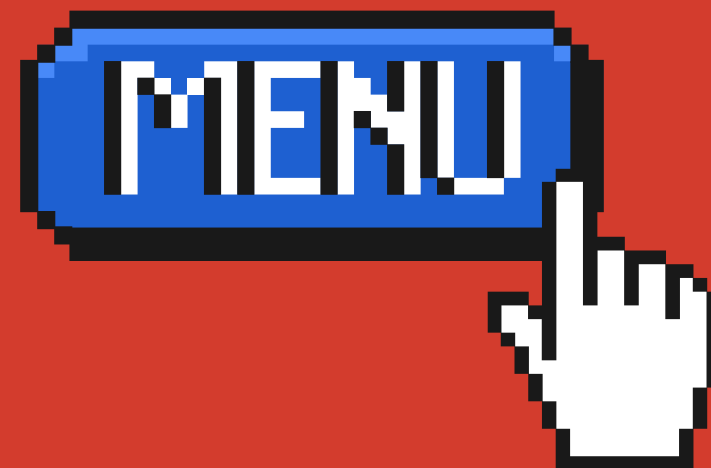


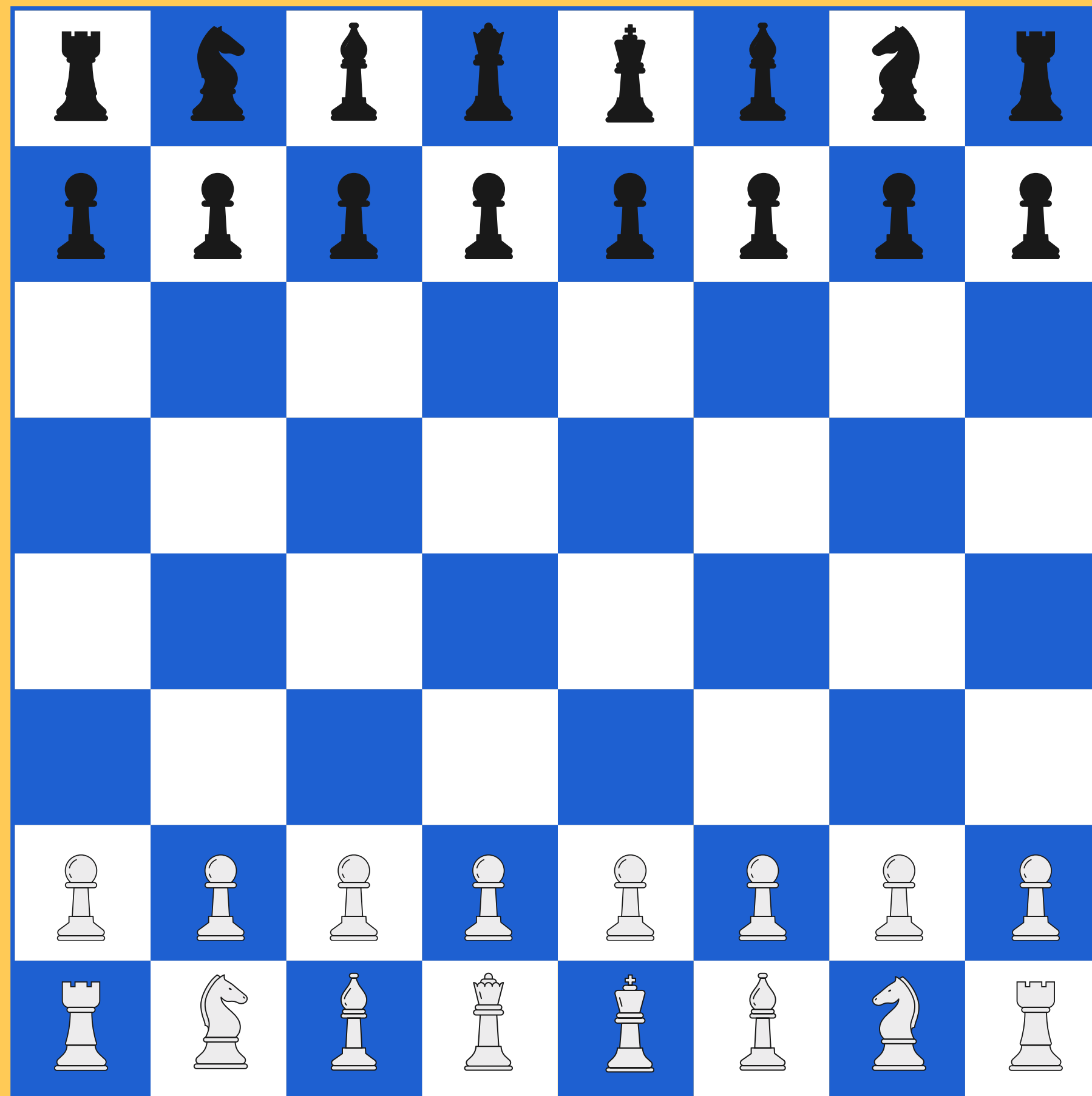
SZACHY

Schemat UML

Norbert Bajurski
Mateusz Kowalczyk
Aleksandra Labeda
Aleksander Tytus



Player 1



Player 2

PIECE

- Klasa abstrakcyjna po której dziedziczą klasy wszystkich bierok.
- Zawiera zmienne reprezentujące gracza (Player) i nazwę figury (String)

SQUARE

- Klasa odwzorowująca pole na szachownicy
- Zawiera koordynaty x,y (int) oraz obiekt klasy Piece który reprezentuje figurę stojącą na danym polu
- Możliwość późniejszego wykorzystania w GUI

MOVE

- Klasa reprezentująca pojedynczy ruch w grze
- Zawiera 2 obiekty typu Square, które informują o miejscu w którym zaczyna się ruch i w którym się kończy
- Zawiera też informację czy dany ruch jest promocją pionka i jeśli tak na jaką figurę następuje zmiana
- Przesyłana między graczami za pośrednictwem serwera

CHESSBOARD

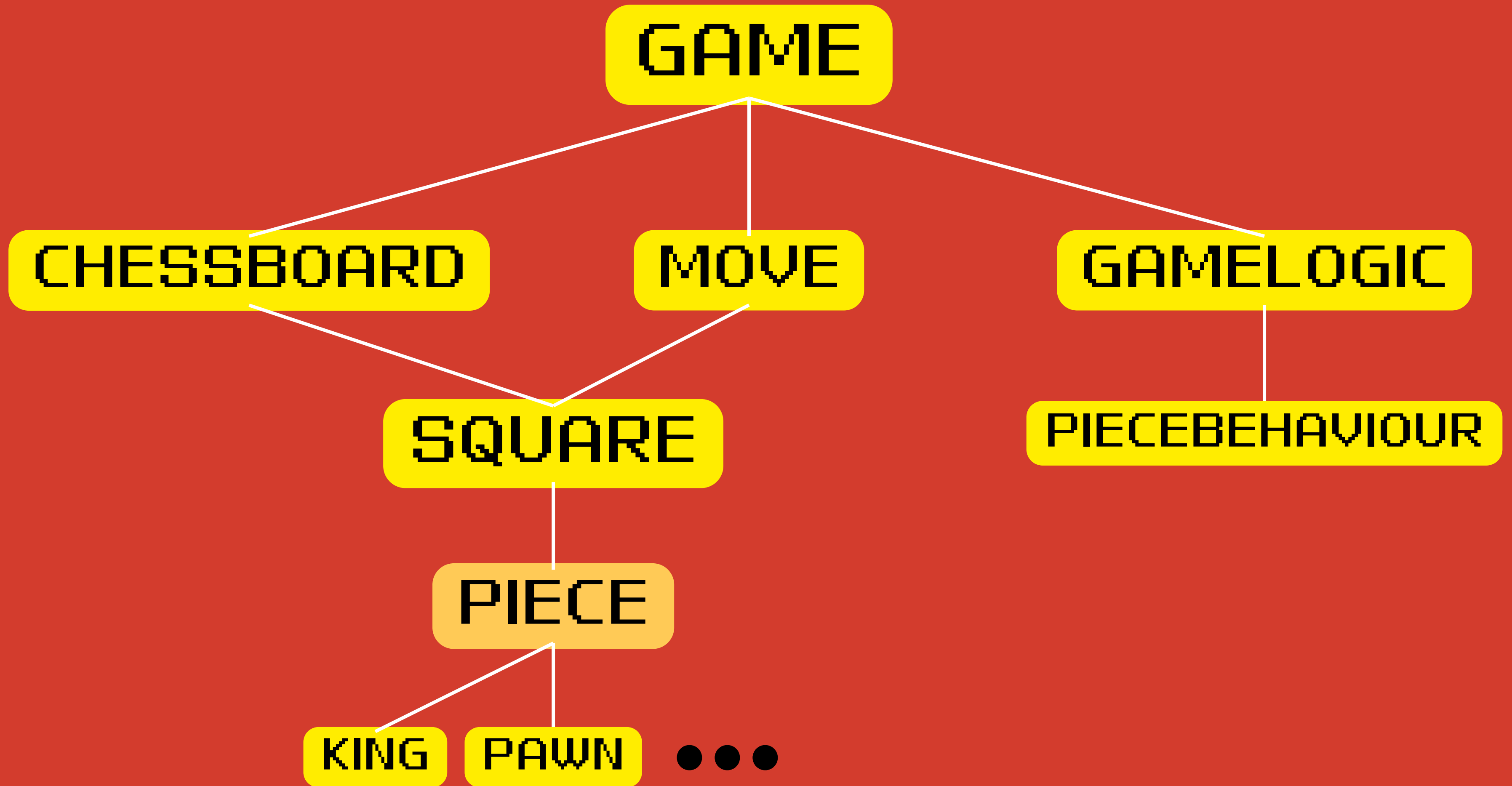
- Klasa reprezentująca planszę do gry.
- Zawiera tablice 8x8 klasy Square, która odwzorowuje stan planszy
- Obsługuje wszystkie operacje związane ze tą tablicą, takie jak inicjalizacja, wykonywanie ruchów i manipulacja polami na planszy.

GAME

- Klasa kontrolująca samą grę (inicjalizacja rozgrywki, wykonywanie ruchu, koniec gry)
- Posiada obiekty: Chessboard, Player (1 czarny, 1 biały)
- Posiada zapis wszystkich ruchów: `ArrayList<Move>`

GAMELOGIC

- Klasa zawiera metody statyczne, które sprawdzają stan gry
- Metody pozwalają m. in. na sprawdzenie, czy dany ruch jest legalny, czy jest szach, czy nastąpiła wygrana bądź remis (pat)



SERVER

- Klasa zarządzająca komunikacją sieciową w grze.
- Metody:
 - `runServer()` : metoda odpowiedzialna za start serwera
 - `run()` : służy do nasłuchiwania połączeń od klientów
 - `newTable(int tableID, String password)` : tworzenie nowego “stołu”, do którego mogą dołączyć klienci
 - `isRunning()` : sprawdzenie czy serwer aktualnie działa

TABLE

- Klasa służąca komunikacji między dwoma graczami
- Metody:
 - `sendMoveToOther(ServerClient sender, Move move, String promoted)`
: wysyła wykonany ruch do drugiego gracza
 - `boolean isAllPlayers()` : sprawdzenie czy do gry dołączyło dwóch graczy
 - `addPlayer()` : dodanie klienta do "stołu"

SERVERCLIENT

- Klasa służąca komunikacji z klientem
- Metody:
 - run() : nasłuchiwanie wiadomości od klienta i ich obsługa

CLIENT

- Klasa służąca do komunikacji z serwerem
- Metody:
 - `boolean join(int tableID, String password, String username)`
: zwraca true, jeśli udało się dołączyć do danego stołu, jeśli nie - false
 - `run()` : nasłuchiwanie wiadomości od serwera
 - `handleCommands()` : obsługa komend wysłanych przez serwer
 - `handleNewMoveFromServer()` : obsługa ruchu wykonanego przez drugiego klienta
 - `sendMove(Move move, String promoted)` : wysłanie wykonanego ruchu do serwera

FILE HANDLER

Obsługuje pliki gry, dostarczając metod do zapisywania i wczytywania danych gry do/z plików XML (Czytelność, Wszechstronność)

- Konstruktor: `FileHandler(String filename)`: Inicjuje obsługę pliku z określoną nazwą.

Metody:

- `saveGameToFile(List<Move> moves)`: Zapisuje dane gry do pliku XML. Przyjmuje listę ruchów reprezentujących historię gry.
- `loadGameFromFile()`: Wczytuje dane gry z pliku XML. Zwraca listę ruchów reprezentujących historię gry.



Cel: Reprezentuje gracza AI.

Metoda:

- **getMove(Game game, Move lastMove):** Metoda abstrakcyjna, która zwraca ruch gracza AI w danym stanie gry.

Ten interfejs definiuje sposób, w jaki gracze AI podejmują decyzje w grze, zapewniając elastyczność w implementacji różnych strategii i poziomów trudności.

AI FACTORY

Cel: Fabryka do tworzenia instancji klas AI na podstawie poziomu. Implementuje prosty wzorzec fabryki.

Metoda:

- **getAI(int level):** Metoda statyczna do pobierania instancji AI na podstawie podanego poziomu. Zwraca instancję AI odpowiadającą określonymu poziomowi.

Ta klasa służy jako fabryka do tworzenia instancji AI zgodnie z różnymi poziomami, oferując elastyczność i abstrakcję w zarządzaniu obiektami AI.

LEVEL1, LEVEL 2

Cel: Reprezentuje poziom trudności AI.

Level1

Metoda:

- **getMove(Game game, Move lastMove):** Implementuje metodę z interfejsu AI, zwracając ruch gracza AI w danym stanie gry.

Reprezentuje pierwszy poziom trudności dla gracza AI, zapewniając konkretną implementację sposobu podejmowania decyzji przez AI w grze.

LEVEL 1, LEVEL 2

Cel: Reprezentuje poziom trudności AI.

Level 2

Metoda:

- **getMove(Game game, Move lastMove):** Implementuje metodę z interfejsu AI, zwracając najlepszy ruch dla AI w danym stanie gry.

Implementuje prosty algorytm dla sztucznej inteligencji na drugim poziomie trudności, dostarczając konkretnej strategii podejmowania decyzji przez AI w grze.

ENUMERACJE

Wykorzystywane do definiowania stałych wartości w grze, takich jak kolory, typy figurek oraz stany szachu. Ułatwiają one czytelność kodu oraz zapewniają spójność w całej aplikacji.

1. Color enum:

- Reprezentuje kolory figur na szachownicy: biały i czarny.
- Zawiera pola **colorName** i **symbol**, przechowujące nazwę koloru i symbol w notacji szachowej.
- Udostępnia metody dostępowe, takie jak **getColorName()**, **getSymbol()**, **getSymbolAsString()**.
-

2. PlayerType enum:

- Określa różne typy graczy w grze: człowiek, użytkownik sieciowy, komputer.
- Używany do różnicowania zachowania interfejsu użytkownika lub strategii gry w zależności od typu gracza.

ENUMERACJE

3. Enum: Protocol

- Określa protokoły komunikacyjne w sieci.
- Wartości: `EVERYTHING_OK`, `ERROR_INVALID_TABLE_ID`, `ERROR_TABLE_IS_FULL`, `ERROR_INVALID_PASSWORD`, `MOVE`, `STOP`, `STOPPED`, `NULL_COMMAND`.
- Udostępnia metodę `get(int id)` do pobierania protokołu na podstawie identyfikatora.

Ten enum definiuje różne protokoły komunikacyjne używane w sieci do kontroli przepływu informacji między serwerem a klientami.

KLASY TYPU
PLAYER

PLAYER

- Interfejs reprezentujący gracza, extenduje Serializable
- Metody:
 - getName(): Zwraca nazwę gracza.
 - setName(String): Ustawia nazwę gracza
 - getColor(): Zwraca kolor gracza.
 - setColor(Color): Ustawia kolor gracza.
 - getPlayerType(): Type - zwraca typ gracza.
 - setPlayerType(Type): ustawia typ gracza
 - isGoDown():boolean - zwraca w którą stronę porusza się gracz
 - setGoDown(boolean): - ustawia kierunek ruszania się gracza
 - getPromotionPiece(Chessboard):Piece - zwraca na jaką figurę gracz chce promować

HUMAN PLAYER

- Klasa reprezentuje gracza ludzkiego, implementuje interfejs Player i wszystkie jego metody abstrakcyjne
- Typ tego gracza to enum- "HUMAN"
- W tej klasie implementacja metod musi być dostosowana do gracza ludzkiego

COMPUTER PLAYER

- Klasa reprezentuje gracza sterowanego przez komputer, implementuje interfejs Player i wszystkie jego metody abstrakcyjne
- Typ tego gracza to enum- "COMPUTER"
- W tej klasie implementacja metod musi być dostosowana do gracza sterowanego przez komputer

NETWORK PLAYER

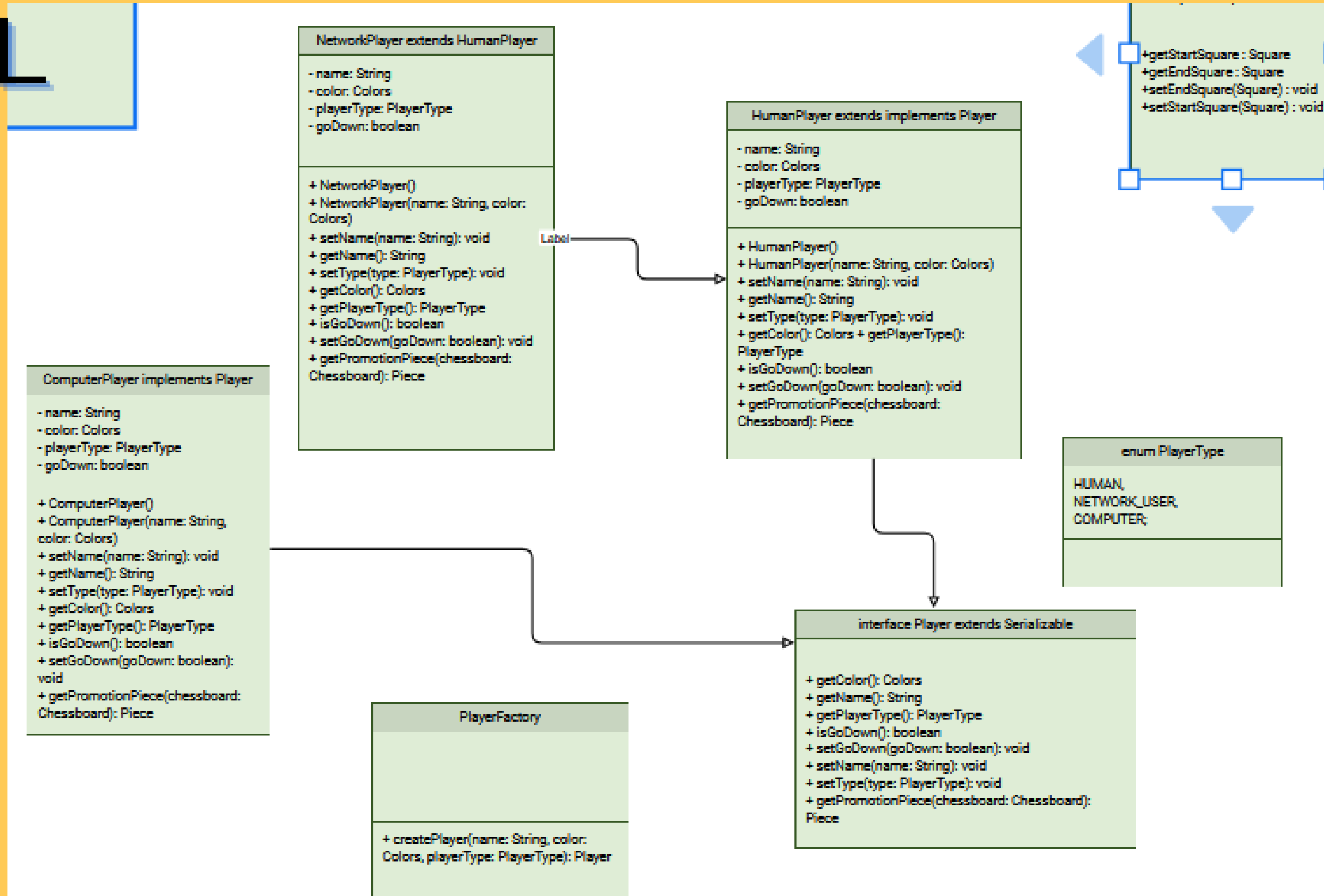
- Klasa reprezentuje gracza grającego przez sieć, extenduje klasę HumanPlayer
- Typ tego gracza to enum- "NETWORK_USER"
- W tej klasie implementacja metod musi być dostosowana do gry sieciowej


PLAYERFACTORY

- Klasa używana do tworzenia instancji różnego rodzaju playerów
- `createPlayer(PlayerType, String name, Color): Player` - tworzy gracza

SCHEMAT UML

- Kawałek schematu dotyczący obsługi playerów





THANK YOU

PLAY AGAIN?

OBVIOUSLY

NO

