

Program wspomagający przeprowadzenie turnieju szachowego

Zespołowe przedsięwzięcie inżynierskie

Prowadzący: Antoni Ligęza

Spis treści

1. Zespołowe przedsięwzięcie inżynierskie	3
1.1. Członkowie zespołu z określeniem funkcji	3
1.2. Uzasadnienie potrzeby realizacji projektu	3
1.3. Cele projektu	3
1.4. Zakres projektu	4
1.5. Grupy docelowe	5
1.6. Struktura podziału prac (zadań) - WBS	5
1.7. Regulamin turnieju	5
1.8. Rozgrywki w systemie szwajcarskim - zrezygnowano w czasie trwania projektu	7
1.8.1. Wytoczne Międzynarodowej Federacji Szachowej (FIDE)	8
1.8.2. Zalety systemu szwajcarskiego	9
1.8.3. Wady systemu szwajcarskiego	9
1.9. Harmonogram	9
1.9.1. Harmonogram prac poszczególnych członków zespołu	9
1.9.1.1. Obsługa GitHuba	11
2. Obsługa programu	17
3. Piotr Jabłoński	28
4. Mirosława Pelc	55

Zespołowe przedsięwzięcie inżynierskie

Zespołowe przedsięwzięcie inżynierskie oznaczać będzie projekt, działanie podjęte w realizacji postawionego celu, realizowane zespołowo.

Projekt jest odpowiedzią na problem/potrzebę, w określonej przestrzeni życia.

1.1. Członkowie zespołu z określeniem funkcji

1. Piotr Jabłoński - programista Java
2. Mirosława Pelc - programista Java
3. Mariusz Lorek - kierownik zespołu, testy programu, przygotowanie dokumentacji projektu

1.2. Uzasadnienie potrzeby realizacji projektu

Celem zespołowego przedsięwzięcia jest przygotowanie programu który wspomogę zorganizowanie turnieju szachowego w którym wzięść może udział dowolna, nieznana wcześniej liczba zawodników. Czas trwania turnieju jest ograniczony przez organizatora. Turniej szachowy jest organizowany cyklicznie, dlatego stworzenie programu wspomagającego jego obsługę znacznie ułatwi przeprowadzanie kolejnych edycji.

1.3. Cele projektu

1. Napisanie programu umożliwiającego przeprowadzenie turnieju szachowego.
2. Przygotowanie instrukcji obsługi do programu dla użytkownika
3. Przygotowanie dokumentacji dla projektu

1.4. Zakres projektu

1. Stworzenie programu do wspomagania organizacji turnieju szachowego według wytycznych zlecniodawcy
2. Stworzenie dokumentacji opisującej postępy prac nad tworzonym projektem z podziałem na czynności które ma wykonywać każdy z członków zespołu

Program ma pozwalać na:

1. Przeprowadzenie turnieju szachowego w systemie kołowym (każdy z każdym) - dwie rundy:
 - eliminacje
 - finał
2. Zapis stanu turnieju w dowolnym momencie
3. Wprowadzenie do programu danych o uczestnikach
 - imię
 - nazwisko
 - wiek
 - kategoria szachowa
4. Edycja danych uczestnika
5. Usunięcie uczestnika z listy uczestników turnieju
6. Podział uczestników turnieju na grupy zgodnie z wybranym kryterium, rosnąco lub malejąco:
 - alfabetycznie
 - według wieku
 - ręcznie
 - losowo
7. Ustalenie optymalnej liczby grup dla danej liczby uczestników turnieju
8. Określenie ilości szachownic na których będzie rozgrywany turniej
9. Określenie czasu trwania pojedynczej rozgrywki
10. Ustalanie uczestników każdego meczu - kolor pionków (biały, czarny) przydzielany do zawodników przed każdym spotkaniem
11. Punktowanie rozegranych spotkań
12. Możliwość wyboru uczestników do rundy finałowej

1.5. Grupy docelowe

Program przeznaczony dla organizatorów turniejów szachowych lub gier o zbliżonych zasadach (np. warcaby)

1.6. Struktura podziału prac (zadań) - WBS

1. Zebranie informacji na temat sposobu przeprowadzania turnieju szachowego od zleceniodawcy.
 - (a) Wybranie systemu w którym będzie przeprowadzany turniej.
 - (b) Przygotowanie regulaminu turnieju.
2. Projekt programu.
 - (a) Określenie jakie elementy muszą się znaleźć w programie
 - (b) Szablon programu
 - (c) Wybór środowiska programistycznego
 - (d) Rozdzielenie zadań dla programistów
3. Tworzenie programu/aplikacji
 - (a) Opracowanie narzędzi bazodanowych przechowujących informacje dotyczące turniejów
 - (b) Przygotowanie elementów środowiska graficznego
 - (c) Integracja narzędzi bazodanowych z elementami środowiska graficznego
 - (d) Wstępna wersja programu
 - (e) Testowanie
 - testy uczestników projektu
 - testy zamawiającego program
4. Eliminacja znalezionych błędów
5. Dodawanie kolejnych funkcji do programu
6. Końcowa wersja programu

1.7. Regulamin turnieju

1. Obowiązują przepisy gry międzynarodowej federacji szachowej (fide).
2. Każdy z zawodników powinien się kierować zasadami fair play.
3. Zasady rozgrywki:
 - (a) Turniej rozgrywany jest w dwóch fazach: grupowej i finałowej.

- (b) Tworzona jest lista startowa według przyjętych przez prowadzącego turniej kryteriów, domyślnie:
 - kategoria szachowa
 - wiek
 - alfabetycznie
- (c) Ilość grup jest zależna od liczby uczestników i ustala ją prowadzący.
- (d) Lista startowa dzielona jest na liczbę części równą liczbie grup
- (e) Następnie zawodnicy z każdej części są losowo rozmieszczani w grupach
- (f) W fazie grupowej prowadzone są rozgrywki, według zasady "każdy z każdym" w danej grupie
- (g) W fazie finałowej zawodnicy wyłonieni z grup (liczbę osób wychodzących z grup ustala prowadzący) grają między sobą.
- (h) Jeżeli zawodnicy grali ze sobą w rundzie eliminacyjnej to w finale przyjmuje się wynik rozgrywki z eliminacji
- (i) Czas trwania turnieju jest ograniczony, podany przez organizatora
 - 10 minut na przyjmowanie zgłoszeń(rejestrację),
 - 10 minut na losowanie spotkań,
 - Na każdą rozgrywaną partię przypada 10 minut. Każdy zawodnik ma 5 minut na wykonanie swoich posunięć.
- (j) Zawodnicy mają do dyspozycji zegar analogowy lub cyfrowy z 2 tarczami lub wyświetlaczami umożliwiającymi odmierzenie czasu rozgrywki dla każdego z zawodników osobno
- (k) Za zajęcie ustalonych przez prowadzącego miejsc w turnieju zawodnicy otrzymują nagrody przewidziane przez organizatora.
- (l) Jeśli prowadzący ustali, uczestnicy będą mieli obowiązek zapisywać swoje ruchy na przeznaczonych do tego kartach.
- (m) Każdy stanowisko do gry ma swój numer identyfikacyjny, który obowiązuje przy rozgrywkach.
- (n) W sali, w której odbywa się turniej szachowy zawodnicy jak i widzowie muszą zachować bezwzględną ciszę, aby nie przeszkadzać graczom w rozgrywce.
- (o) Jeśli jakiś uczestnik turnieju lub widz będzie podpowiadał innemu uczestnikowi, zawodnik otrzymuje od prowadzącego ostrzeżenie, w wypadku powtórzenia się sytuacji gracz któremu pomoc została ponownie udzielona może zostać zdyskwalifikowany z dalszych rozgrywek przez prowadzącego.
- (p) W turnieju obowiązuje punktacja
 - Zwycięstwo - 1pkt
 - Remis - 0,5pkt
 - Porażka - 0pkt
 - Punkty pomocnicze, wykorzystywane gdy kilku zawodników ma taką samą liczbę punktów głównych, przyznawane po zakończeniu etapu (eliminacji lub finału) według schematu:

- Punkty zawodników z którym dany zawodnik wygrał
 - Remis - połowę punktów zawodników z którym dany zawodnik zremisował
 - Porażka - Nie są przyznawane punkty pomocnicze
4. W razie rezygnacji lub dyskwalifikacji zawodnika z turnieju, rozgrywki, które rozegrał nie zostają anulowane, a osoby, które się z nim spotykają w kolejnych rozgrywkach wygrywają walkowerem (otrzymują 1pkt za zwycięstwo)
5. W Sali zostało wydzielone pięć części:
- (a) Pierwsza, w której znajdują się tylko i wyłącznie osoby rozgrywające mecz
 - (b) Druga, w której znajdują się widzowie bądź gracze, którzy obecnie nie rozgrywają żadnego spotkania
 - (c) Trzecia, w której znajdują się stanowiska do gry w szachy poza turniejem
 - (d) Czwarta, w której znajdują się gracze oczekujący na mecz
 - (e) Piąta, w której znajduje się tylko i wyłącznie prowadzący turniej szachowy bądź osoby, które za zezwoleniem mogą znajdować się w tej strefie
6. Zawodnicy, którzy nie grają lub czekają na swoją kolej w obrębie sali lub w niedalekiej odległości od niej w wypadku wezwania do rozgrywki powinni w trybie natychmiastowym zgłosić się do udziału w spotkaniu. W wypadku niestawienia się do rozegrania meczu zawodnik zostaje zdyskwalifikowany.
7. W przypadku, gdy:
- Zawodnik utrudnia przeprowadzanie rozgrywek może zostać zdyskwalifikowany z turnieju lub wyproszony z sali przez Prowadzącego.
 - Widz utrudnia przeprowadzanie rozgrywek może zostać wyproszony z sali przez Prowadzącego.
8. Udział w turnieju szachowym jest równoznaczny z zaakceptowaniem regulaminu

1.8. Rozgrywki w systemie szwajcarskim - zrezygnowano w czasie trwania projektu

System szwajcarski - W systemie szwajcarskim z góry określa się liczbę rund, które należy rozegrać.

Na rundę składają się bezpośrednie pojedynki (gry) rozgrywane jednocześnie.

Za zwycięstwo w grze uczestnik otrzymuje jeden punkt, za remis pół punktu (punktacja może być inna).

Dobór par przeciwników w kolejnych rundach zależy od wyników uzyskanych w poprzednich. Pary dobiera się w miarę możliwości spośród tych uczestników, którzy dotychczas zdobyli jednakową liczbę punktów.

Jeśli liczba uczestników zawodów jest nieparzysta, w każdej rundzie jeden z uczestników z najmniejszym dorobkiem punktowym, który jeszcze nie pauzował, otrzymuje wolny los (tzw. bye) czyli dostaje punkt bez gry.

Kojarzenie par w kolejnych rundach jest dość skomplikowane, ponieważ system musi wykluczyć możliwość dwukrotnego spotkania się tych samych przeciwników.

Dodatkową komplikacją jest konieczność zapewnienia "sprawiedliwego" przydziału kolorów bierok w kolejnych pojedynkach.

1.8.1. Wytyczne Międzynarodowej Federacji Szachowej (FIDE)

Międzynarodowa Federacja Szachowa (FIDE) opracowała precyzyjny regulamin rozgrywania zawodów systemem szwajcarskim. Przed zawodami zawodnicy są uszeregowani w kolejności punktacji rankingowej odzwierciedlającej aktualną siłę gry każdego zawodnika. Listy rankingowe są publikowane co miesiąc. Przed kojarzeniem I rundy listę tę dzieli się na dwie części. W górnej połowie listy znajdują się zawodnicy najwyższej szeregowani, w dolnej – pozostali. W pierwszej rundzie zawodnik z nr 1 spotka się z zawodnikiem najwyższej szeregowanym w dolnej grupie i następnie kolejni według tej zasady. Kolor bierok dla pierwszej pary jest losowany, następne pary zawodników otrzymają kolory bierok odmienne.

Podstawowe zasady kojarzenia par w systemie szwajcarskim zostały w regulaminie określone w następujący sposób:

1. dwóch zawodników nie może się spotkać więcej niż jeden raz;
2. zawodnik, który otrzymał punkt bez gry nie może otrzymać wolnego losu;
3. zawodnik może rozegrać jednym kolorem dwie partie z rzędu (lub trzy jeżeli trzecia to ostatnia partia turnieju);
4. kojarzenie do następnej rundy odbywa się w ramach grup punktowych z tą samą liczbą punktów, a jeśli to dla niektórych zawodników jest niemożliwe różnica punktowa pomiędzy kojarzonymi zawodnikami musi być najmniejsza z możliwych;
5. tak wielu zawodnikom, jak to tylko możliwe, należy przydzielić oczekiwany kolor; jest to kolor, którym rozegrali mniej partii niż drugim kolorem, a w przypadku równej liczby partii jest to kolor odmienny od koloru poprzedniej rundy (jeśli dwóch skojarzonych zawodników oczekuje na ten sam kolor – musi być spełniony warunek pkt 3, a oczekiwany kolor bierok otrzyma zawodnik, który ma bardziej nierówny przydział kolorów z poprzednich rund, dodatkowo – jeśli dwóch skojarzonych zawodników ma identyczną historię przydziału koloru z poprzednich rund – oczekiwany kolor otrzyma zawodnik wyżej szeregowany na liście);
6. zawodnik, który grał w poprzedniej rundzie z zawodnikiem o większej (mniejszej) liczbie punktów nie powinien być ponownie skojarzony z zawodnikiem o większej (mniejszej) liczbie punktów;
7. zawodnik, który grał dwie rundy wcześniej z zawodnikiem o większej (mniejszej) liczbie punktów nie powinien być ponownie skojarzony z zawodnikiem o większej (mniejszej) liczbie punktów.

Zasady 1-2 są **bezwarunkowe**, tzn. kojarzenie musi spełnić każdy z tych warunków. Zasady 3-4 są również **bezwarunkowe z wyjątkiem ostatniej rundy**, kiedy wolno

je złamać, jeśli dzięki temu uda się skojarzyć więcej par, w których zawodnicy będą mieli taką samą liczbę punktów. Zasady 5-8 są uszeregowane według ważności i muszą być stosowane we wszystkich przypadkach, w których nie są sprzeczne z zasadami ważniejszymi. Regulamin FIDE zawiera szczegółowy opis algorytmu kojarzenia par.

1.8.2. Zalety systemu szwajcarskiego

Ogromną zaletą systemu szwajcarskiego jest **możliwość rozegrania turnieju w jednej grupie z udziałem dużej liczby zawodników**. W turniejach szachowych rozgrywanych tym systemem nierzadko bierze udział **kilkuset zawodników o różnym poziomie gry**. Początkujący mogą w bezpośrednim pojedynku spotkać się z arcymistrzami, w jednym turnieju mają szansę pokonać zawodników uznawanych za silniejszych i szybko awansować w szachowej hierarchii. Ważne jest również, że **jedna słabsza gra nie przekreśla szans zawodnika**. Takich możliwości nie daje ani system kołowy (ze względu na dużą liczbę koniecznych gier) ani pucharowy, który eliminuje zawodnika po pierwszej porażce.

1.8.3. Wady systemu szwajcarskiego

Wadą systemu szwajcarskiego jest **spory wpływ czynnika losowego**, którego znaczenie ogranicza się poprzez ustalenie początkowej kolejności zawodników według siły gry (zazwyczaj na podstawie rankingu). System szwajcarski jest również **znacznie mniej obiektywny od systemu kołowego**. Zawodnik, który przegrywając w pierwszych rundach, w końcowych zdobywał punkty na słabszych przeciwnikach może w ostatecznej klasyfikacji wyprzedzić zawodnika, który dobrze grał z silniejszymi, lecz w końcowych rundach zdobył mało punktów. **Decydujące znaczenie ostatniej rundy stanowi o specyficznej atrakcyjności systemu szwajcarskiego**. System ten nie sprawdza się też dobrze, w momencie **gdy liczba zawodników jest niewiele większa od liczby rund do rozegrania**. Wówczas w końcowych rundach spotkać się mogą zawodnicy, których różnica punktów jest spora.

1.9. Harmonogram

1.9.1. Harmonogram prac poszczególnych członków zespołu

Mirosława Pelc oraz Piotr Jabłoński wspólna praca programistyczna
Mirosława Pelc - Odpowiedzialna w głównej mierze za interfejs graficzny

Piotr Jabłoński - Programowanie, algorytmy

Zadanie	Data rozpoczęcia	Data zakończenia
Przygotowanie klas odpowiadających za uczestnika, turniej, rozgrywkę; przygotowanie klas odpowiadających za uczestnika, turniej, rozgrywkę	6.10.2015	20.10.2015
Wyszukiwanie możliwych do wykorzystania elementów dostępnych w bibliotekach graficznych dla języka JAVA	6.10.2015	20.10.2015 - zadanie ciągle wykonywane przez cały czas trwania projektu
Integracja z bazą danych SQLite do przechowywania uczestników Integracja z bazą danych SQLite do przechowywania turniejów Integracja z bazą danych SQLite do przechowywania wyników pojedynczych rozgrywek	20.10.2015	27.10.2015
tabela - lista uczestników	27.10.2015	3.11.2015
dodawanie nowego uczestnika	27.10.2015	3.11.2015
usuwanie uczestnika	3.11.2015	10.11.2015
edycja uczestnika	3.11.2015	10.11.2015
dodawanie losowego uczestnika	10.11.2015	17.11.2015
symulacja ilości rozgrywek dla danej liczby uczestników, typu turnieju (systemem szwajcarskim / eliminacje grup)	10.11.2015	17.11.2015
podział graczy na grupy wg listy sortowanej po ustalanych przez prowadzącego turniej (dynamicznie w programie) warunkach takich, jak: kategoria zawodnika, wiek, nazwisko, imię lub przydział manualny	17.11.2015	24.11.2015
tworzenie początkowej listy graczy (sortowanie) do turnieju rozgrywanego systemem szwajcarskim (sortowanie po kategorii, wieku, nazwisko, imię)	17.11.2015	24.11.2015
dobieranie zawodników w pary dla systemu kołowego z eliminacjami w grupach - eliminacje wybór zawodników przechodzących do finałów w rozgrywkach z eliminacjami dobieranie zawodników w pary dla systemu kołowego z eliminacjami w grupach - finały	24.11.2015	1.12.2015
dobór zawodników w systemie kołowym (4 tyg!)		
lista wyników dla turnieju rozgrywanego systemem kołowym z eliminacjami	1.12.2015	8.12.2015
lista wyników dla turnieju rozgrywanego systemem szwajcarskim		
zastosowanie programu do prowadzenia kilku turniejów jednocześnie		
usprawnienia ergonomii interfejsu		praca ciągła do końca trwania projektu
usprawnienia estetyczne interfejsu		praca ciągła do końca trwania projektu

Zadanie	Data rozpoczęcia	Data zakończenia
Przygotowanie dokumentacji dla projektu		cały czas trwania projektu
Rozmowa ze zleceniodawcą na temat projektu		20.10.2015
Wybór systemu w którym przeprowadzany będzie turniej		27.10.2015
Okreslenie regulaminu turnieju (czas trwania, system rozgrywego, określenie zasad uczestnictwa w turnieju, powody do dyskwalifikacji)		3.11.2015
Opis repozytorium GitHub wykorzystywanego do pracy w projekcie		17.11.2015
Przygotowywanie kolejnych części dokumentacji na podstawie informacji dostarczonych przez pozostałych członków zespołu		
Testowanie kolejnych wersji programu, wyszukiwanie błędów sugestie na temat usprawnień - praca ciągła, do końca trwania projektu		
Konsultację ze zleceniodawcą na temat ewentualnych poprawek, dodawania nowych funkcjonalności wymaganych przez zleceniodawcę.		

Przygotowanie środowiska do równoległego opracowania dokumentacji projektu i realizacji przydzielonych zadań poszczególnym członkom zespołu projektowego.

1.9.1.1. Obsługa GitHuba

Repozytorium wykorzystywane w projekcie to "GitHub" aby zacząć korzystać z tego repozytorium należy najpierw założyć konto w serwisie <https://github.com> Wybieramy opcję "Sing up" i wypełniamy formularz rejestracyjny

Create your personal account

Username

This will be your username — you can enter your organization's username next.

Email Address

You will occasionally receive account related emails. We promise not to share your email with anyone.

Password

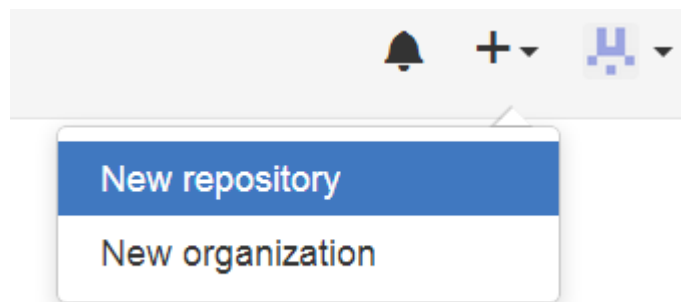
Use at least one lowercase letter, one numeral, and seven characters.

By clicking on "Create an account" below, you are agreeing to the [Terms of Service](#) and the [Privacy Policy](#).

Create an account

Rys. 1.1. Formularz rejestracyjny repozytorium GitHub

Następnie z menu na górze po prawej stronie wybieramy opcję "New repository"



Rys. 1.2. Tworzenie nowego repozytorium


Uzupełniamy dane dotyczące projektu. Musimy mu nadać nazwę, możemy opcjonalnie dodać opis tworzonego repozytorium, oraz zdecydować czy projekt będzie publiczny czy prywatny

Create a new repository

A repository contains all the files for your project, including the revision his

Owner

Repository name


 mariuszlor ▾ / testowe_rep ✓

Great repository names are short and memorable. Need inspiration? How about

Description (optional)

Przykładowy opis repozytorium

☒  **Public**
Anyone can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

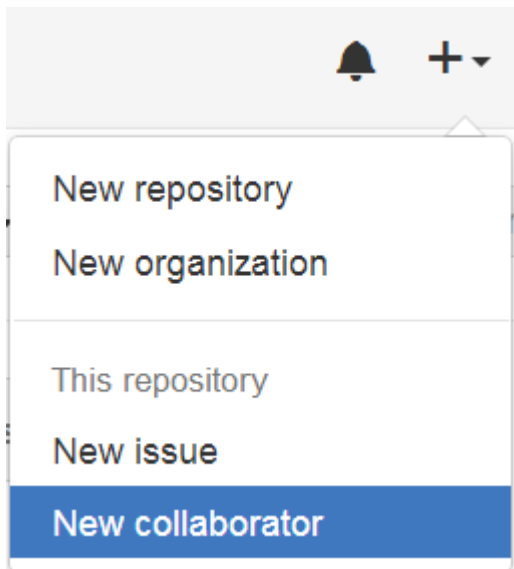
☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if y

Add .gitignore: **None** ▾ | Add a license: **None** ▾ ⓘ

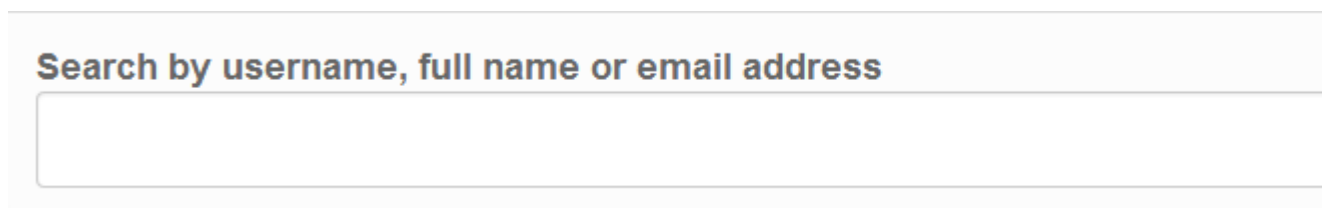
Create repository

Rys. 1.3. Uzupełniamy dane na temat projektu

Teraz możemy dodać kolejnych uczestników projektu wybierając z menu opcję "New collaborator" Uczestników możemy wyszukiwać według różnych kryteriów



Rys. 1.4. Dodawanie nowego uczestnika projektu



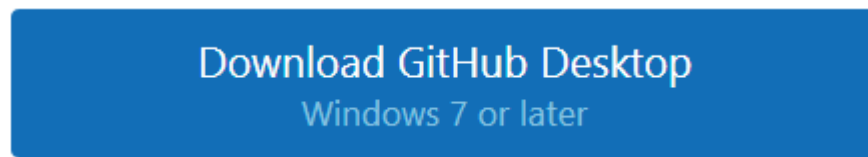
Rys. 1.5. Mamy możliwość wyszukiwania nowych członków według różnych kryteriów

Aby mieć możliwość wysyłania plików do repozytorium musimy zainstalować program na swoim systemie w tym celu wchodzimy na stronę <https://desktop.github.com/>. Program możemy zainstalować w systemach:

1. Windows 7
2. Windows 8/8.1
3. Windows 10

Starsze wersje systemów operacyjnych nie są wspierane

Dostępna jest również wersja dla komputerów MAC z systemem OS X 10.9 lub nowszym



Rys. 1.6. Przycisk umożliwiający pobranie programu

2

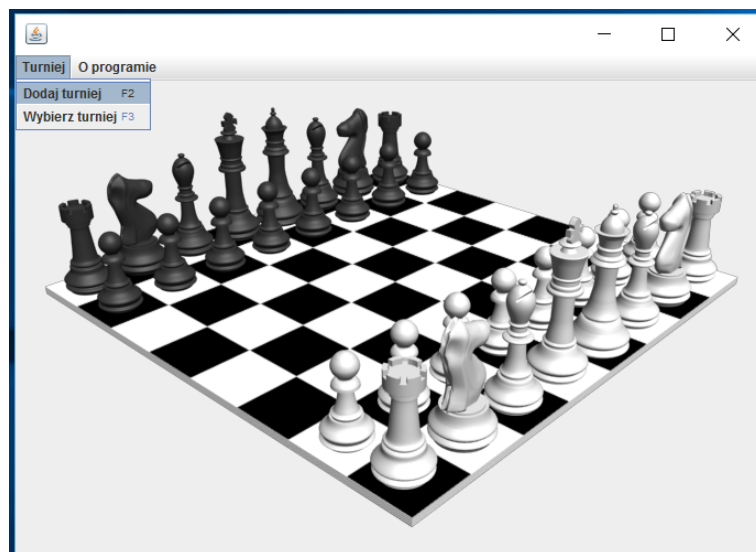
Obsługa programu

Uruchamiamy program



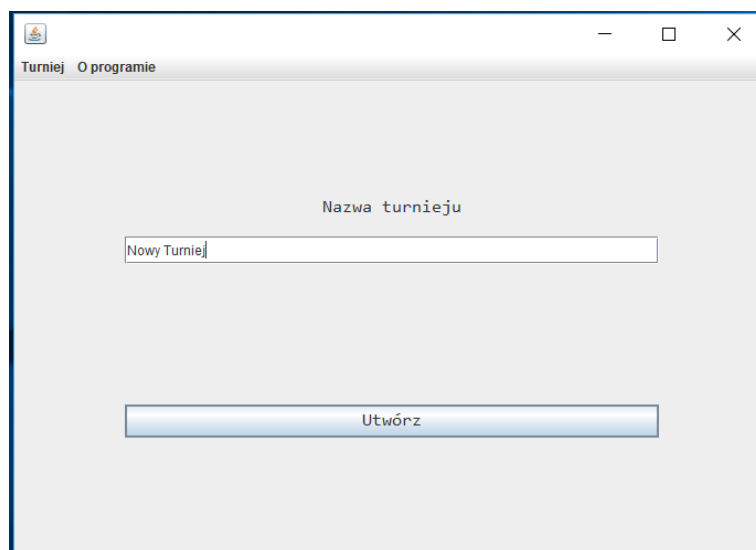
Rys. 2.1. Główne okno programu

Z menu "Turniej" możemy wybrać opcję lub użyć skrótu klawiaturowego



Rys. 2.2. Wybór turnieju

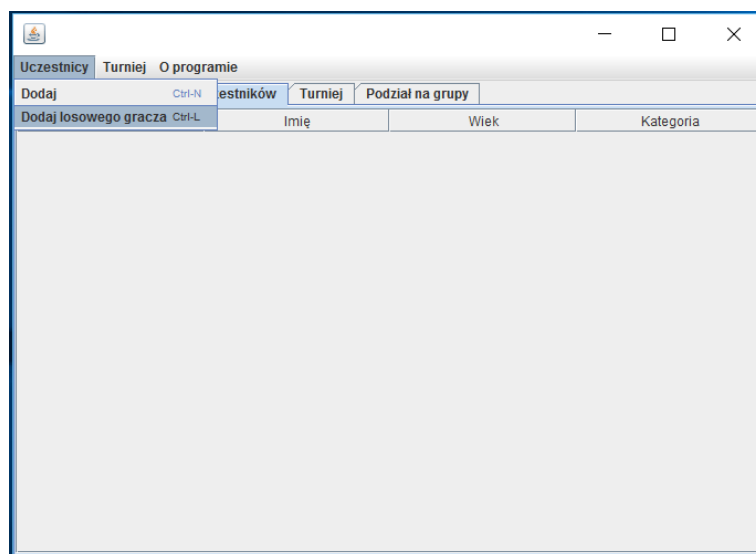
Jeżeli wybierzemy opcję "Dodaj turniej" musimy nadać mu nazwę



Rys. 2.3. Wybór nazwy turnieju

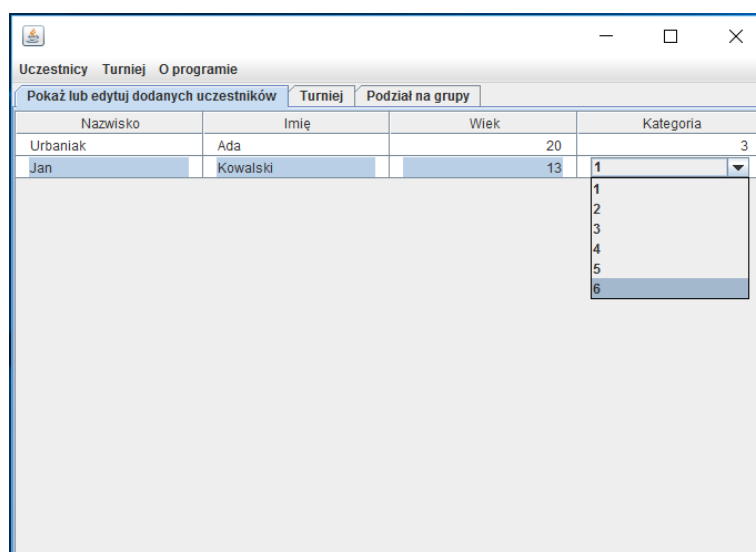
Uczestnika turnieju możemy dodać na 2 sposoby:

1. losowo
2. uzupełniając jego dane



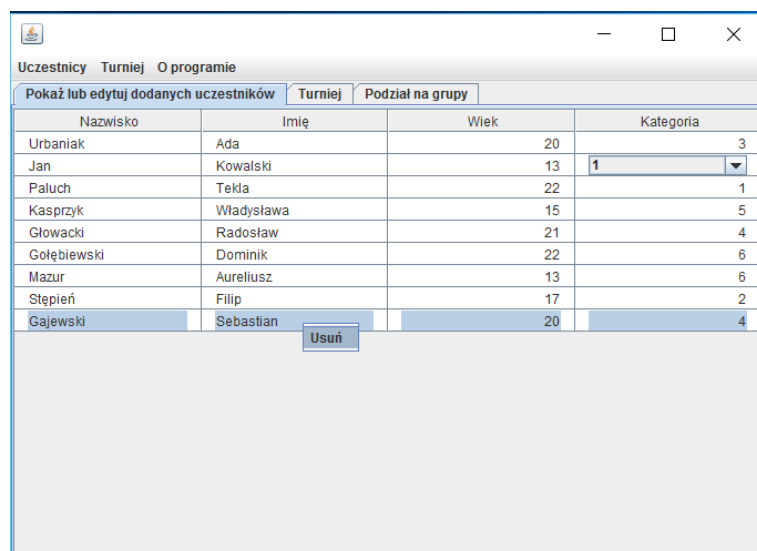
Rys. 2.4. Dodawanie uczestnika do turnieju

Wpisujemy dane użytkownika, kategorie możemy wybrać z listy



Rys. 2.5. Wybór kategorii

Istnieje możliwość usunięcia zawodnika z listy.
Klikamy PPM na wierszu w którym znajdują się dane zawodnika którego chcemy usunąć i wybieramy opcję "usuń"



Rys. 2.6. Usuwanie zawodnika z listy

W zakładce turniej możemy ustawić parametry turnieju takie jak:

1. nazwa turnieju
2. rok
3. czas trwania pojedynczej rozgrywki - minimum 1 minuta, maksimum 20 minut
4. ilość drużyn - program wskaże optymalną ilość grup, którą można zmienić
5. ilość szachownic - minimum 2, maksimum 20

Po ustawieniu żądanych parametrów program wyświetli:

- liczbę rozgrywek
- przewidywany czas trwania eliminacji

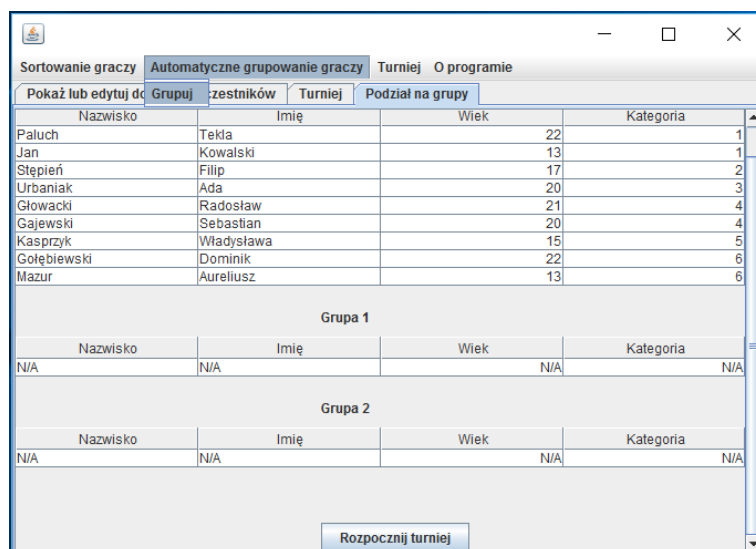
Rys. 2.7. Zakładka turniej

W zakładce "Podział na grupy" z menu "Sortowanie graczy" możemy wybrać sposób sortowania uczestników według kryteriów widocznych poniżej

Imię	Wiek	Kategoria
Tekla	22	1
Kowalski	13	1
Filip	17	2
Ada	20	3
Radosław	21	4
Sebastian	20	4
Władysława	15	5
Dominik	22	6
Aureliusz	13	6

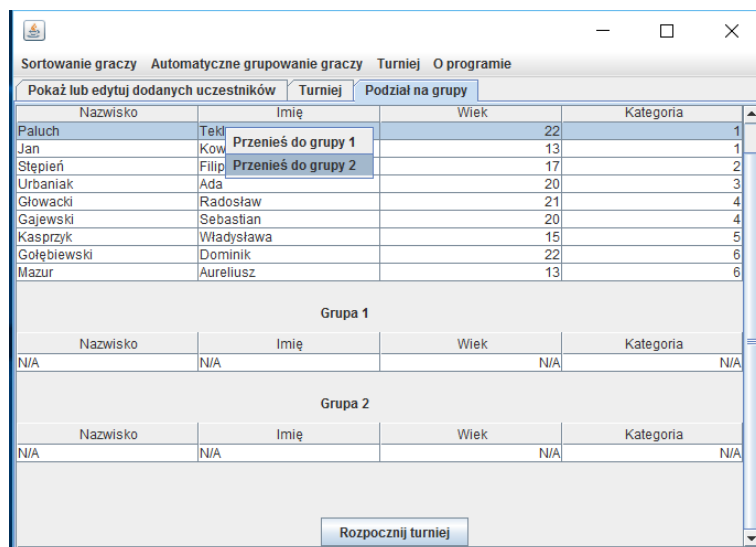
Rys. 2.8. Sposoby sortowania uczestników

Zawodników możemy przydzielić do grup automatycznie



Rys. 2.9. Automatyczny przydział do grup

Zawodnika do grupy możemy przydzielić ręcznie klikając PPM i wybierając z listy grupę do której chcemy przydzielić uczestnika



Rys. 2.10. Ręczny przydział do grup

Jeżeli przydzielimy wszystkich zawodników do grup możemy rozpocząć turniej

Sortowanie graczy Automatyczne grupowanie graczy Turniej O programie

Pokaż lub edytuj dodanych uczestników Turniej Podział na grupy

Kompletny podział na grupy

Grupa 1

Nazwisko	Imię	Wiek	Kategoria
Paluch	Tekla	22	1
Urbaniak	Ada	20	3
Gajewski	Sebastian	20	4
Kasprzyk	Władysława	15	5
Mazur	Aureliusz	13	6

Grupa 2

Nazwisko	Imię	Wiek	Kategoria
Jan	Kowalski	13	1
Stępień	Filip	17	2
Głowacki	Radosław	21	4
Gołębiowski	Dominik	22	6

Rozpocznij turniej

Rys. 2.11. Rozpoczęcie turnieju

Rozgrywki które mogą się odbyć zostają podświetlone na zielono
 Zwycięzcę rozgrywki możemy wybrać z listy lub za pomocą skrótu klawiaturowego:

- "b" - wygrywa kolor biały
- "c" - wygrywa kolor czarny
- "r" - remis

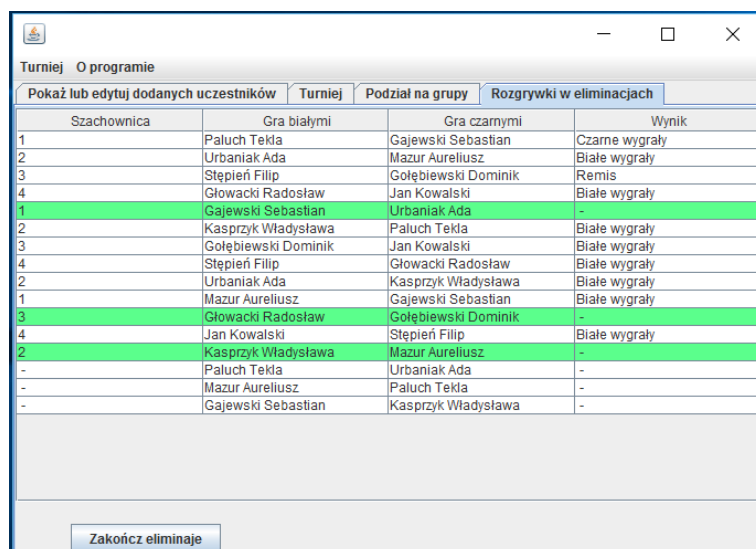
Turniej O programie

Pokaż lub edytuj dodanych uczestników Turniej Podział na grupy Rozgrywki w eliminacjach

Szachownica	Gra białymi	Gra czarnymi	Wynik
1	Paluch Tekla	Gajewski Sebastian	-
2	Urbaniak Ada	Mazur Aureliusz	-
3	Stępień Filip	Gołębiowski Dominik	Białe wygrały
4	Głowacki Radosław	Jan Kowalski	Czarne wygrały
-	Gajewski Sebastian	Urbaniak Ada	Remis
-	Kasprzyk Władysława	Paluch Tekla	-
-	Gołębiowski Dominik	Jan Kowalski	-
-	Stępień Filip	Głowacki Radosław	-
-	Urbaniak Ada	Kasprzyk Władysława	-
-	Mazur Aureliusz	Gajewski Sebastian	-
-	Głowacki Radosław	Gołębiowski Dominik	-
-	Jan Kowalski	Stępień Filip	-
-	Kasprzyk Władysława	Mazur Aureliusz	-
-	Paluch Tekla	Urbaniak Ada	-
-	Mazur Aureliusz	Paluch Tekla	-
-	Gajewski Sebastian	Kasprzyk Władysława	-

Zakończ eliminacje

Rys. 2.12. Możliwe rozgrywki

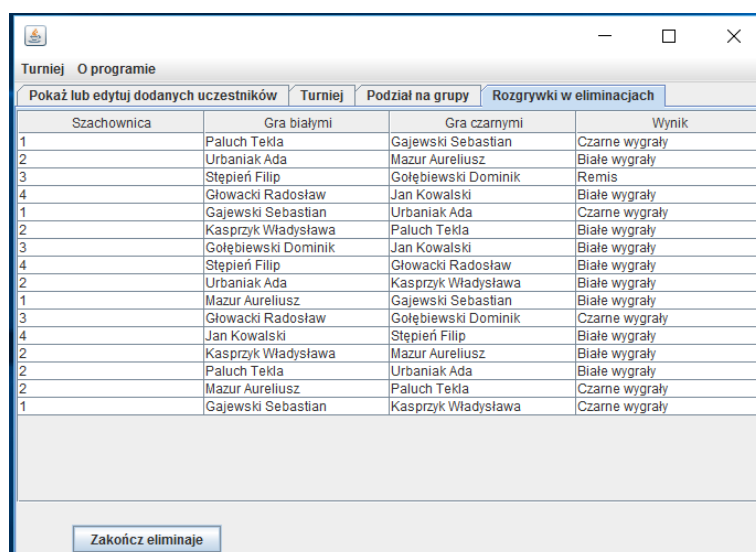


Szachownica	Gra białymi	Gra czarnymi	Wynik
1	Paluch Tekla	Gajewski Sebastian	Czarne wygrały
2	Urbaniak Ada	Mazur Aureliusz	Białe wygrały
3	Stępień Filip	Gołębiewski Dominik	Remis
4	Głowacki Radosław	Jan Kowalski	Białe wygrały
1	Gajewski Sebastian	Urbaniak Ada	-
2	Kasprzyk Władysława	Paluch Tekla	Białe wygrały
3	Gołębiewski Dominik	Jan Kowalski	Białe wygrały
4	Stępień Filip	Głowacki Radosław	Białe wygrały
2	Urbaniak Ada	Kasprzyk Władysława	Białe wygrały
1	Mazur Aureliusz	Gajewski Sebastian	Białe wygrały
3	Głowacki Radosław	Gołębiewski Dominik	-
4	Jan Kowalski	Stępień Filip	Białe wygrały
2	Kasprzyk Władysława	Mazur Aureliusz	-
-	Paluch Tekla	Urbaniak Ada	-
-	Mazur Aureliusz	Paluch Tekla	-
-	Gajewski Sebastian	Kasprzyk Władysława	-

Zakończ eliminacje

Rys. 2.13. Możliwe rozgrywki

Po ustaleniu wyników wszystkich rozgrywek możemy zakończyć eliminacje



Szachownica	Gra białymi	Gra czarnymi	Wynik
1	Paluch Tekla	Gajewski Sebastian	Czarne wygrały
2	Urbaniak Ada	Mazur Aureliusz	Białe wygrały
3	Stępień Filip	Gołębiewski Dominik	Remis
4	Głowacki Radosław	Jan Kowalski	Białe wygrały
1	Gajewski Sebastian	Urbaniak Ada	Czarne wygrały
2	Kasprzyk Władysława	Paluch Tekla	Białe wygrały
3	Gołębiewski Dominik	Jan Kowalski	Białe wygrały
4	Stępień Filip	Głowacki Radosław	Białe wygrały
2	Urbaniak Ada	Kasprzyk Władysława	Białe wygrały
1	Mazur Aureliusz	Gajewski Sebastian	Białe wygrały
3	Głowacki Radosław	Gołębiewski Dominik	Czarne wygrały
4	Jan Kowalski	Stępień Filip	Białe wygrały
2	Kasprzyk Władysława	Mazur Aureliusz	Białe wygrały
2	Paluch Tekla	Urbaniak Ada	Białe wygrały
2	Mazur Aureliusz	Paluch Tekla	Czarne wygrały
1	Gajewski Sebastian	Kasprzyk Władysława	Czarne wygrały

Zakończ eliminacje

Rys. 2.14. Wybór zawodników do finału

Przechodzimy do zakładki "Wybór graczy do finałów"

Z każdej grupy musimy wybrać co najmniej jednego zawodnika który będzie brał udział w finale

Turniej O programie

Rozgrywki w eliminacjach Wybór graczy do finałów

Pokaż lub edytuj dodanych uczestników Turniej Podział na grupy

Grupa 1						
Gracz	Wygranych	Przeigranych	Zakończonych r...	Punkty	Punkty SB	Wchodzi do fin...
Urbaniak Ada	3	1	0	3	5	<input checked="" type="checkbox"/>
Kasprzyk Wład...	3	1	0	3	4	<input checked="" type="checkbox"/>
Paluch Tekla	2	2	0	2	4	<input type="checkbox"/>
Gajewski Seba...	1	3	0	1	2	<input type="checkbox"/>
Mazur Aureliusz	1	3	0	1	1	<input type="checkbox"/>

Grupa 2						
Gracz	Wygranych	Przeigranych	Zakończonych r...	Punkty	Punkty SB	Wchodzi do fin...
Gołębiewski D...	2	0	1	2,5	2,75	<input checked="" type="checkbox"/>
Stepień Filip	1	1	1	1,5	2,25	<input type="checkbox"/>
Jan Kowalski	1	2	0	1	1,5	<input type="checkbox"/>
Głowacki Rado...	1	2	0	1	1	<input type="checkbox"/>

Rozpocznij fazę drugą

Rys. 2.15. Wybór graczy do finału

Jeżeli zawodnicy którzy dostali się do finału grali ze sobą wcześniej, nie grają ponownie. Przyjęty zostaje wynik z poprzedniej rundy.

Turniej O programie

Rozgrywki w eliminacjach Wybór graczy do finałów Rozgrywki finałowe

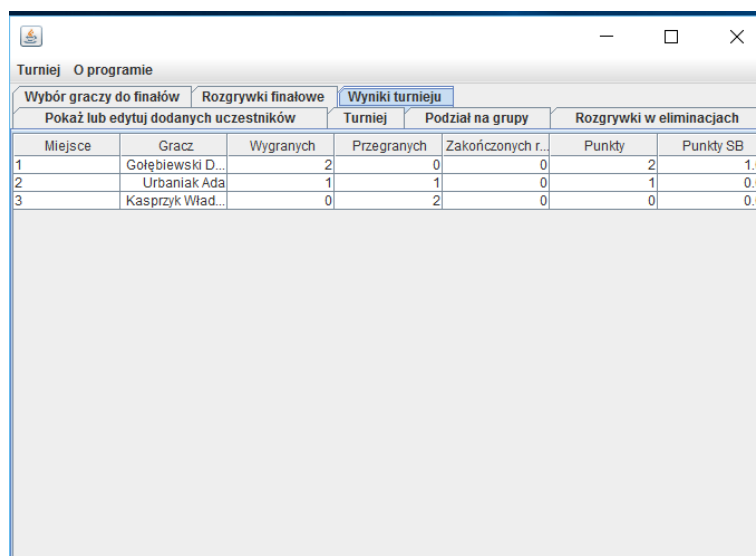
Pokaż lub edytuj dodanych uczestników Turniej Podział na grupy

Szachownica	Gra białymi	Gra czarnymi	Wynik
2	Urbaniak Ada	Kasprzyk Władysława	Białe wygrały
1	Gołębiewski Dominik	Urbaniak Ada	-
1	Kasprzyk Władysława	Gołębiewski Dominik	-

Zakończ turniej

Rys. 2.16. Rozgrywki finałowe

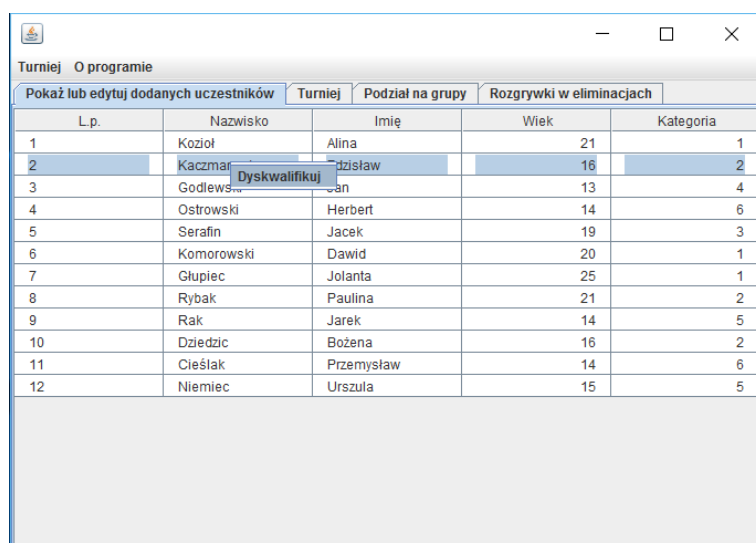
W zakładce "Wyniki turnieju" wyświetlone zostają wyniki z podziałem na miejsca



Turniej Wyniki turnieju						
Pokaż lub edytuj dodanych uczestników		Turniej	Podział na grupy	Rozgrywki w eliminacjach		
Miejsce	Gracz	Wygranych	Przegranych	Zakończonych r...	Punkty	Punkty SB
1	Gołębiewski D...	2	0	0	2	1.0
2	Urbaniak Ada	1	1	0	1	0.0
3	Kasprzyk Wład...	0	2	0	0	0.0

Rys. 2.17. Wyniki turnieju

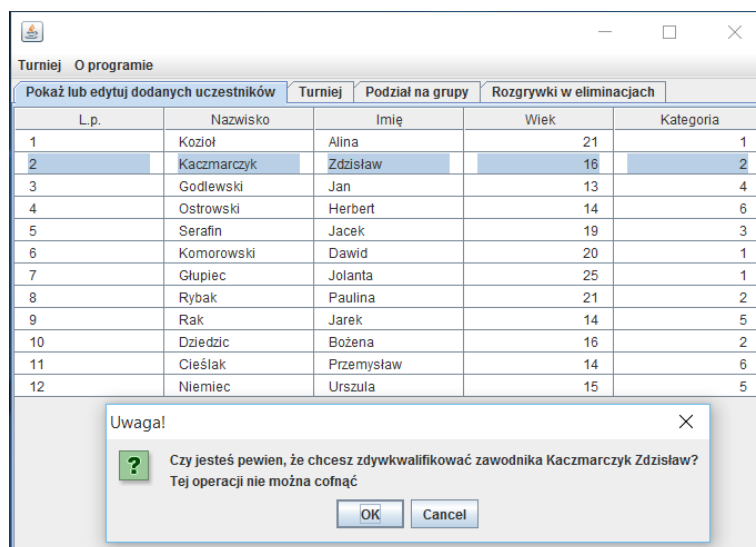
Po rozpoczęciu turnieju w zakładce "Pokaż lub edytuj dodanych uczestników" po kliknięciu PPM na wiersz z danymi uczestnika możemy wybrać opcję "Dyskwalifikuj".



L.p.	Nazwisko	Imię	Wiek	Kategoria
1	Kozioł	Alina	21	1
2	Kaczmarek	Stanisław	16	2
3	Godlewska	Jan	13	4
4	Ostrowski	Herbert	14	6
5	Serafin	Jacek	19	3
6	Komorowski	Dawid	20	1
7	Glupiec	Jolanta	25	1
8	Rybak	Paulina	21	2
9	Rak	Jarek	14	5
10	Dziedzic	Bożena	16	2
11	Cieślak	Przemysław	14	6
12	Niemiec	Urszula	15	5

Rys. 2.18. Dyskwalifikacja zawodnika

Uwaga!!! Z opcji dyskwalifikacji należy korzystać bardzo ostrożnie, tej operacji **nie** można cofnąć.



Rys. 2.19. Potwierdzenie dyskwalifikacji

Jeżeli zawodnik rozegrał już jakieś pojedynki to nie zostają one anulowane a kolejne w których miał brać udział przegrywa walkowerem (przeciwnik dostaje 1 punkt), do nazwiska zawodnika zostaje dodane oznaczenie dyskwalifikacji ”*d”

The screenshot shows the "Rozgrywki w eliminacjach" tab of the tournament management software. It displays a table of matches with columns for "Szachownica" (Board), "Gra białymi" (White), "Gra czarnymi" (Black), and "Wynik" (Result). The results show that Kaczmarczyk Zdzisław *d has lost all matches by walkover. A "Zakończ eliminacje" button is at the bottom.

Szachownica	Gra białymi	Gra czarnymi	Wynik
-	Kaczmarczyk Zdzisław *d	Ostrowski Herbert	Czarne wygrały
1	Kozioł Alina	Godlewski Jan	-
2	Niemiec Urszula	Głupiec Jolanta	-
3	Cieślak Przemysław	Dziedzic Bożena	-
4	Komorowski Dawid	Rybak Paulina	-
5	Serafin Jacek	Rak Jarek	-
-	Ostrowski Herbert	Godlewski Jan	-
-	Kaczmarczyk Zdzisław *d	Kozioł Alina	Czarne wygrały
-	Głupiec Jolanta	Dziedzic Bożena	-
-	Niemiec Urszula	Cieślak Przemysław	-
-	Rybak Paulina	Rak Jarek	-
-	Komorowski Dawid	Serafin Jacek	-
-	Kozioł Alina	Ostrowski Herbert	-
-	Godlewski Jan	Kaczmarczyk Zdzisław *d	Białe wygrały
-	Cieślak Przemysław	Głupiec Jolanta	-
-	Dziedzic Bożena	Niemiec Urszula	-
-	Serafin Jacek	Rybak Paulina	-
-	Rak Jarek	Komorowski Dawid	-

Zakończ eliminacje

Rys. 2.20. Tabela wyników po dyskwalifikacji uczestnika

Pierwszym krokiem tworzenia aplikacji do zarządzania turniejem szachowym było wymyślenie, w jaki sposób przechowywać dane o uczestnikach, turniejach, rozgrywkach. Początkowo zastosowałem bazę SQLite. Po licznych testach doszedłem do wniosku, że posługiwanie się tą bazą bardzo spowalnia program, dlatego zmieniłem zapis danych na zapis do plików. Baza danych programu szachowego wygląda następująco:

```
package model;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.TreeMap;
import java.util.stream.Collectors;

import tools.Dialogs;

/**
 * Klasa odpowiadająca za obsługę bazy danych.
 * Dane zapisywane są w plikach w postaci serializowanych list obiektów
 */
public class Database \{
    Map<Integer,Tournament> tournaments = new TreeMap<>();
    Map<Integer,Competitor> competitors = new TreeMap<>();
    Map<Integer,SingleGame> singleGames = new TreeMap<>();
    public Database() \{
        File theDir = new File("data");
```

```
// if the directory does not exist, create it
if (!theDir.exists()) \{
    System.out.println("Creating data/ directory");
    try\{
        theDir.mkdir();
    \}
    catch(SecurityException se)\{
        System.err.println("Could not create data/ directory");
    \}
\}
\}
\}
public void close()\{\}
private static Object readObject(String filename) \{
    Object result = null;
    try \{
        File file = new File("data/"+filename);
        FileInputStream f = new FileInputStream(file);
        ObjectInputStream s = new ObjectInputStream(f);
        result = s.readObject();
        s.close();
    \} catch(FileNotFoundException e) \{
        System.err.println("Warning: file "+filename+" can not be found");
    \} catch(IOException | ClassNotFoundException e) \{
        e.printStackTrace();
        Dialogs.bladBazy();
    \}
    return result;
\}
\}
public static <T> void writeObject(String filename, Map<Integer,T> o) \{
    try \{
        File file = new File("data/"+filename);
        FileOutputStream f = new FileOutputStream(file);
        ObjectOutputStream s = new ObjectOutputStream(f);
        s.writeObject(new ArrayList<T>(o.values()));
        s.close();
    \} catch(IOException e) \{
        e.printStackTrace();
        Dialogs.bladBazy();
    \}
\}
\}
@SuppressWarnings("unchecked")
public void readTournaments() \{
    ArrayList<Tournament> rawTournaments = (ArrayList<Tournament>)
    readObject("tournaments.data");
    tournaments = rawTournaments==null ? new TreeMap<>() : rawTournaments.stream()
    .collect(Collectors.toMap(o->o.getId(), o->o));
}
```

```
\}  
public void writeTournaments() \{  
    writeObject("tournaments.data", tournaments);  
}  
@SuppressWarnings("unchecked")  
public void readCompetitors(int t) \{  
    ArrayList<Competitor> rawCompetitors = (ArrayList<Competitor>)  
        readObject("competitors"+t+".data");  
    competitors = rawCompetitors==null ? new TreeMap<>() : rawCompetitors.stream()  
        .collect(Collectors.toMap(o->o.getId(), o->o));  
}  
public void writeCompetitors(int t) \{  
    writeObject("competitors"+t+".data", competitors);  
}  
@SuppressWarnings("unchecked")  
public void readSingleGames(int t) \{  
    ArrayList<SingleGame> rawGames = (ArrayList<SingleGame>)  
        readObject("games"+t+".data");  
    singleGames = rawGames==null ? new TreeMap<>() : rawGames.stream()  
        .collect(Collectors.toMap(o->o.getId(), o->o));  
}  
public void writeSingleGames(int t) \{  
    writeObject("games"+t+".data", singleGames);  
}  
public void insertOrUpdateCompetitor(Competitor c, Integer turniej) \{  
    if(c.getId()==null)  
        if(competitors.isEmpty()) c.setId(0);  
        else c.setId(Collections.max(competitors.keySet())+1);  
    competitors.put(c.getId(), c);  
    writeCompetitors(turniej);  
}  
public void insertOrUpdateTournament(Tournament t) \{  
    if(tournaments.isEmpty()) readTournaments();  
    if(t.getId()==null)  
        if(tournaments.isEmpty()) t.setId(0);  
        else t.setId(Collections.max(tournaments.keySet())+1);  
    tournaments.put(t.getId(), t.copy());  
    writeTournaments();  
}  
public void insertOrUpdateSingleGame(SingleGame g, Integer turniej) \{  
    if(g.getId()==null)  
        if(singleGames.isEmpty()) g.setId(0);  
        else g.setId(Collections.max(singleGames.keySet())+1);  
    singleGames.put(g.getId(), g);  
    writeSingleGames(turniej);  
}
```

```
public void insertOrUpdateSingleGame(List<SingleGame> insSingleGames, Integer turniej)
for(SingleGame g : insSingleGames) \{
if(g.getId()==null)
if(singleGames.isEmpty()) g.setId(0);
else g.setId(Collections.max(singleGames.keySet()+1);
singleGames.put(g.getId(), g);
\}
writeSingleGames(turniej);
\}
public List<Competitor> getCompetitors(int turniej) \{
readCompetitors(turniej);
return competitors.values().stream().collect(Collectors.toList());
\}
public List<Tournament> getTournaments() \{
readTournaments();
return tournaments.values().stream().collect(Collectors.toList());
\}
public List<SingleGame> getSingleGames(int turniej, boolean finally) \{
readSingleGames(turniej);
if(!finally) \{
return singleGames.values().stream()
.filter(sg->sg.getRound()!=-1)
.collect(Collectors.toList());
\}
else \{
List<Integer> finaleCompetitorsIds = competitors.values().stream()
.filter(c->c.getGoesFinal())
.map(c->c.getId())
.collect(Collectors.toList());
return singleGames.values().stream()
.filter(sg->
finaleCompetitorsIds.contains(sg.getCompetitorB()) \&\&
finaleCompetitorsIds.contains(sg.getCompetitorW())
).collect(Collectors.toList());
\}
\}
public void removeCompetitor(int id, int turniej) \{
competitors.remove(id);
writeCompetitors(turniej);
\}
\}
```

Na potrzeby klasy odpowiadającej za przechowywanie danych zastosowałem takie kolekcje jak: listę, mapę, treemap, arraylist.

Aby w łatwy sposób przekazywać dane pomiędzy interfejsem użytkownika a bazą danych zastosowałem element wzorca projektowego, którym jest model. Stworzyłem trzy modele: zawodnika, turnieju i rozgrywek. Oto ich kod:

```
//przechowuje dane użytkowników
```

```
package model;
```

```
import java.io.Serializable;
import java.text.Collator;
import java.util.Comparator;
import java.util.EnumMap;
import java.util.Locale;
```

```
import res.Strings;
import tools.ValidatorException;
```

```
/**
```

```
 * Przechowuje podstawowe dane o uczestniku: imię, nazwisko, wiek,
 * kategoria szachowa, id, grupa, czy uczestnik nie jest zdyskwalifikowany.
 * Definiuje sposoby sortowania uczestników
 */
```

```
public class Competitor implements Serializable, Comparable<Competitor> \{
private static final long serialVersionUID = -8356380635352407432L;
private String name;
    private String surname;
    private int age;
    private int chessCategory;
    private Integer id;
    private boolean isDisqualified;
    private Integer group;
    public static EnumMap<SortOption, Comparator<Competitor>> comparators;

    static \{
        comparators = new EnumMap<SortOption, Comparator<Competitor>>(SortOption.class);
        Collator collator = Collator.getInstance(new Locale(Strings.locale));
        comparators.put(SortOption.AGE\_ASC,
            (c1, c2) -> c1.getAge().compareTo(c2.getAge()));
        comparators.put(SortOption.AGE\_DESC,
            (c2, c1) -> c1.getAge().compareTo(c2.getAge()));
        comparators.put(SortOption.CHESSCATEGORY\_ASC,
            (c1, c2) -> c1.getChessCategory().compareTo(c2.getChessCategory()));
        comparators.put(SortOption.CHESSCATEGORY\_DESC,
            (c2, c1) -> c1.getChessCategory().compareTo(c2.getChessCategory()));
        comparators.put(SortOption.NAME\_ASC,
            (c1, c2) -> collator.compare(c1.getName(), c2.getName()));
        comparators.put(SortOption.NAME\_DESC,
            (c2, c1) -> collator.compare(c1.getName(), c2.getName()));
        comparators.put(SortOption.SURNAME\_ASC,
            (c1, c2) -> c1.getSurname().compareTo(c2.getSurname()));
```



```
comparators.put(SortOption.SURNAME\_DESC,
(c2, c1) -> c1.getSurname().compareTo(c2.getSurname()));
    \}

    public Competitor(Integer id, String name, String surname, int age, int chessCateg
        this.id = id;
        this.name = name;
        this.surname = surname;
        this.age = age;
        this.chessCategory = chessCategory;
        this.isDisqualified = isDisqualified;
        this.group = group;
    \}

    public void setId(Integer id) \{
        this.id = id;
    \}

    public Integer getId() \{
        return id;
    \}

    public String getName() \{
        return name;
    \}

    public void setName(String name) throws ValidatorException \{
        this.name = name;
    \}

    public void setAge(int age) throws ValidatorException \{
        if(age<0) throw new ValidatorException(Strings.negativeAge);
        this.age = age;
    \}

    public String getSurname() \{
        return surname;
    \}

    public void setSurname(String surname) throws ValidatorException \{
        this.surname = surname;
    \}

    public Integer getAge() \{
        return age;
    \}
```

```
public Integer getChessCategory() \{
    return chessCategory;
\}

public void setChessCategory(int chessCategory) \{
    this.chessCategory = chessCategory;
\}

public Boolean getIsDisqualified() \{
    return this.isDisqualified;
\}

public void setIsDisqualified(boolean isDisqualified) \{
    this.isDisqualified = isDisqualified;
\}

public Integer getRawGroup() \{
    return group;
\}

public Integer getGroup() \{
    if(group==null) return null;
    return group\%100;
\}

public void setGroup(Integer group) \{
    this.group = group;
\}

public void setGoesFinal(boolean goes) \{
    group\%=100;
    if(goes) group+=100;
\}

public boolean getGoesFinal() \{
    return group>=100;
\}

/**
 * Opcje sortowania dla listy uczestników
 */
public enum SortOption \{
NAME\_ASC, NAME\_DESC,
SURNAME\_ASC, SURNAME\_DESC,
AGE\_ASC, AGE\_DESC,
```

```
CHESSCATEGORY\_ASC, CHESSCATEGORY\_DESC  
\}
```

```
@Override  
public String toString() \{  
    if(getIsDisqualified()) return surname+" "+name+" *d";  
    return surname+" "+name;  
\}
```

```
@Override  
public boolean equals(Object obj) \{  
    if(obj instanceof Competitor) return this.getId() == ((Competitor) obj).getId();  
    return false;  
\}
```

```
@Override  
public int compareTo(Competitor c) \{  
    if(c.getId()-getId()!=0) return c.getId()-getId();  
    return toString().compareTo(c.toString());  
\}  
\}
```

```
//przechowuje dane użytkowników
```

```
package model;
```

```
import java.io.Serializable;
```

```
/**
```

```
 * Przechowuje dane o turnieju - nazwę, rok, ilość szachownic, rund wszystkich oraz ru  
 */
```

```
public class Tournament implements Serializable \{  
    private static final long serialVersionUID = 7913534186353148249L;  
    private Integer id;  
        private String name;  
        private String year;  
    private int boards;  
        private int rounds;  
        private int roundsCompleted;
```

```
    public Tournament(Integer id, String name, String year, int boards, int rounds, in  
        this.id = id;  
        this.name = name;  
        this.year = year;  
        this.boards = boards;  
        this.rounds = rounds;  
        this.roundsCompleted = roundsCompleted;
```

```
\}  
  
public Tournament copy() \{  
    return new Tournament(id, name, year, boards, rounds, roundsCompleted);  
\}  
  
public void setId(Integer id) \{  
    this.id = id;  
\}  
  
public Integer getId() \{  
    return id;  
\}  
  
public void setId(int newId) \{  
    id = newId;  
\}  
  
public String getName() \{  
    return name;  
\}  
  
public void setName(String name) \{  
    this.name = name;  
\}  
  
    public String getYear() \{  
return year;  
\}  
  
public void setYear(String year) \{  
this.year = year;  
\}  
  
public int getBoards() \{  
return boards;  
\}  
  
public void setBoards(int boards) \{  
this.boards = boards;  
\}  
  
public int getRounds() \{  
return rounds;  
\}
```

```
public void setRounds(int rounds) \{
    this.rounds = rounds;
\}

public int getRoundsCompleted() \{
    return roundsCompleted;
\}

public void setRoundsCompleted(int roundsCompleted, boolean overRide) \{
    if(roundsCompleted > this.roundsCompleted || overRide)
        this.roundsCompleted = roundsCompleted;
\}

public void setRoundsCompleted(int roundsCompleted) \{
    setRoundsCompleted(roundsCompleted, false);
\}

public boolean isPlayersEditAllowed() \{
    return roundsCompleted<0;
\}

public boolean isDisqualificationAllowed() \{
    return roundsCompleted==0 || roundsCompleted==2;
\}
\}

package model;

import java.io.Serializable;

/**
 * Przechowuje dane o pojedynczej rozgrywce - id graczy, wynik, oraz
 * informację w której rundzie (czy też w finałach) rozgrywka była zawarta
 */
public class SingleGame implements Serializable \{
    private static final long serialVersionUID = -3007183465072503118L;
    private Integer id;
    private final int competitorW, competitorB;
    private int score;
    private int board;
    private final int round;

    public SingleGame(Integer id, int competitorW, int competitorB, int score,
        int round, int board) \{
        this.id = id;
        this.competitorW = competitorW;
        this.competitorB = competitorB;
        this.score = score;
```

```
this.round = round;
this.board = board;
\}
```

```
public SingleGame(Competitor competitorW, Competitor competitorB, int round, int board)
{
    this.id = null;
    this.competitorW = competitorW.getId();
    this.competitorB = competitorB.getId();
    this.score = 0;
    this.round = round;
    this.board = board;
}
```

```
public SingleGame(Competitor c, int round, int score) \{ // swiss bye / disqualified
{
    this.id = null;
    this.competitorB = c.getId();
    this.competitorW = c.getId();
    this.score = score;
    this.round = round;
}
```

```
public Integer getId() \{
{
    return id;
}
```

```
public void setId(int id) \{
{
    this.id = id;
}
```

```
public int getCompetitorW() \{
{
    return competitorW;
}
```

```
public int getCompetitorB() \{
{
    return competitorB;
}
```

```
public int getScore() \{
{
    return score;
}
```

```
public void setScore(int score) \{
{
    this.score = score;
}
```

```
public Integer getRound() \{
```

```

return round;
\}

public Integer getBoard() \{
return board;
\}

public void setBoard(int board) \{
this.board = board;
\}

@Override
    public boolean equals(Object obj) \{
        if(obj instanceof SingleGame) \{
            SingleGame sg1 = this, sg2=(SingleGame)obj;
            if(sg1.getCompetitorW()==sg2.getCompetitorW() \&\& sg1.getCompetitorB()==sg2.getCompetitorB()) \&\& sg1.getCompetitorW()==sg2.getCompetitorW() \&\& sg1.getCompetitorB()==sg2.getCompetitorB()) \{
                return true;
            \}
            return false;
        \}
    \}

```

Modele zawierają settery, które przechowują wartości i gettery, które te wartości zwracają.

Na potrzeby programu stworzyłem klasę abstrakcyjną, która zawiera metody wykorzystywane przy rozgrywkach: - metoda wykorzystana przy dyskwalifikacji uczestników

```

protected void setDisqualifiedPlayersScores() \{
sortGames();
for(SingleGame sg : singleGames) \{
if(sg.getScore()==0) \{
Competitor cW = competitorMap.get(sg.getCompetitorW());
Competitor cB = competitorMap.get(sg.getCompetitorB());
if(cW.getIsDisqualified() \&\& cB.getIsDisqualified()) sg.setScore(3);
else if(cW.getIsDisqualified()) sg.setScore(2);
else if(cB.getIsDisqualified()) sg.setScore(1);
if(sg.getScore()!=0) DB.insertOrUpdateSingleGame(sg, turniej.getId());
\}
\}
\}

```

- metoda odpowiedzialna za ustawianie klawiszami wyników, np. B oznacza, że wygrał gracz z białymi pionkami

```

protected static void mapKeyActions(JTable table) \{
InputMap im = table.getInputMap(JTable.WHEN\_FOCUSED);
ActionMap am = table.getActionMap();

```

```

im.put(KeyStroke.getKeyStroke(KeyEvent.VK\_BACK\_SPACE, 0), Strings.notPlayedYet);
im.put(KeyStroke.getKeyStroke(KeyEvent.VK\_DELETE, 0), Strings.notPlayedYet);
im.put(KeyStroke.getKeyStroke(KeyEvent.VK\_B, 0), Strings.whiteWon);
im.put(KeyStroke.getKeyStroke(KeyEvent.VK\_C, 0), Strings.blackWon);
im.put(KeyStroke.getKeyStroke(KeyEvent.VK\_R, 0), Strings.tie);
am.put(Strings.notPlayedYet, scoreUpdateAction(table, Strings.notPlayedYet));
am.put(Strings.whiteWon, scoreUpdateAction(table, Strings.whiteWon));
am.put(Strings.blackWon, scoreUpdateAction(table, Strings.blackWon));
am.put(Strings.tie, scoreUpdateAction(table, Strings.tie));
\}

```

- metoda, która odpowiada za aktualizację wyników

```

private static Action scoreUpdateAction(JTable table, String action) \{
MyTableModel model = (MyTableModel) table.getModel();
return new AbstractAction() \{
private static final long serialVersionUID = -5143500614268433363L;
@Override
    public void actionPerformed(ActionEvent e) \{
int rowCount = table.getRowCount(), rowSelected = table.getSelectedRow();
if(rowSelected>=0 \&\& rowSelected<rowCount \&\& model.isCellEditable(rowSelected, 3))
model.setValueAt(action, rowSelected, 3);
table.changeSelection(rowSelected, 3, false, false);
\}
    \}
\};
\}

```

Ważną częścią programu jest są napisane przeze mnie metody, które wyliczają ilość rozgrywek w zależności od ilości zawodników. Oto ich kod:

```

static void rozgrywek\_finaly(int zawodnikow, int izg) \{
System.out.print("Finały: zawodnikow - "+zawodnikow+", "
+izg*(izg-1)+" rozgrywek odbyło się już w fazie eliminacji, pozostało rozgrywek "+
(zawodnikow*(zawodnikow-1)-(zawodnikow/izg)*izg*(izg-1)));
\}

public static int rozgrywek\_eliminacje(int zawodnikow, int grup) \{
List<Integer> grupy = new ArrayList<Integer>();
while(zawodnikow>0) \{
int t = zawodnikow/grup;
grupy.add(t);
zawodnikow-=t;
grup--;
\}
int rozgrywek = 0;
for(Integer i : grupy) \{
rozgrywek+=i*(i-1)/2;
\}

```



```
return rozgrywek;
\}
```

```
public static void roundRobinTable(int g) \{
int gp = (g%2==1) ? g+1 : g;
for(int i=1; i<gp; ++i) \{
if(i%2==1) \{
System.out.print((1+i/2)+"-"+gp);
for(int j=2; j<=gp/2; ++j) \{
System.out.print("\textbackslash{\}t"+(j+i/2)+"-"+((gp-j+i/2)\%(gp-1)+1));
\}
\}
else \{
System.out.print(gp+"-"+(gp/2+i/2));
for(int j=2; j<=gp/2; ++j) \{
System.out.print("\textbackslash{\}t"+((gp/2+j+i/2-2)\%(gp-1)+1)+"-"+((gp/2-j+i/2)\%(gp-1)+1));
\}
\}
System.out.print("\textbackslash{\}n");
\}
\}
```

```
static String pad(String toPad) \{
return (toPad+"").substring(0, 20);
\}
```

Na pola tekstowe komórek JTable i nie tylko nałożyłem ograniczenia w postaci walidacji.

```
private static final long serialVersionUID = -489930074061735703L;
```

```
@Override
public void insertString(int offs, String str, AttributeSet a)
throws BadLocationException \{
if(str.length()>50-offs) str = str.substring(0, 50-offs);
str = str.replaceAll(Strings.forbiddenCharsRegExp, "");
super.insertString(offs, str, a);
\}
```

Poniższy kod wypełnia danymi z bazy pola JTable w zakładce pierwszej:

```
protected class EditCompetitorTableModel extends AbstractTableModel \{
private static final long serialVersionUID = 5974959488451548395L;
final String[] columnNames = \{"L.p.", "Nazwisko", "Imię", "Wiek", "Kategoria"\};
@Override
public Class<?> getColumnClass(int c) \{
return (c==3 || c==4) ? Integer.class : String.class;
\}
```

```

        @Override
        public boolean isCellEditable(int row, int column) \{
            if(column==1 || column==2 || column==3) return true;
            return turniej.isPlayersEditAllowed();
        \}
    @Override
    public int getColumnCount() \{
        return 5;
    \}
    @Override
    public String getColumnName(int columnIndex) \{
        return columnNames[columnIndex];
    \}
    @Override
    public int getRowCount() \{
        return competitors.size();
    \}
    @Override
    public Object getValueAt(int row, int col) \{
        Competitor c = competitors.get(row);
        if(col==0) return row+1;
        if(col==1) return c.getSurname();
        if(col==2) return c.getName();
        if(col==3) return c.getAge();
        if(col==4) return c.getChessCategory();
        return null;
    \}
    @Override
    public void setValueAt(Object value, int row, int column) \{
        Competitor c = competitors.get(row);
        try \{
            switch(column) \{
                case 1: c.setSurname((String)value); break;
                case 2: c.setName((String)value); break;
                case 3: c.setAge((int)value); break;
                case 4: c.setChessCategory((int)value); break;
            \}
            DB.insertOrUpdateCompetitor(c, turniej.getId());
        \} catch(ValidatorException exc) \{
            System.out.print("Błąd walidacji\textbackslash\{\\n"+exc.getMessage());
        \}
        \}
    \}

```

Do tabeli dodałem opcje „Usuń” i „Dyskwalifikuj”, które można wybrać po kliknięciu prawym przyciskiem myszy na wiersz tabeli. Wówczas pojawi się tzw. PopupMenu.

```

table.addMouseListener(new MouseAdapter() \{
    @Override
    public void mouseReleased(MouseEvent e) \{
        int r = table.rowAtPoint(e.getPoint());
        if(r >= 0 \&\& r < table.getRowCount())
            table.setRowSelectionInterval(r, r);
        else
            table.clearSelection();

        final int rowindex = table.getSelectedRow();
        if(rowindex < 0) return;
        if(e.isPopupTrigger() \&\& e.getComponent() instanceof JTable ) \{
            JPopupMenu popup = new JPopupMenu();
            Competitor c = competitors.get(rowindex);
            if(turniej.isPlayersEditAllowed()) \{
                JMenuItem jmi = new JMenuItem(Strings.remove);
                jmi.addActionListener(e2 -> \{
                    DB.removeCompetitor(c.getId(), turniej.getId());
                    setData();
                \});
            }
            popup.add(jmi);
            \}
            if(turniej.isDisqualificationAllowed() \&\& !c.getIsDisqualified())
                JMenuItem jmi = new JMenuItem(Strings.disqualify);
                jmi.addActionListener(e2 -> \{
                    if(!Dialogs.czyZdyskwalifikowac(c)) return;
                    c.setIsDisqualified(true);
                    DB.insertOrUpdateCompetitor(c,turniej.getId());
                    setData();
                \});
            popup.add(jmi);
            \}
            if(popup.getComponentCount(>0) popup.show(e.getComponent(), e.get
        \}
    \}
\});

```

Poniższe metody odświeżają widok JTable danymi pobranymi z bazy.

```

public void setData() \{
    competitors=DB.getCompetitors(turniej.getId());
    ((EditCompetitorTableModel)table.getModel()).fireTableDataChanged();
\}

public void selectLast() \{
    table.changeSelection(table.getRowCount()-1, 1, false, false);
\}

```

Stworzyłem również klasę odpowiadającą za grupowanie graczy. Skorzystałem z zaprogramowanego przez programistę interfejsu graficznego naszej grupy panelu. Połączyłem interfejs graficzny oraz stworzone przeze mnie metody w jedną całość. Dodałem możliwość sortowania graczy przed przydzieleniem do grup, możliwość przydzielenia już dopasowanego gracza do innej grupy (prawy przycisk myszy na wiersz tabeli). Zaprogramowałem mechanizm grania każdy z każdym. Jest to specyficzny rodzaj turnieju kołowego, którego autorem jest zamawiający program.

```
public GroupsPanel(Tournament t, Database db, onTournamentStartListener listener)\{
    this.turniej = t;
    this.DB = db;
    this.setLayout(new BorderLayout());
    container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
    add(new JScrollPane(container));
    initComponents();
    startTournament.setVisible(isEditAllowed());
    startTournament.addActionListener(e -> \{
        Tools.checkGroups(turniej.getRounds(), competitors);
        if(competitors.stream().filter(c->c.getGroup()==null).count()>0)
            Dialogs.graczBezGrupy();
        else \{
            competitors.forEach(c->DB.insertOrUpdateCompetitor(c, turniej.getId())); // słaba wydajność
            int min = competitors.size();
            int max = 0;
            TreeMap<Integer, List<Competitor>> groupsList = Tools.groupsList(competitors);
            for(List<Competitor> groupL : groupsList.values() ) \{
                int size = groupL.size();
                min = Math.min(min, size);
                max = Math.max(max, size);
            \}
            if(max>min+1)
                Dialogs.nierownomiernyPodzial(min, max);
            else \{
                DB.insertOrUpdateSingleGame(
                    Tools.generateSingleGames(groupsList, turniej.getBoards()),
                    turniej.getId());
                startTournament.setVisible(false);
                listener.onTournamentStart();
            \}
        \}
    \});
\}

@FunctionalInterface
public interface onTournamentStartListener \{
    public void onTournamentStart();
\}
```

```
public void initComponents() \{
    competitors = DB.getCompetitors(turniej.getId());
    sortDefault();
    final int groups = turniej.getRounds();
    Tools.checkGroups(groups, competitors);
    container.removeAll();
    tables.clear();
    label = new JLabel(Strings.players, JLabel.CENTER);
    label.setAlignmentX(JLabel.CENTER\_ALIGNMENT);
    container.add(label);
    container.add(Box.createRigidArea(new Dimension(0, 10)));
    tableN = new JTable(new MyTableModel(null));
    tables.put(null, tableN);
    tableN.addMouseListener(new MyMouseListener(tableN, groups));
        container.add(tableNHeader = tableN.getTableHeader());
        container.add(tableN);
        container.add(rigridAfterN = Box.createRigidArea(new Dimension(0, 20)));

    for(int i=0; i<groups; ++i) \{
        container.add(new JLabel(Strings.group+(i+1), JLabel.CENTER));
        container.add(Box.createRigidArea(new Dimension(0, 10)));
        JTable table = new JTable(new MyTableModel(i));
        container.add(table.getTableHeader());
        container.add(table);
        tables.put(i, table);
        container.add(Box.createRigidArea(new Dimension(0, 20)));
            table.addMouseListener(new MyMouseListener(table, groups));
    \}

    container.add(Box.createRigidArea(new Dimension(0, 30)));
    container.add(startTournament);
    updateTables();
    \}

void autoGroup() \{
    int groups = turniej.getRounds(), i = groups, n = -1;
    List<List<Competitor>> groupsLists = new ArrayList<>();
    for(Competitor c : competitors) \{
        if(++i>=groups) \{
            i = 0;
            groupsLists.add(new ArrayList<>());
            n++;
        \}
        groupsLists.get(n).add(c);
    \}
```

```
while(++i<groups) \{ // dopełnienie ostatniej grupy wartościami "pustymi"
groupsLists.get(n).add(new Competitor(null, "", "", 0, 0, false, 0)); //dodać 0
\}
for(List<Competitor> l : groupsLists) \{
Collections.shuffle(l);
int g = 0;
for(Competitor c : l) c.setGroup(g++);
\}
updateTables();
\}

void sortDefault() \{
stableSort(Competitor.SortOption.NAME\_ASC);
stableSort(Competitor.SortOption.SURNAME\_ASC);
stableSort(Competitor.SortOption.AGE\_DESC);
stableSort(Competitor.SortOption.CHESSCATEGORY\_ASC);
\}

void stableSort(Competitor.SortOption o) \{
competitors.sort(Competitor.comparators.get(o));
updateTables();
\}

void shuffle() \{
Collections.shuffle(competitors);
\}

void updateTables() \{
tables.values().forEach((t) -> ((AbstractTableModel)t.getModel()).fireTableDataChanged);
if(tableN!=null \&\& rigridAfterN!=null \&\& tableNHeader!=null) \{
boolean allHaveGroup = (((MyTableModel)tableN.getModel()).rawGetRowCount()==0);
tableN.setVisible(!allHaveGroup);
rigridAfterN.setVisible(!allHaveGroup);
tableNHeader.setVisible(!allHaveGroup);
label.setText(allHaveGroup?"Kompletny podział na grupy":"Uczestnicy nieprzydzieleni do
\}
\}

public boolean isEditAllowed() \{
return turniej.getRoundsCompleted()<0;
\}

class MyMouseListener extends MouseAdapter \{
final int groups;
final JTable table;
MyMouseListener(JTable table, int groups) \{
```

```

super();
this.table = table;
this.groups = groups;
\}
@Override
    public void mouseReleased(MouseEvent e) \{
if(!isEditAllowed()) return;
    int r = table.rowAtPoint(e.getPoint());
    if(r >= 0 \&\& r < table.getRowCount())
        table.setRowSelectionInterval(r, r);
    else
        table.clearSelection();

    final int rowindex = table.getSelectedRow();
    if(rowindex < 0) return;
    if(e.isPopupTrigger() \&\& e.getComponent() instanceof JTable) \{
        JPopupMenu popup = new JPopupMenu();
        if(((MyTableModel) table.getModel()).competitors.isEmpty()) return;
        Competitor c = ((MyTableModel) table.getModel()).competitors.get(rowindex);
        if(c.getGroup()!=null) \{
            MoveToAnotherGroupMenuItem jmiNull = new MoveToAnotherGroupMenuItem(c);
            popup.add(jmiNull);
        \}
        for(int i=0; i<groups; ++i) \{
            if(c.getGroup()!=null \&\& c.getGroup()==i) continue;
            MoveToAnotherGroupMenuItem jmi = new MoveToAnotherGroupMenuItem(i, c);
            popup.add(jmi);
        \}
        popup.show(e.getComponent(), e.getX(), e.getY());
    \}
\}

class MoveToAnotherGroupMenuItem extends JMenuItem \{
private static final long serialVersionUID = -70690392582608352L;

public MoveToAnotherGroupMenuItem(Integer group, Competitor c) \{
    super(group==null?"Usuń z grupy":"Przeniesienie do grupy "+(group+1));
    addActionListener(new ActionListener() \{
        @Override
        public void actionPerformed(ActionEvent e) \{
            Integer oldGroup = c.getGroup();
            c.setGroup(group);
            DB.insertOrUpdateCompetitor(c, turniej.getId());
            ((AbstractTableModel)table.getModel()).fireTableDataChanged();
            ((AbstractTableModel)table.getModel()).fireTableDataChanged();
        \}
    \}
\}

```

```
\}
\});
\}
\}
```

Ważną częścią każdego turnieju szachowego są fazy eliminacyjne. W programie zaimplementowałem wymyślony przeze mnie algorytm odpowiedzialny za rozgrywki eliminacyjne. Oto jak wygląda kod:

```
public GamesPanel(Tournament t, Database db, onEliminationsEndListener listener)\{
    super(t,db);
    this.listener = listener;
\}

public void initComponents() \{
    removeAll();
    competitors = DB.getCompetitors(turniej.getId());
    singleGames = DB.getSingleGames(turniej.getId(), false);
    competitorMap = competitors.stream()
        .collect(Collectors.toMap(c->c.getId(), c->c));
    setDisqualifiedPlayersScores();
    recalcColors();
    JTable table = new JTable(new MyTableModel());
    table.getColumnModel().getColumn(3).setCellEditor(new DefaultCellEditor(
        new JComboBox<String>(new String[] \{Strings.notPlayedYet, Strings.whiteWon,
    }));
    table.setDefaultRenderer(String.class, new MyCellRenderer());
    add(new JScrollPane(table));
    add(Box.createRigidArea(new Dimension(0, 20)));
    finishB.addActionListener((e)->\{
        if(singleGames.stream().filter(sg->sg.getScore()==0).count()>0) \{
            Dialogs.gryBezWyniku();
        \}
        else \{
            finishB.setVisible(false);
            listener.onEliminationsEnd();
        \}
    \});
    if(turniej.getRoundsCompleted()<1) add(finishB);
    mapKeyActions(table);
\}

@FunctionalInterface
public interface onEliminationsEndListener \{
    public void onEliminationsEnd();
\}
```


Następną rzeczą było stworzenie mechanizmu wybierania graczy do finałów. Gracze, którzy pomyślnie przeszli eliminacje mają możliwość zagrania w finałach, ale to od użytkownika aplikacji zależy ilu zawodników z danej grupy przejdzie do finału. Może się zdarzyć, że z jednej grupy przejdą wszyscy a z drugiej żaden zawodnik. Na bieżąco aktualizowana jest zawartość tabel.

```
public GroupsChoosePanel(Tournament t, Database db, onFinaleStartListener listener)\{
    this.turniej = t;
    this.DB = db;
    this.setLayout(new BorderLayout());
    container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
    add(new JScrollPane(container));
    initComponents();
    startFinales.setVisible(turniej.getRoundsCompleted()==1);
    startFinales.addActionListener(e->\{
        boolean everyGroupHasSelectedPlayers=true;
        for(Set<Competitor> s : competitors.stream()
            .collect(Collectors.groupingBy(c->c.getGroup(), Collectors.toSet())).values())\{
            if(s.stream().filter(c->c.getGoesFinal()).count()==0)
                everyGroupHasSelectedPlayers=false;
        }\};
    if(!everyGroupHasSelectedPlayers) \{
        if(!Dialogs.niktZGrupyDoFinalow()) return;
    }\}
    List<Competitor> finaleCompetitors =
        competitors.stream()
            .filter(c->c.getGoesFinal())
            .collect(Collectors.toList());
    DB.insertOrUpdateSingleGame(
        Tools.generateFinaleSingleGames(finaleCompetitors, singleGames, turniej.getBoards()),
        turniej.getId());
    startFinales.setVisible(false);
    listener.onFinaleStart();
\});
\}

@FunctionalInterface
public interface onFinaleStartListener \{
    public void onFinaleStart();
\}

public void initComponents() \{
    competitors = DB.getCompetitors(turniej.getId());
    competitorMap = competitors.stream()
        .collect(Collectors.toMap(c->c.getId(), c->c));
    singleGames = DB.getSingleGames(turniej.getId(), false);
    for(Competitor c : competitors) \{
```

```
competitorGames.put(c, new LinkedList<>());
\}
for(SingleGame sg : singleGames) \{
competitorGames.get(competitorMap.get(sg.getCompetitorW())).add(sg);
competitorGames.get(competitorMap.get(sg.getCompetitorB())).add(sg);
\}
final int groups = turniej.getRounds();
container.removeAll();
tables.clear();
JLabel label = new JLabel(Strings.chooseForFinales, JLabel.CENTER);
label.setAlignmentX(JLabel.CENTER\_ALIGNMENT);

for(int i=0; i<groups; ++i) \{
container.add(Box.createRigidArea(new Dimension(0, 20)));
container.add(new JLabel(Strings.group+(i+1), JLabel.CENTER));
container.add(Box.createRigidArea(new Dimension(0, 10)));
JTable table = new JTable(new MyTableModel(i));
container.add(table.getTableHeader());
container.add(table);
tables.put(i, table);
\}
container.add(Box.createRigidArea(new Dimension(0, 50)));
container.add(startFinales);
updateTables();
\}

void updateTables() \{
for(Competitor c : competitors) \{
competitorWon.put(c, 0);
competitorLost.put(c, 0);
competitorTie.put(c, 0);
for(SingleGame sg : competitorGames.get(c)) \{
Competitor c1 = competitorMap.get(sg.getCompetitorW()); // gra białymi
Competitor c2 = competitorMap.get(sg.getCompetitorB()); // gra czarnymi
int score = sg.getScore(); // 1 - wygrały białe, 2 - czarne, 3 - remis;
if(score==1 \&\& c.equals(c1)) competitorWon.put(c, competitorWon.get(c)+1);
if(score==2 \&\& c.equals(c2)) competitorWon.put(c, competitorWon.get(c)+1);

if(score==1 \&\& c.equals(c2)) competitorLost.put(c, competitorLost.get(c)+1);
if(score==2 \&\& c.equals(c1)) competitorLost.put(c, competitorLost.get(c)+1);

if(score==3) competitorTie.put(c, competitorTie.get(c)+1);
\}
float points = 1.0f*competitorWon.get(c)+0.5f*competitorTie.get(c);
competitorPoints.put(c,points);
\}
```

```

for(Competitor c : competitors) \{
float SBPoints = 0.0f;
for(SingleGame sg : competitorGames.get(c)) \{
Competitor c1 = competitorMap.get(sg.getCompetitorW()); // gra białymi
Competitor c2 = competitorMap.get(sg.getCompetitorB()); // gra czarnymi
int score = sg.getScore(); // 1 - wygrały białe, 2 - czarne, 3 - remis;
if(score==1 \&\& c.equals(c1)) SBPoints+=competitorPoints.get(c2);
if(score==2 \&\& c.equals(c2)) SBPoints+=competitorPoints.get(c1);
if(score==3 \&\& c.equals(c1)) SBPoints+=0.5f*competitorPoints.get(c2);
if(score==3 \&\& c.equals(c2)) SBPoints+=0.5f*competitorPoints.get(c1);
\}
competitorSBPoints.put(c, SBPoints);
\}
competitors.sort((c1,c2)->(int)(4.*(competitorSBPoints.get(c2)-competitorSBPoints.get(c1))))
competitors.sort((c1,c2)->(int)(2.*(competitorPoints.get(c2)-competitorPoints.get(c1))))
tables.values().forEach((t) -> ((AbstractTableModel)t.getModel()).fireTableDataChanged())
\}

public boolean isEditAllowed() \{
return turniej.getRoundsCompleted()<0;
\}

```

Wreszcie nadszedł czas, aby wymyśleć sposób na rozgrywki finałowe. Algorytm jest podobny do tego z rozgrywek eliminacyjnych.

```

public FinaleGamesPanel(Tournament t, Database db, onFinalesEndListener listener)\{
super(t,db);
this.listener = listener;
initComponents();
\}

public void initComponents() \{
removeAll();
competitors = DB.getCompetitors(turniej.getId()).stream()
.filter(c->c.getGoesFinal()).collect(Collectors.toList());
competitorMap = competitors.stream()
.collect(Collectors.toMap(c->c.getId(), c->c));
singleGames = DB.getSingleGames(turniej.getId(), true).stream()
.filter(sg->competitorMap.containsKey(sg.getCompetitorW())\&\&
competitorMap.containsKey(sg.getCompetitorB()))
.collect(Collectors.toList());
setDisqualifiedPlayersScores();
// filtrowanie powyżej, bo baza zwraca również gry,
// gdzie grali (dostał się do finałów) vs (nie dostał się)
recalcColors();
JTable table = new JTable(new MyTableModel());

```

```

table.getColumnModel().getColumn(3).setCellEditor(new DefaultCellEditor(
    new JComboBox<String>(new String[] \{Strings.notPlayedYet, Strings.whiteWon,
    }));
table.setDefaultRenderer(String.class, new MyCellRenderer());
add(new JScrollPane(table));
add(Box.createRigidArea(new Dimension(0, 20)));
finishFinales.addActionListener((e)->\{
    if(singleGames.stream().filter(sg->sg.getScore()==0).count()>0) \{
        Dialogs.gryBezWyniku();
    \}
    else \{
        finishFinales.setVisible(false);
        listener.onFinalesEnd();
    \}
\});
if(turniej.getRoundsCompleted()<3) add(finishFinales);
mapKeyActions(table);
\}

@FunctionalInterface
public interface onFinalesEndListener \{
    public void onFinalesEnd();
\}

```

Na sam koniec zaprogramowałem sposób przydzielania punktów graczom w zależności od ilości wygranych, przegranych bądź remisów. Dzięki temu w łatwy sposób można wyłonić zwycięzców.

```

public FinaleScorePanel(Tournament t, Database db)\{
    this.turniej = t;
    this.DB = db;
    this.setLayout(new BorderLayout());
    initComponents();
\}

public void initComponents() \{
    competitors = DB.getCompetitors(turniej.getId()).stream()
        .filter(c->c.getGoesFinal()).collect(Collectors.toList());
    competitorMap = competitors.stream()
        .collect(Collectors.toMap(c->c.getId(), c->c));
    singleGames = DB.getSingleGames(turniej.getId(), true).stream()
        .filter(sg->competitorMap.containsKey(sg.getCompetitorW())\&\&
            competitorMap.containsKey(sg.getCompetitorB()))
        .collect(Collectors.toList());
    // filtrowanie powyżej, bo baza zwraca również gry,
    // gdzie grali (dostał się do finałów) vs (nie dostał się)
    // można to naprawić w bazie

```

```
for(Competitor c : competitors) \{
competitorGames.put(c, new LinkedList<>());
\}
for(SingleGame sg : singleGames) \{
competitorGames.get(competitorMap.get(sg.getCompetitorW())).add(sg);
competitorGames.get(competitorMap.get(sg.getCompetitorB())).add(sg);
\}
removeAll();
table = new JTable(new MyTableModel());
add(new JScrollPane(table));
updateTables();
\}

void updateTables() \{
for(Competitor c : competitors) \{
competitorWon.put(c, 0);
competitorLost.put(c, 0);
competitorTie.put(c, 0);
for(SingleGame sg : competitorGames.get(c)) \{
Competitor c1 = competitorMap.get(sg.getCompetitorW()); // gra białymi
Competitor c2 = competitorMap.get(sg.getCompetitorB()); // gra czarnymi
int score = sg.getScore(); // 1 - wygrały białe, 2 - czarne, 3 - remis;
if(score==1 \&\& c.equals(c1)) competitorWon.put(c, competitorWon.get(c)+1);
if(score==2 \&\& c.equals(c2)) competitorWon.put(c, competitorWon.get(c)+1);

if(score==1 \&\& c.equals(c2)) competitorLost.put(c, competitorLost.get(c)+1);
if(score==2 \&\& c.equals(c1)) competitorLost.put(c, competitorLost.get(c)+1);

if(score==3) competitorTie.put(c, competitorTie.get(c)+1);
\}
float points = 1.0f*competitorWon.get(c)+0.5f*competitorTie.get(c);
competitorPoints.put(c,points);
\}
for(Competitor c : competitors) \{
float SBPoints = 0.0f;
for(SingleGame sg : competitorGames.get(c)) \{
Competitor c1 = competitorMap.get(sg.getCompetitorW()); // gra białymi
Competitor c2 = competitorMap.get(sg.getCompetitorB()); // gra czarnymi
int score = sg.getScore(); // 1 - wygrały białe, 2 - czarne, 3 - remis;
if(score==1 \&\& c.equals(c1)) SBPoints+=competitorPoints.get(c2);
if(score==2 \&\& c.equals(c2)) SBPoints+=competitorPoints.get(c1);
if(score==3 \&\& c.equals(c1)) SBPoints+=0.5f*competitorPoints.get(c2);
if(score==3 \&\& c.equals(c2)) SBPoints+=0.5f*competitorPoints.get(c1);
\}
competitorSBPoints.put(c, SBPoints);
\}
```

```
competitors.sort((c1,c2)->(int)(4.*(competitorSBPoints.get(c2)-competitorSBPoints.get(c1))
competitors.sort((c1,c2)->(int)(2.*(competitorPoints.get(c2)-competitorPoints.get(c1))
((AbstractTableModel)table.getModel()).fireTableDataChanged();
\}

public boolean isEditAllowed() \{
return turniej.getRoundsCompleted()<0;
\}
```

Mirosława Pelc

Mirosława Pelc Swing to biblioteka graficzna używana w języku programowania Java. To właśnie z tej biblioteki skorzystałam tworząc interfejs graficzny. Tworzenie GUI rozpoczęłam od „bazy” czyli okna głównego, zwanego JFrame w środowisku programistycznym. W tym celu należało rozszerzyć klasę odpowiadającą za okno główne dodając słowo extends a zaraz po nim JFrame.

```
public class MainWindow extends JFrame \{
private static final long serialVersionUID = -4321522332774571523L;

public static void main (String[] args)\{
new MainWindow().setVisible(true);
\}

public MainWindow() \{
setMinimumSize(new Dimension(700, 500));
setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);
setLayout(new BorderLayout());

setJMenuBar(Tools.aboutMenu(new JMenuBar(), MainWindow.this));

try \{
InputStream imgIS = getClass().getResourceAsStream("/szachy.png");
add(new JLabel(new ImageIcon(ImageIO.read(imgIS))), BorderLayout.CENTER);
\} catch (IOException e1) \{
e1.printStackTrace();
\}
\}
\}
```

W powyższym kodzie ustawiłam rozmiar okna komendą setMinimumSize, dodałam obrazek startowy używając InputStream oraz ImageIcon. Stworzyłam pasek menu używając takich komponentów jak JMenu, JMenuItem, MenuBar. Tworzenie paska menu:

```
JMenu mnTurniej = new JMenu("Turniej"),
mn0Programie = new JMenu("0 programie");
```

```
menuBar.add(mnTurniej);
menuBar.add(mnOProgramie);

JMenuItem mntmPomoc = new JMenuItem("Pomoc"),
dodajTurniej = new JMenuItem("Dodaj turniej"),
wybierzTurniej = new JMenuItem("Wybierz turniej"),
mntmAutorzy = new JMenuItem("Autorzy"),
mntmOpis = new JMenuItem("Opis");

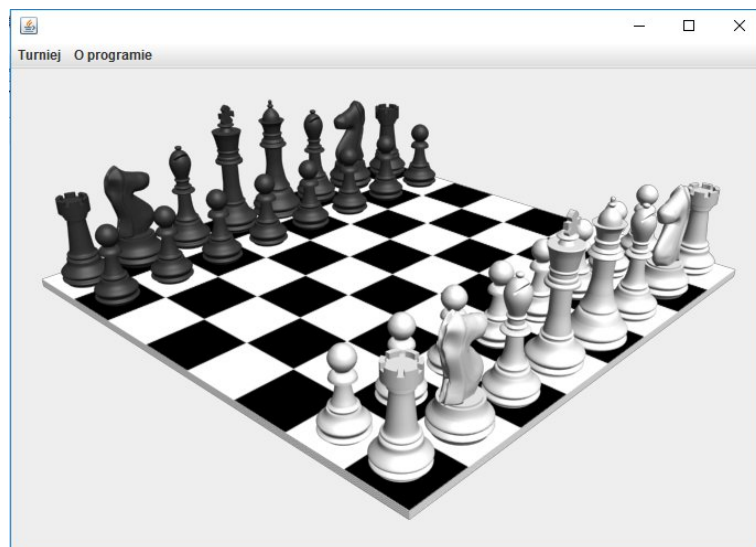
mntmPomoc.setAlignmentY(Component.TOP_ALIGNMENT);

mnTurniej.add(dodajTurniej);
mnTurniej.add(wybierzTurniej);
mnOProgramie.add(mntmPomoc);
mnOProgramie.add(mntmAutorzy);
mnOProgramie.add(mntmOpis);

dodajTurniej.setAccelerator(KeyStroke.getKeyStroke(
    java.awt.event.KeyEvent.VK_F2, 0));
wybierzTurniej.setAccelerator(KeyStroke.getKeyStroke(
    java.awt.event.KeyEvent.VK_F3, 0));

dodajTurniej.addActionListener(e -> \{
    frame.getContentPane().removeAll();
    frame.add(new AddTPanel(frame), BorderLayout.CENTER);
    frame.pack();
\});
wybierzTurniej.addActionListener(e -> \{
    frame.getContentPane().removeAll();
    frame.add(new ShowTPanel(frame), BorderLayout.CENTER);
    frame.pack();
\});
// otwieranie pdf z instrukcją po wybraniu pomocy
mntmPomoc.addActionListener(e->\{
    if(Desktop.isDesktopSupported()) \{
        try \{
            File myFile = new File("turniej.pdf");
            Desktop.getDesktop().open(myFile);
        \} catch (IOException ex) \{
            System.out.println(e);
        \}
    \}
\});
```

Dodane zostały skróty klawiszowe dla poszczególnych JMenuItem. Okno po wykonaniu czynności wygląda następująco:



Rys. 4.1. Skróty klawiszowe dla poszczególnych JMenuItem

Aby utworzyć nowy turniej potrzebnym było stworzenie nowego panelu, który łączy się z bazą danych. Oto jego kod:

```
package window;

import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Calendar;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

import model.Database;
import model.Tournament;
import panel.CompetitorTabbedPane;

/**
 * Panel "nowy turniej"
 */
public class AddTPanel extends JPanel {
    private static final long serialVersionUID = -4930339429679727134L;
    private JTextField textField;
    private String nazwa;
```

```
public AddTPanel(final JFrame jframe)\{
setLayout(null);

JLabel nameTour = new JLabel("Nazwa turnieju");
nameTour.setFont(new Font("Consolas", Font.PLAIN, 16));
nameTour.setHorizontalTextPosition(SwingConstants.CENTER);
nameTour.setHorizontalAlignment(SwingConstants.CENTER);
nameTour.setBounds(100, 100, 484, 30);
add(nameTour);

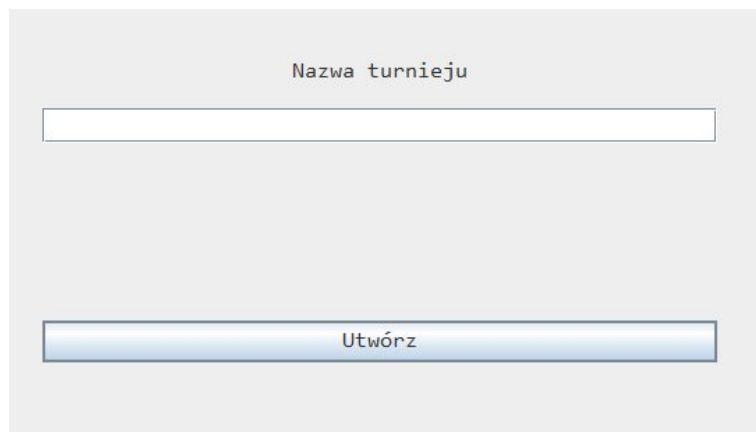
textField = new JTextField();
textField.setBounds(100, 141, 484, 25);
textField.setColumns(10);
add(textField);

JButton addButton = new JButton("Utwórz");
addButton.setFont(new Font("Consolas", Font.PLAIN, 16));
addButton.setBounds(100, 293, 484, 30);
addButton.addActionListener(new ActionListener() \{
public void actionPerformed(ActionEvent e) \{
nazwa=textField.getText();
String year = String.valueOf(Calendar.getInstance().get(Calendar.YEAR));
Tournament t = new Tournament(null,nazwa,year,8,5,-1);
Database db = new Database();
db.insertOrUpdateTournament(t);
jframe.getContentPane().removeAll();
new CompetitorTabbedPane(t, jframe);
db.close();
\}
\});

// po naciśnięciu enter aktywuje się guzik DODAJ
jframe.getRootPane().setDefaultButton(addButton);

add(addButton);
\}
\}
```

Panel zawiera takie komponenty jak JPanel, etykietę JLabel, pole tekstowe JTextField oraz guzik JButton. Stworzony panel wygląda następująco:



Rys. 4.2. Komponenty panelu

Kolejnym krokiem było utworzenie panelu, który odpowiadałby za wybór wcześniej utworzonego turnieju. Kod tego panelu przedstawia się następująco:

```
package window;

import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingConstants;

import model.Database;
import model.Tournament;
import panel.CompetitorTabbedPane;

/**
 * Panel "dodaj turniej"
 */
public class ShowTPanel extends JPanel \{
    private static final long serialVersionUID = -1094699102373510646L;
    private Database db;

    public ShowTPanel(final JFrame jframe) \{
        db = new Database();
        setLayout(null);

        final JComboBox<String> comboBox = new JComboBox<String>();
        comboBox.setFont(new Font("Consolas", Font.PLAIN, 15));
```

```

comboBox.setBounds(10, 100, 664, 30);

for(Tournament t: db.getTournaments())\{
String nazwa = t.getName();
comboBox.addItem(nazwa);
\}

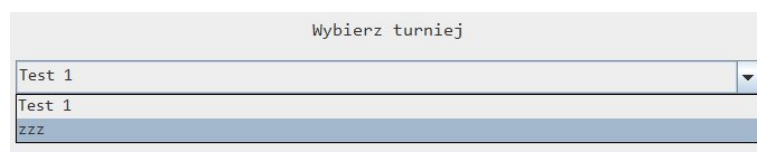
comboBox.addActionListener (new ActionListener () \{
public void actionPerformed(ActionEvent e) \{
int sIndex = comboBox.getSelectedIndex();
jframe.getContentPane().removeAll();
new CompetitorTabbedPane(db.getTournaments().get(sIndex),jframe);
db.close();
\}
\});

add(comboBox);

JLabel wybierz = new JLabel("Wybierz turniej");
wybierz.setFont(new Font("Consolas", Font.PLAIN, 16));
wybierz.setHorizontalTextPosition(SwingConstants.CENTER);
wybierz.setHorizontalAlignment(SwingConstants.CENTER);
wybierz.setBounds(10, 59, 664, 30);
add(wybierz);
\}
\}

```

Użyte komponenty to w tym przypadku etykieta JLabel oraz JComboBox czyli wysuwana lista, która zawiera nazwy zapisanych turniejów. Oto graficzna reprezentacja panelu wyboru turnieju:



Rys. 4.3. Graficzna reprezentacja panelu wyboru turnieju

Po zatwierdzeniu utworzonego turnieju lub po wybraniu już istniejącego pojawia się okno odpowiadające za rozgrywki. Początkowo są trzy zakładki, których kod jest następujący:

```

public CompetitorTabbedPane(Tournament turniej, JFrame frame)\{
this.turniej = turniej;
this.DB = new Database();
showPanel = new ShowEditCompetitorPanel(turniej, DB);
tournamentPanel = new TournamentPanel(turniej, DB);

```

```
[...]
groupsPanel = new GroupsPanel(turniej, DB, tStartlistener);
setMenu(frame);
tabbedPane.add(Strings.showOrEditComp, showPanel);
tabbedPane.add(Strings.tournament, tournamentPanel);
tabbedPane.add(Strings.prepGroups, groupsPanel);
tabbedPane.addChangeListener((e) -> \{
int i = tabbedPane.getSelectedIndex();
if(i==0) showPanel.setData();
if(i==1) tournamentPanel.setSBBounds();
if(i==2) groupsPanel.initComponents();
if(i==3) gamesPanel.initComponents();
if(i==5) finaleGamesPanel.initComponents();
comp.setVisible( turniej.isPlayersEditAllowed() \&\& i==0);
group.setVisible(turniej.isPlayersEditAllowed() \&\& i==2 );
sort.setVisible(i==2);
\});

frame.add(tabbedPane);
setVisible(true);
```

Użyty został komponent JTabbedPane, który odpowiada za panel z zakładkami.

Jak można łatwo zauważyć, program ma wiele tabel. JTable to kolejny komponent biblioteki Swing. Przykładowe tabele:

```
private JTable table;
[...]
table = new EditCompetitorJTable();
[...]
public class EditCompetitorJTable extends JTable \{
private static final long serialVersionUID = -9074329149984999956L;

public EditCompetitorJTable() \{
super();
setIntercellSpacing(new Dimension(25, 2));
setRowHeight(20);
setModel(new EditCompetitorTableModel());

setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
// pole tekstowe akceptujące tylko znaki a-Z, - i spację
final JTextField jtf = new JTextField(new MyPlainDocument(), null, 0);
// przy rozpoczęciu edycji zaznaczenie wszystkiego
jtf.addFocusListener(new FocusAdapter() \{
@Override
public void focusGained(FocusEvent e) \{
jtf.selectAll();
\}
```

```

\});

// dla pól imię i nazwisko ustawiony edytor na podstawie powyższego pola tekstowego
columnModel.getColumn(1).setCellEditor(new DefaultCellEditor(jtf));
columnModel.getColumn(2).setCellEditor(new DefaultCellEditor(jtf));
columnModel.getColumn(4).setCellEditor(new DefaultCellEditor(
new JComboBox<Integer>(new Integer[]\{1,2,3,4,5,6\})
));
\}

// po przejściu do komórki (również tabulatorem) rozpoczęcie edycji
public void changeSelection(int row, int column, boolean toggle, boolean extend) \{
super.changeSelection(row, column, toggle, extend);
if(editCellAt(row, column)) \{
getEditorComponent().requestFocusInWindow();
\}
\}
\}

JTable table = new JTable(new MyTableModel());
table.getColumnModel().getColumn(3).setCellEditor(new DefaultCellEditor(
new JComboBox<String>(new String[] \{Strings.notPlayedYet, Strings.whiteWon, Strings.b
));
table.setDefaultRenderer(String.class, new MyCellRenderer());
add(new JScrollPane(table));

```

A oto kod odpowiadający za formatowanie JTable

```

public class EditCompetitorJTable extends JTable \{
private static final long serialVersionUID = -9074329149984999956L;

public EditCompetitorJTable() \{
super();
setIntercellSpacing(new Dimension(25, 2));
setRowHeight(20);
setModel(new EditCompetitorTableModel());

setSelectionMode(javax.swing.ListSelectionModel.SINGLE\_SELECTION);
// pole tekstowe akceptujące tylko znaki a-Z, - i spację
final JTextField jtf = new JTextField(new MyPlainDocument(), null, 0);
// przy rozpoczęciu edycji zaznaczenie wszystkiego
jtf.addFocusListener(new FocusAdapter() \{
@Override
public void focusGained(FocusEvent e) \{
jtf.selectAll();
\}
\});

```

```
// dla pól imię i nazwisko ustawiony edytor na podstawie powyższego pola tekstowego
columnModel.getColumnModel(1).setCellEditor(new DefaultCellEditor(jtf));
columnModel.getColumnModel(2).setCellEditor(new DefaultCellEditor(jtf));
columnModel.getColumnModel(4).setCellEditor(new DefaultCellEditor(
new JComboBox<Integer>(new Integer[]\{1,2,3,4,5,6\})
));
\}

// po przejściu do komórki (również tabulatorem) rozpoczęcie edycji
public void changeSelection(int row, int column, boolean toggle, boolean extend) \{
super.changeSelection(row, column, toggle, extend);
if(editCellAt(row, column)) \{
getEditorComponent().requestFocusInWindow();
\}
\}
\}
```

Aby nie powielać kodu przy formatowaniu modelu JTable stworzyłam klasę abstrakcyjną modelu rozszerzającą AbstractTableModel, którą przypisuję do tabeli.

```
protected abstract class MyTableModel extends AbstractTableModel \{
private static final long serialVersionUID = -8079013606990307646L;
final String[] columnNames = \{Strings.board, Strings.playsWithWhite, Strings.playsWit
@Override
public Class<?> getColumnClass(int columnIndex) \{
return String.class;
\}
@Override
public int getColumnCount() \{
return 4;
\}
@Override
public String getColumnName(int columnIndex) \{
return columnNames[columnIndex];
\}
@Override
public int getRowCount() \{
return singleGames.size();
\}
@Override
public Object getValueAt(int row, int col) \{
SingleGame sg = singleGames.get(row);
if(col==0 \&\& sg.getBoard()==-1) return " - ";
if(col==0) return sg.getBoard()+1;
if(col==1) return competitorMap.get(sg.getCompetitorW());
if(col==2) return competitorMap.get(sg.getCompetitorB());
```

```

if(col==3) \{
if(sg.getScore()==0) return Strings.notPlayedYet;
if(sg.getScore()==1) return Strings.whiteWon;
if(sg.getScore()==2) return Strings.blackWon;
if(sg.getScore()==3) return Strings.tie;
\}
return null;
\}
public abstract void setValueAt(Object aValue, int row, int col);
\}
\}

```

W celu ułatwienia użytkownikowi programu przydzielanie zawodników do szachownic, komórki JTable zostały pokolorowane. Oto jak wygląda to w kodzie programu:

```

final void recalcColors() \{
currentlyPlayedGames = new ArrayList<>(turniej.getBoards());
currentlyPlayedGames.clear();;
List<Integer> playingCompetitors = new ArrayList<>(2*turniej.getBoards()),
freeBoards = new LinkedList<>();
for(int i=0; i<turniej.getBoards(); ++i) freeBoards.add(i);
for(SingleGame sg : singleGames.stream().filter(g->g.getScore()==0\&\&g.getBoard()>=0)
.collect(Collectors.toList()))\{
freeBoards.remove(sg.getBoard());
playingCompetitors.add(sg.getCompetitorW());
playingCompetitors.add(sg.getCompetitorB());
currentlyPlayedGames.add(sg);
\};
for(SingleGame sg : singleGames.stream().filter(g->g.getScore()==0\&\&g.getBoard()<0)
.collect(Collectors.toList()))\{
if(freeBoards.isEmpty()) break;
if(!playingCompetitors.contains(sg.getCompetitorW()) \&\&
!playingCompetitors.contains(sg.getCompetitorB()))
\{
int board = freeBoards.get(0);
sg.setBoard(board);
freeBoards.remove(0);
playingCompetitors.add(sg.getCompetitorW());
playingCompetitors.add(sg.getCompetitorB());
currentlyPlayedGames.add(sg);
\}
\};
sortGames();
\}

package res;

```



```
import java.awt.Color;

public final class Colors \{
public final static Color
selInProgress = Color.decode("\#5BFFB5"),
selNormal = Color.decode("\#84D4FF"),
InProgress = Color.decode("\#5BFF8C"),
normal = Color.decode("\#FFFFFF");
\}
```

Druga zakładka w oknie turnieju to edycja turnieju, czyli: zmiana tytułu, ustawienie roku rozgrywania turnieju, suwaki odpowiedzialne za ilość szachownic, czas rozgrywki, ilość grup. Użyte komponenty to etykieta JLabel, pole tekstowe JTextField, suwak ScrollBar. Layout panelu ustawiony jest na GridLayout(0,2,...) co oznacza, że panel podzielony jest na dwie kolumny a komponenty dodawane są raz do jednej kolumny, raz do drugiej. Oto kod TournamentPanel:

```
package panel;

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.GridLayout;
import java.awt.Scrollbar;

import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.border.EmptyBorder;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;

import model.Database;
import model.Tournament;
import res.Strings;
import tools.MyPlainDocument;
import tools.Simulator;

/**
 * Zakładka danych o turnieju
 */
public class TournamentPanel extends JPanel\{
private static final long serialVersionUID = -3657045958131642437L;
private final Tournament turniej;
private final Database DB;
private final JLabel nameL, yearL, sgTimeL, groupsL, boardsL, stats1L, stats2L;
final JTextField nameTF, yearTF;
final Scrollbar groupsSB, sgTimeSB, boardsSB;
```

```

final JPanel panel = new JPanel();
/**
 * @param t - id turnieju
 * @param db - baza danych
 */
public TournamentPanel(Tournament t, Database db){
    this.turniej = t;
    this.DB = db;
    nameL = new JLabel("Nazwa turnieju: ");
    yearL = new JLabel("Rok: ");
    sgTimeL = new JLabel(Strings.sgTimeT+"10 min");
    groupsL = new JLabel(Strings.groupsT+"2");
    boardsL = new JLabel(Strings.boardsT);
    stats1L = new JLabel(Strings.gamesT);
    stats2L = new JLabel(Strings.timeRRET);
    nameTF = new JTextField();
    yearTF = new JTextField();
    groupsSB = new Scrollbar(Scrollbar.HORIZONTAL, 2, 1, 2, 9+1);
    sgTimeSB = new Scrollbar(Scrollbar.HORIZONTAL, 20, 4, 2, 40+4);
    boardsSB = new Scrollbar(Scrollbar.HORIZONTAL, 8, 1, 2, 20+1);

    setLayout(new BorderLayout());
    setBorder(new EmptyBorder(10, 10, 10, 10));
    panel.setLayout(new GridLayout(0, 2, 10, 20));
    add(panel, BorderLayout.NORTH);
    nameTF.setDocument(new MyPlainDocument());
    yearTF.setDocument(new MyPlainDocument());

    setComponentsActions();

    Component[] cs = \{
    nameL, nameTF, yearL, yearTF, sgTimeL, sgTimeSB, groupsL, groupsSB, boardsL, boardsSB,
    \};
    for(Component c : cs) panel.add(c);

    nameTF.setText(turniej.getName());
    yearTF.setText(turniej.getYear());
    setSBBounds();
    recalcStats();
    \}

    private void setComponentsActions() \{
    nameTF.getDocument().addDocumentListener(new MyDocumentListener() \{
    @Override
    public void action() \{
    turniej.setName(nameTF.getText());

```

```

DB.insertOrUpdateTournament(turniej);
\}
\});
yearTF.getDocument().addDocumentListener(new MyDocumentListener() \{
@Override
public void action() \{
turniej.setYear(yearTF.getText());
DB.insertOrUpdateTournament(turniej);
\}
\});
groupsSB.addAdjustmentListener(e -> \{
if(!turniej.isPlayersEditAllowed()) \{
groupsSB.setValue(turniej.getRounds());
return;
\}
int g = groupsSB.getValue();
turniej.setRounds(g);
DB.insertOrUpdateTournament(turniej);
recalcStats();
\});
boardsSB.addAdjustmentListener(e -> \{
if(!turniej.isPlayersEditAllowed()) \{
boardsSB.setValue(turniej.getBoards());
return;
\}
int g = boardsSB.getValue();
turniej.setBoards(g);
DB.insertOrUpdateTournament(turniej);
recalcStats();
\});
sgTimeSB.addAdjustmentListener(e -> \{
recalcStats();
\});
\}

private abstract class MyDocumentListener implements DocumentListener \{
public abstract void action();
@Override public final void removeUpdate(DocumentEvent e) \{ action(); \}
@Override public final void insertUpdate(DocumentEvent e) \{ action(); \}
@Override public final void changedUpdate(DocumentEvent e)\{ action(); \}
\}

public void recalcStats() \{ // TODO - poprawić przewidywany czas turnieju
groupsL.setText(Strings.groupsT+groupsSB.getValue());
boardsL.setText(Strings.boardsT+boardsSB.getValue());
sgTimeL.setText(Strings.sgTimeT+(sgTimeSB.getValue()/2f)+" min");

```

```

int graczy = DB.getCompetitors(turniej.getId()).size();
if(turniej.getBoards()<1 || graczy<2) return;
float czasSG = sgTimeSB.getValue()/2f;
int grup = groupsSB.getValue();
int rozgrywek = Simulator.rozgrywek\_eliminacje(graczy, grup);
stats1L.setText(Strings.gamesT+String.valueOf(rozgrywek));
int gier\_naraz = (int)Math.min(Math.floor(graczy/2), turniej.getBoards());
stats2L.setText(Strings.timeRRET+Math.ceil(rozgrywek/gier\_naraz)*czasSG+" min");
\}

public void setSBBounds() \{
int graczy = DB.getCompetitors(turniej.getId()).size();
groupsSB.setMinimum(2);
groupsSB.setMaximum((int)Math.ceil(graczy/2)+2);
groupsSB.setValue(turniej.getRounds());
recalcStats();
\}
\}

```

Poniższy rysunek przedstawia kod po skompilowaniu:

Rys. 4.4. Edycja turnieju

Program nie dopuszcza do zaistnienia pewnych zdarzeń, jeśli użytkownik wprowadzi błędne dane lub dobierze zawodników źle w grupy wówczas wyskoczy błąd. Za wiadomości o błędach odpowiada JOptionPane. Na potrzeby programu stworzyłam kilka wiadomości o błędach. Kod:

```

package tools;

import javax.swing.JOptionPane;

import model.Competitor;

/**
 * Definiuje okna błędów, ostrzeżeń oraz informacji
 */
public class Dialogs \{
public static void bladBazy() \{
JOptionPane.showMessageDialog(

```

```
null,
"Błąd odczytu / zapisu",
"Błąd bazy",
JOptionPane.ERROR_MESSAGE);
\}

public static void graczBezGrupy() \{
JOptionPane.showMessageDialog(
null,
"Aby móc rozpocząć turniej, każdy gracz musi być przydzielony do grupy",
"Gracz bez grupy",
JOptionPane.ERROR_MESSAGE);
\}

public static void nierownomiernyPodzial(int min, int max) \{
JOptionPane.showMessageDialog(
null,
"Największa grupa: "+max+" uczestników, \textbackslash{\}n"+
"Najmniejsza grupa: "+min+" uczestników \textbackslash{\}n"+
"Różnica pomiędzy tymi wartościami nie może być większa od 1",
"Nierównomierny podział",
JOptionPane.ERROR_MESSAGE);
\}

public static void gryBezWyniku() \{
JOptionPane.showMessageDialog(
null,
"Aby zakończyć eliminacje, wszystkie gry tej fazy muszą być ukończone",
"Nieukończone rozgrywki",
JOptionPane.ERROR_MESSAGE);
\}

public static void autorzy() \{
JOptionPane.showMessageDialog(
null,
"Autorzy: \textbackslash{\}n"+
"Piotr Jabłoński\textbackslash{\}n"+
"Mirosława Pelc\textbackslash{\}n"+
"Mariusz Lorek",
"Autorzy", JOptionPane.UNDEFINED_CONDITION);
\}

public static void opis() \{
JOptionPane.showMessageDialog(
null,
"Program powstał w ramach zaliczenia Zespołowych Przedsięwzięć Inżynierskich"+
```

```

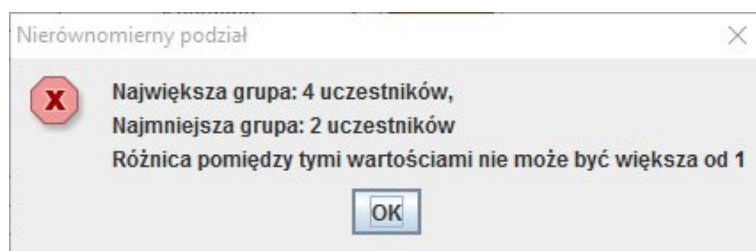
"na Państwowej Wyższej Szkole Zawodowej w Nowym Sączu.\textbackslash\{\}\nProwadzący pr
"dr Antoni Ligęza\textbackslash\{\}\nProgram obsługuje turniej szachowy odbywający się
"Małopolskiej Nocy Naukowców.",
"Opis", JOptionPane.UNDEFINED\_CONDITION);
\}

/**
 * @return Czy kontynuować mimo ostrzeżenia
 */
public static boolean niktZGrupyDoFinalow() \{
int r = JOptionPane.showConfirmDialog(
null,
"Istnieje grupa, w której nie wybrano graczy przechodzących do finału. Kontynuować?",
"Uwaga!",
JOptionPane.OK\_CANCEL\_OPTION);
return r==JOptionPane.OK\_OPTION;
\}

/**
 * @return Czy na pewno zdyskwalifikować
 */
public static boolean czyZdyskwalifikowac(Competitor c) \{
int r = JOptionPane.showConfirmDialog(
null,
"Czy jesteś pewien, że chcesz zdyskwalifikować zawodnika "+c+"\textbackslash\{\}\nTej
"Uwaga!",
JOptionPane.OK\_CANCEL\_OPTION);
return r==JOptionPane.OK\_OPTION;
\}
\}

```

Tak wygląda przykładowe okno informujące o błędzie:



Rys. 4.5. Okno informujące o błędzie

Podczas generowania losowych graczy imiona i nazwiska pobierane są z plików imiona.txt oraz nazwiska.txt, które zawarte są w pliku jar. Poniżej pokazałam jak to się odbywa w programie:

```
public static Competitor RandomPlayer() throws IOException \{
```

```
Random rn = new Random();
int a = 10+rn.nextInt(10)+rn.nextInt(10);
int c = rn.nextInt(6)+1;

int randomInt = rn.nextInt(300);
String imie = null, nazwisko = null;

InputStream imionaIS = JFrame.class.getResourceAsStream("/imiona.txt");
InputStream nazwiskaIS = JFrame.class.getResourceAsStream("/nazwiska.txt");

imieReader = new BufferedReader(new InputStreamReader(imionaIS, "UTF-8"));
nazwiskoReader = new BufferedReader(new InputStreamReader(nazwiskaIS, "UTF-8"));

imieReader.mark(0);
nazwiskoReader.mark(0);

do \{
    imieReader.reset();
    nazwiskoReader.reset();
    for (int i = 0; i < randomInt; i++) \{
        imie = imieReader.readLine();
        nazwisko = nazwiskoReader.readLine();
    \}
    \} while(imie==null || nazwisko==null || (imie.endsWith("a") \&\& nazwisko.endsWith("k")
return new Competitor(null, imie, nazwisko, a, c, false, null);
\}
```