

Programowanie w języku Java (Projekt)

Temat: Gra Saper

Grupa: 2ID13A

Zespół: **Daniel Rogowski, Adrian Rubak**

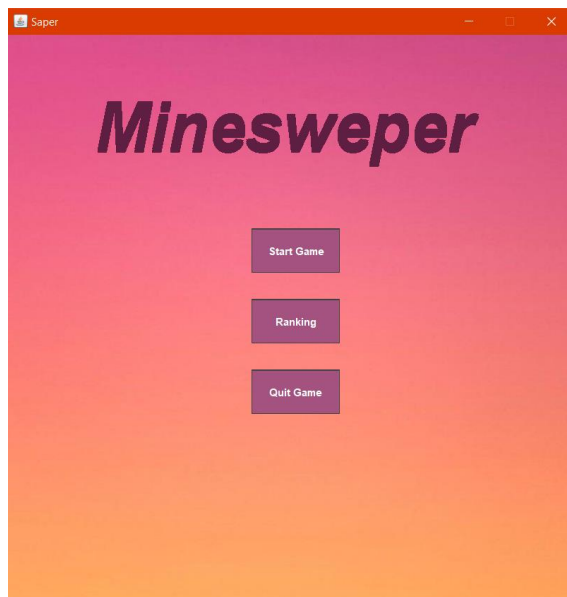
Opis:

Gra Saper- Zadaniem gracza jest odkrycie największej pól, tak aby nie trafić na bombę. Gracz wygrywa w momencie gdy wszystkie możliwe pola są odkryte, a na potencjalnych polach z bombą postawione są flagi. Gracz ma możliwość wybrania odpowiedniego dla siebie pola w postaci kwadratu do gry. Zakres pola wynosi od 36 do 625 pól (gdzie każda wartość pola jest pierwiastkowana tak aby mógł być zbudowany kwadrat. Na przykładzie pola 36 powstanie kwadrat o wymiarach 6x6). Wyżej wspomniane bomby mogą występować w różnych ilościach – gracz decyduje ile bomb chce dodać do swojej rozgrywki. Aby było sprawiedliwe to ilość bomb jest równa ilości flag, tak aby była możliwość wygrania gry. W przypadku wybrania przez gracza konkretnego pola oraz zerowej ilości bomb, to gracz po jednym kliknięciu w obojętnie jakie pole wygrywa grę.

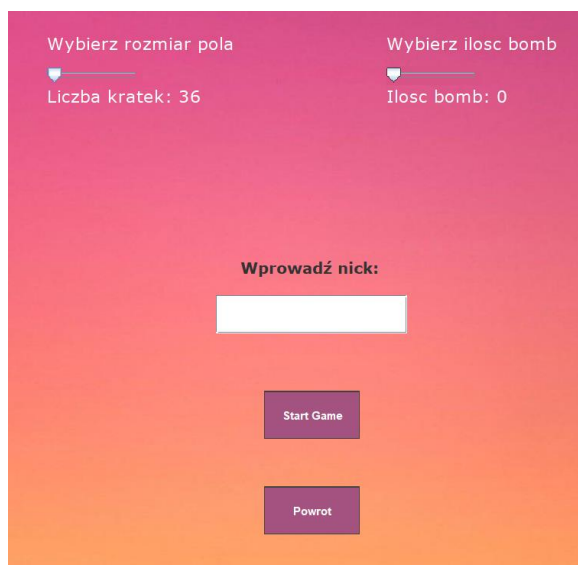
W grze wykorzystujemy dwa przyciski:

- Lewy przycisk myszy – ten przycisk oprócz poruszania się po aplikacji – m.in. uruchomieni rozgrywki czy sprawdzenie rankingu używamy do rozgrywki. Tym przyciskiem wybieramy wybrane przez gracza pole do sprawdzenia – możemy na trafić na puste pole (czyli takie, które nie kończy grę lub bombę – która zakańcza rozgrywkę).
- Prawy przycisk myszy – jest używany tylko w rozgrywce. Służy do stawiania flag na konkretnym polu, gdzie potencjalnie znajduje się bomba.

Menu gry:

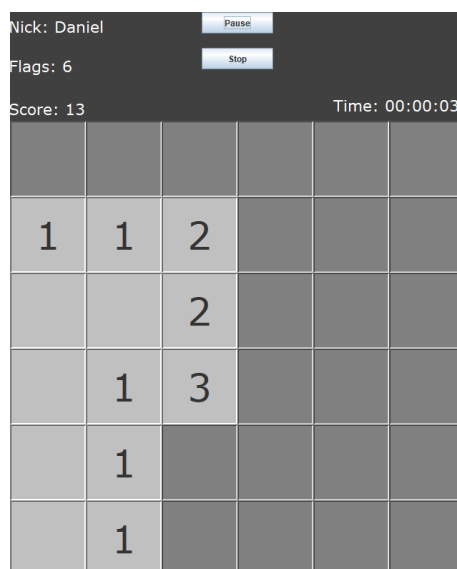
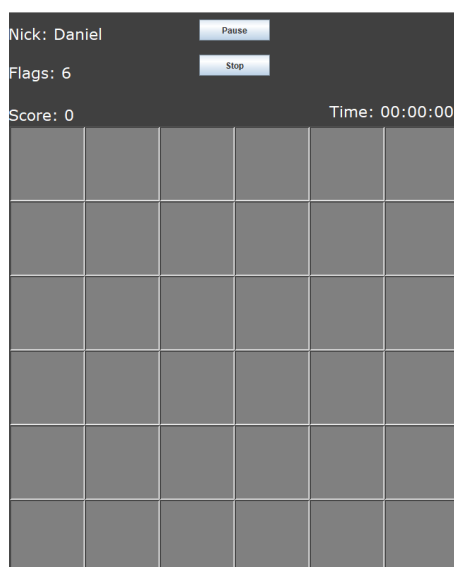


Wybór pola, nicku (imienia w grze) oraz ilości bomb:



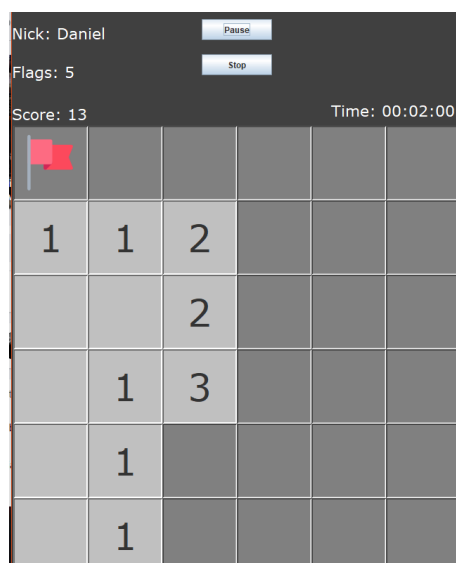
Naciskając **Start Game** bez wpisanego nicku zostanie wykonana klauzula sprawdzająca, czy użytkownik podał nick, w przypadku gdy nie poda zostanie wyświetlony poniższy komunikat:

Pole gry:

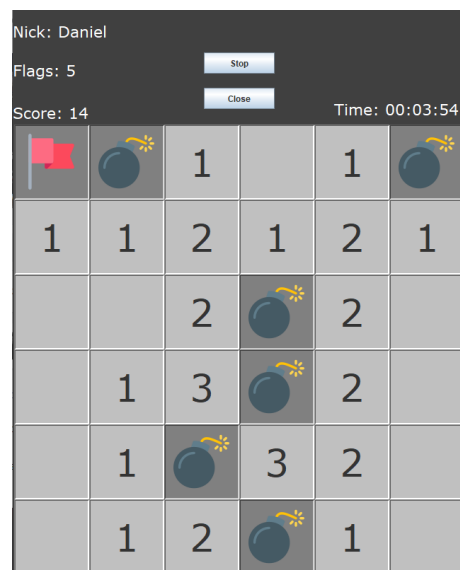
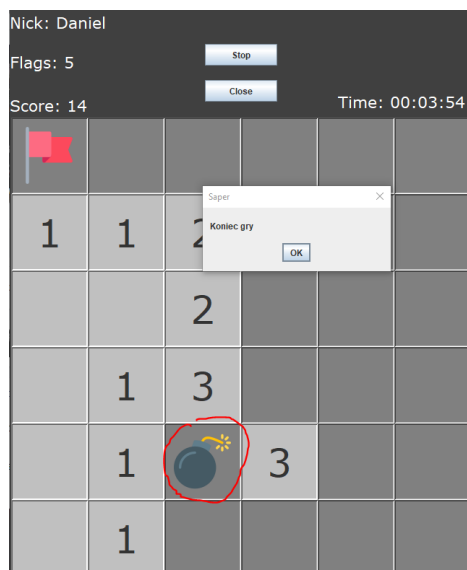


Ciemno szare pole oznacza miejsce nie odkryte. Natomiast jasno szare pole oznacza miejsce odkryte. Liczby na polach informują nas o otaczających bombach. W przypadku gdy spodziewamy się na danym polu bomby stawiamy flagę.

Postawienie flagi:

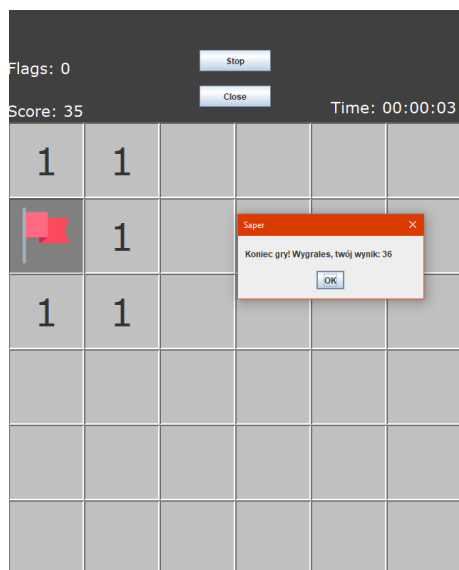


Natrafienie na bombę:

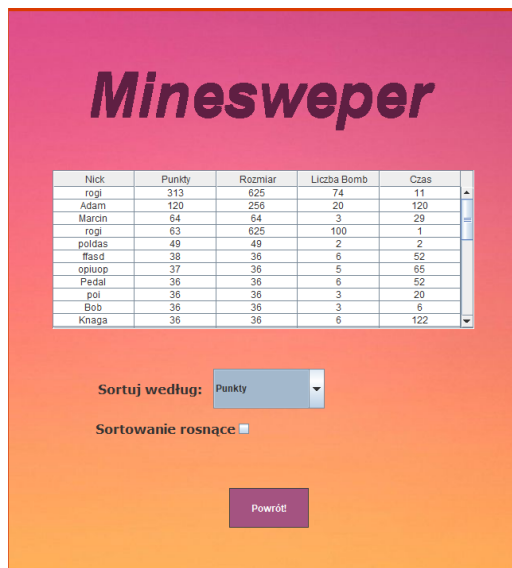


Natrafienie na bombę (czerwony zaznaczony fragment) powoduje wyświetlenie się alertu o końcu gry. Po wciśnięciu **ok** odkrywana jest cała plansza wraz z bombami. Wynik gracza zostanie zapisany i zostanie przekazany do Rankingu.

Przykładowa wygrana rozgrywka:



Ranking:



Nick	Punkty	Rozmiar	Liczba Bomb	Czas
rog	313	625	74	11
Adam	120	256	20	120
Marcin	64	64	3	29
rog	63	625	100	1
poldas	49	49	2	2
ffasd	38	36	6	52
opluop	37	36	5	65
Pedat	36	36	6	52
pol	36	36	3	20
Bob	36	36	3	6
Knaga	36	36	6	122

Sortuj według: Punkty

Sortowanie rosnące ☐

Powrót!

Ranking pokazuje najlepszych graczy wraz z ilością punktów, rozmiarem pola, liczbą bomb oraz czasem. Jest możliwość sortowania wyniku według różnych kryteriów, m.in. : Punkty, rozmiar planszy, liczba bomb czy czas.

Środowisko:

Projekt był wykonywany w środowisku programistycznym Javy IntelliJ 2020.3.2 przy użyciu biblioteki Java Swing FX oraz do płynnego przesyłania poprawek między zespołem i koordynowania działań zostało użyte repozytorium Git.

Uruchomienie projektu:

Należy zaimportować projekt z Git (link: <https://github.com/rogowskid/SaperJava.git>) - repozytorium jest publiczne. Następnie włączyć klasę **Main** i uruchomić ją. Lub po pobraniu repozytorium wejść w folder **targert** a następnie uruchomić plik .jar

Maven

Projekt został stworzony w oparciu o Maven, w którym wykorzystaliśmy zależności: **junit-jupiter-engine** oraz **batik-swing**. Pierwsza zależność posłużyła nam do napisania kilku prostych testów dla klasy rankingu. Batik-swing wykorzystaliśmy aby wykorzystać grafikę **SVG** w celu zachowania responsywności w panelu gry.

Najciekawsze funkcjonalności

1. Wykorzystanie tokenizacji stringów

W celu zapisywania rankingu po zamknięciu gry oraz zachowaniu jego stanu wykorzystaliśmy tokenizację stringów. Ranking jest zapisywany do pliku pojedyncze właściwości danego rekordu są oddzielane z wykorzystaniem symbolu „;”. Funkcjonalność tą zrealizowaliśmy z wykorzystując klasę **StringTokenizer**.

```
public static RankingElement deTokenize(String line, String separator){
    RankingElement tmp = new RankingElement();
    StringTokenizer tokenizer = new StringTokenizer(line, separator);
    tmp.setUsername(tokenizer.nextToken());
    tmp.setScore(Integer.parseInt(tokenizer.nextToken()));
    tmp.setBoardSize(Integer.parseInt(tokenizer.nextToken()));
    tmp.setNumberOfBombs(Integer.parseInt(tokenizer.nextToken()));
    tmp.setSeconds(Integer.parseInt(tokenizer.nextToken()));
    return tmp;
}
```

2. Wykorzystanie rekurencji

W algorytmie odkrywającym sąsiednie „puste” pola wykorzystaliśmy rekurencję. Ponieważ maksymalna liczba pól wynosi 625 zdecydowaliśmy się na to rozwiązanie, gdyż rekurencja w znaczny sposób ułatwia napisanie tej funkcjonalności.

Opis algorytmu:

Krok 1. Sprawdź czy pole ma flagę lub jest oznaczone flagą

Jeśli **tak** zakończ działanie funkcji.

Jeśli **nie** idź do krok 2.

Krok 2. Oznacz pole jako odwiedzone, odkryj pole. Idź do Krok 3

Krok 3. Sprawdź czy wartość pola (liczba bomb wokół) różna od zera

Jeśli **tak** zakończ działanie funkcji .

Jeśli **nie** idź do krok 4.

Krok 4. Sprawdź sąsiednie pola

```

public void selectEmptyFields(int index){
    if(fields[index].isChecked() || fields[index].isHasFlag()){
        return;
    }
    fields[index].setChecked(true);
    fields[index].drawField();
    if(fields[index].getValue() != 0){
        return;
    }
    //Po prawej
    if( (index+1 < fields.length) && !isInLastColumn(index, fields.length))
        selectEmptyFields( index: index+1);
    //Po lewej
    if((index % numberOfFieldsSqrt) != 0)
        selectEmptyFields( index: index-1);
    //Dół
    if((index + numberOfFieldsSqrt) < fields.length)
        selectEmptyFields( index: index + numberOfFieldsSqrt);
    //Góra
    if((index - numberOfFieldsSqrt) >= 0)
        selectEmptyFields( index: index - numberOfFieldsSqrt);
    //Dół prawo
    if((index + numberOfFieldsSqrt+1) < fields.length && !isInLastColumn(index, fields.length))
        selectEmptyFields( index: index + numberOfFieldsSqrt + 1);
    //Dół lewo
    if(index % numberOfFieldsSqrt != 0 && (index + numberOfFieldsSqrt-1) < fields.length)
        selectEmptyFields( index: index + numberOfFieldsSqrt - 1);
    //Góra prawo
    if((index - numberOfFieldsSqrt+1) >= 0 && !isInLastColumn(index, fields.length))
        selectEmptyFields( index: index - numberOfFieldsSqrt + 1);
    //Góra lewo
    if((index - numberOfFieldsSqrt-1) >= 0 && index % numberOfFieldsSqrt != 0)
        selectEmptyFields( index: index - numberOfFieldsSqrt - 1);
}

```

3. Wykorzystanie grafiki SVG

W naszym projekcie gracz może wybrać różną liczbę pól, dlatego aby w zależności od wybranej liczby pól, potrzebowaliśmy responsywnej grafiki dlatego zdecydowaliśmy się na wykorzystanie właśnie grafiki SVG (do oznaczenia pola flagą oraz wyświetlenia bomby). Poprzez skorzystanie z zależności batik-swing.

```

<dependency>
<groupId>batik</groupId>
<artifactId>batik-swing</artifactId>
<version>1.6-1</version>
</dependency>

```



```
private void drawBomb(){
    svgCanvasBomb.setURI(urlImageBomb.toString());
    thisField.setLayout(null);
    svgCanvasBomb.setLocation( x: (thisField.getWidth() / 4) / 2, y: (thisField.getHeight() / 4) / 2);
    svgCanvasBomb.setSize((int) (thisField.getWidth() * 0.75), (int) (thisField.getHeight() * 0.75));
    thisField.add(svgCanvasBomb);
}
```

Ikonki bomby oraz flagi pobraliśmy ze strony: <https://www.flaticon.com/>

Krótki opis klas i metod w projekcie

ButtonPainter – klasa odpowiadająca za zmianę wyglądu przycisku

- **Jbutton paintButton(JButton button)** - edytuje wygląd przycisku, jako parametry przyjmuje przycisk, którego wygląd będzie edytowany. Zwraca referencje do nowego przycisku

Field – klasa reprezentująca pojedyncze pole na planszy

- **void drawField()** - rysuje pole na planszy
- **int getValue()** - zwraca wartość pola (ilosc bomb wokół danego pola). Zwraca wartość danego pola
- **void setValue(int value)** – ustawia wartość danego pola, jako parametr przyjmuje wartość
- **boolean isBomb()** - sprawdza czy dane pole jest bombą. Zwraca true jeśli pole jest bombą lub false jeśli nie jest bombą.
- **void setBomb(boolean bomb)** – ustawia czy dane pole jest bombą, jako parametr przyjmuje true lub false
- **boolean isHasFlag()** - sprawdza czy pole jest oznaczone flagą. Zwraca true jeśli pole jest flagą lub false jeśli nie jest flagą.
- **void setHasFlag(boolean hasFlag)** – ustawia czy pole ma flage, jako parametr przyjmuje true lub false
- **boolean isCanBeBomb()** - sprawdza czy pole może być bomba. Zwraca true jeśli pole może być bombą lub false jeśli nie może być.
- **void setCanBeBomb(boolean canBeBomb)** – ustawia czy pole może być bombą, jako parametr przyjmuje true lub false

- **int getIntdex()** - zwraca index danego pola z planszy. Zwraca index danego pola z tablicy
- **void setChecked(boolean checked)** – ustawia czy pole było odwiedzone, jako parametr przyjmuje true lub false
- **boolean isChecked()** - sprawdza czy pole było wcześniej odwiedzone
- **void drawBomb()** - rysuje bombe na polu
- **void drawFlag()** - rysuje flage na polu

GamePanel – klasa reprezentująca panel gry

- void generateBombs(Field[] fields, int number_of_bombs)** – generuje bomby na planszy, jako argumenty przyjmuje tablice pól oraz liczbę bomb do rozstawienia na planszy
- **int checkNeighbor(Field[] fields, int index)** – sprawdza ile bomb jest wokół danego pola, jako argumenty przyjmuje tablice pól oraz index pola w tej tablicy. Zwraca ile bomb jest wokół danego pola
- **boolean isLastColumn(int index, int length)** – sprawdza czy dany index jest w ostatniej kolumnie planszy, jako argumenty przyjmuje index danego pola i ilość wszystkich pól. Zwraca true jeśli pole jest w ostatniej kolumnie lub false jeśli nie jest w ostatniej.
- **boolean isGameOver()** - sprawdza czy gra została zakończona. Zwraca true jeśli gra została zakończona lub false jeśli nie została zakończona
- **void setGameOver(boolean gameOver)** – ustawia czy gra ma zostać zakończona
- **void setCloseGame(boolean check)** – ustawia czy gra została zakończona
- **void firstClick(int index)** – wykonuje funkcjonalności po pierwszym kliknięciu, jako argument przyjmuje index pola w które kliknięto podczas pierwszego kliku
- **boolean isAfterFirstClick()** - sprawdza czy nastąpiło już pierwsze kliknięcie. Zwraca true jeśli nastąpiło już pierwsze kliknięcie lub false jeśli nie nastąpiło.
- **void selectEmptyFields(int index)** – funkcja rekurencyjna odkrywająca sąsiednie pola, jako argument przyjmuje index od którego funkcja rozpoczyna odkrywanie kolejnych pól na planszy.
- **void drawAllFields()** - odkrywa wszystkie pola
- **void setAfterFirstClick(boolean afterFirstClick)** – ustawia czy nastąpiło już pierwsze kliknięcie, jako argument przyjmuje wartość true or false.

Main – klasa zawierająca funkcję main

- **void main(String[] args)** – funkcja główna main

MainFrame – okno główne programu

MenuPanel – klasa reprezentująca panel w którym znajduje się menu gry

- **void paintComponent(Graphics g)** – odpowiada za ustawienie wyglądu panelu, jako argument przyjmuje obiekt grafiki

Ranking – klasa reprezentująca ranking graczy

- **void addElement(RankingElement rankingElement)** - dodaje element do rankingu. Jako argument przyjmuje element do dodania.
- **void loadFromFile(String filename)** – ładuje ranking z określonego pliku, jako argument przyjmuje nazwę określonego pliku z którego zostanie odczytany ranking
- **FileWriter saveToFile(String fileName)** – zapisuje ranking do pliku, jako argument przyjmuje nazwę pliku do którego zostanie zapisany ranking. Zwraca referencje do pliku w którym zapisano ranking.
- **void selectionSort(int option, boolean growing)** – sortuje ranking, jako argumenty przyjmuje opcje sortowania oraz flagę informującą o kierunku sortowania (rosnące lub malejące)
- **void swap(int indexA, int indexB)** – zamienia dwa elementy rankingu
- **List<RankingElement> getListRankingowa()** - zwraca listę z rankingiem graczy. Zwraca listę rankingową

RankingElement – klasa reprezentująca pojedynczy element rankingu

- **void setUsername(String username)** – ustawia nazwę użytkownika, jako argument przyjmuje nazwę użytkownika
- **void setScore(int score)** – ustawia ilość punktów, które zdobył gracz, jako argument przyjmuje ilość punktów
- **void setBoardSize(int boardSize)** – ustawia rozmiar planszy na której grał gracz, jako argument przyjmuje rozmiar planszy
- **void setNumberOfBombs(int numberOfBombs)** – ustawia ilość bomb, które gracz miał na planszy, jako argument przyjmuje liczbę bomb
- **void setSeconds(int seconds)** – ustawia liczbę czasu który zdobył gracz, jako argument przyjmuje liczbę sekund (ile trwała rozgrywka)
- **RankingElement deTokenize(String line, String separator)** – detokenizuje pojedynczy wiersz z pliku, jako argumenty przyjmuje linie z pliku oraz znak separatora. Zwraca z detokenizowany element rankingu.
- **String tokenize(RankingElement rankingElement)** – tokenizuje pojedynczy element rankingu, jako argument przyjmuje element rankingu. Zwraca ztokenizowany element rankingu w Stringu.
- **int getScore()** - zwraca liczbę punktów które posiada dany gracz

- **int getBoardSize()** - zwraca rozmiar planszy na której grał gracz
- **int getNumberOfBombs()** - zwraca liczbę bomb z planszy na której grał gracz
- **int getSeconds()** - zwraca czas zdobyty przez gracza
- **String getUsername()** - zwraca nazwę danego użytkownika.

RankingPanel – klasa reprezentująca panel, który wyświetla ranking

- **void sortTable(String option, boolean flag)** – sortuje tabele z rankingiem, jako argumenty przyjmuje opcje sortowania i flagę jako kierunek sortowania
- **void initializeRankingTable()** - inicjalizuje tabele z rankingiem
- **void initializeScrollPane()** - odświeża widok tabeli
- **void paintComponent(Graphics g)** – wyświetla napis Minesweeper oraz rysuje tło, jako argument przyjmuje obiekt grafiki

StartGamePanel – klasa reprezentująca panel w którym wybieramy liczbę bomb oraz wpisujemy nick

- **void paintComponent(Graphics g)** – rysuje tło, jako argument przyjmuje obiekt grafiki

Stopwatch – klasa reprezentująca odmierzony czas

- **int getTimeInSeconds()** - zwraca czas gry w sekundach.

Podział prac

Adrian Rubak	Daniel Rogowski
Implementacja ranking graczy (klasy Ranking, RankingElement, tokenizacja stringów, napisanie kilku prostych testów)	Implementacja interfejsu graficznego (Ramka główna, MenuPanel, StartGamePanel)
Implementacja panelu ranking, wykorzystanie algorytmu SelectionSort w celu sortowania rankingów. Wyświetlanie rankingów w tabeli, wykorzystanie klasy Jtable i ScrollPane.	Implementacja klasy Stopwatch (obliczanie czasu w grze) oraz zakończenie gry
Implementacja rozgrywki: algorytmu odkrywania pól, interfejs panelu gry.	Funkcjonalność przechodzenia między oknami (różnymi panelami)
Wykorzystanie grafiki SVG w celu zachowania responsywności dla różnej ilości pól.	Implementacja funkcjonalności w StartGamePanel. (warunki rozpoczęcia gry, wybór ilości bomb i rozmiaru planszy, wprowadzenie nazwy gracza)
Dokumentacja projektu	
Sprawozdanie	