POLITECHNIKA RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Projekt z przedmiotu
Programowanie w języku c++

Temat:

Statki

Opracowanie:

Marcin Wiśniowski

Rok: 2EF-DI
Grupa laboratoryjna: L02
Data: 13.01.2024

Dokumentacja techniczna

Spis treści

Klasy i metody

1. MainWindow

Najbardziej rozbudowana klasa służąca obsłudze okna, wpisywania danych przez użytkownika, grze w statki, za obsługę front-endu odpowiada technologia Qt dla języka programowania c++. Klasa ta została podzielona na 3 części, public, private oraz private slots służąca obsłudze przycisków.

a. MainWindow(QWidget *parent = nullptr); ~MainWindow();

Konstruktor klasy MainWindow, inicjalizuje potrzebne obiekty, zmienne oraz vectory zawierające przyciski i etykiety, ukrywa niektóre widgety, łączy bota z regulacją czasu w celu późniejszego sterowania nim, a także wywołana jest funkcja *srand(time(NULL)),* w celu zapewnienia losowych liczb. Dekonstruktor po wszystkich operacjach ustawia zmienne na wartości domyślne oraz usuwa zainicjalizowane obiekty

b. void initialize_Labels(); void initialize_Border(); void initialize_Border_GP(); void initialize_Border_GP_1(); void initialize_Buttons_GP(); void initialize_Buttons_GP_1();

Wszystkie 6 funkcji inicjalizuje wektory zawierające przyciski lub etykiety znajdujące się w GUI całej aplikacji na różnych widżetach, w celu późniejszego dynamicznego odwoływania się do nich. Ich dołączenie do wektora następuje poprzez wyszukanie odpowiedniej nazwy obiektu znajdującego się w aplikacji. Przykładowo poszukujemy w pętli pola *Line Edit* oznaczonego nazwą *lineEdit_i,* gdzie *i* to odpowiedni numer indeksu pola edycyjnego funkcją *findChild* oraz dopisujemy go do wektora. Dzięki takiej operacji możemy przez nazwę wektora i numer indeksu odwoływać się do odpowiednich pól. W przypadku przycisków ich przypisanie przeprowadzane jest analogicznie oraz występuje także podłączenie do nich odpowiedniego emitowanego sygnału i reakcja na niego oraz funkcja *lambda* wykonywana po jego wciśnięciu.

c. void resetGame();

Metoda przywracająca stan początkowy dla wszystkich elementów front i back-endu. Resetuje ona tablicę na której układane są statki poprzez wywołanie metody *„set_to_defaults()”,* a także ustawia odpowiednie parametry dla przycisków używanych do wprowadzania statków, czyści pola edycyjne i zmienia arkusze stylów dla przycisków służących do gry oraz pól wyświetlających jej postęp.

d. void cords_for_bot(); void gen_Ships_for_players();

Metody, w którą zaimplementowany został algorytm losowego ustawiania statków, poprzez użycie pętli oraz instrukcji warunkowych oraz metod klasy *Resources* sprawdzających poprawność układania statków. Pętla iteruje od 0 do 20, przedziałami są ustawiane statki dla indeksu < 4 są to statki 1 masztowe, indeks < 7 statki 2 masztowe, indeks < 9 statki 3 masztowe indeks równy 10 statek 4 masztowy. Używając funkcji *rand % 10*, losowane są liczby z zakresu <0; 9>, w następnym kroku sprawdzane jest czy możliwe jest ich ustawienie w tablicy poprzez funkcje *check_the_ships* oraz *test_correct_positioning,* przed wykonaniem tej funkcji sprawdzany jest również wybór użytkownika czy statek 4 masztowy może mieć postać litery L. Obie funkcje zwracają logiczne wartości poprzez użycie odpowiedniej ilości flag dodawany jest statek oraz pętla przechodzi do kolejnej iteracji. Oba te algorytmy działają identycznie zarówno jak i dla użytkownika jak i dla bota, jedyną różnicą między nimi jest wypisanie w odpowiednich miejscach i pokazanie graficznie ułożenia statków dla użytkownika.

e. void shot_for_player(); void shot_for_second_player();

Metody wywoływane przy wciśnięciu przycisku przez gracza służące do obsługi strzału oraz sprawdzające czy aktualna gra jest przeprowadzana z komputerem czy innym graczem. Działając na odpowiednich obiektach sprawdzają czy dany strzał był w puste pole czy maszt statku. Jeżeli strzał był nietrafiony pole, w które trafiliśmy zmienia kolor na szary i wywoływana jest metoda strzału bota dla 1 wymienionej w nagłówku funkcji lub strzału dla 2 drugiego gracza dla drugiej. Przy trafionym strzale sprawdzany jest również warunek zakończenia gry oraz obliczane punkty, metodą *calculate_points* i wypisywane w odpowiedniej klatce GUI. Sprawdzany jest również trafiony statek, jeżeli jest on trafiony i zatopiony pola wokół niego są odpowiednio kolorowane, następnie możliwe jest ponowne oddanie strzału. W obydwu przypadkach gry jedno i wieloosobowej zastosowany jest delay, w pierwszym przypadku wywołuje on

funkcje strzału bota, w drugim funkcje lambda która opóźnia pokazanie GUI dla strzału drugiego gracza, oraz obydwie wyświetlają na ekran liczbę oddanych strzałów oraz aktualną turę gracza.

f. void shot_for_bot();

Metoda losująca i sprawdzająca strzał dla komputera, gdy strzał jest oddany w puste pole kolorowane jest ono tak jak w poprzednim przypadku na kolor szary i odblokowywana jest możliwość strzału gracza. Gdy strzał oddany jest w maszt statku sprawdzone zostaje czy jest on cały zatopiony oraz strzał jest powtarzany, przy ograniczeniu terenu dla komputera. Gdy flaga *lastShotHit* ustawiona jest na wartość *true* komputer przeszukuje kwadrat 3x3 wokół koordynat aktualnego masztu, gdy zostanie pierwszy maszt znaleziony następuje kolejny strzał w jego koordynaty. W przypadku gdy statek nie ma innych masztów do trafienia flaga resetuje swoją wartość oraz wywołuje ponownie funkcje strzału dla bota i wartości dla strzału losowane są z całej planszy 9x9. Gdy bot ponownie trafi w wcześniej trafione miejsce strzał jest powtarzany. Sprawdzany jest także warunek zakończenia gry oraz czy trafiony statek jest w całości zatopiony, jeżeli tak, pola wokół niego są kolorowane na szaro.

g. vector<pair<int, int>> check_fields(int xCord, int yCord, resources* object);

Metoda sprawdzająca czy aktualnie trafiony maszt jest ostatnim do zestrzelenia. Zwraca ona wektor par do którego są dodawane koordynaty masztów sprawdzanego statku w przypadku, gdy jest on zatopiony, algorytm sprawdza pola w pionie i poziomie, gdy zostanie napotkany niezatopiony maszt zwracany jest pusty wektor. Po sprawdzeniu pozycji sprawdzane są koordynaty pod względem ich wystąpienia w wektorze w celu wyeliminowania powtórzeń.

h. bool check_ships();

Metoda służąca do sprawdzenia czy wszystkie pola do wpisywania statków zostały wypełnione, wykorzystywana w celu niedoprowadzenia do powstania gracza bez ustawionym statków. Pętlą sprawdzane są wszystkie pola edycyjne, jeżeli któreś jest puste zwracana jest wartość *false* i nie możliwe jest przejście do gry w statki.

i. void on_pushButton_Ch_1_clicked(); void on_pushButton_Ch_2_clicked();

Metody do obsługi przycisków wyboru trybu gry, zostają one wywołane po wciśnięciu odpowiedniego przycisku. Chowają odpowiednie widżety GUI, a także ustawiają odpowiedni tytuł w etykiecie, placeholdery dla pól edycyjnych, a także wartość zmiennej ukazującej wybrany tryb gry.

j. void on_pushButton_TB_clicked(); void on_pushButton_EG_2_clicked();

Metody wywołujące przeczytanie danych z pliku i wyświetlenie na ekranie aktualnej tablicy wyników.

k. void on_pushButton_NS_1_clicked(); void on_pushButton_clicked();

Metody wywoławany przy wciśnięciu przycisku powrotu przez gracza, wracające do wyboru trybu gry oraz resetujące wpisane nazwy gracza.

l. void on_pushButton_NS_2_clicked();

Metoda pobierająca nazwę lub nazwy graczy w zależności od wybranego trybu gry, gdy nazwy nie zostały wpisane następuje nadanie domyślnych wartości.

m. void on_pushButton_EG_1_clicked();

Metoda przywracająca początkowe wartości dla ustawień statków oraz pól obsługujących grę, wraca ona na sam początek i pozwala graczowi na rozgranie kolejnej rozgrywki.

n. void on_pushButton_EG_3_clicked();

Metoda przycisku zamykającego całą aplikacje

o. void on_pushButton_3_clicked();

Metoda przycisku *GRAJ!* Tworzy ona odpowiednie obiekty w zależności od wybranego trybu gry, sprawdza czy wpisane są wartości dla masztów poszczególnych statków. W tej metodzie wywoływane jest również generowanie statków dla komputera. A także odpowiednie statki są wpisywane w etykiety na widżetach.

p. void on_pushButton_4_clicked(); void on_pushButton_5_clicked(); void on_pushButton_6_clicked(); void on_pushButton_7_clicked(); void on_pushButton_8_clicked(); void on_pushButton_9_clicked(); void on_pushButton_10_clicked(); void on_pushButton_11_clicked(); void on_pushButton_12_clicked(); void on_pushButton_13_clicked();

Metody przycisków *dodaj statek* sprawdzają w pierwszej kolejności czy został poprawnie wpisany statek czy pole edycyjne nie jest puste oraz czy koordynaty są oddzielone przecinkiem. Jeżeli te dwa warunki zostały spełnione, pobierane oraz dzielone są wartości z pola edycyjnego, przekształcane są one w liczby odpowiednie do użycia jako indeksów tablicy sprawdzane są również odpowiednie flagi, takie jak możliwość wprowadzenia w danym miejscu statku oraz sprawdzenie czy dany statek może być umieszczony w tym miejscu metody: test_correct_positioning oraz check_the_ships. Jeżeli wszystkie wartości logiczne są prawdą to dodawany jest statek oraz wyświetla się on w odpowiednim miejscu na tablicy etykiet.

q. void on_pushButton_14_clicked();

Metoda przycisku *dodaj 2 gracza* sprawdza ona poprawność wpisanych wartości oraz dodaje obiekt pierwszego gracza i możliwość dodania drugiego.

r. void on_pushButton_gen_clicked();

Metoda blokująca możliwość dodania statków ręcznie oraz wywołująca generowanie i ułożenie statków przez algorytm.

s. void on_pushButton_res_clicked();

Metoda dla przycisku wywołująca funkcje *resetGame.*

2. Resources

Klasa służąca do inicjalizacji tablicy 9x9 służącej do rozgrywki w statki oraz operacji na niej, klasa głównie składa się z publicznych metod obsługujących potrzebne zdarzenia, jedyną zmienną prywatną jest sama tablica zawierająca układ statków.

a. resources (); ~resources(); resources (const resources& other);

Konstruktor inicjalizujący tablice z wartościami domyślnymi na 0 oraz zmienna counter również na zero. Dekonstruktor usuwający wszystkie dane z tablicy. Konstruktor kopiujący używany do utworzenia obiektów dla wszystkich graczy.

b. int cordsX_to_numbers(const char* x); int cordsY_to_numbers(int y);

Metody zwracające odpowiednie wartości liczbowe indeksów danych wpisywanym przez użytkownika *cordsX_to_numbers* zmienia litery na odpowiednie liczby dla tablic, *cordsY_to_numbers* pomniejsza o 1 wpisany przez użytkownika indeks Y koordynatu statku.

c. const char* cordsX_for_Bot(int x);

Metoda zwracają na podstawie wpisanej liczby odpowiednia literę.

d. void set_to_defaults();

Metoda przywracająca wartości zer dla całej tablicy.

e. void set_gamespace(const char* x, int y, int value, int pointer);

Metoda wpisująca statek dla obiektu w odpowiednie podane miejsce poprzez użycie metod *cordsX_to_numbers, cordsY_to_numbers* poprawiających ułożenie danych.

f. void set_gamespace_after_shoot(int x, int y, int value);

Metoda modyfikująca tablicę w zależności od wartości parametru *value* w odpowiednich indeksach *x i y*.

g. bool test_correct_positioning(int x, int y, int x1, int y1, int height);

Metoda sprawdzająca ułożenie statku w pionie i poziomie w zależności od podanej wysokości. Gdy podana jest wysokość 2 statek sprawdzany jest o jedna jednostkę do tyłu, jeżeli wysokość wynosi 3 o 2 jednostki, jeżeli 4 to o 3 jednostki. W zależności czy statek spełnia zależności dla podanej wysokości zwracane są odpowiednie wartości logiczne.

h. bool check_the_ships(int x, int y);

Metoda iterująca w pętli w kwadracie 3x3 od podanych wartości parametrów *x, y.* Funkcje max(…) i min(…) zostały użyte w celu zagwarantowania, że pętla nie wyjdzie poza zakres tablicy *gamespace.* Max jest ograniczeniem pętli od lewej strony, żeby nie wyszła poniżej 0, min z prawej by wartości nie wyszły poza zakres 9.

i. int check_cord_for_Bot(int x, int y);

Metoda sprawdzająca przy dodawaniu statku czy w podanych koordynatach znajduje się statek, zwraca ona 0, gdy koordynaty wskazują na puste miejsce, 1 gdy na statek.

j. int check_shot(int x, int y) const;

Metoda sprawdzająca czy oddany strzał o podanych koordynatach został trafiony w statek czy w puste pole, oraz czy podane parametry się nie powtarzają. Metoda zwraca odpowiednie wartości w zależności od wyniku porównania.

3. ScoreBoard : public QObject

Klasa dziedzicząca po publicznej klasie QObject. Klasa służąca do obliczenia punktów oraz zliczania czasu rozgrywki, oraz wpisywaniu wyników rozgrywki do pliku.

a. ScoreBoard(); ~ ScoreBoard();

Konstruktor klasy inicjalizujący wszystkie zmienne oraz obiekt timer, który następnie podłączany jest do metody *Update_time* w celu obliczania minut i sekund rozgrywki. Dekonstruktor przywracający wszystkie zmienne do stanu początkowego.

b. Int calculate_points(int moves);

Metoda obliczająca ilość punktów na podstawie parametru moves, następnie pobierana jest aktualna data funkcja time(nullptr) zwraca aktualny czas systemowy, przypisanie go do struktury *tm* pozwala na łatwy dostęp do daty w celu dalszych operacji. W metodzie również występują operacje wpisania danych do pliku *Data.txt* takich jak zazwa gracza obliczone punkty, liczba ruchów, aktualną datę oraz czas rozgrywki.

c. void Start_time(); void Stop_time();

Metoda inicjalizująca timer, wartość ustalona na 1000 oznacza, że będzie on włączany co sekundę.

d. void Update_time();

Metoda aktualizująca czas co 1 sekundę, jeżeli wartość sekund osiągnie 60 to są one zerowane i wartość minut wzrasta.

4. ScoreWidget

Klasa dziedzicząca po klase MainWindow, służy ona do wywołania nowego okna w celu pokazania tablicy wyników z poprzednich gier, zawierająca jedną funkcje pobierającą dane z pliku do zmiennej.

a. ScoreWidget(QWidget *parent = nullptr); ~ScoreWidget();

Konstruktor inicjuje zmienna string oraz zmienna obsługującą GUI. Dekonstruktor czyści pole tekstowe oraz wartości zmiennej *text.*

b. void read_data();

Metoda czytająca dane z pliku *Data.txt* w trybie do odczytu. Jeżeli zostanie on poprawnie otwarty pole tekstowe jest czyszczone oraz pętlą czytane są dane po jednej linii oraz dołączane do pola tekstowego.