1. Temat

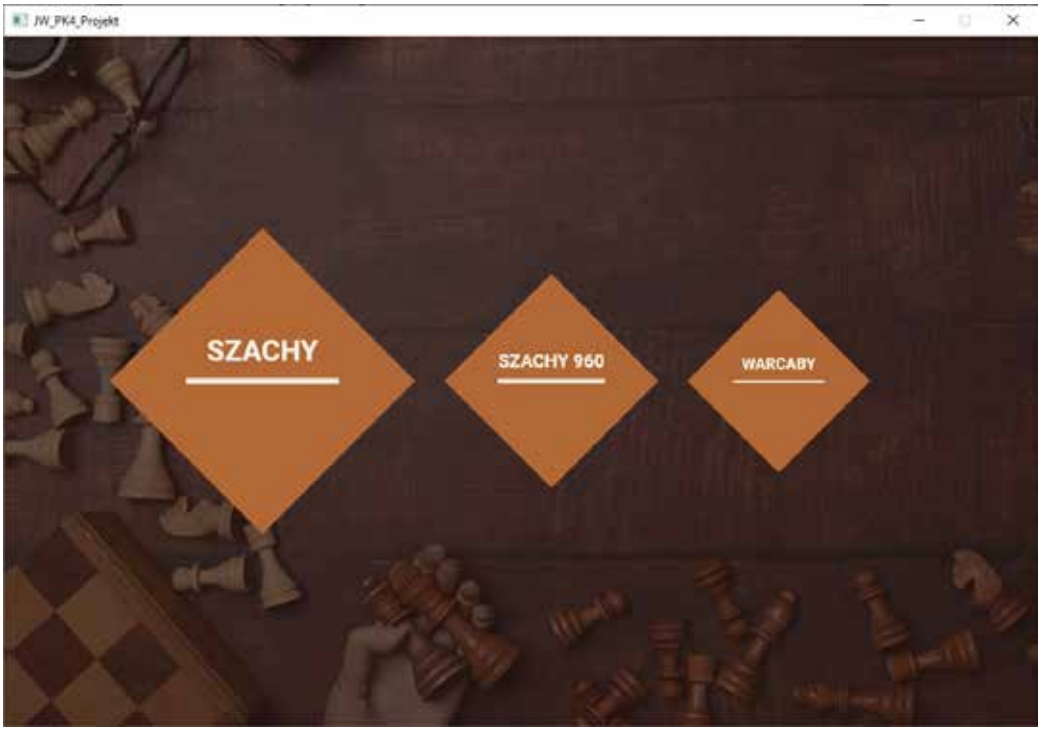
Program do gry w szachy, szachy Fischera oraz warcaby.

2. Analiza tematu

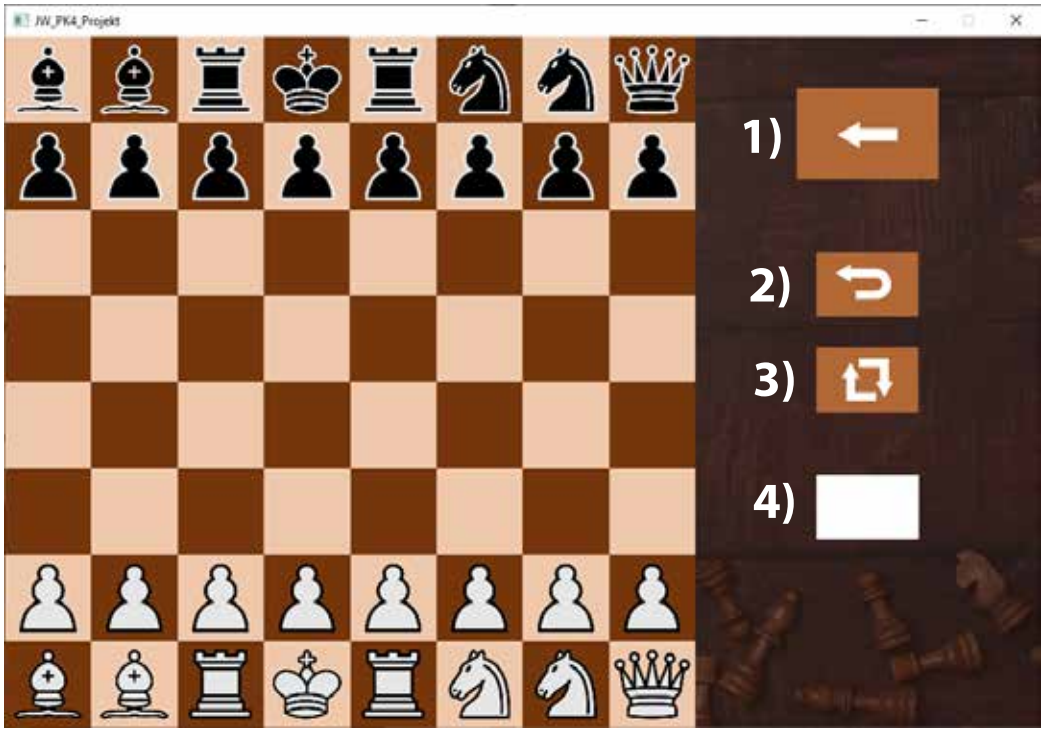
Zadanie polega na napisaniu programu okienkowego do rozgrywki w szachy, szachy Fischera (wariant szachów, w których figury na pierwszej i ostatniej linii są ułożone losowo) oraz warcaby. Rozgrywka odbywa się na jednym komputerze, jednak w przypadku klasycznych szachów możliwa jest również rozgrywka sieciowa. Do stworzenia projektu wykorzystałem bibliotekę SFML, która posiada szereg przydatnych klas i funkcji do obsługi okna, obiektów graficznych i gry sieciowej.

3. Specyfikacja zewnętrzna

Po uruchomieniu programu na ekranie wyświetla się okienko z menu głównym.



Użytkownik ma do wyboru 3 rozgrywki - w przypadku wyboru szachów 960 (inna nazwa dla szachów Fischera) i warcabów rozgrywka zaczyna się natychmiastowo.



1) Przycisk powrotu - po jego naciśnięciu użytkownik cofa się do menu głównego, a aktualna rozgrywka zostaje utracona.

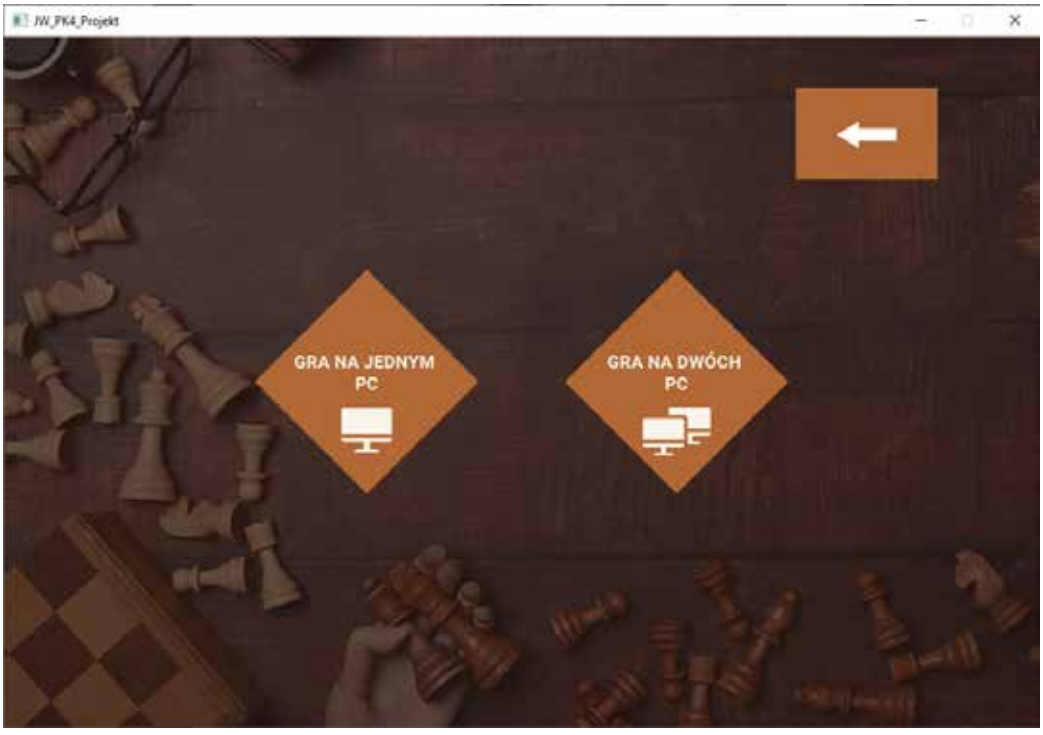
2) Przycisk cofnięcia ruchu - po jego naciśnięciu zostaje wykonany powrót do poprzedniej pozycji.

3) Przycisk obrotu planszy - po jego naciśnięciu plansza zostaje obrócona, razem z figurami.

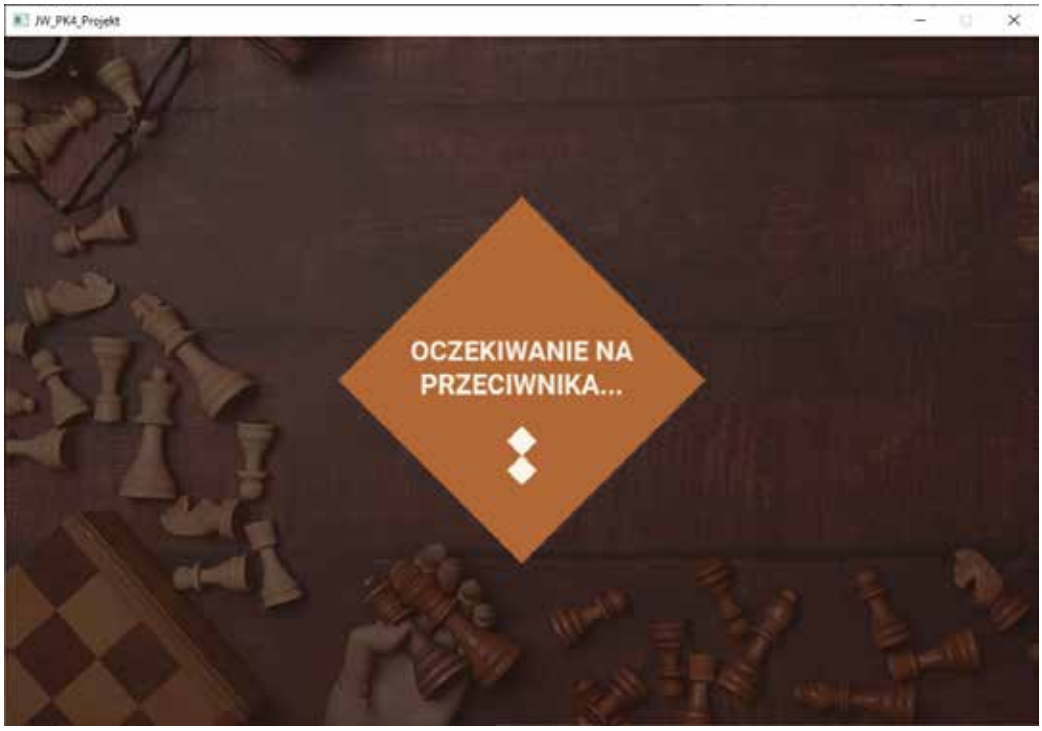
4) Aktualny ruch - prostokąt koloru białego wskazuje na ruch białego, a koloru czarnego - na ruch czarnego.

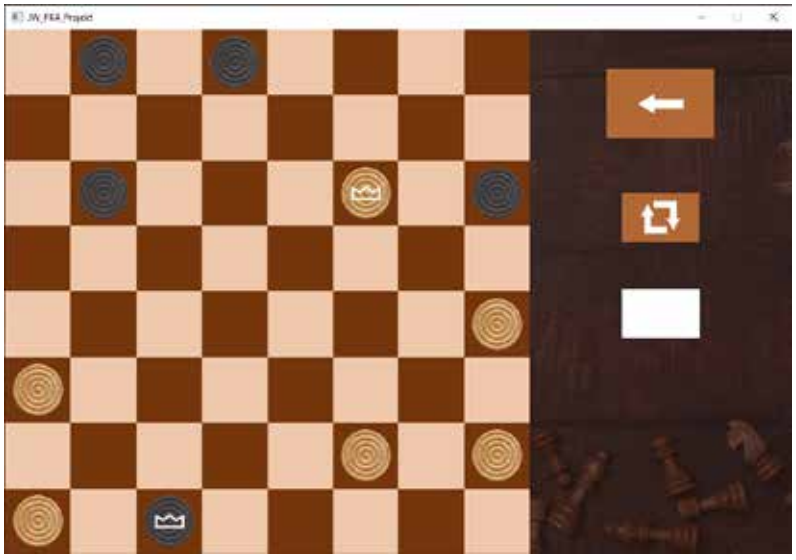
Ruchy są wykonywane poprzez przeciąganie figur myszką. By wykonać roszadę należy wykonać ruch królem na wieżę.
„Złapać” można każdą figurę, jednak ruch zostanie wykonany tylko wtedy, jeśli jest on poprawny.
Po zakończeniu zakończeniu rozgrywki poprzez mata lub pata zostaje wyświetlony odpowiedni komunikat. Po cofnięciu ruchu z pozycji rezultatywnej rozgrywka może być kontynowana, jednak komunikat wyniku pozostaje.

W przypadku wyboru szachów klasycznych użytkownik ma do wyboru grę na jednym komputerze lub grę sieciową.



Po wybraniu gry na dwóch PC zostaje wyświetlony ekran oczekiwania na przeciwnika. Po wybraniu tej opcji na dwóch komputerach w sieci lokalnej rozgrywka się rozpoczyna. Użytkownik, który wybrał tę opcję jako pierwszy gra białymi. W rozgrywce sieciowej nie jest możliwe cofanie ruchu ani wykonywanie ruchów bierkami przeciwnika.





Zrzut ekranu z rozgrywki w warcaby

4. Specyfikacja wewnętrzna

4.1 Klasy

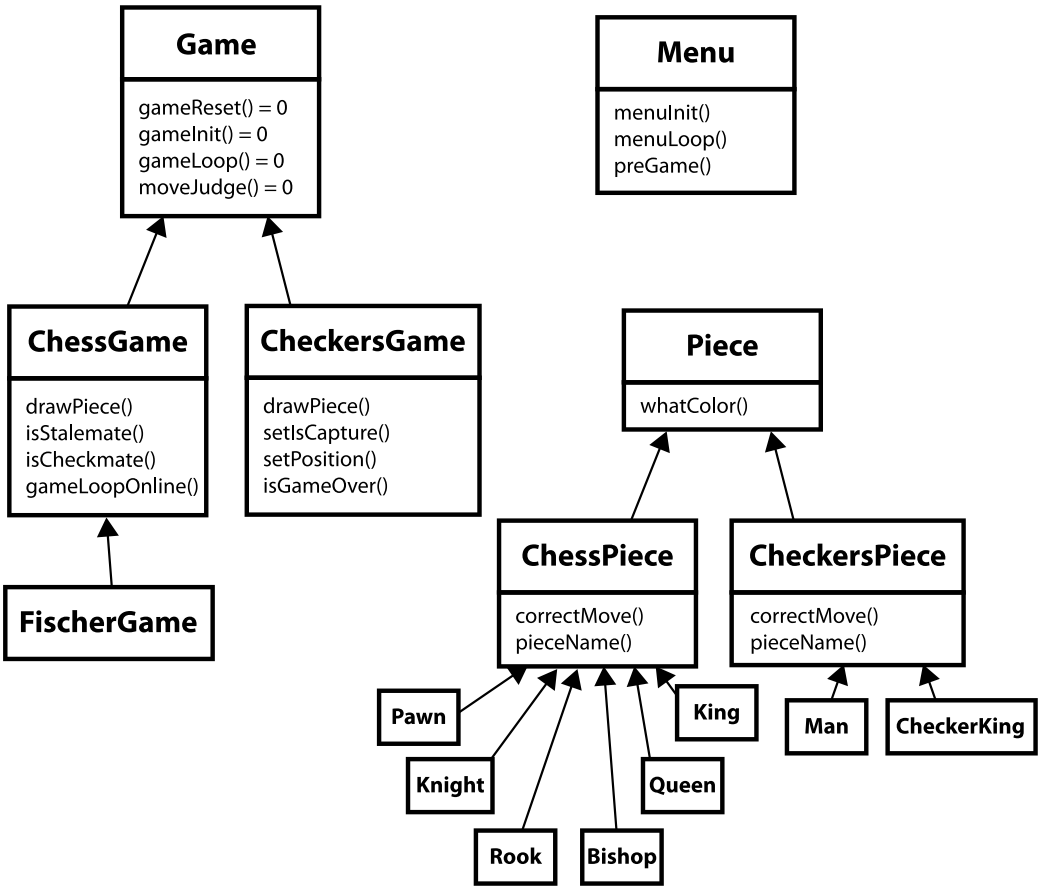


Diagram klas z najważniejszymi funkcjami

Opis klasy użytych w programie:

• **`ChessPiece, CheckersPiece`** - klasy abstrakcyjne stanowiące interfejs dla figur w poszczególnych grach.
`- correctMove(int, int, int, int, Piece* arr[8][8])` - metoda typu bool sprawdzająca „teoretyczną poprawność ruchu”, ruch jest teoretycznie poprawny, jeśli bierka danego typu jest w stanie fizycznie poruszyć się na dane pole - ignorując pozostałe reguły gry.
`- pieceName()` - zwraca nazwę danej bierki w postaci znaku (np. ‘Q’ oznacza hetmana w szachach).

• **Game** - klasa abstrakcyjna z zadeklarowanymi atrybutami typu **sf::Sprite** i **sf::Texture** (obiekty do wyświetlania planszy, figur oraz przycisków) oraz metodami wirtualnymi, które w klasach pochodnych pełnią kluczowe role:
- **gameReset()** - inicjuje początkową pozycję w tablicy wskaźników na obiekty typu **Piece**;
- **gameInit()** - inicjalizuje atrybuty typu **sf::Sprite** i **sf::Texture** (pobiera pliki png oraz ustawia rozmiar obiektów);
- **gameLoop(sf::RenderWindow&, sf::RectangleShape&, sf::Sprite)** - główna
pętla gry. Zajmuje się obsługą wydarzeń (takich jak np. najechanie kursorem na przycisk lub zamknięcie okna), wywołuje pozostałe metody w klasach dziedziczących po klasie **Game**.
- **moveJudge(int, int, int, int)** - metoda typu bool oceniająca poprawność ruchu wykonanego przez użytkownika. Wykorzystując polimorfizm wywołuje metodę **correctMove()** i zwraca true jeśli ruch jest poprawny pod pozostałymi kątami (np. czy ruch nie wywołuje szacha na królu w przypadku gry w szachy).

• **Menu** - klasa obsługująca menu.
- **menuInit()** - inicjalizuje główne okno programu (które jest następnie przekazywane do odpowiednich funkcji jako parametr), obiekty przycisków widocznych przed rozgrywką oraz obraz tła.
- **menuLoop()** - główna pętla menu - zajmuje się obsługą wydarzeń, inicjuje odpowiednie klasy oraz wywołuje odpowiednie funkcje w zależności od wyborów dokonanych przez użytkownika.
- **preGame()** - menu widoczne po wybraniu gry w szachy klasyczne - inicjuje obiekt klasy **ChessGame** i wywołuje metodę **gameLoop()** lub **gameLoopOnline()** w zależności od wyboru użytkownika.

• **Piece** - klasa nadrzędna dla klas **ChessPiece** i **CheckersPiece**, które to dziedziczą po niej konstruktor i metodę **whatColor()**.
- **whatColor()** - zwraca kolor danej bierki w postaci znaku ('W' - biały, 'B' - czarny).

• **ChessGame, CheckersGame** - klasy dziedziczące po klasie **Game** zajmujące się obsługą rozgrywki. Przechowują tablicę wskaźników na obiekty typu **Piece** (**game_pos**) stanowiącą aktualny stan partii.

- **drawPiece(Piece* arr[8][8], int, int, int, bool)** - metoda zwracająca obiekt typu **sf::Sprite** z ustawioną odpowiednią teksturą i pozycją względem planszy. Wykorzystywana przez funkcję **draw()** z biblioteki SFML do rysowania figur na planszy (łącznie z rysowaniem aktualnie „przeciąganych" figur).
- **gameLoopOnline(sf::RenderWindow&, sf::RectangleShape&, sf::Sprite)** - wersja funkcji **ChessGame::gameLoop()** przeznaczona do rozgrywki sieciowej. Odpowiada za poprawne połączenie dwóch komputerów w sieci lokalnej używając protokołu TCP oraz wysyłanie i odbieranie danych zawierających współrzędne ruchu wykonanego przez przeciwnika. Wywołuje m.in. metodę **moveMaker(int, int, int, int)** modyfikującą tablicę **game_pos** na podstawie współrzędnych.

4.2 Użyte techniki obiektowe

<thread> i <semaphore>

```
     std::thread t1(sconnect, std::ref(socket),
server, port);
     sf::Event e;

     while (c.try_acquire()) {
          while (w.pollEvent(e)) {
               if (e.type == sf::Event::Closed)
                    w.close();
          }
          drawLoad(w, bg, wait1, wait2);
          c.release();
     }
     t1.join();
```

W programie użyto współbieżności, by można wyświetlić ekran oczekiwania na przeciwnika w metodzie **ChessGame::gameLoopOnline().**
Funkcja **sconnect()**, która jest uruchamiana poprzez utworzenie wątku **t1** odpowiada za połączenie klienta z serwerem (lub utworzenie serwera i czekanie na klienta) i wykorzystuje metodę **connect()** dla obiektu klasy **sf::TcpSocket.**
Metoda ta jest blokująca (i dla poprawnego działania musi taką pozostać), więc przy oczekiwaniu na klienta bez współbieżności program zawiesza się do

momentu utworzenia połączenia z drugim komputerem. Dzięki użyciu <thread> równocześnie z funkcją **sconnect()** wykonuje się pętla while umożliwiająca wyjście z programu oraz wywołująca funkcję **drawLoad()**, która rysuje animację ładowania.
Pętla wykonuje się do momentu zdekrementowania semafora **c** - robi to funkcja **sconnect()**, kiedy nawiąże połączenie z drugim komputerem.

```
     if (a.try_acquire()) {
          client = true;
          std::cout << "Polaczono z serwerem: " <<
server << std::endl;
     }
```

sconnect() w przypadku nawiązania połączenia jako klient inkrementuje semafor **a**, dzięki czemu zmiennej client zostaje przypisana odpowiednia wartość (jest ona istotna do rozpoczęcia rozgrywki siecowej w pętli).

<filesystem> i <regex>

Przykład użycia:

```
     std::string path = "Images/Chess/", str_path;
     std::regex expr(path+"[w|b]\\w{1}.png");
     int i = 0;

     for (const auto& entry : fs::directory_itera-
tor(path)) {
          str_path = entry.path().generic_string();
          if (std::regex_match(str_path, expr)) {
               t[i].loadFromFile(str_path);
               i++;
          }
     }
```

W powyższym przykładzie dzięki wyrażeniu regularnemu **<regex>** oraz iteratorowi ścieżki z biblioteki **<filesystem>** ładowane są odpowiednie tekstury (w tym przypadku są to bierki szachowe) do tablicy obiektów klasy sf::Texture.
fs::directory_iterator iteruje po podanej ścieżce „Images/Chess/”, a funkcja **std::regex_match** filtruje wyniki tak, aby wczytać same bierki.

4.3 Struktury danych

Program oprócz kilkunastu tablic **`sf::Texture`** przechowujących wczytane z plików tekstury dla obiektów typu **`sf::Sprite`** zawiera również tablicę wskaźników na obiekty typu **`Piece`** o wymiarach 8x8 - przechowuje ona aktualny stan partii.
Zawiera również wektor zmiennych typu **`arrPosition`** (własna struktura), potrzebny do przechowywania historii partii, w celu umożliwienia cofania ruchów.