PROJEKT WYKONAŁ: MATEUSZ MOGIELNICKI
GRUPA PS: 9
INDEX: 110879

DOKUMENTACJA – BATTLESHIPS

OPIS PROJEKTU

Głównym celem projektu jest stworzenie interaktywnej wersji gry "Battleships", znanej również jako "Statki". Gra będzie dostępna w formie aplikacji okienkowej, korzystającej z interfejsu graficznego (oraz tekstowego jeśli gracz będzie chciał zagrać w takim interfejsie). Realizacja projektu odbyła się w języku Java, z wykorzystaniem biblioteki graficznej Java Swing. Wprowadzenie gry do świata interaktywnego pozwoli na przekształcenie klasycznej rozgrywki planszowej w emocjonujące doświadczenie wirtualne.

Główne założenia

- ***Cel projektu:*** Przeniesienie klasycznej gry planszowej "Battleships" do formy interaktywnej w aplikacji okienkowej z użyciem interfejsu graficznego i biblioteki Swing.

- ***Istota aplikacji:*** Umożliwienie graczom planowania taktyki, rozmieszczania statków oraz skutecznego strzelania w celu zniszczenia floty przeciwnika, przy jednoczesnym wykorzystaniu prostego i intuicyjnego interfejsu tekstowego.

OPIS FUNKCJONALNOŚCI

1. Gra przeciwko Komputerowi:
   - Gracze mają opcję rozpoczęcia rozgrywki przeciwko komputerowi, zaprogramowanego poprzez programistę.
   - Gracze będą mieli możliwość interaktywnego rozmieszczania swoich statków na planszy przed rozpoczęciem rozgrywki. Za pomocą prostego interfejsu graficznego będą wybierać pozycje i orientacje statków poprzez klikanie w odpowiednie kontrolki (guziki) na ekranie.
   - Komputer będzie prowadził swoją flotę, a gracz będzie musiał zastosować swoje umiejętności taktyczne, aby go pokonać.
   - Gra umożliwia na zmianę wykonywanie ruchów, strzelając w konkretne pola na planszy przeciwnika. Po każdym strzale, gra poinformuje gracza, czy trafienie było udane, czy też nie.
   - Gracz, który jako pierwszy zatopi wszystkie statki przeciwnika, zostanie uznany za zwycięzcę. Gra wyświetli stosowny komunikat o wyniku, a gracze będą mieli możliwość rozpoczęcia nowej rozgrywki.

- Na koniec gry zostaną przydzielone punkty w rankingu (tylko w przypadku gdy gracz wygra), gdzie można potem zobaczyć swoje wyniki w odpowiedniej zakładce RANKING w menu.
- Za punkty w rankingu można kupić w sklepie statek lub barierę.

2. Symulacja gry miedzy komputerami:
    - Gra umożliwia również symulację rozgrywki między dwoma komputerami. Algorytm zaimplementowany w programie rywalizuje z drugim takim samym algorytmem, a użytkownik może obserwować przebieg symulowanej bitwy.

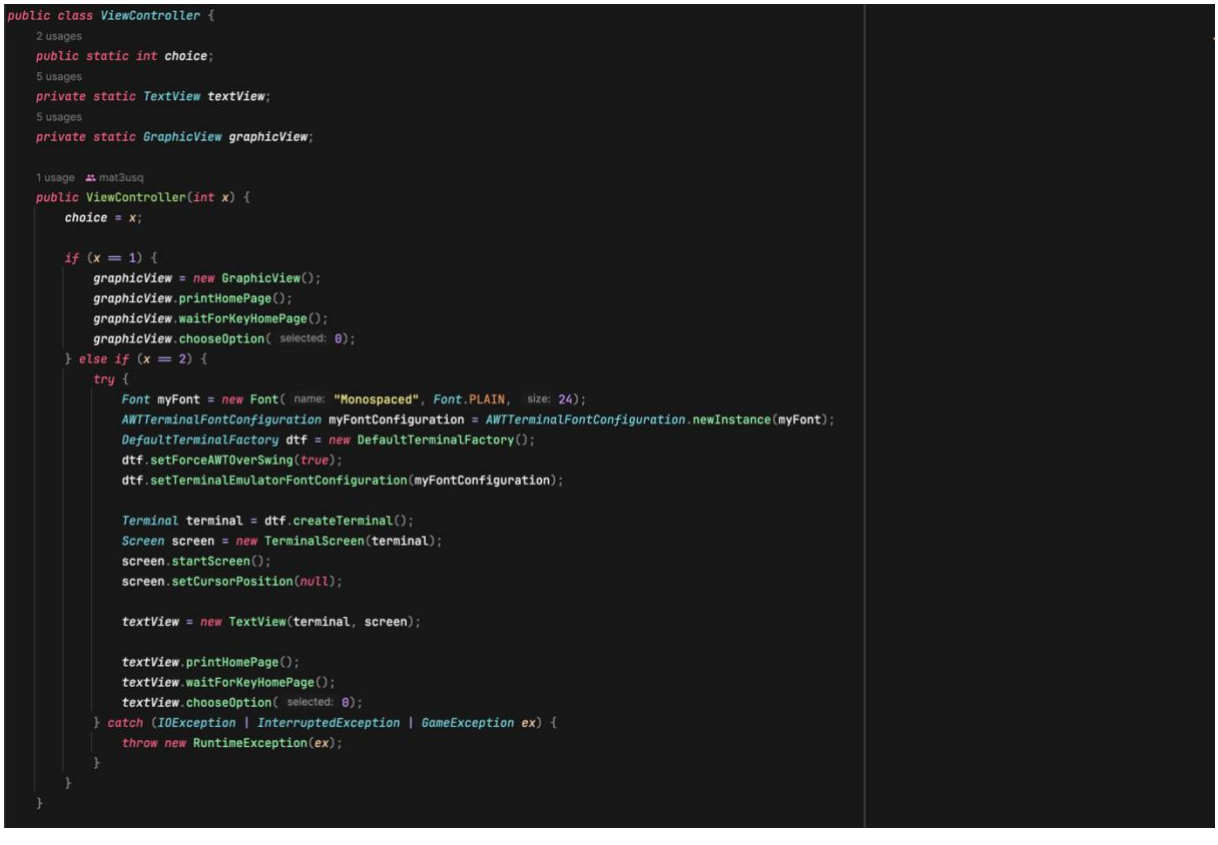
3. Zasady Gry
    - Dostęp do zasad gry w trakcie rozgrywki, umożliwiające graczom łatwe zrozumienie reguł oraz specyfiki rozgrywki w dowolnym momencie.

4. Zakupy w Sklepie
    - Gracze będą mieli możliwość zdobycia specjalnych statków lub barier poprzez zakupy w wirtualnym sklepie. To dodatkowe elementy taktyczne, które mogą wpłynąć na przebieg rozgrywki.

5. Ranking
    - System rankingowy pozwala graczom śledzić swoje postępy i porównywać swoje umiejętności z innymi graczami. Wyświetlenie rankingu dostarcza dodatkowej motywacji do poprawy swoich wyników.

OPIS REALIZACJI ZMIANY WARSTWY PREZENTACJI

1. Opis realizacji zmiany warstwy prezentacji został opracowany z myślą o elastyczności i oddzieleniu od logiki aplikacyjnej. Widoczne jest to w ViewController.java, który pełni rolę kontrolera do rozdzielania widoku, który został wybrany przez użytkownika.



```
public class ViewController {
    2 usages
    public static int choice;
    5 usages
    private static TextView textView;
    5 usages
    private static GraphicView graphicView;

    1 usage  mat3usq
    public ViewController(int x) {
        choice = x;

        if (x == 1) {
            graphicView = new GraphicView();
            graphicView.printHomePage();
            graphicView.waitForKeyHomePage();
            graphicView.chooseOption( selected: 0);
        } else if (x == 2) {
            try {
                Font myFont = new Font( name: "Monospaced", Font.PLAIN,  size: 24);
                AWTTerminalFontConfiguration myFontConfiguration = AWTTerminalFontConfiguration.newInstance(myFont);
                DefaultTerminalFactory dtf = new DefaultTerminalFactory();
                dtf.setForceAWTOverSwing(true);
                dtf.setTerminalEmulatorFontConfiguration(myFontConfiguration);

                Terminal terminal = dtf.createTerminal();
                Screen screen = new TerminalScreen(terminal);
                screen.startScreen();
                screen.setCursorPosition(null);

                textView = new TextView(terminal, screen);

                textView.printHomePage();
                textView.waitForKeyHomePage();
                textView.chooseOption( selected: 0);
            } catch (IOException | InterruptedException | GameException ex) {
                throw new RuntimeException(ex);
            }
        }
    }
```

2. Użytkownik wybiera rodzaj widoku w konsoli w głównej klasie programu. Następnie ViewController.java tworzy instancje odpowiedniego widoku i zwraca go do klasy, która wywołała metodę *getInstance()* np. do kontrolera Game.java, która zarządza cala logiką bitew. Ten kontroler nie ma wglądu do tego jaki widok jest aktualnie rozpatrywany, ponieważ jest on przystosowany do zarzadzania widokami abstrakcyjnej klasy *View.java*.

```java
public static View getInstance() {
    if (choice == 1)
        return graphicView;
    return textView;
}
```

```java
public static void main(String[] args) {
    System.out.println("Wybierz rodzaj wyswietlania gry:");
    System.out.println("1. Interfejs Graficzny");
    System.out.println("2. Interfejs Tekstowy");
    new ViewController(new Scanner(System.in).nextInt());
}
```

3. Abstrakcyjna klasa *View.java* stanowi kręgosłup warstwy prezentacji, definiując strukturę i oczekiwaną funkcjonalność różnych widoków bez angażowania się w konkretną implementację interfejsu użytkownika. Taka abstrakcja pozwala na czyste oddzielenie warstw, gdzie szczegóły prezentacji są odłączone od logiki aplikacji, umożliwiając bezproblemowe zmiany interfejsu użytkownika bez wpływu na podstawową funkcjonalność. (Fragment abstrakcyjnej klasy poniżej)

```java
public abstract class View {
    2 usages  2 implementations  mat3usq
    public abstract void printHomePage();

    2 usages  2 implementations  mat3usq
    public abstract void waitForKeyHomePage() throws IOException;

    2 usages  2 implementations  mat3usq
    public abstract void printLoginPage();

    23 usages  2 implementations  mat3usq
    public abstract void printMenuPage(int selected);

    6 usages  2 implementations  mat3usq
    public abstract void printExit();

    6 usages  2 implementations  mat3usq
    public abstract void chooseOption(int selected) throws IOException, GameException, InterruptedException;

    9 usages  2 implementations  mat3usq
    public abstract void option(int selected) throws IOException, GameException, InterruptedException;

    2 usages  2 implementations  mat3usq
    public abstract void printRules() throws IOException, GameException, InterruptedException;
```

4. Konkretne implementacje klasy *View*, a mianowicie GraphicView.java i TextView.java, demonstrują przystosowalność projektu.
    - GraphicView.java dostarcza graficzny interfejs użytkownika wykorzystując bibliotekę Swing, z obszernymi metodami do obsługi elementów graficznych i interakcji z użytkownikiem.
    - Z kolei TextView.java oferuje interfejs tekstowy, używając biblioteki Lanterna do zarządzania wyjściem i wejściem tekstowym w oknie terminala.

5. Zarówno GraphicView, jak i TextView dziedziczą po *View*, zapewniając spójność w zachowaniu interfejsu użytkownika przy jednoczesnym pozwalaniu na różnorodne sposoby prezentacji. Taki wzorzec projektowy ułatwia niezależny rozwój i modyfikację komponentów interfejsu użytkownika, co pokazuje znaczna liczba metod specyficznych dla każdej implementacji widoku (zapewniają one większą czytelność kodu), z 103 metodami w GraphicView i 70 w TextView. Dzięki czemu kontroler Game może swobodnie wywoływać metody z widoku nie zważając na to jaki to jest widok.

```
public boolean playTurn(Player attacker, Player defender, Boolean reverse) {
    Position shoot = null;
    boolean isHit;

    if (attacker.isAI() && defender.isAI())
        view.showOptionToSimulatedGame();
    else
        view.showOptionToPlay();

    if (attacker.areShipsStillSailing()) {
        if (defender.getDurabilityForceField() > 0) {
            defender.setDurabilityForceField(defender.getDurabilityForceField() - 1);
            view.printBarrier(defender);
        } else {
            if (attacker.isAI()) {
                boolean isAddHit;
                do {
                    try {
                        shoot = attacker.shoot(defender.getBattleField().getbattleFieldHideShips());
                        isAddHit = defender.addShoot(shoot);
                    } catch (GameException e) {
                        isAddHit = false;
                    }
                } while (!isAddHit);
```

6. Oddzielenie warstwy prezentacji od logiki aplikacji jest dodatkowo wzmacniane przez zastosowanie kontrolera, który abstrahuje obsługę wejścia użytkownika, pozwalając widokom skupić się wyłącznie na prezentacji. Takie podejście upraszcza proces zmiany lub dodawania nowych strategii prezentacji, jako że podstawowa logika pozostaje niezakłócona przez te modyfikacje.

7. Dzięki *Architekturze MVC* dodanie nowych widoków w przyszłości jest o wiele prostsze, jeśli zaistnieje taka potrzeba.

INTERESUJĄCE ZAGADNIENIA PROJEKTOWE

1. Wykorzystanie biblioteki Swing
    - Prosta integracja biblioteki Swing do obsługi interfejsu graficznego, pozwalająca na interaktywność w grze, a jednocześnie zapewniająca czytelność i estetykę.
    - Została ona wykorzysta do wyświetlania wszystkich widoków, ale dzięki jej funkcjom można sprawnie wyświetlać np. menu główne, które składa się zakładek w postaci paneli.

```
public class MainScreen extends JFrame {
    19 usages
    public JPanelShop shopPanel;
    39 usages
    public JPanelMenu menuPanel;
    21 usages
    public JDialogPopup popup;
    9 usages
    public JPanelRules rules;
    10 usages
    public JPanelRanking ranking;
    10 usages
    public JPanelResults results;

    mat3usq +1
    public MainScreen() {
        super( title: "Menu - Pirate Edition");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);
        this.requestFocusInWindow();
        this.setFocusable(true);
        this.setSize( width: 600,  height: 800);

        Dimension dimension = Toolkit.getDefaultToolkit().getScreenSize();
        int x = (int) ((dimension.getWidth() - this.getWidth()) / 2);
        int y = (int) ((dimension.getHeight() - this.getHeight()) / 2);
        this.setLocation(x, y);

        menuPanel = new JPanelMenu();
        menuPanel.setVisible(true);
        this.add(menuPanel);

        shopPanel = new JPanelShop();
        shopPanel.setVisible(false);
        this.add(shopPanel);

        popup = new JDialogPopup( parentFrame: this,  title: "Sklep",  message: "Czy napewno chcial(a)bys to kupic ?");
        popup.setVisible(false);

        rules = new JPanelRules();
        rules.setVisible(false);
        this.add(rules);

        ranking = new JPanelRanking();
        ranking.setVisible(false);
        this.add(ranking);

        results = new JPanelResults();
        results.setVisible(false);
        this.add(results);

        this.setVisible(true);
    }
```

2. Obsługa zdarzeń interaktywnych
    - Zaimplementowanie obsługi interakcji z graczem poprzez interfejs graficzny, umożliwiając jednocześnie płynną rozgrywkę i intuicyjne działanie.
    - Do tego zostało użyte nasłuchiwanie na klawisze poprzez interfejs KeyListeners oraz ActionListeners, gdzie po menu można się poruszać poprzez klikniecie strzałki.
    - Oto jedna z funkcji klasy GraphicView.java gdzie można przechwytywać klawisze oraz klikniecie w guziki z zakładami (w tym przypadku funkcja do poruszania się po Menu za pomocą strzałek oraz myszy):

```java
private void addMenuKeyListeners() {
    mat3usq +1
    mainScreen.menuPanel.addKeyListener(new KeyAdapter() {
        mat3usq +1
        @Override
        public void keyPressed(KeyEvent e) {
            super.keyPressed(e);
            switch (e.getKeyCode()) {
                case KeyEvent.VK_ENTER:
                    option(menuSelected);
                    break;
                case KeyEvent.VK_ESCAPE:
                    option( selected: 5);
                    break;
                case KeyEvent.VK_DOWN:
                    if (menuSelected + 1 < sizeOptions)
                        printMenuPage(++menuSelected);
                    break;
                case KeyEvent.VK_UP:
                    if (menuSelected - 1 >= 0)
                        printMenuPage(--menuSelected);
                    break;
            }
        }
    });
}
```

```java
private void addMenuActionsListeners() {
    mainScreen.menuPanel.playGame.addActionListener(ev -> {
        printMenuPage( selected: 0);
        option( selected: 0);
    });

    mainScreen.menuPanel.simulateGame.addActionListener(ev -> {
        printMenuPage( selected: 1);
        option( selected: 1);
    });

    mainScreen.menuPanel.shop.addActionListener(ev -> {
        printMenuPage( selected: 2);
        option( selected: 2);
    });

    mainScreen.menuPanel.rules.addActionListener(ev -> {
        printMenuPage( selected: 3);
        option( selected: 3);
    });

    mainScreen.menuPanel.ranking.addActionListener(ev -> {
        printMenuPage( selected: 4);
        option( selected: 4);
    });

    mainScreen.menuPanel.exit.addActionListener(ev -> {
        printMenuPage( selected: 5);
        option( selected: 5);
    });
}
```

3. Struktura danych planszy
    - Optymalne zorganizowanie struktur danych planszy gry, tak aby umożliwić efektywne rozmieszczanie statków, szybkie sprawdzanie rezultatów strzałów i obsługę dodatkowych elementów taktycznych

4. Logika strzałów komputera w symulacji oraz komputer vs gracz:
    - Komputer na początku tak jak gracz losowo strzela w plansze, ale gdy trafi w statek to zapisuje jego położenie.

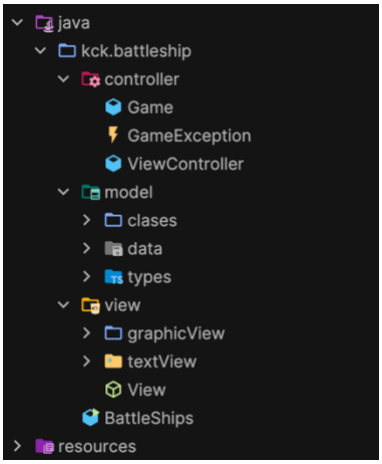
- Następnie strzela do pól otaczających poprzedni trafiony strzał, do momentu gdy zatopi cały statek, a potem powtarza cały algorytm od początku szukając innych statków.

```java
public Position ComputerShoot(BattleField defenderBattleField) throws GameException {
    if (shoots.isEmpty()) return Position.randPosition();
    else {
        nextShoots.addAll(defenderBattleField.getAdjacentValidPositions(getLastShoot()));

        if (nextShoots.isEmpty())
            return Position.randPosition();

        Position nextPos = nextShoots.get(0);
        nextShoots.remove( index: 0);
        return nextPos;
    }
}
```

5. Użycie podziału na MVC(model, view, controller)
    - Dzięki czemu został zachowany odpowiedni podział na klasy.
    - Struktura projektu stała się czytelniejsza oraz nie ma problemu w znalezieniu szukanych klas.



6. Dokładne przemyślenie i zaplanowanie logiki projektu umożliwia łatwe rozszerzanie aplikacji o nowe widoki. Aby dodać kolejny sposób prezentacji, wystarczy jedynie opracować nową klasę, która będzie zarządzać tym nowym widokiem.

INSTRUKCJE

INSTRUKCJA INSTALACJI

1. Pobierz pliki źródłowe gry.
2. Uruchom dany projekt w swoim edytorze kodu.
3. Wejdź w plik BattleShips.java.
4. Uruchom plik.
5. Wybierz w terminalu tryb wyświetlania.
6. Postępuj zgodnie z instrukcjami na ekranie.

INSTRUKCJA KONFIGURACJI

Dodatkowe kroki konfiguracyjne nie są wymagane. Gra jest gotowa do użycia po pobraniu z repozytorium.

INSTRUKCJA UŻYTKOWNIKA

1. Uruchom aplikacje.
2. Postępuj zgodnie z instrukcjami na ekranie.
3. Wprowadź swój NICK.
4. Poruszaj się po menu za pomocą strzałek lub myszy.
5. W razie problemów wejdź w zakładkę zasady gry w menu.
6. Jeśli chcesz zagrać kliknij w zakładkę Rozpocznij Grę i rozmieść swoje statki na planszy.
7. Następnie gra się rozpocznie.
8. Nie daj się wyeliminować Wrogowi!

WNIOSKI

Projekt ten nie tylko oferuje doskonałą zabawę i wciągającą rozgrywkę, ale także prezentuje się jako imponujące wyzwanie z zakresu programowania. Wykorzystanie biblioteki Swing do stworzenia interaktywnego interfejsu graficznego oraz do tekstowego - Lanterny umożliwia odświeżenie klasycznej gry w "Statki" poprzez przeniesienie jej do nowoczesnego środowiska, które wzbogaca doświadczenie użytkownika o dodatkowe elementy taktyczne. Ponadto, wprowadzenie trybu symulacji walki komputer kontra komputer otwiera nowe możliwości dla strategii, podnosząc poziom trudności i innowacyjności projektu.

SAMOOCENA

Projekt uważam za udany, gdyż nie tylko osiągnął zakładane cele, ale i przekroczył oczekiwania w zakresie doświadczenia programistycznego. Realizacja projektu odzwierciedla silne podstawy architektoniczne, z kodem, który nie tylko wykazuje elastyczność i łatwość w rozbudowie, ale jest również przystosowany do efektywnego zarządzania. Projekt wykazuje istotne możliwości rozwoju, szczególnie w aspekcie wprowadzania nowych funkcji oraz innowacyjnych widoków interfejsu użytkownika, które mogą wzbogacić interakcję z aplikacją. Obecne wersje widoków - tekstowy i graficzny - są dopracowane i oferują solidną bazę do dalszego rozwoju. Jestem bardzo zadowolony z tego projektu i liczę na pozytywne odzwierciedlenie w pozytywnej ocenie.