POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Dokumentacja z projektu z przedmiotu Programowanie w języku C++

Projekt:

Gra w statki, dokumentacja

Opracowanie:

Wojciech Zajączkowski

Rok: 2EF-DI

Grupa projektowa: P02

Data: 20.01.2024

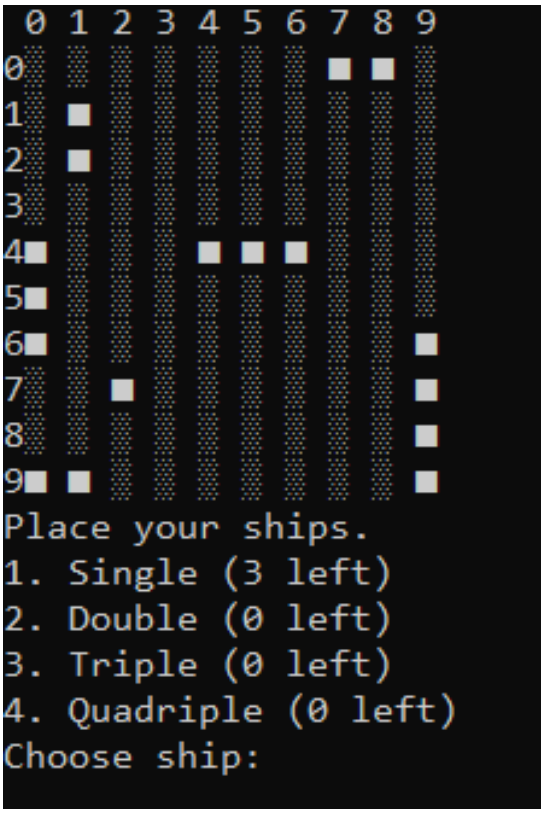
Temat

Zrealizowanie klasycznej wersji gry statki i wzbogacenie jej o unikalne elementy, nadające dynamikę rozgrywce.

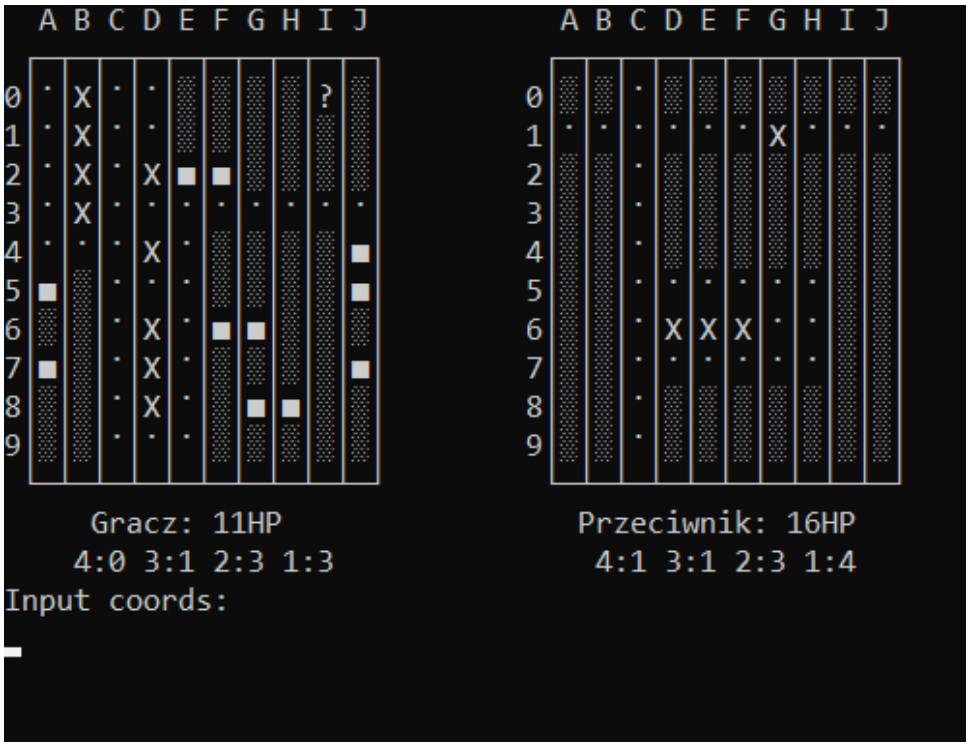
Screeny z gry

Scena tytułowa:

Kreowanie morza:



W trakcie gry:



Zasady gry

- Gra rozgrywa się na dwóch planszach 10x10 – przeciwnika i gracza.
- Gra przebiega naprzemiennie: najpierw gracz oddaje strzał, jeśli uda się mu trafić, może kontynuować „strzelanie” do momentu, aż chybi (z wyjątkiem używania bomb i rakiet – w takim wypadku nawet po trafieniu statku nie przysługuje dodatkowy ruch).
- Po jego turze następuje kolej bota.
- Zatopienie statku ma miejsce, kiedy wszystkie kratki symbolizujące ten okręt zostaną „trafione” przez drugiego gracza.

Elementy rozszerzające zwykłą grę

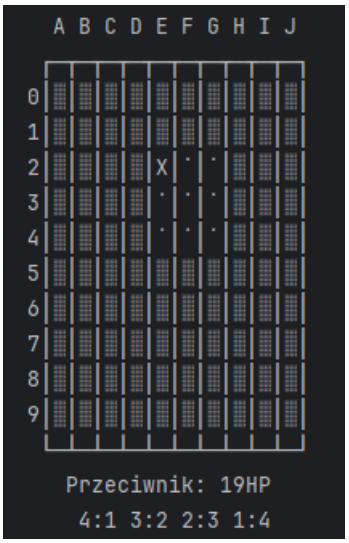
- Bomby,

- Rakiety,

- Ukryty skarb,

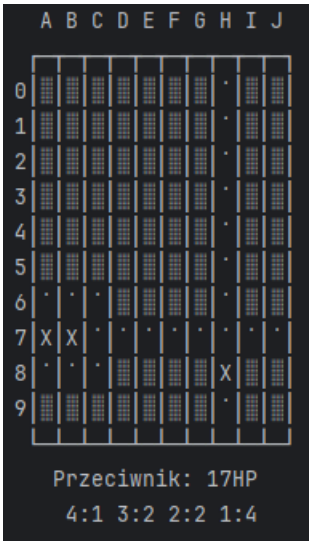
Bomby – Równoznaczna z 9-cioma strzałami na raz. Pokrywa pole 3x3. Gracz otrzymuje bombę co 3 zbite statki. Brak możliwości ponownego strzału.

Strzał w miejsce 3F:



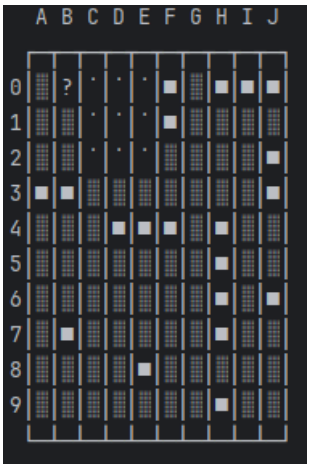
Rakiety – jak poniżej; gracz otrzymuje rakietę co 7 zbitych statków. Brak możliwości ponownego strzału.

Strzał w miejsce 7H:



Ukryty skarb – występuje tylko raz na mapie. Aktualnie brak zastosowania.

Oznaczenie na mapie:



Instrukcja gry

1. W Przypadku wyświetlenia poleceń podobnych do:

Choose bot difficulty: **1. Easy** **2. World class champion**

lub

Place your ships. 1. Single (4 left) 2. Double (3 left) 3. Triple (2 left) 4. Quadriple (1 left) Choose ship:

Należy kliknąć na klawiaturze przycisk np. „1” lub „2” a następnie zaakceptować enterem.

W przypadku konieczności wprowadzenia koordynatów, należy je wprowadzić następująco:

1A

Lub

A1

I zaakceptować enterem.

Dokumentacja kodu

Board

Pola

Publiczne:

```
int** table;
```
Tworzy tablicę dynamiczną, dwuwymiarową.

```
int width, height;
```
Zmienne przechowujące wymiary planszy.

```
int aliveFields = 0;
```
Zmienna przechowująca ilość żywych statków na planszy.

```
int bombCounter = 1;
```
Zmienna przechowująca, ile bomb posiada gracz (W uproszczeniu, gracz == plansza).

```
int missleCounter = 1;
```
Zmienna przechowująca, ile rakiet posiada.

```
int treasureCounter = 0;
```
Zmienna przechowująca, ile skarbów posiada gracz.

```
int quadraCounter = 1;
```
Zmienna przechowująca, ile statków o długości 4 posiada gracz.

```
int tripleCounter = 2;
```
Zmienna przechowująca, ile statków o długości 3 posiada gracz.

```
int doubleCounter = 3;
```
Zmienna przechowująca, ile statków o długości 2 posiada gracz.

```
int singleCounter = 4;
```
Zmienna przechowująca, ile statków o długości 1 posiada gracz.

Metody

Publiczne:

```
Board(int width, int height);
```
Konstruktor nadaje wartość width i height oraz tworzy tablicę jednowymiarową o długości height, a następnie każdą jej komórkę wypełnia adresem innej tablicy, o długości width.

```
int coordX(string ciag);
```
Wyłuskuje ze stringa wejściowego cyfrę, a następnie ją zwraca.

```
int coordY(string ciag);
```
Wyłuskuje ze stringa literę, a następnie przekazuje ją do zamienNaKoord.

```
int zamienNaKoord(char znak);
```
Zamienia char na liczbę (A=0, B=1, C=2…) i ją zwraca.

```
bool shoot(int x, int y);
```

Weryfikuje prawidłowość danych wejściowych. Jeśli są niedobre, zwraca 1. (1 – pozwolenie na kolejny strzał, 0 – tura przeciwnika). W takim przypadku funkcja zapętla się az do skutku. Jeśli zostaną wprowadzone odpowiednie koordynaty, oddawany jest strzał;

```
//what a field number means
//0 - empty (no ship part)
//1 - alive ship part
//2 - dead ship part
//3 - field with missed shot
//4 - completely dead ship
//5 - auto-filled nothing
//6 - unopened treasure
//7 - opened treasure
```

-Jeśli na polu strzału znajduje się 0 – zamieniana jest wartość pola na 3.

-Jeśli na polu strzału znajduje się 1 – zamieniana jest wartość pola na 2.

Następnie weryfikowane jest, czy graczowi należy się bomba/rakieta.

Następnie wyszukiwani są wszystkie części obecnego statku, ich koordynaty zostają zapisane do sasiedzi[][]. Następnie sprawdza, czy wszystkie te części statku są martwe. Jeśli tak, wartości tablicy dla koordynatów z tablicy sąsiedzi, zostają zamieniane na 4. Jeśli statek jest całkiem martwy, „rysowane" jest wokół niego obramowanie. Następnie odejmujemy 1 od licznika statków o tej długości.

-Jeśli na polu strzału znajduje się 6 – zamieniana jest wartość pola na 7.

```
void putTreasure();
```
W jednym polu na całej planszy, jednocześnie nie sąsiadującym z jakimkolwiek statkiem stawia 6 – ukryty skarb.

```
void useTreasure(int x, int y)
```
Używa skarbu – obecnie nie robi nic.

```
void printBoard(int mode) const;
```
Wypisuje tablicę w jednym z dwóch trybów - ukrytym i normalnym.

Ukryty – widać tylko tam gdzie oddaliśmy strzał

Normalny – widać wszystkie pola, pozycje statków, skarby itp.

```
int handleCoordInput(string ciag, char ktoryKoord);
```
Wspomaga się funkcjami coordX i coordY. Zwraca koordynat X lub y, w zależności od podanego char.

```
bool areCoordinatesValid(int x, int y) const;
```
Test, czy wprowadzone koordynaty należą do tablicy.

```
void useBomb(int x, int y);
```
Używa bombe – oddaje łącznie 9 strzałów - we wszystkie sąsiadujące pola, i swoje. Korzysta z metody shoot(x,y).

```
void useMissle(int x, int y);
```
Używa rakietę na podanych koordynatach; używa shoot() na wybranych polach na krzyż.

```
bool placeShip(int x, int y, int orientation,string type);
```
Metoda stworzona do stawiania statku. Podajemy koordynaty, typ statku oraz orientacje. 1- pionowa, 2 – pozioma.

```
void shipCreator();
```
Wywołuje kreator stawiania statków. Posiada dwie opcje – manualne stawianie statków oraz losowe.

```
bool canPartBePlacedProperly(int x, int y) const;
```
Test, czy część statku może zostać bezpiecznie postawiona – czy nie będzie sąsiadowała z innym statkiem i czy koordynaty znajdują się w zakresie wielkości tablicy.

```
void fillBoardWithEmpties() const;
```
Wypełnia tablicę pustymi polami. (zerami)

```
void fillWithRandomShips();
```
Metoda wypełniająca według reguł gry tablicę statkami. Na końcu używa funkcji putTreasure().

Prywatne:

```
bool canShipBePlacedProperly(int x, int y, int shipType, int orientation) const;
```
Sprawdza, czy cały statek może zostać poprawnie ustawiony, na podstawie jego wielkości oraz orientacji.

```
bool placePart(int x, int y) const;
```
Stawia pojedyńcze pole statku.

```
bool placeShipSingle(int x, int y, int orientation) const;
```
Stawia statek składający się z jednego pola.

```
bool placeShipDouble(int x, int y, int orientation) const;
```
Stawia statek składający się z dwóch pola.

```
bool placeShipTriple(int x, int y, int orientation) const;
```
Stawia statek składający się z trzech pola.

```
bool placeShipQuadra(int x, int y, int orientation) const;
```
Stawia statek składający się z czterech pól.

Bot

Pola

Publiczne:

```
int nrRuchu=0;
```
Licznik wykonanych ruchów z tablicy ruchy[][].

```
int difficulty=0;
```
Zmienna zapamiętywująca poziom trudności bota.

```
int ruchy[50][2]={
        {3,0},{2,1},{1,0},{1,2},{0,1},{0,3},
        {0,5},{1,6},{0,7},{2,7},{1,8},{0,9},
        {3,8},{2,9},{4,9},{5,0},{6,1},{7,0},
        {7,2},{8,1},{9,0},{9,2},{8,3},{9,4},
        {9,6},{8,7},{9,8},{7,8},{8,9},{6,9},
        {5,8},{6,7},{7,6},{7,4},{8,5},{6,5},
        {5,6},{4,7},{3,6},{2,5},{1,4},{3,4},
        {2,3},{4,5},{5,4},{6,3},{4,3},{5,2},
        {3,2},{4,1}
                };
```
Tablica Przechowująca określoną kolejność oddawania strzałów – brak możliwości wypełnienia algorytmem.

Metody

Publiczne:

```
Bot();
```
Konstruktor Bota. Wypełnia tablicę dwuwymiarową, zawierającą strzały w określonej kolejności.

```
void takeShot(Board& board);
```
Zwraca uwagę na wybrany poziom trudności. Podejmuje decyzję, w jakie miejsce oddać strzał. W pierwszej kolejności sprawdza, czy ma bombę, następnie wybiera najoptymalniejsze miejsce na oddanie strzału na całej planszy. Jeśli nie ma bomby, używa rakiety, ponownie wybierając najbardziej optymalne miejsce na mapie. Jeśli nie ma ani bomby, ani rakiety, oddaje zwykły strzał według tablicy ruchy[50][20]. Gdy tablica zostanie „wyczerpana”, oddaje losowe strzały. Funkcja się zapętla, jeśli zostanie oddany celny strzał.

Najbardziej optymalne miejsce na mapie – miejsce, po strzale w które zostanie odkrytych najwięcej pól.

Private:

```
void randomShot(Board &board);
```
Oddaje strzał w losowe miejsce na planszy.

Game

Pola

Publiczne:

```
Board board1;
```

Przechowuje pierwszą tablicę. (gracza)

```
Board board2;
```

Przechowuje drugą tablicę. (bota)

Metody

Publiczne:

```
Game(const Board& board1, const Board& board2) : board1(board1), board2(board2)
```

Konstruktor gry; przyjmuje za argumenty dwie tablice.

```
bool isGameWon();
```

Sprawdza, czy gra została zakończona, tzn. czy ilość żywych statków na którejkolwiek planszy jest równa 0.

```
void printWinner();
```

Jeśli gra została wygrana, funkcja drukuje na ekranie komunikat o wygranej/przegranej gracza.

Menu

Pola

Publiczne:

```
Game game;
```
Przechowuje oryginał obiektu game.

```
Bot bot;
```
Przechowuje oryginał obiektu bot.

```
int trybGry;
```
Zapamiętuje tryb gry: 0-gra z botem, 1-gra dwóch graczy.

Metody

Publiczne:

```
Menu(const Game& game):game(game)
```
Konstruktor przyjmujący za argument obiekt game.

```
void runGame();
```
Metoda odpowiada za wyświetlenie ekranu tytułowego, wyświetlenie main menu, przyjmuje od użytkownika wszystkie parametry o grze (np. poziom trudności bota, tryb gry) oraz steruje wypełnieniem plansz statkami. Odpowiada za wyświetlanie i sterowanie całym przebiegiem gry.

```
void printGame();
```
Metoda wywołuje dwie inne metody; printBoards() i printChoiceMenu().

```
void printBoards();
```
Metoda drukuje obok siebie w konsoli stan obu plansz.

```
void printChoiceMenu();
```
Metoda steruje działaniami gracza; czy używa zwykłego strzału, bomby itp.

```
bool takeAction(Board &board, string choose);
```
Metoda jest elementem weryfikującym prawidłowość danych wpisanych w printChoiceMenu() a następnie komunikuje się i steruje obiema planszami.

```
void mainMenu();
```
Metoda drukująca ekran wyboru trybu gry i ew. poziomu trudności bota. W zależności od parametrów modyfikuje pola.

```
void firstScreen();
```
Wywołuje printMainTitle().

```
void printInstruction();
```
Metoda drukuje w konsoli instrukcje.

```
void printMainTitle();
```
Wypisuje napis “Ships game” I czeka na wciśnięcie klawisza aby wystartować gre.