Gliwice 19.01.2019r.

Laboratorium z przedmiotu Programowanie Komputerów 3

Sprawozdanie z Projektu - Warcaby

AEI Informatyka gr.2

Michał Nalepa

**1. Opis projektu**

Celem projektu było stworzenie gry w warcaby w języku C++, która korzystałaby z obiektowych możliwości języka. Projekt miał zawierać co najmniej 5 powiązanych klas, dziedziczenie, przeciążone operatory w tym strumieniowe, zapis/odczyt z pliku oraz korzystać z pamięci dynamicznej. Gra została napisana w oparciu o następujące zasady:

Gra warcaby klasyczne jest rozgrywana na planszy 8x8 pól pokolorowanych na przemian na kolor jasny i ciemny.

Każdy gracz rozpoczyna grę z dwunastoma pionami (jeden koloru białego, drugi – czarnego) ustawionymi na ciemniejszych polach planszy.

Jako pierwszy ruch wykonuje grający pionami białymi, po czym gracze wykonują na zmianę kolejne ruchy.

Celem gry jest zbicie wszystkich pionów przeciwnika (w tym damek) albo zablokowanie wszystkich, które pozostają na planszy, pozbawiając przeciwnika możliwości wykonania ruchu.

Piony mogą poruszać się o jedno pole do przodu po przekątnej na wolne pola.

Bicie pionem następuje przez przeskoczenie sąsiedniego pionu (lub damki) przeciwnika na pole znajdujące się tuż za nim po przekątnej (pole to musi być wolne). Zbite piony są usuwane z planszy po zakończeniu ruchu.

Piony mogą bić zarówno do przodu, jak i do tyłu.

W jednym ruchu wolno wykonać więcej niż jedno bicie tym samym pionem, przeskakując przez kolejne piony lub damki przeciwnika.

Bicia są obowiązkowe.

Pion, który dojdzie do ostatniego rzędu planszy, staje się damką, przy czym jeśli w jednym ruchu w wyniku wielokrotnego bicia przejdzie przez ostatni rząd, ale nie zakończy na nim ruchu, to nie staje się damką i kończy ruch jako pionek.

Kiedy pion staje się damką, kolejny ruch przypada dla przeciwnika.

Damki mogą poruszać się w jednym ruchu o dowolną liczbę pól do przodu lub do tyłu po przekątnej, zatrzymując się na wolnych polach.

Bicie damką jest możliwe z dowolnej odległości po linii przekątnej i następuje przez przeskoczenie pionu (lub damki) przeciwnika, za którym musi znajdować się co najmniej jedno wolne pole – damka przeskakuje na dowolne z tych pól i może kontynuować bicie na tej samej lub prostopadłej linii.

Podczas bicia nie można przeskakiwać więcej niż jeden raz przez ten sam pion lub damkę.

*Źródło z zasadami: https://pl.wikipedia.org/wiki/Warcaby*

Gra jest skierowana do dwóch graczy którzy na przemian wykonują swoje ruchy. Zaczyna gracz posiadający pionki białe. Wszystkie zasady oraz poprawność wykonywanych ruchów jest sprawdzana na bieżąco przez program. Grafika w projekcie oparta jest na bibliotece graficznej SFML w wersji 2.5.1.

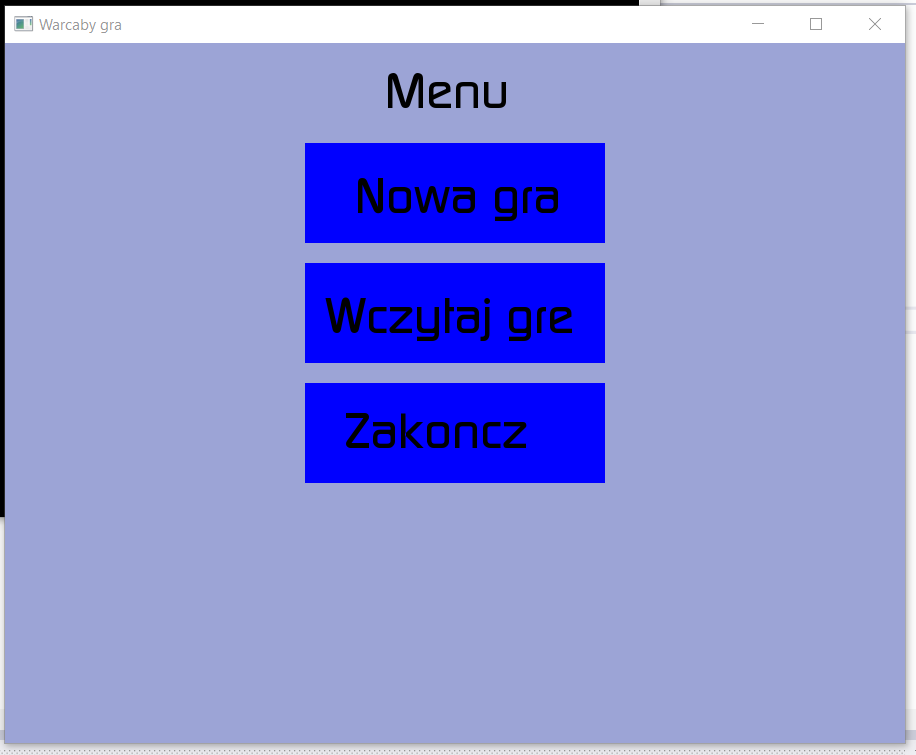
**2. Zaproponowane algorytmy**

W programie zastosowano szereg algorytmów matematycznych które służą m.in. do tego aby sprawdzać poprawność wykonanego ruchu, czy wykonano wszystkie możliwe bicia, czy nie zrobiono ruchu gdy było możliwe bicie, czy nie ma blokady ruchów itp. (ich szczegółowy opis w punkcie 4).

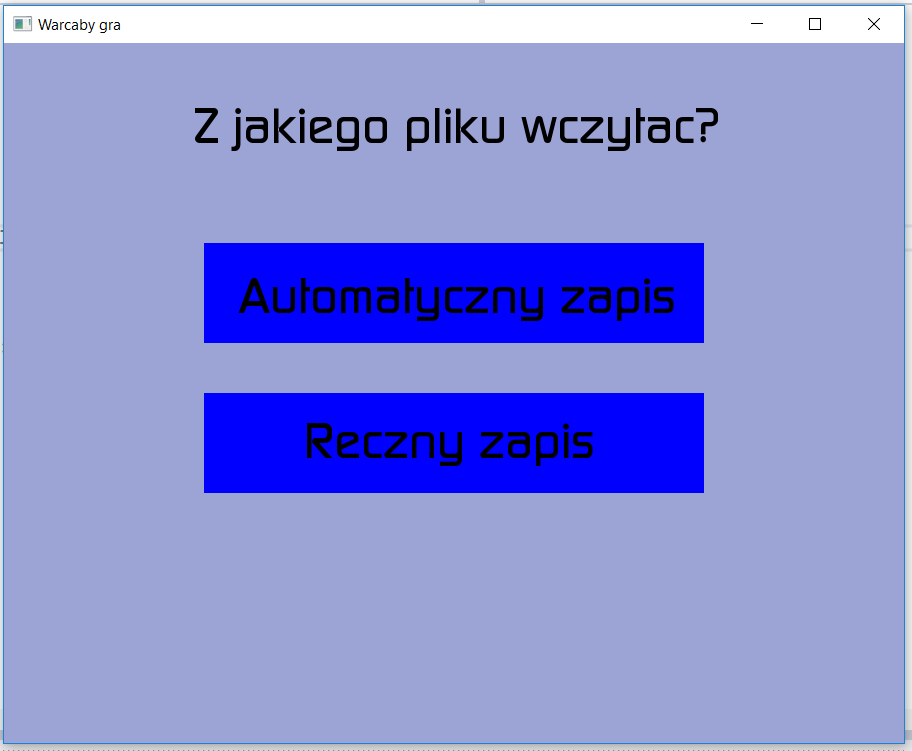
**3. Instrukcja obsługi**

Menu:

Po uruchomieniu programu pojawi się ekran Menu z trzema opcjami do wyboru. Aby wybrać wybraną opcję należy kliknąć w odpowiednią kafelkę z podpisem. Po najechaniu na odpowiednie okno kursorem zmieni one swój kolor na zielony.

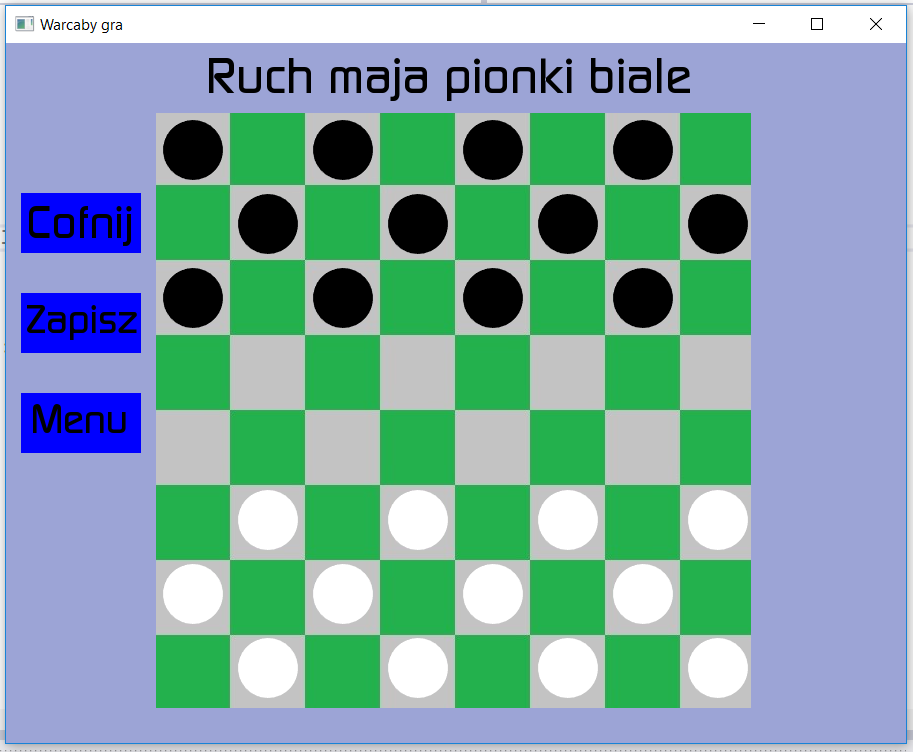


Opcja „Nowa Gra” powoduje stworzenie nowej gry z domyślnym ustawieniem pionków na planszy. Jeżeli chcemy wczytać zapisaną wcześniej grę należy wybrać opcję „Wczytaj grę”. Opcja „Zakończ” zamyka program. Po wybraniu opcji „Wczytaj grę” pojawi się podmenu służące do wyboru z jakiego źródła gra powinna być wczytana. Sposób wyboru odpowiedniej opcji jest analogiczny do głównego Menu. Zrzut ekranu przedstawiający omawiane podmenu:



Opcja „Automatyczny zapis” powoduje wczytanie stanu gry z pliku ”auto\_save\_data.txt”. Gra zapisuje się automatycznie co 30s (pojawia się wtedy informujący o tym komunikat). Analogicznie opcja „Ręczny zapis” powoduje wczytanie stanu gry z pliku ”save\_data.txt” w którym znajduje się zapis użytkownika.

Pole gry:



3

2

1

Opis pola gry

1. Na samej górze znajduje się komunikat informujący o tym który gracz ma ruch, a w przypadku wygranej informuje kto wygrywa i w jaki sposób (zbicie wszystkich pionków lub blokada).

2. Po lewej stronie znajdują się trzy przyciski służące do:

- cofania ruchu, po naciśnięciu cofany jest dokładnie jeden ruch. Jeżeli na ten ruch składało się wiele bić to wszystkie zostaną cofnięte. Jeżeli przycisk cofnij zostanie naciśnięty **w trakcie** bicia czyli ruch nie jest kompletny zostanie zapisany stan gry bez wykonywanego ruchu,

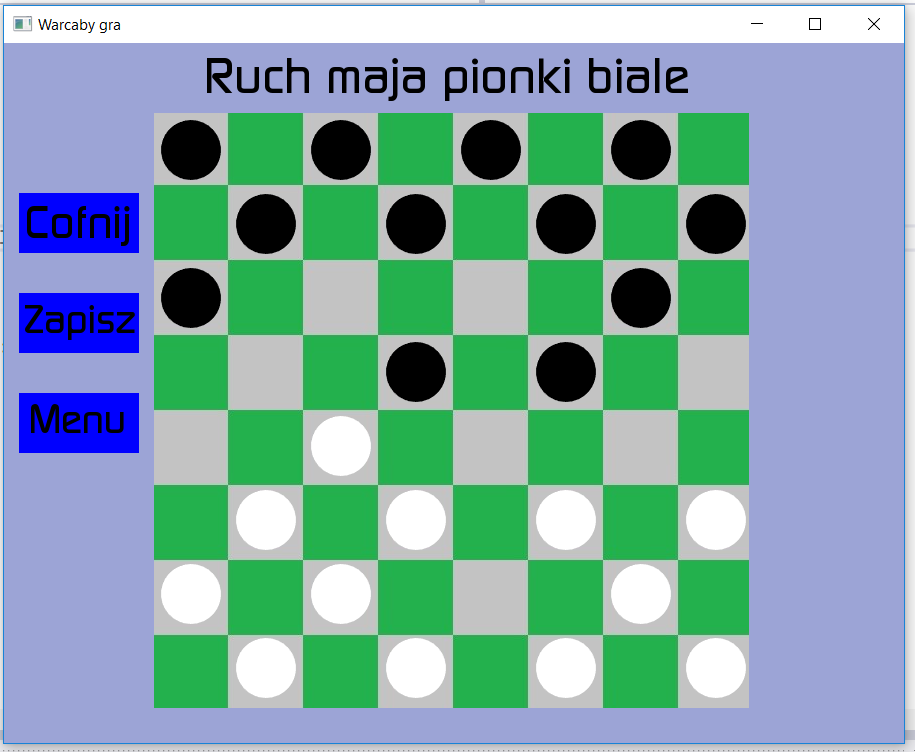
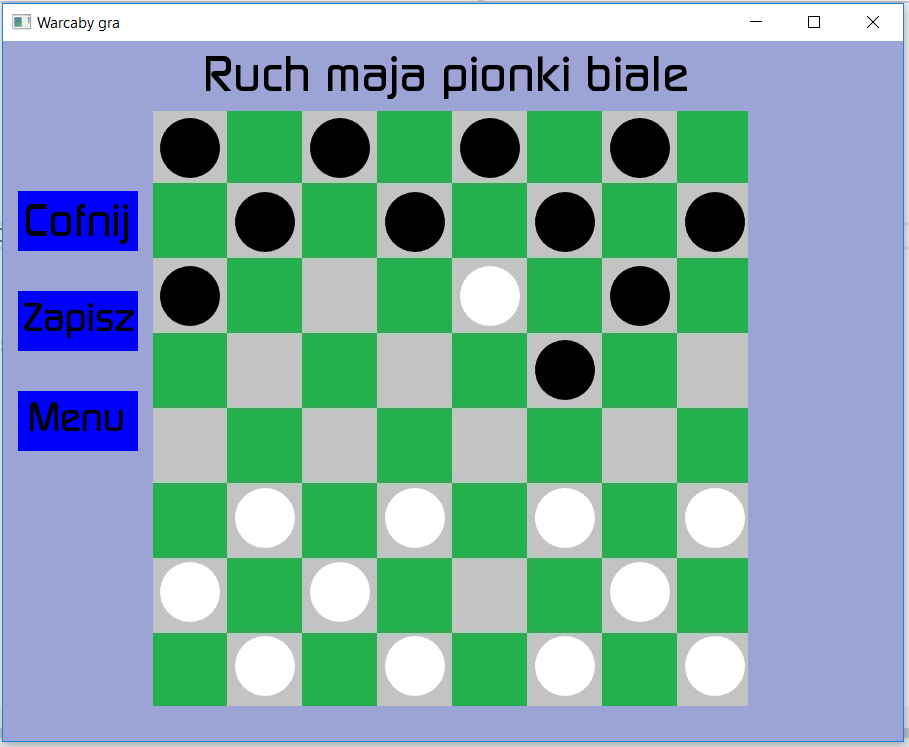
- ręcznego zapisu stanu gry do pliku „save\_data.txt”,

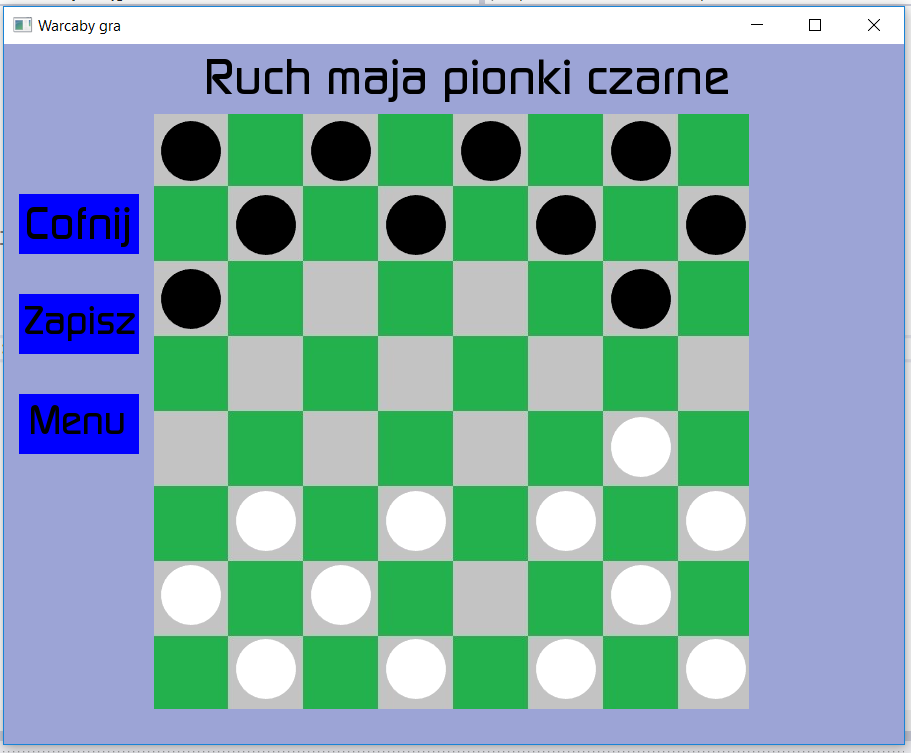
- powrotu do menu.

3. W centralnym polu znajduje się plansza wraz z pionkami. Pionkami są czarne lub białe kółka, a damki posiadają dodatkową żółtą obwódkę.

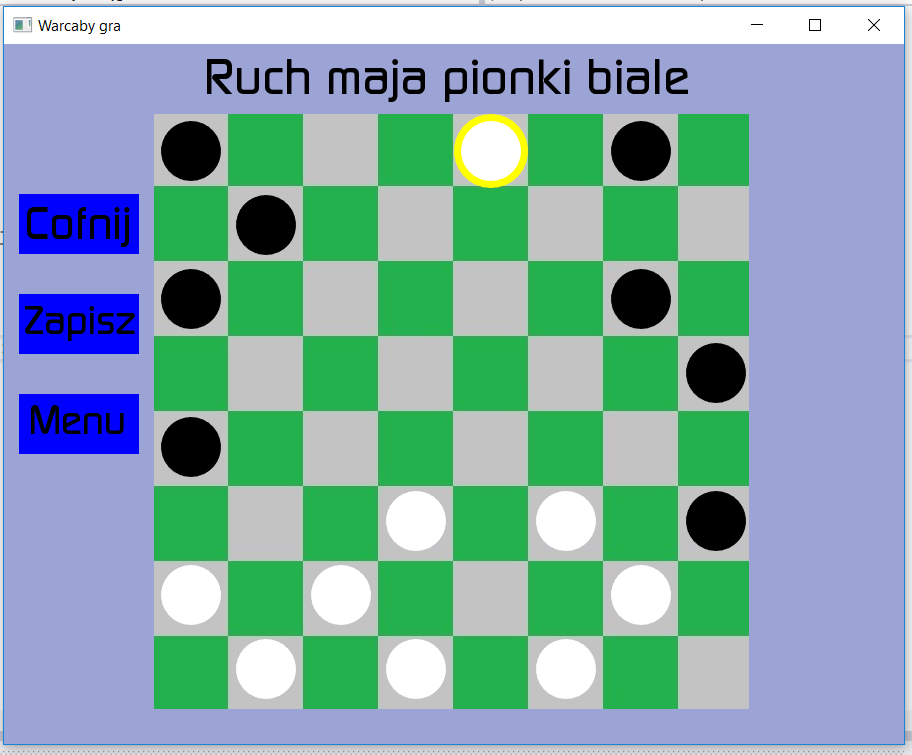
Jak grać?

Aby wykonać ruch należy kliknąć na dany pionek i cały czas trzymając naciśnięty lewy przycisk myszy przeciągnąć go na pole na które chcemy go przenieść. Poprawność wykonywanych ruchów jest na bieżąco sprawdzana przez program. W przypadku, gdy wykonany ruch nie było poprawny, pionek wraca na pole na którym poprzednio stał, a ruch nie jest zaliczany. W takim przypadku gracz na możliwość wykonania ruchu ponownie. Bicie pionka polega na przeciągnięciu tego pionka na miejsce na którym ma się znaleźć po zbiciu innego. W przypadku gdy możliwe jest bicie wielu pionków należy je wykonywać pojedynczo, a gra przechodzi w tzw. tryb bicia w którym ruch ma cały czas ten sam gracz aż do wykonania wszystkich możliwych bić. Przykład ilustrujący taką sytuacje poniżej:

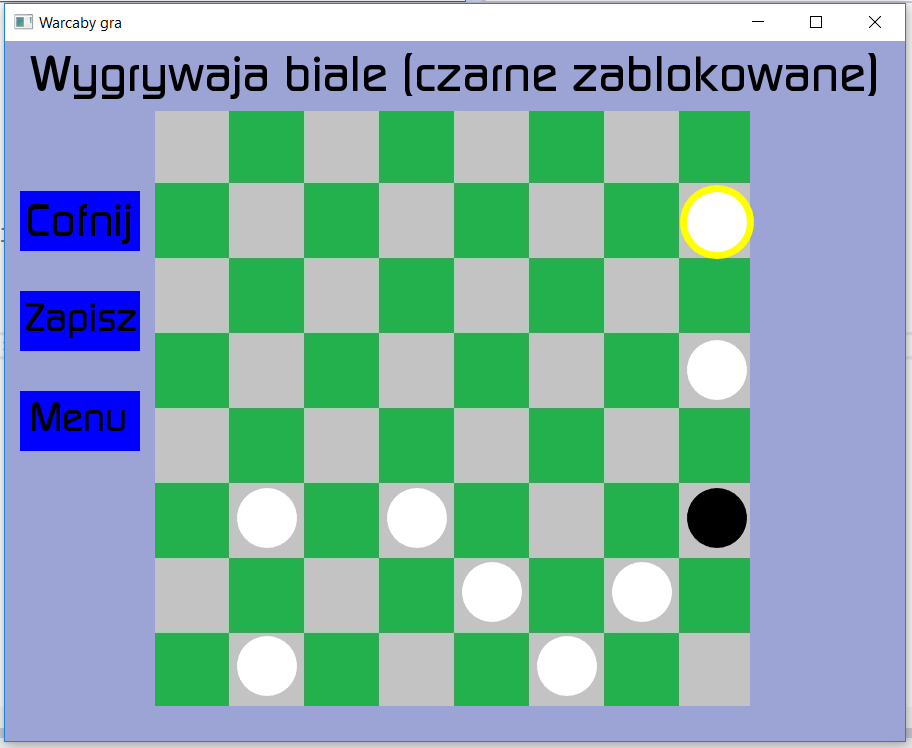




W przypadku gdy pionek stanie na ostatnie pole znajdujące po przeciwnej stronie (oraz nie jest w „trybie bicia”) zamienia się na damkę który wygląda następująco:



W przypadku wygranej jednego z graczy pojawia się odpowiedni komunikat który mówi kto wygrał i w jaki sposób:



**2. Opis klas oraz ich elementów**

**1. Klasa Menu**

Jest to klasa odpowiedzialna za wyświetlanie jak i obsługę głównego menu wraz z podmenu służącego do wyboru z jakiego pliku należy wczytać grę. Ma ona na celu zebrać od użytkownika dane potrzebne do uruchomienia nowej gry. Z racji tego że napisany program zawiera grafikę, a klasa menu jest ściśle graficzna, większość zmiennych składowych klasy pochodzi z biblioteki graficznej SFML. Są to:

- sf::Font font; - przechowuje czcionkę wykorzystywaną przez program, czcionka ta jest wczytywana z pliku „contm.ttf” w konstruktorze.

- sf::RenderWindow game\_window; - zmienna ta jest oknem na którym jest rysowane jest menu główne, podmenu jak i gra. Są na nim wykonywane wszystkie operacje przez użytkownika, jest graficznym interfejsem gry. Tworzona w konstruktorze z argumentem „sf::ContextSettings settings” który służy do ustawienia stopnia antialiasingu na 8 co znacznie wyostrzyło obraz w grze i przez co stał się on bardziej estetyczny.

- sf::Text text[4]; - tablica przechowująca 4 teksty wyświetlające się w Menu.

- sf::RectangleShape rectangle[3]; - tablica przechowująca 4 prostokąty wyświetlające się w Menu pełniące rolę przycisków.

- int choice; - zmienna przechowujący wybór użytkownika w menu głównym jak i podmenu. Zmienna ta jest przekazywana do klasy Game.

Klasa posiada 4 metody na które składają się:

Menu(sf::ContextSettings settings); - konstruktor wczytujący wszystkie zmienne i ustawiający ich prawidłowe wartości.

void main\_menu(); - funkcja wyświetlająca główne menu, pozwalająca użytkownikowi na wybranie jednej z 3 opcji: Nowa gra, Wczytaj grę, Zakończ. Po wybraniu odpowiedniej opcji albo przechodzi się do podmenu albo tworzy obiekt klasy Game czyli nową grę której przekazuje się jaką opcję wybrał użytkownik.

void loading\_menu(); - funkcja wyświetlająca podmenu, pozwalająca użytkownikowi na wybranie jednej z 2 opcji: Automatyczny Zapis lub Ręczny Zapis. Jej działanie jest analogiczne do metody main\_menu().

~Menu(); - destruktor

**2. Klasa Game**

Klasa ta służy do obsługi i wyświetlania aktualnie rozgrywanej gry. Jest interfejsem służącym do komunikacji z użytkownikiem. To niej gracz wykonuje wszystkie swoje operacje. Z racji tego że napisany program zawiera grafikę, a klasa Game jest w sporym zakresie graficzna, wiele zmiennych składowych klasy pochodzi z biblioteki graficznej SFML. Są to:

- sf::Font font;- podobnie jak dla klasy Menu przechowuje czcionkę,

- sf::Text text[5]; - podobnie jak dla klasy Menu tablica ta przechowuje 5 napisów,

- sf::RectangleShape rectangle[3];- podobnie jak dla klasy Menu tablica ta przechowuje 3 prostokąty będące przyciskami służącymi do cofania ruchu, zapisu i powrotu do menu,

- sf::Event game\_event; - zmienna przechowująca czynności wykonane przez użytkownika takie jak np. zamknięcie okna gry,

- sf::Vector2i mouse\_position; - zmienna ta przechowuje położenie myszki na ekranie,

- bool is\_move; int dx, dy; - zmienne pomocnicze dzięki który jest możliwe przeciąganie pionków w czasie rzeczywistym

- const int size = 8; - zmienna przechowująca długość tablicy będącej planszą na której rozgrywana jest gra

bool during\_the\_beating; - zmienna przechowująca informacje czy gra jest w tzw. ”trybie bicia czy nie”

List\_Item\* Ptr; - wskaźnik na element listy na którym aktualnie operuje gra. Wszystkie stany tablicy planszy zapisywane są na liście dwukierunkowej, aby możliwe było cofanie ruchów.

field\_index beating\_pawn, first\_index; - proste struktury przechowujące indeksy pól planszy. Beating\_pawn przechowuje indeks pionka który aktualnie jest w trybie bicia, a first\_index indeks pionka którym gracz poruszył.

Klasa posiada 9 metod na które składają się:

Game(); - bezargumentowy konstruktor wczytujący wszystkie zmienne i ustawiający ich prawidłowe wartości.

void add\_new\_game(); - metoda jest uruchamiana w przypadku, gdy gracz wybrał nową grę. Uruchamia ona metodę klasy Board która ustawia domyślne początkowe położenie pionków na planszy.

void load\_game(const std::string &file\_name); - metoda ta jest uruchamiana w przypadku, gdy gracz wybrał wczytanie gry. Tworzy ona nowy obiekt klasy Game\_Loading oraz uruchamia metodę Load() z tej klasy, która to zwraca element listy (obiekt klasy List\_item) z wczytaną tablicą zawierającą informacje o położeniu pionków na planszy. Ten element listy jest przypisywany za pomocą przeciążonego operatora przypisania klasy List\_item do Heada klasy Game.

void play(sf::RenderWindow &game\_window, const int &choice); - metoda ta przyjmująca jako argumenty okno z menu oraz wybór dokonany przez użytkownika jest graficznym interfejsem gry. To w niej wszystkie elementy są rysowane na ekranie, a gracz podejmuje szereg decyzji. Zawiera ona obsługę przycisków w głównej grze, obsługę przenoszenia pionków przez gracza w czasie rzeczywistym i uruchamia inne metody służące m.in. do sprawdzania poprawności ruchów czy stanu gry.

void movement\_support(const field\_index &first\_index, const field\_index &second\_index); - metoda ta uruchamia metody z innych klas służące do sprawdzenia poprawności wykonywanych ruchów. W przypadku gdy jest poprawny tworzy nowy element listy będący kolejnym ruchem, a jeżeli nie plansza wraca do poprzedniego ułożenia. Uruchamia także metodę check\_clashing\_posibility() klasy Board które przypisuje do każdego pionka informacje czy może on dokonać bicia czy też nie, metodę statement\_change() zmieniającą komunikat, oraz check\_result() która sprawdza czy gra się zakończyła.

void statement\_change() – prosta metoda zmieniająca komunikat o tym kto ma aktualnie ruch

void check\_result() – metoda sprawdzający czy jeden z graczy nie wygrał partii gry w warcaby. Sprawdza dwie możliwości wygranej:

- przeciwnik nie ma już pionków

- pionki przeciwnika są zablokowane

void Game::check\_pawn\_to\_queen\_transformation(const field\_index &temp\_index) –

metoda sprawdzająca czy jest możliwa zamiana pionka w damę, jeżeli tak wywołuję metodę pawn\_to\_queen\_transformation() klasy Pawn

Game::~Game() – destruktor uruchamiany przy zamykaniu gry czyszczący całą pamięć

W tym samym pliku znajdują się też prosta przeciążona funkcja globalna index. W programie korzystam z tablicy 1 wymiarowej tak jak z tablicy 2 wymiarowej w celu zaoszczędzenia pamięci. Funkcja index na podstawie przekazywanych indeksów tablicy dwuwymiarowej zwraca indeks tablicy jednowymiarowej.

Klasy Menu jak i Game jako jedyne zawierają tak wiele elementów graficznych biblioteki SFML. W pozostałych klasach te elementy nie zawierają się wcale lub występują bardzo rzadko. Dzięki temu zamiana biblioteki graficznej na inną byłaby dosyć prosta i możliwa do wykonania bez ingerencji w pozostałe klasy.

**3. Klasa List\_Item**

Klasa ta jest pojedynczym elementem listy dwukierunkowej zawierającym jeden stan gry. Z tego powodu posiada wskaźniki na poprzedni List\_Item\* Previous i następny element listy List\_Item\* Next. colour\_of\_pawn next\_move\_colour przechowuje informację jakiego koloru pionek ma aktualnie ruch. Obiekt klasy Board Actual\_Board przechowuje cała planszę z ułożeniem pionków. Klasa posiada 4 metody:

- List\_Item(); - konstruktor bezargumentowy

- void add\_board(); - metoda ta tworzy nowy element listy i zapisuje do niej tablice z aktualnej (na tej aktualnej tablicy gracz będzie przeprowadzał operacje)

- void delete\_last\_board() – metoda ta usuwa ostatni element listy

- List\_Item & operator =(const List\_Item list\_item1) – przeciążony operator przypisania pozwalający przypisywać wszystkie składowe obiektu klasy List\_Item

- ~List\_Item() – destruktor

**4. Klasa Board**

Jest to zdecydowanie największa i najbardziej rozbudowana klasa. Reprezentuje ona planszę do gry. Z racji tego że plansza składa się z pół zawiera ona jednowymiarową tablicę obiektów pole Field\* Tab alokowaną w sposób dynamiczny. Jej domyślny rozmiar jest zawarty w zmiennej const int size = 8, a liczba pionków w const int number\_of\_pawns = 24.Pozostałymi składowymi tej klasy są sf::Texture texture oraz sf::Sprite checkers\_board które przechowują grafikę z planszą. Z racji tego że aby sprawdzać m.in. poprawność wykonywanych ruchów potrzebujemy informacji o ułożeniu figur na całej planszy większość metod odpowiedzialnych na logikę zawiera się w tej klasie. Klasa posiada 10 metod:

- Board(std::string file); – konstruktor wczytujący grafikę z planszą, alokujący nową tablice obiektów pole (Field), ustawiający dla każdego pola jego współrzędne x1, x2, y1, y2 oraz domyślne współrzędne ułożenia pionka (współrzędne pola na których figury mają być rysowane)

- void initial\_positions(); - metoda ta tworzy określoną liczbę pionków i ustawia je na planszy w sposób domyślny. Aby to zrobić wywołuje metodę add\_pawn() klasy Field dla odpowiednich pól.

- field\_index check\_mouse\_position(const int &mouse\_position\_x, const int &mouse\_position\_y) – metoda ta na podstawie współrzędnych myszki zwraca indeks pola planszy na którym znajdują się owe współrzędne.

- bool check\_move(const field\_index &first\_index, const field\_index &second\_index, const colour\_of\_pawn &colour\_pom, bool &during\_the\_beating, field\_index &beating\_pawn); - jest to pierwsza metoda sprawdzająca czy wykonany ruch jest poprawny. Jeżeli wykonywany jest zwykły ruch bez bicia metoda ta wykonuje ten ruch na danej planszy. Jeżeli natomiast wykonywany ruch to bicie metoda ta wywołuje check\_move2() odpowiedzialną za bicie.

- bool Board::check\_move\_2(const field\_index &first\_index, const field\_index &second\_index, const colour\_of\_pawn &move\_colour) – metoda ta sprawdza czy wykonywane bicie jej poprawne i jeżeli tak to je wykonuje na danej planszy.

- void Display(); - pomocnicza metoda wyświetlająca planszę i jej zawartość wykorzystywana podczas projektowania gry

- bool check\_clashing\_posibility(const field\_index &second\_index);

- metoda ta zwraca wartość logiczną true jeżeli figura której indeks został przekazany może dokonać bicia innej figury lub false jeżeli nie może

- void check\_clashing\_posibility(); - metoda ta wywołuje wymienioną wyżej metodę dla każdego obiektu klasy Pawn na planszy i zapisuje do niej informację o tym czy może dokonać bicia czy nie

- bool Board::check\_move\_posibility(const field\_index &temp\_index)

- metoda ta zwraca wartość logiczną true jeżeli figura której indeks został przekazany może dokonać ruchu na inne pole bez bicia lub false jeżeli nie może. Wykorzystywana jest tylko podczas sprawdzania czy nie wystąpiła blokada gracza

- Board & Board::operator =(const Board& board1) – przeciążony operator przypisania służący do przypisania wartości obiektu klasy Board do innego obiektu tej samej klasy

- Board::~Board() - destruktor

**5. Klasa Field**

Obiekty tej klasy reprezentują pojedyncze pole na planszy. Z tego powodu jej zmienne przechowują współrzędne tego pola int field\_position\_x1, field\_position\_x2, field\_position\_y1, field\_position\_y2; a także współrzędne domyślne na których mają być rysowane figury znajdujące się na tym polu. Na polu może stać pionek lub też nie co przedstawia wskaźnik na obiekt klasy Pawn\* Pawn\_ptr. Jeżeli pole jest puste wskaźnik jest nullptr, jeżeli znajduje się na nim figura wskazuje na obiekt klasy Pawn. Jest to bardzo wygodne gdyż w przypadku gdy figura zmieni pole na którym się znajduje wystarczy tylko przepiąć wskaźnik na nią wskazujący na inne pole. Klasa posiada 7 metod:

- Field(); - konstruktor bezargumentowy

- Field & operator=(const Field &field1); - przeciążony operator przypisania służący do przypisania wartości obiektu klasy Field do innego obiektu tej samej klasy

- void add\_pawn(type\_of\_pawn type\_pom, colour\_of\_pawn colour\_pom);

metoda ta tworzy nowy pionek na polu o parametrach przekazanych w argumentach

- void set\_field\_position(int x1, int x2, int y1, int y2);

- void set\_default\_circle\_position(float x1, float x2);

Proste metody ustawiające watości zmiennych składowych

- void clash(); - metoda likwidująca figurę na polu i ustawiająca wskaźnik z powrotem na nullptr

- ~Field(); - destruktor

**6. Klasa Pawn**

Obiekty tej klasy reprezentują figury znajdujące się na planszy. Pojedyncza figura może być pionkiem lub damą, o kolorze białym lub czarnym. Informacje te przechowują zmienne składowe type\_of\_pawn type oraz colour\_of\_pawn colour które są prostymi zmiennymi typu wyliczeniowego:

enum type\_of\_pawn {pawn, queen};

enum colour\_of\_pawn {white, black};

Zdaje sobie sprawę że w tym przypadku odpowiedni byłby także typ bool aczkolwiek dla większej czytelności kodu zdecydowałem się na samodzielne zdefiniowanie typu wyliczeniowego. Zmienna składowa sf::CircleShape Circle z biblioteki SFML jest graficznym odzwierciedleniem obiektu klasy Pawn który jest rysowany na ekranie. Zmienna bool clashing przyjmuje wartość true jeżeli figura może dokonać bicia lub false w przeciwnym przypadku. Jest używana podczas sprawdzania ruchów. Klasa posiada 14 metod jednakże większość z nich to metody get i set pobierające i ustawiające wartości zmiennych składowych(także obieku Circle). Z pozostałych warte dokładniejszego omówienia są metody:

- void Pawn::draw(sf::RenderWindow &game\_window) – metoda ta rysuje Circle na przekazanym oknie. Pozycja koła do rysowania, jego kolor itp. jest przypisana do obiektu Circle.

- bool Pawn::Circle\_getGlobalBounds\_contains(const int &mouse\_position\_x, const int& mouse\_position\_y) – metoda ta zwraca wartość logiczną true gdy współrzędne położenia myszki znajdują na figurze lub false w przeciwnym przypadku.

- void Pawn::pawn\_to\_queen\_transformation() – metoda ta zmienia typ figury z pionka na damę.

**7. Klasa Game\_Saving**

Klasa ta służy do zapisu aktualnego stanu gry. Z tego powodu zawiera obiekt klasy std::ofstream file, nazwę pliku do którego ma być zapisana gra std::string file\_name oraz wskaźnik na element listy List\_Item\* Ptr który ma być zapisany do pliku. Klasa posiada 4 metody:

- Game\_Saving(); - bezargumentowy konstruktor

- Game\_Saving(std::string file\_name, List\_Item\* Head); - konstruktor dwuargumentowy

- void Save(); - metodę która zapisuje element listy do pliku z wykorzystaniem przeciążonego operatora przypisania

- ~Game\_Saving(); - destruktor

W pliku z tą klasą znajduje się także globalny przeciążony operator strumieniowy <<

std::ostream & operator <<(std::ostream & output, const List\_Item \*Ptr)

**8. Klasa Automatic\_Game\_Saving**

Klasa ta dziedziczy publicznie po klasie Game\_Saving dodając do niej obsługę czasu. Obsługa czasu oparta jest o bibliotekę SFML. Zatem posiada obiekt sf::Clock clock z którego pobiera się aktualny czas oraz sf::Time time służący do przechowywania pobranego czasu. Zmienna const float time\_period1 = 30 jest odstępem czasu co jaki ma się zapisywać gra, a zmienna const float time\_period2 = 2 czasem przez jaki ma się wyświetlać komunikat że gra została zapisana. Obie wartości podane w sekundach. bool statement\_time mówi nam czy aktualnie jest wyświetlany komunikat. Klasa posiada 4 metody:

- Automatic\_Game\_Saving(); - konstruktor bezargumentowy

- void set(std::string file\_name1, List\_Item\* Ptr1) - metoda ustawiająca przekazane wartości

- bool check\_time() – metoda obsługująca czas, głównie odliczająca dane odstępy czasu

- bool get\_statement\_time() - metoda zwracająca wartość zmiennej statement\_time

- ~Automatic\_Game\_Saving() – destruktor

**9. Klasa Game\_Loading**

Klasa jest odpowiedzialna za odczytanie danych z pliku tekstowego i utworzenia na ich podstawie odpowiedniego stanu gry, ułożenia figur na planszy i kolejności ruchu. Klasa zawiera dwie zmienne składowe: std::string file\_name z nazwą pliku tekstowego txt z danymi do wczytania oraz std::ifstream file służącą do obsługi wejścia plikowego. Klasa zawiera 3 metody:

- Game\_Loading(std::string file\_name); - konstruktor

- List\_Item\* Load(); - metoda ta dokonuje procesu wczytania stanu gry z pliku

- ~Game\_Loading(); - destruktor

W pliku z tą klasą znajduje się także globalny przeciążony operator strumieniowy >>

std::istream & operator >>(std::istream & input, List\_Item \*Ptr);

**5. Diagram klas**

Diagram klas w modelu UML znajduje się pod adresem: <https://go.gliffy.com/go/share/s6jeo29ne44c6u9jnc85>

**6. Projekt**

Projekt wraz ze wszystkimi plikami źródłowymi, txt, jpg niezbędnymi do prawidłowego działania programu znajduje się pod adresem:

**7. Wnioski**