

# ***Dokumentacja projektowa***

*Przedmiot: Programowanie aplikacyjne [PAP]*

*Realizacja: semestr 21L*

*Prowadzący: mgr inż. Piotr Maciąg*

*Projekt zrealizowali: Jan Kaniuka, Przemysław Krasnodębski*

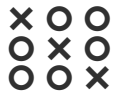
*Link do repozytorium: <https://gitlab-stud.elka.pw.edu.pl/pkrasnod/pap21l-z12>*



## #1 etap projektu

# Projekt TIC-TAC-TOE - opis i założenia








---





**Temat projektu:** *Gra w kółko i krzyżyk "TIC-TAC-TOE" (wersja rozszerzona)*

**Specyfikacja, początkowe wymagania, opis tekstowy aplikacji:**

**Typ aplikacji :** desktopowa 

- ekran startowy aplikacji będzie zawierał następujące przyciski do wyboru:
  -  START
  -  POMOC
  -  WYJDŹ(podczas przebywania w menu w tle będzie grała muzyka i będą wyświetlały się animacje - jak to zwykle bywa w ekranach startowych gier 🎮)
- po kliknięciu START gracz zostanie poproszony o podanie imienia/nicku
- do wyboru będą 4 tryby gry:
  1. pojedyncza partia / wiele partii
  2. gracz  komputer (przewidujemy 3 poziomy trudności gry ze strony komputera : Beginner, Medium, Expert -> realizacja poprzez stosowne algorytmy)
  3. gracz  gracz (lokalnie na jednym komputerze)
  4. gracz  komputer SPEEDRUN np. 1,5 sekundy na ruch z odliczaniem na ekranie 🕒
- po przegranej/wygranej pojawi się odpowiedni komunikat z melodią i pytaniem o dalszą chęć gry
- statystyki gry w formie np. **nick**, **liczba wygranych**, **liczba przegranych**, **liczba gier nierozstrzygniętych** zostaną dodane jako rekord do **bazy danych**  (np. sqllite)
- po zakończeniu gry możliwe będzie wyjście z aplikacji poprzedzone zapisaniem statystyk lub powrót do menu głównego w celu np. zmiany poziomu trudności w trybie gracz vs. komputer

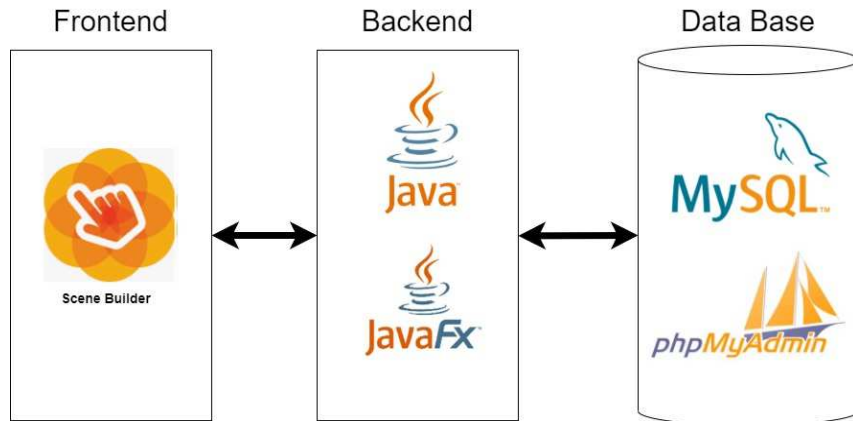
**Wstępna specyfikacja technologii/narzędzi** 

- technologia : JavaFX 
- narzędzia : np. ORMLite - framework zapewniający komunikację Java  Baza danych

## #2 etap projektu



### Prototyp aplikacji

#### Architektura aplikacji



**Frontend** - przy użyciu Scene Builder stworzyliśmy ekrany menu aplikacji z przyciskami funkcyjnymi. Projektowanie odbywa się metodą *drag&drop*, a rezultat zapisywany jest w pliku o formacie *.fxml*.

**Backend** - logika aplikacji obsługuje na ten moment dwa tryby gry:

- Gracz  Komputer (komputer losowo wybiera miejsce postawienia znaku)
- Gracz1  Gracz2 (lokalnie, na jednej maszynie)

**Data Base** - w relacyjnej bazie MySQL stworzyliśmy pierwszą tabelę *users* (przy użyciu oprogramowania *phpMyAdmin*).

Encja w tej tabeli wygląda w następujący sposób:

**<id, nick, liczba wygranych, liczba przegranych, liczb remisów>**

Dla klasy *BaseMenager()* obsługującej bazę danych zrealizowaliśmy następujące funkcjonalności:

**C** - create

**R** - read

**U** - update

### Wymagania środowiskowe

Przy tworzeniu gry korzystamy z:

- Visual Studio Code
- **JavaFX** w wersji 16
- **Java Development Kit** w wersji 15.02
- biblioteki do obsługi bazy danych: **mysql connector java 8.0.22.jar**

### Instrukcja zbudowania i uruchomienia aplikacji

1. Na początku należy dołączyć biblioteki z pakietu JavaFX oraz do obsługi MySQL.  
W Visual Studio Code: *Add Referenced Libraries*  
Poprawność dołączenia bibliotek można sprawdzić w pliku *settings.json*
2. Aplikację uruchamiamy poprzez uruchomienie pliku głównego - **App.java**

## #3 etap projektu

### Działająca aplikacja

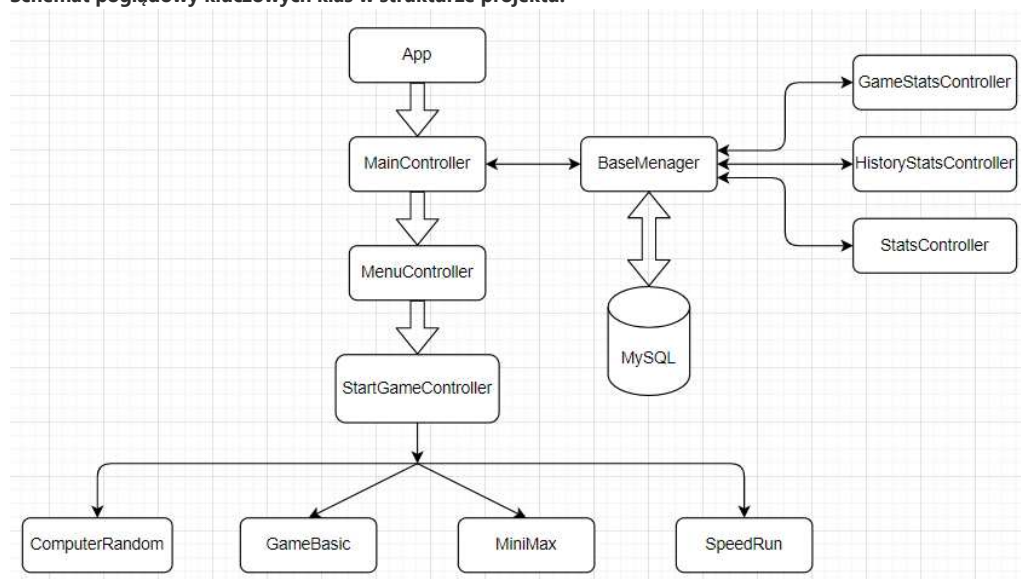
#### Spotkanie w połowie etapu:

Spotkanie odbyło się 29.04.2021 o godzinie 16:30 na platformie Teams.

#### Projekt aplikacji:

##### - Architektura

Schemat poglądowy kluczowych klas w strukturze projektu:



##### - Wykorzystywane narzędzia, biblioteki, zależności



- **Scene Builder** - tworzenie bardziej złożonych ekranów z wieloma polami wejściowymi lub wyboru i przyciskami - metoda Drag&Drop
- **phpMyAdmin** - tworzenie nowych tabel, zarządzanie bazą danych



- biblioteki **JavaFx** w wersji 16
- **mysql connector java 8.0.22.jar**
- **junit-platform-console-standalone.jar**

##### - Główne klasy:

Struktura projektu jest rozbudowana i składa się z wielu klas. Niektóre z nich mają zbliżoną strukturę oraz zastosowanie. Z tego powodu opiszemy poniżej po jednej klasie reprezentatywnej dla kolejno : 1) bazy danych, 2) logiki gry, 3) kontrolera realizującego żądania użytkownika:

- **BaseMenager.java** - w tej klasie zaimplementowano metody odpowiedzialne między innymi za: połączenie/rozłączenie z DB, dodanie nowego gracza/zespołu oraz dodanie statystyk gry.
- **GameBasic.java** - klasa odpowiada za logikę gry (konkretnie w trybie Multiplayer). Klasy takie jak *Speedrun*, *MiniMax* i inne mają podobną strukturę, lecz różnią się implementacją algorytmów determinujących ruchy komputera.

- **StartGameController.java** - obsługuje przyjmowanie *nicku* od gracza, powrót do poprzedniego ekranu ze statystykami/instrukcją oraz wybór rodzaju gry wraz z liczbą tychże gier (bez konieczności wielokrotnego uwierzytelniania).

## - Odniesienie do zastosowanego wzorca projektowego □

Zastosowaliśmy koncepcję **Model-Widok-Kontroler**.

**Model** - stosowany jest do pobierania i przygotowania rekordów z bazy danych

**Widok** - reprezentuje to, co widzi użytkownik

**Kontroler** - odpowiada m.in. za przetwarzanie danych pobranych za pomocą modelu i przekazanie ich użytkownikowi oraz zapisanie danych przez niego podanych (poprzez widok)

## Implementacja: □

### - struktura systemu (organizacja warstw)

- **Frontend** - użytkownik może oddziaływać na grę z poziomu GUI, ma możliwość przeglądania statystyk, historii gier, wyboru liczby/trybów/czasu gier oraz może także przejść do sekcji *HowTo* w razie wątpliwości.
- **Backend** - zapewnia działanie wielu trybów gier (multiplayer, z ograniczonym czasem i innych)
- **Data Base** - połączenie z bazą danych, pobieranie rekordów w celu wyświetlenia tabeli statystyk oraz tworzenie *statement'ów* z nowymi osiągnięciami graczy

### - zgodność z standardami programowanie w Javie



Każdy z członków zespołu zainstalował w Visual Studio Code plugin *Checkstyle* i przy jego pomocy poprawione zostały wszystkie błędy leksykograficzne oraz strukturalne. Za wzorec przyjęliśmy **Google Check** - *google\_checks.xml*

### - testy jednostkowe dla wybranych klas/metod

Przy wykonywaniu testów jednostkowych korzystaliśmy z **JUnit5**.

Wspomniane testy znajdują się w następujących plikach:

- **GameLogicTest.java** - testy funkcji realizujących poszczególne elementy logiki aplikacji (sprawdzenie istnienia wygranej/ końca gry bez rozstrzygnięcia, sprawdzenie poprawności zapisu sekwencji ruchów)
- **BaseMenegerTest.java** - testy sprawdzające poprawność komunikacji z bazą danych (połączenie/rozłączenie, zapis/pobieranie statystyk)

## Dokumentacja:

### - Wymagania środowiskowe - Java Development Kit w wersji 15.02:

1. Aplikacja powstawała w środowisku Visual Studio Code.
2. Biblioteki potrzebne do uruchomienia aplikacji z poziomu IDE zostały już opisane powyżej.
3. Instrukcja uruchomienia (zarówno pliku *.jar* jak i kodu w IDE) znajduje się w dalszej części tego sprawozdania.

### - Instrukcja budowania kodu i uruchomienia aplikacji

IDE (kod) 



1. Należy sprawdzić, czy wszystkie niezbędne biblioteki wymienione powyżej są dołączone w zakładce *Referenced Libraries*.
2. Uruchamiamy plik **App.java**.


### Uruchomienie przez gracza



W konsoli wpisujemy **java --module-path c:/javafx-sdk-16/lib/ --add-modules javafx.controls,javafx.fxml -jar Game.jar** i rozpoczynamy rozgrywkę.

□ W powyższej komendzie należy dokonać wskazania na lokalizację bibliotek JavaFX zainstalowanych lokalnie na dysku komputera oraz ewentualnie inną lokalizację pliku z aplikacją.


### - Raport Checkstyle

Kod pozytywnie przeszedł walidację *Checkstyle*  - dowodem tego są liczne commity opisane jako "Zmiany leksykograficzne i strukturalne - Checkstyle". Nie dysponujemy jednak raportem, ponieważ nie wykorzystywaliśmy narzędzia *Maven* w czasie prac nad projektem.

## #4 etap projektu

### Działająca aplikacja

### (uwzględniająca uwagi Prowadzącego z 3-go etapu)

→ Termin wykonania  do 14.06.2021

→ Maksymalna liczba punktów do zdobycia -> 10p



### Implementacja

- wprowadzenie poprawek wskazanych przez Prowadzącego:  
Na koniec trzeciego etapu Prowadzący zasugerował następujące modyfikacje:

Komentarz

Sądzę, że pliki w głównym katalogu powinny zostać uporządkowane w ramach katalogów/pakietów. Warto też dokładniej opisać zadania poszczególnych plików/klas.


### Wdrożenie zasugerowanych zmian

- Z pomocą narzędzia **Maven** dokonaliśmy uporządkowania plików w pakiety i stosowne katalogi   
**TicTacToe** to główny katalog z projektem zbudowanym przez Maven'a, wewnątrz niego zostały wydzielone między innymi katalogi:
  - /src - który wprowadza następującą strukturę plików: main/java - główne klasy aplikacji, main/resources - formatki .xml, pliki ze stylami .css, oraz czcionkami .ttf, test/java - klasy testujące.
  - / .idea - określa używane w projekcie biblioteki, które pobierane są z repozytorium *Maven'a*
- Rozszerzyliśmy dotychczasowy opis klas czyniąc go bardziej szczegółowym, sprecyzowaliśmy zadania/funkcjonalności poszczególnych klas . Aby wspomniane zadania przestawić w możliwie przejrzysty sposób dodaliśmy dodatkową zakładkę w Wiki - [Opis klas](#).

### Dokumentacja końcowa

Krytyczna analiza rozwiązania:   

- **znane ograniczenia** 

- Aplikacja niestety nie jest cross-platformowa.
- Może potencjalnie zajmować dużo miejsca na dysku (szczególnie w formie tzw. *fatjar*)
- W przypadku aktualizacji to użytkownicy muszą ręcznie zainstalować *update*.
- W przypadku aplikacji webowych  wymagania systemowe nie są istotne, ale w naszej aplikacji desktopowej już tak.

- **rozwiązane problemy** 

- Dodaliśmy możliwość odtwarzania przebiegu gry. Początkowo planowaliśmy dodanie interfejsu, w którym podaje się numer rozgrywki, a następnie otwiera się nowe okno z animacją. Stwierdziliśmy jednak, że nie jest to interfejs przyjazdy użytkownikowi i niepotrzebnie otwierane są kolejne, nowe okna. Problem rozwiązała funkcjonalność JavaFX pozwalająca na wykonanie akcji po najechaniu na np. obszar tekstu - `text.hoverProperty().addListener()`.
- W całym projekcie intensywnie korzystaliśmy z podobnych możliwości oferowanych przez JavaFX - np. `setOnCloseRequest()` pozwalało nam wykonać dodatkowe działania takie jak przerwanie konkretnego wątku przy zamykaniu któregoś z okien. Pozwoliły nam one rozwiązać problemy podobne do opisanego powyżej.

## - możliwości dalszego rozwoju aplikacji ✂

- Uważamy, że aplikacja ma duży potencjał na ewentualny rozwój w przyszłości. Sami zaczęliśmy implementację od podstawowej wersji kółko i krzyżyk ✕ ○, ale udało nam się stworzyć jej modyfikacje. Tryby takie jak *Speedrun* □, czy tryb z algorytmem heurystycznym po stronie komputera to jedno z wielu możliwości. Kreatywność nie zna granic, więc w grze można zaimplementować inne ciekawe modyfikacje podstawowej wersji gry.
- Nasza aplikacja jest desktopowa, więc gra w trybie *multiplayer* odbywa się na jednym urządzeniu. Potencjalnie można zmodyfikować ten tryb i umożliwić grę w dwie osoby na dwóch różnych maszynach. Wtedy jedna z maszyn byłaby klientem, a druga serwerem.

## - inne wnioski/spostrzeżenia 😊

### (Kwestia dystrybucji i udostępniania aplikacji użytkownikom)

- W naszej aplikacji wykorzystujemy biblioteki pakietu **JavaFX 16**, bibliotekę obsługującą driver do MySQL DB oraz bibliotekę dołączającą *framework* do testowania. Podczas pracy nad projektem powyższe biblioteki mieliśmy pobrane oraz dołączone w VS Code w zakładce *Referenced Libraries*. Jesteśmy jednak świadomi, że użytkownik chcący uruchomić grę nie będzie chciał tego robić przez IDE 📦 (nie musi być w szczególności programistą □). Gra powinna dać się uruchomić z poziomu pulpitu. Z tego względu zdecydowaliśmy się utworzyć plik **JAR**. Minusem takiego rozwiązania jest konieczność wskazania na lokalizację bibliotek JavaFX podczas uruchamiania programu (komenda poniżej):

```
java --module-path c:/javafx-sdk-16/lib/ --add-modules javafx.controls,javafx.fxml -jar Game.jar
```

Wymaga to od każdego użytkownika lokalnego zainstalowania bibliotek JavaFX, a według nas 99% potencjalnych graczy nie ma zainstalowanych wspomnianych bibliotek u siebie na komputerze 📁.

- W trakcie projektu nie korzystaliśmy z narzędzi automatyzujących budowę oprogramowania na platformę Java. Wynika to z faktu, iż dowiedzieliśmy się o nich z wykładu w momencie gdy projekt był już w zaawansowanej fazie. Mimo to pod koniec projektu wykorzystaliśmy narzędzie **Maven**. Pomogło nam ono uporządkować strukturę plików w katalogu. Najprawdopodobniej gdybyśmy wcześniej wiedzieli o narzędziach takich jak właśnie *Maven*, czy *Gradle* to użylibyśmy ich w projekcie ✂.

□ W trakcie pracy nad projektem tworzyliśmy dokumentację w postaci kolejnych stron w zakładce **Wiki** naszego repozytorium. Pod koniec 4-go etapu całość zawartości Wiki została przekonwertowana do pliku *.pdf* i umieszczona na czacie MS Teams, którego członkami są Prowadzący oraz studenci wykonujący projekt.

## Prezentacja 🗣️

Prezentacja wprowadzonych zmian odbyła się na platformie **MS Teams** □ 🗨️

## Elementy dodatkowe ✨

- Stworzona przez nas gra jest aplikacją desktopową, więc nie było (z definicji) możliwe jej wdrożenie w środowisku chmurowym ☁ (np. Heroku).
- Aby zwiększyć faktyczną użyteczność naszej aplikacji postanowiliśmy dodatkowo stworzyć plik **JAR** pozwalający uruchomić grę z poziomu pulpitu/ linii poleceń, a nie tylko przy użyciu IDE takiego jak VS Code.



## Opis klas

## Szczegółowy opis zadań poszczególnych klas

- poniżej opisano wszystkie klasy zaimplementowane w finalnej wersji projektu,
- w znakomitej większości przypadków omawiana klasa (*public*) jest jedyną klasą w pliku o tej samej nazwie -> jeżeli w danym pliku występują też inne klasy zostanie do odpowiednio zaznaczone,
- klasy opisano w porządku alfabetycznym.

### App

- klasa bazowa, odpowiada za uruchomienie pierwszego okna programu,
- inicjuje połączenie z *DB*,
- odpowiada za wyświetlenie ekranu startowego z animacją i załadowanie formatki *.fxml* z Menu.

### BaseMenager

- klasa ta posiada statyczne pola prywatne zawierające informacje umożliwiające połączenie z bazą danych (*username*, *password*, *connection string*),
- obsługuje również rozłączenie z bazą po wyjściu z gry,
- jej metody pozwalają na dodanie nowego gracza/zespołu/wyniku do DB jak i odczytanie tych rekordów z DB,
- obsługuje również komunikaty błędów związane z problem przy nawiązaniu połączenia z DB.

### BaseMenagerTest

- klasa testująca metody obsługi DB,
- zaimplementowano w niej testy jednostkowe z asercjami wykorzystując JUnit.

### Board (klasa wewnętrzna)

- na tej klasie oparte są wszystkie rodzaje rozgrywki,
- jej atrybutem jest *token*, czyli znak na polu planszy ( *o* lub *x* ),
- metody tej klasy pozwalają na pobranie/odczytanie znaku z danego pola oraz na postawienie nowego znaku przez gracza/komputer.

### ComputerRandom

- w klasie zaimplementowano obsługę gier w trybach *Easy* oraz *Medium*,
- zadaniem tej klasy jest utworzenie obszaru gry (*GridPane*) oraz obsługa interakcji z graczem klikającym odpowiednie miejsca na planszy,
- w trybie *Easy* komputer losowo wykonuje następny ruch, a w trybie *Medium* wykorzystuje algorytm zachłanny,
- do innych zadań klasy należy także: obsługa komunikatów informujących o rezultacie gry, czy ciągłe sprawdzanie sytuacji na planszy pod kątem ewentualnego remisu.

### GameBasic

- zadaniem klasy jest obsługa trybu *Multiplayer*,
- umożliwia dwóm osobom rozgrywkę na jednej maszynie,
- ma identyczne zadania jak klasa *ComputerRandom*.

### GameLogic

- klasa grupująca wszystkie metody odpowiedzialne za logikę gry w jednym miejscu,
- powstała wyłącznie na potrzeby testów jednostkowych z JUnit i nie zawiera elementów pakietu JavaFX takich jak pokazywanie Stage'a itd. - ich obecność podczas testowania powoduje problemy z oddzielnymi wątkami wykorzystywanymi przez JavaFX

### GameLogicTest

- klasa testująca logikę gry,
- zawiera proste testy jednostkowe z asercjami wykorzystujące JUnit.

## GameStatsController

---

- zadaniem tej klasy jest wyciągnięcie z DB odpowiednich rekordów oraz przedstawienie ich w przejrzystej dla użytkownika formie tabeli,

## HistoryStatsController

---

- klasa odpowiada za prezentację zapisanych historii ruchów z rozgrywek,
- komunikuje się z DB i wyciąga potrzebne rekordy w celu ich późniejszego wyświetlenia w formie animacji *mini-plans* z przebiegiem rozgrywki,

## MainController

---

- do zadań tej klasy należy "wstrzyknięcie" głównego *Pane'a* w formacie *.fxml* do kontrolera,
- ładuje również ekrany podrzędne go ekranu głównego.

## MenuController

---

- klasa obsługuje główne menu gry,
- do jej zadań należą m.in. przejście do menu wyboru gry, umożliwienie przejrzania statystyk, wyświetlenie instrukcji oraz wyjście z całej aplikacji po zakończeniu rozgrywki,

## MiniMax

---

- zawiera już opisaną klasę wewnętrzną *Board*,
- jej zadaniem jest określenie następnego ruchu komputera bazując na algorytmie heurystycznym,
- tak jak inne klasy obsługujące rozgrywkę sprawdza na bieżąco sytuację na planszy i odpowiada za interakcję z graczem przez komunikaty.

## SpeedRun

---

- zadaniem klasy jest symulacja ruchów po stronie komputera z ograniczonym od góry limitem czasu podawanym przez użytkownika na początku rozgrywki,
- częścią struktury tego trybu gry jest również poprawna obsługa interfejsu *Slider* i przekazanie wartości limitu czasowego do konstruktora klasy *SpeedRun*.

## StartGameController

---

- klasa zawiera metody, z których każda jest odpowiedzialna za obsłużenie konkretnego trybu rozgrywki,
- wspomniane metody wymagają od gracza podanie *nick'u*, oraz przygotowują odpowiedni *statement* do zapisania w DB,
- klasa ma także za zadanie obsługę interfejsów wyboru np. *checkbox* z wyborem poziomu trudności, *slider* z wyborem limitu czasu na ruch, czy pole wyboru figury,
- do jej funkcjonalności zalicza się również obsługa wielu gier, bez konieczności wielokrotnego uwierzytelniania oraz sprawdzenie, czy gracz już widnieje w DB.

## StatsController

---

- umożliwia przejście do ekranu z historią rozgrywek,
- zbiera dane z DB i wyświetla na ich podstawie wykres kołowy pokazujący wzajemny stosunek wygranych do przegranych i remisów,
- obsługuje ładowanie formatki *.fxml* z dedykowanym ekranem do wyświetlania statystyk.