Politechnika Wrocławska

Zadanie rekrutacyjne Tietoevry

Turn Base Game

Prace wykonał

Marcin Cichocki

Wydział i kierunek studiów

W12N, Automatyka i Robotyka

autor szablonu:

Jakub Marczak

Spis treści

1 Ogólne zasady

Rozważając kontekst opisanego zadania związane z grą turową, zauważamy trzy główne podmioty: "Mediator" oraz gracze pierwszy i drugi. Zgodnie z przeczytanym poleceniem, zadaniem tych programów jest planowanie działań w trakcie rozgrywki. Mają one wspólną strukturę funkcjonalności, która obejmuje planowanie ruchu jednostek, strategię ataku w kierunku przeciwnika oraz możliwość ataku na bazę przeciwnika.

Przedstawiamy koncepcję zjednoczonego programu gracza, który umożliwia zredukowanie duplikacji kodu. Ten program posiada zdolność przystosowania się do obecnie aktywnego gracza poprzez dostarczenie argumentu "1" lub "2". To umożliwia jednemu programowi wykonywanie tych samych działań w zależności od aktualnego uczestnika rozgrywki. Ważną uwagą jest to, że oba gracze realizują te same zadania, a jedyna różnica polega na określeniu, czy aktualny ruch jest dla gracza "P" (pierwszy) lub "E" (drugi).

Warto zaznaczyć, że program gracza skupia się na planowaniu akcji, które są następnie zapisywane do pliku "rozkazy.txt". Ten proces nie ma wpływu na bieżący stan gry, a jedynie dostarcza planowanych ruchów. Natomiast rola "Mediatora" jest wykonywanie zaplanowanych akcji oraz aktualizowanie stanu gry poprzez modyfikację pliku "status.txt". Ostatecznie, ten plik zawiera kluczowe informacje, które pozwalają drugiemu graczowi na zrozumienie bieżącego stanu rozgrywki.

2 Program Turn Base Game

2.1 Program player

W ramach realizacji programu o nazwie "player", wykorzystana została prosta, aczkolwiek skuteczna strategia operacyjna. Na wstępie, program inicjuje proces działania poprzez odczyt danych z dwóch plików źródłowych: "mapa.txt" oraz "status.txt". Na podstawie tych informacji, program podejmuje decyzje dotyczące ruchów przypisanych jednostek, pod warunkiem ich istnienia na planszy.

Niezmiernie istotne jest, że złożoność implementacji rozbudowanej sztucznej inteligencji przewyższała dostępne zasoby czasowe i umysłowe. W związku z tym, postanowiono zastosować podejście oparte na regułach, które jest bardziej przystępne. W toku rozgrywki, program przykłada szczególną uwagę do budowy jednostek w oparciu o dostępną ilość zasobu w postaci złota. W przypadku obecności kopalni na planszy, priorytetowym celem staje się wytworzenie jednostki klasy "worker" w celu efektywnego gospodarowania zasobem.

Ruchy jednostek programu są starannie zaplanowane, dostosowując się do celu strategicznego. Wszystkie jednostki o charakterze ofensywnym zmierzają w kierunku bazy przeciwnika, mając za zadanie ją zaatakować. W tym samym czasie, jednostki klasy "worker" kierują się w stronę najbliższej dostępnej kopalni na mapie. Wyjątkową wartość stanowi fakt, że program realizuje to działanie we współpracy z warunkiem, który mówi, że kopalnia musi istnieć na planszy.

Główna pętla programu zarysowuje się w trzech kluczowych etapach:

1. Wczytanie mapy z pliku "mapa.txt" oraz identyfikacja położenia ewentualnych kopalni. Pozycje tychże kopalni zostają zarejestrowane, tworząc bazę informacyjną.

2. Ekstrakcja danych dotyczących własnych jednostek oraz przeciwnika z pliku "status.txt". To daje programowi podstawę do dalszych decyzji.

3. Realizacja procesu budowy nowej jednostki o losowym charakterze, pod warunkiem dostępności kopalni na planszy. W sytuacji braku pracującego "workera", program stawia na wytworzenie właśnie takiej jednostki.

Następnie, program przystępuje do zaplanowania ruchów jednostek w następujący sposób:

1. Jednostki o charakterze ofensywnym kierują się w kierunku bazy przeciwnika, koncentrując swoje ataki na tym celu.

2. Jednostki klasy "worker" dążą do najbliższej kopalni, skupiając się na efektywnym pozyskiwaniu zasobów.

Warto podkreślić, że po dotarciu na ustalone pozycje, jednostki przeprowadzają skrupulatne przeszukanie okolicznych obszarów w celu znalezienia potencjalnych celów ataku. W hierarchii priorytetów, najważniejszym celem jest baza przeciwnika. Jeśli jednak baza nie jest obecna, jednostka atakuje pierwszy napotkany cel znajdujący się w jej zasięgu ataku. Wszystkie zaplanowanie akcje zapisywane sa w pliku "rozkazy.txt"

Podsumowując, powyższy opis ukazuje techniczne i strategiczne założenia programu "player". Ten prosty, ale pragmatyczny model działania został opracowany w odpowiedzi na wyzwania związane z ograniczeniem dostępnych zasobów oraz skomplikowanym charakterem implementacji zaawansowanej sztucznej inteligencji. Ostatecznie, program dąży do optymalizacji zarządzania jednostkami oraz skutecznego wykonywania działań w kontekście rozgrywki.

2.1.1 Główna pętla programu Player

```
void Game::mainAlgorithm()
{
    // Call all functions sequentially, main loop of the program
    loadMap();
    loadEntities();
    buildRandomEntity();
    moveWithAllEntity();
    attackWithAllEntity();
}
```

Rysunek 1: Główne funkcje wywołane podczas pracy programu Player

2.2 Program Mediator

W roli niezwykle istotnego aktora w kontekście kierowania procesem rozgrywki, Mediator manifestuje swoją kluczową rolę jako główny orchestrator, odpowiedzialny za przeprowadzenie rozgrywki od inicjacji do finalizacji. Mediator, stanowiący ośrodek sterujący, zajmuje się kierowaniem biegiem zdarzeń na przestrzeni całego procesu gry.

Na samym początku sekwencji, Mediator weryfikuje stan plików "rozkazy.txt" oraz "status.txt" i podejmuje działanie polegające na ich wyczyszczeniu, tworząc w ten sposób pustą płaszczyznę dla nowych informacji. Kolejnym istotnym etapem dla Mediatora jest wytworzenie bazowego stanu gry, zaczynając od pliku "status.txt". Przy tej okazji, Mediator przekazuje fundamentalne parametry, takie jak początkowa ilość złota dla pierwszego gracza, a także kluczowe informacje o bazie gracza oznaczonego jako "P", jak również analogiczne dane odnośnie bazy przeciwnika "E".

Ogromne znaczenie przypada również inicjacji mapy oraz identyfikacji strategicznie istotnych punktów, takich jak lokalizacja kopalni. Proces ten stanowi elementarny składnik umożliwiający Mediatorowi dostarczanie istotnych informacji uczestnikom rozgrywki.

W centralnym elemencie sekwencji, Mediator realizuje cykl, w którym naprzemiennie wywołuje program gracza o numerze 1 i 2, w sposób rytmiczny. Po zakończeniu działania programu gracza, Mediator przechodzi do dekodowania zawartych w pliku "rozkazy.txt" instrukcji, które to stanowią strategiczne decyzje gracza. Te instrukcje zostają przetworzone w kontekście stanu gry, co owocuje aktualizacją bieżącej sytuacji. Po skompletowaniu cyklu związanej z jednym z uczestników, Mediator z kolei przystępuje do wywołania programu reprezentującego drugiego gracza, realizując analogiczny proces dekodowania i aktualizacji.

W taki sposób, Mediator manifestuje swoją centralną rolę jako interakcyjny motor napędowy procesu rozgrywki. Przez zapewnienie stałego cyklu wymiany informacji oraz dynamicznej adaptacji do decyzji uczestników, Mediator kształtuje przebieg gry, wprowadzając strategiczne elementy, które składają się na kompletny obraz interakcji między uczestnikami rozgrywki.

2.2.1 Główna pętla programu

```cpp
int main()
{
    int turnCounter = 0;

    cleanOrdersOrStatus(rozkazy);
    cleanOrdersOrStatus(status);
    generateFirstStatus();

    std::string commandPlayer1 = program + " " + mapa + " " + status + " " + rozkazy + " " + player1;
    std::string commandPlayer2 = program + " " + mapa + " " + status + " " + rozkazy + " " + player2;


    while (turnCounter < MAX_TURNS)
    {
        std::cout << "ruch plaeyr 1\n";
        system(commandPlayer1.c_str()); //odpalenie tury dla gracza pierwszego
        allActions(1);

        std::cout << "ruch plaeyr 2\n";
        system(commandPlayer2.c_str()); //odpalenie tury dla gracza drugiego
        allActions(2);

        turnCounter++;
    }


    checkWinAfterTurnsEnd();

}
```

Rysunek 2: Główne funkcje wywołane podczas pracy programu Mediator

2.3 Wzajemne działanie dwóch programów

W kontekście związku między programami "Mediator" i "Player" ukazuje się, że istnieje ściśle uzależniony charakter tych dwóch komponentów. Program "Player" pełni kluczową rolę jako generujący rozkazy uczestnik rozgrywki. Bezpośrednio z nim związane jest wykonywanie operacji na plikach, takich jak "mapa.txt", "status.txt" oraz "rozkazy.txt". W istocie, "Player" może działać w izolacji poprzez podanie wymaganych plików w linii poleceń. Niemniej jednak jego funkcjonalność zostaje ograniczona do analizy i reakcji na aktualny stan pliku "status.txt". Zauważalne jest, że sam "Player" nie posiada zdolności do manipulacji tym plikiem, ograniczając się jedynie do dostarczania "Mediatorowi" kolejnych rozkazów.

"Mediator", z drugiej strony, manifestuje swoją główną rolę w przetwarzaniu i realizacji przekazywanych rozkazów. Stanowi on łącznik pomiędzy dynamicznymi działaniami "Playera" a rozwojem rzeczywistej rozgrywki. "Mediator" jest w istocie oparty na funkcjonowaniu "Playera", bowiem to on dostarcza niezbędne decyzje do analizy i wdrażania. Zadaniem "Mediatora" jest zatem wykonywanie instrukcji zawartych w pliku "rozkazy.txt", co przekształca się w efektywne aktualizacje rozgrywki, uwzględniające zmiany przyniesione przez uczestników.

Ta wzajemna zależność ukazuje konieczność obecności "Playera" jako podstawowego źródła informacji i decyzji, które "Mediator" następnie przekształca w dynamiczną ewolucję stanu rozgrywki. W istocie, koncept "Mediatora" istotnie opiera się na kooperacji z "Playerem", co implikuje, że działanie "Mediatora" bez obecności "Playera" nie posiada merytorycznego sensu, ponieważ nie posiadałby on źródła danych i instrukcji niezbędnych do generowania zmian w rozgrywce.

3 Program Players Turn Base Game

3.1 Opis działania programów

Działanie programów "Player1" oraz "Player2" w żadnym aspekcie nie odbiega od funkcjonalności programu "Player," jak to opisano w rozdziale 2.1. Wszystkie te programy zostały zaprojektowane z myślą o kompilacji w celu obsługi zadania przy użyciu odmiennego mediatora, który nie jest identyczny z tym, który wskazałem. Jedyną rozbieżnością między tymi trzema implementacjami ("Player," "Player1," oraz "Player2") jest wartość argumentu (przekazywanego poprzez argv[4]), dostarczanego do programu. W tej wartości zawarte jest oznaczenie gracza, który w danej chwili posiada prawo wykonywania ruchu ("1" lub "2").

```cpp
int main(int argc, char* argv[])
{
    //Game game; //stworzenie obiektu game, nim bedziemy sterowac

    if (argc >= 2)
    {
        Game game(argv[1], argv[2], argv[3], "1");

        try
        {
            game.mainAlgorithm();
        }
        catch (const std::exception& e)
        {
            std::cout << "Wystapil wyjatek:" << e.what() << '\n';
            return 1;
        }
    }

    else
    {
        std::cout << "Brak przekazanego argumentu." << std::endl;
    }
}
```

Rysunek 3: Główne pętla programu Player1

```cpp
int main(int argc, char* argv[])
{
    //Game game; //stworzenie obiektu game, nim bedziemy sterowac

    if (argc >= 2)
    {
        Game game(argv[1], argv[2], argv[3], "2");

        try
        {
            game.mainAlgorithm();
        }
        catch (const std::exception& e)
        {
            std::cout << "Wystapil wyjatek:" << e.what() << '\n';
            return 1;
        }
    }

    else
    {
        std::cout << "Brak przekazanego argumentu." << std::endl;
    }
}
```

Rysunek 4: Główne pętla programu Player2

4 Sposób zbudowania programu Turn Base Game

Aby zbudować program do gry turowej, zgodnie z instrukcjami dostępnymi w repozytorium pod adresem Program Turn Based Game, wymagane są pewne kroki proceduralne. Procedury te obejmują pobranie plików z repozytorium, wykonanie określonych poleceń kompilacyjnych oraz modyfikację konkretnych plików źródłowych. Oto kroki, które należy podjąć:

1. Pobranie i rozmieszczenie plików:
   - Pobierz pliki z repozytorium dostępnego pod linkiem Program Turn Based Game.
   - Po pobraniu plików, rozpakuj je w miejscu wyznaczonym do tego celu.
2. Kompilacja programu "Player":
   Wejdź do folderu "Player" i w terminalu wykonaj poniższe kroki:
   - Stwórz nowy katalog kompilacyjny poprzez wpisanie:
     ```
     mkdir build && cd build
     ```
   - Wywołaj narzędzie CMake poprzez wpisanie:
     ```
     cmake ..
     ```
   - Skompiluj program przy użyciu polecenia:
     ```
     make
     ```
3. Wstawienie ścieżki pliku wykonawczego "Player":
   Skopiuj dokładną ścieżkę do pliku wykonywalnego "Player," który został skompilowany w poprzednim kroku. Następnie w pliku "main.cpp" znajdującym się w folderze "Mediator," w linijce 13, wklej tę ścieżkę.
4. Wstawienie ścieżek do plików "mapa.txt," "status.txt" oraz "rozkazy.txt":
   Przejdź do folderu "Files" i skopiuj ścieżki dostępu do plików "mapa.txt," "status.txt" oraz "rozkazy.txt." Następnie w pliku "main.cpp" w folderze "Mediator," w linijkach 14-16, wklej te ścieżki w odpowiednie miejsca.
5. Kompilacja programu "Mediator":
   Przejdź do folderu "Mediator" w terminalu i wykonaj poniższe czynności:
   - Stwórz nowy katalog kompilacyjny poprzez wpisanie:
     ```
     mkdir build && cd build
     ```
   - Wywołaj narzędzie CMake poprzez wpisanie:
     ```
     cmake ..
     ```
   - Skompiluj program przy użyciu polecenia:
     ```
     make
     ```
6. Uruchomienie programu "Mediator":
   Po skompilowaniu programu "Mediator," uruchom terminal w folderze "Mediator" i wpisz:
   ```
   ./Mediator
   ```

5 Sposób zbudowania programu Players Turn Base Game

Na samym poczatku nalezy sciagnac program z repozytorium na githubie Players Turn Base Game i wypakowac zawartosc w dostosowanym do tego miejscu. Ta wersja programu przypominam jest jedyna opcją do testowania programu przy uzyciu innego Mediatora niz w programie TurnBaseGame.

1. Pobranie i rozmieszczenie plików:
   - Pobierz pliki z repozytorium dostępnego pod linkiem Players Turn Base Game.
   - Po pobraniu plików, rozpakuj je w miejscu wyznaczonym do tego celu.
2. Kompilacja programu "Player1":
   Wejdź do folderu "Player1" i w terminalu wykonaj poniższe kroki:
   - Stwórz nowy katalog kompilacyjny poprzez wpisanie:
     ```
     mkdir build && cd build
     ```
   - Wywołaj narzędzie CMake poprzez wpisanie:
     ```
     cmake ..
     ```
   - Skompiluj program przy użyciu polecenia:
     ```
     make
     ```
   - Uruchomienie programu "Player1":
     Po skompilowaniu programu "Player1," uruchom terminal w folderze "Player1" i wpisz:
     ```
     ./Player1 mapa.txt status.txt rozkazy.txt "czas"
     ```

Dla programu Player2 nastepują takie same procedury kompilacji. Nalezy pamietać zeby podać w lini polecen pliki mapa,status oraz rozkazy i "czas" wyrazony w sekundach.

6 Wnioski i uwagi

Realizacja zadania była wymagająca czasowo, lecz nie stanowiła głęboko trudnej przeszkody. W trakcie procesu implementacji napotkano wyzwanie związanego z nieprzewidywalnymi błędami, które czasem były trudne do dedukcji i wyjaśnienia na podstawie otrzymywanych informacji zwrotnych w korespondencji e-mailowej. Przyjrzenie się różnym mapom oraz uruchomienie programu na różnych komputerach miało miejsce, co przyczyniło się do osiągnięcia pewności, że projekt jest w fazie finalnej i działa sprawnie na każdej instancji.

Podkreślić należy, iż w aspekcie logiki rozgrywki gry turowej zidentyfikowano pewne luki. Z brakujących elementów kluczowymi są mechanizmy związane z pozyskiwaniem surowców oraz konwersją tychże surowców na jednostki. W gatunku gier turowych surowce pełnią rolę fundamentalną jako podstawa produkcji jednostek, a ich implementacja byłaby kluczowa dla uzyskania pełni realizmu mechaniki gry. Niezwykle istotne byłoby uwzględnienie różnych rodzajów surowców, takich jak kamień, drewno czy podobne, aby odzwierciedlić różnorodność i złożoność procesu produkcji jednostek.

Wśród elementów, które można by wprowadzić dla wzbogacenia rozgrywki, znajdują się jednostki przeznaczone do eksploatacji surowców, takie jak drwale. Rozważenie wprowadzenia jednostek, które poprawiają atrybuty innych jednostek, jak na przykład palladyni, którzy byliby zdolni do uzdrawiania jednostek w swoim otoczeniu, albo pracownicy, których obecność w bazie przyspieszałaby proces produkcji jednostek, stanowi wartościowe posunięcie w kontekście zróżnicowania strategii oraz tworzenia bardziej zaawansowanych mechanik interakcji między jednostkami.

Podsumowując, pomimo wymaganego nakładu czasu, realizacja zadania przebiegła sprawnie. W trakcie procesu napotkano wyzwania związane z identyfikacją i usuwaniem błędów, jednakże po owocnym udziale w procesie iteracyjnym, ostateczna wersja projektu jest zdolna do działania w różnych warunkach i spełnia wyznaczone kryteria. Rozważając dalszy rozwój, wprowadzenie opisanych brakujących elementów oraz rozbudowa interakcji między jednostkami mogą znacząco wzbogacić rozgrywkę i stworzyć bardziej satysfakcjonujące doświadczenie dla graczy.