// Помещаем в ячейку число четыре 4

mas[i][j + 2] = 4;

// Окружаем минус двойками

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

// Помещаем в ячейку число четыре 4

mas[i][j + 1] = 4;

// Окружаем минус двойками

okrBegin(mas, i, j + 1,-2);

}

else if (napr == 2) // вниз

{

// Помещаем в ячейку число четыре 4

mas[i + 3][j] = 4;

// Окружаем минус двойками

okrBegin(mas, i + 3, j,-2);

// Помещаем в ячейку число четыре 4

mas[i + 2][j] = 4;

// Окружаем минус двойками

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

// Помещаем в ячейку число четыре 4

mas[i + 1][j] = 4;

// Окружаем минус двойками

okrBegin(mas, i + 1, j,-2);

}

else if (napr == 3) // влево

{

// Помещаем в ячейку число четыре 4

mas[i][j - 3] = 4;

// Окружаем минус двойками

okrBegin(mas, i, j - 3,-2);

// Помещаем в ячейку число четыре 4

mas[i][j - 2] = 4;

// Окружаем минус двойками

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

// Помещаем в ячейку число четыре 4

mas[i][j - 1] = 4;

// Окружаем минус двойками

okrBegin(mas, i, j - 1,-2);

}

//Конечное окружение

okrEnd(mas);

}

Чтобы этот метод выполнял свою работу, поместим его вызов в методе start():

//Запуск игры - начало игры

public void start() {

//Очищаем игровое поле игрока

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

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

masPlay[i][j] = 0;

}

}

//Создание четырехпалубного корабля

make4P(masPlay);

//Создание однопалубных кораблей

make1P(masPlay);

}

Обратите внимание, что построение четырехпалубного корабля идет первым!Только потом создаются однопалубные корабли. Запустим приложение (см. рис. 18).



Рис. 18

Реализация игры Морской бой для второго уровня сложности закончена. Полный программный код класса game для второго уровня сложности выглядит так:

// Класс реализации логики игры

public class game {

// Массив для игрового поля игрока

public int[][] masPlay;

// Конструктор класса

public game() {

//Создаем массив 10x10 - игровое поле игрока

masPlay = new int[10][10];

}

// Запуск игры - начало игры

public void start() {

//Очищаем игровое поле игрока

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

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

masPlay[i][j] = 0;

}

}

//Создание четырехпалубного корабля

make4P(masPlay);

//Создание однопалубных кораблей

make1P(masPlay);

}

// 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 одного четырехпалубного корабля

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

//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);

//Помещаем в ячейку число четыре 4

mas[i][j] = 4;

//Окружаем минус двойками

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

//Выбираем случайное направление построения корабля

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

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

if (napr == 0) // вверх

{

// Если выход за границы массива

if (testNewPaluba(mas, i - 3, j) == false)

napr = 2; // меняем на вниз

}

else if (napr == 1) // вправо

{

// Если выход за границы массива

if (testNewPaluba(mas, i, j + 3) == false)

napr = 3; // меняем на влево

}

else if (napr == 2) // вниз

{

// Если выход за границы массива

if (testNewPaluba(mas, i + 3, j) == false)

napr = 0; // меняем на вверх

}

else if (napr == 3) // влево

{

// Если выход за границы массива

if (testNewPaluba(mas, i, j - 3) == false)

napr = 1; // меняем на вправо

}

if (napr == 0) // вверх

{

// Помещаем в ячейку число четыре 4

mas[i - 3][j] = 4;

// Окружаем минус двойками

okrBegin(mas, i - 3, j,-2);

// Помещаем в ячейку число четыре 4

mas[i - 2][j] = 4;

// Окружаем минус двойками

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

// Помещаем в ячейку число четыре 4

mas[i - 1][j] = 4;

// Окружаем минус двойками

okrBegin(mas, i - 1, j,-2);

}

else if (napr == 1) // вправо

{

// Помещаем в ячейку число четыре 4

mas[i][j + 3] = 4;

// Окружаем минус двойками

okrBegin(mas, i, j + 3,-2);

// Помещаем в ячейку число четыре 4

mas[i][j + 2] = 4;

// Окружаем минус двойками

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

// Помещаем в ячейку число четыре 4

mas[i][j + 1] = 4;

// Окружаем минус двойками

okrBegin(mas, i, j + 1,-2);

}

else if (napr == 2) // вниз

{

// Помещаем в ячейку число четыре 4

mas[i + 3][j] = 4;

// Окружаем минус двойками

okrBegin(mas, i + 3, j,-2);

// Помещаем в ячейку число четыре 4

mas[i + 2][j] = 4;

// Окружаем минус двойками

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

// Помещаем в ячейку число четыре 4

mas[i + 1][j] = 4;

// Окружаем минус двойками

okrBegin(mas, i + 1, j,-2);

}

else if (napr == 3) // влево

{

// Помещаем в ячейку число четыре 4

mas[i][j - 3] = 4;

// Окружаем минус двойками

okrBegin(mas, i, j - 3,-2);

// Помещаем в ячейку число четыре 4

mas[i][j - 2] = 4;

// Окружаем минус двойками

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

// Помещаем в ячейку число четыре 4

mas[i][j - 1] = 4;

// Окружаем минус двойками

okrBegin(mas, i, j - 1,-2);

}

//Конечное окружение

okrEnd(mas);

}

}