Projekt Czołg

Bartosz Staszyński i Wiktor Kępiński 27.01.2021

Wydział Elektryczny, Elektromobilność

Cel projektu

Celem projektu było stworzenie programu z implementacją sztucznej inteligencji, która poprzez komunikację z serwerem oraz analizę odpowiedzi serwerowych w plikach typu *jSON* zwraca mapę świata zamieszczonego na serwerze jako obraz *PNG*.

Uruchamianie

Program należy uruchomić poprzez konsolę komendą *make test-bot.* W pliku *makefile* znajdują się też komendy do testowania konkretnych funkcji programu oraz kontrolowania wycieków pamięci za pomocą komendy *valgrind.* Lista komend dostępnych w *makefile:*

- *make* - kompiluje wszystkie pliki *.c* i tworzy plik wykonywalny o nazwie *"program"*
- *make test* – uruchamia program główny podając dwa argumenty: token
- *make test-load* - uruchamia test modułu odpowiedzialnego za komunikację z serwerem. Wypisuje do pliku *derulo.json* dwie odpowiedzi.
- *make test-response* - uruchamia test modułu odpowiedzialnego za uzupełnienie odpowiednich struktur danymi z napisu *.json.* Wyświetla na ekranie dwie przykładowe informacje.
- *make test-map* - uruchamia test modułu odpowiedzialnego za zarządzanie strukturą mapy. Wczytuje mapę z pliku *mapa.txt,* wyświetla ją na ekranie oraz zapisuje do nowego pliku *mapa_wypisz.txt.*
- *make test-logic* - uruchamia test modułu odpowiedzialnego za logikę. Resetuje stan mapy i uruchamia algorytm eksploracji
- *make test-image* - uruchamia test modułu odpowiedzialnego za obsługę biblioteki *.png,* Wczytuje mapę z pliku tekstowego *mapa.txt* i zapisuje do pliku *mapa.png*
- *make memory* - uruchamia program z komendą valgrind
- *make memory-load* - uruchamia test modułu *load* z komendą *valgrind*
- *make memory-response* - uruchamia test modułu *response* z komendą *valgrind*
- *make memory-map* - uruchamia test modułu *map* z komendą *valgrind*
- *make memory-logic* - uruchamia test modułu *logic* z komendą *valgrind*
- *make memory-image* - uruchamia test modułu *image* z komendą *valgrind*
- *make clean* - usuwa wszystkie pliki *.o* oraz plik *program* i pliki testowe.

Moduły

Program podzielono na pięć modułów ze względu na ich funkcje i udział w całości programu.

Load.h

Moduł *load.h* zawiera deklaracje struktury *Memory* oraz funkcji odpowiedzialnych za komunikacje z serwerem API przy pomocy biblioteki *cURL*. Funkcje konwertują podaną przez użytkownika komendę na zapytanie do serwera i zapisują odpowiedź do zmiennej *response,* która jest napisem w formacie *cJSON*.

- *write_callback* - funkcja biblioteki *curl*
- *make_request* - wykonuje zapytanie do serwera za pomocą przekazanego w argumencie linku i zwraca wartość (w tym przypadku napis w formacie *.json)* z serwera API. Do komunikacji z serwerem używa biblioteki *curl.*
- *get_request* - tworzy odpowiedni link na podstawie przekazanego w argumencie tokenu oraz żądanej komendy. Zwraca odpowiedź z funkcji *make_request* jako napis formacie *.json.*

Response.h

Moduł odpowiedzialny jest za analizę odpowiedzi serwera przy pomocy biblioteki *cJSON* i zwrócenie dane w postaci strukturalnej. Korzysta do tego z struktur *List*, *Field* i *Response.*

- *get_struct* - zapisuje informacje z tekstu *.json* pobrane z serwera do struktury *Response* za pomocą biblioteki *cJSON*
- *get_explore* - zapisuje informacje z tekstu *.json* pobrane z serwera do struktury *List* za pomocą biblioteki *cJSON*. Ta funkcja działa analogicznie do funkcji *get_struct,* jednak do zapytania "explore" potrzebna jest inna struktura ze względu na inną formę odpowiedzi z serwera.
- *free_explore* – zwalnia pamięć zaalokowaną pod adresem struktury *List*
- *free_response* – zwalnia pamięć zaalokowaną pod adresem struktury *Response*

Map.h

Jest to część programu zajmująca się tworzeniem, wczytywaniem oraz zapisywaniem struktury *Map* do pliku tekstowego.

- *create_map* – tworzy strukturę *map* o podanych rozmiarach, alokuje pamięć
- *load_map* – wczytuje z pliku strukturę *map*
- *print_map* – wypisuje mapę w konsoli
- *fprint_map* – zapisuje do pliku tekstowego mapę
- *free_map* – zwalnia przypisaną pamięć do struktury *map*

Logic.h

Najbardziej rozbudowany moduł odpowiedzialny za elementy sztucznej inteligencji eksplorującej i zapisującej mapę, konwertowanie współrzędnych serwerowych i lokalnych oraz dynamiczne powiększanie mapy zależne od sytuacji oraz interpretacja danych w strukturach *List, Field, Response* i wypełnienie nimi struktury *Map*.

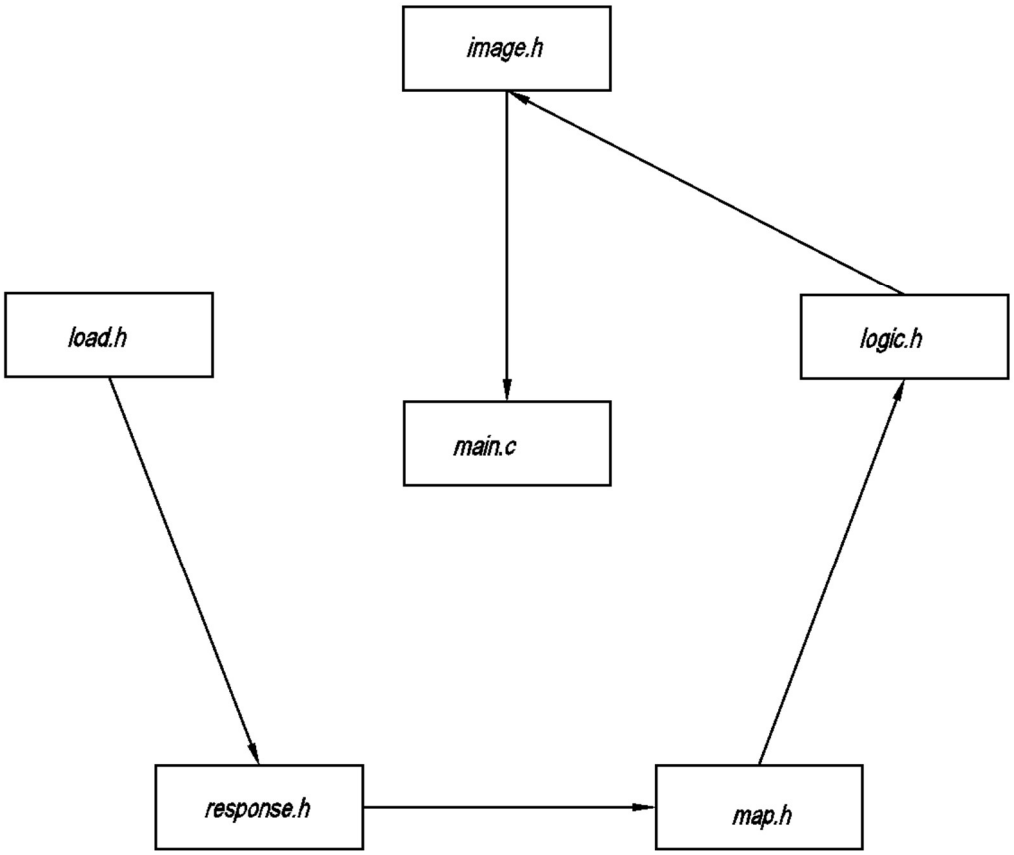
- *type* – analizuje pole, na którym jest czołg, zwraca wartość *int* odpowiednio *grass – 1, sand – 2, wall – 3.*
- *get_dx* – określa przesunięcie współrzędnych lokalnych środka mapy względem koordynat czołgu w osi x. Dzięki temu nasz czołg zawsze zaczyna na środku.
- *get_dy* – określa przesunięcie współrzędnych lokalnych środka mapy względem koordynat czołgu w osi y. Dzięki temu nasz czołg zawsze zaczyna na środku.
- *offsetx* – zamienia współrzędną globalną x na lokalną przy użyciu elementu *dx* w strukturze *Map.*
- *offsety* – zamienia współrzędną globalną y na lokalną przy użyciu elementu *dy* w strukturze *Map.*
- *loc_to_globx* – zamienia lokalne współrzędne x na globalne.
- *loc_to_globy* – zamienia lokalne współrzędne y na globalne.
- *set_map* – tworzy mapę za pomocą funkcji *create_map* z modułu *map.h*, ustawia początkowe parametry mapy oraz wczytuje odpowiedź *"info"* z serwera, żeby zlokalizować nasz czołg.
- *reset_map* – wykonuje zapytanie do serwera *reset* i zeruje strukturę *map.*

- *seek_wall* – wykonuje operacje move do momentu zetknięcia się ze ścianą. Po napotkaniu ściany wywołuje *turn_right*.
- *turn_right* – wykonuje obrót w prawo, a następnie wczytuje odpowiedź "explore" z serwera
- *continue_forward* – wykonuje ruch do przodu, a następnie wczytuje odpowiedź "explore" z serwera
- *take_left_corner* – wykonuje sekwencje ruchu do przodu, obrotu w lewo i wczytuje odpowiedź "explore" z serwera.
- *get_out_of_blind_corner* – wykonuje próbę ruchu, jeśli nie uda się obraca się w prawo i powtarza to do momentu zmiany koordynat. Wynika to z tego, że po znalezieniu ściany czołg rusza się, aż wróci do początkowych współrzędnych x0 i y0. Dlatego najpierw musimy opuścić pole o współrzędnych x0 i y0, żeby rozpocząć *explore_border*.
- *seek_left_corner* – podąża wzdłuż ściany. Jeżeli napotka przed sobą ścianę to wywołuje *turn_right,* jeżeli napotka możliwość skrętu w lewo to wywołuje *take_left_corner,* a jeżeli może iść dalej wzdłuż ściany to wywołuje *continue_forward.*
- *explore_border* – wywołuje funkcję seek_left_corner dopóki nie okrąży przeszkody.
- *bot* – wykonuje w pętli sekwencje funkcji: *seek_wall, get_out_of_blind_corner, explore_border.* Jeżeli eksplorowana w poprzednim przejściu pętli otoczka jest **otoczką wewnętrzną**, pętla się powtarza. Odkrycie **otoczki zewnętrznej** przerywa pętlę i kończy działanie funkcji *bot*.
- *interpret_explore* - aktualizuje naszą strukturę *Map* danymi ze struktury *Lista* przekazanej z modułu *response.c*
- *interpret_response* - aktualizuje naszą strukturę *Map* danymi ze struktury *Response* przekazanej z modułu *response.c*
- *edge* - wypisuje w postaci napisu, przy którym brzegu mapy czołg się znajduje. Jeżeli nie jest przy żadnym brzegu to zwraca napis *"null"*
- *add_chunk* – W zależności od tego, przy którym brzegu znajduje się czołg, tworzy nową mapę dwa razy większą, kopiuje dane z poprzedniej mapy i zwalnia pierwszą mapę. Jeżeli czołg nie znajduję się przy żadnym brzegu, to funkcja *add_chunk* zwraca pierwotną mapę bez zmian.

Image.h

Moduł zajmujący się zapisem wyniku programu do postaci pliku PNG przy pomocy biblioteki *PNG.c*

- *read_png_file* - Funkcja biblioteki *png.* Wczytuje plik *.png,* z którego będzie korzystać jako „palety".
- *write_png_file* - wypisuje do pliku *.png* utworzony obraz.
- *process_png_file* - zmienia atrybuty każdego piksela przekazanego pliku *.png* takie jak jasność, balans kolorów, przezroczystość za pomocą kodowania w formacie RGB + przezroczystość.
- *init_png_file* - tworzy strukturę *png_bytep row_pointers* i wypełnia ją białymi pikselami.
- *copy_tile* – kopiuję określoną część z pliku bazowego do struktury row_pointers.
- *tile_number* – przypisuje odpowiadający identyfikator „kafelka" z pliku źródłowego na podstawie wartości *field_type* danego pola.
- *png_map* – zwraca mapę w postaci pliku *.png*
- *free_row_pointers* – zwalnia przypisaną pamięć potrzebną do stworzenia pliku *.png*



Obraz 1: Relacja plików nagłówkowych

Struktury danych

Struktura *Memory* używana jest do komunikacji z serwerem i składa się ze zmiennej, która przechowuje odpowiedź serwera oraz zmiennej określającej rozmiar odpowiedzi.

Ze względu na dwie formy plików *JSON* zwracanych przez serwer wyróżniono trzy struktury, do których postaci konwertowane są odpowiedzi są to: *List, Field* i *Response.* Struktura *List* służy do przechowania wyniku zapytania *explore* i składa się z napisu określającego status operacji oraz trzech zmiennych typu *Field* które składają się z koordynat oraz typu pola. Struktura *Response* używana jest do wszystkich pozostałych typów zapytań do serwera i mieści w sobie zmienne do zapisu: statusu odpowiedzi, nazwy świata, koordynat, numeru sesji, kierunku, w który zwrócony jest czołg, ilości wykonanych kroków, typu pola, na którym znajduje się czołg, oraz wartości bonusu pola.

W celu stworzenia mapy świata, po którym porusza się czołg stworzono strukturę *Map* zawierającą wymiary mapy, informacje na temat koordynat, kierunku poruszania się czołgu, ilości wykonanych kroków, typu pola, na którym jest lub znajdował się czołg, oraz dwie zmienne odpowiedzialne za konwersję koordynat serwerowych oraz lokalnych, które aktualizowane są przy powiększaniu się mapy.

Komunikacja

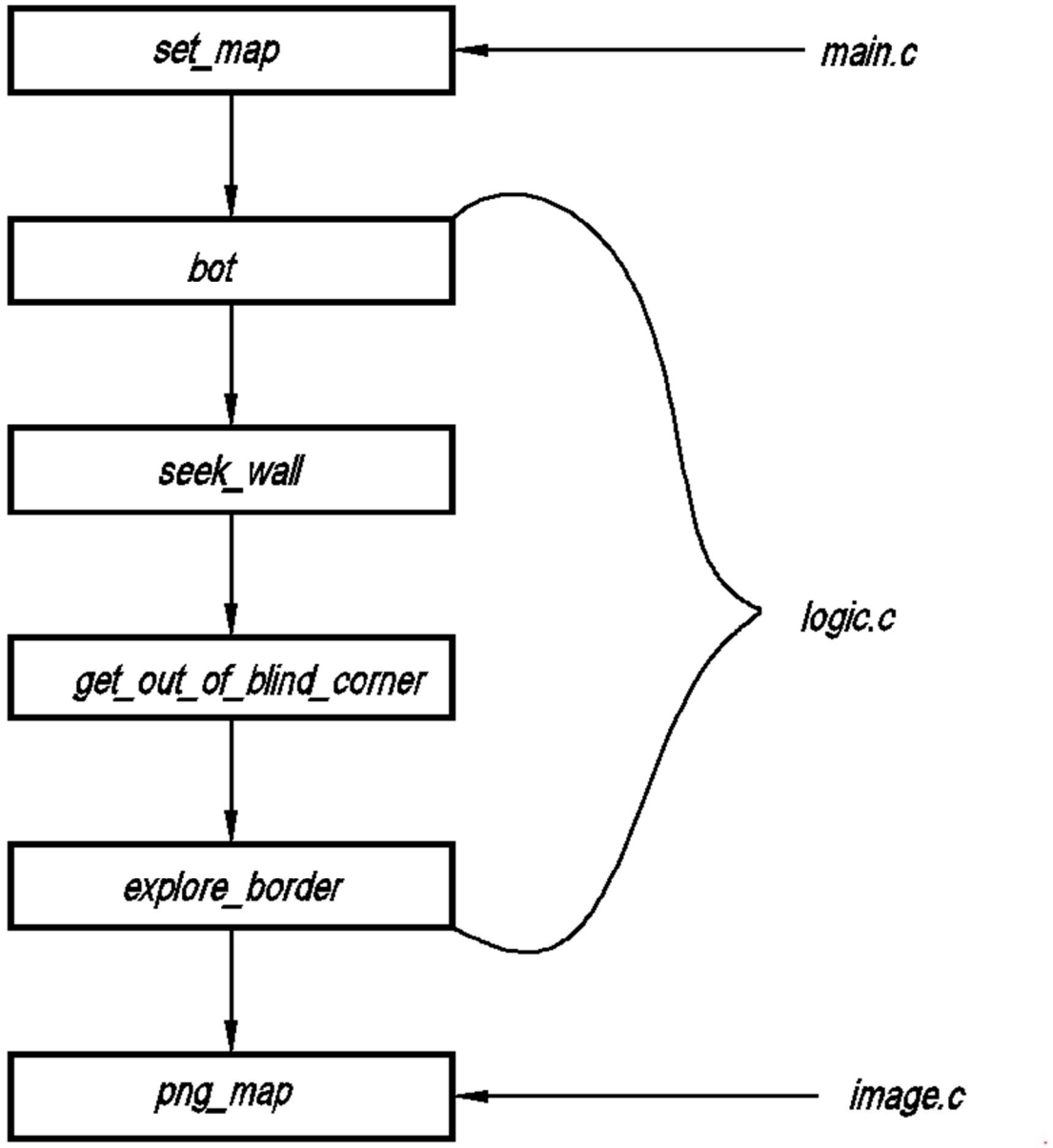
Program w celu wykonania ruchu komunikuje się przy pomocy biblioteki *cURL* z serwerem o adresie http://edi.iem.pw.edu.pl:30000/. W celu komunikacji korzystaliśmy z funkcji *make_request* w module *load.c*. Zapytania do serwera przekazywaliśmy w formie linku o odpowiednich parametrach. W odpowiedzi dostawaliśmy treść w formacie *.json*

Scenariusz działania

Po uruchomieniu programu komenda *make test-bot* program wykonuje funkcję *set_map* o wymiarach 5 na 5 jednocześnie wykonuje operację *info* aby uzyskać koordynaty i określić wartości *dx* i *dy.* Następnie wywołuje funkcje *bot*:

- wywołanie funkcji *seek_wall,* która wykonuje operacje *move* do momentu kiedy koordynaty po wykonanym ruchu są takie same jak przed nim co oznacza, że czołg znajduje się przy ścianie następnie wykonywane są operacje *rotate_right* oraz *explore*.
- Wywołanie funkcji *get_out_of_blind_corner* – obraca się i próbuje ruszyć, aż wyjdzie z pola.
- wywołanie funkcji *explore_border,* która wywołuje funkcję *seek_left_corner* i powtarza to działanie w pętli do czasu, aż znajdzie się w miejscu, w którym zaczynał.

Na końcu po przejściu mapy zapisuje ją w pliku *.png*



Obraz 2: Schemat działania programu