Laboratorium Podstawy Grafiki Komputerowej			
Grupa	**Zespół**	**Temat Projektu**	**Rok Studiów**
2ID13A	Sebastian Kalemba Bartosz Kaczmarczyk	„Tetris w Bibliotece LibGDX”	**Drugi**

1. Wstęp Teoretyczny:

Niniejsze sprawozdanie winno potwierdzać postęp uczyniony przez nas w ramach pracy nad obowiązkowym projektem z przedmiotu „Programowanie Obiektowe 2”. Tematem projektu, którego zrealizowania podjęliśmy próbę był „Tetris” a językiem programistycznym „Java”.

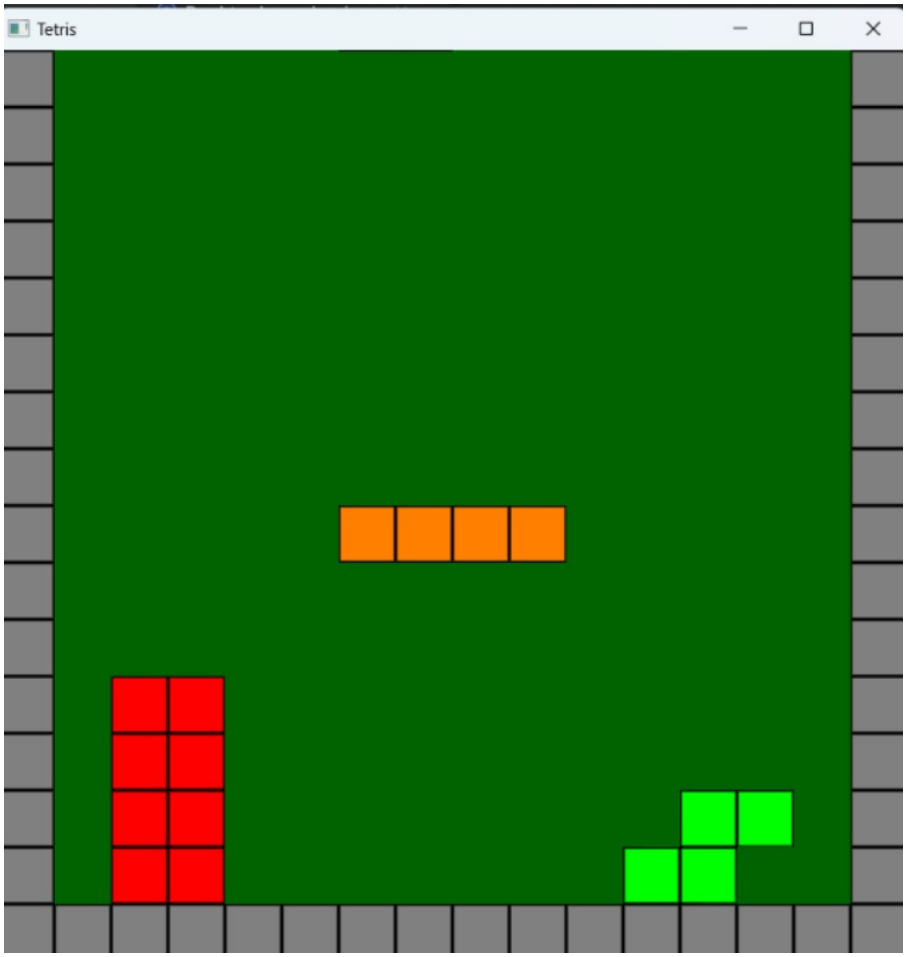
2. Opis:

2.1. Ogólny opis projektu

Celem naszego projektu było utworzenie interaktywnego, graficznego środowiska (gry) „Tetris”. Biblioteka z jakiej skorzystaliśmy to LibGDX. LibGDX to biblioteka programistyczna używana do tworzenia gier na różne platformy, takie jak Android, iOS, PC czy HTML5. Pomaga programistom w obszarach takich jak grafika, dźwięk, obsługa wejścia, fizyka i wiele innych, umożliwiając łatwe tworzenie wieloplatformowych gier przy użyciu języka Java.

2.2. Funkcjonalność projektu

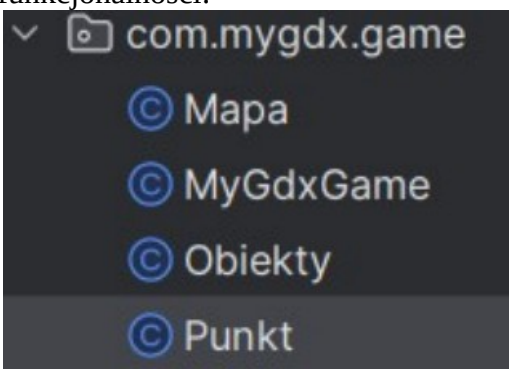
Jak sam tytuł wskazuje, rezultatem naszego projektu było utworzenie grywalnego prototypu popularnej gry wideo „Tetris”. Po uruchomieniu na ekranie wyświetla się prostokątne okienko, w jakim odgrywają się zdarzenia w obrębie funkcjonalności gry. Celem gracza jest układanie poziomych pasków z klocków. Jeżeli gracz (w jakiejkolwiek kombinacji) ułoży spójny poziomy pasek z klocków (od lewej do prawej), jest on usuwany a graczowi są dodawane punkty.



Okno projektu

2.3. Obsługa projektu
Aby uruchomić projekt, należy plik build.gradle uruchomić za pomocą programu rozumiejącego język Java (preferowanym wyborem jest InteliJ). Następnie wybierając plik DesktopLauncher w dekstop/src/com.mygdx.game należy skompilować cały projekt. Uruchomi się wtedy wyżej wspomniane okno. Aby sterować, konieczne jest manewrowanie klawiszami w, s, d, które wpływają na to jak zachowują się poszczególne losowo generowane klocki. Jeżeli gracz doprowadzi do sytuacji w wyniku której nie jest możliwe wygenerowanie żadnego nowego klocka (brak miejsca), gra się kończy i zamyka.

2.4. Szczegółowy opis projektu
Oprócz klasy DesktopLauncher, projekt zawiera łącznie cztery inne klasy, które obsługują różne funkcjonalności:



Oto krótki opis co się dzieje w każdej klasie podczas kompilacji:
DesktopLauncher
>Importowane są potrzebne klasy z biblioteki LibGDX.
>Ustawiane są parametry konfiguracyjne dla aplikacji (np. szerokość i wysokość okna, liczba klatek na sekundę, tytuł okna).
>Tworzony jest obiekt konfiguracji (Lwjgl3ApplicationConfiguration) i ustawiane są na nim odpowiednie parametry.
>Inicjalizowane są pewne wartości związane z obiektem klasy Mapa.
>Tworzony jest obiekt Lwjgl3Application, który odpowiada za uruchomienie gry (MyGdxGame) z zadaną konfiguracją.
>Jest to główna klasa, a zatem wywoływane, tworzone i inicjalizowane są w tej klasie wszystkie inne klasy oraz obiekty.

Mapa
>Konstruktor Mapa(): Inicjalizuje różne parametry, w tym rozmiary planszy, tablicę Plansza_do_gry, oraz ustawia wartości domyślne. Następnie wykonuje test wyświetlający stan planszy.
>Metoda test(): Wyświetla aktualny stan planszy w konsoli.
>Metoda Kolizja(Obiekty obiekty, boolean poziomo, int odcinek): Sprawdza kolizję pomiędzy blokami na planszy a obiektami ruchomymi. Określa, czy jest możliwy ruch w danym kierunku.
>Metoda zapiszDoPliku(String nazwaPliku, int[][] tablica): Zapisuje zawartość planszy do pliku tekstowego.
>Metoda Rysuj_od_nowa(): Tworzy nową mapę w formie obrazu, zapisuje ją do pliku PNG, a następnie aktualizuje obiekt gry (MyGdxGame).
>Metoda Sprawdz_Czy_Jest_blok(int repeat): Sprawdza, czy na planszy istnieją pełne rzędy bloków, usuwa je i przesuwa pozostałe bloki w dół. Następnie aktualizuje planszę i obiekt gry.

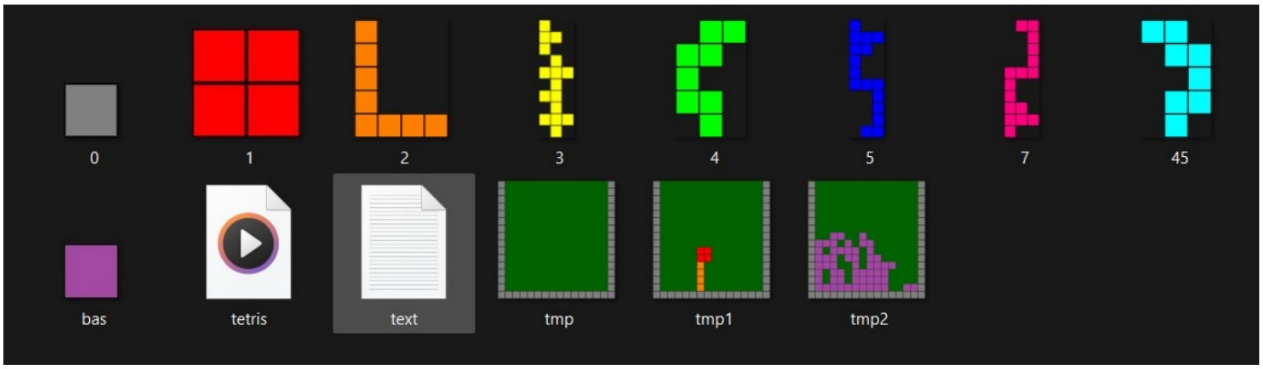
MyGdxGame
>Metoda create(): Inicjalizuje muzykę, obiekt klasy Obiekty, teksturę tymczasową (tmp), oraz region tekstury (textureRegion). Tworzy również obiekt klasy Mapa.
>Metoda render(): Odpowiada za renderowanie gry. Sprawdza, czy wystąpił update planszy lub upadek bloku, a następnie rysuje planszę i aktualny klocek na ekranie. Wywołuje funkcję zapisującą stan planszy, jeśli flaga Zapis jest ustawiona na true.
>Metoda dispose(): Zwalnia zasoby, w tym obiekt klasy SpriteBatch.
>Metody update() i upadek(): Obsługują logikę ruchu klocka. update() reaguje na klawisze kierunkowe oraz spacje, a upadek() sprawdza, czy klocek powinien opaść o jedną pozycję w dół.
>Metoda zapis(): Tworzy zrzut ekranu i zapisuje go do pliku PNG o nazwie "tmp1.png". Aktualizuje teksturę tymczasową.
>Metoda jednorazowo(): Rysuje początkową planszę.

Obiekty
Pola klasy:
x i y: Określają pozycję obiektu na planszy.
tekstura: Przechowuje teksturę obiektu.
blok: Lista punktów (klasa Punkt) reprezentujących bloki składające się na obiekt.
szerokosc i wysokosc: Określają wymiary obiektu.
X_Region i Y_Region: Określają fragment tekstury, który ma zostać wyświetlony.
ktory_klocek i ktory_obrot:
Przechowują informacje o aktualnie wybranym klocku i jego obrocie.
max_y: Przechowuje maksymalną wartość na osi Y dla obiektu.
>Metoda Losowanie(): Inicjalizuje losowy obiekt (klocek) poprzez wybór liczby od 1 do 5. Tworzy teksturę obiektu na podstawie wylosowanej liczby. Dodaje punkty (bloki) do listy blok w zależności od wylosowanego klocka.
>Metoda Tranformacja(): Modyfikuje obiekt w zależności od jego typu:
>Metoda Poruszanie(int odcinek, boolean Poziomo): Zmienia pozycję obiektu o określoną wartość odcinek, zarówno w poziomie, jak i w pionie, w zależności od wartości logicznej Poziomo.
>Metoda Zapisz_do_tablicy_polozenie(int tab[][]): Zapisuje aktualne położenie obiektu do tablicy tab, oznaczając bloki na planszy wartością 7.

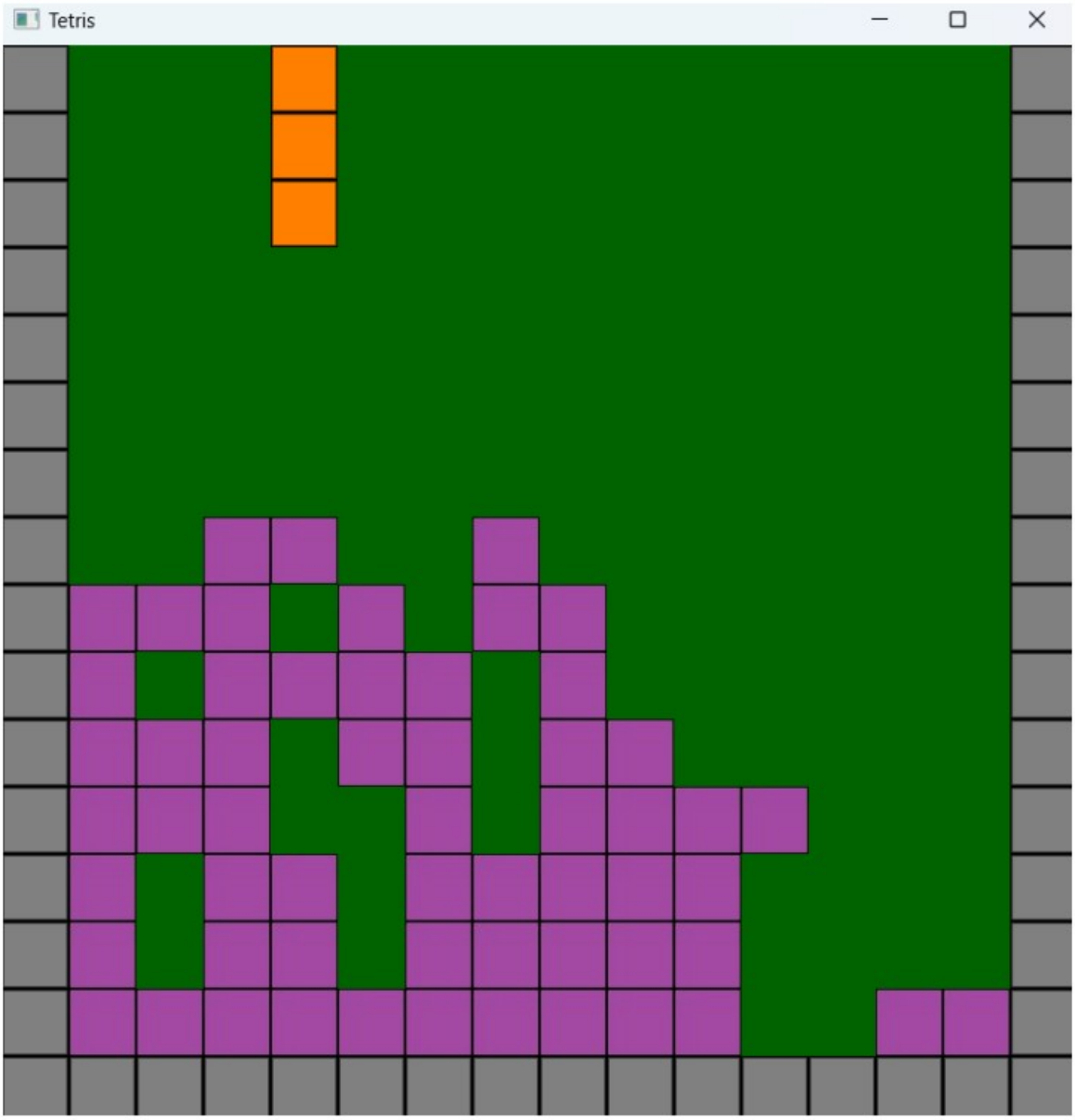
Punkt
Pola klasy: x i y:
Przechowują współrzędne punktu.
Konstruktor Punkt(int x, int y):
Inicjalizuje obiekt klasy Punkt przyjmując współrzędne punktu jako argumenty.

Na ekranie wyświetlane są bloki, z każdy z nich posiadający swoje unikalne współrzędne na bitmapie. Punkty te stanowią zarówno miejsce kolizji, jak i potencjalne miejsce, gdzie może dotrzeć nasz bloczek. Kiedy bloczek osiągnie dane współrzędne, zaznaczone na macierzy kolizji (plik text.txt), zniknie, a inne bloki zamieniają się w ten sam kształt. Następnie, w celu imitacji opadania, pozostałe bloki przemieszczają się o trzy jednostki w dół.

Wartym wspomnienia faktem jest to, że napotkaliśmy wiele problemów w trakcie pracy nad projektem. Przede wszystkim wystąpiły problemy z implementacją oraz obsługą oprawy graficznej, zarówno logiczne jak i programistyczne. Błędy z kolizją oraz błędy z wychodzeniem poza indeks tablicy również utrudniały nam prace nad projektem. Właśnie ta trójka – graficzne, indeksowe oraz kolizyjne były problemami jakie zabrały nam najwięcej czasu na debuggowaniu. Na potrzeby gry utworzyliśmy wiele plików graficznych oraz przygotowaliśmy plik dźwiękowy w formacie .wav, jaki stanowi ścieżkę dźwiękową gry, która jest stale odtwarzana w tle.



Pliki zasobów



Rezultat po utworzeniu poziomego paska

2.5. Testy jednostkowe
DESKTOPLAUNCHER.JAVA

```java
public class DesktopLauncher {
    public static void main (String[] arg) {
        Lwjgl3ApplicationConfiguration config = new Lwjgl3ApplicationConfiguration();
        Mapa.wysokosc = 800;
        Mapa.szerokosc = 800;
        config.setWindowedMode(Mapa.szerokosc,Mapa.wysokosc);
        config.setForegroundFPS(60);

        config.setTitle("Tetris");
        new Lwjgl3Application(new MyGdxGame(), config);
    }

}
```

Klasa DesktopLauncher. Jej głównym celem jest uruchomienie gry na platformie desktopowej. W metodzie main są konfigurowane ustawienia okna gry, takie jak rozmiar, tytuł i ilość klatek na sekundę. Następnie tworzona jest konfiguracja dla aplikacji libGDX, a na końcu inicjowana jest sama gra (MyGdxGame) na platformie desktopowej przy użyciu tych ustawień konfiguracyjnych.

PUNKT.JAVA

```java
package com.mygdx.game;

public class Punkt {
    6 usages
    public   int x;
    6 usages
    public int y;

    24 usages
    public Punkt(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Klasa Punkt reprezentuje punkt w dwuwymiarowej przestrzeni o współrzędnych x i y. Konstruktor klasy przyjmuje te współrzędne jako parametry i inicjuje nimi obiekt punktu. Oznaczenia pól jako public sugerują, że są one dostępne z zewnątrz klasy.

OBIEKTY.JAVA

```
2 usages
void Losowanie() {


    Random random = new Random();
    int a = random.nextInt( bound: 5) + 1;
   //int a = 5;
    ktory_klocek = a;

    String scieszka = Integer.toHexString(a) + ".bmp";

    this.tekstura = new Texture(scieszka);

    switch (a) {
```

Metoda Losowanie jest częścią klasy Obiekty.java i ma na celu losowe wybieranie i inicjowanie bloku lub kształtu do gry. W zależności od wylosowanej wartości a, metoda tworzy blok o określonych współrzędnych punktów, szerokości i wysokości.

Podstawowy przebieg metody obejmuje:

Losowanie liczby całkowitej a z zakresu od 1 do 5 (włącznie).
Na podstawie wylosowanej liczby a tworzenie odpowiedniego bloku (kształtu) o określonych współrzędnych punktów, szerokości i wysokości.
Dodawanie punktów tworzących blok do listy blok.
Ustawienie wartości zmiennych X_Region, Y_Region i ktory_obrot na stałe wartości.

```
void Tranformacja() {


    switch (ktory_klocek) {
        case 2:
            if (wysokosc > szerokosc) {
                Y_Region = wysokosc;
            } else {
                Y_Region = 0;
            }

            int tmp = szerokosc;
            szerokosc = wysokosc;
            wysokosc = tmp;
            break;
        case 3:
            ktory_obrot++;
```

Metoda Tranformacja również jest związana z operacjami na blokach lub kształtach w grze. Jest odpowiedzialna za transformacje/obrót aktualnie używanego bloku, w zależności od jego rodzaju (ktory_klocek). Oto ogólna funkcjonalność każdego przypadku:

Case 2:

Sprawdza, czy wysokość bloku jest większa niż szerokość.
Jeśli tak, ustawia Y_Region na wysokość bloku; w przeciwnym razie ustawia na 0.
Zamienia ze sobą wartości szerokości i wysokości.

Case 3:

Inkrementuje zmienną ktory_obrot i resetuje ją do 0, jeśli osiągnie 4.
Zamienia ze sobą wartości szerokości i wysokości.
W zależności od wartości ktory_obrot ustawia Y_Region na określoną wartość.

Case 4:

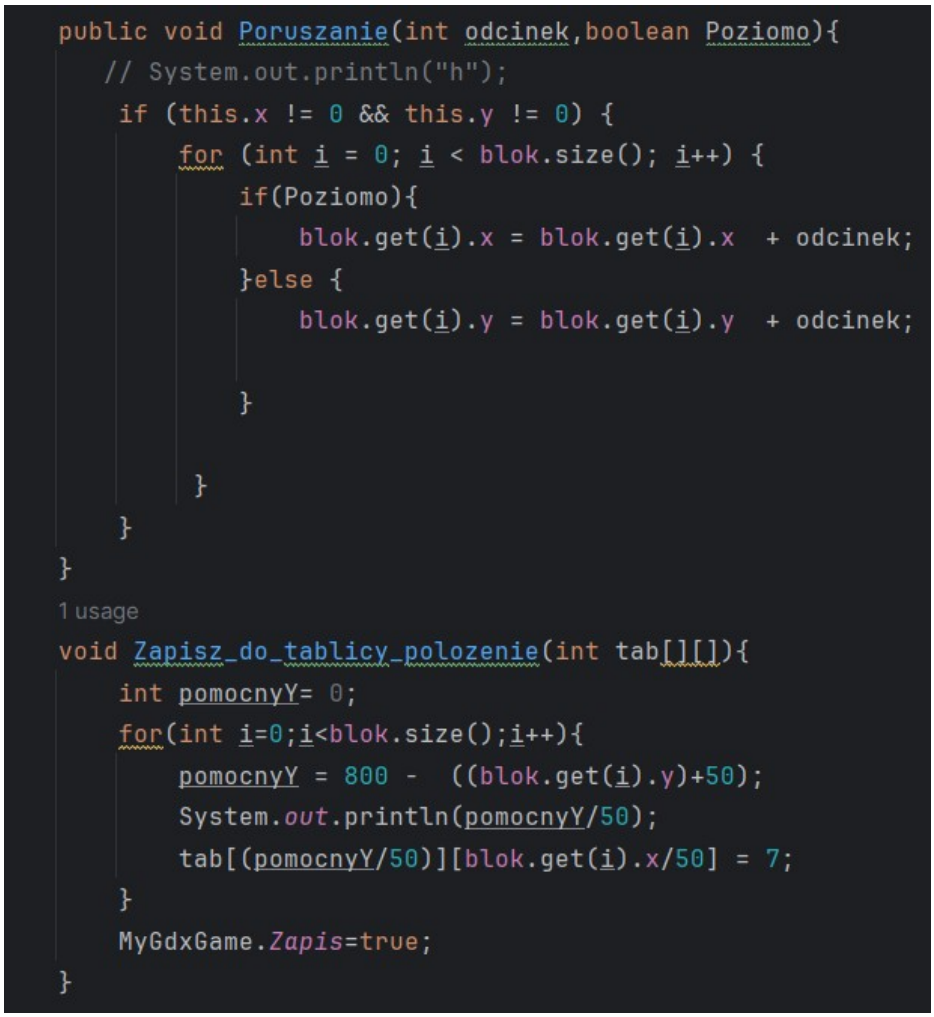
Sprawdza, czy wysokość bloku jest mniejsza niż szerokość.
Jeśli tak, ustawia Y_Region na wysokość bloku; w przeciwnym razie ustawia na 0.
Zamienia ze sobą wartości szerokości i wysokości.

Case 5:

Ustawia sztywne wartości szerokości, wysokości, X_Region i Y_Region.
Wcześniejsze fragmenty kodu, które są zakomentowane, sugerują różne warianty ustawień.

Case 6:

Pusta sekcja, brak operacji dla tego przypadku.



```
public void Poruszanie(int odcinek,boolean Poziomo){
   // System.out.println("h");
    if (this.x != 0 && this.y != 0) {
        for (int i = 0; i < blok.size(); i++) {
            if(Poziomo){
                blok.get(i).x = blok.get(i).x  + odcinek;
            }else {
                blok.get(i).y = blok.get(i).y  + odcinek;

            }

         }
    }
}
1 usage
void Zapisz_do_tablicy_polozenie(int tab[][]){
    int pomocnyY= 0;
    for(int i=0;i<blok.size();i++){
        pomocnyY = 800 -  ((blok.get(i).y)+50);
        System.out.println(pomocnyY/50);
        tab[(pomocnyY/50)][blok.get(i).x/50] = 7;
    }
    MyGdxGame.Zapis=true;
}
```

Metoda Poruszanie:

Przesuwa punkty bloku w poziomie lub pionie, zależnie od wartości Poziomo i parametru odcinek. Warunki if sprawdzają, czy wartości x i y nie są równe 0, aby uniknąć błędów przy próbie przesunięcia punktów.

Metoda Zapisz_do_tablicy_polozenie:

Przechodzi przez każdy punkt bloku i przelicza jego współrzędne, aby dostosować je do tablicy dwuwymiarowej tab.
Wartości x dzielone są przez 50, aby uzyskać odpowiedni indeks w tablicy, a wartość y jest przekształcana na współrzędną pionową.
W tablicy tab ustawiane są wartości 7, co może być pewnym oznaczeniem dla gry.
MyGdxGame.Zapis ustawiane jest na true, co może wskazywać na fakt zapisania pozycji bloku do tablicy.

MYGDXGAME.JAVA

```java
@Override
public void create() {
    music = Gdx.audio.newMusic(Gdx.files.internal( path: "tetris.wav")); // Podaj nazwę swojego pliku

    music.setLooping(true);
    music.setVolume(0.5f);
    music.play();
    batch = new SpriteBatch();

    obj = new Obiekty();


    tmp = new Texture( internalPath: "tmp.png");
    posiada_klocek = true;
    obj.Losowanie();

     textureRegion = new TextureRegion(obj.getTekstura(),obj.X_Region,obj.Y_Region, obj.szerokosc, obj.wysokosc);
    mapa = new Mapa();

}
```

Metoda create jest wykorzystywana w MyGdxGame.java

-Tworzy obiekt muzyki (music) na podstawie pliku dźwiękowego o nazwie "tetris.wav". Następnie ustawia opcje, takie jak pętla, głośność i rozpoczyna odtwarzanie.

-Inicjalizuje obiekt SpriteBatch (batch), który jest używany do efektywnego rysowania wielu obiektów w jednej klatce.

-Tworzy obiekt Obiekty (obj), który wydaje się związany z operacjami na blokach lub kształtach w grze.

-Ładuje teksturę z pliku "tmp.png" i ustawia flagę posiada_klocek na true.

-Wywołuje metodę Losowanie z obiektu obj, co może oznaczać losowanie nowego klocka w grze.

-Tworzy obiekt TextureRegion (textureRegion) na podstawie tekstury z obiektu obj oraz ustawia współrzędne regionu i jego rozmiar.

-Tworzy obiekt Mapa (mapa), który prawdopodobnie jest związany z logiką mapy w grze.

```
@Override
public void render() {



    Gdx.gl.glClearColor( red: 0,  green: 0.39f,  blue: 0,  alpha: 1); // Wartości RGB dla ciemnego zielonego koloru
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);







    if(update()){}
        else
    upadek();


    batch.begin();

    batch.draw(tmp, x: 0, y: 0);
    if (posiada_klocek == true) {
        batch.draw(textureRegion, obj.x, obj.y);



    }
    batch.end();
    if(Zapis){
        zapis();
        Zapis= false;
    }
}
```

Ta metoda jest częścią tej samej, wyżej wspomnianej klasy:

-Ustawia kolor tła (Gdx.gl.glClearColor) na ciemny zielony (wartości RGB).

-Czyści bufor kolorów (Gdx.gl.glClear) na podstawie ustawionego koloru tła.

-Wywołuje metodę update i jeśli zwróci true, to wykonuje odpowiednią akcję (upadek).

-Rozpoczyna rysowanie (batch.begin()).

-Rysuje obrazek z tekstury "tmp.png" na pozycji (0, 0).

-Jeśli posiada klocek (posiada_klocek == true), rysuje teksturę klocka na ekranie w pozycji i rozmiarze zdefiniowanym przez textureRegion i współrzędne obiektu obj.

-Kończy rysowanie (batch.end()).

-Sprawdza, czy flaga Zapis jest ustawiona na true, a jeśli tak, wywołuje metodę zapis() i ustawia flagę na false.

```java
@Override
public void dispose() {
    batch.dispose();

}
    1 usage
    private void upadek(){
        elapsedTime1 += Gdx.graphics.getDeltaTime();
        if (elapsedTime1 >= 1.0f && mapa.Kolizja(obj, poziomo: false, odcinek: -50)) {
            elapsedTime1 = 0;
            obj.y -= 50;
            odcinek = (-50);
            obj.Poruszanie(odcinek, Poziomo: false);
        }
    }
```

Metoda dispose jest wykorzystywana w libGDX do zwalniania zasobów i pamięci przed zamknięciem aplikacji. W tym przypadku, jedynym zasobem do zwolnienia jest obiekt batch (SpriteBatch), co jest wykonywane poprzez wywołanie dispose() na tym obiekcie.

Metoda upadek odpowiada natomiast za sprawdzanie i obsługę opadania klocka w dół co sekundę, jeśli nie zachodzi kolizja z mapą.

```java
private boolean update() {
    boolean poziomo = false;
    boolean rusza_sie = false;
    elapsedTime += Gdx.graphics.getDeltaTime();

    if(elapsedTime >= 0.15f) {
        elapsedTime = 0;

         {

            if (Gdx.input.isKeyPressed(Input.Keys.A)) {
                rusza_sie = true;
                poziomo = true;
```

Metoda update obsługuje logikę związana z ruchem i aktualizacją gry.
Inkrementuje elapsedTime o czas, który upłynął od ostatniego wywołania metody (mierzonego w sekundach).

Jeśli elapsedTime przekroczy 0.15 sekundy, wykonuje się następujące kroki:

a. Sprawdza naciśnięcie klawiszy (A, D, S):

Jeśli naciśnięto klawisz A, ustawia zmienne rusza_sie i poziomo na true, a odcinek na -50.
Jeśli naciśnięto klawisz D, ustawia zmienne rusza_sie i poziomo na true, a odcinek na 50.
Jeśli naciśnięto klawisz S, ustawia zmienne rusza_sie i poziomo na false, a odcinek na -50.

b. Jeśli zachodzi kolizja z mapą (mapa.Kolizja(obj, poziomo, odcinek)), a jednocześnie obiekt rusza się (rusza_sie jest true):

Wywołuje metodę Poruszanie obiektu obj z odpowiednim przesunięciem (odcinek).
Aktualizuje pozycję obiektu obj w zależności od kierunku ruchu.
Jeśli posiada_klocek wynosi false, co oznacza, że nie ma już aktywnego bloku:

Ustawia posiada_klocek na true.
Tworzy nowy obiekt Obiekty i losuje nowy blok.
Aktualizuje textureRegion na podstawie nowego bloku.
Zwraca true, jeśli w tej klatce doszło do aktualizacji (ruchu klocka w dół lub bocznego przesunięcia), w przeciwnym razie zwraca false.

```
static public void zapis(){
    try {
        byte[] pixels = ScreenUtils.getFrameBufferPixels( x: 0,  y: 0, Gdx.graphics.getWidth(), Gdx.graphic
        Pixmap pixmap = new Pixmap(Gdx.graphics.getWidth(), Gdx.graphics.getHeight(), Pixmap.Format.RGBA8
        BufferUtils.copy(pixels,  srcOffset: 0, pixmap.getPixels(), pixels.length);
        String TMP = currentPath.toString()+"\\tmp1.png";
        FileHandle fileHandle = Gdx.files.absolute(TMP);

        PixmapIO.writePNG(fileHandle, pixmap);
        pixmap.dispose();

        System.out.println("Zrzut ekranu został zapisany do pliku PNG: " + fileHandle.file().getAbsoluteP
    } catch (Exception e) {
        System.err.println("Błąd podczas zapisywania do pliku PNG: " + e.getMessage());
    }

    posiada_klocek = false;
    tmp = new Texture( internalPath: "tmp1.png");
    Zapis = false;
}
no usages
void jednorazowo(){
    Texture texture = new Texture( internalPath: "0.bmp");
    for(int i=0;i<16;i++){
        batch.draw(texture, x: 0, y: i*50);
    }
    for(int i=0;i<16;i++){
        batch.draw(texture, x: 750, y: i*50);
    }
    for(int i=0;i<16;i++){
```

Metoda zapis jest odpowiedzialna za zapisanie zrzutu ekranu do pliku PNG:
-Pobiera piksele z bieżącego bufora ramki, reprezentującego zrzut ekranu.
-Tworzy obiekt Pixmap na podstawie pikseli.
-Określa ścieżkę pliku docelowego i tworzy obiekt FileHandle na podstawie tej ścieżki.
-Zapisuje zawartość Pixmap do pliku PNG za pomocą PixmapIO.writePNG.
-Usuwa obiekt Pixmap po zakończeniu operacji.
-Ustawia flagę posiada_klocek na false.
-Ładuje nową teksturę "tmp1.png".
-Ustawia flagę Zapis na false.

MAPA.JAVA

Metoda jednorazowo rysuje prostokąty z tekstury "0.bmp" w jednorazowym trybie. Obejmuje to rysowanie krawędzi planszy, co może być związane z elementami otaczającymi planszę gry.

```
public Mapa() {
    this.x = (szerokosc / 50);
    this.y = (wysokosc / 50);
    Plansza_do_gry = new int[y][x];
    for (int i = 0; i < y; i++) {
        for (int j = 0; j < x; j++) {
            Plansza_do_gry[i][j] = 0;
        }
    }
    //  Plansza_do_gry[0][0]=1;


    //Kolizje
    for (int i = 0; i < y; i++) {
        Plansza_do_gry[i][0] = 9;
    }
    for (int i = 0; i < y; i++) {
        Plansza_do_gry[i][x - 1] = 9;
    }
    for (int i = 0; i < x; i++) {
        Plansza_do_gry[y - 1][i] = 9;
    }
    test();
    zapiszDoPliku( nazwaPliku: "text.txt", Plansza_do_gry);
}
```

Konstruktor Mapa() w klasie Mapa.java inicjalizuje planszę gry, ustawia krawędzie planszy i zapisuje stan planszy do pliku tekstowego.

```
void test() {
    for (int i = 0; i < y; i++) {
        for (int j = 0; j < x; j++)
            System.out.print(Plansza_do_gry[i][j]);
        System.out.println();
    }
}
```

Metoda test służy do wyświetlenia zawartości planszy do konsoli w celu przeprowadzenia testów.

```
boolean Kolizja(Obiekty obiekty, boolean poziomo, int odcinek) {
    int PomocniczeY = 0;
    for (int i = 0; i < obiekty.blok.size(); i++) {
        //System.out.println(obiekty.blok.get(i).x );
        PomocniczeY = 800 - (obiekty.blok.get(i).y);

        //System.out.println(i);
        //System.out.println(Plansza_do_gry[( / 50)][(obiekty.blok.get(i).y / 50)]);
        if (obiekty.blok.get(i).y < 801) {
            //System.out.println(obiekty.blok.get(i).x);
            if (poziomo) {
```

Metoda ta sprawdza, czy przemieszczenie obiektu (bloku) w danym kierunku jest możliwe, czy zachodzi kolizja z planszą, a także podejmuje odpowiednie akcje w przypadku kolizji (zapisuje położenie do tablicy i sprawdza możliwość utworzenia nowego bloku).

```
public static void zapiszDoPliku(String nazwaPliku, int[][] tablica) {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(nazwaPliku))) {
        for (int[] wiersz : tablica) {
            for (int element : wiersz) {
                writer.write(Integer.toString(element));
                writer.write( str: " ");
            }
            writer.newLine();
        }
        System.out.println("Tablica została zapisana do pliku: " + nazwaPliku);
    } catch (IOException e) {
        System.err.println("Błąd podczas zapisywania do pliku: " + e.getMessage());
    }
}
```

Metoda ta pozwala na zapisanie zawartości dwuwymiarowej tablicy do pliku tekstowego w formie siatki, gdzie wartości oddzielone są spacjami, a każdy wiersz znajduje się w nowej linii.

```
void Rysuj_od_nowa() {
    Pixmap pixmap = new Pixmap(Gdx.graphics.getWidth(), Gdx.graphics.getHeight(), Pixmap.Format.RGBA8888);

    int color = Color.rgba8888( r: 0,  g: 0.39f,  b: 0,  a: 1);
    pixmap.setColor(color);

    pixmap.fill();
    for (int i = y - 1; i > -1; i--) {
        for (int j = 0; j < x; j++) {
            if (Plansza_do_gry[i][j] == 9) {
                pixmap.drawPixmap(scianaPixmap,  x: j * 50,  y: i * 50);
            } else if (Plansza_do_gry[i][j] == 7) {
```

Metoda ta rysuje planszę gry na nowo, zapisuje ją do pliku PNG i wykonuje pewne operacje związane z aktualizacją stanu gry (ustawienie flag i zmiana tekstury).

```
void Sprawdz_Czy_Jest_blok(int repeat) {
    ArrayList<Integer> listaa = new ArrayList<>();
    System.out.println("Tak wykonuje sie");
    // Sprawdź, czy w którymkolwiek rzędzie jest pełny rząd bloków
    for (int i = y - 2; i > -1; i--) {
        int tmp = Plansza_do_gry[i][1];
        int a = 0;
        System.out.println(tmp);
        for (int j = 1; j < x - 1; j++) {
            if (tmp == 0) break;
```

Ta metoda natomiast jest odpowiedzialna za sprawdzanie, czy istnieją pełne rzędy bloków na planszy gry. Ogólnie rzecz biorąc, metoda ta pozwala na detekcję i usunięcie pełnych rzędów bloków na planszy, a następnie aktualizuje planszę i rysuje ją na nowo.

2.6. Podział prac

Sebastian Kalemba:
>Znajdywanie i weryfikowanie informacji na temat biblioteki LibGDX
>Projektowanie oprawy graficznej
>Implementacja oprawy graficznej
>Dokonanie sprawozdania
>Dokonanie dokumentacji
Bartosz Kaczmarczyk:
>Zaplanowanie, zaprojektowanie i zaimplementowanie kodu źródłowego gry
>Implementacja zasad gry
>Debugowanie
>Testowanie
>Archiwizacja projektu

3. Wnioski

Dokonawszy zadań, które otrzymaliśmy w ramach realizacji projektu silnika graficznego dwuwymiarowego, zrozumieliśmy od bardziej technicznej strony na jakiej zasadzie działają programy pisane w C++, które implementują rozwiązania graficzne. Dowiedzieliśmy się także jak implementować zasady powiązane z oświetleniem i cieniowaniem, rozwiązując powierzone nam zadania. Nabyta w ten sposób wiedza przyda się nam w trakcie przyszłych obcowań z grafiką 3D. Tym samym, uznajemy wszystkie wymogi silnika 3D za (w większości) zrealizowane i zakończone sukcesem.