JOURNAL
COMPUTER SCIENCES INSTITUTE

JCSI 20 (2021) 217–224
Received: 21 June 2021
Accepted: 29 June 2021

Comparative analysis of the proprietary navigation system and the built-in Unity engine tool

Analiza porównawcza autorskiego systemu nawigacji i wbudowanego narzędzia silnika Unity

Maciej Kempny*, Marcin Barszcz

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract
Navigation systems are advanced tools that allow you to create characters intelligently moving around the game world. The purpose of this thesis was to conduct a comparative analysis of the proprietary navigation system "AlchemyNavigation" and the built-in tool of the Unity engine - "NavMesh". An proprietary application created with the Unity engine was used to conduct the research, under which identical scenarios based on the tested systems were implemented. Finally, on the basis of the collected results and documentation of the research objects, a comparative analysis was carried out and proved the thesis that own solutions can match the default solutions.

Keywords: navigation systems; unity; navmesh; alchemynavigation

Streszczenie
Systemy nawigacyjne to zaawansowane narzędzia, które pozwalają tworzyć postacie inteligentnie poruszające się po świecie gry. W ramach niniejszej pracy przeprowadzono analizę porównawczą autorskiego systemu nawigacji o nazwie AlchemyNavigation oraz wbudowanego narzędzia silnika Unity - systemu NavMesh. Do przeprowadzenia badań wykorzystano autorską aplikację zbudowaną przy pomocy silnika Unity, w ramach której zaimplementowano tożsame scenariusze badawcze bazujące na porównywanych systemach. Finalnie, na podstawie zebranych wyników oraz dokumentacji badanych systemów zrealizowana została analiza porównawcza, która dowiodła tezy że własne rozwiązania mogą dorównać rozwiązaniom domyślnym, a także, że pozwalają wprowadzać nowe funkcjonalności.

Słowa kluczowe: systemy nawigacji; unity; navmesh; alchemynavigation

*Corresponding author
Email address: maciej.kempny@pollub.edu.pl (M. Kempny)

1. Wstęp

W dzisiejszych czasach twórcy gier oraz symulacji bardzo rzadko decydują się na tworzenie własnych rozwiązań dla problemów, które pojawiają się w większości tego typu produkcji. Kierunek rozwoju technologii sprawił, że na rynku pojawiły się gotowe narzędzia ogólnego przeznaczenia, które niższym kosztem i nakładem pracy pozwalają na osiągnięcie zamierzonych (bądź zbliżonych do zamierzonych) efektów. Przykładem takiego zjawiska jest Unity - silnik gier, który między innymi dostarcza rozwiązań dla problemów takich jak: proces renderowania, symulacja fizyki oraz nawigacja agentów. Poza wykorzystaniem wbudowanych elementów Unity pozwala również na tworzenie własnych, które mogą kontrolować, uzupełnić bądź zastąpić gotowe rozwiązania.

Systemy nawigacji to zaawansowane narzędzia pozwalające na tworzenie postaci, które mogą, w inteligentny sposób, poruszać się po świecie gry bądź symulacji. Są to rozwiązania wielopoziomowe, otóż sprawny system powinien zapewniać następujące funkcjonalności: określenie powierzchni nawigacyjnej, wytyczanie optymalnych ścieżek oraz naturalny ruch agentów. NavMesh to wbudowany system nawigacji silnika Unity. W niniejszej pracy zostanie on porównany pod kątem wydajności i uniwersalności z autorskim rozwiązaniem o nazwie AlchemyNavigation.

Wymienione poniżej pozycje literaturowe dowodzą że tematy związane z nawigacją w grach komputerowych i robotyce nie są obce w zbiorach prac naukowych. Nie mniej jednak są to tematy bardzo złożone, i z tego powodu większość z dostępnych materiałów skupia się na badaniu i ulepszaniu tylko konkretnych elementów wchodzących w skład tego zagadnienia. Natomiast temat porównania wydajności i funkcjonalności całych systemów jest poruszany bardzo rzadko.

W artykule [1] omawiany jest problem kooperacyjnego wyszukiwania ścieżek - wyszukiwania i planowania ścieżek dla wielu agentów, w taki sposób aby ograniczyć zachodzące między nimi kolizje. Autor zwraca uwagę na to jak ważna, w kontekście nowoczesnych gier, jest funkcjonalność unikania kolizji oraz redukcja generowanego przez nią obciążenia podzespołów stacji komputerowych. Prezentowane jest nowe podejście – CA* (Cooperative A*), które pozwala na efektywne rozwiązywanie tego problemu, oraz badania poddające analizie jego funkcjonalności. Ostatecznie udowodniono tezę, że zaproponowane przez autora rozwiązanie jest efektywne i pozwala na lepsze wybieranie ścieżki niż inne analizowane algorytmy.

W artykule [2] prezentowana jest metoda polegająca na przedstawieniu przestrzeni nawigacyjnej w oparciu o ograniczoną triangulację Delaunay (ang. constrained Delaunay triangulation), dzięki której możliwe jest znaczące zmniejszenie złożoności procesu wyszukiwania ścieżek. Autor w swoich badaniach skupia się na udowodnieniu przewagi przedstawianej metody nad innymi znanymi sposobami, oraz potwierdzeniu jej skuteczności i przetestowaniu jej w różnych scenariuszach badawczych.

Waga problemu wygładzania ścieżek tworzonych przez algorytmy wyszukiwania trasy jest bardzo duża w kontekście systemów nawigacji. Autor artykułu [3] motywuje tą tezę oraz tłumaczy w jaki sposób powinna wyglądać ścieżka, aby agenci mogli z niej poprawnie korzystać. Jednak, głównym celem artykułu i badań jest omówienie metody, która bazuje na kwadratowej interpolacji i pozwala na uzyskanie odpowiedniego efektu. W toku pracy działanie algorytmu i jego atuty zostają dokładnie przedstawione.

Artykuł [4] został poświęcony przeglądowi i analizie zagadnienia wyszukiwania ścieżek. Autor podejmuje próbę przedstawienia, w prostej formie, elementów oraz sekwencji operacji składających się na proces wytyczania trasy. Szczegółowo omówione są różne etapy procesu, logika stojąca za najbardziej popularnymi algorytmami (takimi jak: algorytm A* lub algorytm Dijkstry). Przeglądowi poddane zostały również algorytmy rozszerzające, które zapewniają funkcjonalności takie jak: płynny ruch agentów i unikanie kolizji. Finalnie, dla wybranych rozwiązań, przedstawione są również sposoby dzięki którym możliwe jest uzyskanie większej efektywności.

W artykule [5] autorzy dokonują analizy działania i sposobów wykorzystywania systemu nawigacji dostarczonego przez silnik gier Unity. Szczegółowo omawiane są zastosowane algorytmy, sposób w jaki są wykorzystywane oraz jaki wpływ ma to na rozwój gier komputerowych

Finalne na rynku dostępnych jest również wiele książek odnoszących się do tematyki nawigacji [6]. Pojawiają się tytułu dotyczące sztucznej inteligencji - ruchu i wyszukiwania ścieżek w grach [7], zestawy gotowych do implementacji rozwiązań dla konkretnych platform [8], oraz teorii stojącej za popularnymi rozwiązaniami [9]. Ostatecznie często spotykane są również tytuły przeznaczone dla nowicjuszy wprowadzające w tajniki wykorzystywania silników gier w ramach, których rozwijany jest temat nawigacji [10], bądź zestawy wiedzy dla bardziej zaawansowanych użytkowników [11].

2. Lista cech uniwersalnego systemu nawigacji

Lista cech uniwersalnego systemu nawigacji (tabela 1) została opracowana na podstawie przeglądu gier dostępnych aktualnie na rynku.

Tabela 1: Lista cech uniwersalnego systemu nawigacji

Lp.	Cecha systemu nawigacji
1	Możliwość tworzenia predefiniowanych części przestrzeni nawigacyjnej
2	Możliwość tworzenia i modyfikowania przestrzeni nawigacyjnej w czasie wykonania programu
3	Możliwość wprowadzenia podziału przestrzeni nawigacyjnej na mniejsze elementy (ang. chunks) i płynnego włączania i wyłączania ich
4	Możliwość tworzenia agentów przemieszczających się w naturalny sposób po przestrzeni nawigacyjnej
5	Możliwość ustalania parametrów ruchu dla różnych agentów
6	Możliwość przypisywania wag decydujących o częstości wybierania określonych fragmentów przestrzeni nawigacyjnej w procesie poszukiwania ścieżki
7	Możliwość definiowania alternatywnych powierzchni nawigacyjnych dla różnego typu agentów
8	Zapewnienie funkcji unikania kolizji dla agentów
9	Wsparcie dla dużej liczby (rzędu setek) agentów jednocześnie
10	Możliwość kontrolowania przemieszczania się agentów pomiędzy oddzielonymi fragmentami przestrzeni nawigacyjnej

3. Obiekt badań

Ten rozdział został poświęcony opisowi badanych systemów, który został opracowany w oparciu o ich dokumentacje techniczne [12, 13].

3.1. AlchemyNavigation

AlchemyNavigation to autorski system nawigacji, który może zostać dołączony do projektu Unity za pomocą narzędzia Package Manager. Charakteryzuje się rozbudowanymi funkcjonalnościami, które pozwalają na tworzenie i modyfikowanie przestrzeni nawigacyjnej w czasie wykonania programu oraz przystosowanymi do nadpisywania modułami.

Najważniejszym elementem systemu jest komponent AlchemyNavigationSystem, który zapewnia działanie wszystkich kluczowych funkcji, oraz pozwala na dostosowanie ustawień nawigacji, w tym przedstawia podział przestrzeni nawigacyjnej na 32 powierzchnie i warstwy.

System oferuje trzy sposoby tworzenia przestrzeni nawigacyjnej, są to:

1. Tworzenie w oparciu o skrypty - użytkownik może pisać skrypty, które w dowolnym momencie rozszerzają przestrzeń nawigacyjną o kolejne trójkąty. Każda operacja dodania zwraca unikalny uchwyt, który można wykorzystać do usunięcia dodanego trójkąta.
2. Tworzenie z poziomu komponentu FacesHolder - komponentu wysokiego poziomu, który ukrywa złożoność tworzenia przestrzeni nawigacyjnej w oparciu o skrypty - pozwala na wizualne modyfikowanie trójkątów za pomocą edytora. Dodatkowo komponent wspiera wszystkie domyślne funkcje obiektów gry Unity, to znaczy w dowolnym momencie może być włączany, wyłączany, przemieszczany, rotowany i skalowany, a dokonane zmiany zostaną odzwierciedlone na aktywnej powierzchni nawigacyjnej.
3. Wypalanie na podstawie geometrii sceny - w procesie wypalania, użytkownik może tworzyć przestrzeń nawigacyjną w oparciu o geometrię znajdującą się

na scenie gry. Proces ten może być realizowany statycznie lub dynamicznie. Statycznie wypalanie wykonywane jest za pomocą narzędzi edytora, a jego rezultatem jest zestaw obiektów z dodanymi komponentami FacesHolder (które mogą być dalej modyfikowane przez użytkownika). Natomiast wypalanie dynamiczne (w czasie działania programu) wykonywane jest za pomocą skryptów, a jego rezultatem są zestawy trójkątów, które mogą zostać dodane do przestrzeni nawigacyjnej.

Na etapie tworzenia agentów, system pozwala na obranie jednej z dwóch dróg: tworzenie i wykorzystanie własnych niestandardowych agentów, lub wykorzystanie wbudowanych prostych agentów. Prości agenci zapewniają standardowy ruch, który jest odpowiedni dla większości postaci spotykanych w grach komputerowych. Dodatkowo ich zachowanie można modyfikować poprzez dodawanie specjalnych komponentów nazywanych modyfikatorami ruchu. Ta funkcja może zostać wykorzystana do implementacji zachowania polegającego na unikaniu kolizji.

3.2. NavMesh

NavMesh to system nawigacji dostępny domyślnie w każdym projekcie rozwijanym przy pomocy Unity. Charakteryzuje się bardzo wysokim poziomem integracji ze środowiskiem i uniwersalnością. Sam system składa się czterech głównych elementów, są to:

1. NavMesh - struktura danych opisująca powierzchnię, po której mogą poruszać się agenci.
2. NavMesh Agent - komponent pozwalający na tworzenie postaci, które mogą poruszać się po powierzchni nawigacyjnej oraz unikać kolizji z innymi postaciami i przeszkodami.
3. NavMesh Obstacle - komponent pozwalający na tworzenie dynamicznych przeszkód, które powinny być unikane przez agentów.
4. Off-Mesh Link - komponent pozwalający na tworzenie skrótów pomiędzy miejscami znajdującymi się na powierzchni nawigacyjnej.

Poza głównymi składnikami, Unity pozwala również na skorzystanie z wysokopoziomowych narzędzi, które dostarczają dodatkowej kontroli nad procesem budowania powierzchni nawigacyjnej (w czasie wykonania programu oraz w edytorze), są to:

1. NavMesh Surface - komponent pozwalający na budowanie powierzchni nawigacyjnej dla jednego typu agenta oraz przełączanie jej aktywności.
2. NavMesh Modifier - komponent pozwalający na oznaczanie typu obszaru powierzchni nawigacyjnej w oparciu o hierarchię obiektów na scenie.
3. NavMesh Modifier Volume - komponent pozwalający na oznaczanie typu obszaru powierzchni nawigacyjnej w oparciu o zdefiniowany obszar.
4. NavMesh Link - komponent pozwalający na dynamiczne tworzenie połączeń między dwiema lokacjami znajdującymi się na powierzchniach określanych przez NavMesh Surface.

W ramach systemu NavMesh powierzchnia nawigacyjna tworzona jest w procesie wypalania, który pobiera komponenty renderujące ze wszystkich obiektów oznaczonych na scenie jako statyczne nawigacyjnie, a następnie przetwarza zawarte w nich dane dotyczące geometrii, i na tej podstawie aproksymuje powierzchnie po których mogą poruszać się agenci.

4. Metodyka badań

Badania zostały przeprowadzone w sposób identyczny na trzech różnych stacjach, które różnią się podzespołami i parametrami. Specyfikacja maszyn została przedstawiona w tabeli 2.

Tabela 2: Specyfikacja stacji wykorzystanych do przeprowadzenia badań

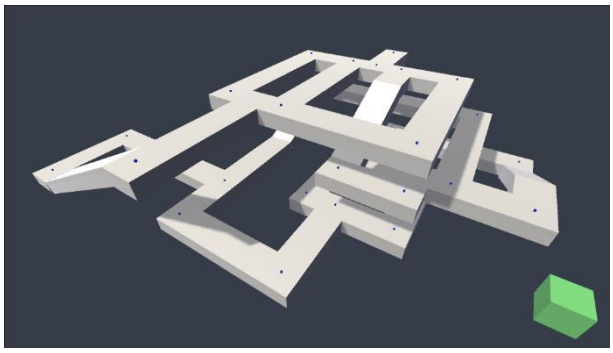
Numer stacji	Procesor	Taktowanie procesora (GHz)	Karta graficzna	Pamięć RAM (GB)
1	Intel Core i7-6700K	4,00	Nvidia GeForce GTX 1660 SUPER	32
2	Intel Core i7-8750H	2,20	Nvidia GeForce GTX 1050Ti	16
3	Intel Core i7-7200U	2,50	Nvidia GeForce 940MX	4

4.1. Badanie obciążenia procesora i pamięci RAM

Badania obciążenia procesora i pamięci RAM zostały przeprowadzone w oparciu o autorską aplikację, która wykorzystuje badane systemy oraz nie wykorzystuje systemów nawigacji (skala odniesienia). Kryteria porównawcze stanowią:

1. Średni czas trwania klatki liczony w sekundowych odstępach w dwudziestominutowym okresie działania aplikacji.
2. Średnia ilość pamięci RAM wykorzystywanej przez aplikację liczona w sekundowych odstępach w dwudziestominutowym okresie działania aplikacji.

Na potrzeby badań utworzony został projekt unity w ramach którego przygotowana została scena bazowa (rysunek 1), która posiada cechy opisane w tabeli 3. Przygotowane zostały również warianty sceny bazowej (tabela 4) różniące się aktywnym systemem nawigacji i jego konfiguracją.



Rysunek 1: Scena bazowa autorskiej aplikacji testowej.

Tabela 3: Cechy sceny bazowej

Lp.	Cecha sceny
1	Scena jest zbudowana ze stu prostopadłościennych obiektów.
2	Obiekty na scenie umieszczone są na różnych poziomach względem osi pionowej, w taki sposób że tworzą piętra.
3	Piętra są ze sobą połączone, w taki sposób że obiekty tworzą większą nie prymitywną figurę.

Tabela 4: Warianty sceny bazowej

Lp.	Wariant sceny
1	Brak aktywnego systemu nawigacji.
2	Aktywny system AlchemyNavigation (bez unikania kolizji).
3	Aktywny system AlchemyNavigation (z unikaniem kolizji).
4	Aktywny system NavMesh (bez unikania kolizji).
5	Aktywny system NavMesh (z unikaniem kolizji).

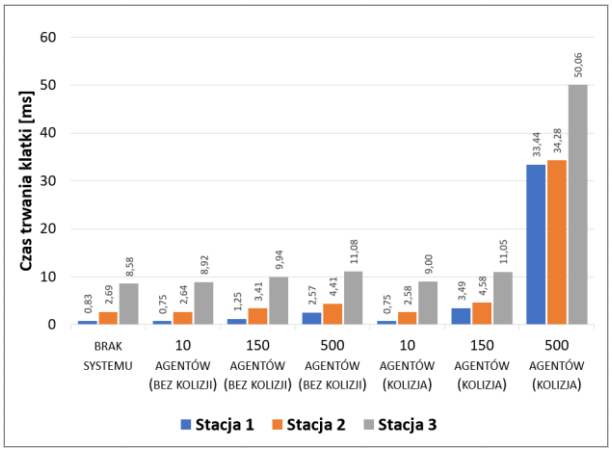
Na każdym z wariantów sceny bazowej wykonano pomiary dla 10, 150 i 500 agentów, na trzech różniących się parametrami maszynach. Do wykonania pomiarów wykorzystano skrypt bazujący na narzędziach do profilowania, które są dostępne w domyślnych bibliotekach Unity - Unity.Profiling [14].

4.2. Analiza porównawcza

Ta część badań skupia się na przeprowadzaniu analizy porównawczej cech badanych systemów w oparciu o ich dokumentację oraz wykonane pomiary. Szczególna uwaga została poświęcona cechom przedstawionym na liście cech uniwersalnego systemu nawigacji (tabela 1).

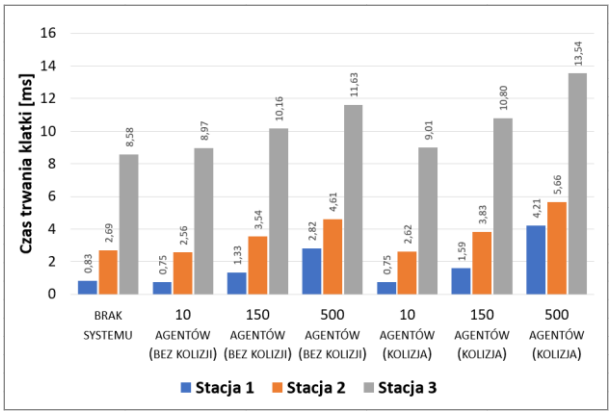
5. Wyniki badań i dyskusja

Na rysunkach 2 i 3 przedstawiono średnie wyniki pomiarów wykonanych dla badanych systemów pod kątem czasu trwania klatki.



Rysunek 2: Średni czas trwania klatki dla systemu AlchemyNavigation.

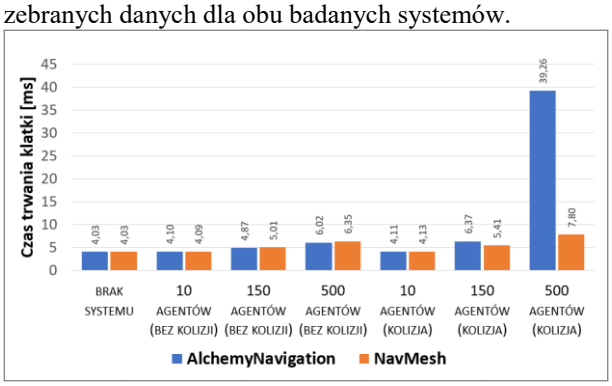
Na podstawie otrzymanych wyników badań dla systemu AlchemyNavigation można stwierdzić, że podzespoły maszyny i liczba agentów mają zauważalny wpływ na czas trwania klatki. Wartym uwagi jest również fakt, że w przypadku większych liczb aktywnych agentów, włączona funkcja unikania kolizji powoduje gwałtowny wzrost wykorzystania procesora. Dla stu pięćdziesięciu agentów największa różnica wyniosła 2,24 ms, natomiast dla pięciuset agentów aż 38,98 ms.



Rysunek 3: Średni czas trwania klatki dla systemu NavMesh.

W oparciu o wyniki dla systemu NavMesh można stwierdzić, że liczba agentów i podzespoły maszyny mają widoczny wpływ na czas trwania klatki. Wartym uwagi jest również fakt, że wzrost obciążenia procesora generowany poprzez włączenie funkcji unikania kolizji jest zauważalny, ale stabilny i niewysoki (w najgorszej próbie ~50%).

Na rysunku 4 przedstawiono uśrednione porównanie zebranych danych dla obu badanych systemów.



Rysunek 4: Porównanie uśrednionych czasów trwania klatki pobranych ze wszystkich stacji dla badanych systemów.

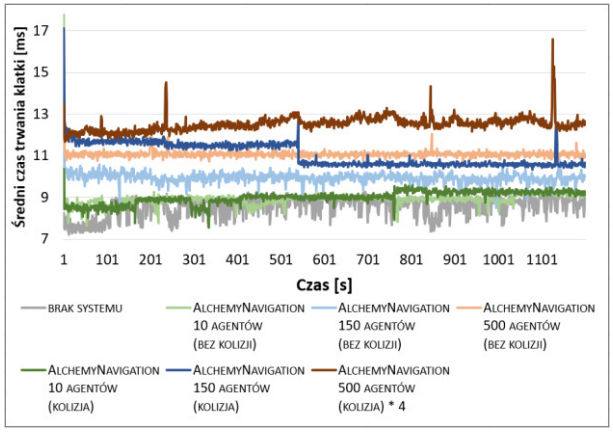
Pierwszym nasuwającym się wnioskiem jest fakt, że w przypadku niskiej liczby agentów (10), niezależnie od systemu i tego czy funkcja unikania kolizji jest włączona, wzrost czasu klatki jest niemalże niezauważalny - maksymalnie ~2%. System AlchemyNavigation jest nieznacznie szybszy, gdy funkcja unikania kolizji jest wyłączona. W przeciwnej sytuacji, system NavMesh zużywa mniej zasobów procesora i jest bardziej stabilny, w szczególności kiedy liczba aktywnych agentów jest wysoka (różnica 31,46 milisekund w próbie 500 agentów z aktywnym unikaniem kolizji).

Na rysunkach 5 i 6, przedstawiono średnie przebiegi czasowe prób, odpowiednio dla systemu AlchemyNavigation i NavMesh.

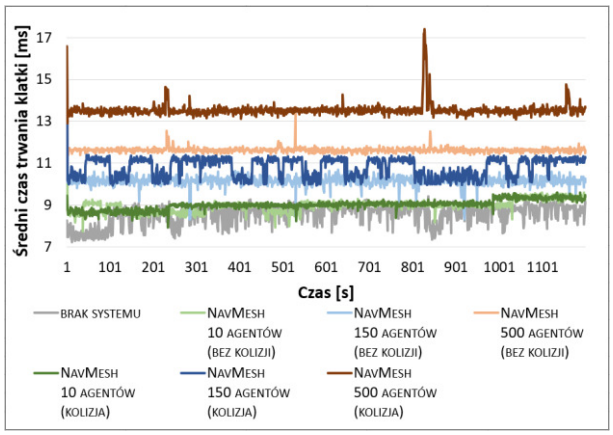
Na podstawie przebiegów czasowych obu badanych systemów, można zauważyć następującą zależność - próby z wyłączoną funkcją omijania kolizji są stabilniejsze - zauważalne zmiany występują rzadziej i są mniejsze. W przypadku kiedy unikanie kolizji jest włączone i liczba agentów wynosi 150 lub 500, można również dostrzec tendencje do okresowych zmian długości czasu trwania klatki. Na przykład, w próbie stu

pięćdziesięciu agentów systemu AlchemyNavigation wartości początkowo stabilizują się na poziomie 11,5 ms, a następnie na poziomie 11 ms. Natomiast w tożsamej próbie systemu NavMesh wartości naprzemiennie stabilizują się na poziomie 9,5 ms i 11,5 ms. Tendencja ta może wynikać z faktu, że na wąskich korytarzach mapy tworzą się zatory i tego jak systemy radzą sobie z kontrolą tłumu.

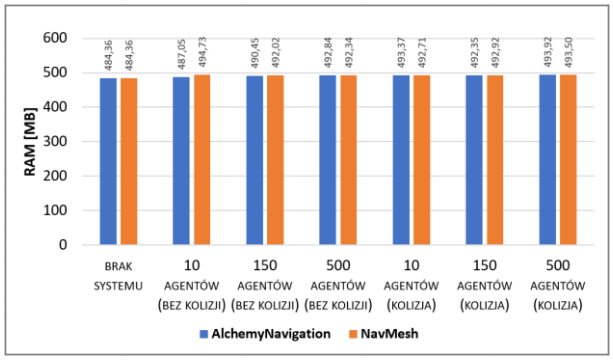
Na rysunku 7 porównano średnie wartości użycia pamięci RAM uzyskane we wszystkich próbach.



Rysunek 5: Średnie wartości czasu trwania klatki (ms) prób systemu AlchemyNavigation.



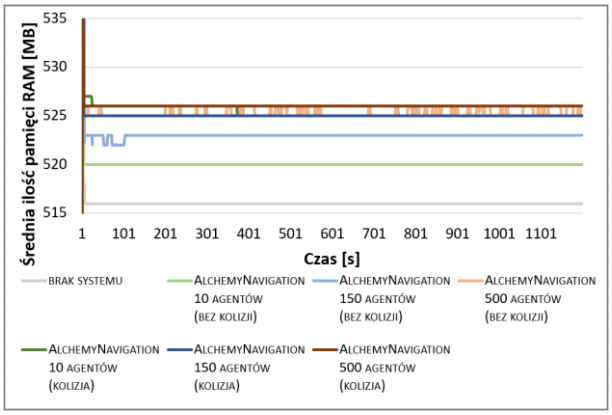
Rysunek 6: Średnie wartości czasu trwania klatki (ms) prób systemu NavMesh.



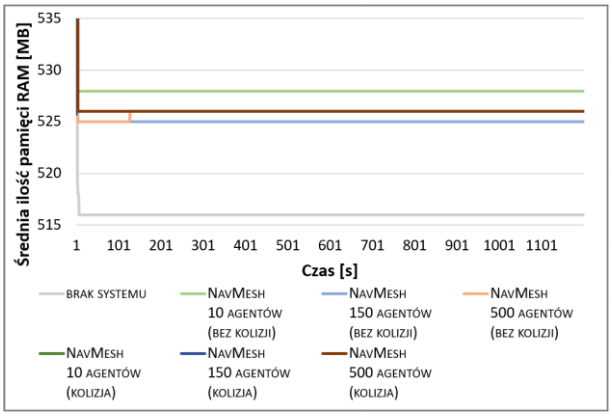
Rysunek 7: Porównanie uśrednionego wykorzystania pamięci RAM wszystkich stacji dla badanych systemów.

W przypadku obu badanych systemów, z rezultatów badań wynika, że nie ma zauważalnej relacji pomiędzy liczbą agentów, a wykorzystaniem pamięci RAM. Największa odnotowana różnica wyników wyniosła 1,57%. Jest to wynik na tyle niski, że można uznać różnice w wykorzystaniu pamięci przez badane systemy za nieistotną.

Rysunki 8 i 9 przedstawiają średnie wartości wykorzystania pamięci RAM wykonanych prób dla obu systemów.



Rysunek 8: Średnie wartości wykorzystania pamięci RAM dla prób systemu AlchemyNavigation.



Rysunek 9: Średnie wartości wykorzystania pamięci RAM dla prób systemu NavMesh.

W oparciu o przebiegi czasowe można dostrzec, że w przypadku obu badanych systemów wykorzystanie pamięci RAM jest bardzo stabilne - występujące zmiany nie przekraczają wartości jednego Megabajta. Należy również zauważyć, że w pierwszych sekundach użycie pamięci RAM zmienia się gwałtownie, może to wynikać z faktu, że w tym okresie inicjalizowane są przestrzenie nawigacyjne i wszyscy agenci żądają ścieżek w jednym momencie. Wartym spostrzeżenia jest fakt, że w przypadku prób systemu AlchemyNavigation z mniejszą liczbą agentów wartości stabilizują się na nieznacznie niższych poziomach (520 MB dla dziesięciu agentów i 523 MB dla stu pięćdziesięciu agentów). Natomiast, w przypadku systemu NavMesh wszystkie wartości stabilizują się przy podobnym zużyciu pamięci ~526 MB.

6. Analiza porównawcza cech badanych systemów

W tym rozdziale, badane systemy porównano pod kątem posiadanych cech w oparciu o wyniki badań i dokumentacje [12, 13]. Szczególną uwagę poświęcono

cechom znajdującym się na liście cech uniwersalnego systemu nawigacji (tabela 1).

6.1. Tworzenie predefiniowanych części przestrzeni nawigacyjnej

Dla systemu AlchemyNavigation tworzenie predefiniowanych części przestrzeni nawigacyjnej jest czynnością domyślną. Przestrzeń nawigacyjna jest budowana poprzez sekwencyjnie dodawanie trójkątów, które mogą być przechowywane w formie serializowanej (za pomocą komponentu FacesHolder lub innych typów zdefiniowanych przez użytkownika). Z kolei serializowane dane mogą być przechowywane na scenie lub w formach prefabrykatu (ang. prefab).

W przypadku systemu NavMesh funkcjonalność ta nie jest domyślnie dostępna, a cała przestrzeń zapisywana jest na scenie. Żeby korzystać z opcji tworzenia predefiniowanych części przestrzeni nawigacyjnej w formie prefabrykatów należy pobrać paczkę narzędzi wysokiego poziomu i użyć komponentu NavMesh Surface.

6.2. Tworzenie i modyfikowanie przestrzeni nawigacyjnej w czasie wykonania programu

System AlchemyNavigation pozwala na swobodne modyfikowanie przestrzeni nawigacyjnej w czasie wykonania programu, poprzez dodawanie i usuwanie należących do niej trójkątów. Operacje te wykonywane są z użyciem wątków, a potrzebne dane mogą być przechowywane w dowolnej serializowanej formie (np. plik lub komponent) lub generowane proceduralnie. Użytkownicy mogą również wykorzystać komponent FacesHolder w sposób opisany w punkcie 6.1.

W tej dziedzinie system NavMesh posiada mniejsze możliwości. Użytkownicy mogą wykorzystywać sposób opisany w punkcie 6.1 do dynamicznego przełączania aktywności wybranych elementów przestrzeni nawigacyjnej oraz dodatkowo rozmieszczać komponenty NavMesh Obstacle, które definiują kształt, w obrębie którego zdefiniowana przestrzeń zostanie wyłączona.

6.3. Podział przestrzeni nawigacyjnej na mniejsze elementy (ang. chunks)

Żaden z badanych systemów nie posiada dedykowanego narzędzia, które służy do dynamicznego przeładowywania fragmentów przestrzeni i zapewnia czyszczenie pamięci podręcznej. Aczkolwiek takie narzędzie może zostać łatwo opracowane w oparciu o sposób bazujący na wykorzystaniu prefabrykatów opisany w punkcie 6.1 (i innych form serializacji w przypadku systemu AlchemyNavigation) i systemu zasobów adresowalnych [15] (Addressables), który jest jednym z udostępnionych przez Unity pakietów.

6.4. Tworzenie agentów przemieszczających się w naturalny sposób po przestrzeni nawigacyjnej

Oba badane systemy posiadają możliwość tworzenia agentów, którzy symulują naturalny ruch w trakcie przemieszczania się po przestrzeni nawigacyjnej. W przypadku systemu AlchemyNavigation odpowiedzialny za tą funkcję jest komponent SimpleAgent, a w przypadku systemu NavMesh - NavMesh Agent.

6.5. Ustalanie parametrów ruchu dla różnych agentów

Oba badane systemy posiadają możliwość ingerowania w sposób, w jaki przemieszczają się agenci. System AlchemyNavigation posiada trzy możliwe scenariusze w ramach których takie zmiany są możliwe:

1. W oparciu o dziedziczenie po abstrakcyjnej klasie BasicAgent możliwe jest definiowanie nowych typów agentów, którzy w sposób całkowicie zależny od programisty przemieszczają się po obliczonych przez system ścieżkach.
2. Wykorzystując komponent SimpleAgent, który zapewnia naturalny ruch agentów i pozwala dostosować takie parametry jak: maksymalna prędkość, maksymalne przyspieszenie i prędkość rotacji.
3. Wykorzystując komponent SimpleAgent i podsystem MovementModifier, który pozwala definiować dodatkowe modyfikatory wpływające na sposób w jaki porusza się agent.

W przypadku systemu NavMesh, możliwe jest dostosowanie parametrów: prędkość, prędkość kątowa i przyspieszenie.

6.6. Przypisywanie wag decydujących o częstości wybierania określonych fragmentów przestrzeni nawigacyjnej w procesie poszukiwania ścieżki

Analizowane systemy nawigacji implementują bardzo podobne rozwiązania w kwestii przypisywania wag wybranym fragmentom przestrzeni nawigacyjnej. W obu przypadkach użytkownik może zdefiniować do 32 powierzchni (ang. areas). Powierzchnie posiadają wagę (AlchemyNavigation) lub koszt (NavMesh), które modyfikują częstość wyszukiwania prowadzących przez nie ścieżek. Oba systemy również pozwalają na definiowanie masek wskazujących na powierzchnie po których mogą przemieszczać się agenci.

6.7. Definiowanie alternatywnych powierzchni nawigacyjnych dla różnego typu agentów

Oba badane systemy posiadają opcje definiowania niezależnych przestrzeni nawigacyjnych, które można wykorzystać do budowania zaawansowanej logiki przemieszczania się agentów lub do celów optymalizacyjnych. W przypadku systemu AlchemyNavigation jest to podział na warstwy (ang. layers), a dla systemu NavMesh - typy agentów.

6.8. Unikanie kolizji

System AlchemyNavigation implementuje zachowanie unikania kolizji przy pomocy modyfikatora ruchu - AvoidanceModifier. Nawiązując do dokumentacji jest to bardzo prosta implementacja algorytmu, która jest odpowiednia tylko dla sytuacji, w których liczba agentów nie jest wysoka lub są oni odpowiednio podzieleni na grupy. Informację tą potwierdzają również wyniki badań, otóż średni wzrost czasu trwania klatki dla agentów wykorzystujących modyfikator względem tych

którzy go nie wykorzystywali wynosił: 0,23% dla dziesięciu, 31,80% dla stu pięćdziesięciu i 552,16% dla pięciuset agentów. Rekomendowanym alternatywnym sposobem, jest wykorzystanie klasy MovementModifier do przygotowania własnego modyfikatora dostosowanego do potrzeb użytkownika.

System NavMesh posiada wbudowany system unikania kolizji, który oferuje prace w pięciu trybach jakości: żaden (unikanie kolizji wyłączone), niski, średni, dobry i wysoki. Nawiązując do wykonanych badań średni wzrost czasu trwania klatki po przełączeniu trybu z "żaden" na "wysoki" wynosił odpowiednio: 0,98% dla dziesięciu, 7,98% dla stu pięćdziesięciu i 22,83% dla pięciuset agentów. Co oznacza że oferowany moduł jest znacznie wydajniejszy niż ten, który jest wykorzystywany przez system AlchemyNavigation.

6.9. Wsparcie dla dużej liczby (rzędu setek) agentów jednocześnie

Wsparcie dla dużej liczby agentów oznacza, że aplikacje wykorzystujące systemy nawigacji powinny działać ze stabilną liczbą klatek na sekundę [16], w tabeli 5 przedstawiono zestawienie standardowych progów liczby klatek na sekundę w grach.

Tabela 5: Standardowe progi liczby klatek na sekundę w grach komputerowych

Próg	Liczba klatek na sekundę	Przybliżony czas trwania klatki (ms)	Opis
Minimalny	30	33	Minimalna liczba klatek na sekundę uznawana za płynną w grach komputerowych
Standardowy	60	17	Liczba klatek na sekundę odpowiadająca najczęściej spotykanej częstotliwości odświeżania monitorów na rynku
Wysoki	144	7	Liczba klatek na sekundę odpowiadająca częstotliwości monitorów przeznaczonych dla graczy

Dla badanej aplikacji próg minimalny nie został osiągnięty jedynie w przypadku próby dotyczącej pięciuset agentów systemu AlchemyNavigation (z włączoną funkcją unikania kolizji). Wszystkie pozostałe próby stacji trzeciej osiągnęły poziom standardowy, a stacji jeden i dwa – wysoki.

6.10. Kontrolowanie przemieszczania się agentów pomiędzy oddzielonymi fragmentami przestrzeni nawigacyjnej

Przemieszczanie agentów pomiędzy oddzielonymi fragmentami przestrzeni nawigacyjnej w systemie AlchemyNavigation może zostać wykonane poprzez dynamiczne dodanie do przestrzeni trójkątów, które łączą wybrane fragmenty. Sposób ten jest naturalny i logiczny, ale może okazać się czasochłonny, ponieważ wymaga dodatkowego rozplanowania predefiniowanych połączeń lub przygotowania algorytmów odpowiedzialnych za ten proces.

Podejście prezentowane przez system jest prostsze - polega na wskazaniu dwóch punktów, pomiędzy którymi agenci będą się przemieszczać. Operacji tej dokonuje się za pomocą komponentu należącego do paczki narzędzi wysokiego poziomu - NavMesh Link.

6.11. Dostęp do kodu źródłowego, dokumentacji oraz forów społeczności

Kod źródłowy systemu AlchemyNavigation został udostępniony na warunkach licencji MIT [17], tym samym jest on dostępny do wglądu, modyfikacji i komercyjnego użycia. Dostępna jest również zawierająca przykłady i samouczki dokumentacja. System nie posiada jednak aktywnych forów, które mogły by posłużyć do wymiany wiedzy przez użytkowników. Taki układ może być preferowany przez średnio-zaawansowanych i zaawansowanych programistów, którzy bazując na swoim doświadczeniu są w stanie szybciej uzyskiwać odpowiedzi na pytania poprzez analizę dokumentacji i kodu.

Natomiast w przypadku systemu NavMesh dostępna jest bardzo szczegółowa dokumentacja i fora społeczności, a kod jest zamknięty. W tym wypadku grupą faworyzowaną są użytkownicy początkujący, którzy szukają gotowych rozwiązań lub samouczków.

7. Wnioski

Systemy nawigacyjne to złożone i wielopoziomowe narzędzia, które wciąż mogą być rozwijane ze względu na oferowane funkcjonalności oraz wykorzystywane algorytmy.

AlchemyNavigation i NavMesh stanowią przykład w pełni rozwiniętych systemów nawigacji, które oferują szeroką gamę funkcji i gotowych rozwiązań. W zakresie silnika Unity stanowią dobry punkt początkowy do implementacji sztucznej inteligencji agentów, których celem jest przemierzanie świata gry.

Po dokonanej analizie rezultatów i cech badanych systemów należy stwierdzić, że oba systemy nawigacji posiadają podobny zestaw funkcji i generują zbliżone obciążenie. Posiadają również cechy, dzięki którym można uznać je za rozwiązania uniwersalne.

System AlchemyNavigation znajduje lepsze zastosowanie w sytuacjach, kiedy potrzebna jest duża swoboda ingerencji w przestrzeń nawigacyjną w czasie wykonania programu. Mogą być to sytuacje, kiedy gracz posiada opcje budowania środowiska gry lub kiedy jest ono generowane proceduralnie. Ten wniosek motywowany jest faktem, że system opiera się na dynamicznym modyfikowaniu przestrzeni nawigacyjnej (z dokładnością do trójkątów). Natomiast dostarczane komponenty, za pomocą abstrakcji, ukrywają złożoność systemu i pozwalają na wykonywanie uproszczonych operacji.

Kolejną ważną cechą systemu AlchemyNavigation jest dostępny do wglądu i edycji kod źródłowy oraz domyślna możliwości tworzenia i zastępowania poszczególnych modułów (np. możliwość stworzenia własnego typu agentów i budowania modyfikatorów

agenta domyślnego). Takie właściwości w dużym stopniu pozwalają dostosować system do wymogów produktu końcowego, ale znajdą lepsze zastosowanie u lepiej doświadczonych użytkowników.

Ważnym odnotowanym dla systemu AlchemyNavigation wynikiem jest wysoki wzrost obciążenia zasobów stacji testowych, kiedy aktywnych jest wielu agentów oraz funkcja unikania kolizji. System udostępnia możliwości optymalizacji, takiej jak dzielenie agentów na grupy lub nadpisanie modułu kolizji, ale dla niedoświadczonych programistów takie podejście może stanowić poważną przeszkodę.

System NavMesh posiada nieznacznie mniejszą liczbę funkcji, które dodatkowo zostały podzielone na pakiet domyślny i rozszerzający. Takie podejście, wspierane przez szczegółową dokumentację oraz aktywne fora, skutkuje znacznie większą przystępnością narzędzia. Przez co system jest odpowiedni zarówno dla użytkowników doświadczonych, jak i początkujących.

Kluczową cechą systemu NavMesh jest również jego stabilność. Liczba klatek na sekundę jest wysoka, nawet kiedy aktywne są setki agentów, a różnica obciążenia generowane przez aktywność funkcji unikania kolizji nie jest duża.

Finalnie, przedstawione badania i analiza dowodzą tezy, że własne rozwiązania mogą dorównywać rozwiązaniom domyślnym, a także, że pozwalają wprowadzać nowe ciekawe funkcjonalności.

Literatura

[1] D. Silver, Cooperative Pathfinding, In Proceedings of the 1st Conference on Artificial Intelligence and Interactive Digital Entertainmen (2005) 117-122.

[2] D. Demyen, M. Buro, Efficient Triangulation-Based Pathfinding, In Proceedings of the 21st National Conference on Artificial Intelligence (2006) 942-947.

[3] U. Huh, S. Chang, A G² Continuous Path-smoothing Algorithm Using Modified Quadratic Polynomial Interpolation, International Journal of Advanced Robotic Systems 11 (2014) 224-234.

[4] P. Lester, A* Pathfinding for Beginners, https://www.gamedev.net/reference/articles/article2003.asp, [20.06.2021].

[5] Z. He, M. Shi, C. Li, Research and application of path-finding algorithm based on unity 3D, In Proceedings of the 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS) (2016) 1-4.

[6] S. Rabin, Game AI Pro 2: Collected Wisdom of Game AI Professionals, CRC Press, Boca Raton, 2015.

[7] S. Rabin, Game AI Pro 360: Guide to Tactics and Strategy, CRC Press, Boca Raton, 2019.

[8] J. Palacios, Unity 2018 Artificial Intelligence Cookbook: Over 90 recipes to build and customize AI entities for your games with Unity, 2nd Edition, Packt Publishing, Birmingham, 2018.

[9] A. Thorn, Geometric and Discrete Path Planning for Interactive Virtual Worlds, Morgan & Claypool Publishers, San Rafael, 2016.

[10] D. Aversa, A. S. Kyaw, C. Peters, Unity Artificial Intelligence Programming: Add powerful, believable, and fun AI entities in your game with the power of Unity 2018!, 4th Edition, Apress Publishing, Birmingham, 2018.

[11] R. Barrera, Unity 2017 Game AI Programming - Third Edition: Leverage the power of Artificial Intelligence to program smart entities for your games, Packt Publishing, Birmingham, 2018.

[12] Dokumentacja systemu AlchemyNavigation, https://kempnymaciej.github.io/alchemy-navigation/index.html, [20.06.2021].

[13] Dokumentacja systemu NavMesh, https://docs.unity3d.com/Manual/Navigation.html, [20.06.2021].

[14] Dokumentacja pakietu Unity.Profiling, https://docs.unity3d.com/ScriptReference/Unity.Profiling.ProfilerRecorder.html, [20.06.2021].

[15] Dokumentacja pakietu Unity.Addressables, https://docs.unity3d.com/Packages/com.unity.addressables@0.4/manual/index.html, [20.06.2021].

[16] F. Metzger, A. Rafetseder, C. Schwartz, T. Hoßfeld, Games and Frames: A Strange Tale of QoE Studies, In Proceedings of the International Conference on Quality of Multimedia Experience (2016) 1-2.

[17] Licencja MIT, https://choosealicense.com/licenses/mit/, [20.06.2021].