Politechnika Świętokrzyska	
Studia stacjonarne (semestr letni)	
Projekt JAVA	**2022/2023**
Imię i nazwisko: Bartłomiej Wydrzycki Karol Górnicki	**Grupa:** 2ID12B
Temat projektu: Gra SAPER	

1. Co to za gra oraz na czym polega “Saper”?

Saper to klasyczna jednoosobowa gra komputerowa napisana w 1981 roku przez Roberta Donnera, dostępna jako akcesorium w każdym systemie Microsoft Windows do wersji 7.
Od wersji 8 i RT dostępne do ściągnięcia w sklepie Windows(istnieją też wersje dla innych systemów operacyjnych). Gra polega na odkrywaniu na planszy poszczególnych pól w taki sposób, aby nie natrafić na minę. Na każdym z odkrytych pól napisana jest liczba min, które bezpośrednio stykają się z danym polem (od zera do ośmiu). Jeśli oznaczymy dane pole flagą, jest ono zabezpieczone przed odsłonięciem, dzięki czemu przez przypadek nie odsłonimy miny.

Oryginalna wersja sapera na systemie Windows

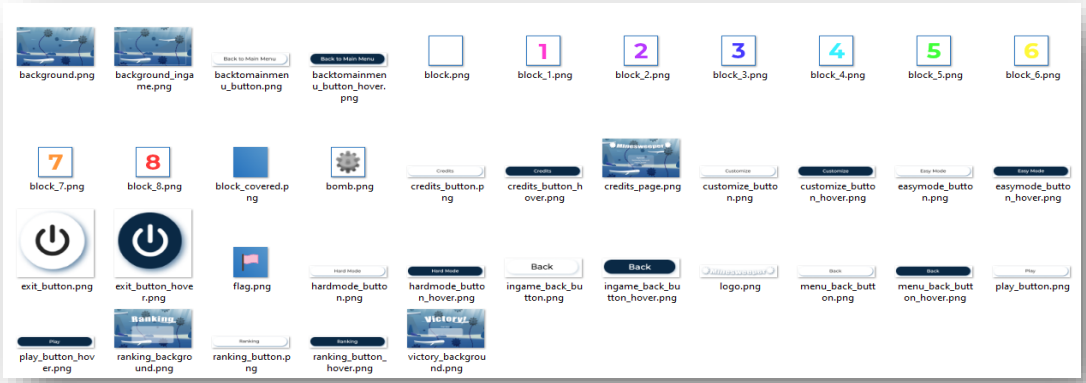
2. Przebieg prac nad projektem

- **Grafiki**

Prace nad projektem zaczęliśmy od zaprojektowania oraz wykonaniem potrzebnych grafik aby przyjemniej oraz płynniej można było implementować potrzebne algorytmy.

Grafiki zostały wykonane w programach: Figma, Adobe Illustrator, Adobe XD

Wszystkie wykonane grafiki znajdujące się w folderze 'img'



- **Menu**

Po ukończeniu grafik zabraliśmy się za implementację menu głównego. Menu składa się z 3 przycisków, którymi możemy poruszać się po aplikacji

Menu główne

Po naciśnięciu pierwszego przycisku “Play” aplikacja przekieruję nas do drugiego menu z którego możemy wybrać poziom trudności gry.

Easy mode - Plansza składająca się z 50 min (w ramach testu ustawiona jest jedna mina).

Hard mode - Plansza składająca się z 100 min.

Customize - Poziom, dzięki któremu możemy sami wybrać ilość bomb, która będzie wygenerowana na planszy.

Menu wyboru poziomu trudności

Możliwość wyboru ilości min

- **Algorytm generowania planszy**

Następnym krokiem było napisanie algorytmu, który będzie generował pożądaną ilość min w planszy o wymiarach 23x42.
Zajmuję się tym funkcja, która najpierw wypełnia tablicę zerami a następnie w losowych miejscach wstawia bomby.

Pętla wypełniająca całą tablicę zerami

```
for(int i = 0; i < size_row; i++)
{
    for(int j = 0; j < size_col;j++)
    {
        board[i][j] = 0;
        is_revealed[i][j] = 0;
        if_flag[i][j] = false;
    }
}
```

Pętla losująca miejsce w tablicy a następnie wstawiająca liczbę 9 co oznacza bombę

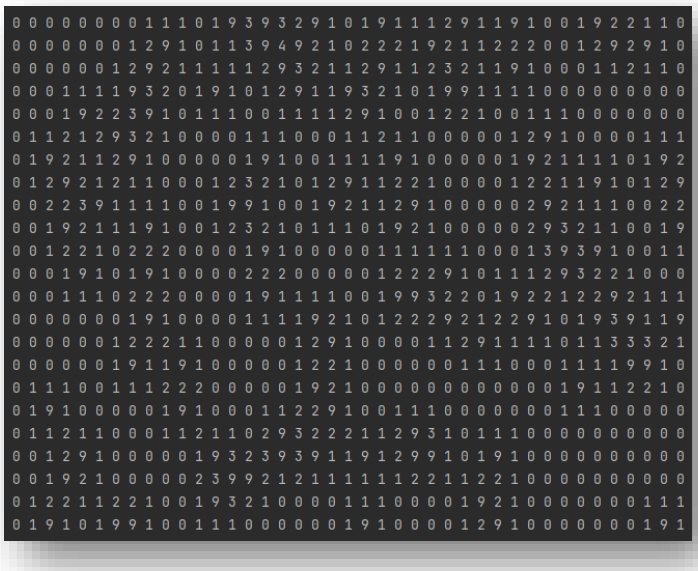
```
//Losowanie miejsc bomb w planszy
for(int i=0;i<bombs_number;i++) {
    random_row = random.nextInt((size_row));
    random_col = random.nextInt((size_col));
    if(board[random_row][random_col] == 9) {
        i--;
    }
    else {
        board[random_row][random_col] = 9;
    }

}
```

Ostatnią rzeczą, która brakowała aby algorytm działał prawidłowo, było zaimplementowanie funkcji która zlicza miny wokół każdego pola w tablicy. Funkcja zlicza najpierw pole które znajdują się w rogach planszy następnie zlicza krawędzie a na koniec cały środek.

Fragment funkcji zliczającej ilość min wokół pól

```
int count = 0;
//Lewy gorny rog
if(board[0][0]!=9)
{
    if(board[0][1]==9)
    {
        count++;
    }
    if(board[1][1]==9)
    {
        count++;
    }
    if(board[1][0]==9)
    {
        count++;
    }
    board[0][0] = count;
    count = 0;
}
```

Po skończeniu pisania algorytmu wyświetlamy zawartość planszy w konsoli



- ## Generowanie planszy na ekranie

Do wygenerowania planszy na ekranie używamy funkcji repaint, która sprawdza dla każdego pola, który klocek wyświetlić w danym polu.

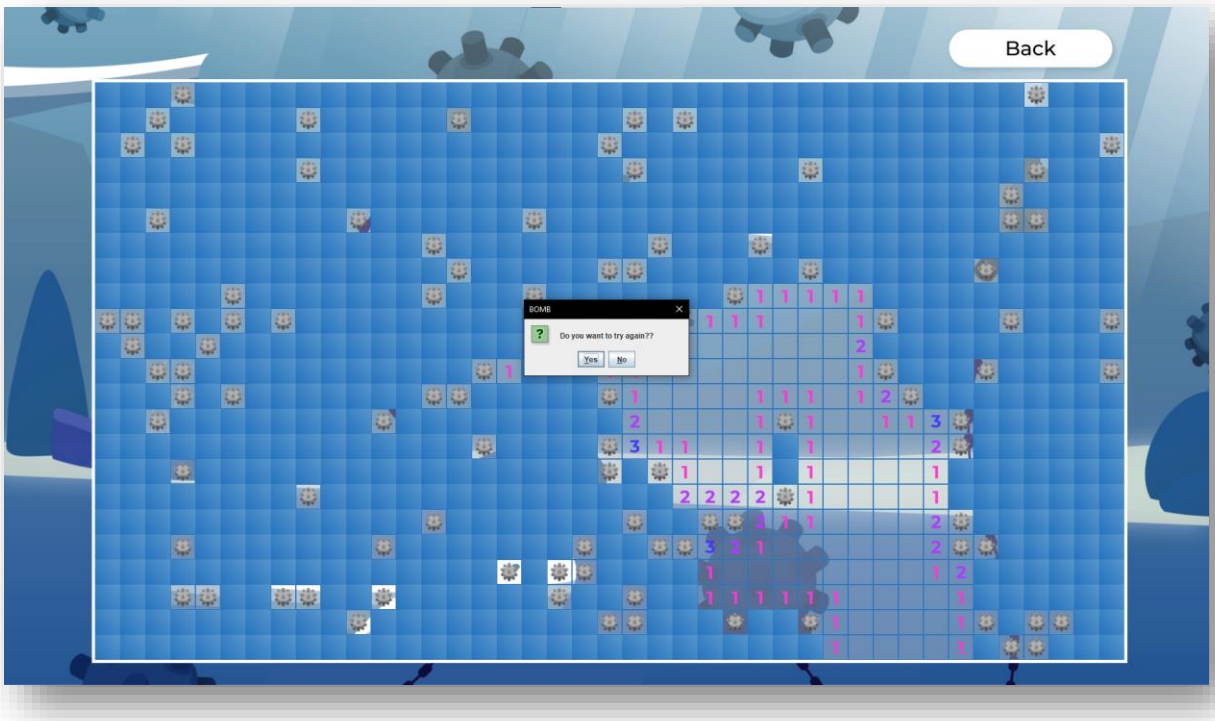
Jest parę warunków, które określają jaki klocek wyświetlić.
To jest na przykład:

Tablica is_revealed – Określa czy pole zostało już odkryte przez użytkownika.
Tablica if_flag - Określa czy na polu znajduję się flaga.

Ostatnim warunkiem który jest sprawdzany, to jaka cyfra znajduję się na polu, które zostało odkryte.

Jeżeli ta cyfra to 9 (co oznacza bombę) to gra w takim wypadku jest przerywana oraz wyświetla się komunikat, czy użytkownik chcę spróbować jeszcze raz.
Jeżeli tak to plansza zostanie wygenerowana od początku i gra się zacznie od nowa.
Jeżeli nie to aplikacja wróci do menu głównego.

Aplikacja sprawdza po każdym kliknięciu koordynaty X oraz Y które potrzebne są do sprawdzenia miejsca kliknięcia w tablicy.



- **Funkcja odkrywająca sąsiednie pola**

Bardzo ważnym algorytmem do prawidłowego działania gry saper jest odkrywanie pól, które sąsiadują także z pustymi polami.
Na przykład jeżeli użytkownik kliknie w pole które w tablicy jest równe 0 to wtedy wszystkie pola równe 0-8 zostaną także odkryte, jednak od każdego pola równego 0 wywołuję się także rekurencja tej samej funkcji.
Dzięki temu można uzyskać efekt odsłania nawet po kliknięciu w środek planszy.

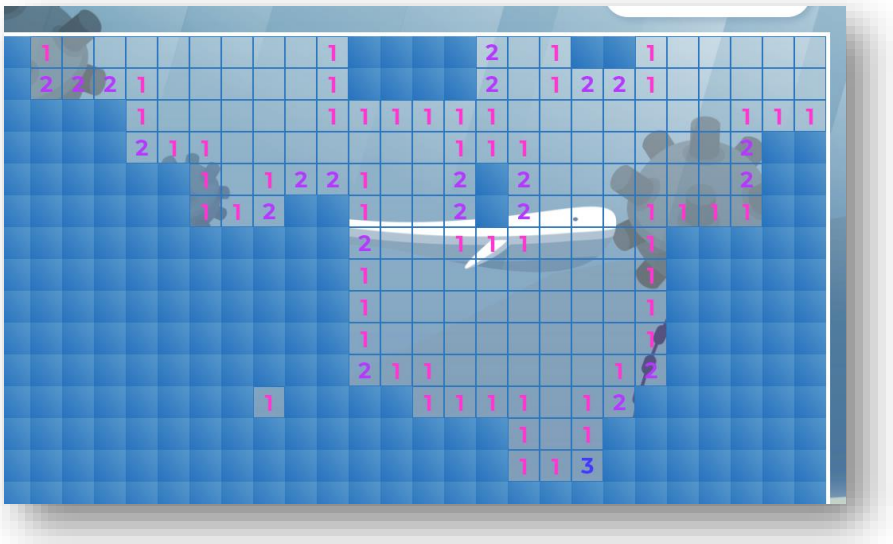
Funkcja revealing służąca do odkrywania pól

```
static void revealing(final int[][] board, double x_p
    if(i<0 || j<0 || i>size_row-1 || j>size_col-1)
    {
        return;
    }
    if(!if_flag[i][j])
    {
        if(board[i][j]==1)
            is_revealed[i][j]=1;
        if(board[i][j]==2)
            is_revealed[i][j]=1;
        if(board[i][j]==3)
            is_revealed[i][j]=1;
        if(board[i][j]==4)
            is_revealed[i][j]=1;
        if(board[i][j]==5)
            is_revealed[i][j]=1;
        if(board[i][j]==6)
            is_revealed[i][j]=1;
        if(board[i][j]==7)
            is_revealed[i][j]=1;
        if(board[i][j]==8)
            is_revealed[i][j]=1;
    }
    else
    {
        is_revealed[i][j]=0;
    }
```

Rekurencja zawarta w funkcji revealing

```
if(is_revealed[i][j] == 0 && board[i][j]==0)
{
    if(!if_flag[i][j])
        is_revealed[i][j] = 1;
    revealing(board,x_pom, y_pom: y_pom-space, i: i-1,j,space,size_row,size_col,is_revealed,if_flag);
    revealing(board,x_pom, y_pom: y_pom+space, i: i+1,j,space,size_row,size_col,is_revealed,if_flag);
    revealing(board, x_pom: x_pom-space,y_pom,i, j: j-1,space,size_row,size_col,is_revealed,if_flag);
    revealing(board, x_pom: x_pom+space,y_pom,i, j: j+1,space,size_row,size_col,is_revealed,if_flag);
    revealing(board, x_pom: x_pom-space, y_pom: y_pom-space, i: i-1, j: j-1,space,size_row,size_col,is_revealed,if_flag);
    revealing(board, x_pom: x_pom+space, y_pom: y_pom+space, i: i+1, j: j+1,space,size_row,size_col,is_revealed,if_flag);
    revealing(board, x_pom: x_pom+space, y_pom: y_pom-space, i: i-1, j: j+1,space,size_row,size_col,is_revealed,if_flag);
    revealing(board, x_pom: x_pom-space, y_pom: y_pom+space, i: i+1, j: j-1,space,size_row,size_col,is_revealed,if_flag);
}
```

Przykład działania funkcji odkrywającej



- **System działania flag**

W momencie, gdy użytkownik kliknie prawy przycisk na planszy to wywołuje się funkcja, która najpierw sprawdza czy dane pole zostało wcześniej odkryte. Jeżeli nie to kolejny warunek sprawdza, czy na danym polu znajdowała już wcześniej flaga. Jeżeli się znajdowała to flaga zostaje zdjęta jednak jeżeli jej nei było to zostaje nałożona.

Funkcja systemu działania flag

```
if(e.getButton() == MouseEvent.BUTTON3) {
    double mouseX = MouseInfo.getPointerInfo().getLocation().getX();
    double mouseY = MouseInfo.getPointerInfo().getLocation().getY();
    double x_pom = 145,y_pom = 120, space = 40;
    for(int i=0;i<size_row;i++) {
        for(int j=0;j<size_col;j++) {
            if(mouseX>x_pom && mouseX<x_pom+space && mouseY>y_pom && mouseY<y_pom+space) {
                if(is_revealed[i][j] == 0) {
                    if(!if_flag[i][j]) {
                        if_flag[i][j] = true;
                        if(board[i][j] == 9) {
                            correct_bombs_flagged[0] +=1;
                        }
                        else {
                            correct_bombs_flagged[0] -= 1;
                        }
                    }
                    else {
                        if_flag[i][j] = false;
                        if(board[i][j] == 9) {
                            correct_bombs_flagged[0] -=1;
                        }
                        else {
                            correct_bombs_flagged[0] +=1;

                        }
                    }
                }
```

- **Wygrana gracza**

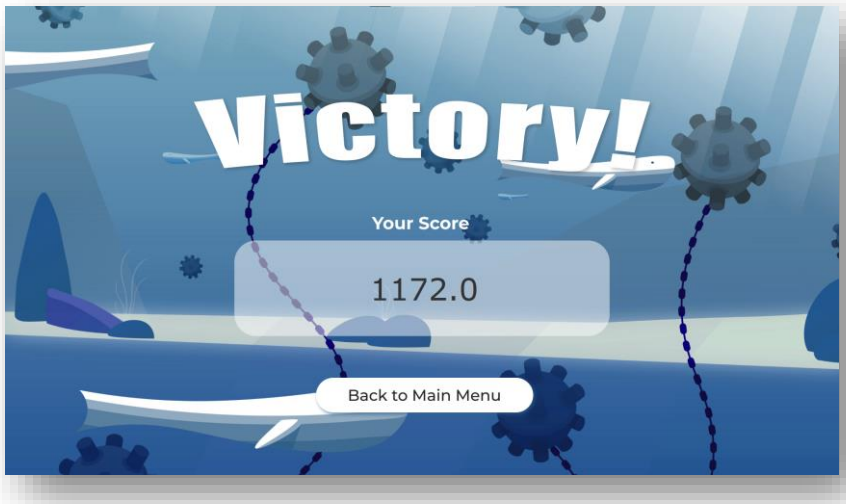
Wygrana gracza następuje w momencie, kiedy użytkownik oflaguje wszystkie pola na którym znajdują się flagi.
W momencie kiedy użytkownik oflaguję pola na których nie znajdują się bomby to wtedy gra się nie zakończy.

Kiedy wygrana nastąpi, to gra się przerywa oraz użytkownik zostaje przeniesiony na ekran końcowy, który pokazuje końcowy wynik który jest obliczany za pomocą ilości bomb, które użytkownik miał na planszy, oraz czas jaki zajął użytkownikowi.

Fragment kodu odpowiedzialnego za wykrywanie wygranej użytkownika

```
if(correct_bombs_flagged[0] == bombs_number) {
    System.out.print("\n WYGRANA \n");
    //Tworzenie ekranu końcowego
    double executionTime = (System.currentTimeMillis() - millisActualTime)/1000; // czas wykonania programu w ekundach
    frame.getContentPane().removeAll();
    frame.removeMouseListener( l: this);
    Win win_scr = new Win();
    try {
        win_scr.scoreSaving(executionTime,frame,bombs_number);
    } catch (FileNotFoundException ex) {
        ex.printStackTrace();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    System.out.print("\n PO FUNKCJI \n");
    return;
}
```

Ekran końcowy z wynikiem gracza



- **Ranking**

W momencie działania funkcji odpowiedzialnej za wygraną użytkownika wynik jest zapisywany do pliku ranking.txt

Obliczanie wyniku gracza oraz wprowadzanie go do pliku za pomocą FileWriter

```
executionTime = executionTime*bombs_number*1000;

executionTime = (int) executionTime;

FileOperations operations = new FileOperations();
operations.save( name: "ranking.txt", (int) executionTime);
```

Klasa odpowiedzialna za zapis do pliku

```
class FileOperations{
    public void save(String name, int number) throws IOException {
        FileWriter fstream = new FileWriter(name, append: true);
        BufferedWriter out = new BufferedWriter(fstream);
        out.write( str: number + "\n");
        out.close();
    }
}
```

Fragment pliku „ranking.txt”

```
0
1952
2690
1172
```

Aby wyświetlać ranking najlepszych wyników w saperze napisaliśmy klasę, która po kliknięciu przycisku „Ranking” w menu głównym tworzy listę, sortuje w niej najlepsze wyniki, a następnie wypisuje na ekranie osiem najlepszych wyników uzyskanych przez wszystkich użytkowników.

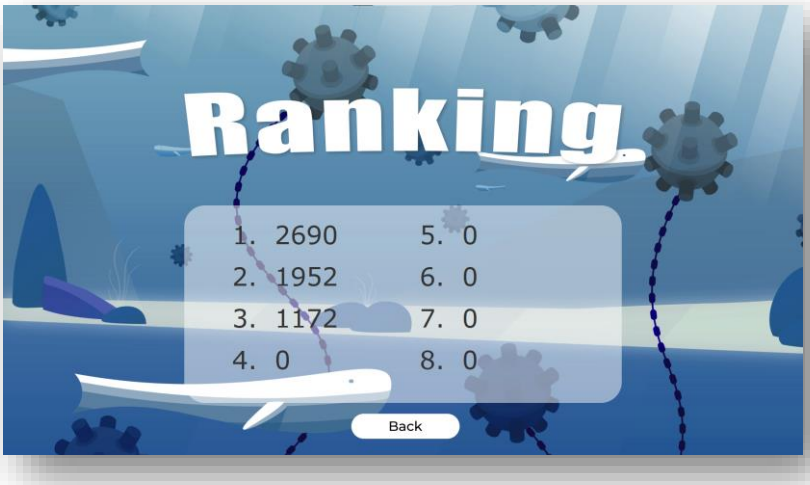
Fragment, który tworzy nową listę, wywołuję funkcję służącą za odczytywanie wyników z pliku do listy oraz sortowanie tej listy.

```
List<Integer> scoreList = new ArrayList<Integer>();
Ranking ranking2 = new Ranking();
ranking2.read( name: "ranking.txt",scoreList);
Collections.sort(scoreList);
```

Odczytywanie wyników do listy

```
public void read(String name, List<Integer> scoreList) throws FileNotFoundException {
    System.out.println("Odczytuje z pliku");

    File myObj = new File(name);

    Scanner myReader = new Scanner(myObj);
    while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
        scoreList.add(Integer.valueOf(data));
    }
    myReader.close();
}
```

Ekran z najlepszymi wynikami użytkowników.



- **Credits**

Ostatnią rzeczą, jaką zaimplementowaliśmy była „zakładka” Credits w menu głównym, która pokazuję członków zespołu odpowiedzialnych za stworzenie gry.

- **Wnioski**

Wszystkie cele z naszych początkowych założeń zostały wykonane.