

Politechnika Świętokrzyska w Kielcach Wydział Elektrotechniki, Automatyki i Informatyki	
Programowanie obiektowe (JAVA) - Projekt Informatyka - II rok, Rok akademicki -2022/2023	
Temat projektu: Gra 2048	Wykonał: Futrzyński Hubert Grupa: 2ID13A

1. Ogólny opis projektu wraz informacjami o technologiach, framework'ach, bibliotekach użytych w projekcie:

Projekt wykonany w swingu i awt gdyż o wiele więcej praktycznych wbudowanych metod w nim było w odróżnieniu do LibGDX. W aktualnym silniku graficznym mogłem robić zaokrąglone kwadraty, przeźroczyste tło, gradient tła. Co nie było możliwe w LibGDX bez tworzenia własnych metod które by za to odpowiadały.

Projekt gry 2048 został napisany w języku Java i korzysta z kilku bibliotek i narzędzi, które ułatwiają implementację różnych funkcji i funkcjonalności. Projekt skorzystał z różnych bibliotek Java, takich jak:

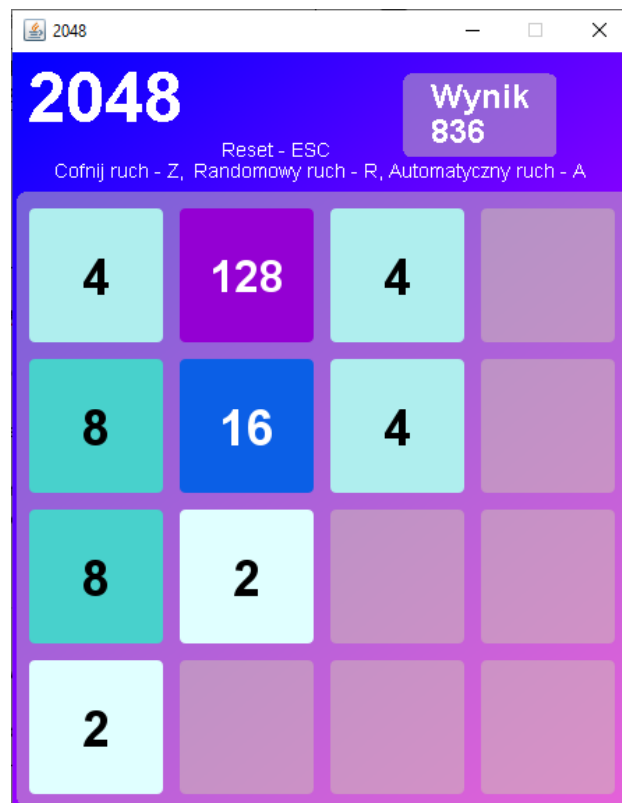
- **java.awt.event.KeyAdapter** i **java.awt.event.KeyEvent**: do obsługi zdarzeń związanych z klawiaturą.
- **javax.swing.***: do tworzenia i obsługi interfejsu użytkownika.
- **java.awt.event.***: do ogólnej obsługi zdarzeń w interfejsie użytkownika.
- **java.awt.***: do tworzenia i obsługi elementów interfejsu użytkownika oraz grafiki.
- **java.util.***: do obsługi struktur danych i różnych operacji narzędziowych.

Projekt wykorzystuje także bibliotekę JUnit 5 do testowania jednostkowego, co pozwala na sprawdzenie poprawności działania poszczególnych elementów kodu. JUnit jest najpopularniejszym frameworkiem do testowania jednostkowego w Javie.

Dodatkowo, używamy narzędzi dostarczanych przez JUnit Platform Launcher do odkrywania i uruchamiania testów jednostkowych w ramach naszego projektu. Te narzędzia to:

- **org.junit.platform.engine.discovery.DiscoverySelectors**
- **org.junit.platform.launcher.Launcher**
- **org.junit.platform.launcher.LauncherDiscoveryRequest**
- **org.junit.platform.launcher.core.LauncherDiscoveryRequestBuilder**
- **org.junit.platform.launcher.core.LauncherFactory**
- **org.junit.platform.launcher.listeners.SummaryGeneratingListener**
- **org.junit.platform.launcher.listeners.TestExecutionSummary**

Wszystkie te biblioteki i narzędzia są integralną częścią naszego projektu, umożliwiając tworzenie interaktywnego interfejsu użytkownika, obsługę zdarzeń wejściowych, zarządzanie danymi oraz przeprowadzanie i kontrolowanie testów jednostkowych.



Rysunek 1 Przykładowy Zrzut ekranu gry

2. Informacje na temat funkcjonalności projektu:

Gra obsługuje klawiaturę oraz myszkę jako kontrolery. Dzięki czemu mamy wybór jak chcemy grać.

Gra myśle obsługuje większość funkcjonalności standardowej gry 2048.

W punktach wypisze najważniejsze aspekty gry:

- Możliwość kontynuowania gry po osiągnięciu wartości klocka 2048, dzięki czemu możemy grać dopóki plansza się nie zapełni czyli nie będzie możliwy ruch w grze.
- Poruszamy się strzałkami, mamy także dodatkowe opcje takie jak:
Cofnięcie ruchu (Wielokrotnie, klawisz Z), Randomowy ruch (Klawisz R), Najbardziej dochodowy ruch (Taki automatyczny ruch, Klawisz A), Reset gry (Klawisz ESC)
- Poruszanie się myszką obsługuje to samo co strzałki. Jedno pociągnięcie to jeden ruch.

3. Informacje na temat sposobu uruchomienia oraz obsługi projektu:

Kompilacja gry z poziomu „cmd” albo „shell”:

Przechodzimy do folderu src/main/java i wpisujemy komendę:

- javac com/my2048/game/DesktopLauncher.java**

Odpalanie gry z poziomu „cmd” albo „shell”:

Przechodzimy do folderu src/main/java i wpisujemy komendę:

- java com/my2048/game/DesktopLauncher**

Obsługa to albo granie myszką, albo granie klawiaturą przyciski rozpisane na ekranie by użytkownik wiedział jakie ma opcje do wyboru.

4. Informacje na temat stworzonych klas, metod, funkcji (bez kodu źródłowego) z opisem ich podstawowej funkcjonalności (przyjmowanymi parametrami, wartościami zwracanymi) oraz ich przeznaczeniem:

4.1. Klasa „**Controller**” jest centralnym punktem zarządzania grą 2048. Implementuje interfejsy „**KeyAdapter**”, „**MouseListener**” i „**MouseMotionListener**”, które umożliwiają jej obsługę zdarzeń klawiatury i myszy.

Główne pola w tej klasie to:

- „**model**”: Obiekt klasy „**Model**”, który reprezentuje aktualny stan gry.
- „**view**”: Obiekt klasy „**View**”, który odpowiada za graficzne przedstawienie stanu gry.
- „**WINNING_TILE**”: Stała o wartości 2048, reprezentująca tytułową wartość kafelka, który oznacza wygraną w grze.
- „**initialClick**”: Obiekt klasy „**Point**”, który przechowuje początkową pozycję kliknięcia myszy.
- „**moveMade**”: Flaga informująca, czy ruch został wykonany.
- „**isMouseDown**”: Flaga informująca, czy mysz jest obecnie przeciągana.

Główne metody w tej klasie to:

- „**Controller(Model model)**”: To jest konstruktor klasy „**Controller**”. Przyjmuje jeden parametr:
 - „**model**”: obiekt typu „**Model**”, reprezentujący stan gry. Metoda ta nie zwraca żadnej wartości. Zadaniem konstruktora jest inicjalizacja modelu i widoku, a także dodanie nasłuchiwanie zdarzeń myszy do widoku.
- „**resetGame()**”: To jest metoda, która nie przyjmuje żadnych parametrów i nie zwraca żadnej wartości. Jej zadaniem jest resetowanie stanu gry na EDT (Event Dispatch Thread) - wątku, który jest odpowiedzialny za wykonywanie zdarzeń interfejsu użytkownika. Resetuje wynik gry, status gry (czy gracz przegrał czy wygrał), maksymalny kafelek i generuje nowe kafelki dla gry.
- „**keyPressed(KeyEvent e)**”: Metoda ta przyjmuje jeden parametr:
 - „**e**”: obiekt typu „**KeyEvent**”, reprezentujący zdarzenie naciśnięcia klawisza. Metoda nie zwraca żadnej wartości. Jest odpowiedzialna za obsługę zdarzeń naciśnięcia klawiszy, takie jak przesuwanie kafelków, resetowanie gry, cofanie ruchu, wykonywanie ruchu losowego czy automatycznego itp.
- „**mousePressed(MouseEvent e)**”, „**mouseDragged(MouseEvent e)**”, „**mouseReleased(MouseEvent e)**”: Te metody obsługują zdarzenia myszy. Każda z nich przyjmuje jeden parametr:
 - „**e**”: obiekt typu „**MouseEvent**” reprezentujący zdarzenie myszy. Te metody nie zwracają żadnej wartości. Są odpowiedzialne za obsługę zdarzeń związanych z przyciskami myszy oraz przeciąganiem myszy, które umożliwiają przesunięcie kafelków za pomocą myszy.
- „**checkGameStatus()**”: To jest metoda, która nie przyjmuje żadnych parametrów i nie zwraca żadnej wartości. Jej zadaniem jest sprawdzić, czy gracz wygrał lub przegrał grę, i wyświetlić odpowiednie okno dialogowe w zależności od stanu gry.

4.2.Klasa „**DesktopLauncher**” nie posiada żadnych pól. Jest to prosta klasa o jednej metodzie, służącej jako punkt wejścia dla aplikacji. Znajduje się w niej tylko jedna metoda:

„**main(String[] args)**”: Jest to metoda statyczna, która nie zwraca żadnej wartości (typu void) i przyjmuje jako argumenty tablicę Stringów (które są argumentami z linii poleceń). Jest to standardowa metoda uruchomieniowa dla aplikacji Java. Jej zadaniem jest inicjalizacja i uruchomienie gry 2048, co obejmuje stworzenie modelu i kontrolera gry, a także okna gry.

Podsumowując, głównym zadaniem klasy „**DesktopLauncher**” jest uruchomienie gry, zainicjowanie modelu i kontrolera oraz utworzenie interfejsu użytkownika.

4.3.Klasa „**Model**” jest główną klasą modelu w grze 2048, która zawiera logikę gry. Poniżej znajduje się lista pól i metod tej klasy:

Pola:

- „**FIELD_WIDTH**”: Statyczna, finalna zmienna, która określa szerokość pola gry (4x4).
- „**gameTiles**”: Dwuwymiarowa tablica przechowująca kafelki gry.
- „**score**”: Przechowuje aktualny wynik gry.
- „**maxTile**”: Przechowuje wartość najwyższego kafelka.
- „**previousStates**”: Stos przechowujący poprzednie stany gry. Używane do cofania ruchów.
- „**previousScores**”: Stos przechowujący poprzednie wyniki. Używane do cofania ruchów.
- „**isSaveNeeded**”: Flaga używana do decydowania, czy należy zapisać stan gry.

Metody:

- „**Model()**”: Konstruktor, który resetuje kafelki gry.
- „**addTile()**”: Dodaje nowy kafelek do gry.
- „**autoMove()**”: Wykonuje ruch automatycznie na podstawie efektywności ruchu.
- „**hasBoardChanged()**”: Sprawdza, czy stan planszy gry się zmienił. Zwraca wartość „**boolean: true**”, jeśli plansza gry uległa zmianie, „**false**” w przeciwnym razie.
- „**getMoveEfficiency(Move move)**”: Przyjmuje obiekt typu „**Move**” jako parametr i zwraca obiekt „**MoveEfficiency**”, który reprezentuje efektywność danego ruchu.
- „**randomMove()**”: Wykonuje losowy ruch.
- „**saveState(Tile[][] tiles)**”: Przyjmuje dwuwymiarową tablicę obiektów Tile jako parametr i zapisuje obecny stan kafelków gry.
- „**rollback()**”: Przywraca poprzedni stan gry.
- „**getEmptyTiles()**”: Zwraca listę pustych kafelków.
- „**resetGameTiles()**”: Resetuje kafelki gry.
- „**canMove()**”: Sprawdza, czy są możliwe ruchy. Zwraca wartość „**boolean: true**”, jeśli są możliwe ruchy, „**false**” w przeciwnym razie.
- „**compressTiles(Tile[] tiles), mergeTiles(Tile[] tiles)**”: Metody pomocnicze do przeprowadzania ruchów w grze.
- „**left(), right(), up(), down()**”: Metody odpowiadające za ruchy kafelków w odpowiednich kierunkach.
- „**rotateleft()**”: Obraca planszę o 90 stopni w lewo.
- „**getGameTiles()**”: Zwraca aktualne kafelki gry.

Klasa „**Model**” odpowiada za wszystkie operacje i manipulacje na kafelkach gry, takie jak przesuwanie, łączenie, dodawanie nowych kafelków, a także za sprawdzanie stanu gry.

4.4.Klasa „**MoveEfficiency**” służy do oceny efektywności ruchu w grze 2048. Poniżej opisuje jej podstawowe metody i funkcjonalności:

- „**MoveEfficiency(int numberOfEmptyTiles, int score, Move move)**”: Konstruktor klasy, który przyjmuje trzy argumenty: liczbę pustych kafelków („**numberOfEmptyTiles**”), wynik („**score**”) i ruch (move). Konstruktor nie zwraca żadnej wartości, ale służy do inicjacji nowego obiektu „**MoveEfficiency**”.
- „**getMove()**”: Metoda bez parametrów, która zwraca obiekt „**Move**” przypisany do obiektu „**MoveEfficiency**”.
- „**compareTo(MoveEfficiency o)**”: Metoda przyjmująca jako parametr inny obiekt „**MoveEfficiency**” i porównująca go z obecnym obiektem „**MoveEfficiency**”. Zwraca liczbę całkowitą: jeśli obecny obiekt jest mniejszy niż porównywany, zwraca wartość ujemną; jeśli obiekt jest większy, zwraca wartość dodatnią; jeśli obiekty są równe, zwraca zero.

Klasa ta jest używana do oceny i porównania efektywności różnych ruchów, co pozwala na wybór najbardziej optymalnego ruchu podczas automatycznego działania gry. Porównuje efektywność na podstawie liczby pustych kafelków i wyniku, które są kluczowymi czynnikami w strategii gry 2048.

4.5.Klasa „**Tile**” reprezentuje pojedynczy kafelek w grze 2048. Poniżej przedstawiam opis jej podstawowych funkcji i zastosowań:

- „**Tile()**”: Jest to konstruktor klasy, który inicjalizuje wartość kafelka na 0. Nie przyjmuje żadnych parametrów ani nie zwraca żadnej wartości.
- „**isEmpty()**”: Metoda bez parametrów, która zwraca wartość boolean wskazującą, czy kafelek jest pusty, czy nie (czy wartość kafelka wynosi 0).
- „**getFontColor()**”: Metoda bez parametrów, która zwraca obiekt „**Color**”, reprezentujący kolor czcionki na kafelku. Kolor jest różny w zależności od wartości kafelka.
- „**getTileColor()**”: Metoda bez parametrów, która zwraca obiekt „**Color**”, reprezentujący kolor kafelka. Kolor jest różny w zależności od wartości kafelka.

W grze 2048, każdy kafelek na planszy reprezentowany jest przez obiekt „**Tile**”. Metody w tej klasie służą do manipulacji i wyświetlania właściwości kafelków, takich jak wartość, kolor i status (czy jest pusty, czy nie).

4.6.Klasa „**View**” dziedziczy po „**JPanel**” i odpowiada za prezentowanie graficznego interfejsu użytkownika gry 2048. Poniżej przedstawiam opis jej podstawowych funkcji i zastosowań:

- **„View(Controller controller)”**: Jest to konstruktor klasy, który przyjmuje obiekt klasy „**Controller**” jako parametr. Kontroler jest ustawiany jako atrybut klasy, dodawane są też do niego Listenery do zdarzeń klawiatury i myszy.
- **„paint(Graphics g)”**: Metoda przyjmująca obiekt Graphics jako parametr, odpowiedzialna za rysowanie elementów na panelu, w tym tła, napisów, a także kafelków gry.
- **„drawTile(Graphics g2, Tile tile, int x, int y)”**: Metoda rysująca pojedynczy kafelek. Przyjmuje obiekt „**Graphics**”, obiekt „**Tile**” oraz współrzędne x, y kafelka jako parametry. Nie zwraca żadnej wartości.
- **„offsetCoors(int arg)”**: Metoda przyjmuje liczbę całkowitą jako parametr i zwraca przesunięcie dla danego argumentu. Jest używana do wyliczania prawidłowych pozycji kafelków na planszy.

W klasie „**View**” są również trzy zmienne boolean „**isGameWon**”, „**isGameLost**” i „**hasPlayerWonBefore**”, które śledzą stan gry (czy gracz wygrał lub przegrał) oraz czy gracz wygrał grę wcześniej. Ta klasa jest kluczowa dla prezentowania stanu gry użytkownikowi, jak również dla obsługi interakcji użytkownika z grą.

5. Informacje na temat ilości pracy włożonej przez poszczególnych członków zespołu w tworzenie projektu

Projekt wykonałem samodzielnie jako iż byłem zespołem jednoosobowym.