Laboratorium
Programowania Komputerów
Temat:
Gra w statki

Autor: Filip Gaida
Informatyka, sem.2, grupa 3, sekcja
Prowadzący:Mgr Inż. Grzegorz Kwiatkowski
Ścieżka:

1.Temat

Napisać konsolową grę w statki. Gra powinna mieć możliwość cofania ruchu, rozgrywki w dwie osoby i z komputerem.

2.Analiza, projektowanie

2.1 Algorytmy

Głównym algorytmem będzie sprawdzanie poprawności ruchu gracza. Algorytm powinien wykonywać jak najmniej zbędnych operacji, ale także nie powinien zostawiać błędów po nieudanym ruchu.

Algorytm cofania ruchu będzie rekurencyjnie przechodził po dynamicznej liście, odczytywał ruch gracza, cofał jego efekty i wymagał ponownego wykonania cofniętego ruchu. Rekurencyjne wykonanie powinno pozwolić na cofnięcie dowolnej ilości ruchów i pozwolić na ich wykonanie w pierwotnej kolejności.

Gra z komputerem powinna stosować algorytm celowania przez komputer. W grze w statki umiejętność analitycznego wyboru miejsca jest potrzebna dopiero pod koniec gry, dlatego algorytm będzie ustawiał statki losowo. Pierwsze ruchy również będą losowe, dopiero po trafieniu komputer będzie poszukiwał reszty statku tak długo, aż całego nie zatopi.

2.2 Struktury danych

Plansza, na której gracze będą umieszczać statki, powinna być dwu-wymiarową tablicą liczb całkowitych. Ułatwi to możliwość modyfikacji w przypadku potrzeby rozszerzenia możliwości aplikacji oraz czytelność wykonywanych operacji w funkcjach zmieniające zawartość tej tablicy.

Liczba pozostałych statków dla gracza powinna być liczbą całkowitą umieszczoną w jednej strukturze z planszą danego gracza. Zapobiegnie to przypadkowym błędom i ułatwi przekazywanie danych do funkcji.

Cofanie ruchów będzie wymagać dynamicznie alokowanej listy. Pojedynczy element powinien zawierać podane koordynaty oraz rodzaj wykonanej akcji. Dynamiczna alokacja pozwoli na oszczędzanie pamięci, ponieważ nie wiemy z góry ile ruchów zostanie tam zapisane.

Algorytm celowania dla gry z komputerem będzie wymagał używania różnych funkcji wewnątrz tej samej procedury. Użycie tablicy wskaźników na funkcje jest tutaj najlepszą możliwością, ponieważ implementacja jest prosta i łatwo modyfikowalna.

2.3 Ograniczenia specyfikacji

Aplikacja będzie wymagać interfejsu konsolowego, dlatego jej działanie na niektórych komputerach może być nieprzewidywalne. Spowoduje to potrzebę zdefiniowania odpowiedniego sposobu obsługi konsoli dla jak największej liczby systemów, co zmniejszy szansę na błędy wyświetlania.

3. Specyfikacja zewnętrzna

Gra domyślnie rozgrywa się na planszy 10 na 10. Na początku gracze ustawiają swoje statki. Dostępne są następujące statki:

- 1 czteromasztowiec;
- 2 trzymasztowce;
- 3 dwumasztowce;
- 4 jednomasztowce;

Statek powinien mieścić w polu oraz nie powinien sąsiadować z innym statkiem.

Gracz po trafieniu w statek przeciwnika może wykonać kolejny strzał. Może wykonywać strzały do momentu nietrafienia. Po spudłowaniu następuje tura gracza drugiego na identycznych zasadach. Gracze wykonują swoje tury do sytuacji, w której któryś z graczy nie posiada już statków.

3.1 Obsługa programu

Przygotowanie do gry

Program można uruchomić bezpośrednio z pliku .exe lub z wiersza poleceń. W przypadku uruchomienia z wiersza poleceń można zadać przełączniki, które pozwolą ominąć początkowe ustawienia np. tryb gry wybierając je już w trakcie uruchamiania.

Dostępne przełączniki:

- **-l** uruchomienie rozpoczyna się od wczytania gry ostatniego zapisu. Można dodatkowo podać ścieżkę z której chcemy otworzyć plik zapisu. W przypadku nie podania ścieżki zapis zostanie utworzony w domyślnej lokalizacji (*Zapis.xml*);
- **-h** gra w trybie dla 2 graczy (hotseat). Rozpoczyna się od razu etapem ustawiania statków dla gracza pierwszego;
- **-c** gra z komputerem. Można po nim dodatkowo zadać drugi przełącznik **-a**, który pominie ustawianie statków dla użytkownika ustawiając je w losowy sposób. W innym przypadku gra rozpoczyna się etapem ustawiania statków użytkownika.

W przypadku błędnego zapisu przełączników gra uruchomi się tak jak w przypadku gdyby nie podano żadnego.

Wprowadzanie danych

Wygląd gry minimalnie różni się od wersji papierowej gry. Pola są opisane liczbami, dzięki czemu wystarczy popatrzeć w odpowiednie miejsce i natychmiast znamy jego poprawne koordynaty.

Przy ustawieniach program restrykcyjnie wymaga podania wartości **-1**, **0** lub **1**. W przypadku podania innej wartości wyświetli się odpowiedni komunikat, a program będzie czekał na ponowne wprowadzenie danych.
W późniejszej części gry program próbuje odczytać komendę lub koordynaty. Dopiero w przypadku braku możliwości znalezienia obu w wprowadzonych danych, program wyświetla komunikat o błędnych danych.

Po wyborze trybu będzie można w dowolnym momencie używać komend wykonujących specjalne akcje. Komendy powinny być podane dokładnie tak jak podano poniżej.
Dostępne komendy:

- **cofnij** -komenda cofa ustawienie statku. Użycie w innym momencie niż etap ustawiania statku wyświetli komunikat, a następnie pozwoli na ponowne podanie danych.
- **zapisz** -komenda wykonuje zapis aktualnego stanu gry. Można wykonać tylko w czasie walki, po ustawieniu wszystkich statków przez obu graczy.
- **wczytaj** -komenda próbuje wczytać ostatni zapis.
- **flota** -komenda wypisuje niezatopione statki przeciwnika. Można użyć tylko w czasie walki.

Ustawianie statków wymaga podania dwóch liczb całkowitych. Pierwsza jest miejscem ,w którym zamierzamy ustawić statek, druga kierunkiem w którym zamierzamy ustawić (**0**, horyzontalnie. **1**, wertykalnie). Wyjątkiem są jednomasztowce, dla których wystarczy podać miejsce.

Strzelanie do statków wymaga tylko podania miejsca.

Informacje na mapie

Program na odpowiednim polu wyświetla jedną z następujących informacji:

- **Numer (np.32)**- dokładne koordynaty pola. Pierwsza i druga cyfra to odpowiednio pozycja od góry i od lewej strony. Po podaniu tej liczby jako polecenia zostanie wykonana akcja zależna od aktualnej fazy gry;
- **V** – informacja o tym, że na danym polu znajduje się posiadany statek. W obsługiwanych systemach wyświetla się na zielono (V);
- **U** – informacja o tym, że strzelano już w dane pole i nie trafiono. W obsługiwanych systemach wyświetla się na niebiesko (U);
- X – informacja o tym, że na tym polu znajduje się trafiony statek. W obsługiwanych systemach wyświetla się na czerwono (X);

Plik konfiguracji

Program korzysta z pliku konfiguracyjnego (*config.ini*). Domyślny plik wygląda następująco:

```
ROZMIAR_POLA_X 10
ROZMIAR_POLA_Y 10
NO_SYS_NEWLINE 100
TRUDNOSC 1
```

W pliku można zmieniać wartości żeby zmienić działanie gry.

Pole można ustawiać na wartości od 5 do 10, co zmieni liczbę dostępnych statków. Minimalna liczba statków (dla pola 5 na 5) to 1 dwumasztowiec i 1 jednomasztowiec. Liczba statków rośnie względem przyrostu rozmiaru pola do domyślnej wartości.

Trudność można ustawić na 0 lub na 1 gdzie 0 jest trybem w którym komputer strzela losowo.

NO_SYS_NEWLINE pozwala zmienić liczbę nowych linii w nieobsługiwanych systemach.

3.2 Komunikaty

- „proba cofania”-komunikat informujący o cofnięciu ruchu.
- „proba zapisu”-komunikat informujący o poprawnym wprowadzeniu komendy „zapisz”. Wynik zapisywania zostanie wypisany później.
- „proba wczytywania”- komunikat informujący o poprawnym wprowadzeniu komendy „wczytaj”. Wynik wczytywania zostanie wypisany później.
- „Raport:...”- komunikat rozpoczynający wypisywanie statków przeciwnika. Występuje tylko po poprawnym użyciu komendy „flota”.
- „W te pole nie mozna wycelowac. Podaj ponownie:”- nie można strzelić w podane koordynaty. Sprawdź czy mieszczą się one na planszy oraz czy nie celujesz w miejsce, w które już strzelano.
- „Nieudane wczytywanie, sprawdz czy plik zapisu istnieje i nie jest otwarty. Plik moze byc uszkodzony.”- program nie był wstanie odczytać pliku, lub dane w nim zawarte nie są zgodne z formatem. Istnieje możliwość, że plik przestał istnieć.
- „Udane wczytanie”- gra została wczytana poprawnie i aktualna rozgrywka toczy się od momentu ostatniego zapisu.
- „Nieudane zapisywanie, sprawdz czy plik zapisu nie jest otwarty.”- komunikat o tym, że zapis nie był możliwy. Najczęściej jest to spowodowane odmową dostępu do zapisu w pliku. Najprawdopodobniej plik jest otwarty lub został usunięty w trakcie rozgrywki i nie można go utworzyć. W drugim występującym wielokrotnie zalecane jest sprawdzenie zabezpieczeń komputera lub uruchamianie w trybie administratora.

- „Udany zapis”- informacja o tym, że zapis został wykonany poprawnie. Możesz teraz bezpiecznie zamknąć grę.
- „Nie mozna teraz wykonac tej akcji.”- podana komenda nie ma zastosowania w momencie jej wprowadzenia. Niektóre komendy mają użyteczność dopiero w odpowiedniej fazie gry. Wszystko zostało opisane w poprzednim podrozdziale.
- „Trafiony!”, „Trafiony zatopiony!”, „Pudlo :(”-informacja o wyniku strzału.
- „Nie mozna tu ustawic statku. Podaj ponownie: ”- na podanych koordynatach nie można było ustawić statku. Sprawdź czy koordynaty mieszczą się na planszy i wybrano odpowiedni kierunek tak by nie nachodził na pola innego statku.
- „Kordynaty podany w blednym formacie. Odpowiedni format to <numer_pola> <kierunek>”- program nie jest w stanie przetworzyć podanych danych. Dane powinno podawać się zgodnie z odpowiednim formatem przedstawionym dalej. Nawiasy („<” i „>”) zostały podane dla czytelności i nie powinno ich się podawać przy wprowadzaniu danych.
- „Wybierz miedzy 0 i 1!”- podane dane nie zostały dopasowane przez program. Po wystąpieniu powinno się podać tylko 0 lub 1, zgodnie z instrukcjami nad komunikatem.
- „Bledna alokacja pamieci. Sprobuj uruchomic ponownie.” - program nie był wstanie za alokować pamięci. W przypadku wielokrotnie powtarzającego się tego komunikatu zaleca się uruchomienie programu jako administrator.
- „Nie znaleziono zapisu lub plik z zapisem jest uszkodzony. Sprobuj uruchomic gre ponownie lub zaczac od poczatku”- komunikat pojawia się przy próbie rozpoczęcia od wczytania. Gdy parokrotnie się to nie udało plik z zapisem może być uszkodzony, wymaga to uruchomienia gry ponownie bez wczytywania.
- „Nacisnij dowolny klawisz”-pauza potrzebna na zmianę graczy. Wystarczy naciśnięcie klawisza Enter lub wprowadzenie dowolnego znaku.
- „Wygral ...”-informacja o tym który gracz wygrał, po tym komunikacie gra się kończy.
- „Blad pliku konfiguracji lub pliku kodowania”- plik config.ini lub Kod.txt zostały usunięte, uszkodzone lub program nie ma do nich dostępu. Należy ponownie pobrać oba pliki.
- „PLIK KONIFIGURACJI : Podano rozmiar poza zakresem.Dozwolony zakres to między 5, a 10” -podane rozmiary pola są za duże. Gra ogranicza rozmiar tak, by zachować ją w dużej dowolności, przy jednoczesnym zachowaniu grywalności.

4. Specyfikacja zewnętrzna

Funkcje służące do zapisu i odczytu stanu gry korzystają z zewnętrznej biblioteki Mini-XML.

4.1.1 Ważne zmienne

`trybGry`	Zmienna typu `int` przechowująca tryb gry. Steruje ścieżką, którą przejdzie program. Przyjmuje następujące wartości: **-1** - gdy wybrano tryb gry z komputerem i automatyczne rozmieszczenie; **0** - gdy wybrano tryb gry z komputerem; **1** - gdy wybrano tryb dla dwóch graczy.
`statki`	Zmienna typu `int` przechowująca informacje o pozostałych statkach. Znajduje się w strukturze `Gracz`. Informacja w niej jest zakodowana w systemie ósemkowym. Każda pozycja oznacza inny statek, a cyfra na tej pozycji oznacza ile pól tego statku nie zostało zatopione.

`(*stan[6])(int poprzedniePole)`	Tablica wskaźników na funkcje przyjmujących argument typu `int` i zwracających typ `int`. Jest to tablica przechowująca funkcje zwracające komputerowi gdzie powinien strzelić na podstawie pola w które celowano poprzednio. Tablica powinna być odpowiednio zainicjowana przed użyciem w jakiejkolwiek funkcji wymagającej ruchu komputera.
`konfiguracja`	Struktura danych przechowująca ustawienia gry oraz kody zwracane przez funkcje. Dane do niej przekazywane są przez parametry wiersza poleceń i pliki konfiguracyjne.
`rozgrywka`	Struktura danych przechowująca: obu graczy, drzewo xml, listę ruchów, strukturę sterującą sztuczną inteligencją.

4.1.2 Struktury danych i własne typy

4.1.2.1 Gracz
Opis: Struktura przechowująca dane gracza.
Zmienne:

`int** pole`	Dynamicznie alokowana tablica dwuwymiarowa – plansza.
`int statki`	Liczba reprezentująca statki gracza opisana w poprzednim podrozdziale.

4.1.2.2 Wybor
Opis: Struktura obsługująca sztuczną inteligencję.
Zmienne:

`int stanPoprzedni`	Indeks informujący, który z elementów tablicy wskaźników na funkcje będzie używany w trakcie decyzji.
`int(*stan[6])(int poprzedniePole)`	Tablica wskaźników na funkcje opisana w poprzednim podrozdziale.
`int aktualnePole`	Koordynaty pola na które ostatnio celowano.

4.1.2.3 Lista (Historia)
Opis: Lista dynamicza, będąca historią ruchów, które zostały wykonane. Stworzona jako kolejka, do której dane dodawane są na jej początek.
Zmienne:

`Zadanie zadanie`	Zmienna typu wyliczeniowego służąca jako etykieta dla danego ruchu.
`int argument`	Argument jakiego użyto podczas wykonywania ruchu.
`int rodzaj`	Pomocniczy argument dla niektórych typów ruchu.
`struct Lista* pPoprzednia`	Wskaźnik na poprzedni element w liście.

4.1.2.4 Rozgrywka

gracz1,gracz2	Struktury przechowujące graczy.
xml	Drzewo xml
tura_n, czyja_n	Wskaźniki na miejsca w drzewie z najczęściej używanymi informacjami
ruchy	Lista ruchów gracza
AI	Struktura sterująca sztuczną inteligencją.

4.1.2.5 Konfiguracja

`TrybGry, automat, wczytaj,`	Zmienne pobierane jako parametry wiersza poleceń
`plikZapisu`	Ścieżka pod którą znajduje się plik zapisu
`rozmiarPola_x, rozmiarPola_y, trudnosc, no_system_new_line_number`	Zmienne pobierane z pliku konfiguracyjnego

`zad_brak,`
`zad_wczytaj,`
`zad_zapisz,`
`zad_cofnij,`
`zad_flota,`
`b_koniec,`
`b_niekoniec,`
`b_wczytaj,`
`st_blad,`
`st_pudlo,`
`st_cel,`
`st_zatop`
	Kody zwracane przez funkcje

4.1.2.6 Dzialania (Zadanie)

Opis: Typ wyliczeniowy wykorzystywany jako etykieta dla danego ruchu. Określa jego charakter oraz akcje wykonaną pozwalając na odpowiednie odczytanie argumentów.

Możliwe wartości:

start	Inicjalizacja listy. Pozwala określić blokadę, za którą funkcję nie powinny wychodzić. Umożliwia to korzystanie z listy bez potrzeby usuwania i bez ryzyka wypisania nieodpowiednich danych.
ustaw	Ustawienie statku. Pozwala w odpowiedni sposób odczytać dane o ustawianiu statku.
strzal	Strzał. Pozwala w odpowiedni sposób odczytać dane o ostrzale.

4.1.2.7 Kolor

Opis: Typ wyliczeniowy określający kolor dla funkcji `ZmienKolor`. Jego rzutowanie na typ całkowity odpowiada interpretacji kolorów konsoli w systemie Windows.

zielony	Kolor zielony, ustawiony na numer 10
niebieski	Kolor niebieski, ustawiony na numer 11
czerwony	Kolor czerwony, ustawiony na numer 12
bialy	Kolor biały, ustawiony na numer 15

4.1.3 Funkcje

Funkcje wypisane są w porządku alfabetycznym.

4.1.3.1 AutoRozmieszczenie()

```
void AutoRozmieszczenie (
            Konfiguracja * konfiguracja,
            Gracz * gracz )
```

Funkcja rozpoczyna przygotowanie do gry losowo umieszczając wszystkie statki.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
gracz	Gracz, którego pole gry będzie ustalane poprzez losowe rozmieszczenie statków

4.1.3.2 Bitwa()

```
int Bitwa (
            Konfiguracja * konfiguracja,
            Rozgrywka * rozgrywka )
```

Komunikacja z użytkownikiem po zakończeniu przygotowań. Pojedyncze wywołanie jest pojedynczą turą jednego gracza

Parametry

konfiguracja	struktura przechowująca ustawienia gry
rozgrywka	struktura przechowująca dane obecnej rozgrywki

Zwraca

0 (B_KONIEC) Informacja o tym, że jeden z graczy stracił wszystkie statki. Wartość ta kończy pętle gry
1 (B_NIEKONIEC) Informacja o tym, że każdy gracz ma jeszcze statki.
2 (B_WCZYTAJ) Wartość zwracana w przypadku udanego wcytania gry.

4.1.3.3 BitwaAI()

```
int BitwaAI (
            Konfiguracja * konfiguracja,
            Rozgrywka * rozgrywka,
            Wybor * AI )
```

Funkcja obsługująca turę dla gry z komputerem.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
rozgrywka	struktura przechowująca dane obecnej rozgrywki
AI	Struktura posiadająca zainicjowaną tablice wskaźników na funckje celujące oraz indeks następnej funkcji do wykonania

Zwraca

0 (B_KONIEC) Informacja o tym, że jeden z graczy stracił wszystkie statki. Wartość ta kończy pętle gry
1 (B_NIEKONIEC) Informacja o tym, że każdy gracz ma jeszcze statki.

4.1.3.4 DodajdoFloty()

```
void DodajdoFloty (
            Gracz * gracz,
            int pozycja,
            int dlugosc )
```

Funkcja dodaje odpowiednią ilość punktów życia do statków gracza, na odpowiedniej pozycji.

Parametry

gracz	Gracz do którego dodane zostaną punkty życia
pozycja	Pozycja w systemie usemkowym na którą zostanie dodana liczba punktów życia
dlugosc	Dodawana liczba punktów życia, zależna od statku

4.1.3.5 DodajdoListy()

```
Historia* DodajdoListy (
            Historia **lista,
            Zadanie zadanie,
            int argument,
            int rodzaj )
```

Funkcja do dająca element do listy na jej początek.

Parametry

lista	Wskaźnik na listę do której należy dodać element
zadanie	Wartość określa jaki typ ruchy został wykonany.
argument	Informacja do zapsiania na liście. Może być dowolna, ale została przygotowana na koordynaty
rodzaj	informacja do zapsiania na liście. Może być dowolna, ale została przygotowana na kierunek ustawienia statku lub wynik strzału

Zwraca

Wskaznik na liste
NULL, gdy nie udało się zaalokować pamięci

4.1.3.6 GameLoop()

```
void GameLoop (
            Konfiguracja * konfiguracja,
            Rozgrywka * rozgrywka )
```

Funkcja stanowi główną pętle gry. Przekazuje graczom odpowiednie informacje porządkujące grę. Tylko dla gry w trybie hotseat

Parametry

konfiguracja	struktura przechowująca ustawienia gry
rozgrywka	struktura przechowująca dane obecnej rozgrywki

4.1.3.7 GameLoopAI()

```
void GameLoopAI (
            Konfiguracja * konfiguracja,
            Rozgrywka * rozgrywka )
```

Funkcja stanowi główną pętle gry. Przekazuje graczowi odpowiednie informacje porządkujące grę. Tylko dla gry z komputerem.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
rozgrywka	struktura przechowująca dane obecnej rozgrywki

4.1.3.8 IdzE()

```
int IdzE (
            Konfiguracja * konfiguracja,
            int poprzedniePole )
```

Funkcja przenosząca celowanie o jedno pole w prawo.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
poprzedniePole	Poprzednie pole w które celowano

Zwraca

Nowe pole, w które należy wycelować

4.1.3.9 IdzN()

```
int IdzN (
            Konfiguracja * konfiguracja,
            int poprzedniePole )
```

Funkcja przenosząca celowanie o jedno pole do góry.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
poprzedniePole	Poprzednie pole w które celowano

Zwraca

Nowe pole, w które należy wycelować

4.1.3.10 IdzS()

```
int IdzS (
            Konfiguracja * konfiguracja,
            int poprzedniePole )
```

Funkcja przenosząca celowanie o jedno pole w dół.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
poprzedniePole	Poprzednie pole w które celowano

Zwraca

Nowe pole, w które należy wycelować

4.1.3.11 IdzSkos()

```
int IdzSkos (
            Konfiguracja * konfiguracja,
            int poprzedniePole )
```

Funkcja przenosząca celowanie o jedno pole w prawo i jedno w dół.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
poprzedniePole	Poprzednie pole w które celowano

Zwraca

Nowe pole, w które należy wycelować

4.1.3.12 IdzW()

```
int IdzW (
            Konfiguracja * konfiguracja,
            int poprzedniePole )
```

Funkcja przenosząca celowanie o jedno pole w lewo.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
poprzedniePole	Poprzednie pole w które celowano

Zwraca

Nowe pole, w które należy wycelować

4.1.3.13 InicializujAI()

```
void InicializujAI (
            Wybor * AI,
            Konfiguracja * konfiguracja )
```

Funkcja ustawia sztuczną inteligencję zgdonie z konfiguracją.

Parametry

AI,struktura	przechowująca dane sztucznej inteligencji
konfiguracja	struktura przechowująca ustawienia gry

4.1.3.14 Iniciuj()

```
int Iniciuj (
            Konfiguracja * konfiguracja,
            Rozgrywka * rozgrywka )
```

Funkcja przeprowadza całą początkową fazę gry.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
rozgrywka	struktura przechowująca dane obecnej rozgrywki

Zwraca

0, gdy nie udało się poprawnie rozpocząć gry
1, gdy cały proces przebiegł pomyślnie

4.1.3.15 Losuj()

```
int Losuj (
            Konfiguracja * konfiguracja,
            int poprzedniePole )
```

Funckcja losująca koordynaty dla algorytmu celowania komputera.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
poprzedniePole	Poprzednie pole w które celowano

Zwraca

Nowe pole, w które należy wycelować

4.1.3.16 Oczysc()

```
void Oczysc (
            Konfiguracja * konfiguracja )
```

Funkcja czyszcząca konsolę. Zabezpieczenie przed podglądaniem pól przeciwnika. W przyadku nieobsługiwanego systemu operacyjnego funkcja wypisze dostateczną ilość nowych linii, aby gracze nie widzieli w swojej turze statków przeciwnika.

Parametry

konfiguracja	struktura przechowująca ustawienia gry

4.1.3.17 PobierzKodowanie()

```
int PobierzKodowanie (
            Konfiguracja * konfiguracja )
```

Funkcja pobiera kodowanie z pliku kod.txt.

Parametry

konfiguracja	struktura przechowująca ustawienia gry

Zwraca

0, gdy nie znaleziono pliku z kodowaniem
1, gdy odczytano kodowanie z pliku. Nie gwarantuje to poprawności kodowania w przypadku zmiany kodu źródłowego programu.

4.1.3.18 PobierzKoordynaty()

```
void PobierzKoordynaty (
            Konfiguracja * konfiguracja,
            int dlugosc,
            Gracz * gracz,
            Historia **historia,
            int rodzaj )
```

Funkcja pobiera od użytkownika informacje gdzie powinien zostać umieszczony pojedyczy statek.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
dlugosc	Długość statku, który należy umieścić
gracz	Gracz, który aktualnie ustawia statki
historia	Lista ruchów gracza. Pozwala na cofanie statków
rodzaj	identyfikator statku. Ta wartość zostanie wpisana na pole w momencie ustawiania statku

4.1.3.19 PobierzParametry()

```
int PobierzParametry (
            int ileArg,
            char * arg[ ],
            Konfiguracja * konfiguracja )
```

Funkcja pobiera argumenty wiersza poleceń i ustawia według nich konfiguracje.

Parametry

ileArg	ilość argrumentów wiersza poleceń
arg	argumenty wiersza poleceń
konfiguracja	struktura przechowująca ustawienia gry

Zwraca

0, gdy pobranie parametrów było nieudane
1, gdy pobranie prametrów powiodło się.

4.1.3.20 Rozmieszczenie()

```
void Rozmieszczenie (
            Konfiguracja * konfiguracja,
            Gracz * gracz )
```

Funkcja rozpoczyna przygotowanie do gry w przypadku, gdy nie rozpoczęto od wczytania zapisu. Funkcja komunikuje się z użytkwonikiem przez konsole.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
gracz	Gracz, którego pole gry będzie ustalane na podstawie informacji otrzymywanych od użytkownika

4.1.3.21 RysujPlansze()

```
void RysujPlansze (
            Konfiguracja * konfiguracja,
            Gracz gracz,
            int dyskrecja )
```

Funkcja wyświetlająca graczowi planszę jego lub przeciwnika.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
gracz	Gracz, którego planszę należy wyświetlić
dyskrecja	Parametr określający czy należy wyświetlić statki. Dla zmiennej równej 0, statki są wyświetlane, w przeciwnym wypadku są pomijane.

4.1.3.22 Skonfiguruj()

```
int Skonfiguruj (
            Konfiguracja * konfiguracja )
```

Funkcja pobiera konfiguracje z pliku config.ini

Parametry

konfiguracja	struktura przechowująca ustawienia gry 0, gdy znaleziono plik i odczytano konfiguracje

Zwraca

1, gdy znaleziono plik i odczytano konfiguracje
0, gdy nie znaleziono pliku
-1, gdy podany zakres był niedozwolony

4.1.3.23 SprawdzZgodnosc()

```
int SprawdzZgodnosc (
            mxml_node_t * xml,
            Konfiguracja * konfiguracja )
```

Funkcja sprawdza zgodność pliku zapisu z obecną konfiguracją gry.

Parametry

xml	Drzewo xml odczytane z pliku zapisu
konfiguracja	struktura przechowująca ustawienia gry

Zwraca

0, gdy dane nie są zgodne
1, gdy dane są zgodne

4.1.3.24 Strzal()

```
int Strzal (
            Konfiguracja * konfiguracja,
            Gracz * atakowanyGracz,
            int pole )
```

Funkcja sprawdzająca czy strzał jest możliwy do wykonanania oraz zwracaj ąca informacja o tym jaki był wynik strzału.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
atakowanyGracz	Gracz, w którego pole zostanie wykonany strzał
pole	Pole, w które należy strzelić. Koordynaty są podane jako liczba dwucyfrowa. Pierwsza cyfra określa pole na osi X, a druga na osi Y

Zwraca

0 (ST_BLAD) Informacja o tym, że strzał nie może sotać wykonany w podane pole
1 (ST_PUDLO) Informacja o tym, że strzał został wykonany poprawnie w pole na którym nie było statku
2 (ST_CEL) Informacja o tym, że strzał został wykonany poprawnie w pole na którym znajdował się statek
3 (ST_ZATOP) Informacja o tym, że strzał został wykonany poprawnie oraz statek, który trafiono został zatopiony

4.1.3.25 UstawParametry()

```
void UstawParametry (
            Konfiguracja * konfiguracja )
```

Funckcja ustawia brakujące parametry, przekazywane zwykle przez wiersz poleceń, komunikując się z użytkownikiem.

Parametry

konfiguracja	struktura przechowująca ustawienia gry

4.1.3.26 UstawStatek()

```
int UstawStatek (
            Konfiguracja * konfiguracja,
            Gracz * gracz,
            int dlugosc,
            int pole,
            int kierunek,
            int rodzajStatku )
```

Funkcja ustawiająca statki na planszy. Sprawdza czy statek może zostać ustawiony, a następnie go ustawia zmieniając wartości znajdujące się na tablicy dwu wymiarowej na identyfikator statku.

Parametry

gracz	Gracz, który ustawia statek
dlugosc	Długość ustawianego statku
pole	Pole, w którym zaczyna się statek. Jest to liczba dwucyfrowa, której pierwsza cyfra to pole na osi X, a druga to pole na osi Y
kierunek	Informacja, w którym kierunku powinien być ustawiony statek. 1- wertykalnie. 0- horyzontalnie
rodzajStatku	Identyfikator statku. Ta liczba zostanie wpisana na pole podczas ustawiania. Powinna być mniejsza od -3 oraz unikalna (wyjątkiem są statki zajmujące jedno pole)

Zwraca

0, gdy na podanych koordynatach nie można ustawić statku w danym kierunku
1, gdy statek został ustawiony poprawnie

4.1.3.27 UsunListe()

```
void UsunListe (
            Historia **lista )
```

Usuwanie dynamicznie zaalokowanej listy.

Parametry

lista	Wskaźnik na listę, której pamięc należy usunąć

4.1.3.28 UsunStatek()

```
int UsunStatek (
            Konfiguracja * konfiguracja,
            Historia **historia,
            Gracz * gracz )
```

Funkcja usuwająca statek z planszy. Używana tylko w trakcie przygotowania do gry na żądanie gracza. Następnie prosi o podanie nowych koordynatów tak jakby poprzedni ruch nie został wykonany .Do poprawnego działania lista historia powinna być poprawnie zainicjowana, a jej najstarszy element powinien być utworzony z typem wyliczeniowym start.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
historia	Wskaźnik na liste ruchów
gracz	Gracz, który aktualnie wykonuje ruch

Zwraca

1, gdy usunięto i ponownie ustawiono statek
0, gdy nie ma już ruchów do cofnięcia na liście
-1, gdy lista została źle utworzona (jej pierwszy element nie ma określonego odpowiedniego typu wyliczeniowego)

4.1.3.29 UsunTablice()

```
void UsunTablice (
            Konfiguracja * konfiguracja,
            Gracz * gracz1,
            Gracz * gracz2 )
```

Procedura usuwania dynamicznie zaalokownaych tablic.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
gracz1	Gracz, którego tablicę należy wypełnić
gracz2	Gracz, którego tablicę należy wypełnić. Powinien być rózny od gracza pierwszego

4.1.3.30 UtworzZapis()

```
int UtworzZapis (
            Konfiguracja * konfiguracja,
            Rozgrywka * rozgrywka )
```

Funkcja tworzy zapis na podstawie ustawie ń i zakończonego, przez obu graczy, przygotowania do gry. Tworzy drzewo xml, które pozwala na łatwe zapisywanie w późniejszej częsci gry.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
rozgrywka	struktura przechowująca dane obecnej rozgrywki

Zwraca

1, gdy poprawnie wykonano zapis do pliku
0, gdy nie można było zapisać do pliku. W tym przypadku drzewo zostało utworzone tylko w pamięci aplikacji

4.1.3.31 Wczytaj()

```
int Wczytaj (
            Konfiguracja * konfiguracja,
            Rozgrywka * rozgrywka )
```

Funkcja wczytuje dane z pliku z zapisem nadpisując obecne informacje. Nadpisane zostaną dane graczy, drzewo xml oraz lista wykonanych ruchów.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
rozgrywka	struktura przechowująca dane obecnej rozgrywki

Zwraca

0, gdy nie można było otworzyć pliku lub załadowanie drzewa z tego pliku było niemożliwe
1, gdy procedura odczytywania przebiegła pomyślnie

4.1.3.32 WczytajGracza()

```
void WczytajGracza (
            mxml_node_t * parent,
            Konfiguracja * konfiguracja,
            Gracz * gracz )
```

Funkcja odczytuje dane gracza z drzewa xml.

Parametry

parent	Część drzewa pod które zapisane zostały dane gracza
konfiguracja	struktura przechowująca ustawienia gry
gracz	Struktura z danymi gracza czytanymi z drzewa

4.1.3.33 WprowadzZadanie()

```
int WprowadzZadanie (
            int liczbaArgumentow )
```

Funkcja do komunikacji z użytkownikiem, pozwalająca na używanie komend w dowolnej chwili. Po wprowadzeniu komendy wypisuje informacje, które działanie próbowano wykonać.

Parametry

liczbaArgumentow	Parametr określający jak dużo danych chcemy otrzymać. Dla 0 wymaga naciśnięcia klawisza "Enter", dla 1 wymaga jednej wartości, dla 2 dwóch wartości. Inne wartości nie są obsługiwane

Zwraca

-2 (ZAD_FLOTA) Zwracana, gdy użyto komendy "flota"
-3 (ZAD_COFNIJ) Zwracana, gdy użyto komendy "cofnij"
-4 (ZAD_ZAPISZ) Zwracana, gdy użyto komendy "zapisz"
-5 (ZAD_WCZYTAJ) Zwracana, gdy użyto komendy "wczytaj"
0 (ZAD_BRAK) Zwracana, gdy liczbaArgumentow wynosi 0 i nie podano komedy
Liczba całkowita, której cyfra jedności ma wartość drugiego argumentu a pozostała część liczby wartość pierwszego. Zwracana tylko gdy nie użyto żadnej komendy

4.1.3.34 WyczyscBufor()

```
void WyczyscBufor ( )
```

Funkcja czyszcząca pozostałości z bufora.

4.1.3.35 WypelnijTablice()

```
int WypelnijTablice (
            Konfiguracja * konfiguracja,
            Gracz * gracz1,
            Gracz * gracz2 )
```

Procedura alokowania i wypełnienia tablic będących planszami dla graczy. Tablice alokowane są dynamicznie korzystając z rozmiaru określonego w makro ROZMIAR_POLA. Gdy alokacja pamięci będzie niemożliwa, funkcja zwolni zaalokowaną przez nią pamięć.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
gracz1	Gracz, którego tablicę należy wypełnić
gracz2	Gracz, którego tablicę należy wypełnić. Powinien być różny od gracza pierwszego

Zwraca

1, gdy alokacja zakończyła się sukcesem
0, gdy wystąpiły błędy alokacji. Pamięć zaalokowana do błędu zostanie w funkcji zwolniona

4.1.3.36 WypiszFlote()

```
void WypiszFlote (
            Gracz * przeciwnik )
```

Funkcja wypisująca statki przeciwnika, które nie zostały zatopione.

Parametry

przeciwnik	Gracz, którego niezatopione statki należy wypisać

4.1.3.37 WypiszRuchy()

```
void WypiszRuchy (
            Historia * ruchy )
```

Funkcja wypisująca ruchy poprzedniego gracza. Nie modyfikuje listy. Wypisuje do pierwszego sppotkanego elementu z typem wyliczeniowym określonym jako "start".

Parametry

ruchy	Lista wykonanych ruchów.

4.1.3.38 ZakonczGre()

```
void ZakonczGre (
            Konfiguracja * konfiguracja,
            Rozgrywka * rozgrywka,
            int wynik )
```

Funkcja kończy grę ogłaszając gracza i usuwając dynamicznie zaalokowaną pamięć.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
rozgrywka	struktura przechowująca dane obecnej rozgrywki
wynik	Wynik na podstawie którego zaostaje wytypowany wygrywający gracz

4.1.3.39 ZamianaGraczy()

```
void ZamianaGraczy (
            Rozgrywka * rozgrywka )
```

Funkcja zamienia pozycjami graczy by gracz drugi mógł wykonać swoją turę.

Parametry

rozgrywka	struktura przechowuj ąca dane obecnej rozgrywki

4.1.3.40 Zapisz()

```
int Zapisz (
            Konfiguracja * konfiguracja,
            Rozgrywka * rozgrywka )
```

Funkcja wykonująca zapis w tracie rozgrywki. Początkowo funkcja aktualizuje drzewo danymi, których ciągły zapis do drzewa byłby nieoptymalny. Dane aktualizowane znajdują sie w strukturach Gracz.

Parametry

konfiguracja	struktura przechowująca ustawienia gry
rozgrywka	struktura przechowująca dane obecnej rozgrywki

Zwraca

1, gdy poprawnie wykonano zapis do pliku
0, gdy nie można było zapisać do pliku. W tym przypadku drzewo zostało utworzone tylko w pamięci aplikacji

4.1.3.41 ZapiszGracza()

```
void ZapiszGracza (
            mxml_node_t * parent,
            Konfiguracja * konfiguracja,
            Gracz gracz )
```

Funkcja zapisuje dane gracza do drzewa xml.

Parametry

parent	Część drzewa pod które zapisane zostaną dane gracza
konfiguracja	struktura przechowująca ustawienia gry
gracz	Struktura z danymi gracza zapisywanymi do drzewa

4.1.3.42 ZmienKolor()

```
void ZmienKolor (
            Kolor typ )
```

Funkcja zmieniająca kolor tekstu. Ma zastosowanie estetyczne. Dla nieobsługiwanych systemów nie wykonażadnej instrukcji.

Parametry

typ	Kod koloru na który zostanie zamieniony kolor tekstu w konsoli

5. Testowanie

Program został przetestowany. Gra w przypadku gry z komputerem, jak i w trybie dla dwóch graczy uruchamia się poprawnie niezależnie od sposobu uruchamiania. Gra nie zawiesza się oraz nie generuje nieoczekiwanych komunikatów.

Brak pliku zapisem nie powoduje błędów działania. Uszkodzony plik zapisu może spowodować błędy, gdy dane zmienione nie przez program są niezbędnie potrzebne by program działał poprawnie. Zmiana niektórych danych (np. licznika tur) w pliku może zmodyfikować rozgrywkę i fragmenty estetyki programu nie wpływając na poprawność wykonania programu.

Dla systemu Windows wszystko działa poprawnie. Dla systemów UNIX również powinien działać, jednak przetestowałem działanie funkcji zmieniającej kolor i czyszczącej konsolę tylko dla Linux'a.

6. Wnioski

Wszystkie założenia z fazy projektowania zostały spełnione. Najwięcej kłopotów sprawiła zgodność do jak największej ilości systemów i podzielenie programu na osobne pliki podczas gdy był już gotowy.

Projekt pozwolił mi poprawić błędy, które popełniałem w języku C oraz przećwiczyć przydatne zastosowanie wskaźników na funkcje. Nauczyłem się dodawać biblioteki zewnętrzne w Visual Studio oraz jak korzystać z takich bibliotek i ich dokumentacji. Dowiedziałem się, że interfejs konsolowy może być przydatny nie tylko w przypadku programów wykonujących obliczenia, ale także w przypadku tworzenia gier. Zrozumiałem jak dużo uwagi wymaga stworzenie odpowiedniego i czytelnego interfejsu użytkownika. Przećwiczyłem kodowanie informacji w typie całkowitym, co pozwala na używanie mniejszej ilości zmiennych i łatwiejsze zastosowanie pętli.