

WIRTUALNY SYMULATOR CZOŁGU – DOKUMENTACJA PROJEKTOWA

Autorzy:

Bartosz Śledzikowski, Bartłomiej Ignaciuk

Prowadzący:

Prof. Robert Szmurło

Warszawa, 27 stycznia 2021

1. Cel projektu

Celem projektu było stworzenie oprogramowania w języku C łączącego się z serwerem za pomocą API i umożliwiającego sterowanie czołgiem. Zadanie zostało wykonane z wykorzystaniem bibliotek takich jak cJSON[1] (umożliwia łatwe parsowanie danych w formacie json w języku C) i curl[2] (odpowiada za komunikację z serwerem). Do generowania obrazu w formacie PNG została wykorzystana biblioteka libpng[4]. Ostatnim już z wykorzystanych zewnętrznych elementów był program indent[5]. Umożliwia on szybkie formatowanie napisanego kodu co znacznie poprawia jego czytelność.

Przykładowa komenda, którą możemy zrealizować taką operację:

```
indent -kr main.c
```

2. Kompilacja

Do kompilacji programu został napisany makefile. Wygląda on następująco:

```
main:
    gcc main.c apiConnector.c mapGenerator.c jsonConverter.c algorithm.c pngGenerator.c -Wall
    -lcurl -lcjson -lpng16 -o tanks

test:
    gcc tests/mainTest.c tests/apiConnectorTest.c tests/mapGeneratorTest.c tests/
    jsonConverterTest.c tests/algorithmTest.c -
    Wall -lcurl -lcjson -lpng -o tanksTest

test-memory:
    valgrind ./tanks --leak-check=full clean:
    rm -f tanks
```

Wykorzystanie `-Wall` pozwala na wykrywanie problemów w kompilowanym kodzie. Informuje również o problemach, które mogłyby zostać pominięte przez ustawiony stopień optymalizacji.

Posiada on cztery opcje:

- `main` - kompiluje główny program.

- test - kompiluje program do testów.
- test-memory - uruchamia program główny w celu zbadania wycieków pamięci za pomocą narzędzia valgrind[3].
- clean - usuwa skompilowany program.

3. Moduły programu

Program składa się z różnych modułów. Możemy wyróżnić dwa główne:

- Główny - odpowiada za działanie docelowego programu.
- Testowy - pozwala na przeprowadzenie testów zaimplementowanych w głównym module metod.

W ramach głównego programu kod został podzielony na pięć różnych modułów:

- main.c - moduł główny, odpowiada za wyświetlanie menu i uruchamianie wybranych przez użytkownika opcji.
- apiConnector.c - realizuje połączenie z serwerem.
- mapGenerator.c - przechowuje strukturę mapy i odpowiada za jej obsługę.
- jsonConverter.c - wyciąga dane przesłane z serwera w formacie json.
- algorithm.c - zawiera implementację algorytmu umożliwiającego automatyczne przemieszczanie się po mapie.

Wersja do testów posiada analogiczne moduły. Odpowiadają one za testowanie swoich odpowiedników z wersji docelowej programu.

4. Mapa

Struktura mapy w programie prezentuje się następująco:

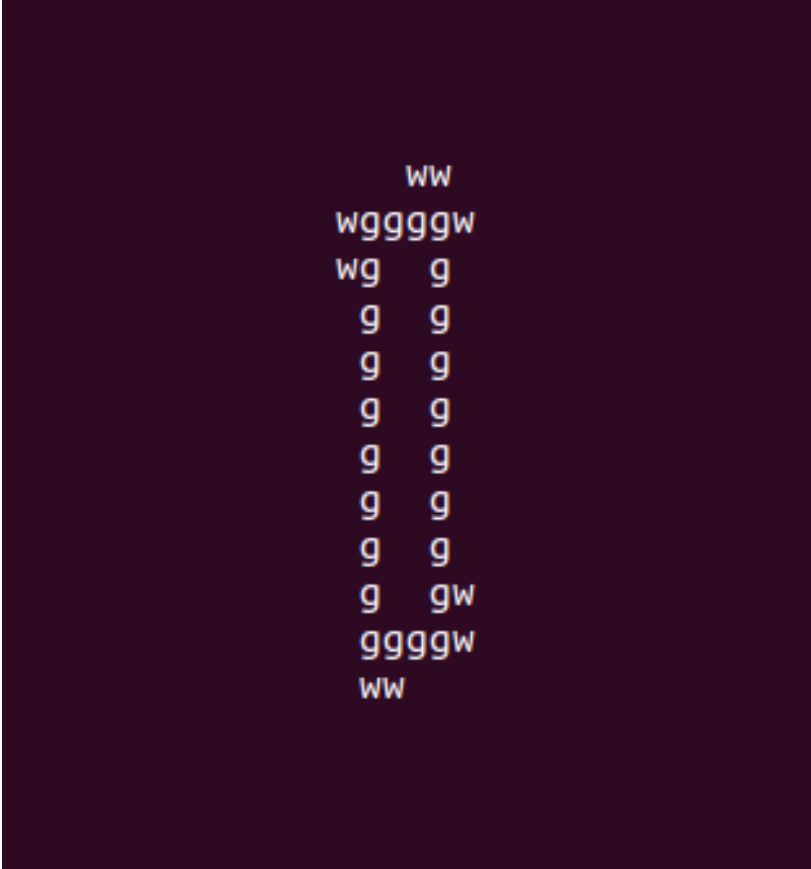
```
struct Map{
    char*name;
    intwidth;
    intlength;
    intcurrentX;
    intcurrentY;
    char currentField; char
    currentDirection;
    char*fields;
};
```

Przechowuje aktualne informacje o badanym świecie oraz macierz z informacjami o odwiedzonych już polach. Są to znaki char o następującym znaczeniu:

- g – trawa (grass)
- w – ściana (wall)

Rysunek 1:

Prezentacja mapy w konsoli



```
      ww
wggggw
wg  g
g  g
g  g
g  g
g  g
g  g
g  gw
ggggw
ww
```

5. Menu

Program w celu wygodnego użytkowania został wyposażony w wyświetlane w konsoli menu.

Posiada ono dwa poziomy:

- Na pierwszym poziomie użytkownik wybiera tryb pracy (reczny lub automatyczny).
- Po wybraniu trybu pracy wyświetlane są odpowiednie dla niego opcje.

```
###WYBOR TRYBU PRACY###
1 - reczny
2 - automatyczny
0 - zamknij
opcja:
```

Rysunek 2: Pierwszy poziom menu

```
###MENU STEROWANIA RECZNEGO###
1 - powrot do poprzedniego menu
2 - info
3 - ruch
4 - obrot w lewo
5 - obrot w prawo
6 - odkryj swiat
7 - resetuj swiat
8 - zapisz mape
9 - wyswietl mape
10 - generuj PNG mapy
0 - zamkniecie aplikacji
opcja:
```

Rysunek 3: Menu ręczne

```
###MENU STEROWANIA AUTOMATYCZNEGO###
1 - powrot do poprzedniego menu
2 - uruchom
3 - wyswietl mape
4 - zapisz mape
5 - generuj PNG mapy
0 - zamkniecie aplikacji
opcja:
```

Rysunek 4: Menu automatyczne

6. Instrukcja użytkownika

Zadaniem programu jest odkrywanie nieznanej mapy. W celu realizacji tego zadania rozwiązanie zostało wyposażone w wiele opcji, które zostaną opisane w tym punkcie.

Program można użytkować w dwóch trybach (ręczny i automatyczny). W trybie ręcznym można samemu poruszać się po mapie używając odpowiednich klawiszy na klawiaturze:

- 2 - wyświetla informacje o aktualnym położeniu
- 3 - wykonuje ruch do przodu
- 4 - wykonuje obrót w lewo o 90 stopni
- 5 - wykonuje obrót w prawo o 90 stopni
- 6 - wyświetla informacje o sąsiadujących polach
- 7 - resetuje świat
- 8 - zapisuje mapy do pliku .txt
- 9 - wyświetla aktualny stan mapy ze struktury danych
- 10 - generuje obraz w formacie .png bazując na aktualnym stanie struktury mapy
- 0 - zamyka aplikację

W trybie automatycznym użytkownik ma mniejszą ilość opcji do wyboru:

- 2 - uruchamia algorytm odkrywania świata
- 3 - tak jak w trybie ręcznym wyświetla aktualny stan struktury mapy
- 4 - zapisuje aktualny stan struktury mapy do pliku .txt
- 5 - generuje obraz w formacie .png
- 0 - zamknięcie aplikacji

7. Komunikacja z serwerem

Komunikacja z serwerem została zrealizowana przez implementację odpowiednich funkcji wykonujących zapytania do API. W zależności od trybu pracy programu jego użytkownik lub algorytm wybiera jakie zapytanie chce zadać. Pobierana jest wówczas odpowiedź z serwera w formacie JSON, która jest już przetwarzana w osobnym module.

Zapytania, jakie można wysłać to:

- Info - zwraca informacje o obecnym położeniu.
- Move - próbuje wykonać ruch do przodu.
- RotateLeft - wykonuje obrót w lewo o 90 stopni.
- RotateRight - wykonuje obrót w prawo o 90 stopni.
- Reset - resetuje stan świata.
- ExploreWorld - zwraca informacje o sąsiadujących polach.

8. Odczyt danych z JSONa

Do implementacji parsera do danych w formacie JSON wykorzystana została biblioteka cJSON. W module apiConverter.c została zaimplementowana metoda jsonParserPayload, która przyjmuje w parametrze dane przesłane przez serwer.

```
void jsonParserPayload(char*json_string) {
    cJSON*name=NULL;
    cJSON*root=cJSON_Parse(json_string);

    root=cJSON_GetObjectItem(root, "payload");

    name=cJSON_GetObjectItem(root, "current_x");
    current_x=name->valueint;

    name=cJSON_GetObjectItem(root, "current_y");
    current_y=name->valueint;

    name=cJSON_GetObjectItem(root, "direction");
    direction=name->valuelstring[0];

    name=cJSON_GetObjectItem(root, "field_type");
    field_type=name->valuelstring[0];
}
```

Poszczególne dane są odczytywane przez odpowiednie metody biblioteki cJSON. Następnie są wykorzystywane w aktualnej strukturze mapy.

9. Algorytm

Algorytm zaimplementowany do automatycznego odkrywania świata działa następująco:

1. wykonuje ruch naprzód aż do napotkania ściany.
2. po napotkaniu ściany wykonuje obrót w prawo.
3. wykonujemy jeden ruch do przodu i obrót w lewo.
4. próbujemy wykonać ruch do przodu, jeśli się nie da wykonujemy obrót w lewo i idziemy naprzód.
5. powtarzamy punkty 2-4 aż do powrotu do punktu startowego.

W trakcie wykonywania algorytmu zliczamy ilość obrotów w lewo oraz w prawo. W naszym konkretnym przypadku większa ilość skrętów w lewo oznacza, że znaleźliśmy ścianę wewnętrzną.

W tym przypadku rozpoczynamy proces odkrywania świata na nowo. Jeśli jest na odwrót to oznacza, że udało nam się znaleźć otoczkę zewnętrzną badanego świata. W takim przypadku możemy rozpocząć przeszukiwanie pozostałych pól świata.

10. Złożoność obliczeniowa

Złożoność obliczeniowa rozpatrywanego problemu w dużej mierze zależy od wymiarów badanego świata. Zakładając najbardziej optymistyczny przypadek, nasz świat ma wymiary $n \times n$ i nie posiada żadnych wewnętrznych przeszkód możemy odkryć go wykonując n^2+1 ruchów. Wówczas złożoność obliczeniowa wyniesie $O(n^2)$. Ten przypadek nie uwzględnia jednak opóźnień, jakie mogą być generowane przez oczekiwanie na odpowiedź serwera. W realnym przypadku dla większego rozmiaru n może być to znacząca wartość. Kolejne opóźnienia w realnym przykładzie mogą być generowane przez wewnętrzne przeszkody napotkane podczas eksploracji świata. Koszt odkrycia każdej przeszkody możemy przyjąć jako ilość ruchów konieczna do wykonania w celu jej okrążenia. W przypadku bardziej skomplikowanego problemu tego typu dobrze mogłyby się sprawdzić algorytmy sztucznej inteligencji.

11. Literatura

1. Biblioteka cJSON - <https://github.com/DaveGamble/cJSON> Data dostępu 18.01.2021
2. Biblioteka curl - <https://curl.se/> Data dostępu 19.01.2021
3. Valgrind - <https://valgrind.org/> Data dostępu 19.01.2021
4. Libpng - <http://www.libpng.org/pub/png/libpng.html> Data dostępu 23.01.2021
5. Indent - <https://www.gnu.org/software/indent/> Data dostępu 23.01.2021