|  |  |
| --- | --- |
|  | **AKADEMIA GÓRNICZO-HUTNICZA** |

Dokumentacja do projektu

**GameChess**

z przedmiotu

**Języki programowania obiektowego**

Elektronika 2021/2022

*Michał Nizioł*

czwartek 14:40

prowadzący: Rafał Frączek

10.06.2022

# Opis projektu

Projekt GameChess jest programem umożliwiającym użytkownikowi rozegranie partii w szachy. Partię można rozegrać z drugą osobą lub z prostym botem szachowym.

# Project description

Project GameChess is program that allows user to play chess. User can play against other player or against simple chess bot.

# Instrukcja użytkownika

Po całym interfejsie poruszamy się klawiszami [W,A,S,D]. Po kliknięciu [Enter] wybieramy daną opcję. Aby wykonać ruch, należy najpierw wybrać figurę, którą chcemy się poruszyć, a następnie wybieramy pole, na które chcemy ruszyć wybraną figurą. Jeśli ruch będzie nie prawidłowy, to należy powtórzyć podane czynności. Opcja *New game* pozwala nam zacząć partię od nowa. *Save position* pozwala nam na zapisanie obecnej pozycji. *Load position* wczytuje ostatnią zapisaną pozycję. *Save game record* zapisuje zapis partii, czyli wszystkie wykonane ruch w partii do tego momentu. Zapis będzie się znajdował w pliku tekstowym zapis\_partii.txt. Po przez wybranie opcji *Quit* wychodzimy z programu.

Szachy jest to gra planszowa, którą rozgrywa się na szachownicy 8x8, każda ze stron zaczyna z 16 figurami. Celem gry jest danie mata przeciwnikowi, to znaczy doprowadzenia do takiej sytuacji, w której król jest atakowany i nie może się ruszyć na żadne inne pole bez groźby zbicia w następnym ruchu.

# Kompilacja

Program został napisany w Visual Studio 2022. Kompilacja standardowa.

Przez użycie biblioteki Windows.h ten program działa jedynie na systemach Windows.

# Pliki źródłowe

Projekt składa się z następujących plików źródłowych:

* *GUI.h*, *GUI.cpp* – deklaracja oraz implementacja prostego graficznego interfejsu użytkownika
* *Interface.h, Interface.cpp* – deklaracja oraz implementacja funkcji związanych z poruszaniem się użytkownika po interfejsie oraz zapisywania i wczytywania stanu programu
* *Position*.*h, Position,cpp* – deklaracja oraz implementacja klasy Position odpowiadającej za wyświetlanie figur na szachownicy
* *ChessPieces.h, ChessPieces.cpp* – deklaracja oraz implementacja klasy ChessPieces
* *Bot.h, Bot.cpp* – deklaracja oraz implementacja klasy Bot
* *GameChess.cpp* – właściwy program, symulujący partię szachową

# Zależności

W projekcie wykorzystano następujące dodatkowe biblioteki:

* Windows API – implementuje rożne funkcje związane z Windows API, dodaje wiele przydatnych funkcji związanych z manipulacją kolorem tekstu i tła oraz kursorem w konsoli
* Conio.h – implementuje funkcje związane z wejście/wyjściem konsoli, pozwala na sprawną komunikację programu z użytkownikiem

# Opis klas

W projekcie utworzono następujące klasy:

* GUI – klasa zawiera metody odpowiedzialne za utworzenie interfejsu graficznego
  + static void CreateBoard() – wyświetla w konsoli szachownicę
  + static void CreateMenu() – wyświetla w konsoli menu programu oraz instrukcję użytkowania
* Position – klasa zawiera statyczne metody związane z wyświetlaniem figur na szachownicy
  + static void PiecePosition(int, int, ChessPieces&) – zapisuje bieżący stan programu
  + static void SetAllPieces(vector<ChessPieces>&, vector<ChessPieces>&) – ustawia wszystkie figury na swoje pozycje startowe
  + static void Clear() – czyści szachownicę ze wszystkich figur, które się na niej znajdują
* Interface – klasa zawierająca metody pozwalające użytkownikowi na poruszanie się po interfejsie oraz zapisywania i wczytywania stanu programu
  + void Coord() – dokonuje konwersji wirtualnych współrzędnych związanych z wyborem figury i ruchu na prawdziwe współrzędne kursora w konsoli
  + void CoordMenu() – dokonuje konwersji wirtualnych współrzędnych związanych z menu na prawdziwe współrzędne kursora w konsoli
  + void CoordP()– dokonuje konwersji wirtualnych współrzędnych związanych z wyborem promowanej figury na prawdziwe współrzędne kursora w konsoli
  + void Choice() – pozwala użytkownikowi na wykonanie ruchu oraz ewentualnie dokonanie innych akcji (np. wyjście z programu, zapis itp.)
  + void Menu() – odpowiada za poruszanie się po menu przez użytkownika
  + void Return() – pozwala użytkownikowi na cofnięcie ruchu
  + void Move() – tworzy listę wykonanych do tej pory ruchów
  + void Promotion() – pozwalana użytkownikowi na wybranie figury, w którą chce wypromować swojego piona na ostatniej linii
* ChessPieces – reprezentuje figurę szachową, zawiera wszystkie metody odpowiedzialne za sprawdzanie legalności danych ruchów
  + void Set\_Symbols() – ustawia symbole zależne od typu figury widoczne w konsoli
  + void Set\_ID() – ustawia unikalne ID dla danej figury składające się z pozycji na szachownicy oraz typu figury
  + static bool AllowedMoves(int, int, vector<ChessPieces>&) – metoda zwraca *true* jeśli na współrzędnych (x,y) nie znajduje się figura tego samego koloru
  + static bool AllowedMovesKing(int, int, vector<ChessPieces>&, vector<ChessPieces>&, Color) – zwraca *true* jeśli król może się poruszyć na dane współrzędne (x,y)
  + bool AllowedMoves(int, int, vector<ChessPieces>&, vector<ChessPieces>&, Color, int) – zwraca *true* jeśli figura może się poruszyć na dane pole (uwzględnia obecność innych figur na szachownicy, ale ignoruje szachy)
  + bool Castle(int, int, vector<ChessPieces>&, vector<ChessPieces>&, Color) – zwraca *true* jeśli moża wykonać roszadę w prawo
  + bool CastleL(int, int, vector<ChessPieces>&, vector<ChessPieces>&, Color) – zwraca *true* jeśli można wykonać roszadę w lewo
  + bool Block(int, int, vector<ChessPieces>, vector<ChessPieces>, Color, int) – zwraca *false* jeśli po ruchu na współrzędne (x,y) nasz król był by szachowany.(w języku szachowym - sprawdza czy dana figura jest związana - przy wartości *false* jest związana)
  + bool Cover(int, int, ChessPieces&, ChessPieces&, vector<ChessPieces>&, vector<ChessPieces>&) – zwraca *true* jeśli można się ruszyć na współrzędne (x,y) znajdującą się pomiędzy szachowanym królem, a figurą szachującą
  + static ChessPieces TypeOfPieces(int, int, vector<ChessPieces>&, vector<ChessPieces>&, Color) – zwraca figurę, która szachuje króla w danym momencie
  + static bool Mate(vector<ChessPieces>&, vector<ChessPieces>&, Color) – zwraca *true* jeśli został dokonany mat
  + static void Take(int, int, vector<ChessPieces>&, vector<ChessPieces>&, Color, string&) – sprawdza czy po ruchu na współrzędne (x,y) została zdobyta jakaś figura
  + void Promotion() – metoda odpowiadająca za dokonanie promocji piona na wybraną figurę
  + void QBR(int, Matrix&, vector<ChessPieces>&, vector<ChessPieces>&) – sprawdza poprawność ruchów figury typu Queen, Bishop, Rook w danej pozycji
  + Matrix AllowedMoves(vector<ChessPieces>&, vector<ChessPieces>&, Color, int) – zwraca macierz 8x8 składającą się z wartości boolowskich, jeśli wartość macierzy [x,y] jest równa *true,* to figura może się poruszyć na współrzędne (x+1,y+1)
  + void AllowedMoves(Matrix&, vector<ChessPieces>&, vector<ChessPieces>&) – modyfikuje macierz zwracaną z metody Matrix AllowedMoves, uwzględniając pozycje, w których król jest szachowany
  + static vector<string> AllowedMoves(Matrix&) – zwraca wektor składający się ze wszystkich możliwych ruchów dla danej figury
* *Bot* – klasa zawiera metody odpowiedzialne za działanie tzw. bota szachowego;
  + VectorOfPieces(vector<string>&)– tworzy listę obiektów *ChessPieces*
  + bool Is\_legal(string&, char, char) – zwraca *true* jeśli ruch jest legalny
  + vector<string> Legal\_moves(string&, vector<string>&)– zwraca wektor legalnych ruchów dla danej figury
  + string Move(vector<string>&, int, int) – metoda odpowiadająca za wybranie ruchu, który ma zostać wykonany przez bota
  + string Take(vector<string>&, int) – szuka tzw. „killer moves” – czyli ruchów, które zdobywają figurę przeciwnika
  + int Power(vector<string>&, char) – zwraca siłę wszystkich figur danej strony
  + void Power(char, int&)– zwraca siłę pojedynczej figury
  + void Take(vector<ChessPieces>&, vector<ChessPieces>&, int&, vector<string>&, vector<string>&) – szuka tzw. „killer moves” – czyli ruchów, które zdobywają figurę przeciwnika (łatwiejszy bot)
  + void \_Move(int, int , vector<ChessPieces>& , vector<ChessPieces>& , int , string& , int& , vector<string>&)– metoda pomocnicza do metody void Move(…)
  + void Move(int&, int&, vector<ChessPieces>&, vector<ChessPieces>&, int&, vector<string>&, vector<string>&)– metoda ta odpowiada za wykonanie ruchu przez łatwiejszego bota

# Zasoby

brak

# Dalszy rozwój i ulepszenia

* Ulepszenie bota szachowego, dodanie różnych poziomów trudności
* Dodanie możliwości analizy partii. Utworzenie silnika szachowego.
* Możliwość ustawiania własnych pozycji.

# Inne

W programie są użyte znaki z Unicode, np. symbole przedstawiające figury szachowe, które są widoczne jedynie przy niektórych czcionkach. Dlatego po uruchomieniu programu pierwszy raz należy kliknąć prawym przyciskiem myszy na okienko konsoli i wybrać opcję właściwości. Następnie należy ustawić rozmiar czcionki na 18 oraz typ czcionki na MS Gothic. Wtedy szachownica wraz z figurami na pewno zostanie wyświetlona prawidłowo.