ZPR 2024Z Sprawozdanie z Projektu Semestralnego Heroes AI

Zespół Segmentation Fault: Wojciech Sarwiński, Maciej Scheffer

26 stycznia 2025

1 Temat

Uproszczona wersja Heroes III, bez oprawy dźwiękowej, animacji ani wielu ras. Funkcjonalność sprowadza się do grywalnej wersji gry człowieka z komputerem wraz z uproszczonym odkrywaniem mapy, zdobywaniem zasobów oraz prowadzeniem walk.

2 Funkcjonalność

Z założonej funkcjonalności zadania z powodów czasowych nie udało się zrealizować: rozwoju bohatera, wyświetlania ścieżki na mapie oraz ekranu porażki. Statystyki jednostek są zatem statyczne, a warunkiem wygranej jest pokonanie wszystkich przeciwników oraz zdobycie wszystkich zasobów, porażka w walce usunie przeciwnika z mapy, aby możliwe było ukończenie rozgrywki.

3 Problemy na które natrafiliśmy

- **Deficyt czasu:** Największym problemem okazał się ograniczony czas w semestrze, jednak dzięki wyjątkowo dobrej komunikacji i organizacji pracy udało się zrealizować znaczną większość funkcjonalności oraz dopracować prezentację gry.
- **Planowanie systemu:** Projekt, z uwagi na charakter tworzenia gier komputerowych, składa się z dużej liczby współpracujących klas. Momentami trudno było nam przestrzegać zasad SOLID oraz KISS. Problematyczne okazało się również szczegółowe zaplanowanie struktury klas, co zmusiło nas do zastosowania zwinnego podejścia do pracy. Reagowaliśmy na bieżąco na nowe pomysły, problemy i rozwiązania. Wykorzystanie wzorców projektowych oraz materiałów z wykładów pozwoliło rozwiązać większość problemów architektonicznych, choć zbliżające się terminy wpłynęły na obniżenie jakości kodu.
- **Zarządzanie pamięcią:** Był to nasz pierwszy tak duży projekt w języku C++, co spowodowało trudności w pracy ze wskaźnikami. Często używaliśmy nieoptymalnych rozwiązań, takich jak niepotrzebne shared pointery, lub próbowaliśmy korzystać z raw pointerów. Ostatecznie, po analizie, ujednoliciliśmy zarządzanie pamięcią, stawiając głównie na unique pointery.
- **Problemy z merge'ami i ciągły refactor:** Częstym błędem było wprowadzanie zbyt dużych zmian w jednym commicie, co prowadziło do trudności przy scalaniu gałęzi. Dodatkowo, częste zmiany w koncepcjach klas wynikające z modyfikacji założeń projektowych również komplikowały ten proces. Ostatecznie udało się wszystko ujednolicić podczas długich sesji merge'owania oraz wspólnych burz mózgów.

3.1 Wnioski wynikające z napotkanych problemów

Projekt był dużym wyzwaniem dla nas obojga, jednak również bardzo dużo nas nauczył. Teraz wiemy, że nawet długie planowanie projektu ostatecznie przekłada się na czas zaoszczędzony na refactorowaniu. Zapoznaliśmy się także jak w praktyce wygląda zarządzenie pamięcią w tak dużym systemie i że wymaga ono osobnego planowania, tak aby całość była spójna, nie powodowała żadnych wycieków pamięci ani cyklicznych referencji. Dzięki temu projektowi będziemy teraz o wiele bardziej świadomi wyzwań czekających przy większych projektach C++ oraz będziemy na nie lepiej przygotowani.

4 Dokumentacja Użytkownika

4.1 Kompilacja programu

Program najprościej skompilować używając gotowych plików boot.sh lub boot.bat w zależności od systemu operacyjnego. Pliki nazwane są lekko inaczej oraz mają rozszerzenie .txt aby możliwe było ich przesyłanie serwisem Outlook.

- **Windows:** Najpierw należy usunąć plik CMakeLists.txt oraz zmienić nazwę pliku CMakeLists-Windows.txt na CMakeLists.txt. Przed uruchomieniem pliku boot.bat należy zainstalować program vcpkg, a następnie za jego pomocą zainstalować pakiety: **sdl2**, **sdl2_image**, **sdl2_ttf** oraz **boost_test**. Po poprawnym zainstalowaniu pakietu w pliku boot.bat należy podmienić zmienną VCPKG na ścieżkę do pliku vcpkg.cmake. Po takim skonfigurowaniu środowiska z poziomu foldera z projektem komendą **./boot.bat build** tworzymy pliki wykonywalne. Komendą **./boot.bat main** uruchamiamy skompilowaną grę, a komendą **./boot.bat tests** uruchamiamy testy.
- **Ubuntu:** Budowa projektu możliwa jest za pomocą zautomatyzowanego skryptu **boot.sh**, komendą **sh boot.sh**, **lub po komendzie chmod 777 boot.sh uruchomić plik boot.sh** który wykonany z poziomu projektu zainstaluje niezbędne zależności, oraz zbuduje główny plik wykonywalny jak i wszystkie testy. Po poprawnym zbudowaniu plików zostaną one kolejno uruchomione.

4.2 Przebieg rozgrywki

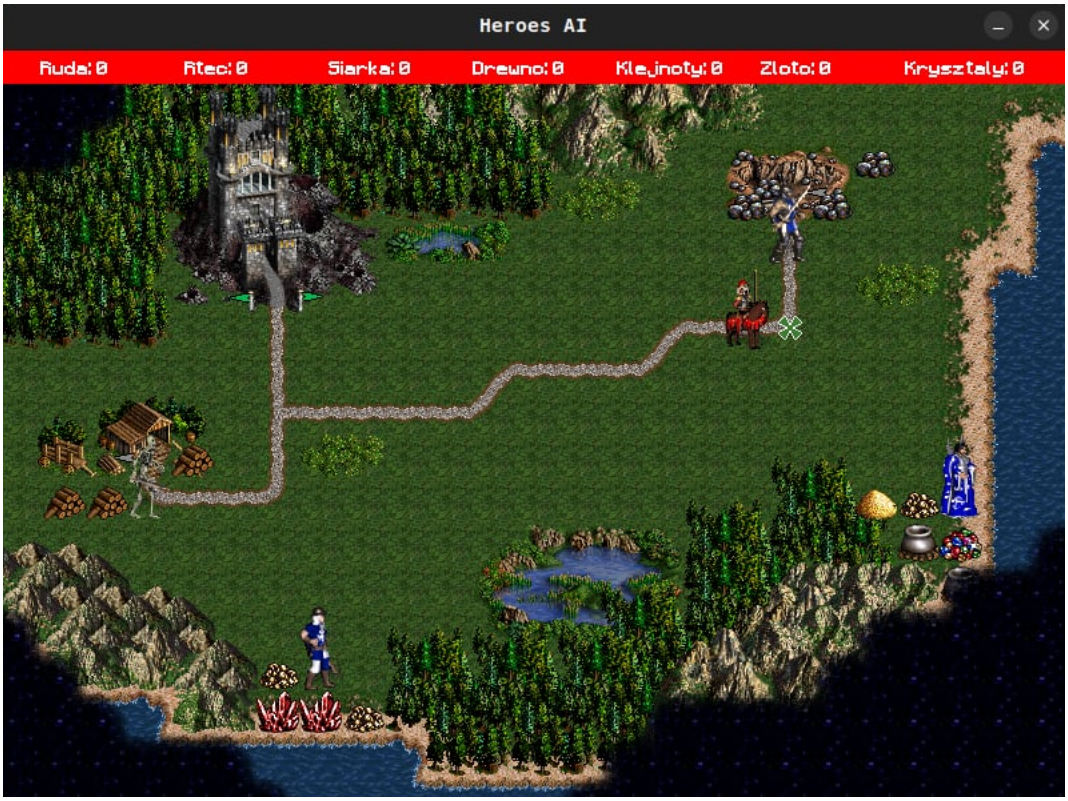
Gra dzieli się na dwie fazy: fazę eksploracji oraz fazę walki, które następują po sobie. Celem gry jest wyeliminowanie wszystkich przeciwników na mapie oraz zdobycie wszystkich dostępnych na niej zasobów.

4.2.1 Poruszanie się po mapie

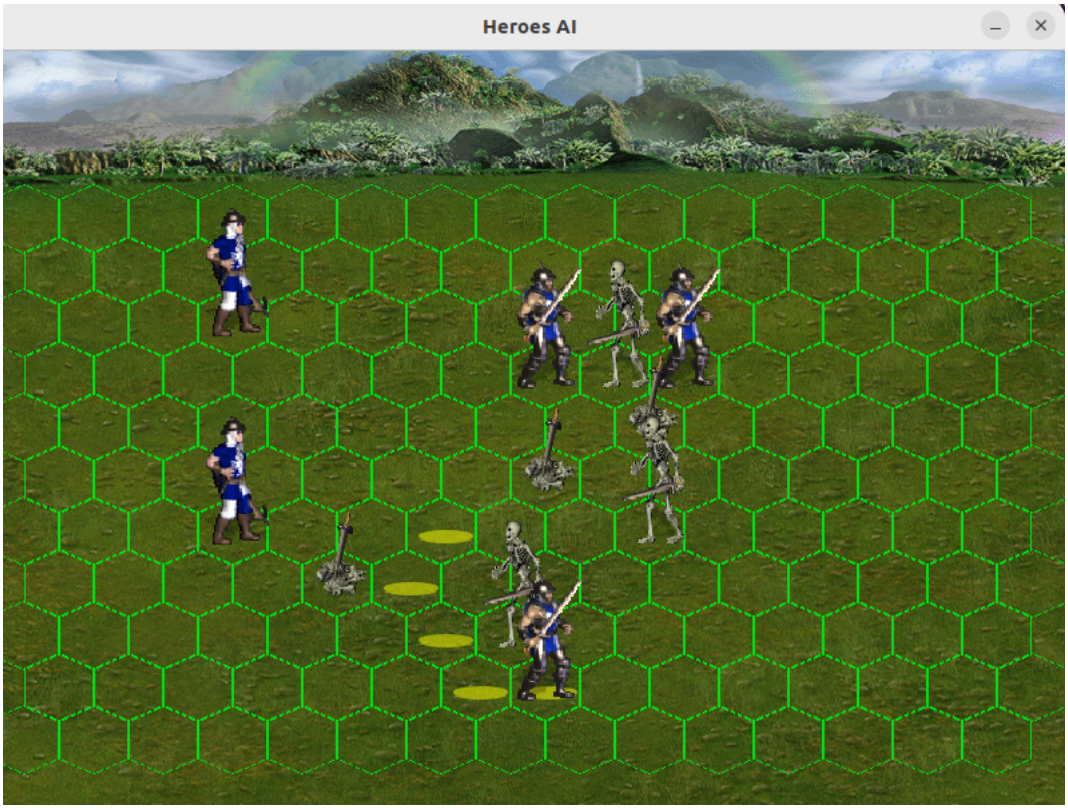
Aby poruszać się na mapie należy wybrać lewym przyciskiem pole, na które chcemy się poruszyć. Ruch potwierdzamy przez ponowne naciśnięcie powstałego na wybranym polu wskaźnika. Walkę z przeciwnikiem rozpoczynamy w momencie ustawienia swojego bochatera na polu obok lub bezpośrednio na polu przeciwnika.

4.2.2 Wykonywanie ruchów na polu bitwy

Ruch jednostki składa się z dwóch ruchów: przemieszczenie się na inne pole i zaatakowanie jednostki przeciwnika. Na początku naszego ruchu wskaźniki na mapie symbolizują pola, na które aktualna jednostka może się przesunąć. Możemy wybrać jedno z tych pól lewym przyciskiem myszy, co wyświetli nam ścieżkę, po której przemieści się jednostka. Przemieszczenie się potwierdzamy ponowmym wybraniem wcześniej wybranego pola. Można odznaczyć wybrane pole poprzez wybranie innego pola na mapie. Po przemieszczeniu się jednostki nastąpi tura ataku. Wskaźniki powinny przedstawić nam jednostki, które są w odległości maksymalnego zasięgu naszego ataku. Możemy zaznaczyć jednostkę, którą chcemy zaatakować lub jeśli nie chemy atakować albo nie ma żadnych jednostek w zasięgu ataku możemy zaznaczyć naszą aktywną jednostkę co symbolizuje rezygnacje z ruchu. Wybór potwierdzamy poprzez ponowny wybór zaznaczonego wcześniej pola. Komunikaty ruchów jednostek oraz statystki związane z działaniem algorytmu minimax będą logowane w terminalu.



Rysunek 1: Widok Fazy Eksploracji



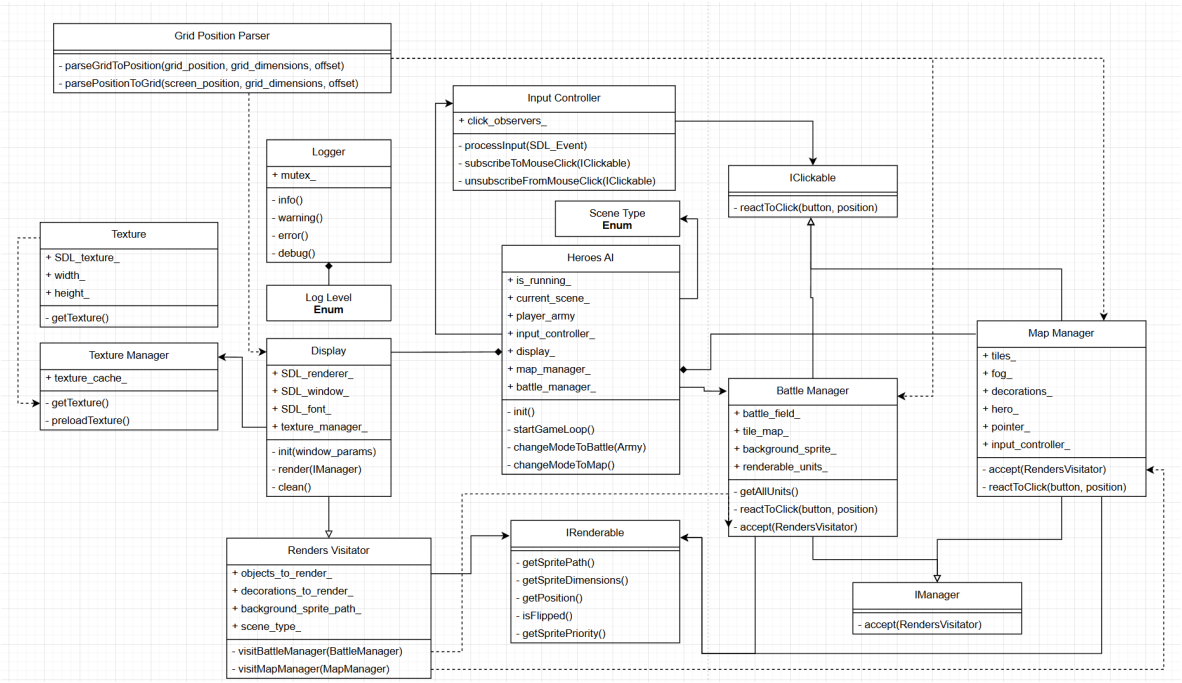
Rysunek 2: Widok Fazy Walki

5 Jakość Kodu

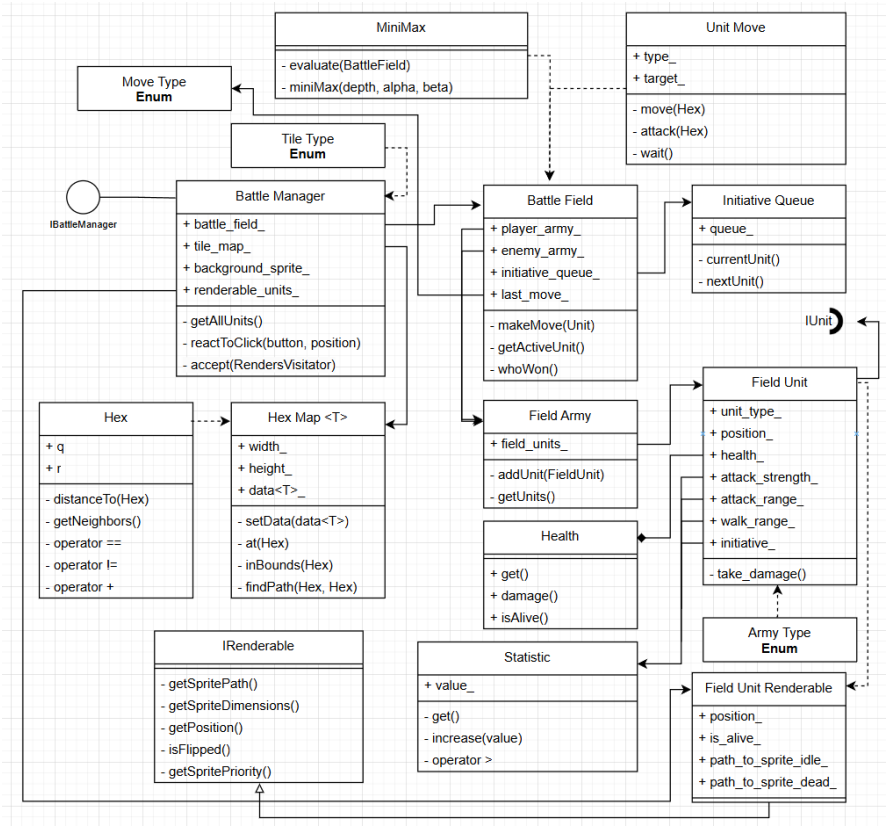
Poprzez rozmiar projektu oraz ograniczony czas bardzo ciężko nam było zachować spójny styl kodowania, trzymać się zasad SOLID, oraz spójnie zarządzać pamięcią, jednak dzięki trzymaniu się narzuconego stylu, wykorzystaniu wzorców oraz sprytnych wskaźników udało nam się zachować wyznaczone wymogi.

5.1 Struktura Programu

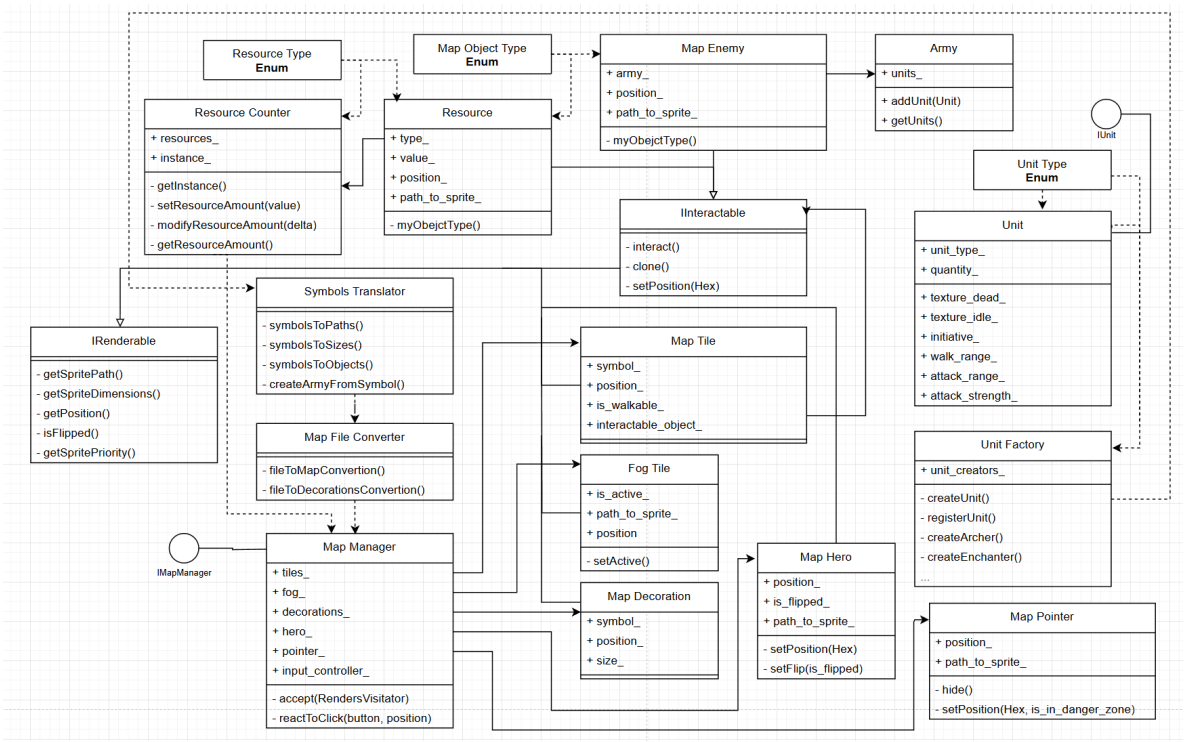
Najważniejszymi filarami programu jest klasa zarządzająca pętlą gry **HeroesAI**. Posiada ona wskaźniki na klasę zarządzającą widokiem mapy **MapManager**, klasę zarządzającą widokiem bitwy **BattleManager**, klasę odpowiedzialną za wyświetlanie wszystkich tekstur **Display** oraz kontroler wejścia myszki **InputController**. W programie obecne jest także wiele innych klas będących częściami powyższych, odciążających je z nadmiernej funkcjonalności, oraz reprezentujące poszczególne obiekty w grze.



Rysunek 3: Diagram UML najważniejszych klas projektu



Rysunek 4: Diagram UML modułu walki



Rysunek 5: Diagram modułu eksploracji

5.2 Wykorzystane Rozwiązania

W projekcie wykorzystaliśmy wiele rozwiązań z wykładów, wliczając w to wzorce projektowe, sprytne wskaźniki oraz algorytmy.

- **Algorytm minimax z przycinaniem alfa-beta i tabelą transpozycji:** wykorzystany przez nas do implementacji ruchów komputera w widoku walki. Algorytm bierze pod uwagę jednostki gracza jak i komputera, kolejność tur oraz statystyki jednostek takie jak zdrowie, zasięg ruchu, obrażenia oraz zasięg ataku.
  - **Funkcja heurystyczna:** Heurystyczna jakość pozycji jest szacowana na podstawie sumy życia jednostek pomnożona przez ich siłę ataku.
  - **Tabela transpozycji:** Pozycje po ewaluacji są hashowane i końcowa wartość ewaluacji wraz z głębią tej ewaluacji są zapisywane do tabeli transpozycji i przed ewaluacją kolejnych pozycji sprawdzamy, czy ewaluacja pozycji została już wykonana na zadanej głębi lub lepiej.
  - **Przycinanie alfa-beta:** Ruchy są sortowane po ich rodzaju: atak, przemieszczenie się, rezygnacja z ruchu, w celu polepszenia efektywności przycinania alfa-beta.
  - **Asynchroniczna ewaluacja:** algorytm minmax jest wykonywany przy pomocy `std::async`, aby działać asynchronicznie, co pozwala na to, aby program pozostał responsywny, podczas gdy zadanie jest przetwarzane w tle.
- **Algorytm BFS:** pozwala odnaleźć możliwy ruch jednostek na heksagonalnej planszy. Na polu bitwy nie znajdują się przeszkody, ale poprawnie uwzględnia pozycje zajęte przez inne jednostki.
- **Wzorce projektowe:** przykładowe wzorce użyte w projekcie to: **Odwiedzający** (zbieranie ścieżek do tekstur obiektów), **Singleton** (manager zasobów gracza), **Obserwator** (subskrypcja wydarzenia kliknięcia myszki), **Fabryka** (tworzenie nowych jednostek do armii komputera oraz gracza)
- **Sprytne wskaźniki:** duży nacisk kładliśmy na jak największe ograniczenie używania raw pointerów zamieniając je na sprytne wskaźniki, przez co nigdy nie potrzebne było użycie delete. Niekiedy jednak użycie raw pointerów było nieuniknione ponieważ używana przez nas biblioteka graficzna SDL2 ich wymagała, zadbaliśmy zatem aby wskaźniki te były prawidłowo usuwane w destruktorach odpowiednich obiektów.
- **Szablony:** W projekcie obecny jest również szablon HexMap<T> umożliwiający przechowanie dowolnego obiektu w strukturze heksagonalnej siatki wykorzystanej na polu bitwy. Szablon ten jest wykorzystywany np. przy algorytmie BFS do przechowywania wartości czy pola zostały już odwiedzone i współrzędne Hex pól poprzednich.

5.3 Czytelność Kodu

Pomimo tego, że w zespole mieliśmy odmienne style kodowania i nawyki, ujednoliciliśmy styl projektu dopasowując go do wymagań stylu kodowania ze strony prowadzącego.

6 Testy Jednostkowe

6.1 Jakość Testów

Przez charakter projektu ciężko było nam zachować duże pokrycie kodu testami ponieaż wiele z funkcji jest zależne od aktualnego stanu aplikacji lub służą wykonywaniu pomniejszych elementów większych, istniejących już obiektów. Prawidłowość działania nieprzetestowanych funkcji sprawdzaliśmy mechanizmem logowania zrealizowanego przez naszą klasę **Logger.h**

6.2 Automatyzacja

Testy uruchamiane są automatycznie w skrypcie boot.sh, jednak możliwe jest również uruchomienie poszczególnych testów za pomocą cmake.

7 Pracochłonność Projektu

Do tabeli powinniśmy jeszcze dopisać wiele godzin spędzonych na refactorowaniu, mergowaniu testowaniu oraz planowaniu, jednak z racji, iż trudno zmierzyć ten czas, postanowiliśmy trzymać się tabeli z dokumentacji wstępnej.

Zadanie	**Zakładane**	**Faktyczne**
Utworzenie heksagonalnej siatki pola bitwy z podstawowymi metodami	3	6
Implementacja pola bitwy opartego na heksagonalnej siatce	5	10
Implementacja klas jednostek ze statystykami	2	8
Możliwość rozstawienia jednostek na polu bitwy	3	2
Obliczenie możliwych pól ruchu jednostki algorytmem BFS	2	4
Możliwość przejścia jednostki na dozwolone pole	2	1
Implementacja kolejki tur jednostek	2	2
Implementacja ataku jednostek	2	2
Wykonywanie optymalnych ruchów w pozycji metodą min-max	10	16
Utworzenie kwadratowej siatki służącej do przemieszania się po mapie	2	1
Implementacja algorytmu Dijkstry do wyznaczania optymalnych ścieżek	2	-
Ładowanie mapy z pliku	5	6
Poruszanie się po mapie bohaterem	1	3
Implementacja stanu zasobów	1	1
Implementacja zbierania zasobów na mapie	2	3
Implementacja "mgły wojny"	4	2
Implementacja rozwoju bohatera	2	-
Możliwość potyczki z napotkanymi przeciwnikami	2	2
Wyświetlenie pola bitwy	10	16
Reagowanie na wejście z myszki na polu bitwy	5	4
Wyświetlenie pól, na które może poruszyć się jednostka na polu bitwy	2	4
Wyświetlenie optymalnej ścieżki bohatera	3	-
Wyświetlenie widoku mapy	10	12
Wyświetlenie zasobów na mapie	2	3
Wyświetlenie przeciwników na mapie	2	3
Wyświetlenie widoku zasobów	2	1
Reagowanie na wejście z myszki mapie	5	3
Wyświetlenie panelu rozwoju bohatera	3	-
Przejście z trybu eksploracji na tryb walki	2	2
Powrót z trybu walki na tryb eksploracji	1	2
Wyświetlenie panelu zwycięstwa	2	2
Wyświetlenie panelu porażki	2	-
SUMA GODZIN	**103**	**121**

Tabela 1: Tabela zakładanych oraz faktycznych godzin pracy nad projektem