Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki

Programowanie Komputerów 2

„Gra w szachy”

Autor	Dawid Musiał
Prowadzący	mgr inż. Wojciech Dudzik
rok akademicki	2017/2018
kierunek	informatyka
rodzaj studiów	SSI
semestr	2
termin laboratorium	czwartek tydzień nieparzysty, 10:00 – 11:30
grupa	3
termin oddania sprawozdania	2018-06-25
data oddania sprawozdania	2018-06-25

1.Treść zadania

Napisać w pełni działającą grę w szachy z zapisem i odczytem z pliku.

2.Analiza zadania

Zagadnienie przedstawia problem utworzenia programu, który będzie wyświetlał szachownicę z obecnym położeniem bierek, reagował na błędne posunięcia któregoś z graczy, sygnalizował posunięcie szachowe (zagrożenie króla), bądź w razie zamatowania króla stwierdzał, że takowy szach - mat wystąpił i po której stronie (który gracz wygrał). W dowolnym momencie gry można jej obecny stan zapisać do pliku (polecane rozszerzenie .txt). Zaś włączając program w startowym menu można wybrać czy chcemy zacząć nową grę czy wczytać stan poprzedni z pliku.

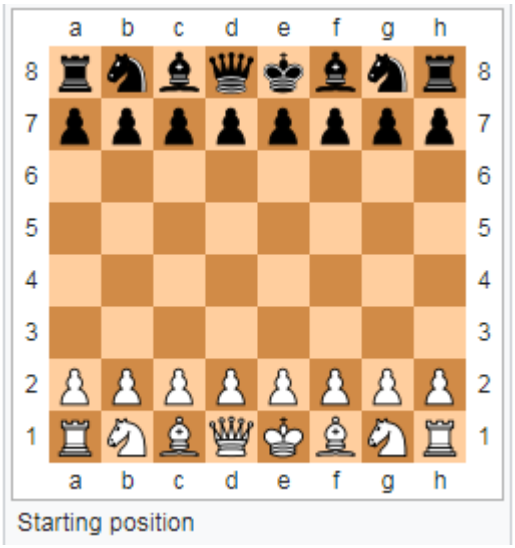
2.1 Struktury danych

W programie utworzono dwuwymiarową dynamiczną tablicę[8][8] struktury `Board_struct`, zdefiniowanej przez programistę, która odpowiada za gromadzenie wszelkich danych na temat znajdujących się na szachownicy bierek i innych informacji, które zostaną opisane w dalszej części sprawozdania.

Poza tym na potrzeby programu utworzono wiele zmiennych lokalnych potrzebnych do obsługi programu jak np. `Player turn`, która przechowuje informacji, który gracz powinien wykonać poprawny ruch, bądź `Bool white_King_threatened`, `black_King_threatened`, które informują wystąpieniu zagrożeniu dla króla danego koloru.

2.2 Algorytmy

Dla każdego typu bierek zdefiniowano oddzielne algorytmy sprawdzania czy dana bierka może poruszyć się na dane pole. Dla przykładu wieża porusza się tylko w linii prostej poziomo lub pionowo o dowolną ilość pól, pod warunkiem, że na polu docelowym nie znajduje się bierka tego samego koloru i na drodze pomiędzy polem startowym, a docelowym nie przeszkadza żadna inna bierka.
Zaś dla skoczka może on zawsze się poruszać o dwa pola w pionie i jedno w poziomie lub odwrotnie tzn. dwa w poziomie i jedno w pionie (kształt litery L w ośmiu wariantach dla różnych obrotów i odbić tejże litery).



Początkowe ułożenie bierek w szachach i oznaczenia danych pól.

Sprawdzanie czy w danym ułożeniu bierek występuje szach w polegało na odnalezieniu na szachownicy króla i po kolei każdej z bierek przeciwnika, a następnie bazując na funkcjach odpowiedzialnych za poruszanie się danych figur stwierdzenie czy dana bierka zagraża królowi.
Bardzo ciekawym zagadnieniem było sprawdzanie czy w danej kombinacji ułożenia bierek na szachownicy i kolejki danego gracza występuje szach – mat. Dla danej sytuacji jedna z funkcji wyszukuje bierki zawodnika, który musi bronić się przed szachem i sprawdza na jakie pola może się poruszyć, następnie przestawia je tam (zawsze tylko jedną na raz) i sprawdza czy w danej kombinacji nadal występuje szach, (po całej operacji przestawia bierki na poprzednie miejsce) jeśli dla wszystkich bierek i wszystkich ich możliwych ruchów, ani jedna kombinacja nie ustrzegła przed szachem zostaje stwierdzony koniec gry i gracz, który zaszachował króla wyrywa.

3. Specyfikacja zewnętrzna

Przy uruchomieniu programu naszym oczom ukazuje się przywitanie i pytanie co chcemy zrobić?
0 - rozpoczęcie gry z startowym ustawieniem bierek, 1 - wczytanie ułożenia bierek z pliku. Wybierając '0' naszym oczom ukazuje się szachownica z oznaczeniami pól (liczba i litera), a na niej bierki w pozycji startowej. Zgodnie z zasadami gry w szachy zaczynają białe. Następnie operując strzałkami należy najechać na figurę, którą chcemy ruszyć, potem klikając klawisz Enter zatwierdzamy nasz wybór, jeżeli została wybrana prawidłowa bierka (jej kolor), to zostaje wyróżniona innym kolorem, dalej znowu operując strzałkami wybieram miejsce, na które ma przemieścić się bierka i dalej Enterem zatwierdzam wybór. Gdyby któraś z decyzji została podjęta nieprawidłowo program wypisze stosowny komunikat i poprosi o ponowne wybranie bierki. Gdy występuje szach po którejś ze stron na ekranie zostaje wyświetlony stosowny komunikat, przewidziano, że gracz czasami może nieświadomie narazić się na szacha, co w przepisach gry jest niedozwolone, wtedy program również o tym poinformuje. Podczas rozgrywki wybierając klawiatury klawisz 's' zapiszemy obecny stan gry do pliku, lecz najpierw wyświetli się prośba o wpisanie nazwy pliku, w którym ma być zapisana gra. Wybierając '1' zaraz po uruchomieniu programu zostaje wysłana prośba o podanie nazwy pliku z którego ma zostać wczytana gra. Jeśli wpisane dane będą poprawne naszym oczom ukaże się szachownica z wczytanym stanem gry.
Dla promocji piona tj. jego dojście do końca szachownicy (zgodnie z zasadami gry) wyświetlana jest prośba z podaniem znaku odpowiadającemu danej figury, na którą chcemy zamienić piona, gdy już wybierzemy odpowiednią opcję pion zostanie zamieniony na wybraną figurę.

4. Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem strukturalnym. W programie rozdzielono interfejs (komunikację z użytkownikiem) od logiki aplikacji (wybieranie bierek, pól docelowych, sprawdzenia czy dany ruch jest prawidłowy, sprawdzania wystąpienia sytuacji zagrożeń dla króla (szach), zakończenia rozgrywki(szach-mat), wyświetlania obecnego stanu gry i sytuacji na szachownic).

4.1 Typy zdefiniowane w programie

W programie zdefiniowano następujące typy wyliczeniowe:

```
typedef enum { False = 0, True = 1 } Bool;
typedef enum { Black = -1, Nobody = 0, White = 1 } Player;
typedef enum {
      Empty = 0, White_King = 1, White_Queen = 2, White_Rook = 3, White_Bishop
= 4, White_Knight = 5, White_Pawn = 6,
      Black_King = -1, Black_Queen = -2, Black_Rook = -3, Black_Bishop = -4,
Black_Knight = -5, Black_Pawn = -6} Pieces;
typedef enum {Horizontally = 0, Upright = 1} Orientation;
```

`Bool` jest reprezentacją własnego typu logicznego bool (nazwa zapożyczona z C++).
`Player` służy do określania, który gracz obecnie powinien wykonać swój ruch, bądź który z graczy zwyciężył.
`Pieces` jest reprezentacją wszystkich możliwych bierek na szachownicy oraz ich brak.
`Orientation` służy do określania orientacji na poziomą lub pionową.
Oraz własna struktura:

```
typedef struct {
      Pieces Piece_on_square;
      char letter;
      char number;
      Bool have_it_been_moved;
      Bool is_it_chosen;
} Board_struct;
```

Piece_on_square; informuje o bierce występującej na danym polu szachownicy,
letter; to litera, którą oznaczone jest pole,
number; to liczba, którą oznaczone jest pole,
have_it_been_moved; informuje czy dana bierka została wcześniej poruszona podczas gry,
is_it_chosen; przechowje informacje czy w danym momencie bierka znajdująca na tym polu zostaje wyróżniona.

4.2 Ogólna struktura programu

Przy uruchomieniu programu naszym oczom ukazuje się powitanie użytkownika i zapytanie czy chce zaczynać grę od nowa czy wczytać stan gry z pliku. W przypadku wybrania pierwszej opcji następuje stworzenie wskaźnika do dwuwymiarowej tablicy, a następnie poprzez funkcję **Board_struct** Make_Board()**; stworzenie dynamicznej tablicy dwuwymiarowej typu **Board_struct**. Dalej szachownica zostaje wypełniona wartościami początkowymi poprzez funkcję **Board_struct** Fill_Board.** Zostają zainicjalizowane zmienne potrzebne do rozpoczęcia gry, takie jak np. **Player turn**, która informuje o tym, który gracz powinien wykonać ruch, potem zostaje wywołana funkcja **playing,** która czuwa nad przebiegiem gry, aż do stwierdzenia przez funkcję wystąpienie szach – mata. Podczas rozgrywki istnieje możliwość zapisania stanu gry, wystarczy nacisnąć klawisz ‘s’, a następnie dowolny klawisz inny od klawiszy funkcyjnych (np. dowolną literę).
W przypadku wybrania drugiej opcji (wczytanie gry z pliku) użytkownik/użytkownicy zostaną poproszeni o podanie nazwy pliku, jeśli zostanie podana niepoprawna nazwa pliku użytkownik zostanie poproszony o ponowne uruchomienie programu. Gdy plik zostanie odnaleziony i stan gry będzie wczytany za pomocą funkcji **read_from_file**

Program sprawdza czy gra została zapisana w stanie szacha czy bez. W zależności od tego czy ta sytuacja występuje są uruchamiane funkcje:

1) brak szacha – funkcja **playing** (jej definicja była podana wyżej)
2) wystąpienie szacha – funkcja **Player support_for_condition_from_check,** a następnie również funkcja **playing,** jak w pierwszym przypadku,
   aż do stwierdzenia szacha. Dalej już tylko zostaje wyświetlony komunikat o zwycięzcy i gra po 200 sekundach zamyka się.

4.3 Szczegółowy opis implementacji funkcji

Uwaga, opisy omawiające argumenty pobierane przez poniższe funkcje podane są w kolejności ich deklaracji w funkcji.

a) wyświetlanie

```
void display_Piece(Board_struct** Board, int i, int j);
```

funkcja pomocnicza służąca do obsługi wyświetlania pojedynczej bierki na szachownicy, funkcja pobiera: wskaźnik na szachownicę i współrzędne na szachownicy **i, j**, gdzie może zostać wyświetlona bierka. Nic nie jest przez nią zwracane.

```
void display_letters();
```

służy do wyświetlenia liter na górze, bądź dole szachownicy.

```
void display_top_of_row();
```

funkcja nie przyjmuje argumentów, jej zadaniem jest wyświetlanie znaków odpowiedzialnych za górną reprezentację jednego z pól szachownicy.

```
void display_middle_square(Board_struct** Board, int i, int j, int x, int y,
Pieces colour_of_piece);
```

odpowiedzialna za wyświetlenie zawartości (bierki lub jej braku) na szachownicy, **Board**, to wskaźnik na szachownicę, dalej współrzędne **i, j,** gdzie ma zostać wyświetlona bierka**, x, y** to współrzędne kursora na szachownicy, dalej zmienna informująca o bierce znajdującej się na współrzędnych **i** oraz **j.**

```
void display_instructions(Player *turn, char *info);
```
funkcja wyświetla instrukcje pod szachownicą, argumenty to: wskaźnik na zmienną informującą o tym, który gracz powinien poruszyć swoją bierką oraz łańcuch znakowy z informacją do wyświetlenia.

```
void displayChessboard(Board_struct** Board, Player *turn, int x, int y,
char* info);
```
służy do wyświetlania obecnego ułożenia bierek na szachownicy i komunikatów z nim związanych, przyjmuje jako argumenty wskaźnik na szachownicę, wskaźnik na zmienną przechowującą informacje o tym, kto ma wykonać ruch, współrzędne **i, j** do podświetlania bierek, by móc zlokalizować obecne położenie na szachownicy, oraz łańcuch znaków, w którym zawarty jest komunikat do wyświetlenia.

```
void new_cordinate_arrow(int *x, int *y, int * arrow);
```
funkcja odpowiedzialna za przesuwanie kursora po szachownicy, argumenty to: wskaźniki na współrzędne kursora **x** i **y**, wskaźnik na zmienną przyjmującą wartość klawisza strzałki z klawiatury.

```
void arrow_control(Board_struct **Board, Player *turn, int *x, int *y, char
*str1, int *Which_function, Bool *white_King_threatened,
Bool *black_King_threatened, Board_struct *last_move);.
```

Jej zadaniem jest sterowanie bierkami podczas gry za pomocą przycisków strzałek i /lub zapisu obecnego stanu gry do pliku. Jej argumenty to: wskaźnik na szachownicę, współrzędne **i, j** do podświetlania obecnie wybranego pola, łańcuch znakowy służący do wyświetlania komunikatu, zmienną, która ma za zadanie określić czy po odczytaniu gry z pliku był szach czy nie (1 oznacza szacha, 0 jego brak), wskaźnik na informację typu logicznego o zagrożeniu białego króla, następnie czarnego króla i wskaźnik na informację o ostatnim ruchu przeciwnika.

b) Sprawdzanie czy król jest zagrożony

Poniższe funkcje są bliźniaczo podobne, mają za zadanie stwierdzić czy dany rodzaj figury szachuje króla, dlatego ich argumentami są: wskaźnik na szachownicę, zmienna przechowująca informację o kolorze króla i zmienna informująca o bierce hipotetycznie szachującej i jej kolorze,

```
Bool which_piece_threaten(Board_struct **Board, Pieces colour_of_piece, int a,
int b, int i, int j);
```
Funkcja jest wywoływana w funkcji **Does_piece_threaten_King,** gdzie z postawionych warunków wywołuje odpowiednią funkcję dla sprawdzenia czy król danego koloru jest zagrożony. Jej argumentami są: wskaźnik na szachownicę, zmienna reprezentująca bierkę hipotetycznie szachującą, współrzędne **a** i **b** bierki szachujące oraz współrzędne **i, j** określające położenie króla. Zwraca wartość **True**, gdy król jest zagrożony bądź **False**, gdy nie jest.

```
Bool Does_piece_threaten_King(Board_struct **Board, Pieces colour_of_King,
Pieces colour_of_threating);
```
Funkcja sprawdzająca czy król jest zagrożony przez daną bierkę, argumenty: wskaźnik na szachownicę, zmienna oznaczająca bierkę szachującą oraz zmienna informująca o kolorze bierki.

Poniższa funkcja korzysta z 2 funkcji powyżej, jest funkcją zbiorczą, tzn, daje końcową odpowiedź czy szach występuje na podstawie 5 wywołań (dla każdej z bierek). Jej argumenty to: wskaźnik na szachownicę, kolor króla, i dowolna szachująca bierka (ważny jest tak naprawdę jej kolor). Zwraca wartość **True**, bądź **False.**

```
Bool Is_King_threatened(Board_struct** Board, Pieces colour_of_King, Pieces
colour_of_threating);
```

c) Zapis i odczyt z pliku

Pierwsza z nich służy do zapisu do pliku, druga odczytu, ich argumenty są takie same tzn.:
Łańcuch znakowy z informacją o nazwie pliku, łańcuch znakowy zawierający komunikat wyświetlany na ekranie, wskaźnik na szachownicę, wskaźnik na zmienną informującą o kolejce, wskaźnik na zmienną informującą o tym czy w danej sytuacji któryś z króli znajduje się w sytuacji zagrożenia(program po odczytaniu z pliku potrzebuje wiedzieć w którym miejscu ma zacząć kierować rozgrywką), wskaźnik na informacje o zagrożeniu białego króla, wskaźnik na informację o zagrożeniu czarnego króla, wskaźnik na informację o ostatnim ruchu przeciwnika. Zwracają wartość **`True`** w przypadku prawidłowego zapisu/odczytu, bądź **`False`** w razie niepowodzenia**.**

```
Bool save_to_file(char* name_of_file, char* info, Board_struct **Board,
Player *turn, int *Which_function, Bool *white_King_threatened,
Bool *black_King_threatened, Board_struct *last_move);
```

```
Bool read_from_file(char* name_of_file, char* info, Board_struct **Board,
Player* turn, int *Which_function, Bool *white_King_threatened,
     Bool *black_King_threatened, Board_struct *last_move);
```

d) funkcje odpowiedzialne za tworzenie szachownicy i wypełnienie jej wartościami początkowymi (ustawienie bierek w położeniu początkowym)

```
Board_struct** Make_Board();
```
Funkcja ma na celu stworzenie szachownicy (dwuwymiarowej tablicy dynamicznej struktury `Board_struct` zdefiniowanej przez programistę. Nie pobiera żadnych argumentów.

```
Board_struct** Fill_Board(Board_struct** Board);
```
Funkcja wypełnia szachownicę wartościami początkowymi, jedynym pobieranym argumentem jest wskaźnik na szachownicę

e) możliwe posunięcia bierek – funkcje zwracają odpowiedź typu logicznego czy dany ruch jest możliwy z podstawowych zasad poruszania się bierek. Podane w kolejności: król, wieża, skoczek, goniec, hetman, pion. Ich pierwsze cztery argumenty to zawsze litera, liczba pola, z którego porusza się figura, liczba i litera pola na które figura ma zostać przesunięta, zmienna informująca co znajduje się na polu docelowym (jaka bierka, lub czy jest tam pusto) oraz zmienna zawierająca informację o kolorze poruszającej się bierki.

```
Bool King_move(int letter_move_to, int number_move_to, int number_move_from,
int letter_move_from, Pieces Is_it_empty,  Pieces colour);
```

```
Bool Rook_move(int letter_move_to, int number_move_to, int number_move_from,
int letter_move_from, Pieces Is_it_empty, Pieces colour, Board_struct** Board);
```
Dodatkowym jej argumentem jest wskaźnik na szachownicę

```
Bool Knight_move(int letter_move_to, int number_move_to, int number_move_from,
int letter_move_from, Pieces Is_it_empty, Pieces colour);
```

```
Bool Bishop_move(int letter_move_to, int number_move_to, int number_move_from,
int letter_move_from, Pieces Is_it_empty, Pieces colour, Board_struct** Board);
```
Dodatkowy argument - wskaźnik na szachownicę.

```
Bool Queen_move(int letter_move_to, int number_move_to, int number_move_from,
int letter_move_from, Pieces Is_it_empty, Pieces colour, Board_struct** Board);
```
Dodatkowy argument - wskaźnik na szachownicę

```
Bool Pawn_move(int letter_move_to, int number_move_to, int number_move_from,
int letter_move_from, Pieces Is_it_empty, Pieces colour, Board_struct **Board,
Bool Have_it_been_moved, Board_struct last_enemy_move, Player turn);
```
Dodatkowe argumenty: wskaźnik na szachownicę, informacja o tym czy wybrany pion był wcześniej poruszany, informacja o ostatnim ruchu przeciwnika (potrzebna do poprawnego zdefiniowania zbicia w przelocie) oraz informacja o tym, który kolor wykonuje ruch.

```
void Pawn_promotion(Board_struct** Board, int number_move_to,
int letter_move_to, Player *turn, int *x, int *y);
```
Funkcja zmienia piona na żądaną przez użytkownika figurę w ramach promocji piona. Dodatkowe argumenty to wskaźnik na zmienną informującą o ruchu, wskaźniki x, y wskazujące na zmienne określające położenie piona, który ma zostać zamieniony.

f) Funkcje odpowiedzialne za obsługiwanie gry i wyznaczania czyja kolej:
nazwy argumentów funkcji o tych samych nazwach oznaczają to samo: (te z gwiazdką są wskaźnikami na poniżej opisane zmienne)
Board – wskaźnik na szachownicę,
Which_one – zmienna reprezentująca bierkę, która została wybrana, aby ją poruszyć
x, y, - współrzędne pola, na które ma zostać przesunięta bierka
oldX, oldY – współrzędne pola z którego bierka ma zostać przemieszczona
last_move - informacje o polu na jakie poruszył się ostatnio przeciwnik
turn – zmienna mówiąca o tym czyj jest obecnie ruch
is_it_correct – zmienna informująca o tym czy wybrano prawidłowy kolor bierki
possible_move – zmienna informująca o tym, czy wybrany ruch jest dozwolony
info – łańcuch znakowy z informacją, który ma zostać wyświetlony w konsoli
What_was_there – zmienna przechowująca informację o tym jaka bierka znajdowała się na docelowym polu przed ruchem bierki przeciwnika
black_King_threatened – zmienna informująca czy czarny król jest zagrożony
white_King_threatened – zmienna informującą czy biały król jest zagrożony?
Which_function – zmienna informująca w jakiej obecnie funkcji program się znajduje (potrzebna dla prawidłowego prowadzenia gry po wczytaniu stanu gry z pliku)

```
Bool Is_it_correct_move(Board_struct** Board, Pieces Which_one, int x, int y,
int oldX, int oldY, Board_struct last_move, Player turn);
```
Funkcja informująca czy dany ruch jest możliwy do wykonania z podstawowych warunków poruszania się. Zwraca wartość **True**, bądź **False.**

```
Bool Help_for_choosing_piece(Player turn, Pieces Which_one);
```
Funkcja zwracająca informację o tym czy wybrano prawidłowy kolor bierki do ruchu, jej argumenty to: informacja o kolejce i bierka, która została wybrana. Zwraca wartość **True**, bądź **False.**

```
void return_from_conditions(Board_struct **Board, Bool *is_it_correct,
Bool *possible_move, char **info, int *oldX, int *oldY, int *x, int *y,
Pieces *What_was_there, Pieces *Which_one, Bool *black_King_threatened,
Bool *white_King_threatened);
```
Funkcja odpowiedzialna za wysyłanie odpowiednich komunikatów na do funkcji wyświetlającej je w konsoli oraz w pewnym przypadku ustawienie bierek do pozycji sprzed poprzedniego ruchu

```
void correct_piece_condition(Board_struct **Board, Bool *possible_move, Bool
*is_it_correct, Player *turn, int *x, int *y,int *oldX, int *oldY,
Bool *white_King_threatened, Bool *black_King_threatened, int *Which_function,
char *info, Pieces *Which_one, Board_struct *last_move);
```
Funkcja odpowiedzialna za uruchomienie funkcji dla wybrania miejsca docelowego dla bierki.

```
void checking_condition(Board_struct **Board, Pieces *What_was_there, Pieces
*Which_one, Bool *black_King_threatened, Bool *white_King_threatened, int *x,
int *y, int *oldX, int *oldY);
```
Funkcja przestawiająca bierkę na docelowe miejsce i wywołująca funkcję sprawdzającą czy król jest szachowany.

```
Player checkmate_after_move(Board_struct **Board, Bool *possible_move,
Bool *is_it_correct, Bool *black_King_threatened, Bool *white_King_threatened,
Bool *Maybe_it_is_checkmate, int *x, int *y, int *oldX, int *oldY,
int *Which_function, Pieces *Which_one, Pieces *What_was_there,
Board_struct *last_move, char *info, Player *turn);
```
Funkcja odpowiedzialna za wywoływanie funkcji do sprawdzenia czy w obecnym ułożeniu bierek występuje szach – mat (również po promocji piona). Zwraca kolor zwycięzcy w przypadku wystąpienia szach – mata.

```
Player playing(Player *turn, Board_struct** Board, int *Which_function);
```
Funkcja sterująca całą rozgrywką i zwracająca informację o kolorze zwycięzcy w przypadku wystąpienia szach – mata.

g) Wsparcie dla obsługi sytuacji z szachem – nazwy argumentów funkcji o tych samych nazwach oznaczają to samo:
Board – wskaźnik na szachownicę,
colour_of_piece - bierka koloru króla, która może hipotetycznie ochronić króla przed szachem,
colour_of_King – kolor króla,
last_enemy_move – ostatni ruch przeciwnika,
turn – informacja o obecnej kolejce,
is_it_correct – wskaźnik na informację czy wybrano odpowiedni kolor bierki,
possible_move - wskaźnik na informację czy dany ruch jest możliwy z podstawowych zasad poruszania się bierek,
black_King_threatened – wskaźnik na informację czy czarny król znajduje się w pozycji zagrożonej,
white_King_threatened – wskaźnik na informację czy biały król znajduje się w pozycji zagrożonej,
x, y – wskaźniki na współrzędne obecnie wskazywane przez użytkownika,
oldX, oldY – wskaźniki na współrzędne, pola, z którego chce poruszyć się bierka,
Which_one, - wskaźnik na informację którą bierką gracz chce się przesunąć,
last_move – wskaźnik na informację o ostatnim ruchu przeciwnika,
info – łańcuch znakowy, w którym zawarty jest komunikat do wyświetlania w konsoli,
What_was_there – wskaźnik na zmienną informującą jaka bierka znajdowała się na polu docelowym dla ruchu innej bierki (potrzebna, by w razie zrobienia niedozwolonego ruchu móc wrócić do ułożenia bierek sprzed przestawienia)
Which_function - wskaźnik na informację, która funkcja ma zostać uruchomiona zaraz po odczytaniu z pliku gry.
start_from_file – zmienna informująca czy gra została wznowiona z pliku w stanie zagrożenia króla.

```
int negate(int argument);
```
Funkcja mająca za zadanie zwracanie wartości przeciwnej do podanej w argumencie.

```
Bool is_it_checkmate_after_move(Board_struct **Board, Pieces *makeshift_piece,
Pieces colour_of_King, Pieces colour_of_piece, int a, int b, int i, int j);
```
Funkcja przestawiająca bierkę i sprawdzająca czy po nim dalej występuje szach. Po wszystkim bierki są przestawiane na wcześniejsze pozycje. Zwraca wartość **True**, bądź **False.**

```
Bool Is_it_chekmate_for_one_piece(Board_struct** Board, Pieces colour_of_piece,
Pieces colour_of_King, Board_struct last_enemy_move, Player turn);
```
Funkcja sprawdzająca czy dana bierka jest powodem szach – mata. Jeśli tak, zwraca zwraca wartość **True**, w przeciwnym wypadku **False**

```
Bool Is_it_checkmate_main_function(Board_struct** Board, Pieces colour_of_King,
Pieces colour_of_piece, Board_struct last_enemy_move, Player turn);
```
Funkcja sprawdzająca czy występuje szach – mat na danym królu. Zwraca wartość **True**, bądź **False.**

```
Player checks_support(Bool *is_it_correct, Bool *possible_move, Bool
*black_King_threatened, Bool *white_King_threatened, int *oldX, int *oldY,
int *x, int *y, Pieces *Which_one, Board_struct *last_move,
Board_struct **Board, char* info, Pieces *What_was_there,
Player *turn, Pieces colour_of_King, int *Which_function,
Bool start_from_file);
```
Funkcja sterująca grą w stanie zagrożenia któregoś z królów. Zwraca kolor bierek zwycięzcy rozgrywki.

```
Player support_for_condition_from_check(Bool *black_King_threatened, Bool
*is_it_correct, Bool *possible_move, Bool *white_King_threatened, int *oldX,
int *oldY, int *x, int *y, Pieces *Which_one, Board_struct *last_move,
Board_struct** Board, char* info, Pieces *What_was_there, Player *turn,
int *Which_function, Bool start_from_file);
```
Funkcja sprawdzająca czy występuje szach – mat dla któregokolwiek z królów, zwraca kolor bierek zwycięzcy (o ile nastąpił szach – mat)

h) wsparcie dla możliwych posunięć bierek

```
Bool first_condition_for_move(Pieces Is_it_empty, Pieces colour);
```
Funkcja sprawdzająca warunek czy bierka nie przemieści się na pole, gdzie znajduje się bierka własnego koloru. Zwraca wartość **True**, bądź **False.**

```
void differences_for_move(int array_diff[4], int letter_move_to, int
number_move_to, int number_move_from, int letter_move_from);
```
Funkcja wypełniająca tablicę wartościami różnic pomiędzy literami i liczbami pomiędzy polami startowym i docelowym dla ruchu bierki. Argumenty: czteroelementowa tablica, litera i liczba pola, na które ma poruszyć się bierka, liczba i litera pola, z którego bierka się porusza.

```
Bool Check_is_it_empty_between_squares_Rook(Orientation orient, int difference,
int sign_of_difference, int number_move_from, int letter_move_from,
Board_struct** Board);
```
Funkcja sprawdzająca czy pomiędzy polem startowym, a docelowym pionowo lub poziomo wszystkie pola są puste. Argumenty: zmienna sygnalizująca czy sprawdzane pola będą w orientacji poziomej czy pionowej, litera i liczba pola, na które ma poruszyć się bierka, liczba i litera pola, z którego bierka się porusza.

```
Bool Check_is_it_empty_between_squares_Bishop(int difference_horizontally,
int sign_of_difference_horizontally, int sign_of_difference_upright,
int number_move_from, int letter_move_from, Board_struct** Board);
```
Funkcja sprawdzająca czy pomiędzy polem startowym, a docelowym w skosie wszystkie pola są puste Argumenty: różnica pól w poziomie, znak różnicy pól w poziomie, znak różnicy pól w pionie, liczba i litera pola z którego goniec, bądź królowa się porusza, wskaźnik na szachownicę. Zwraca odpowiedź typu logicznego czy pomiędzy polami wszystkie pola są puste czyli **True**, bądź **False.**

```
Bool Main_funct_is_it_empty_Rook(Board_struct** Board, int letter_move_to,
int number_move_to, int number_move_from, int letter_move_from);
```
Funkcja tworząca czteroelementową tablicę, która potem wypełniana jest przez funkcję **differences_for_move**, następnie uruchamia funkcje sprawdzające czy pomiędzy polem startowym a docelowym dla wieży, bądź królowej wszystkie pola są puste. Zwraca wartość **True**, bądź **False.**

```
Bool Main_funct_is_it_empty_Bishop(int letter_move_to, int number_move_to,
int number_move_from, int letter_move_from, Board_struct** Board);
```
Funkcja tworząca czteroelementową tablicę, która potem wypełniana jest przez funkcję **differences_for_move**, następnie uruchamia funkcje sprawdzające czy pomiędzy polem startowym a docelowym dla gońca, bądź królowej wszystkie pola są puste. Zwraca wartość **True**, bądź **False.**

i) pozostałe funkcje

```
void timer(int sekundy);
```
Funkcja odliczająca zadany czas podany w argumencie w sekundach.

```
void choose_game(char *choose);
```
Funkcja odpowiedzialna za pobranie z klawiatury zmiennej, za pomocą której gracz określa, z jakiego stanu gry chce rozpocząć grę. Przyjmuje jako argument wskaźnik na zmienną, która będzie wczytywana z klawiatury.

```
void switch_game(char choose);
```
Funkcja przełączająca w zależności od wartości zmiennej przekazanej jako argument do dwóch różnych stanów gry: początek rozgrywki, bądź stan wczytany z pliku.

5. Testowanie

Program był testowany wielokrotnie przez wielu użytkowników, nie stwierdzili problemów z obsługą programu. Nie powinny w nim występować wycieki pamięci – szachownica stworzona dynamicznie jest usuwana za pomocą odpowiednich komend.

6. Wnioski

Program wydaje się być dosyć skomplikowanym z powodu dużej ilości warunków potrzebnych do tego, by gra spełniała postawione przed nią wymagania. Podczas wykonywania projektu miałem możliwość przećwiczenia swoich umiejętności programistycznych i poszukiwania różnych rozwiązań danych problemów.

Link do github'a - https://github.com/dmusial98/Repository1

Literatura

- Jerzy Grębosz „Opus Magnum C++11" wyd. Helion Gliwice 2017
- http://cpp0x.pl
- https://www.youtube.com/watch?v=rMKHLz3liuk&feature=share