Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

**Звіт з навчальної практики**

**Завдання №3.**

**«Гра Морський бій»**

Підготували студенти 1-го року навчання

Факультету інформатики

Спеціальність “ІПЗ”

Мокрий Михайло та Коваленко Андрій

5-та група

Викладач:

Пєчкурова О. М.

Київ 2018

Зміст

[1.Постановка задачі 2](#_Toc517014855)

[2. Опис проекту 2](#_Toc517014856)

[3.Структура програми 3](#_Toc517014857)

[1) Клас Comp2 3](#_Toc517014858)

[2) Клас GameLogic 3](#_Toc517014859)

[3) Клас MainClass 4](#_Toc517014860)

[4) Клас SeaField 4](#_Toc517014861)

[5) Клас Ship 4](#_Toc517014862)

[6) Клас Sound 4](#_Toc517014863)

[4. Інструкція користувача 4](#_Toc517014864)

[5.Розподіл задач 7](#_Toc517014865)

[6. Висновки 7](#_Toc517014866)

**Список ілюстрацій**

[Рис. 1 Стартове вікно гри 5](#_Toc517015657)

[Рис. 2 Гра з штучним інтелектом 6](#_Toc517015658)

[Рис. 3 Розставлення кораблів першого гравця 7](#_Toc517015659)

[Рис. 4 Розставлення кораблів другого гравця 8](#_Toc517015660)

# Постановка задачі

1. Написати гру Морський бій
2. Грати можна з комп’ютером та іншим гравцем.
3. Створити малюнок, що побудований з використанням графічних бібліотек
4. Написати графічну програму, в якій присутня анімація
5. Написати інтерактивну графічну гру. Мінімум 5 різних дій від користувача.

Опис проекту

На екрані з’являється панель вибору між локальною грою з іншим гравцем або з комп’ютером. Після вибору режиму гри, гравці обирають розстановку своїх кораблів (їх обмежена кількість, різної величини) на сітці 10 на 10.  Далі розташування кораблів стає невидимим для суперника і кожен гравець «скидає бомбу» на ворожий корабель по черзі. Будь-яке попадання відзначається замальованим квадратиком, промах – точкою. Виграти вдається тому гравцю, який першим потопив всі ворожі судна.

# Структура програми

Програма складається з 6 класів:

1) Клас Comp2 відповідає за дії комп’ютера. Код написаній таким чином щоб було 3 рівня складності. Перший рівень стріляє повністю рандомно. Другий та 3 вже має певну логіку і застосовує наступні методи. Приватні:

checkInjured(int, int)- у метод поступає точка куди пораненого корабля, він перевіряє чи нема ще поранення навколо цієї точки, аби зрозуміти як розташований корабель і намагається поцілити саме у наступну палубу корабля.

checkAround(int, int)- метод групує у собі всі приватні методи цього класу, і надає точку для пострілу навколо поданої якщо це можливо

checkDown(int, int)- перевіряє чи можна стрельнути вниз.

checkUp(int, int)- перевіряє чи можна стрельнути вгору.

checkRight(int, int)- перевіряє чи можна стрельнути вправо.

checkLeft(int, int)- перевіряє чи можна стрельнути вліво.

Публічні:

tryShot()- виклик метод для того щоб комп’ютер обрав точку для пострілу в залежності від рівня складності.

2) Клас GameLogic зберігає в собі дані полів, який корабель стоїть на клітці чи був постріл, має методі для розтановки кораблів та перевірки полів навколо них. Публічні:

startGame()- створює колекції кораблів для обох полів та очищує поле від попередньої гри.

cleanField()- метод для очищення поля.

addShipsOnCooserDeck() та addShipsOnDeck() – додають кораблі на поле шляхом змінення значення в масиві полей.

isShipAround()- перевіярє чи нема кораблів навколо для правильного розташування.

fillField()- розставляє кораблі на поле.

setShip()- ставить корабель у зазначене місце.

sendShoot()- метод змінює значення в масиві поля на пораненого коробля.

setFieldAroundHorizontalShip() та setFieldAroundVerticalShip()- при вбиті корабля ставить бомьи навколо.

getSizeShips() повертає розмір корабля.

3) Клас MainClass – викликає клас SeaFieald, встановлює розмір фрейму та запускає програму.

4) Клас SeaField - основний клас, що викликає всі інші класи в залежності від дій користувача. Клас малює саме поле та гру. Має слухачів для взаємодії з користувачем. Приватні методи:

mainMenuFrame()- метод, що малює основне меню гри.

gameFrame()- метод, що малює два поля для гри

setFieldFrame() та setField2Frame()- методи для обрання розташування кораблів на полі.

5) Клас Ship – клас що зберігає у собі всі данні корабля: розмір, життя, розташування, горизонтальність. Метод setShipCor встановлює кординати корабля.

6) Клас Sound взятий з опенсорсу для додачі музику в гру.

## Діаграма

# Інструкція користувача



Рис. 1 Стартове вікно гри

При запуску програми вас зустрічає меню, де ви можете обрати варіант гри з комп’ютером на 3 рівнях складності або з іншим гравцем. Також можна вийти з програми.

При виборі пунктів гри з комп’ютером або гравцем, далі можна розташувати кораблі на першому полі.

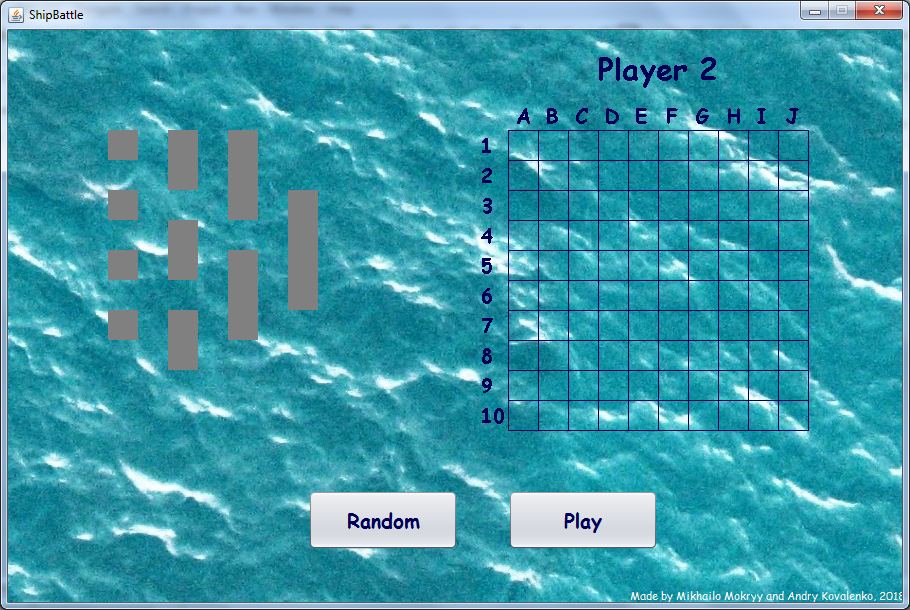


Рис. 2 Гра з штучним інтелектом

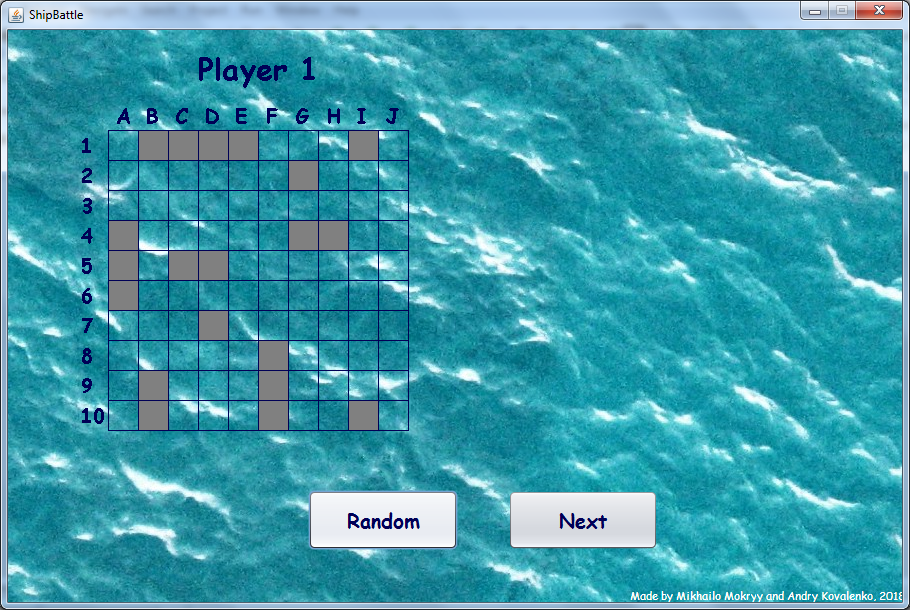


Рис. 3 Розставлення кораблів першого гравця

І далі для другого гравця при натисканні на Next рандомне розташування при Random

Наступний гравець

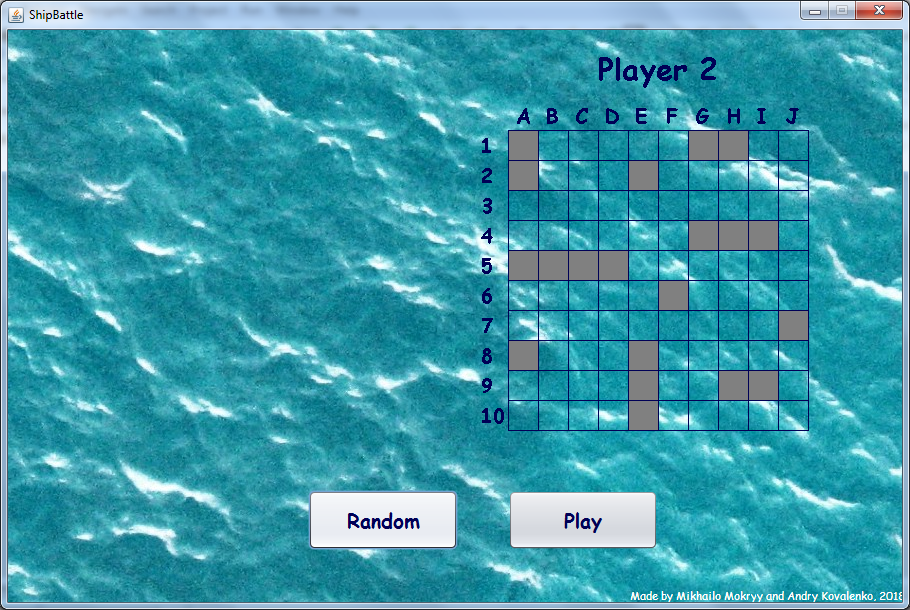
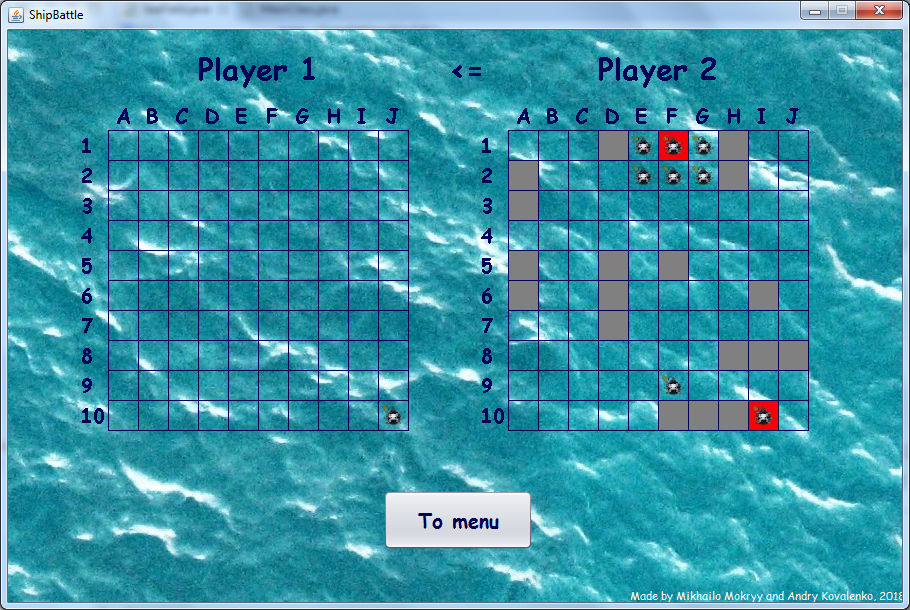


Рис. 4 Розставлення кораблів другого гравця

Далі йде сама гра де задача кожного гравця знайти та потопити ворожий флот

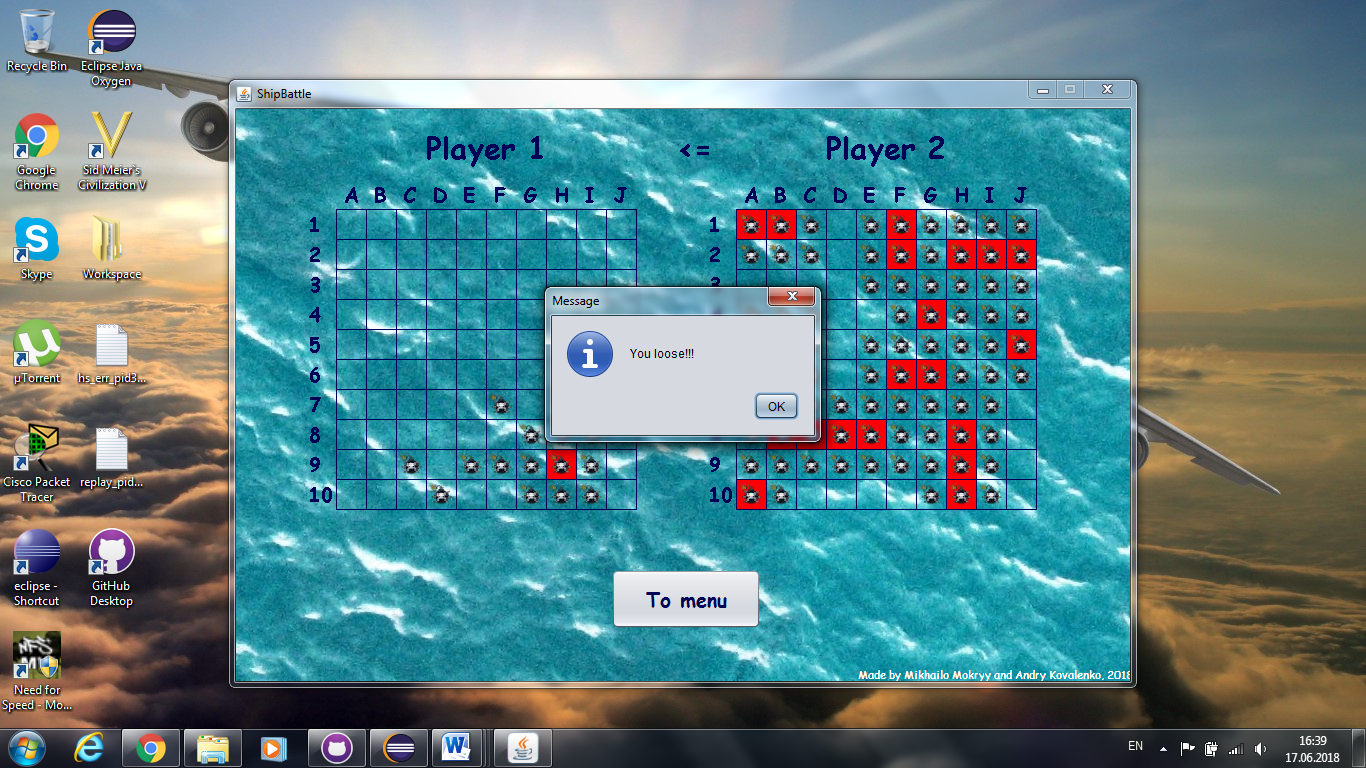
Гра з комп’ютером:



Гра з гравцем:



Показ переможця:



# Розподіл задач

Коваленко Андрій розробив класи Ship, Comp2, Sound.

Мокрий Михайло розробив класи MainClass, GameLogic, SeaField.

Під час програмування доповнювали класи кожного.

# Висновки

Була розроблена гра, яка надала нам досвід роботі с джава та її графічним інтерфейсом. Реалізовано 3 рівня складності. Робота проводилася в команді, що теж є чи малим досвідом. Розроблена інструкція. Однак під час виконання ми зіткнулися з проблемою джави яка не викликає метод коли нам потрібно, витративши декілька днів та прочитавши не одну статтю, ми не змогли вирішити цю проблему.

# Лістенінг програмного коду

**import** java.util.Random;

/\*\*

\* Computer intelligence smart

\*/

**public** **class** Comp2 {

**private** **int**[][] fieldArray;

**private** **int** x, y;

**private** **int** cleverLvl;

Random rand;

**public** Comp2(**int**[][] fieldArray, **int** lvl) {

**this**.fieldArray = fieldArray;

cleverLvl = lvl;

**this**.rand = **new** Random();

}

**private** **boolean** checkInjured(**int** x, **int** y) {

**if** (y < 9) {

**if** (fieldArray[x][y + 1] == -2)

**if** (checkUp(x, y))

**return** **true**;

}

**if** (y > 0) {

**if** (fieldArray[x][y - 1] == -2)

**if** (checkDown(x, y))

**return** **true**;

}

**if** (x < 9) {

**if** (fieldArray[x + 1][y] == -2)

**if** (checkLeft(x, y))

**return** **true**;

}

**if** (x > 0) {

**if** (fieldArray[x - 1][y] == -2)

**if** (checkRight(x, y))

**return** **true**;

}

**return** **false**;

}

**private** **boolean** checkAround(**int** x, **int** y) {

**if** (checkInjured(x, y))

**return** **true**;

**if** (checkUp(x, y)) {

**return** **true**;

} **else** **if** (checkDown(x, y)) {

**return** **true**;

} **else** **if** (checkRight(x, y)) {

**return** **true**;

} **else** **if** (checkLeft(x, y)) {

**return** **true**;

} **else**

**return** **false**;

}

**private** **boolean** checkDown(**int** x, **int** y) {

**if** (y < 9) {

**if** (fieldArray[x][y + 1] != -1) {

**if** (fieldArray[x][y + 1] == -2) {

// this.x = x;

//

// this.y = y + 2;

**return** checkDown(x, y + 1);

} **else** {

**this**.x = x;

**this**.y = y + 1;

**return** **true**;

}

}

}

**return** **false**;

}

**private** **boolean** checkUp(**int** x, **int** y) {

**if** (y > 0) {

**if** (fieldArray[x][y - 1] != -1) {

**if** (fieldArray[x][y - 1] == -2) {

// this.x = x;

//

// this.y = y - 2;

**return** checkUp(x, y - 1);

} **else** {

**this**.x = x;

**this**.y = y - 1;

**return** **true**;

}

}

}

**return** **false**;

}

**private** **boolean** checkRight(**int** x, **int** y) {

**if** (x < 9) {

**if** (fieldArray[x + 1][y] != -1) {

**if** (fieldArray[x + 1][y] == -2) {

// if (x < 8 && fieldArray[x + 2][y] != -1 && fieldArray[x + 2][y] != -2) {

// this.x = x + 2;

// } else {

// return false;

// }

// this.y = y;

**return** checkRight(x + 1, y);

} **else** {

**this**.x = x + 1;

**this**.y = y;

**return** **true**;

}

}

}

**return** **false**;

}

**private** **boolean** checkLeft(**int** x, **int** y) {

**if** (x > 0) {

**if** (fieldArray[x - 1][y] != -1) {

**if** (fieldArray[x - 1][y] == -2) {

// if (x > 2 && fieldArray[x - 2][y] != -1 && fieldArray[x - 2][y] != -2) {

// this.x = x - 2;

// } else {

// this.x = x + 1;

// }

// this.y = y;

**return** checkLeft(x - 1, y);

} **else** {

**this**.x = x - 1;

**this**.y = y;

**return** **true**;

}

}

}

**return** **false**;

}

// try to make a shot

**public** **void** tryShot() {

x = rand.nextInt(10);

y = rand.nextInt(10);

**if** (cleverLvl == 1) {

**while** (fieldArray[x][y] == -1 || fieldArray[x][y] == -2) {

x = rand.nextInt(10);

y = rand.nextInt(10);

}

}

**if** (cleverLvl == 2 || cleverLvl == 3) {

**boolean** stop = **true**;

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

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

**if** (fieldArray[j][i] == -2) {

**if** (checkAround(j, i)) {

stop = **false**;

**break**;

}

}

}

**if** (stop) {

// x = rand.nextInt(10);

// y = rand.nextInt(10);

**if** (cleverLvl == 3) {

**while** (fieldArray[x][y] != 1&&fieldArray[x][y] != 2&&fieldArray[x][y] != 3&&fieldArray[x][y] != 4) {

x = rand.nextInt(10);

y = rand.nextInt(10);

}

} **else** {

**while** (fieldArray[x][y] == -1 || fieldArray[x][y] == -2) {

x = rand.nextInt(10);

y = rand.nextInt(10);

}

}

}

}

}

}

/\*\*

\* get X cordinat

\*

\* **@return**

\*/

**public** **int** getX() {

**return** **this**.x;

}

/\*\*

\* get Y cordinat

\*

\* **@return**

\*/

**public** **int** getY() {

**return** **this**.y;

}

**public** **void** setCleverLvl(**int** cleverLvl) {

**this**.cleverLvl = cleverLvl;

}

}

**import** java.util.ArrayList;

**import** java.util.Random;

**public** **class** GameLogic {

**public** **int**[][] fieldArray1;

**public** **int**[][] fieldArray2;

**public** **int**[][] chooserArray;

**public** ArrayList<Ship> shipArray1;

**public** ArrayList<Ship> shipArray2;

**private** **int**[] sizeShips = {4,3,3,2,2,2,1,1,1,1};

**public** GameLogic() {

fieldArray1 = **new** **int**[10][10];

fieldArray2 = **new** **int**[10][10];

chooserArray = **new** **int**[8][7];

}

**public** **void** startGame() {

shipArray1 = **new** ArrayList<Ship>();

shipArray2 = **new** ArrayList<Ship>();

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

**for** (**int** k = 0; k < 10; k++) {

fieldArray1[i][k] = 0;

fieldArray2[i][k] = 0;

}

}

}

**public** **void** cleanField(**int**[][] fieldArray) {

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

**for** (**int** k = 0; k < 10; k++) {

fieldArray[i][k] = 0;

}

}

}

**public** **void** cleanEditorField() {

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

**for** (**int** k = 0; k < 7; k++) {

chooserArray[i][k] = 0;

}

}

}

**public** **void** addShipsOnCooserDeck(Ship sh) {

**if** (sh.isHorizontal()) {

**for** (**int** i = 0; i < sh.getDeckNum(); i++) {

chooserArray[sh.getXcor()][sh.getYcor() + i] = sh.getDeckNum();

}

}

**else** **if** (!sh.isHorizontal()) {

**for** (**int** i = 0; i < sh.getDeckNum(); i++) {

chooserArray[sh.getXcor() + i][sh.getYcor()] = sh.getDeckNum();

}

}

}

**public** **void** addShipsOnDeck(Ship sh, **int**[][] fieldArray) {

**if** (sh.isHorizontal()) {

**for** (**int** i = 0; i < sh.getDeckNum(); i++) {

fieldArray[sh.getXcor()][sh.getYcor() + i] = sh.getDeckNum();

}

}

**else** **if** (!sh.isHorizontal()) {

**for** (**int** i = 0; i < sh.getDeckNum(); i++) {

fieldArray[sh.getXcor() + i][sh.getYcor()] = sh.getDeckNum();

}

}

**if** (fieldArray.equals(fieldArray1))

shipArray1.add(sh);

**else** **if** (fieldArray.equals(fieldArray2))

shipArray2.add(sh);

}

// Checking are other ships directly around of ship

**private** **boolean** isShipAround(**int** row, **int** column, **int**[][] fieldArray) {

**boolean** isShip = **false**;

**if** ((row > 0) && (fieldArray[row - 1][column] > 0))

isShip = **true**;

**else** **if** ((row > 0) && (column < 9) && (fieldArray[row - 1][column + 1] > 0))

isShip = **true**;

**else** **if** ((column < 9) && (fieldArray[row][column + 1] > 0))

isShip = **true**;

**else** **if** ((row < 9) && (column < 9) && (fieldArray[row + 1][column + 1] > 0))

isShip = **true**;

**else** **if** ((row < 9) && (fieldArray[row + 1][column] > 0))

isShip = **true**;

**else** **if** ((row < 9) && (column > 0) && (fieldArray[row + 1][column - 1] > 0))

isShip = **true**;

**else** **if** ((column > 0) && (fieldArray[row][column - 1] > 0))

isShip = **true**;

**else** **if** ((row > 0) && (column > 0) && (fieldArray[row - 1][column - 1] > 0))

isShip = **true**;

**return** isShip;

}

// Input coordinates of ship manually or random

**public** **void** fillField(**int**[][] fieldArray, **boolean** isRandom) {

**boolean** isRepeat;

**int** row = 0, column = 0;

**int** num1 = 0, num2 =0, num3= 0;

**boolean** orientation = **false**;

Random rand = **new** Random();

chooserArray = **new** **int**[8][7];

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

**if** (!isRandom) {

orientation = **false**;

**if**(getSizeShips()[i]==1) {

row = num1++;

column = 0;

num1 += getSizeShips()[i];

}

**else** **if**(getSizeShips()[i]==2) {

row = num2++;

column = 2;

num2 += getSizeShips()[i];

}

**else** **if**(getSizeShips()[i]==3) {

row = num3++;

column = 4;

num3 += getSizeShips()[i];

}

**else** **if**(getSizeShips()[i]==4) {

row = 2;

column = 6 ;

}

addShipsOnCooserDeck(**new** Ship(row, column, getSizeShips()[i], orientation));

}

isRepeat = **true**;

**do** {

**if** (!isRandom) {

isRepeat = **false**;

} **else** {

row = rand.nextInt(10);

column = rand.nextInt(10);

orientation = rand.nextBoolean();

}

**if** (setShip(i,row, column, orientation, fieldArray))

isRepeat = **false**;

} **while** (isRepeat);

}

}

// Checking of opportunity to set ship by coordinates by row or column

**private** **boolean** setShip(**int** index, **int** row, **int** column, **boolean** orientation, **int**[][] fieldArray) {

**boolean** isSet = **true**;

**if** (fieldArray[row][column] > 0)

isSet = **false**;

**else** **if** (orientation)

**for** (**int** i = 0; i < getSizeShips()[index]; i++)

**if** ((column + i >= 10) || (isShipAround(row, column + i, fieldArray)))

isSet = **false**;

**else**

;

**else** **if** (!orientation)

**for** (**int** i = 0; i < getSizeShips()[index]; i++)

**if** ((row + i >= 10) || (isShipAround(row + i, column, fieldArray)))

isSet = **false**;

**else**

;

**if** (isSet) {

addShipsOnDeck(**new** Ship(row, column, getSizeShips()[index], orientation), fieldArray);

}

**return** isSet;

}

**public** **void** sendShoot(**int** row, **int** column, **int**[][] fieldArray) {

ArrayList<Ship> shipArray = **new** ArrayList<Ship>();

**int** field = 0;

**if** (fieldArray.equals(fieldArray1)) {

shipArray= shipArray1;

field = 1;

} **else** **if** (fieldArray.equals(fieldArray2)) {

shipArray=shipArray2;

field = 2;

}

**for** (Ship s : shipArray) {

**if**(!s.isDead()&&fieldArray[row][column]==s.getDeckNum()) {

**if** (s.isHorizontal()) {

**for** (**int** i = 0; i < s.getDeckNum(); i++) {

**if** (s.getXcor() == row && (s.getYcor() + i) == column&&fieldArray[s.getXcor()][s.getYcor() + i]==s.getDeckNum()) {

**if** (s.getHealth() > 0) {

s.setHealth(s.getHealth() - 1);

}**else** {

System.***out***.println(s.getDeckNum() + " - killed horizontal Field: " + field);

setFieldAroundHorizontalShip(s, fieldArray);

}

}

}

}

**else** **if** (!s.isHorizontal()) {

**for** (**int** i = 0; i < s.getDeckNum(); i++) {

**if** ((s.getXcor() + i) == row && s.getYcor() == column&&fieldArray[s.getXcor()+i][s.getYcor()]==s.getDeckNum()) {

**if** (s.getHealth() > 0) {

s.setHealth(s.getHealth() - 1);

}**else** {

System.***out***.println(s.getDeckNum() + " - killed vertical Field: " + field);

setFieldAroundVerticalShip(s, fieldArray);

}

}

}

}

}

}

}

**private** **void** setFieldAroundHorizontalShip(Ship s, **int**[][] fieldArray) {

**for** (**int** i = 0; i < s.getDeckNum(); i++) {

**if**(s.getXcor()>0&&fieldArray[s.getXcor()-1][s.getYcor()+i] == 0)

fieldArray[s.getXcor()-1][s.getYcor()+i] = -1;

**if**(s.getXcor()<9&&fieldArray[s.getXcor()+1][s.getYcor()+i] == 0)

fieldArray[s.getXcor()+1][s.getYcor()+i] = -1;

**if**(s.getXcor()>0&&s.getYcor()>0&&fieldArray[s.getXcor()-1][s.getYcor()-1] == 0)

fieldArray[s.getXcor()-1][s.getYcor()-1] = -1;

**if**(s.getYcor()>0&&fieldArray[s.getXcor()][s.getYcor()-1] == 0)

fieldArray[s.getXcor()][s.getYcor()-1] = -1;

**if**(s.getXcor()<9&&s.getYcor()>0&&fieldArray[s.getXcor()+1][s.getYcor()-1] == 0)

fieldArray[s.getXcor()+1][s.getYcor()-1] = -1;

**if**(s.getXcor()>0&&(s.getYcor()+s.getDeckNum())<10&&fieldArray[s.getXcor()-1][s.getYcor()+s.getDeckNum()] == 0)

fieldArray[s.getXcor()-1][s.getYcor()+s.getDeckNum()] = -1;

**if**((s.getYcor()+s.getDeckNum())<10&&fieldArray[s.getXcor()][s.getYcor()+s.getDeckNum()] == 0)

fieldArray[s.getXcor()][s.getYcor()+s.getDeckNum()] = -1;

**if**(s.getXcor()<9&&(s.getYcor()+s.getDeckNum())<10&&fieldArray[s.getXcor()+1][s.getYcor()+s.getDeckNum()] == 0)

fieldArray[s.getXcor()+1][s.getYcor()+s.getDeckNum()] = -1;

}

}

**private** **void** setFieldAroundVerticalShip(Ship s, **int**[][] fieldArray) {

**for** (**int** i = 0; i < s.getDeckNum(); i++) {

**if**(s.getYcor()>0&&fieldArray[s.getXcor()+i][s.getYcor()-1] == 0)

fieldArray[s.getXcor()+i][s.getYcor()-1] = -1;

**if**(s.getYcor()<9&&fieldArray[s.getXcor()+i][s.getYcor()+1] == 0)

fieldArray[s.getXcor()+i][s.getYcor()+1] = -1;

**if**(s.getXcor()>0&&s.getYcor()>0&&fieldArray[s.getXcor()-1][s.getYcor()-1] == 0)

fieldArray[s.getXcor()-1][s.getYcor()-1] = -1;

**if**(s.getXcor()>0&&fieldArray[s.getXcor()-1][s.getYcor()] == 0)

fieldArray[s.getXcor()-1][s.getYcor()] = -1;

**if**(s.getXcor()>0&&s.getYcor()<9&&fieldArray[s.getXcor()-1][s.getYcor()+1] == 0)

fieldArray[s.getXcor()-1][s.getYcor()+1] = -1;

**if**((s.getXcor()+s.getDeckNum())<10&&s.getYcor()>0&&fieldArray[s.getXcor()+s.getDeckNum()][s.getYcor()-1] == 0)

fieldArray[s.getXcor()+s.getDeckNum()][s.getYcor()-1] = -1;

**if**((s.getXcor()+s.getDeckNum())<10&&fieldArray[s.getXcor()+s.getDeckNum()][s.getYcor()] == 0)

fieldArray[s.getXcor()+s.getDeckNum()][s.getYcor()] = -1;

**if**((s.getXcor()+s.getDeckNum())<10&&s.getYcor()<9&&fieldArray[s.getXcor()+s.getDeckNum()][s.getYcor()+1] == 0)

fieldArray[s.getXcor()+s.getDeckNum()][s.getYcor()+1] = -1;

}

}

**public** **int**[] getSizeShips() {

**return** sizeShips;

}

}

**import** java.awt.Dimension;

**import** java.awt.EventQueue;

**import** java.awt.Toolkit;

**import** javax.swing.JFrame;

**import** javax.swing.JPanel;

**import** javax.swing.UIManager;

**import** javax.swing.UnsupportedLookAndFeelException;

**import** javax.swing.border.EmptyBorder;

**import** javax.swing.plaf.nimbus.NimbusLookAndFeel;

**public** **class** MainClass **extends** JFrame {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** JPanel contentPane;

**private** Dimension dim;

/\*\*

\* Launch the application.

\*

\* **@throws** UnsupportedLookAndFeelException

\*/

**public** **static** **void** main(String[] args) {

EventQueue.*invokeLater*(**new** Runnable() {

**public** **void** run() {

**try** {

UIManager.*setLookAndFeel*(**new** NimbusLookAndFeel());

MainClass frame = **new** MainClass();

frame.setVisible(**true**);

frame.setResizable(**false**);

} **catch** (Exception e) {

e.printStackTrace();

}

}

});

}

/\*\*

\* Create the frame.

\*/

**public** MainClass() {

setTitle("ShipBattle");

dim = Toolkit.*getDefaultToolkit*().getScreenSize();

dim = Toolkit.*getDefaultToolkit*().getScreenSize();

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

setBounds((dim.width-900)/2, (dim.height-600)/2, 900, 600);

contentPane = **new** JPanel();

contentPane.setBorder(**new** EmptyBorder(5, 5, 5, 5));

setContentPane(contentPane);

SeaField sf = **new** SeaField();

sf.setBounds(0, 0, 900, 600);

contentPane.add(sf);

contentPane.setLayout(**null**);

}

}

**import** java.awt.Color;

**import** java.awt.Component;

**import** java.awt.Cursor;

**import** java.awt.Dimension;

**import** java.awt.Font;

**import** java.awt.Graphics;

**import** java.awt.Graphics2D;

**import** java.awt.GridBagConstraints;

**import** java.awt.GridBagLayout;

**import** java.awt.Image;

**import** java.awt.Toolkit;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.MouseEvent;

**import** java.awt.event.MouseListener;

**import** java.awt.event.MouseMotionListener;

**import** java.io.File;

**import** javax.imageio.ImageIO;

**import** javax.swing.JButton;

**import** javax.swing.JLabel;

**import** javax.swing.JOptionPane;

**import** javax.swing.JPanel;

**public** **class** SeaField **extends** JPanel {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** Color DARK\_BLUE = Color.***BLUE***.darker().darker().darker();

**private** Font font = **new** Font("Comic Sans MS", Font.***BOLD***, 15);

**private** Image background, shipPart, hitPart, bomb;

**private** Graphics2D g2d;

**private** MyActionListener action;

**private** GameLogic game;

**private** **int** x, y;

**private** JLabel devInfo;

**private** JButton newGamePVE,newGamePVE2,newGamePVE3,newGamePVP,goToMenuBtn, exitBtn, playBtn, randomBtn1,randomBtn2, nextBtn;

**private** Comp2 robot;

**private** **int** turn;

**private** String pl;

**private** **int** prog;

**private** **boolean** field1Vis;

**private** **boolean** field2Vis;

**private** **boolean** isMenu,fieldEdible;

**private** **boolean** isFieldSet1,isFieldSet2,isField1,isField2, flag, isPVP;

**private** **int** hits1;

**private** **int** hits2;

**int** imageX, imageY;

**public** SeaField() {

game = **new** GameLogic();

game.startGame();

robot = **new** Comp2(game.fieldArray2, 2);

turn = 1;

pl = "<=";

prog = 2;

field1Vis=**true**;

field2Vis=**true**;

addMouseListener(**new** MyMouseListener());

addMouseMotionListener(**new** MyMouseMotionListener());

setFocusable(**true**);

**try** {

background = ImageIO.*read*(**new** File("Pics/water.png")); // **TODO** change PATH

shipPart = ImageIO.*read*(**new** File("Pics/gray\_sqr.png"));

hitPart = ImageIO.*read*(**new** File("Pics/red\_sqr.png"));

bomb = ImageIO.*read*(**new** File("Pics/bomb.png"));

} **catch** (Exception e) {

e.printStackTrace();

System.***err***.println("Change pictures directory!");

}

setLayout(**null**);

action = **new** MyActionListener();

mainMenuFrame();

//Sound.playSound("Sounds/Wot2.wav").play();

// RepaintManager.currentManager(null).setDoubleBufferingEnabled(false);

// shipPart = shipPart.getScaledInstance(30, 30, Image.SCALE\_DEFAULT);

//**TODO**

}

@Override

**public** **void** paintComponent(Graphics g) {

**super**.paintComponent(g);

g2d = (Graphics2D) g;

g2d.drawImage(background, 0, 0, 900, 600, **null**);

g2d.setFont(font.deriveFont(30f));

g2d.setColor(DARK\_BLUE);

**if**(!isMenu()) {

**if**(isFieldSet2())

g2d.drawString("Player 1", 190, 50);

**if**(isFieldSet1())

g2d.drawString("Player 2", 590, 50);

**if**(isField1()&&isField2())

g2d.drawString(pl, 440, 50);

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

**for** (**int** k = 0; k < 10; k++) {

**if**(field1Vis) {

**if**(isField1()) {

**if** (game.fieldArray1[i][k] >= 1&&game.fieldArray1[i][k] <= 4) {

g.drawImage(shipPart, 100 + k \* 30, 100 + i \* 30, 30, 30, **null**);

}

}

}

**if** (game.fieldArray1[i][k] == -1) {

g.drawImage(bomb, 100 + k \* 30, 100 + i \* 30, 30, 30, **null**);

}

**if** (game.fieldArray1[i][k] == -2) {

g.drawImage(hitPart, 100 + k \* 30, 100 + i \* 30, 30, 30, **null**);

g.drawImage(bomb, 100 + k \* 30, 100 + i \* 30, 30, 30, **null**);

}

**if**(field2Vis) {

**if**(isField2()) {

**if** (game.fieldArray2[i][k] >= 1&&game.fieldArray2[i][k] <= 4) {

g.drawImage(shipPart, 500 + k \* 30, 100 + i \* 30, 30, 30, **null**);

}

}

}

**if** (game.fieldArray2[i][k] == -1) {

g.drawImage(bomb, 500 + k \* 30, 100 + i \* 30, 30, 30, **null**);

}

**if** (game.fieldArray2[i][k] == -2) {

g.drawImage(hitPart, 500 + k \* 30, 100 + i \* 30, 30, 30, **null**);

g.drawImage(bomb, 500 + k \* 30, 100 + i \* 30, 30, 30, **null**);

}

}

}

}

g2d.setColor(DARK\_BLUE);

**if**(!isFieldSet2()&&isFieldSet1()&&flag) {

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

**for** (**int** k = 0; k < 7; k++) {

**if** (game.chooserArray[i][k] >= 1&&game.chooserArray[i][k] <= 4) {

g.drawImage(shipPart, 100 + k \* 30, 100 + i \* 30, 30, 30, **null**);

}

}

}

}

**if**(isFieldSet2()&&!isFieldSet1()&&!isField1()&&!flag) {

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

**for** (**int** k = 0; k < 7; k++) {

**if** (game.chooserArray[i][k] >= 1&&game.chooserArray[i][k] <= 4) {

g.drawImage(shipPart, 500 + k \* 30, 100 + i \* 30, 30, 30, **null**);

}

}

}

}

// //**TODO**

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

// for (int k = 0; k < 10; k++) {

//

// if(imageX>=90 + k \* 30&&imageX<=110 + k \* 30&&imageY>=90 + i \* 30&&imageY<=110 + i \* 30) {

// game.fieldArray2[i][k] = 1;

// g.drawImage(shipPart, 100 + k \* 30, 100 + i \* 30, 30, 30, null);

//

//

//

// }

// }

//

// g.drawImage(shipPart, imageX, imageY, null);

// }

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

**if**(isFieldSet2()) {

g2d.drawLine(100 + i \* 30, 100, 100 + i \* 30, 400);

g2d.drawLine(100, 100 + i \* 30, 400, 100 + i \* 30);

}

**if**(isFieldSet1()) {

g2d.drawLine(500 + i \* 30, 100, 500 + i \* 30, 400);

g2d.drawLine(500, 100 + i \* 30, 800, 100 + i \* 30);

}

}

g2d.setFont(font.deriveFont(20f));

g2d.setColor(DARK\_BLUE);

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

String numb = Integer.*toString*(i);

**if**(isFieldSet2()) {

g2d.drawString(numb, 73, 93 + i \* 30);

g2d.drawString("" + (**char**) ('A' + i - 1), 78 + i \* 30, 93);

}

**if**(isFieldSet1()) {

g2d.drawString(numb, 473, 93 + i \* 30);

g2d.drawString("" + (**char**) ('A' + i - 1), 478 + i \* 30, 93);

}

}

}

/\*\*

\* РњРµС‚РѕРґ РІС–РґРѕР±СЂР°Р¶РµРЅРЅСЏ РѕСЃРЅРѕРІРЅРѕРіРѕ РјРµРЅСЋ.

\*/

**private** **void** mainMenuFrame() {

setLayout(**null**);

setVisible(**true**);

newGamePVE = **new** JButton();

newGamePVE.setText("Low");

newGamePVE.setForeground(DARK\_BLUE);

newGamePVE.setFont(font.deriveFont(20f));

newGamePVE.setBounds(350, 120, 200, 60);

newGamePVE.addActionListener(action);

add(newGamePVE);

newGamePVE2 = **new** JButton();

newGamePVE2.setText("Medium");

newGamePVE2.setForeground(DARK\_BLUE);

newGamePVE2.setFont(font.deriveFont(20f));

newGamePVE2.setBounds(350, 200, 200, 60);

newGamePVE2.addActionListener(action);

add(newGamePVE2);

newGamePVE3 = **new** JButton();

newGamePVE3.setText("Hard");

newGamePVE3.setForeground(DARK\_BLUE);

newGamePVE3.setFont(font.deriveFont(20f));

newGamePVE3.setBounds(350, 280, 200, 60);

newGamePVE3.addActionListener(action);

add(newGamePVE3);

newGamePVP = **new** JButton();

newGamePVP.setText("With Player");

newGamePVP.setForeground(DARK\_BLUE);

newGamePVP.setFont(font.deriveFont(20f));

newGamePVP.setBounds(350, 360, 200, 60);

newGamePVP.addActionListener(action);

add(newGamePVP);

exitBtn = **new** JButton();

exitBtn.setText("Exit");

exitBtn.setForeground(DARK\_BLUE);

exitBtn.setFont(font.deriveFont(20f));

exitBtn.setBounds(350, 440, 200, 60);

exitBtn.addActionListener(action);

add(exitBtn);

devInfo = **new** JLabel("<html>Made by Mikhailo Mokryy and Andry Kovalenko, 2018</html>");

devInfo.setForeground(**new** Color(255, 255, 255));

devInfo.setFont(font.deriveFont(Font.***PLAIN*** ,11f));

devInfo.setBackground(**new** Color(255, 255, 255));

devInfo.setBounds(622, 547, 300, 37);

add(devInfo);

setFieldSet1(**false**);

setFieldSet2(**false**);

setFieldEdible(**false**);

setField1(**false**);

setMenu(**true**);

}

**private** **void** gameFrame() {

playBtn.setVisible(**false**);

randomBtn2.setVisible(**false**);

setVisible(**true**);

setLayout(**null**);

goToMenuBtn = **new** JButton();

goToMenuBtn.setText("To menu");

goToMenuBtn.setForeground(DARK\_BLUE);

goToMenuBtn.setFont(font.deriveFont(20f));

goToMenuBtn.setBounds(375, 460, 150, 60);

goToMenuBtn.addActionListener(action);

add(goToMenuBtn);

setField2(**true**);

setField1(**true**);

setMenu(**false**);

setFieldSet1(**true**);

setFieldSet2(**true**);

setFieldEdible(**true**);

setField1(**true**);

}

**private** **void** setField2Frame() {

newGamePVP.setVisible(**false**);

newGamePVE.setVisible(**false**);

newGamePVE2.setVisible(**false**);

newGamePVE3.setVisible(**false**);

exitBtn.setVisible(**false**);

setVisible(**true**);

setLayout(**null**);

playBtn = **new** JButton();

playBtn.setText("Play");

playBtn.setForeground(DARK\_BLUE);

playBtn.setFont(font.deriveFont(20f));

playBtn.setBounds(500, 460, 150, 60);

playBtn.addActionListener(action);

add(playBtn);

randomBtn2 = **new** JButton();

randomBtn2.setText("Random");

randomBtn2.setForeground(DARK\_BLUE);

randomBtn2.setFont(font.deriveFont(20f));

randomBtn2.setBounds(300, 460, 150, 60);

randomBtn2.addActionListener(action);

add(randomBtn2);

game.cleanEditorField();

game.fillField(game.fieldArray2, **false**);

flag =**true**;

setMenu(**false**);

setFieldSet1(**true**);

setFieldSet2(**false**);

}

**private** **void** setFieldFrame() {

newGamePVP.setVisible(**false**);

newGamePVE.setVisible(**false**);

newGamePVE2.setVisible(**false**);

newGamePVE3.setVisible(**false**);

exitBtn.setVisible(**false**);

setVisible(**true**);

setLayout(**null**);

nextBtn = **new** JButton();

nextBtn.setText("Next");

nextBtn.setForeground(DARK\_BLUE);

nextBtn.setFont(font.deriveFont(20f));

nextBtn.setBounds(500, 460, 150, 60);

nextBtn.addActionListener(action);

add(nextBtn);

randomBtn1 = **new** JButton();

randomBtn1.setText("Random");

randomBtn1.setForeground(DARK\_BLUE);

randomBtn1.setFont(font.deriveFont(20f));

randomBtn1.setBounds(300, 460, 150, 60);

randomBtn1.addActionListener(action);

add(randomBtn1);

game.cleanEditorField();

game.fillField(game.fieldArray1, **false**);

flag =**false**;

setMenu(**false**);

setFieldSet1(**false**);

setFieldSet2(**true**);

}

**public** **class** MyMouseListener **implements** MouseListener {

@Override

**public** **void** mouseClicked(MouseEvent e) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** mousePressed(MouseEvent e) {

**if**(isFieldSet1()&&isFieldSet2()&&isFieldEdible()) {

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

**if** (turn % 2 != 0) {

x = e.getX();

y = e.getY();

**if** ((x > 100) && (y > 100) && (x < 400) && (y < 400)) {

**int** i = (y - 100) / 30;

**int** k = (x - 100) / 30;

**if** (game.fieldArray1[i][k] >= 1&&game.fieldArray1[i][k] <= 4) {

game.sendShoot(i, k, game.fieldArray1);

game.fieldArray1[i][k] = -2;

hits1++;

pl = "<=";

turn+=2;

Sound.*playSound*("Sounds/exp.wav").play();

} **else** **if** (game.fieldArray1[i][k] == 0) {

game.fieldArray1[i][k] = -1;

Sound.*playSound*("Sounds/bubble.wav").play();

pl = "=>";

**if** (!isPVP) {

pl = "<=";

turn++;

robot.tryShot();

i = robot.getX();

k = robot.getY();

**if** (game.fieldArray2[i][k] >= 1&&game.fieldArray2[i][k] <= 4) {

**while**(game.fieldArray2[i][k] >= 1&&game.fieldArray2[i][k] <= 4) {

hits2++;

game.sendShoot(i, k, game.fieldArray2);

game.fieldArray2[i][k] = -2;

**if**(hits2==20)

**break**;

robot.tryShot();

i = robot.getX();

k = robot.getY();

**if** (game.fieldArray2[i][k] == 0) {

game.fieldArray2[i][k] = -1;

Sound.*playSound*("Sounds/bubble.wav").join();

}

repaint();

revalidate();

Sound.*playSound*("Sounds/exp.wav").join();

}

} **else** **if** (game.fieldArray2[i][k] == 0)

game.fieldArray2[i][k] = -1;

Sound.*playSound*("Sounds/bubble.wav").play();

}

turn++;

}

repaint();

revalidate();

}

} **else** {

**if** (isPVP) {

**if** ((x > 500) && (y > 100) && (x < 800) && (y < 400)) {

**int** i = (y - 100) / 30;

**int** k = (x - 500) / 30;

**if** (game.fieldArray2[i][k] >= 1&&game.fieldArray2[i][k] <= 4) {

game.sendShoot(i, k, game.fieldArray2);

game.fieldArray2[i][k] = -2;

hits2++;

pl = "=>";

Sound.*playSound*("Sounds/exp.wav").play();

turn+=2;

} **else** **if** (game.fieldArray2[i][k] == 0) {

game.fieldArray2[i][k] = -1;

Sound.*playSound*("Sounds/bubble.wav").play();

pl = "<=";

**if** (!isPVP) {

pl = "=>";

turn++;

robot.tryShot();

i = robot.getX();

k = robot.getY();

**if** (game.fieldArray1[i][k] >= 1&&game.fieldArray1[i][k] <= 4) {

**while**(game.fieldArray1[i][k] >= 1&&game.fieldArray1[i][k] <= 4) {

hits1++;

game.sendShoot(i, k, game.fieldArray1);

game.fieldArray1[i][k] = -2;

**if**(hits1==20)

**break**;

robot.tryShot();

i = robot.getX();

k = robot.getY();

**if** (game.fieldArray1[i][k] == 0) {

game.fieldArray1[i][k] = -1;

Sound.*playSound*("Sounds/bubble.wav").play();

}

repaint();

revalidate();

Sound.*playSound*("Sounds/exp.wav").play();

}

} **else** **if** (game.fieldArray1[i][k] == 0) {

game.fieldArray1[i][k] = -1;

Sound.*playSound*("Sounds/bubble.wav").play();

}

}

turn++;

}

repaint();

revalidate();

}

}

}

}

}

**if**(hits1==20&&isFieldEdible()) {

setFieldEdible(**false**);

Sound.*playSound*("Sounds/win.wav").play();

JOptionPane.*showMessageDialog*(**null**,

"Player 1 win!!!", "Message",

JOptionPane.***INFORMATION\_MESSAGE***);

field1Vis=**true**;

field2Vis=**true**;

repaint();

revalidate();

}

**if**(hits2==20&&isFieldEdible()) {;

setFieldEdible(**false**);

**if**(isPVP) {

Sound.*playSound*("Sounds/win.wav").play();

JOptionPane.*showMessageDialog*(**null**,

"Player 2 win!!!", "Message",

JOptionPane.***INFORMATION\_MESSAGE***);

}**else** {

Sound.*playSound*("Sounds/defeat.wav").play();

JOptionPane.*showMessageDialog*(**null**,

"You loose!!!", "Message",

JOptionPane.***INFORMATION\_MESSAGE***);

}

field1Vis=**true**;

field2Vis=**true**;

repaint();

revalidate();

}

}

@Override

**public** **void** mouseReleased(MouseEvent e) {

}

@Override

**public** **void** mouseEntered(MouseEvent e) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** mouseExited(MouseEvent e) {

// **TODO** Auto-generated method stub

}

}

**public** **class** MyMouseMotionListener **implements** MouseMotionListener {

@Override

**public** **void** mouseDragged(MouseEvent e) {

// if(isFieldSet()&&!isFieldSet2()&&!isFieldEdible()) {

// imageX = e.getX()-15;

// imageY = e.getY()-15;

// repaint();

// }

}

@Override

**public** **void** mouseMoved(MouseEvent e) {

x = e.getX();

y = e.getY();

**if** (((x > 100) && (y > 100) && (x < 400) && (y < 400))

|| ((x > 500) && (y > 100) && (x < 800) && (y < 400)))

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

**else**

setCursor(**new** Cursor(Cursor.***HAND\_CURSOR***));

}

}

@Override

**public** Dimension getPreferredSize() {

**return** **new** Dimension(900, 600);

}

**private** **class** MyActionListener **implements** ActionListener {

**public** **void** actionPerformed(ActionEvent e) {

Sound.*playSound*("Sounds/pressBtn.wav").play(**true**);

**if** (e.getSource().equals(newGamePVE)) {

setField2Frame();

game.startGame();

game.fillField(game.fieldArray1, **true**);

game.fillField(game.fieldArray2, **false**);

hits1=0;

hits2=0;

turn = 1;

pl = "<=";

prog = 2;

robot.setCleverLvl(1);

isPVP = **false**;

field1Vis=**true**;

field2Vis = **false**;

}

**if** (e.getSource().equals(newGamePVE2)) {

setField2Frame();

game.startGame();

game.fillField(game.fieldArray1, **true**);

game.fillField(game.fieldArray2, **false**);

hits1=0;

hits2=0;

turn = 1;

pl = "<=";

prog = 2;

robot.setCleverLvl(2);

isPVP = **false**;

field1Vis=**true**;

field2Vis = **false**;

}

**if** (e.getSource().equals(newGamePVE3)) {

setField2Frame();

game.startGame();

game.fillField(game.fieldArray1, **true**);

game.fillField(game.fieldArray2, **false**);

hits1=0;

hits2=0;

turn = 1;

pl = "<=";

prog = 2;

robot.setCleverLvl(3);

isPVP = **false**;

field1Vis=**true**;

field2Vis = **false**;

}

**if** (e.getSource().equals(newGamePVP)) {

setFieldFrame();

game.startGame();

game.fillField(game.fieldArray1, **false**);

game.fillField(game.fieldArray2, **false**);

hits1=0;

hits2=0;

turn = 1;

pl = "<=";

prog = 1;

isPVP = **true**;

field1Vis=**true**;

field2Vis=**true**;

}

**if** (e.getSource().equals(nextBtn)) {

setField2Frame();

nextBtn.setVisible(**false**);

randomBtn1.setVisible(**false**);

field1Vis =**false**;

field2Vis = **false**;

}

**if** (e.getSource().equals(goToMenuBtn)) {

mainMenuFrame();

goToMenuBtn.setVisible(**false**);

game.cleanEditorField();

game.startGame();

}

**if** (e.getSource().equals(playBtn)) {

gameFrame();

field1Vis = **false**;

**if**(isPVP)

field2Vis =**false**;

**else** field2Vis =**true**;

// field1Vis=true;

// field2Vis=true;

}

**if** (e.getSource().equals(randomBtn2)) {

field2Vis = **true**;

setField2(**true**);

setField1(**false**);

setFieldSet2(**false**);

setFieldSet1(**true**);

game.cleanEditorField();

game.cleanField(game.fieldArray2);

game.fillField(game.fieldArray2, **true**);

**if**(!isPVP) {

game.cleanField(game.fieldArray1);

game.fillField(game.fieldArray1, **true**);

}

}

**if** (e.getSource().equals(randomBtn1)) {

setField1(**true**);

setField2(**false**);

setFieldSet2(**true**);

setFieldSet1(**false**);

game.cleanEditorField();

game.cleanField(game.fieldArray1);

game.fillField(game.fieldArray1, **true**);

}

**if** (e.getSource().equals(exitBtn)) {

System.*exit*(0);

}

repaint();

revalidate();

}

}

/\*\*

\* **@return** the isMenu

\*/

**public** **boolean** isMenu() {

**return** isMenu;

}

/\*\*

\* **@param** isMenu the isMenu to set

\*/

**public** **void** setMenu(**boolean** isMenu) {

**this**.isMenu = isMenu;

}

**public** **boolean** isFieldSet1() {

**return** isFieldSet1;

}

**public** **void** setFieldSet1(**boolean** isFieldSet) {

**this**.isFieldSet1 = isFieldSet;

}

**public** **boolean** isFieldSet2() {

**return** isFieldSet2;

}

**public** **void** setFieldSet2(**boolean** isFieldSet2) {

**this**.isFieldSet2 = isFieldSet2;

}

**public** **boolean** isFieldEdible() {

**return** fieldEdible;

}

**public** **void** setFieldEdible(**boolean** fieldEdible) {

**this**.fieldEdible = fieldEdible;

}

**public** **boolean** isField1() {

**return** isField1;

}

**public** **void** setField1(**boolean** isField) {

**this**.isField1 = isField;

}

**public** **boolean** isField2() {

**return** isField2;

}

**public** **void** setField2(**boolean** isField2) {

**this**.isField2 = isField2;

}

}

**public** **class** Ship {

**private** **int** deckNum;

**private** **int** health;

**private** **int** Xcor;// cordinats of first field

**private** **int** Ycor;

**private** **int**[][] shipCor1;

**private** **int**[][] shipCor2 ;

**private** **boolean** horizontal;

Ship() {

}

Ship(**int** x, **int** y, **int** numOfDeck, **boolean** alighment) {

**this**.Xcor = x;

**this**.Ycor = y;

**this**.deckNum = numOfDeck;

**this**.health = numOfDeck-1;

**this**.horizontal = alighment;

//setShipCor(x, y,numOfDeck, alighment, field);

}

**public** **boolean** isDead() {

**if**(**this**.health >= 0)

**return** **false**;

**else**

**return** **true**;

}

/\*\*

\* **@return** the shipCor

\*/

**public** **int**[][] getShipCor1() {

**return** shipCor1;

}

/\*\*

\* **@return** the shipCor

\*/

**public** **int**[][] getShipCor2() {

**return** shipCor2;

}

/\*\*

\* **@param** shipCor the shipCor to set

\*/

**public** **void** setShipCor(**int** x, **int** y, **int** deckNum, **boolean** horizontal, **int** field) {

System.***out***.print(field +"в„– Field ");

**if**(field == 1) {

System.***out***.println("| DeckNum: "+deckNum+" Horizontal "+horizontal);

**if** (horizontal) {

shipCor1 = **new** **int**[x+1][y +deckNum];

**for** (**int** i = 0; i < deckNum; i++) {

shipCor1[x][y + i] = 1;

System.***out***.println("Coord: "+(x+1)+" "+ (**char**) ('A' + y+ i ));

}

}

**if** (!horizontal) {

shipCor1 = **new** **int**[x+deckNum][y +1];

**for** (**int** i = 0; i < deckNum; i++) {

shipCor1[x + i][y] = 1;

System.***out***.println("Coord: "+(x+i+1)+" "+(**char**) ('A' + y ));

}

}

}**else** **if** (field == 2) {

System.***out***.println("| DeckNum: "+deckNum+" Horizontal "+horizontal);

**if** (horizontal) {

shipCor2 = **new** **int**[x+1][y +deckNum];

**for** (**int** i = 0; i < deckNum; i++) {

shipCor2[x][y + i] = 1;

System.***out***.println("Coord: "+(x+1)+" "+ (**char**) ('A' + y+i ));

}

}

**if** (!horizontal) {

shipCor2 = **new** **int**[x+deckNum][y +1];

**for** (**int** i = 0; i < deckNum; i++) {

shipCor2[x + i][y] = 1;

System.***out***.println("Coord: "+(x+i+1)+" "+(**char**) ('A' + y ));

}

}

}

}

/\*\*

\* **@return** the health

\*/

**public** **int** getHealth() {

**return** health;

}

/\*\*

\* **@param** health

\* the health to set

\*/

**public** **void** setHealth(**int** health) {

**this**.health = health;

}

/\*\*

\* **@return** the xcor

\*/

**public** **int** getXcor() {

**return** Xcor;

}

/\*\*

\* **@param** xcor

\* the xcor to set

\*/

**public** **void** setXcor(**int** xcor) {

Xcor = xcor;

}

/\*\*

\* **@return** the ycor

\*/

**public** **int** getYcor() {

**return** Ycor;

}

/\*\*

\* **@param** ycor

\* the ycor to set

\*/

**public** **void** setYcor(**int** ycor) {

Ycor = ycor;

}

/\*\*

\* **@return** the horizontal

\*/

**public** **boolean** isHorizontal() {

**return** horizontal;

}

/\*\*

\* **@param** horizontal

\* the horizontal to set

\*/

**public** **void** setHorizontal(**boolean** horizontal) {

**this**.horizontal = horizontal;

}

/\*\*

\* **@return** the deckNum

\*/

**public** **int** getDeckNum() {

**return** deckNum;

}

/\*\*

\* **@param** deckNum

\* the deckNum to set

\*/

**public** **void** setDeckNum(**int** deckNum) {

**this**.deckNum = deckNum;

}

}

**import** java.io.File;

**import** java.io.IOException;

**import** javax.sound.sampled.AudioInputStream;

**import** javax.sound.sampled.AudioSystem;

**import** javax.sound.sampled.Clip;

**import** javax.sound.sampled.FloatControl;

**import** javax.sound.sampled.LineEvent;

**import** javax.sound.sampled.LineListener;

**import** javax.sound.sampled.LineUnavailableException;

**import** javax.sound.sampled.UnsupportedAudioFileException;

**public** **class** Sound **implements** AutoCloseable {

**private** **boolean** released = **false**;

**private** AudioInputStream stream = **null**;

**private** Clip clip = **null**;

**private** FloatControl volumeControl = **null**;

**private** **boolean** playing = **false**;

**public** Sound(File f) {

**try** {

stream = AudioSystem.*getAudioInputStream*(f);

clip = AudioSystem.*getClip*();

clip.open(stream);

clip.addLineListener(**new** Listener());

volumeControl = (FloatControl) clip.getControl(FloatControl.Type.***MASTER\_GAIN***);

released = **true**;

} **catch** (IOException | UnsupportedAudioFileException | LineUnavailableException exc) {

exc.printStackTrace();

released = **false**;

close();

}

}

// true if sound download, false if some trouble

**public** **boolean** isReleased() {

**return** released;

}

// if music is plaiyng now

**public** **boolean** isPlaying() {

**return** playing;

}

// Start

/\*

breakOld defines behavior if the sound is already playing

If breakOld == true, o the sound is interrupted and restarted

Otherwise nothing will happen

\*/

**public** **void** play(**boolean** breakOld) {

**if** (released) {

**if** (breakOld) {

clip.stop();

clip.setFramePosition(0);

clip.start();

playing = **true**;

} **else** **if** (!isPlaying()) {

clip.setFramePosition(0);

clip.start();

playing = **true**;

}

}

}

// the same as play(true)

**public** **void** play() {

play(**true**);

}

// stop playing

**public** **void** stop() {

**if** (playing) {

clip.stop();

}

}

**public** **void** close() {

**if** (clip != **null**)

clip.close();

**if** (stream != **null**)

**try** {

stream.close();

} **catch** (IOException exc) {

exc.printStackTrace();

}

}

//Setting the volume

/\*

\*x must be between 0 and 1 (from the quietest to the loudest)

\*/

**public** **void** setVolume(**float** x) {

**if** (x<0) x = 0;

**if** (x>1) x = 1;

**float** min = volumeControl.getMinimum();

**float** max = volumeControl.getMaximum();

volumeControl.setValue((max-min)\*x+min);

}

// Returns the current volume (number from 0 to 1)

**public** **float** getVolume() {

**float** v = volumeControl.getValue();

**float** min = volumeControl.getMinimum();

**float** max = volumeControl.getMaximum();

**return** (v-min)/(max-min);

}

// Waiting for the sound to stop playing

**public** **void** join() {

**if** (!released) **return**;

**synchronized**(clip) {

**try** {

**while** (playing)

clip.wait();

} **catch** (InterruptedException exc) {}

}

}

// Static method

**public** **static** Sound playSound(String path) {

File f = **new** File(path);

Sound snd = **new** Sound(f);

snd.play();

**return** snd;

}

**private** **class** Listener **implements** LineListener {

**public** **void** update(LineEvent ev) {

**if** (ev.getType() == LineEvent.Type.***STOP***) {

playing = **false**;

**synchronized**(clip) {

clip.notify();

}

}

}

}

}