UNIWERSYTET TECHNOLOGICZNO-PRZYRODNICZY
IM. J. I J. ŚNIADECKICH W BYDGOSZCZY

WYDZIAŁ TELEKOMUNIKACJI, INFORMATYKI I ELEKTROTECHNIKI
ZAKŁAD TECHNIKI CYFROWEJ

PROGRAMOWANIE OBIEKTOWE

LAB 3 – KÓŁKO I KRZYŻYK

AUTOR:
MATEUSZ BIRKHOLZ
DATA WYKONANIA
28.01.2021
DATA ODDANIA
28.01.2021

KIERUNEK:
INFORMATYKA STOSOWANA
GRUPA 1
SEMESTR 3
ROK AKADEMICKI 2020/2021
TRYB STUDIÓW: STACJONARNE

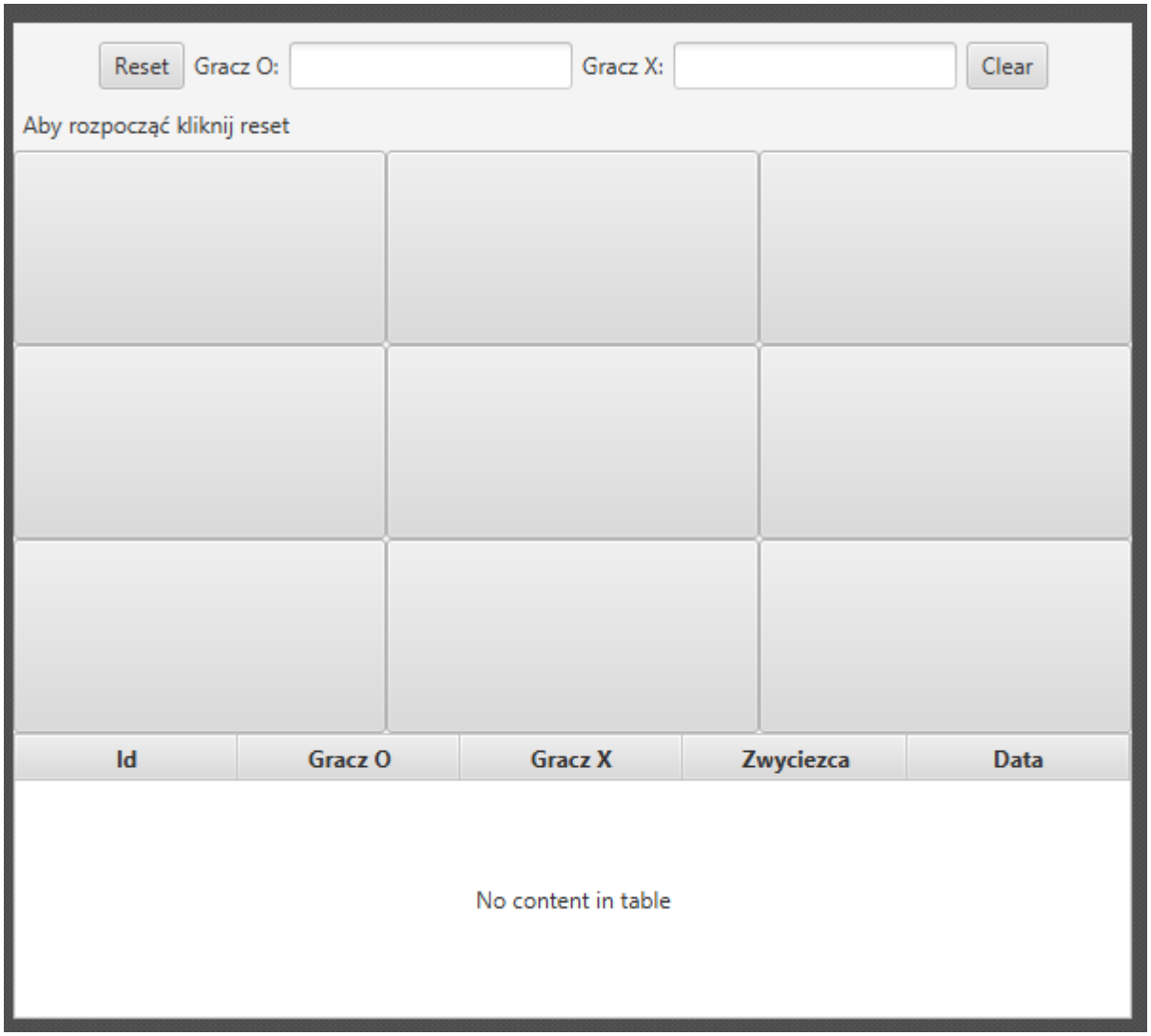
Cel:

Celem zadania była nauka wykorzystania bazy danych do przechowywania wyników programu a także wykorzystania obiektu loggera i executora.

Przebieg ćwiczenia:

Prace nad programem zacząłem od przygotowania bazy danych za pomocą hsqldb. Wykorzystałem do tego kod SQL z instrukcji.

Następnie przygotowałem interfejs graficzny aplikacji za pomocą SceneBuildera. Aplikacja posiada 9 przycisków do gry, przyciski Reset oraz Clear, 2 pola tekstowe na wpisywanie imion, label do wyświetlania informacji dla graczy oraz tabele która zawierać będzie dane z rekordów bazy danych.



W pliku Main tworzę nowy obiekt FXMLLoadera a następnie za jego pomocą odczytuje plik fxml oraz przypisuję mu klasę MainController podając jej konstruktor. Następnie tak jak w poprzednim zadaniu ładuję loader, ustawiam scene, odczytuje plik css i wyświetlam scene. Dodatkowo ustawiam scene tak że w momenci jej zamknięcie wywoła się funkcja shutdown() która zatrzymuje działanie executora.

```java
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        try {
            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(getClass().getResource("Main.fxml"));
            loader.setControllerFactory(c -> {
                return new MainController(new RozgrywkaDAOImpl(), new OXGameImpl());
            });

            Parent root = loader.load();
            Scene scene = new Scene(root,400,400);
            scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

            MainController controller = loader.getController();
            primaryStage.setOnCloseRequest(event -> {
                controller.shutdown();
                Platform.exit();
            });

            primaryStage.setScene(scene);
            primaryStage.show();
            primaryStage.setMinWidth(400);
            primaryStage.setMinHeight(600);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Tworzę klasę DataSource która jest odpowiedzialne za połączenie z bazą danych. Wykorzystuje bibliotekę Hikari do ustanawiania połączenia z bazą przy pomocy zdefiniowanych w klasie informacji.

```java
package lab.oxgame.datasource;

import java.sql.Connection;


public class DataSource
{
    private static HikariConfig config;
    private static HikariDataSource ds;


    static {
        config = new HikariConfig();
        config.setJdbcUrl("jdbc:hsqldb:file:db/rozgrywki");
        config.setUsername("admin");
        config.setPassword("admin");
        config.setMaximumPoolSize(1);
        ds = new HikariDataSource(config);
    }


    private DataSource() {}

    public static Connection getConnection() throws SQLException
    {
        return ds.getConnection();
    }
}
```

Tworzę nowy typ wyliczeniowy OXEnum. Może mieć on 3 wartości – *X, O, BRAK.* Konstruktor przypisuje do zmiennej podany w jego wywołaniu string. Funkcja toString zwraca wartość tego typu w postaci stringa. Z kolei funkcja fromString wykonuje odwrotną czynność zwracając odpowienią wartość typu wyliczeniowego.

```
package lab.oxgame.model;

public enum OXEnum {
    O("O"), X("X"), BRAK("");

    private String str;

    private OXEnum(String str) {
        this.str = str;
    }

    @Override
    public String toString() {
        return str;
    }

    public static OXEnum fromString(String value) {
        if(value == null || value.isEmpty())
        {
            return BRAK;
        }
        else if (O.str.equalsIgnoreCase(value))
        {
            return O;
        }
        else if (X.str.equalsIgnoreCase(value))
        {
            return X;
        }
        return null;
    }
}
```

Tworzę klasę Rozgrywka która jest obiektową reprezentacją rozgrywki. Dlatego też zawiera takie informacje jak id rozgrywki, nazwy graczy, zwycięzcę i czas rozgrywki. Klasa ta posiada 3 konstruktory różniące się przyjmowanymi wartościami. Każdy z nich otrzymane wartości zapisuje do zmiennych.

```
package lab.oxgame.model;

import java.time.LocalDateTime;

public class Rozgrywka {

    private Integer rozgrywkaId;
    private String graczX;
    private String graczO;
    private OXEnum zwyciezca;
    private LocalDateTime dataczasRozgrywki;

    public Rozgrywka() {

    }

    public Rozgrywka(Integer rozgrywkaId, String graczX, String graczO, OXEnum zwyciezca,
            LocalDateTime dataczasRozgrywki) {
        this.rozgrywkaId = rozgrywkaId;
        this.graczX = graczX;
        this.graczO = graczO;
        this.zwyciezca = zwyciezca;
        this.dataczasRozgrywki = dataczasRozgrywki;
    }

    public Rozgrywka(String graczX, String graczO, OXEnum zwyciezca,
            LocalDateTime dataczasRozgrywki) {
        this.graczX = graczX;
        this.graczO = graczO;
        this.zwyciezca = zwyciezca;
        this.dataczasRozgrywki = dataczasRozgrywki;
    }
```

Klasa ta posiada również zestaw getterów i seterów dzięki którym można pobierać poszczególne informacje o rozgrywce oraz je modyfikować

```java
public Integer getRozgrywkaId() {
    return rozgrywkaId;
}

public void setRozgrywkaId(Integer rozgrywkaId) {
    this.rozgrywkaId = rozgrywkaId;
}

public String getGraczX() {
    return graczX;
}

public void setGraczX(String graczX) {
    this.graczX = graczX;
}

public String getGraczO() {
    return graczO;
}

public void setGraczO(String graczO) {
    this.graczO = graczO;
}

public OXEnum getZwyciezca() {
    return zwyciezca;
}

public void setZwyciezca(OXEnum zwyciezca) {
    this.zwyciezca = zwyciezca;
}

public LocalDateTime getDataczasRozgrywki() {
    return dataczasRozgrywki;
}

public void setDataczasRozgrywki(LocalDateTime dataczasRozgrywki) {
    this.dataczasRozgrywki = dataczasRozgrywki;
}
```

Następnie tworze interfejs OXGame w którym deklaruję funkcje potrzebne do inicjalizacji gry, ustawienia wartości pola, pobranie wartości pola, pobranie kolejki i pobranie zwycięzcy.

```java
package lab.oxgame.engine;

import lab.oxgame.model.OXEnum;

public interface OXGame {

    //0,1,2, 3,4,5, 6,7,8
    void inicjalizuj(); //1. losowanie kolejnosci tj. X lub O

    void setPole(int indeks); // aktualizacja stanu gry
        //(zmiana kolejnosci, sprawdzanie czy jest zwycięzca)

    OXEnum getPole(int indeks); //w zasadzie niepotrzebna

    OXEnum getKolej(); //gdy OXEnum.Brak to koniec gry

    OXEnum getZwyciezca(); //na koniec sprawdzamy zwycięzcę

}
```

Tworzę klasę OXGameImpl która implementuje utworzony interfejs. Klasa ta deklaruje tablice zmiennych wyliczeniowych OXEnum która symuluję planszę do gry. Deklaruję również zmienne wyliczeniowe kolej i zwycięzca które przechowują kolejnego gracza oraz zwycięzcę gry, oraz zmienną krok która kontroluje ilość zapełnionych pól. Funkcja inicjalizuje ustawia krok na 0 i zwycięzcę na *null.* Czyści tabele ustawiając wszystkie pola na *BRAK* oraz losuje gracza który rozpocznie gre.

```
package lab.oxgame.engine;

import lab.oxgame.model.OXEnum;

public class OXGameImpl implements OXGame {

    //0 1 2
    //3 4 5
    //6 7 8
    private OXEnum[] plansza = new OXEnum[9];
    OXEnum kolej;
    OXEnum zwyciezca;
    int krok;

    @Override
    public void inicjalizuj() {
        krok = 0;
        zwyciezca = null;
        for(int i=0;i<9;i++)
        {
            plansza[i] = OXEnum.BRAK;
        }

        if(Math.random()<0.5)
        {
            kolej = OXEnum.X;
        }
        else
        {
            kolej = OXEnum.O;
        }
    }
```

Funkcja setPole ustawia wybrane pole na wartość zmiennej kolej, zwiększa krok i wywołuje funkcję aktualizuj.

```
@Override
public void setPole(int indeks) {
    plansza[indeks] = kolej;
    krok++;
    aktualizuj();
}
```

Funkcja aktualizuj sprawdza czy gra została wygrana sprawdzając wartości w liniach. Jeśli tak, ustawia wartość zmiennej zwycięzca na wartość zmiennej kolej. Jeśli nie, sprawdza czy liczba kroków jest większa niż 9. Jeśli tak to zwycięzca jest ustawiany na *BRAK* i ogłaszany jest remis. W innym wypadku po prostu zmieniana jest wartość zmiennej kolej na przeciną.

```
private void aktualizuj()
{
    //aktualizuj krok
    if(kolej.equals(plansza[0]) && kolej.equals(plansza[1]) && kolej.equals(plansza[2])
    || (kolej == plansza[3] && kolej == plansza[4] && kolej == plansza[5])
    || (kolej == plansza[6] && kolej == plansza[7] && kolej == plansza[8])
    || (kolej == plansza[0] && kolej == plansza[3] && kolej == plansza[6])
    || (kolej == plansza[1] && kolej == plansza[4] && kolej == plansza[7])
    || (kolej == plansza[2] && kolej == plansza[5] && kolej == plansza[8])
    || (kolej == plansza[0] && kolej == plansza[4] && kolej == plansza[8])
    || (kolej == plansza[6] && kolej == plansza[4] && kolej == plansza[2]))
    {
        zwyciezca = kolej;
    }
    else if(krok>=9)
    {
        zwyciezca = OXEnum.BRAK;
        System.out.println("Remis");

    }
    if(kolej==OXEnum.X)
    {
        kolej = OXEnum.O;
    }
    else if(kolej==OXEnum.O)
    {
        kolej = OXEnum.X;
    }
}
```

Dodatkowo klasa posiada gettery do pobrania wartości zmiennych kolej, zwycięzca i pole.

```
@Override
public OXEnum getPole(int indeks) {
    return plansza[indeks];
}

@Override
public OXEnum getKolej() {
    return kolej;
}

@Override
public OXEnum getZwyciezca() {
    return zwyciezca;
}
```

Tworzę nowy interfejs RozgrywkaDAO i deklaruję w nim 3 funkcje – na zapisanie rozgrywki, pobranie rozgrywek i usunięcie rozgrywek.

```
package lab.oxgame.dao;

import java.util.List;

public interface RozgrywkaDAO {

    int zapiszRozgrywke(Rozgrywka rozgrywka);
    List<Rozgrywka> pobierzRozgrywki(Integer odWiersza, Integer liczbaWierszy);
    int usunRozgrywki();

}
```

Następnie tworzę klasę która implementuje ten interfejs. Definiuję w niej obiekt loggera który pomoże mi sprawdzić działanie aplikacji. Następnie definiuje zawarte w interfejsie funkcje. Funkcja zapiszRozgrywke pobiera obiekt rozgrywki. W zmiennej String przechowuje wzór zapytania SQL dodającego rekord do bazy danych. Następnie w bloku try catch tworzy obiekt połączenia z bazą za pomocą funkcji z klasy DataSource. Przygotowuje również zapytanie bazując na przygotowanym wzorcu i wywołuje zapytanie. Zapytanie zwraca liczbę dodanych wierszy która jest zapisywana do zmiennej. Jeśli liczba dodanych wierszy jest większa od 0, czyli rekord został dodany, ustawia id następnej rozgrywki. Jeśli coś pójdzie nie tak, logger wyśle informacje o błędzie zapisu. Funkcja zwraca liczbę dodanych wierszy.

```
@Override
public int zapiszRozgrywke(Rozgrywka rozgrywka) {
    int liczbaDodanychWierszy = 0;

    String query = "INSERT INTO rozgrywka(gracz_o, gracz_x, zwyciezca, dataczas_rozgrywki) VALUES (?, ?, ?, ?)";
    try (Connection connect = DataSource.getConnection(); PreparedStatement preparedStmt = connect.prepareStatement(query, Statement.RETURN_GENERATED_KEYS))
    {
        preparedStmt.setString(1, rozgrywka.getGraczO());
        preparedStmt.setString(2, rozgrywka.getGraczX());
        preparedStmt.setString(3, rozgrywka.getZwyciezca().toString());
        preparedStmt.setObject(4, rozgrywka.getDataczasRozgrywki());
        liczbaDodanychWierszy = preparedStmt.executeUpdate();
        if (liczbaDodanychWierszy > 0)
        {
            ResultSet keys = preparedStmt.getGeneratedKeys();
            if(keys.next())
            {
                rozgrywka.setRozgrywkaId(keys.getInt(1));
            }
            keys.close();
        }
    }
    catch (SQLException e)
    {
        logger.error("Błąd podczas zapisywania rozgrywki!", e);
    }

    return liczbaDodanychWierszy;
}
```

Funkcja pobierzRozgrywki przyjmuje wiersz początkowy i liczbe wierszy. Deklaruje listę obiektów typu Rozgrywka. Tak jak w przypadku dodawania zapisuje szablon zapytania SQL w stringu, nawiązuje połącznie z bazą danych i przygotowuje zapytanie. Następnie sprawdza czy wprowadzone wartości są równe null i w zależności od tego odpowiednio modyfikuje zapytanie. Następnie wysyła zapytanie i zapisuje w zmiennej typu ResultSet. Uruchamiana jest pętli while która przechodzi przez wszystkie wyniki w tej zmiennej. Z każdym przejściem pobiera informacje z aktualnego rekordu i za pomocą tych informacji tworzy obiekt rozgrywki, który później dodaje do utworzonej wcześniej listy. Jeśli coś pójdzie nie tam logger wyśle informacje. Na końcu zwraca utworzoną listę rozgrywek.

```
@Override
public List<Rozgrywka> pobierzRozgrywki(Integer odWiersza, Integer liczbaWierszy) {
    List<Rozgrywka> rozgrywki = new ArrayList<>();

    String query = "SELECT * FROM rozgrywka ORDER BY dataczas_rozgrywki DESC" + (odWiersza != null ? " OFFSET ?" : "") + (liczbaWierszy != null ? " LIMIT ?" : "");
    try (Connection connect = DataSource.getConnection(); PreparedStatement preparedStmt = connect.prepareStatement(query))
    {

        if (odWiersza != null)
        {
            preparedStmt.setInt(1, odWiersza);
        }
        if (liczbaWierszy != null)
        {
            preparedStmt.setInt(odWiersza != null ? 2 : 1, liczbaWierszy);
        }

        ResultSet rs = preparedStmt.executeQuery();
        while(rs.next())
        {
            Integer rozgrywkaId = rs.getInt("rozgrywka_id");
            String graczO = rs.getString("gracz_o");
            String graczX = rs.getString("gracz_x");
            OXEnum zwyciezca = OXEnum.fromString(rs.getString("zwyciezca"));
            LocalDateTime dataczasRozgrywki = rs.getTimestamp("dataczas_rozgrywki").toLocalDateTime();
            Rozgrywka rozgrywka = new Rozgrywka(rozgrywkaId, graczX, graczO, zwyciezca, dataczasRozgrywki);
            rozgrywki.add(rozgrywka);
        }
        rs.close();

    }
    catch (SQLException e)
    {
        logger.error("Błąd podczas pobierania rozgrywki!", e);
    }

    return rozgrywki;
}
```

Funkcja usunRozgrywki przygotowuje zapytanie w stringu, nawiązuje połącznie z bazą, przygotowuje zapytanie i w bloku try catch wysyła je. Jeśli coś pójdzie nie tak to logger wyśle informacje. Funkcja zwraca liczbę usuniętych rekordów.

```java
@Override
public int usunRozgrywki() {
    int liczbaUsunietychWierszy = 0;

    String query = "DELETE FROM rozgrywka";
    try (Connection connect = DataSource.getConnection(); PreparedStatement preparedStmt = connect.prepareStatement(query))
    {
        liczbaUsunietychWierszy = preparedStmt.executeUpdate();
    }
    catch (SQLException e)
    {
        logger.error("Błąd podczas usuwania rozgrywki!", e);
    }

    return liczbaUsunietychWierszy;
}
```

W klasie MainController definiuję wszystkie utworzone w Scene Builderze obiektu oraz obiekty Listy obserwowalnej, ExecutorServices, RozgrywkaDAO i OXGame.

```java
package lab.oxgame;

import java.time.LocalDateTime;

public class MainController {

    private static final Logger logger = LoggerFactory.getLogger(MainController.class);
    @FXML
    Button btn0,btn1,btn2,btn3,btn4,btn5,btn6,btn7,btn8;

    @FXML
    private Label lblinfo;

    @FXML
    private Button btnReset, btnClear;

    @FXML TextField txtGraczO, txtGraczX;

    @FXML
    private TableView<Rozgrywka> rozgrywkaTable;

    @FXML
    private TableColumn<Rozgrywka, Integer> rozgrywkaIdColumn;

    @FXML
    private TableColumn<Rozgrywka, String> graczXColumn;

    @FXML
    private TableColumn<Rozgrywka, String> graczOColumn;

    @FXML
    private TableColumn<Rozgrywka, OXEnum> zwyciezcaColumn;

    @FXML
    private TableColumn<Rozgrywka, LocalDateTime> dataczasRozgrywkiColumn;

    private ObservableList<Rozgrywka> history;

    private ExecutorService wykonawca;

    RozgrywkaDAO rozgrywkaDAO;
    OXGame oxGame;
```

MainController posiada 3 konstruktory różniące się przyjmowanymi wartościami. Przypisują one otrzymane elementy do zdefiniowanych zmiennych lub tworzą nowe obiekty jeśli konstruktor ich nie otrzymał.

```
public MainController() {
    rozgrywkaDAO = new RozgrywkaDAOImpl();
    oxGame = new OXGameImpl();
}

public MainController(RozgrywkaDAO rozgrywkaDAO) {
    this.rozgrywkaDAO = rozgrywkaDAO;
}

public MainController(RozgrywkaDAO rozgrywkaDAO, OXGame gameImpl) {
    this.rozgrywkaDAO = rozgrywkaDAO;
    this.oxGame = gameImpl;
}
```

Funkcja initialize uruchamiania jest na początku pracy kontrolera. Łączy elementy widoku tabeli z odpowiednimi wartościami otrzymywanymi z bazy danych definiując ich nazwy (np dataczasRozgrywki) oraz przypisując im odpowiednie typy zmiennych. Następnie tworzy nowy obiekt kolekcji i przypisuje go do listy. Po tym ustawia tą liste jako przedmiot widoku tabeli. W ten sposób, poprzez wykorzystanie listy obserwowalnej możemy połączyć wyniki z bazy danych z widokiem tabeli. Następnie tworzony jest obiekt executora któremu przypisywane jest zadanie wczytania do listy 100 istniejących już w bazie rekordów.

```
@FXML
public void initialize()
{
    rozgrywkaIdColumn.setCellValueFactory(new PropertyValueFactory<Rozgrywka, Integer>("rozgrywkaId"));
    graczOColumn.setCellValueFactory(new PropertyValueFactory<Rozgrywka, String>("graczO"));
    graczXColumn.setCellValueFactory(new PropertyValueFactory<Rozgrywka, String>("graczX"));
    zwyciezcaColumn.setCellValueFactory(new PropertyValueFactory<Rozgrywka, OXEnum>("zwyciezca"));
    dataczasRozgrywkiColumn.setCellValueFactory(new PropertyValueFactory<Rozgrywka, LocalDateTime>("dataczasRozgrywki"));

    history = FXCollections.observableArrayList();
    rozgrywkaTable.setItems(history);

    wykonawca = Executors.newSingleThreadExecutor();
    wykonawca.execute(() -> {
        List<Rozgrywka> rozgrywki = rozgrywkaDAO.pobierzRozgrywki(0, 100);
        if (rozgrywki != null)
        {
            Platform.runLater(() -> {
                history.addAll(rozgrywki);
            });
        }
    });
}
```

Funkcja onActionBtnClear wywołuje się w momencie wciśnięcia przycisku Clear i wywołuje ona obiekt wykonawcy aby usunął z bazy wszystkie rekordy a następnie wyczyścił listę.

```
@FXML
public void onActionBtnClear(ActionEvent event)
{
    wykonawca = Executors.newSingleThreadExecutor();
    wykonawca.execute(() -> {
        int rozgrywki = rozgrywkaDAO.usunRozgrywki();
        if (rozgrywki > 0)
        {
            history.clear();
        }
    });
}
```

Funkcja onActionBtnReset wywołuje się w momencie wciśnięcia przycisku Reset. Funkcja sprawdza najpierw czy wprowadzone są imiona graczy. Jeśli nie to wyświetla informację w labelu. Następnie inicjalizuje gre metodą inicjalizuj klasy OXGame, wypisuje w labelu który gracz ma teraz turę oraz czyści zawartośći przycisków.

```
@FXML
public void onActionBtnReset(ActionEvent event){
    if(txtGraczO.getText().isEmpty() && txtGraczX.getText().isEmpty())
    {
        lblinfo.setText("Podaj imiona graczy!");
    }
    else
    {
        oxGame.inicjalizuj();
        lblinfo.setText("Kolej gracza "+oxGame.getKolej().toString());
        btn0.setText("");
        btn1.setText("");
        btn2.setText("");
        btn3.setText("");
        btn4.setText("");
        btn5.setText("");
        btn6.setText("");
        btn7.setText("");
        btn8.setText("");
    }
}
```

Funkcja onActionBtn wywołuje się po wciśnięciu któregoś z 9 przycisków pól. Najpierw sprawdza czy zwycięzca został już wybrany. Jeśli tak to nie zrobi nic. Jeśli nie to zapisze przycisk który wywołał funkcję. Jeśli zawartość tego pola jest różna pusta, wywoła funkcję ruch, podając przycisk który został wciśnięty. Następnie znowu sprawdzi czy został wybrany zwyciezca i jeśli tak to w labelu wyświetli się napis o zwycięzcy oraz powołany zostanie wykonawca który utworzy nowy obiekt rozgrywki i doda go do bazy danych jak i do listy obserwowalnej. Jeśli pojawi się jakiś problem powiadomi nas o tym logger.

```
public void onActionBtn(ActionEvent event)
{
    if(oxGame.getZwyciezca() == null)
    {
        Button btn = (Button)event.getSource();
        if(btn.getText().isEmpty())
        {
            ruch(btn);
        }
        if(oxGame.getZwyciezca() != null)
        {
            lblinfo.setText("Wygrał gracz "+oxGame.getZwyciezca().toString());
            wykonawca.execute(() -> {
                Rozgrywka rozgrywka = new Rozgrywka(txtGraczX.getText(), txtGraczO.getText(), oxGame.getZwyciezca(), LocalDateTime.now());
                rozgrywkaDAO.zapiszRozgrywke(rozgrywka);
                if(rozgrywka.getRozgrywkaId() != null)
                {
                    Platform.runLater(() -> {
                        history.add(0,rozgrywka);
                    });
                }
                else
                {
                    logger.error("Wynik rozgrywki nie został zapisany w bazie danych!");
                }
            });
        }
    }
}
```

Funkcja ruch ustawia tekst przycisku na symbol gracza który go wcisnął. Następnie za pomocą instrukcji switch i id przycisku wywołuje funkcję setPole z obiektu oxGame, podając odpowiedni numer pola. Na końcu aktualizuje label aby pokazywał kolej kolejnego gracza.

```
void ruch(Button btn)
{
    btn.setText(oxGame.getKolej().toString());
    switch(btn.getId()) {
        case "btn0":
            oxGame.setPole(0);
            break;
        case "btn1":
            oxGame.setPole(1);
            break;
        case "btn2":
            oxGame.setPole(2);
            break;
        case "btn3":
            oxGame.setPole(3);
            break;
        case "btn4":
            oxGame.setPole(4);
            break;
        case "btn5":
            oxGame.setPole(5);
            break;
        case "btn6":
            oxGame.setPole(6);
            break;
        case "btn7":
            oxGame.setPole(7);
            break;
        case "btn8":
            oxGame.setPole(8);
            break;
    }
    lblinfo.setText("Kolej gracza "+oxGame.getKolej().toString());
}
```

Jest jeszcze funkcja shutdown która kończy pracę wykonawcy. Wywoływana jest w momencie zamknięcia programu.

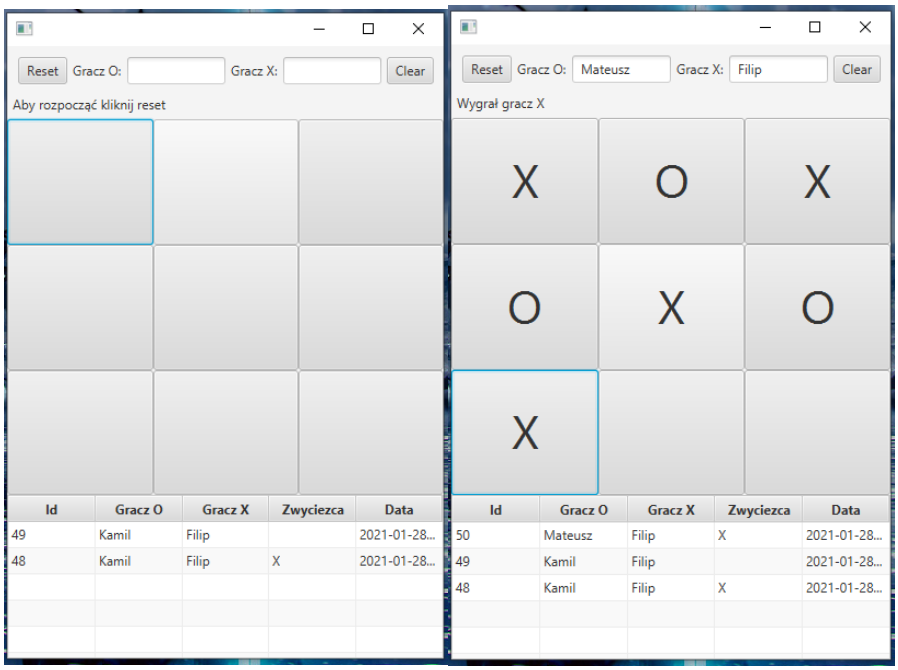
```
public void shutdown() {
    wykonawca.shutdown();
}
```

Dodatkowo do pliku css aplikacji dodałem klasę XOButton która zmienia wielkość czcionki na 350%

```
/* JavaFX CSS - Leave this
.XOButton {
    -fx-font-size: 350%;
}
```

Klasę tą przypisałem 9 przyciskom pól.

Gotowa gra prezentuje się następująco:



Podsumowanie i wnioski:

Cel laboratorium został zrealizowany. Na zajęciach nauczyłem się w jaki sposób łączyć program z bazą danych, jak wysyłać zapytania z poziomu programu, jak połączyć bazę danych z widokiem tabeli, jak wykorzystywać loggera i w jaki sposób wykorzystywać do tego wątki executora.

Wnioski:

- ExecutorService wymaga upewnienia się że został zatrzymany.
- Odpowiedzi bazy danych przychodzą w formie obiektu po którym można przechodzić za pomocą pętli.
- Za pomocą Arkuszów CSS można modyfikować wygląd elementów okna.