[AiSD] Projekt Zaliczeniowy – Sebastian Sęczyk

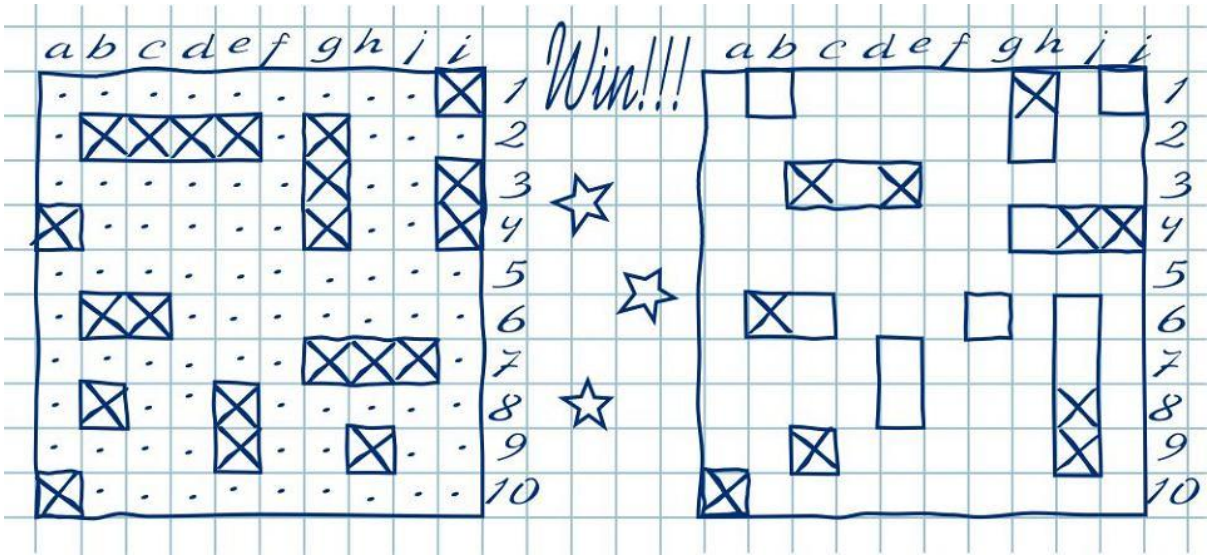
(Gra w Statki)

1. Wprowadzenie:

„Gra w Statki” to jedna ze starszych, prostych gier. Polega ona na zniszczeniu wszystkich statków przeciwnika. Gracze na początku ustawiają swoje statki w dowolny sposób na planszy $\mathbf{10} \times \mathbf{10}$ zachowując przy tym kilka zasad (np. statki muszą być rozstawione **minimum 1** kratkę odległości od siebie).

Wariantów Gry w Statki jest wiele, natomiast jednym z popularniejszych jest wariant, w którym każdy z graczy posiada **10 statków** odpowiednio:

- 1 o rozmiarze 4,
- 2 o rozmiarze 3,
- 3 o rozmiarze 2,
- 4 o rozmiarze 1.



Oprócz zwykłych statków do programu dodana została również funkcja **miny morskiej**, która zostaje utworzona w losowym miejscu po ustawieniu statków. Po jej trafieniu wszystko w obszarze $\mathbf{3} \times \mathbf{3}$ od jej środka wybucha (zostaje trafione).

2. Algorytmy:

2.1. Losowy układ statków

Algorytm tworzenia i ustawiania statków działa na następującej zasadzie. Dla danego rozmiaru statku losuje (*w pionie lub w poziomie*) współrzędne w taki sposób, aby statek **zmieścił się** na planszy.

```
int whichWay = rand() % 2;

if( whichWay == 0 ) {   //  horizontal

    Losowe koordynaty
    char X = 'A' + rand() % (10-shipNumber);
    int  Y = rand() % 10;
    Coordinates cords( X, Y );
```

Następnie sprawdza czy na wylosowanych współrzędnych można ustawić statek. Jeśli współrzędne te są już przez coś zajęte, zostają wylosowane **nowe koordynaty**.

Jeśli okaże się, że statek może zostać tam ustawiony to we wcześniej *„wyzerowanej”* tablicy **bool** w miejscu statku oraz na jego „obrzeżach” zajmuję miejsce jako zajęte przez statek.

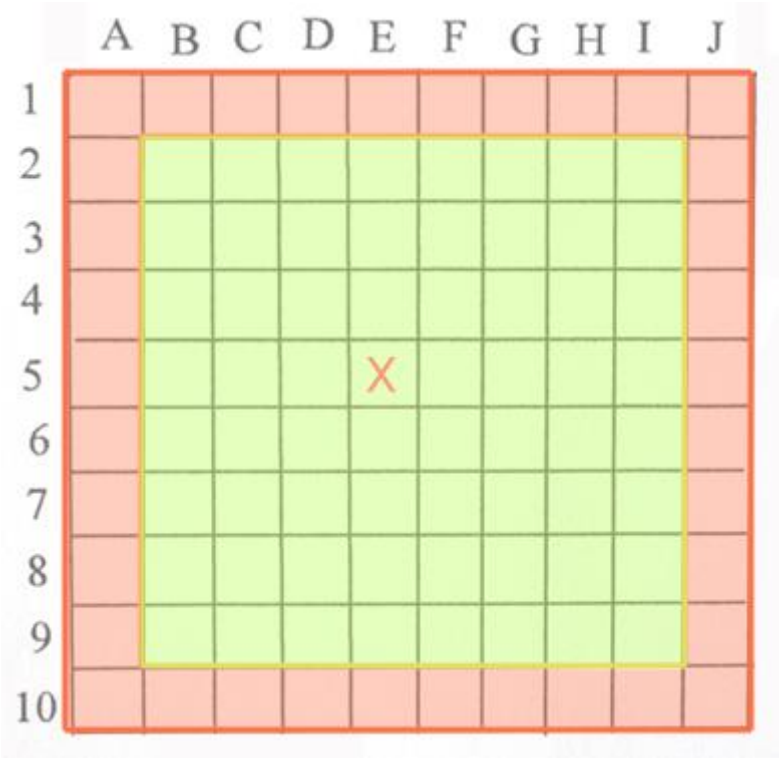
```
    |~~~~~~~~~~~~~~STATKI WROGA~~~~~~~~~~~~~~|
    | A | B | C | D | E | F | G | H | I | J |
=============================================
 1 |   |   |   |   |   |   |   |   |   |   |
 2 |   |   |   |   |   |   |   |   |   |   |
 3 |   |   |   |   |   |   |   |   |   |   |
 4 |   |   |   |   |   |   |   |   |   |   |
 5 |   |   |   |   |   |   |   |   | 4 |   |
 6 |   |   |   |   |   |   |   |   | 4 |   |
 7 |   |   |   |   |   |   |   |   | 4 |   |
 8 |   |   |   |   |   |   |   |   | 4 |   |
 9 |   |   |   |   |   |   |   |   |   |   |
10 |   |   | X |   |   |   |   |   |   |   |
=============================================
    | Player:   1                Enemy:   1 |

=============================================


=============================================

    |~~~~~~~~~~~~~~STATKI WROGA~~~~~~~~~~~~~~|
    | A | B | C | D | E | F | G | H | I | J |
=============================================
 1 |   |   |   |   |   |   |   |   |   |   |
 2 |   |   |   |   |   |   |   |   |   |   |
 3 |   |   |   |   |   |   |   |   |   |   |
 4 |   |   |   |   |   |   |   | 1 | 1 | 1 |
 5 |   |   |   |   |   |   |   | 1 | 1 | 1 |
 6 |   |   |   |   |   |   |   | 1 | 1 | 1 |
 7 |   |   |   |   |   |   |   | 1 | 1 | 1 |
 8 |   |   |   |   |   |   |   | 1 | 1 | 1 |
 9 |   |   |   |   |   |   |   | 1 | 1 | 1 |
10 |   |   |   |   |   |   |   |   |   |   |
=============================================
```

Następnie utworzony statek zostaje dodany do **wektora** przechowującego wszystkie statki gracza, po czym zostaje położony na planszy. Algorytm tworzy w ten sposób statki dopóki nie ustawi **wszystkich** wymaganych na planszy.

2.2. *Generowanie Miny Morskiej*

Po ustawieniu **wszystkich** statków zostaje wygenerowana **mina morska**. Jest to obiekt o współrzędnych różnych od współrzędnych statków na planszy danego gracza oraz różnych od brzegów planszy.

Oznacza to, że mina morska **nie może** zostać wygenerowana na polu statku lub „na plaży” (ponieważ jest tam za płytko :)).



** Przykładowe wygenerowanie miny morskiej*

2.3. *Przeciwnik AI*

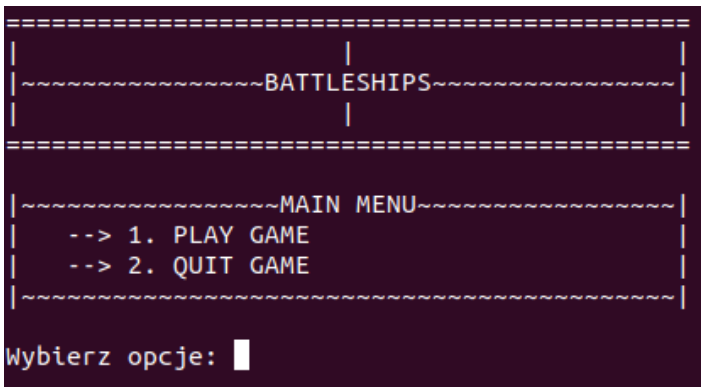
Program posiada wyłącznie tryb ***„Player VS AI”***. AI podczas swojej tury wybiera losowo koordynaty do strzału. Nie uwzględnia on poprzednich strzałów.

```
Coordinates randomShot() {

    char X = 'A' + rand() % 10;
    int Y  = rand() % 10;

    Coordinates shot( X, Y );

    return shot;
}
```

3. Rozgrywka:

3.1. Menu Główne

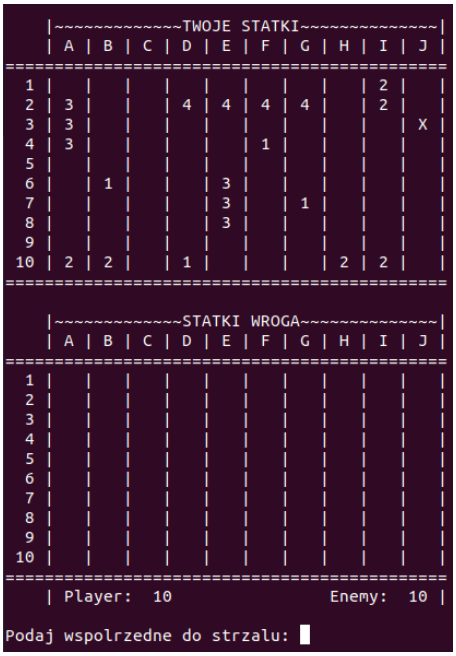
Po uruchomieniu programu zostanie wyświetlone **Menu Główne**. Należy wprowadzić *numer akcji*, którą chcemy wykonać. Jeśli *input* będzie inny niż numery pokazanych akcji, program zakończy swoje działanie.



- Po wybraniu **opcji 1.** program rozpocznie rozgrywkę.
- Po wybraniu **opcji 2.** program zakończy swoje działanie.

3.2. Ekran rozgrywki

Po wybraniu **opcji 1.** w *Menu Głównym* wyświetlone zostaną 2 plansze oraz informacja i ilości niezatopionych statków gracza, jak i wroga.



3.2.1. Wprowadzanie współrzędnych

Program będzie czekał aż dostanie *input* o współrzędnych do **strzału** na planszę wroga. Najpierw należy podać **współrzędną X** (*górna oś*), a potem **współrzędną Y** (*pionowa oś*). Współrzędną X **MUSI** być wprowadzona w formie **DUŻEJ** lub **MAŁEJ** **LITERY** znajdującej się w przedziale $\langle A, J \rangle$. Współrzędna Y **MUSI** natomiast znajdować się w przedziale $\langle 1, 10 \rangle$.

Przykład poprawnego wprowadzenia współrzędnych:

```
Podaj wspolrzedne do strzalu: D7
```

Jeśli podamy nieprawidłowe współrzędne program zakończy swoje działanie:

```
Podaj wspolrzedne do strzalu: T34

Podano nieprawidlowe wspolrzedne!

battleships.x: battleships.h:423: Coordinates Board::wait4move(): Assertion `(in
t)x >= 65 && (int)x <= 74 && y >= 1 && y <= 10' failed.
make: *** [Makefile:25: game] Aborted (core dumped)
```

3.2.2. Strzelanie i zatapianie

Jeśli wprowadzone prawidłowo współrzędne **będą równe** współrzędnym statku (lub części statku) przeciwnika, statek ten otrzyma „obrażenia” przy czym na planszy *„Statki Wroga”* w miejscu strzału zostanie wstawiony znak ***„x”*** oraz zostanie wyświetlony komunikat o *statusie akcji*.

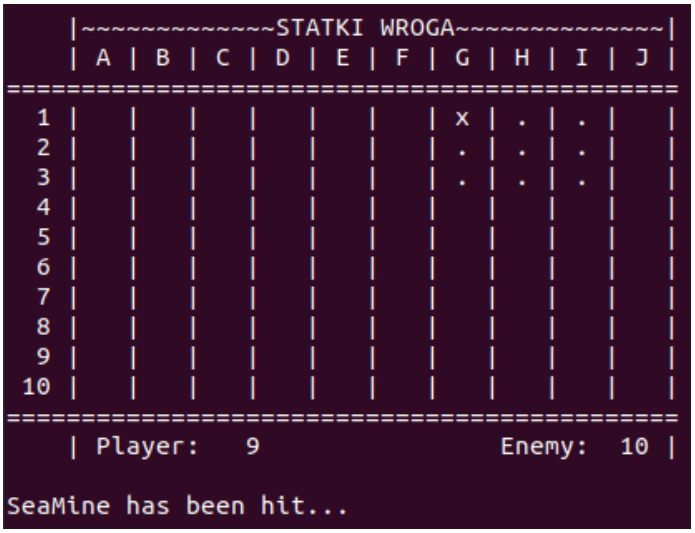
```
    |~~~~~~~~~~~~~~~STATKI WROGA~~~~~~~~~~~~~~~|
    | A | B | C | D | E | F | G | H | I | J |
=============================================
  1 | x |   |   |   |   |   |   |   |   |   |
  2 |   |   |   | . |   |   |   |   |   |   |
  3 |   |   |   |   |   |   |   |   |   |   |
  4 |   |   |   |   |   |   |   |   |   |   |
  5 |   |   |   |   |   |   |   |   |   |   |
  6 |   |   |   |   |   |   |   |   |   |   |
  7 |   |   |   |   |   |   |   |   |   |   |
  8 |   |   |   |   |   |   |   |   |   |   |
  9 |   |   |   |   |   |   |   |   |   |   |
 10 |   |   |   |   |   |   |   |   |   |   |
=============================================
    | Player:  10                Enemy:  10 |

Hit...
```

Jeśli dany statek zostanie trafiony taką ilość razy jakiego jest rozmiaru to zostaje **zatopiony**. Po zatopieniu statku plansza zostaje zaktualizowana w taki sposób, aby wykluczyć miejsca, w których już **na pewno** żadnego statku nie będzie (zasada **minimum 1** kratki odstępu). Aktualizuje się także *wektor* przechowujący statki wroga jak i *status akcji*.

* Po trafieniu statku jak i po zatopieniu aktywny gracz posiada dodatkowy ruch w turze.

3.2.2.1. Mina Morska

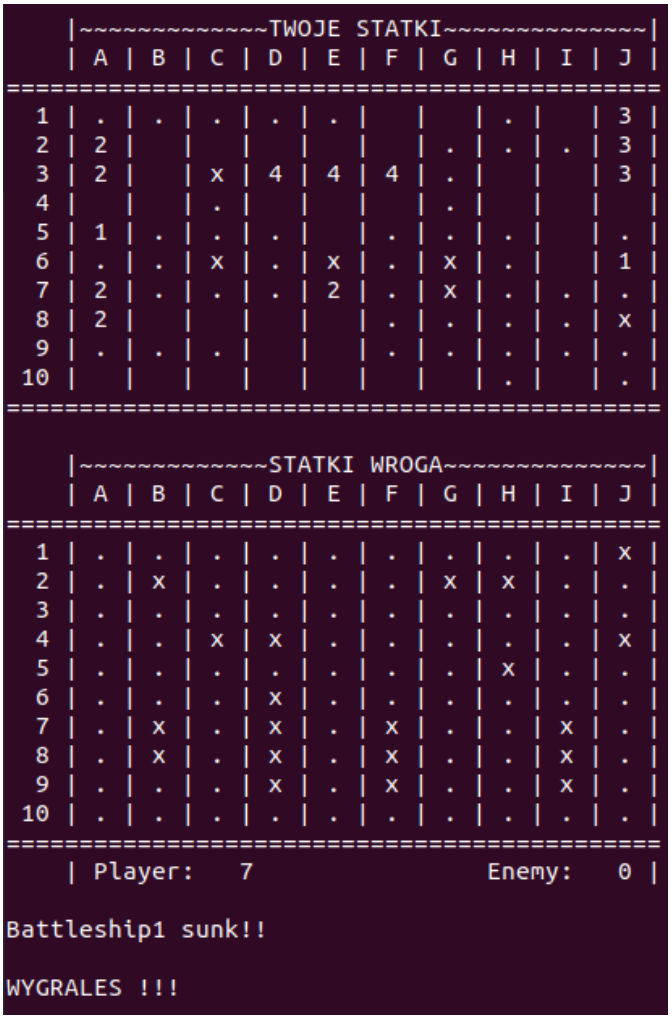
Jeśli wprowadzone współrzędne **będą równe** współrzędnym miny morskiej przeciwnika, zostanie ona *zdetonowana*. Detonacja miny morskiej niszczy **wszystko** co znajduje się w jej obszarze eksplozji (obszar ten wynosi $\mathbf{3} \times \mathbf{3}$).



* Trafienie w *minę morską* **nie daje** dodatkowego strzału dla gracza, który ją trafił (nawet jeśli któryś ze statków *otrzymał obrażenia* lub *został zatopiony*).

3.2.3. Zakończenie rozgrywki

Rozgrywka zostaje zakończona, jeśli jeden z graczy **nie posiada** żadnych niezatopionych statków. Rozgrywkę wygrywa ten, któremu udało zatopić się wszystkie statki przeciwnika.



4. Obiekty:

*4.1. Klasa **Coordinates***:

Klasa **Coordinates** przechowuje 2 zmienne: *char* oraz *int*. Posiada również metody je ustawiające i zwracające (**get(), set()**).

```
class Coordinates {

    char x;
    int y;

public:

//  Konstruktor
    Coordinates( char X, int Y ) {

        x = X;
        y = Y;
    }
```

*4.2. Klasa **Ship***:

Klasa **Ship** posiada: *2 zmienne wektorowe**<int>*** przechowujące współrzędne danych części statku, *3 zmienne int* (rozmiar, ilość trafień oraz w jaki sposób statek jest położony na mapie) oraz *1 string*, czyli nazwę statku.

```
class Ship {

    vector<int> x;
    vector<int> y;
    int size;
    int hit;
    int way;
    string name;
```

Oprócz tego klasa ta posiada również **metody zwracające** dane zmienne, jak i zwracanie nazwy statku w podanych do funkcji współrzędnych, jeśli jakaś jego część znajduje się właśnie na tych koordynatach.

*4.3. Klasa **Board***:

Klasa Board zawiera główną część programu. Posiada 3 tablice znakowe o rozmiarze $\mathbf{10} \times \mathbf{10}$, które przechowują odpowiednie znaki dla danego pola, 2 zmienne bool (sprawdzające, czy cokolwiek zostało zatopione oraz kto ma ruch w danej turze), 1 string będący *statusem akcji*, 2 wektory**<Ship>** (dla gracza oraz przeciwnika) przechowujące wszystkie wygenerowane statki oraz 2 zmienne **Coordinates** będące współrzędnymi **miny morskiej** gracza oraz przeciwnika.

```
class Board {

    char playerSquare[SIZE][SIZE];
    char enemySquare[SIZE][SIZE];
    char enemySquareShown[SIZE][SIZE];

    bool playing = true;
    bool didSunk = false;

    vector<Ship> playerShipsVector;
    vector<Ship> enemyShipsVector;

    string status = "";

    Coordinates playerSeaMine;
    Coordinates enemySeaMine;
```

Klasa ta posiada wszystkie metody potrzebne do rozgrywki m.in.:

- **Void makeShips()** – funkcja tworząca odpowiednią ilość statków oraz min morskich dla danego gracza.
- **Coordinates wait4move()** – funkcja czekająca na input od gracza.
- **Void fire()** – funkcja, która, w zależności od aktywnego gracza, „strzela” w dane współrzędne.
- **Void getStatus()** – funkcja zwracająca aktualny status akcji.
- **Void displayBoard()** – funkcja wyświetlająca przebieg rozgrywki na ekran.