Paulina Nieradzik

Politechnika Śląska

Wydział Informatyki, Elektroniki i Informatyki

Programowanie Komputerów 4

Statki

---

autor **Paulina Nieradzik**

prowadzący dr inż. Piotr Pecka

rok akademicki **2020/2021**

kierunek **informatyka**

rodzaj studiów **SSI**

semestr **4**

termin laboratorium środa, **13:30-15:00**

sekcja **12**

---

1 Treść zadania

Zaimplementować grę w statki dla jednego gracza - przeciwnikiem jest prosta sztuczna inteligencja (algorytmy komputera zgodne z logiką gry).

Tematem mojego projektu jest wykonanie programu umożliwiającego grę w statki. W projekcie są wykorzystywane proste kształty bądź grafika. Program jest uruchamiany z konsoli lecz cała gra będzie odbywać się w osobno otworzonym oknie. Celem projektu było stworzenie gry „planszowej", w której rozgrywka będzie się odbywała pomiędzy graczem(człowiekiem), a komputerem z prostą sztuczną inteligencją – ruchy komputera nie są całkowicie losowe, lecz w momentach gry, gdy jest to możliwe odbywają się one zgodnie z logiką – ze względu na sam typ gry, która polega na odgadnięciu gdzie położone są statki przeciwnika nie zawsze będzie możliwy „przemyślany" ruch. Po wybraniu odpowiednich opcji w menu wyświetlają się dwie plansze. Program umożliwia graczowi rozłożenie swoich statków na swojej planszy oraz rozmieszcza zgodnie z zasadami gry statki komputera. Następnie rozpoczyna się rozgrywka, program sprawdza czy udało się trafić w statek przeciwnika, a jeżeli tak to czy udało się go zatopić.

2 Analiza zadania

Struktury danych

W programie wykorzystano wektory przechowujące tekstury wyświetlanych elementów, czcionki. Innym wykorzystanym kontenerem jest lista jednokierunkowa która zawiera statki ułożone na odpowiedniej planszy. Sama plansza jest zapisana za pomocą tablicy array również z biblioteki STL. W programie wykorzystano również iteratory tej biblioteki. Często wykorzystywane były również inteligentne wskaźniki.

Algorytmy

Algorytm losowego ruchu komputera i algorytm „logiki".

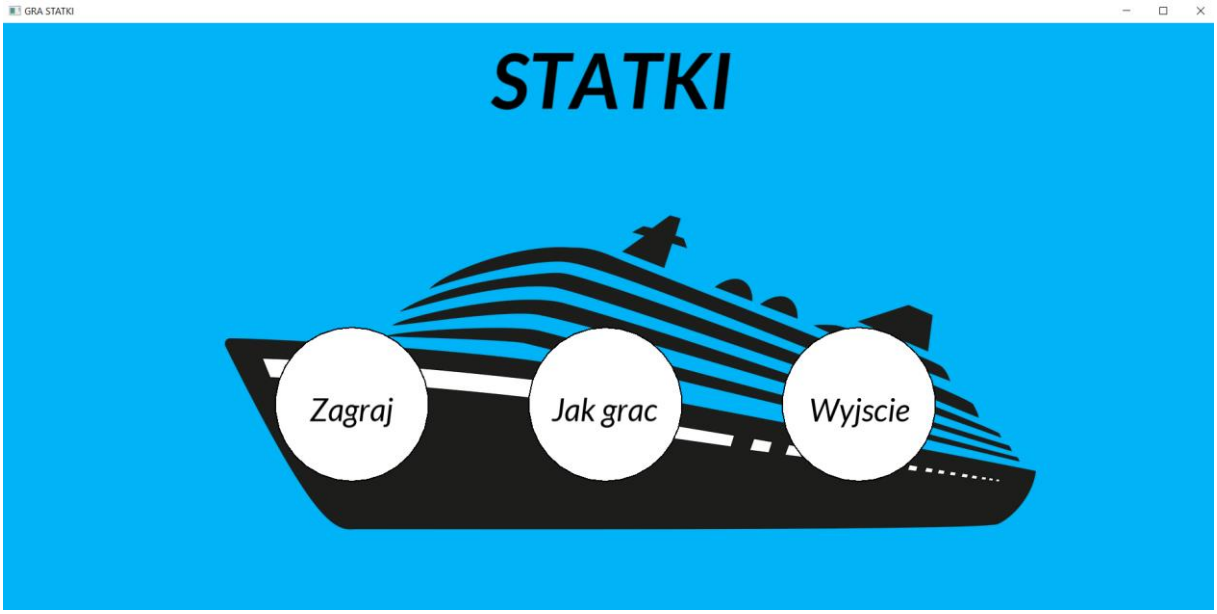
Pole, w które chce strzelić komputer jest polem wybranym losowo (w zależności od obecnego czasu), jeżeli będzie możliwy logiczny ruch to będzie on wykonywany według odpowiedniego algorytmu.

Rozgrywka kończy się gdy po przejściu całej listy statków na planszy przeciwnika gracza który wykonał ruch nie znajduje się statek który pole zatopiony ma równe logicznej wartości false. Jeżeli tak się stanie to usuwane są plansze wypisuje się

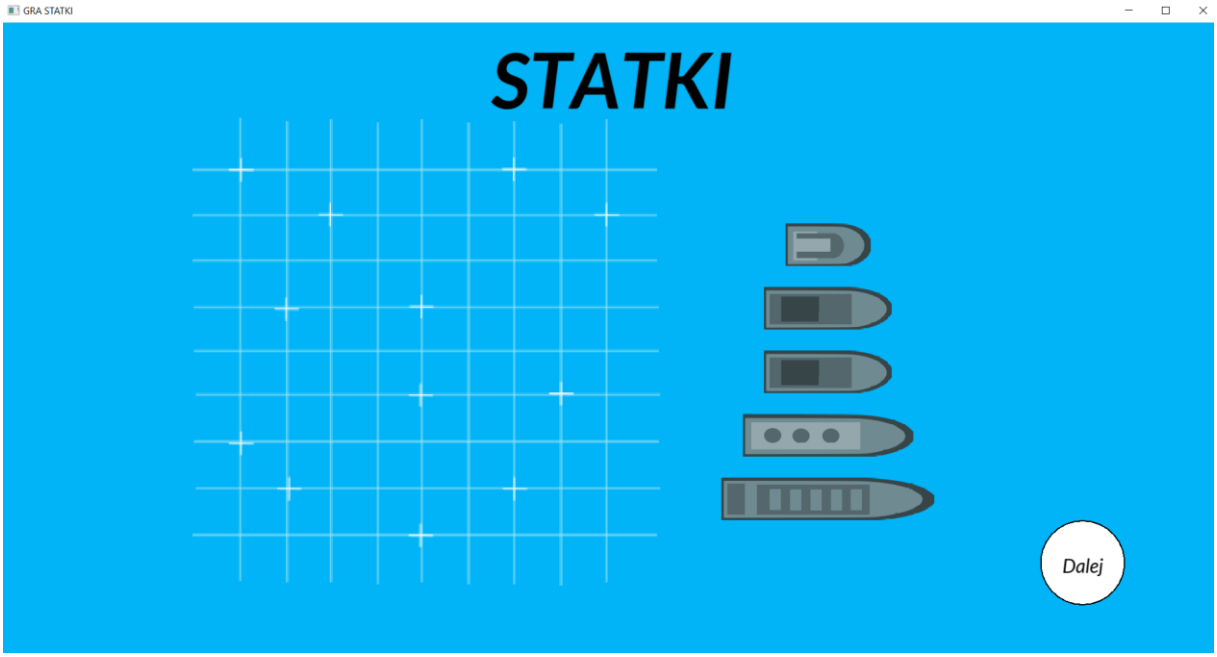
komunikat o rezultacie rozgrywki a program jest przygotowany na rozegranie nowej gry.

3 Specyfikacja zewnętrzna

Po uruchomieniu wyświetla się menu wraz z opcjami do wyboru przez gracza: chcę zagrać, jak grać oraz opcją wyjścia z gry. Opcja chcę zagrać przekieruje do początku gry, natomiast opcja jak zagrać wyświetli zasady i tłumaczy jak poruszać statkami. Przycisk działa w momencie naciśnięcia go myszką (kursorem).

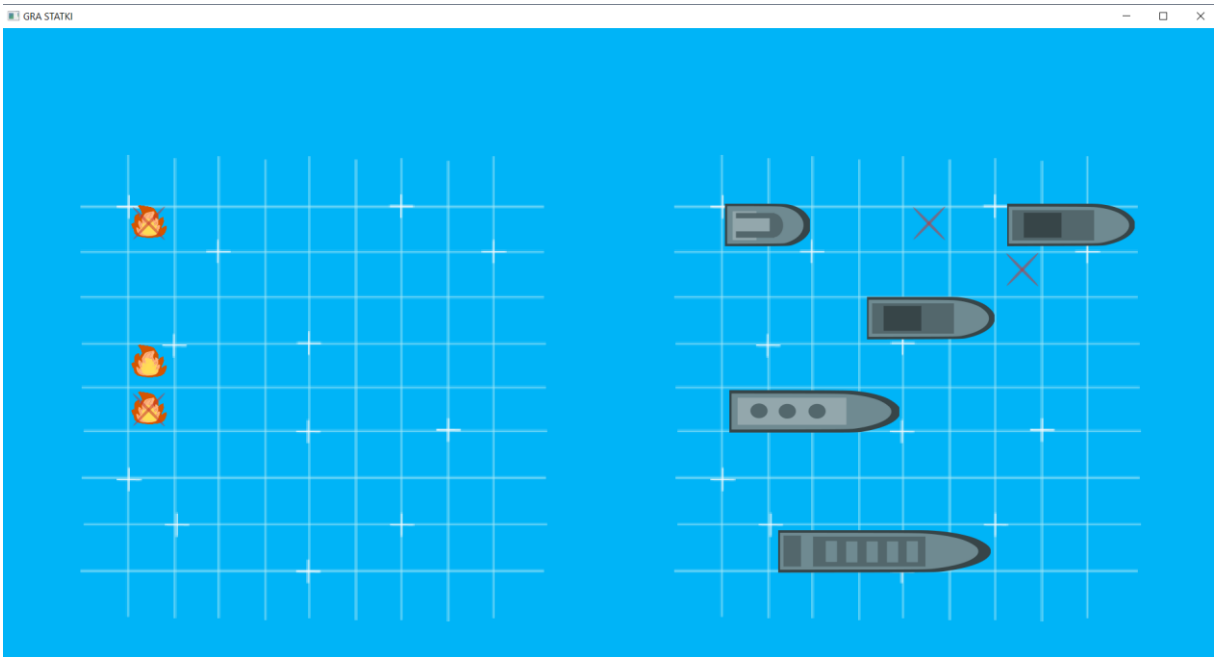


Początek gry: Po uruchomieniu trybu gry wyświetli się plansza i należy rozmieścić na niej statki poprzez przeciągnięcie ich na wybrane miejsce. Zmiana orientacji statku z poziomej na pionową odbywa się poprzez naciśnięcie spacji.



W przypadku złego rozstawienia statków i próby przejścia dalej wyświetla się następujący komunikat:

Gdy gracz rozmieści swoje statki na ekranie pojawiają się dwie plansze (jedna na której musi oddać on swój strzał a druga która ukazuje jakie ruchy wykonał przeciwnik (komputer):



Gra kończy się gdy któryś z graczy zatopi statki przeciwnika.

4 Specyfikacja wewnętrzna

Program został zrealizowany obiektowo.

W projekcie wykorzystane jest czternaście klas:

- Gra – zawiera wskaźnik na okno, w którym wyświetlana jest cała rozgrywka, wektor wskaźników na stany, wektor zapisujący położenie myszy na ekranie, wskaźnik na obiekt Zarzadzanie_gra oraz pole zmienna typu int. Posiada tylko jedną metodę gra – która jest odpowiedzialna za wyświetlanie stanów w odpowiedniej kolejności

```
class Gra
{
    sf::RenderWindow* okno = new sf::RenderWindow;
    std::vector<Stan_gry*> stany;
    sf::Vector2f pozycjaMyszy;
    int zmiana = 0;

public:
    Zarzadzanie_gra* zarzadzanie;
    Gra();
    ~Gra();
    void gra();
};
```

- Zarzadzanie_gra – klasa zawiera wektor struktur potrzebnych do wyświetlania gry, dwa wskaźniki na czcionki oraz wskaźniki na dwóch graczy. Metody tej klasy to ZaladujCzcionki oraz ZaladujTekstury.

```
class Zarzadzanie_gra
{
public:
    std::vector<sf::Texture*> tekstury;
    sf::Font* czcionka1, *czcionka2;
    std::unique_ptr<Czlowiek> czlowiek;
    std::unique_ptr<Komputer> komputer;

    Zarzadzanie_gra();
    ~Zarzadzanie_gra();
    bool ZaladujCzcionki();
    bool ZaladujTekstury();
};
```

- Stan_gry – klasa abstrakcyjna zawiera wskaźnik na okno, zmienną typu bool o nazwie zainicjowany (mówiąca o tym czy stan jest zainicjowany) oraz metody inicjalizuj, uaktualnij oraz rysuj.

```
class Stan_gry
{
public:
    sf::RenderWindow* okno = nullptr;
    bool zainicjowany = false;
    virtual void inicjalizuj(Zarzadzanie_gra*& z, sf::RenderWindow*& o)=0;
    virtual int uaktualnij(sf::RenderWindow*& o, sf::Event& e)=0;
    virtual void rysuj(sf::RenderWindow*& o)=0;
};
```

Bardzo podobnie zbudowane są klasy dziedziczące z klasy Stan_gry: klasa Start, Poczatek, Menu, Rozgrywka oraz Koniec

- Przycisk – klasa obsługująca i wyświetlająca przyciski potrzebne w grze, zawiera napis wyświetlany na poszczególnym przycisku, jego kolory oraz kształt. Metody sprawdzają czy przycisk został naciśnięty, rysują go oraz uaktywniają.

```
class Przycisk
{
    sf::CircleShape ksztalt;
    sf::Color kolor;
    sf::Color kolor2;
    sf::Text napis;
    sf::Font* czcionka;
    friend class Menu;
public:
    bool nacisniety = false;
    Przycisk(sf::Vector2f vec, sf::Color col, sf::Color col2, sf::Font* fon, std::string napis, float promien = 120.f, int rozmCzcionki=50);
    ~Przycisk() {};
    void rysuj(sf::RenderWindow*& o);
    void uaktywnij(sf::RenderWindow*& o);
    bool czyNad(sf::RenderWindow*& o);
};
```

- Statek – klasa zawierająca wszystkie potrzebne wartości do opisania statku- jego pozycji, wielkości, orientacji oraz informacji czy został on

zatopiony czy nie. Zawiera również wszystkie metody to poruszania statkiem.

```
class Statek
{
    friend class Komputer;
    friend class Czlowiek;
    friend class Gracz;
    friend class Poczatek;
    friend class Rozgrywka;
    char znak;
    int pozycja;
    int wielkosc = 0;
    int trafienia = 0;
    Orientacja orientacja = Orientacja::pozioma;
    bool zatopiony = false;
public:
    Statek();
    Statek(char znak, int wielkosc);
    sf::Sprite sprajt;
    void inicjalizuj(sf::Texture*& tekstura);
    void rysuj(sf::RenderWindow*& okno);
    bool czyZatopiony();
    void ustawOrientacje(int o);
    void zmienOrientacje();
    void Trafiony(sf::RenderWindow* okno);
    bool sprawdzeniePozycji(sf::RenderWindow*& o);
};
```

- Plansza – klasa zawierająca dwie tablice na której zapisane jest rozmieszczenie statków oraz metody sprawdzające wykonywane ruchy.

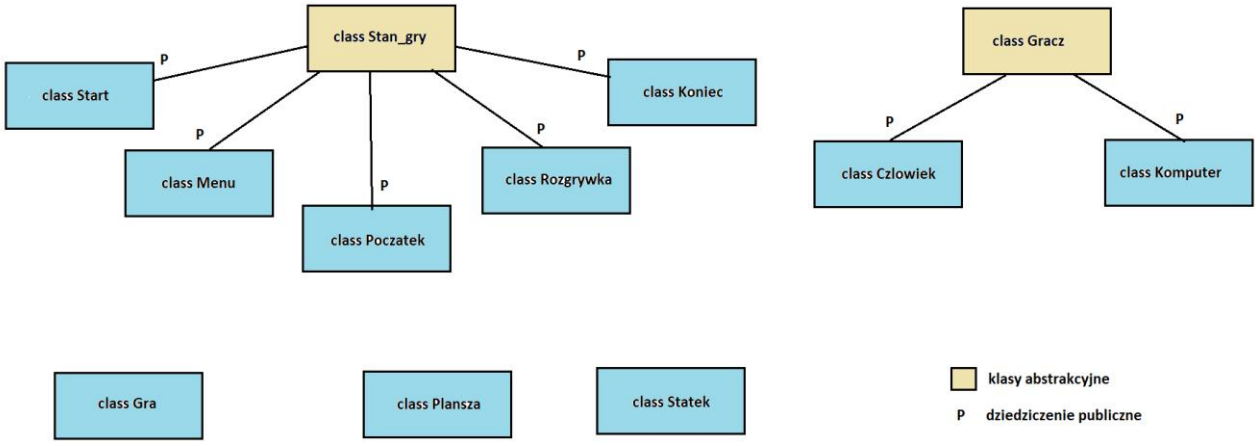
```
class Plansza
{
    friend class Rozgrywka;
    friend class Komputer;
    friend class Czlowiek;
    std::array<std::array<char,10>,10> mojaPlansza;
    std::array<std::array<char,10>,10> planszaPrzeciwnika;
public:
    Plansza();
    bool sprawdzCzyMoznaUstawic(int pozycja, int wielkosc, char znak, int orientacja);
    void zaznaczNaPlanszy(int x, int y, char znak, int wielkosc, int orintacja);
    void wyczyscPlansze();
    char CzyTrafiony(int ruch);
};
```

- Gracz – klasa abstrakcyjna zawierająca wskaźnik na plansze oraz listę statków gracza raz stan poprzedniego ruchu. Metody tej klasy sprawdzają czy jest koniec rozgrywki oraz czy gracz trafił w statek przeciwnika.

```
class Gracz
{
protected:
    std::unique_ptr<Plansza> plansza;
    std::unique_ptr<std::list<Statek>>listaStatkow;
    bool poprzedniTrafiony = false;
public:
    Gracz();
    ~Gracz() {};
    bool KoniecGry();
    void Trafiony(char znak);
};
```

W programie zostaną wykorzystane na pewno klasy: gra, gracz z której będą dziedziczyć klasy czlowiek i komputer. Kolejną klasą będzie klasa plansza, statek. Główna klasą w całym programie jest klasa gra to w niej będzie utworzone wyświetlane okno, zapisane tekstury, czcionki, obrazy wykorzystywane do tworzenia gry. Klasa Stan_gry jest klasą abstrakcyjną, z której dziedziczą klasy odpowiedzialne za poszczególne fragmenty gry. Klasa Start wyświetla nazwę gry, klasa Menu wyświetla menu i umożliwia wybranie odpowiedniej opcji gry. Klasa Początek służy do rozłożenia elementów na planszy, w niej też będzie wykonywany algorytm wybierania miejsca na statki komputera. Klasa rozgrywka zawiera elementy i metody potrzebne do rozgrywki. Klasa koniec będzie kończyć grę i jeżeli będzie to konieczne zapisywać stan niedokończonej gry.

Diagram klas przedstawiony poniżej:



4.1 Ogólna struktura programu

W funkcji main programu tworzony jest obiekt typu gra, w którym wywoływane są po kolei wszystkie stany gry, a w nich odpowiednie metody obiektów tych klas lub klas pochodnych od klasy gracz.

Na początku wyświetla się stan o nazwie Start – jest on tak na prawdę intrem gry, z którego wychodzi się naciskając przycisk myszy. Po nim wyświetla się menu z trzema opcjami do wyboru. Wybór opcji jak grać uruchamia odpowiednią metodę tej klasy. Wyjście zamyka cały program, natomiast wybranie opcji zagraj sprawia że w obiekcie typu gra tworzony jest nowy stan – jest nim stan Poczatek (gry). To w nim wywoływane są wszystkie metody odpowiedzialne za rozmieszczenie statków i sprawdzenie poprawności tego rozmieszczenia. Następnym stanem jest stan Gra - to on odpowiada bezpośrednio za rozgrywkę.

Gra zakończy się gdy jeden z graczy wygra i zatopi wszystkie statki przeciwnika.

5. Testowanie

Program został przetestowany poprzez odbycie kilku rozgrywek gry w statki z jego wykorzystaniem. Jest odporny na złe ustawienie statków przez gracza. W momencie kiedy statki nie są ustawione poprawnie wyświetla się stosowny komunikat, a gracz musi poprawić ustawienie statków. Nie jest możliwe przejście dalej. Po skończonej rozgrywce możliwe jest rozpoczęcie nowej.

6. Wnioski

Program do gry w statki nie jest programem trudnym jednak należało dobrze przemyśleć za co odpowiedzialne poszczególne klasy aby było on logicznie napisany i zrozumiały dla innych. Najtrudniejsze okazało się napisanie programu tak aby dobrze odczytywał ruchy wykonane przez gracza i sprawdzał czy są one poprawne. Napisanie programu z użyciem prostej biblioteki graficznej jaką jest SFML na pewno pokazało nowe możliwości wykorzystania języka jakim jest C++, a pisanie programu obiektowo pozwoliło na odpowiednie rozplanowanie programu.

Literatura

- „Język C++ Szkoła programowania” Stephen Prata
- https://miroslawzelent.pl/kurs-obiektowy-c++/podejscie-obiektowe-obiekt-klasa/
- http://cpp0x.pl/kursy/Kurs-C++/1
- https://www.sfml-dev.org/