UNIWERSYTET EKONOMICZNY W KATOWICACH

KIERUNEK Informatyka

Sebastian Szewczyk

137520

Algorytm wyszukiwania drogi jako element sztucznej inteligencji w grze

The pathfinding algorithm as an element of artificial intelligence in the game

Praca licencjacka
napisana w Katedrze Informatyki
pod kierunkiem dr Artura Strzeleckiego

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem
i stwierdzam, że spełnia wymogi stawiane pracom dyplomowym

Pracę akceptuję

…… ……

(data) (podpis promotora)

KATOWICE ***rok 2021***

Sebastian Szewczyk
Informatyka
Numer albumu: 137520

Katowice, dnia 29.05.2021r.

OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że złożona praca licencjacka pt.: "Algorytm wyszukiwania drogi jako element sztucznej inteligencji w grze" została napisana przeze mnie samodzielnie.

Równocześnie oświadczam, że praca ta nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (tj. Dz. U. z 2018 r., poz. 1191, z późn. zm.) oraz dóbr osobistych chronionych prawem.

Ponadto praca nie zawiera informacji i danych uzyskanych w sposób niedozwolony i nie była wcześniej przedmiotem innych procedur związanych z uzyskaniem dyplomów lub tytułów zawodowych uczelni wyższej.

Wyrażam zgodę na nieodpłatne udostępnienie mojej pracy w celu oceny jej oryginalności przez Jednolity System Antyplagiatowy prowadzony przez Ministra Nauki i Szkolnictwa Wyższego oraz przechowywania jej w Ogólnopolskim Repozytorium Prac Dyplomowych oraz wewnętrznej bazie prac dyplomowych Uniwersytetu Ekonomicznego w Katowicach. Zostałem poinformowany o zasadach dotyczących oceny oryginalności pracy dyplomowej przez Jednolity System Antyplagiatowy.

Oświadczam także, że ostateczna wersja pracy przesłana przeze mnie drogą elektroniczną jest zgodna z plikiem poddanym ocenie w Jednolitym Systemie Antyplagiatowym.

Jednocześnie oświadczam, że jest mi znany przepis art. 233 § 1 Kodeksu karnego określający odpowiedzialność za składanie fałszywych zeznań.

Szewczyk Sebastian

.....

(podpis składającego oświadczenie)

SPIS TREŚCI

Wstęp

Sztuczna inteligencja od lat jest obiektem badań wielu naukowców, którzy w swoich laboratoriach usytuowanych na całym świecie próbują w jak największym stopniu zgłębić wiedzę na jej temat. Obecnie ciężko sobie wyobrazić życie bez wykorzystania wyników ich pracy, albowiem sztuczna inteligencja w mniejszym lub większym stopniu jest wykorzystywana w każdej dziedzinie naszego życia. Istnieją branże, których funkcjonowanie nie jest możliwe bez zastosowania algorytmów AI. Jedną z nich jest branża gier komputerowych. Współcześnie twórcy gier wykorzystują wiedzę w tym zakresie w wielu różnych aspektach rozgrywki.

Sztuczna inteligencja pełni wiele, często bardzo zróżnicowanych funkcji, bez których stworzenie w pełni dopracowanej gry nie byłoby możliwe. Niniejsza praca opisuje te funkcje, jak i metody zastosowane w tym celu, jednak głównie skupia się na jednym zadaniu, do wykonania którego używana jest wcześniej wymieniona dziedzina nauki. Zadaniem tym jest wyszukiwanie drogi w grach, a wykonanie go jest możliwe dzięki zastosowaniu algorytmów do tego przeznaczonych.

Celem pracy jest przybliżenie tematu sztucznej inteligencji w grach, a przede wszystkim algorytmów, które są odpowiedzialne za wyszukiwanie trasy. W tym celu został także opracowany algorytm wyszukiwania drogi, który pozwala na jeszcze dokładniejsze zbadanie tematu pracy. Algorytm jest rozbudowany o optymalizację wielokryterialną, która sprawia, że jest on zdecydowanie bardziej wszechstronny.

Pierwszy rozdział jest teoretycznym wprowadzeniem w świat gier komputerowych, używających algorytmów sztucznej inteligencji w przeróżnych, czasem nieoczywistych celach. Cele te zostały ściśle określone oraz opisane. Następnym zagadnieniem poruszonym w pierwszym rozdziale jest różnorodność metod wykorzystywanych w procesie tworzenia algorytmów sztucznej inteligencji. Chcąc jak najlepiej przybliżyć działanie tych algorytmów, przedstawione zostały gry, które z wielkim sukcesem wprowadzały kolejne coraz bardziej zaawansowane algorytmy oraz wyznaczały trendy w branży gier komputerowych.

Drugi rozdział jest już mocno skoncentrowany na algorytmach wyszukiwania drogi. Pierwsza część rozdziału jest poświęcona sposobom podziału całego środowiska gry na mniejsze kawałki w celu ułatwienia nawigacji. Istnieją sposoby wyznaczania trasy ujęte w poszczególnych algorytmach, które zostały już wcześniej opracowane. Scharakteryzowanie tych metod stanowi kolejną część drugiego rozdziału. Przedstawione zostały także przykładowe zastosowania algorytmów wyszukiwania drogi w różnych grach.

Trzeci rozdział zawiera opis stworzonej na potrzeby pracy gry, zadaniem której jest zobrazowanie działania algorytmu wyszukiwania trasy. Dokładnie opisany został każdy etap tworzenia algorytmu wraz z zaprezentowaniem kodu odpowiedzialnego za daną funkcjonalność. Zaprezentowane zostało w jaki sposób algorytm wyszukuje najkrótszą trasę pomiędzy punktami, w jaki sposób wdrożona została optymalizacja wielokryterialna oraz w jakim celu została ona zastosowana.

1. Sztuczna inteligencja w grach

Pierwszy rozdział stanowi ogólne wprowadzenie w tematykę sztucznej inteligencji w grach komputerowych. Znajdziemy w nim informacje pozwalające nam zrozumieć w jakim celu wykorzystuje się sztuczną inteligencję w grach oraz jak ważną rolę pełni w tych produkcjach. W dalszej części rozdziału przedstawione zostały poszczególne metody sztucznej inteligencji wraz z ich opisem. W ostatniej części pierwszego rozdziału umieszczonych zostało kilka przykładów gier wykorzystujących sztuczną inteligencję, które na przestrzeni lat były przełomowe w procesie rozwoju tej dziedziny.

Sztuczna inteligencja jest bardzo rozbudowaną dziedziną nauki. Badania w jej zakresie, zależnie od czynnika, dążą do zrealizowania dwóch podstawowych celów (Wawrzyński, 2014):

- poznania zasad działania ludzkiego umysłu i próby odwzorowania go - czynnik psychologiczno-filozoficzny;
- stworzenia systemów, które będą wykonywać zadania, w których człowiek wykorzystuje inteligencję - czynnik inżynierski.

1.1. Rola sztucznej inteligencji w grach

Sztuczna inteligencja od dawna jest nieodłącznym elementem gier komputerowych. Początkowo algorytmy były proste i ich działanie nie do końca odzwierciedlało inteligentne zachowania. Jednak wraz z rozwojem technologii i samej sztucznej inteligencji, stawały się one coraz skuteczniejsze i wydajniejsze. Wraz z upływem czasu, wpływ sztucznej inteligencji na branże gier wideo stale rośnie. Gracze są coraz bardziej wymagający i coraz trudniej sprostać ich oczekiwaniom.

Twórcy gier wielką wagę przywiązują do stworzenia jak najlepszego świata gry. Czym jednak byłby świat gry bez zastosowania sztucznej inteligencji? Celem algorytmów sztucznej inteligencji jest jak najdokładniejsze odwzorowanie i zobrazowanie rzeczywistości i jej realiów (Kuźmińska-Sołśnia i Siwiec, 2015). Sterowanie postaciami przez sztuczną inteligencję, należy do jednych z najważniejszych jej zadań. Bardzo duży nacisk nakładany jest na to, aby osoba grająca czuła się tak, jakby grała z drugim człowiekiem a nie z komputerem (Argasiński, 2012). W tym celu, reakcje oraz zachowania agentów przypominają w jak największym stopniu zachowania ludzkie. W sytuacji, gdy gracz napotyka postacie sterowane przez komputer, powinien on spodziewać się zachowania inteligentnego, reagującego na poczynania gracza, a nie postaci, która bezmyślnie wykonuje

z góry założoną czynność. To daje poczucie realności świata. Ponadto algorytmy nie muszą ograniczać się tylko do reakcji na zachowania gracza. Istnieją metody, które pozwalają na osiągnięcie bardziej zadowalających efektów takich jak planowanie kolejnych działań w oparciu o informacje na temat gracza pozyskane w trakcie rozgrywki (Argasiński, 2012). Poprzez zbieranie informacji na temat poszczególnych zachowań i preferencji gracza oraz późniejszej ich analizie, komputer jest w stanie z zadowalającą skutecznością przewidywać kolejne decyzje oraz działania. Dzięki temu poziom trudności gry staje się wyższy, a cała rozgrywka staję się zdecydowanie ciekawsza.

W celu lepszego odwzorowania ludzkiej inteligencji mogą zostać użyte znaczniki, które określają w jaki sposób komputer ma zareagować na różne sytuacje. Częstym znacznikiem stosowanym w grach jest poziom zdrowia postaci (Schreiner, 2009). Dla poszczególnych wartości zdrowia określane są poszczególne zachowania. Na przykład jeśli poziom zdrowia spadnie poniżej danej wartości odpowiednim zachowaniem w celu przetrwania będzie ucieczka i unikanie gracza lub ewentualnie, jeśli istnieje możliwość, ukrycie i uleczenie się przed ponownym podjęciem walki z graczem. Kolejnym często używanym znacznikiem jest ilość amunicji w magazynku (Schreiner, 2009). Gdy postać zorientuje się, że brakuje mu kul w magazynku, znajdzie najbliższą osłonę i schowa się za nią, aby być bezpiecznym w trakcie przeładowywania.

Istnieje wiele gier, zwłaszcza internetowych, w których celowo ogranicza się interakcję z innymi graczami. Zgodnie z koncepcją świata przedstawionego w grze, postacie niezależne sterowane przez AI mogą w sposób aktywny i równorzędny współpracować z użytkownikiem. Ten przykład pokazuje jak bardzo podejście do sztucznej inteligencji w grach różni się od podejścia do czysto naukowego aspektu tego tematu. Głównym celem badań nad szeroko rozumianą sztuczną inteligencją jest jak najlepsze i najdokładniejsze odwzorowanie i symulowanie ludzkiej inteligencji, jednak w grach cel ten jest trochę inny. Użytkownik podczas rozgrywki przede wszystkim powinien mieć wrażenie, że postać sterowana przez komputer, wchodząca z nim w jakąkolwiek interakcję, zachowuje się inteligentnie (Argasiński, 2012).

Kierując się tym podejściem twórcy gier często decydują się na stosowanie różnych sztuczek. Kolejnym powodem stosowania różnych tricków jest możliwość oszczędności zasobów sprzętowych. Przykładem sztuczek stosowanych w grach jest podejmowanie decyzji w pewnych aspektach gry przez sztuczną inteligencję nie na podstawie dokładnej analizy wszystkich danych lecz w sposób mniej lub bardziej, a w niektórych przypadkach nawet

w pełni losowy (Argasiński, 2012). Oznacza to, że komputer podejmuje decyzje w sposób losowy lub analizuje tylko stopień prawdopodobieństwa, który wyznacza jaka istnieje szansa, że użytkownik znajdujący się w tej sytuacji zachowałby się w ten a nie inny sposób. Nie można jednak używać zbyt wiele takich sztuczek. Gracz powinien cały czas mieć wrażenie, że każda podjęta decyzja przez sztuczną inteligencję jest przemyślana oraz w jak najdoskonalszy sposób naśladuje potencjalne zachowanie użytkownika.

System podejmowania decyzji odpowiedzialny jest za definiowanie wszystkich zachowań postaci niezależnych oraz wszystkich czynności przez nich wykonywanych (Millington i Funge, 2009). Zakres działań rozpatrywanych podczas podejmowania decyzji jest zależny od postaci, dla której czynność ta ma zostać wybrana oraz od sytuacji w jakiej aktualnie się znajduje. Oprogramowanie odpowiedzialne za decyzje w sposobie swojego działania uwzględnia wybrany schemat pozyskiwania a później przetwarzania informacji (Argasiński, 2012). Działanie takiego oprogramowania jest stosunkowo łatwe do zrozumienia. Program pobiera dane wejściowe, które są od razu przetwarzane. W wyniku przetworzenia danych wejściowych otrzymujemy dane wyjściowe, na podstawie których podejmowane są decyzje odnośnie danego aspektu gry. Regulacja działania sztucznej inteligencji nie polega na wybraniu i zastosowaniu jednej z przygotowanych wcześniej możliwości. Jest to proces bardziej rozbudowany, który wykorzystuje mniej lub bardziej złożony model. Wybrany model wykorzystuje dane wyjściowe dostarczone przez oprogramowanie do jak najlepszego odwzorowania zachowania inteligentnego gracza.

Za realizację scenariusza określonego w grze odpowiadają algorytmy planowania. Planowanie odpowiedzialne jest za badanie oraz wdrażanie metod, które wykorzystywane są do opracowywania planów prowadzących do osiągnięcia określonych celów (Duarte i in., 2020). Jednak nie jest to jedyny sposób w jaki można wytłumaczyć działania tych algorytmów. Przewidywanie poszczególnych zachowań gracza oraz wszystkich przyszłych interakcji ze środowiskiem także znajduje się w zakresie działania planowania. Celem takiej symulacji jest poznanie długoterminowych konsekwencji poszczególnych działań (Duarte i in., 2020). W celu poprawnego działania oraz skutecznej realizacji scenariusza, każde zachowanie gracza oraz wszystkich autonomicznych jednostek musi być rozpatrywane w kontekście całej rozgrywki.

Do kierowania działaniami postaci niezależnych często stosuję się techniki znane już wcześniej. Przykładem może być zastosowanie drzew decyzyjnych. Jednak istnieją

zastosowania sztucznej inteligencji, które nie są od razu widoczne dla gracza. Do zastosowań tych należy eksploracja danych oraz generowanie treści proceduralnych (Yannakakis, 2012)

Kolejnym bardzo ważnym elementem gry definiowanym przez sztuczną inteligencję jest sterowanie ruchem postaci niezależnych. Podstawą działania algorytmów sterujących ruchem jest przekształcenie podjętej decyzji w jakiś rodzaj ruchu i wykonanie go (Millington i Funge, 2009). W sytuacji, gdy postać znajduje się w sporej odległości od gracza oraz nie posiada żadnej broni pozwalającej na atak z dystansu, algorytm podejmowania decyzji decyduje, że niezbędne jest zbliżenie się do gracza. Ta informacja dociera do algorytmu odpowiedzialnego za sterowanie ruchem, który zaczyna swoją pracę. Określane jest dokładne położenie gracza i trasa, po której postać będzie się przemieszczać do celu.

Interakcja w grze występuje niemalże przez cały czas trwania rozgrywki. Sterowanie nią jest zatem bardzo rozbudowanym elementem sztucznej inteligencji w grze. Istnieją dwa typy interakcji, w których gracz spotyka się z zastosowaniem sztucznego intelektu (Argasiński, 2012). Pierwszym z nich jest interakcja z postaciami niezależnymi i obejmuje wszystko co z nimi związane np.: rozmowę, walkę, współpracę. Drugim rodzajem jest interakcja gracza z otaczającym środowiskiem gry, w które wliczają się wszystkie elementy otoczenia reagujące w jakikolwiek sposób na naszą obecność i poczynania. Interfejsy, które pomagają nam w trakcie rozgrywki np. wyświetlając nam tzw. inteligentne podpowiedzi także zaliczają się do środowiska gry.

We współczesnych grach wideo sztuczna inteligencja jest wykorzystywana także do zdolności polowania. Metoda polowania jest bardzo efektywna i często określana jako bardzo pozytywny element rozgrywki. Początkowo reakcje sztucznej inteligencji były bardzo skrajne. Zależnie od sytuacji lub położenia wroga AI przyjmowała postawę w pełni ofensywną lub defensywną (Schreiner, 2009). Pojęcie polowania w ostatnich latach zostało bardzo mocno rozwinięte. Do śledzenia oraz polowania na gracza, sztuczna inteligencja używa coraz więcej realistycznych znaczników oraz stosownych czujników, takich jak czujnik audio pozwalający usłyszeć różne hałasy wytwarzane przez gracza lub też czujnik wizualny, który pozwala dostrzec naoczne zmiany w otoczeniu (E Shummon Maass, 2019). Zmiany wprowadzone w zdolności polowania poskutkowały wzrostem poziomu rozgrywki oraz zmuszają gracza do większego wysiłku. Dzięki temu gra staje się bardziej atrakcyjna i ciekawa. Gracz musi przeanalizować więcej czynników oraz uważać na każdy swój ruch by

nie zostać wytropionym przez wroga. Dokładniej musi zaplanować atak, gdy jest on konieczny lub uniknąć przeciwnika w bezpieczny sposób.

Instynkt przetrwania jest kolejnym elementem sztucznej inteligencji w grach, który w ostatnim czasie coraz częściej pojawia się w różnych produkcjach (Schreiner, 2009). Otoczenie, w którym znajduje się postać sterowana przez komputer jest sprawdzana w bardzo dokładny sposób. Następnie każdy obiekt zostaje zbadany i oceniania jest jego użyteczność. Komputer w ten sposób określa, które obiekty mogą pomóc mu w przetrwaniu, a które są bezużyteczne. Istnieją sytuacje, w których postać jest zmuszona do wykonania czynności, która chwilowo czyni ją bezbronną. W takiej sytuacji wykorzystywane zostają wszelkie osłony, które znajdują się na polu walki.

Walki w trakcie rozgrywki pojawiają się w wielu gatunkach gier. Istnieją takie gry, w których system walk jest najważniejszym elementem rozgrywki. Definiuje on atrakcyjność tych gier oraz decyduje o ich popularności wśród graczy. Algorytmy stosowane do obsługi systemu walk mają za zadanie sterować agentami podczas walki. Przy wyborze techniki lub akcji, którą w danej chwili ma wykonać agent, biorą pod uwagę wiele czynników, starając się w jak najlepszy sposób symulować ludzką logikę (Kuźmińska-Sołśnia i Siwiec, 2015).

Do zadań sztucznej inteligencji w poszczególnych grach komputerowych należy także komentowanie dokonań gracza oraz wydarzeń, które mają miejsce w trakcie rozgrywki (Kuźmińska-Sołśnia i Siwiec, 2015). Gry sportowe są dobrym przykładem takich gier. Sztuczna inteligencja dokonuje analizy zachowań i dokonań gracza, po czym zajmuje się komentowaniem rozwijającej się sytuacji.

W komercyjnych grach wideo wykorzystuje się inny rodzaj sztucznej inteligencji niż w grach, w których rozgrywka odbywa się na nadludzkim poziomie. Istnieje wiele aplikacji przeznaczonych do gry w klasyczne gry planszowe, które z łatwością są w stanie wygrać z człowiekiem. Przykładem jest między innymi system DeepBlue firmy IBM, który w 1997 roku pokonał rosyjskiego arcymistrza Garry'ego Kasparova (Statt, 2019).

Historia sztucznej inteligencji jest ściśle związana z historią gier. Stało się tak ze względu na fakt, iż prekursorzy badań nad sztucznym intelektem uważali, że gry takie jak szachy są doskonałym środowiskiem do testów i prac badawczych. Traktowali je jako wyznacznik inteligencji i uważali, że nauka komputera tej właśnie gry będzie początkiem większego sukcesu tych badań. Obecnie osoby zajmujące się takimi badaniami uznali, że lepiej będzie skoncentrować się na innych, bardziej skomplikowanych grach. Dużą część ich uwagi przykuwa chińska gra Go (Gault, 2020).

W placówkach zajmujących się badaniami nad sztuczną inteligencją na całym świecie, takich jak laboratorium DeepMind należące do Google lub w Facebookowym dziale badań nad sztuczną inteligencją trwają pracę nad rozwojem oprogramowania przeznaczonego do gry w coraz bardziej zaawansowane gry. Zakres tych badań jest bardzo szeroki i nie obejmuje tylko nowych, wielkich produkcji takich jak Dota 2 od Valve Corporation lecz wszystkie gry, które mogą być dobrym środowiskiem do rozwijania nowych technologii. Przykładami są klasyczne gry Atari a także wcześniej wspomniana gra Go. W badaniach tego typu celem nie jest rozwój pod kątem tworzenia bardziej dynamicznych, interesujących czy też realistycznych gier. Najważniejsza jest możliwość porównania poziomu zaawansowania poszczególnych oprogramowań (Statt, 2019). Skrupulatny system reguł i nagród w wirtualnych światach czyni je bardzo użytecznym środowiskiem do testowania i szkolenia oprogramowania. Z tego powodu światy te bardzo często są wykorzystywane przez badaczy AI. Naukowcy próbują dowiedzieć się jak nauczyć oprogramowanie coraz to bardziej skomplikowanych zadań. W tym celu uczą oprogramowanie grać w gry, z nadzieją, że kiedyś uda im się wykorzystać oprogramowanie do większych celów (Statt, 2019).

Nauka maszynowa jest kolejnym aspektem sztucznej inteligencji, gdzie gry są wykorzystywane jako bardzo dobry poligon do badań. Możliwość dokładnej obserwacji wszystkich elementów gry jest tutaj najważniejszym determinantem, który sprawia, że gry są atrakcyjnym środowiskiem do testowania (Gault, 2020). Jest to także przewaga nad światem rzeczywistym, w którym istnieje wiele aspektów niemożliwych do zbadania w aż tak dokładny sposób. W procesie nauki sieci neuronowych bardzo ważną rolę spełnia system nagród i kar (Pluta, 2020). Określenie czy w trakcie rozgrywki idzie nam dobrze czy jednak źle jest kluczowe. W grach nie jest to trudne, lecz w realnym życiu jest to skomplikowane i niejednoznaczne. Gdy ponosimy szkody lub tracimy życie oczywistym jest, że nie idzie nam dobrze. W przypadku gdy zdobywamy punkty, przechodzimy kolejne poziomy, wzbogacamy się lub pokonujemy kolejnych wrogów mamy pewność, że idzie nam dobrze. W procesie uczenia maszynowego, takie informacje zwrotne dostarczane do systemu są podstawą poprawnego działania algorytmów uczących maszyny (Gault, 2020).

Gracze od wielu lat zmagają się z przeciwnikami sterowanymi przez komputer. Przykładem może być najzwyklejsza rakieta do ping-ponga, która jak najskuteczniej próbuje uniemożliwić nam zdobycie punktu odbijając piłeczkę w naszą stronę. Bowser z bardzo popularnej gry Mario, także próbuje utrudnić nam wykonanie zadania, jakim jest uratowanie księżniczki. Jednak aby zadowolić coraz to bardziej wymagających fanów gier

komputerowych, naukowcy starają się wymyślać coraz to nowe algorytmy oraz rozwiązania. Jednym z ciekawych zagadnień jest konstruowanie algorytmów oraz technologii, które pozwalają graczom tworzyć swoje własne gry. Programy do tworzenia gier stale są rozwijane i wykorzystywane są w nich coraz bardziej zaawansowane mechanizmy pozwalające wszystkim użytkownikom na tworzenie coraz lepszych i bardziej dopracowanych gier (Kobiałka, 2019). W ten sposób każda osoba zainteresowana zagadnieniem gier może spróbować swoich sił i podjąć próbę stworzenia ekscytującej gry na własną rękę.

1.2. Metody sztucznej inteligencji

Algorytmy sztucznej inteligencji oraz algorytmy heurystyczne wewnątrz gry są używane w wielu aspektach rozgrywki i wykonują dużo różnych, zróżnicowanych zadań. Stosowanych jest wiele metod sztucznej inteligencji i są one dobierane na podstawie roli, jaką mają spełnić w grze. Jeden algorytm będzie dużo lepszy do wykonania danego zadania niż inny.

Głównym celem systemów ekspertowych jest rozwiązywanie problemów, w których potrzebna jest wiedza eksperta (Rutkowski, 2006). Działanie takiego systemu nie jest zbyt skomplikowane. Systemy te posiadają swoją bazę wiedzy, w której znajdują się informacje z zakresu danej dziedziny. Elementami bazy wiedzy są także reguły, które określają pewne następstwa. Zdarza się także, że nowe fakty są tworzone poprzez reguły. Program po otrzymaniu pytania od użytkownika, zaczyna proces wnioskowania. Występują różne techniki wnioskowania. W wyniku tych działań, użytkownik dostaje odpowiedź na zadanie pytanie.

Sposobem stosunkowo łatwym w implementacji i zrozumieniu działania jest wykorzystanie drzew decyzyjnych. Ta metoda najlepiej sprawdza się w sytuacjach, gdy nie mamy wielkiej ilości dostępnych zachowań i stanów (Marecki, 2001). Obiekt złożony z różnych parametrów, na początku jest przekazywany do drzewa decyzyjnego i uznawany za dane wejściowe. Wszystkie węzły znajdujące się w drzewie decyzyjnym są powiązane z parametrami. Jeden parametr dotyczy jednego węzła. Od parametru rozchodzą się gałęzie. Są one wartościami, które parametr może osiągnąć. Następnie na podstawie osiągniętej wartości parametru, zwrócona zostaje wartość logiczna, znajdująca się w liściu, na danej gałęzi. Drzewa decyzyjne są bardzo popularną metodą wykorzystywaną w grach. Można je zastosować do wszystkich postaci niezależnych znajdujących się w wirtualnym świecie. Jednak z zastosowaniem tej metody trzeba uważać. W trakcie projektowania takiego drzewa często zdarza się że deweloper pominie jakiś szczegół lub nie uwzględni jakiejś ewentualności. Skutkiem takiego niedopatrzenia są sytuacje, w których postać zachowuje się

w sposób nieplanowany przez programistów. Częstym problemem jest wielokrotne wykonywanie tej samej czynności lub wykonywanie czynności, która z punktu widzenia logiki nie ma sensu (Lara-Cabrera i in., 2015).

Zastosowanie sieci neuronowych, jest kolejnym sposobem implementacji sztucznej inteligencji w grach. Sieci neuronowe swoją nazwę zawdzięczają sposobowi swojego działania, oparty na wykorzystaniu układów nerwowych. Modele biologiczne wyżej wymienionych układów są wykorzystywane do projektowania sieci neuronowych (Drabik, 2018).

Istnieją różne sposoby łączenia neuronów w sieci i są one podstawą do określenia ich rodzaju. Drugim kryterium wykorzystywanym do oceny rodzaju jest kierunek, w jakim przepływa sygnał (Sanocki i Gołda, 2005). Metody wykorzystywane do uczenia są zależne od typu sieci. Spośród wszystkich typów sieci, których jest bardzo wiele, można wyodrębnić kilka podstawowych. Należą do nich (Sanocki i Gołda, 2005):

- sieci jednokierunkowe, które mogą być jednowarstwowe lub wielowarstwowe;
- sieci rekurencyjne;
- sieci komórkowe.

W sieciach jednokierunkowych przepływ danych odbywa się tylko w jednym kierunku. Pierwszym elementem sieci jest warstwa wejściowa. W przypadku gdy mamy do czynienia z siecią jednowarstwową, kolejnym a zarazem ostatnim elementem jest warstwa wyjściowa. W przypadku sieci wielowarstwowej, pomiędzy dwoma wymienionymi wcześniej warstwami znajduje się jeszcze co najmniej jedna warstwa ukryta (Dudzic, 2009).

Sieci rekurencyjne charakteryzują się występowaniem sprzężenia zwrotnego. Każda sieć tego typu posiada co najmniej jedno takie sprzężenie (Sanocki i Gołda, 2005). Specyfika działania polega na przenoszeniu sygnałów wyjściowych na wejście. Skutkiem takiego działania jest zależność sygnałów wejściowych nie tylko od stanu tego wejścia w danym momencie, lecz także od sygnałów wyjściowych wychodzących z cyklu poprzedzającego.

W sieciach komórkowych połączenia mogą występować wyłącznie pomiędzy najbliższymi elementami przetwarzającymi. Do opisania tych sprzężeń służy układ równań różniczkowych. Utworzenie metody projektowania, która będzie odpowiednio efektywna i skuteczna, jest największą trudnością i przeszkodą w stosowaniu tych sieci (Sanocki i Gołda, 2005).

Do metod sztucznej inteligencji wykorzystywanej w grach, możemy także zaliczyć logikę pierwszego rzędu, która jest rozszerzeniem logiki zdań (Marecki, 2001). Jest bardziej

rozwinięta oraz efektywniejsza od swojego poprzednika. Podstawowym założeniem tej metody jest obiektowe podejście do świata. Oznacza to, że elementami otaczającego świata są obiekty. Każdy obiekt posiada własności, indywidualnie je opisujące, służące do ich rozróżnienia. Bardzo ważnym aspektem jest fakt, iż w ten sposób, można opisać zdecydowaną większość rzeczywistości. Podobnie jest ze zdaniami, gdyż zdecydowaną większość zdań można przedstawić wykorzystując metodę logiki pierwszego rzędu. Kategorie obiektów nie są w żaden sposób faworyzowane, a funkcje nie są sprawdzane pod kątem przynależności do poszczególnych obiektów. W odróżnieniu od logiki zdań, w której zdanie pełniło funkcję wyrażenia logicznego, w logice pierwszego rzędu tę funkcje pełnią termy, w których skład wchodzą stałe, zmienne i symbole funkcyjne. Termy odwołują się do obiektów. Predykaty i kwantyfikatory operują na termach i są wykorzystywane do budowy zdań (Marecki, 2001).

W programach komputerowych najczęściej wykorzystywane i przetwarzane są dane lub wartości ściśle określone. Jednak człowiek nie zawsze posługuję się informacjami, które są precyzyjne lub pewne. Działania ludzi na podstawie niepewnych informacji, zainteresowały osoby zajmujące się sztuczną inteligencją. W przypadku gdy w regule występują informacje niepewne, może się zdarzyć sytuacja, w której stwierdzenia są prawdziwe, lecz nie prowadzą one do poprawnej konkluzji. Dzieję się to poprzez doprecyzowanie argumentów. Logika rozmyta jest działem, który zajmuje się właśnie tą kwestią. Elementami logiki rozmytej są (Wawrzyński, 2014):

- zbiory rozmyte, w których na podstawie cech obiektów określa się przynależność do zbiorów. Przynależność może być całkowita, częściowa lub może nie występować;
- rozmyte spójniki logiczne, które mają na celu wykonanie rozmytego, całościowego opisu obiektów;
- rozmyte reguły określają zbiory, których zadaniem jest opisywanie atrybutów sugerując się wartościami innych obiektów;
- bloki wyostrzające zajmują się konkretyzacją rezultatu, który otrzymuje się wskutek rozmytego wnioskowania.

Maszyny stanów skończonych cieszą się dużą popularnością wśród twórców gier komputerowych. Są one bardzo często wykorzystywane do implementacji sztucznej inteligencji. Metoda oparta na automatach skończonych jest stosunkowo łatwa. Maszyna posiada stany oraz przejścia między nimi. Automat w danej chwili nie może przybierać więcej niż jednego stanu. Jest to jedna z podstawowych zasad działania tej metody

(Winiarski, 2018). Określenie warunków, po zrealizowaniu których maszyna przechodzi ze stanu, w którym aktualnie się znajduję, w kolejny stan, jest niezbędne do poprawnego działania. Funkcje mogą być wykorzystywane do sprawdzania warunków, co daje możliwość sprawdzania nie tylko pojedynczych faktów, lecz także całego szeregu warunków. Bardzo ważny jest fakt, że nie może istnieć nieskończona liczba stanów, pomiędzy którymi porusza się maszyna. Działanie maszyny jest intuicyjne i jego zrozumienie nie powinno być problematyczne. Po wprowadzeniu danych wejściowych, następuje ich analiza, na podstawie której określany jest stan, do którego następuje przejście, nazywany stanem wyjściowym. Jeden obiekt nie jest ograniczony co do ilości maszyny stanów, które są odpowiedzialne za sterowanie nim. Dodatkowo warto zaznaczyć, że stany mogą występować w kilku automatach jednocześnie (Winiarski, 2018).

Zastosowaniem dla sztucznej inteligencji najczęściej spotykanym w grach strategicznych czasu rzeczywistego lecz mających także zastosowanie w wielu innych gatunkach gier jest pathfinding. Jest to metoda ściśle związana z przemieszczaniem się w środowisku gry. Algorytmy wyszukiwania drogi mają za zadanie znajdowanie trasy, przy wykorzystaniu której postacie niezależne będą w stanie przedostać się z jednego punktu do drugiego (Verma i in., 2015). Przy wyznaczaniu trasy trzeba uwzględnić wiele czynników takich jak teren po którym się poruszamy, przeszkody znajdujące się na mapie, w niektórych przypadkach "mgłę wojny", która jest niczym innym jak zasłoniętym obszarem na mapie znajdującym się w danym momencie poza zasięgiem naszej postaci (Hagelbäck i Johansson, 2008).

W trakcie analizy dużej ilości danych lub gdy te dane są zróżnicowane tematycznie stosuje się bardzo zaawansowane metody sztucznej inteligencji. Podobnie jest z projektowaniem i tworzeniem inteligentnych systemów informatycznych nowej generacji. Techniki wykorzystywane w tych celach mają bardzo zróżnicowany obszar i zakres działań. Jednak przez lata metody te nie stały się popularne i nie są powszechnie stosowane. Jednym z powodów tej sytuacji jest fakt, iż często projekty badawcze zajmujące się tak zaawansowanymi technikami są po prostu zbyt kosztowne w stosunku do potencjalnych zysków czy to handlowych czy też praktycznych. Studia tworzące gry też nie są bez winy w tej sytuacji. Niechętnie podejmują ryzyko przy wyborze stosowanej technologii, nawet jeśli jest ona bardzo obiecująca. W większości przypadków pozostają przy sprawdzonych w pełni skryptowych technikach.

Sektor gier poważnych najbardziej cierpi ze względu na często zawodzący system waloryzacji wiedzy (Stewart i in., 2013). Wbrew pozorom gry poważne występują w wielu zróżnicowanych dziedzinach nie tylko nauki ale także życia. Są szeroko wykorzystywane w medycynie, sztuce, pomagają zwiedzać, poznawać świat. Problemem, który najbardziej rzutuje na taki a nie inny stan gier poważnych jest brak wielkich studiów tworzących takie gry. Studia zajmujące się tego typu grami są małe i niezależne przez co mają problemy z dostępem do różnych zasobów. Ciężko jest im o dostęp do nowych technologii a także do nowinek bardzo szybko rozwijającej się wiedzy badawczej. Przez takie ograniczenia ich projekty wykorzystują stare, znane już techniki i ciężko o jakikolwiek rozwój technologiczny (Stewart i in., 2013).

Współpraca między przemysłem gier a sektorem zajmującym się badaniami nad różnego rodzaju grami jest bardzo zróżnicowana i przynosi wiele korzyści. W wyniku tych badań powstało wiele ciekawych i użytecznych narzędzi do tworzenia gier i ulepszania całego sektora gier komputerowych. Jednym z takich narzędzi wykorzystywanych w procesie tworzenia gier jest FAtiMA. Jest to zbiór narzędzi open-source'owych odpowiedzialny za tworzenie obliczeniowych modeli emocji a także za podejmowanie decyzji (Dias i in., 2014). Twórcy gier oraz robotów często korzystają z tego zestawu w swoich projektach. Częstym zastosowaniem jest tworzenie postaci Role Play Characters (Dias i in., 2014). Pierwsza zaletą a zarazem ułatwieniem dla użytkowników jest fakt, iż w celu skorzystania z zestawu nie trzeba niczego instalować, ponieważ FAtiMA działa jako biblioteka C#. Jednak głównym powodem tego rozwiązania jest ułatwienie integracji z silnikami gier. Komponenty wchodzące w skład zestawu są w stanie załadować a także zapisać swój stan wewnętrzny do pliku JSON. To z kolei pozwala na dalszą obróbkę przez grę. Jeśli chcemy cokolwiek napisać, aby użyć tego w zestawie to możemy w tym celu użyć dowolnego edytora tekstowego. Jednak komponenty, z których składa się postać role play posiadają specjalne, dedykowane edytory, które są odpowiednio do nich przystosowane. Dedykowane edytory posiadają odpowiednie interfejsy graficzne użytkownika oraz bez większego problemu wykrywają błędy składniowe. Jednak to nie wszystkie korzyści jakie oferują. Umożliwiają także edycję skomplikowanych i splecionych struktur danych, które wykorzystywane są do pokrycia np. pamięci autobiograficznej, zasad oceny czy też emocji postaci (Dias i in., 2014).

Aby wywołać intensywne i jak najbardziej realistyczne wrażenia w trakcie rozgrywki twórcy gier robią wszystko aby postacie niezależne przypominały prawdziwego człowieka.

Już nie chodzi o sam wygląd czy podejmowane decyzje, ale także o zachowanie. Uniwersytet w Utrechcie stworzył Realizator Behavior Mark-up Languange (BML), który znacznie zwiększa możliwości w tym temacie. Zakres jego działań zaczyna się w fazie definiowania a kończy na samym wyświetlaniu postaci wirtualnych na ekranie. Wykorzystywany jest w celu jak najdokładniejszego odwzorowania niewerbalnych zachowań człowieka takich jak gesty, spojrzenia, mimika twarzy czy też mowa ciała (Westera i in., 2020).

W celu osiągnięcia naturalnej płynności w interakcji człowieka z komputerem postacie niezależne muszą być maksymalnie realistyczne i wiarygodne. W tej sytuacji człowiek angażuje się w doświadczenia, które mogą być również edukacyjne. Coraz bardziej czuje autentyczność uczenia się, które oferuje dane środowisko. Eksperymenty oparte na teorii równań medialnych (Reeves i Nass, 1996) wykazały, że jednostki ludzkie reagują społecznie i naturalnie na różne obiekty nieludzkie, takie jak roboty i awatary, ale także na komputery lub jakiekolwiek obiekty graficzne. Ludzki mózg nie radzi sobie z odróżnianiem interakcji międzyludzkich od symbolicznych czy mediowanych i stąd bierze się ludzka tendencja do antropomorfizmu. Przekłada się to na całkowicie naturalną reakcję międzyludzką w trakcie jakiejkolwiek interakcji. Kolejnym zjawiskiem zachodzącym w ludzkim umyśle jest "chętne zawieszenie niewiary". Pod tym terminem kryje się nic innego jak świadoma akceptacja przez człowieka hipotez, które są nierealistyczne jednak niezbędne by czerpać przyjemność z wykonywanej czynności (Reeves i Nass, 1996). Takie hipotezy często występują nie tylko w grach, ale także w literaturze fikcyjnej a także kinie. Gdybyśmy na przykład nie potrafili przyjąć hipotezy na temat umiejętności latania jaką posiada Superman nie czerpalibyśmy satysfakcji z oglądania jego przygód. Dopiero po świadomym zaakceptowaniu tej nierealistycznej tezy damy porwać się wciągającej fabule tego filmu. Pacman jest kolejnym przykładem, w którym musimy zaakceptować nieprawdziwe założenie i uwierzyć że parę pikseli to prawdziwe duchy. To pokazuje, że nie zawsze trzeba dążyć do osiągnięcia jak największego realizmu. Stopień realizmu powinien zależeć od konkretnego celu, treści oraz kontekstu gry i powinien on zostać ustalony podczas projektowania.

1.3. Przykłady użycia algorytmów

Na przestrzeni lat na rynku gier komputerowych ukazało się wiele produkcji, w których wykorzystana została sztuczna inteligencja. Kilka z nich zdecydowanie wyróżniło się na tle pozostałych, odnosząc wielki sukces oraz w dużym stopniu przyczyniając się do rozwoju nie tylko branży gier, ale także samej sztucznej inteligencji.

Przykładem wielkiej produkcji, która na zawsze odmieniła świat gier strategicznych czasu rzeczywistego, jest Warcraft stworzony przez Studio Blizzard w 1994 roku. Największym przełomem dokonanym przez twórców było stworzenie skomplikowanych i rozbudowanych algorytmów wyszukiwania drogi i wykorzystanie ich dla tak dużej ilości jednostek jednocześnie. Była to pierwsza produkcja, której udało się tego dokonać (Zabłocki, 2018). Sterowanie wszystkimi osobnikami jednocześnie tak, aby nie przeszkadzały sobie nawzajem było bardzo ciężkim zadaniem do zrealizowania, zakończonym jednak wielkim sukcesem. Agenci kontrolowani przez sztuczną inteligencję potrafili omijać przeszkody oraz unikać zderzeń z innymi jednostkami (Winiarski, 2016).

Do rozwoju sztucznej inteligencji przyczyniła się produkcja, która nie jest grą a symulatorem. Chodzi o symulator o nazwie Stworzenia z 1996 roku. Celem rozgrywki w stworzeniach jest wychowanie małych zwierzątek zgodnie ze swoimi upodobaniami. To gracz od samego momentu wyklucia się zwierzątka musi wszystkiego go nauczyć. Jest odpowiedzialny za to by nauczyć go jeść, rozmawiać, bronić się a także odpowiednio zachowywać. W procesie nauki naszych podopiecznych zostały użyte sieci neuronowe. Jest to program, który odegrał ważną rolę w rozwoju uczenia maszynowego. Po raz pierwszy interaktywna symulacja zastosowała tę metodę (Champandard, 2007). Badania nad sztucznym życiem także nabrały tempa dzięki tej produkcji. Gra doskonale odzwierciedla zachowanie zwierząt w ich naturalnym środowisku.

Studio Digital Extremes zdecydowało się na stworzenie gry typu first-person shooter z wykorzystaniem samouczących się botów. Realizacja tego pomysłu ukazała się w roku 1999 pod nazwą Unreal Tournament. Boty występujące w grze potrafiły uczyć się zachowań gracza. Wiedzę na temat poszczególnych zachowań potrafiły wykorzystać w dalszej części rozgrywki, dopasowując swój styl gry oraz używając technik stosowanych przez gracza. Boty potrafiły dostosować swój sposób poruszania się oraz wykorzystywać kryjówki, które wcześniej były zajmowane przez gracza. Do dnia dzisiejszego samouczące się boty są wykorzystywane w wielu grach multiplayerowych (Winiarski, 2016).

Kolejnym tytułem, w którym twórcom udało się stworzyć sztuczną inteligencję na bardzo wysokim poziomie jest wydana w 2001 roku gra Halo: Combat Evolved. Jest to gra z gatunku pierwszoosobowej strzelanki. Podczas rozgrywki stajemy do walki z różnymi obcymi stworzeniami, których celem jest odkrycie tajemnicy skrywanej i chronionej przez gracza. Wrogowie, z którymi przychodzi nam się mierzyć zachowują się bardzo mądrze. Z powodzeniem używają różnego rodzaju osłon, a także potrafią umiejętnie stosować ogień

zaporowy. Każda pojedyncza jednostka jest uzależniona od sytuacji, w której znajduje się jej oddział. W przypadku śmierci przywódcy danego oddziału, niektóre jednostki zostały tak zaprogramowane by zaprzestały walkę i zaczęły uciekać. Szczegóły w grze są bardzo przemyślane i dopracowane. Członkowie naszej drużyny wchodzą z nami w interakcję i zwracają nam uwagę co im przeszkadza. Przeciwnicy potrafią także odrzucać rzucone przez nas granaty (Champandard, 2007).

Kolejnym wielkim przełomem w branży gier komputerowych jest stworzona przez studio Monolith Productions gra F.E.A.R.: First Encounter Assault Recon. Produkcja ukazała się w 2005 roku. Jest to psychologiczny horror w pierwszej osobie z ponurą i fascynującą fabułą. Gracz wciela się w role agenta, który walczy z klonami super-żołnierzy, robotami a także innymi istotami paranormalnymi. Po raz pierwszy w mainstreamowej grze sztuczna inteligencja generuje zachowania, które są wrażliwe na kontekst (Champandard, 2007).

Wykorzystywanie terenu przez przeciwników jest aspektem wartym naszej uwagi. Wrogowie, z którymi walczymy bardzo inteligentnie wykorzystują wszystkie elementy otoczenia. W sytuacji gdy muszą się czymś osłonić przed naszym ogniem potrafią w tym celu wykorzystać stoły lub półki z książkami. Otworzenie i przejście przez drzwi lub rozbicie okna w celu przedostania się na drugą stronę też nie stanowi dla nich problemu.

Współpraca w poszczególnych oddziałach oraz ich taktyka w trakcie rozgrywki także okazały się wielkim sukcesem tej produkcji. Bardzo sprawnie wykonują manewry oskrzydlające czy używają tłumienia ognia. Gra z góry nie narzuca postaciom niezależnym wykonywania poszczególnych czynności w określonych momentach. Flankowanie odbywa się po analizie całego otoczenia, która wskazuje na przydatność użycia tej metody w konkretnej sytuacji. Gracz po schowaniu się za przeszkodą też nie może czuć się do końca bezpiecznym. W tej sytuacji sztuczna inteligencja szybko przetwarza informacje i rzuca granat w celu wyeliminowania gracza lub wybawienia go z kryjówki. Komunikacja między wrogimi postaciami także jest na bardzo wysokim poziomie. Postacie niezależne sprawnie przekazują miedzy sobą informacje na temat poczynań gracza oraz reagują na nie. Wróg sterowany przez sztuczną inteligencją w imponujący sposób, który występuje w tej produkcji wciąż jest obiektem zachwytów osób zainteresowanych tematem gier komputerowych. Dbałość o najdrobniejsze szczegóły sprawia, że gra wyróżnia się na tle innych tytułów. Gra wciąż jest jednym z największych dzieł w kategorii gier typu FPS (first-person shooter - strzelanka z perspektywy pierwszej osoby) (Horti, 2017). By w pełni docenić tę bardzo udaną produkcję

należy wspomnieć o fizyce broni oraz animacjach ruchu, które także były przełomowe jak na swoje czasy. Ta gra zdobyła nagrodę GameSpot "2005 Best AI Award" (Lochiatto, 2020).

W pierwszym rozdziale przedstawione zostały podstawowe informacje z zakresu sztucznej inteligencji oraz wykorzystywania jej w branży gier komputerowych. Opisane zostało jak wiele funkcji pełnią algorytmy sztucznej inteligencji w wielu aspektach rozgrywki jednocześnie. W celu jak najlepszego zrozumienia omawianego tematu, przedstawione zostały różne metody tworzenia algorytmów, którymi posługują się twórcy gier w swoich produkcjach. Metody te zostały także krótko scharakteryzowane a także omówione pod kątem sposobu działania. Oprócz czysto teoretycznych zagadnień, w rozdziale pierwszym zostały zaprezentowane gry, na przykładzie których zobrazowane zostało działanie algorytmów w praktyce. Pierwszy rozdział zawiera informacje na temat wszelkich zastosowań sztucznej inteligencji w grze, natomiast w drugim rozdziale zawężony został zakres przedstawionych treści. Skupia się on na algorytmach służących do wyszukiwania drogi w grach. Zostały w nim ujęte metody wykorzystywane do podziału środowiska gry na mniejsze fragmenty co zdecydowanie ułatwia pracę algorytmom wyszukiwania drogi. W drugiej części rozdziału znajdują się opisy poszczególnych algorytmów a także przedstawione zostały ich zastosowania w konkretnych przykładach.

2. Wyszukiwanie drogi w grach

Pierwszy rozdział zawiera podstawowe informacje o wykorzystywaniu sztucznej inteligencji w grach w wielu zróżnicowanych aspektach rozgrywki. Drugi rozdział natomiast skupiony jest w pełni na jednym konkretnym jej zastosowaniu, jakim jest wyszukiwanie drogi. Jest to jeden z najważniejszych elementów sztucznej inteligencji w grze. W związku z tym, zagadnienie to jest dość obszerne, jednak informacje zawarte w tym rozdziale pozwalają poznać oraz zrozumieć podstawy działania algorytmów odpowiedzialnych za wyszukiwanie trasy. W pierwszej części rozdziału znajdują się informacje na temat pierwszego etapu wyszukiwania drogi jakim jest podział środowiska gry. Druga część rozdziału zawiera charakterystykę poszczególnych algorytmów oraz zasady ich działania. W ostatniej części rozdziału opisane zostało zastosowanie konkretnych algorytmów w grach.

Wyszukiwanie drogi znajduje swoje zastosowanie w aplikacjach, których funkcje oraz zakres działania są zróżnicowane. Oczywistym zastosowaniem są aplikacje nawigacyjne lub logistyczne, jednak pathfinding wykorzystywany jest też w robotyce a także w symulacji tłumu (Abd Algfoor i in., 2015). Branża gier wideo jest kolejnym obszarem działania. Wyszukiwanie drogi pełni w niej bardzo ważną rolę. Stworzenie wielu gier nie byłoby możliwe bez implementacji odpowiednich algorytmów opracowanych w tym celu. Dzięki temu omawiane zagadnienie stało się najpopularniejszym i bardzo uciążliwym problemem towarzyszącym programistom w trakcie tworzenia gier (Cui i Shi, 2011).

Badania nad sztuczna inteligencją w zakresie wyszukiwania drogi trwają nieustannie od wielu lat (Cui i Shi, 2011). W trakcie tych badań zostało odkrytych wiele metod, których wykorzystanie pozwalało na skuteczne wdrażanie algorytmów sterujących postaciami. Wraz z postępem towarzyszącym badaniom, odkrywane metody były coraz skuteczniejsze oraz bardziej zoptymalizowane.

W każdej grze, w której przemieszczanie się po świecie przez postacie niezależne jest elementem rozgrywki muszą zostać zastosowane algorytmy wyszukiwania drogi. Ich zadaniem jest określenie trasy, po której możliwe jest przemieszczenie się z dowolnego miejsca świata do innego (Graham i in., 2003).

2.1. Sposoby podziału środowiska gry

Wszystkie obszary znajdujące się w środowisku gry wchodzą w skład mapy. Struktura ta jest odpowiedzialna za przechowywanie informacji na temat geometrii świata gry (Graham i in., 2003). Określenie jakiejkolwiek trasy nie jest możliwe bez tych informacji.

Pierwszym krokiem, który niezbędny jest do późniejszego wyznaczania trasy jest podział mapy na mniejsze, uproszczone kawałki. Celem takiego działania jest jak największe uproszczenie obszaru, który musi zostać przeszukany. Skutkiem takiego działania jest przyspieszenie działania algorytmu wyszukiwania drogi. Do podziału mapy stosuje się: (Verma i in., 2015): drzewa czwórkowe, uogólnione cylindry a także siatki nawigacyjne (regularne lub nieregularne).

Drzewa czwórkowe swoją nazwę zawdzięczają zasadzie swojego działania. Generowanie drzewa polega na wielokrotnym dzieleniu każdego obszaru na cztery równe części (Abd Algfoor i in., 2015). Cała mapa początkowo zostaje podzielona na cztery równe obszary, w których sprawdzane jest występowanie różnego rodzaju przeszkód lub kolizji. Gdy w danym obszarze zostaje wykryte jakiekolwiek z wyżej wymienionych zjawisk, obszar ten ponownie dzielony jest na cztery równe części. Ponownie przeprowadzany jest proces sprawdzania oraz dzielenia kolejnych powierzchni. Takie działanie sprawia, że każdy węzeł w drzewie ma cztery węzły potomne. Chociaż liczba możliwych węzłów jest ogromna, nie jest ona jednak nieskończona. W sytuacji braku jakiejkolwiek kolizji lub przeszkody w sprawdzanym obszarze lub gdy podzielone sektory osiągną minimalną dopuszczalną wielkość, proces dzielenia zostaje zakończony (Abd Algfoor i in., 2015).

Zasada działania uogólnionych cylindrów opiera się na bryle geometrycznej jaką jest walec, który służy do określania przestrzeni między przeszkodami (Stout, 1999). W badanym środowisku, w którym występują blokady oraz miejsca uniemożliwiające poruszanie się, każda przestrzeń między sąsiadującymi ze sobą przeszkodami jest osobno badana. Przyjmuje ona kształt walca a następnie w celu skutecznego działania tej metody, wyznaczana jest jego oś. To właśnie osie poszczególnych brył są wykorzystywane do późniejszego wyznaczania trasy (Stout, 1999). Wyszukiwanie drogi nie jest jedynym zastosowaniem tej metody. Ze skutecznością jest także stosowana do wykonania kilku zróżnicowanych zadań, takich jak (Cao i in., 2006): projektowanie systemów wizyjnych w robotyce, odzyskiwanie kształtów, modelowanie obiektów, modelowanie gałęzi drzew w grafice komputerowej oraz segmentacja i detekcja oparta na modelu.

Do podziału środowiska gry najczęściej stosuje się różnego rodzaju siatki nawigacyjne (Graham i in., 2003). Są one najskuteczniejszym sposobem na reprezentację mapy w jak najprostszej postaci. W ogólnym rozumieniu siatki to nic innego jak grafy, których elementami są punkty lub wierzchołki a ich połączenie następuje poprzez krawędzie (Abd Algfoor i in., 2015). Stworzenie jak najlepszego i najdokładniejszego grafu jest ważnym

elementem całego procesu wyszukiwania drogi. Poszczególne rodzaje siatek różnią się od siebie nie tylko pod względem różnorodności kształtów na jakie dzielą płaszczyznę, ale przede wszystkim sposobem ich definiowania. Właśnie ta różnica sprawia, że siatki mogą być regularne lub nieregularne (Abd Algfoor i in., 2015).

W przypadku siatek regularnych obszar zostaje podzielony na kawałki nie tylko o takim samym rozmiarze ale także o nieodmiennym kształcie, który zależny jest od konkretnego rodzaju siatki z tej grupy. Kształt ten powinien zostać dobrany na podstawie elementów środowiska, aby siatka jak najlepiej się do nich dostosowała. Ważne jest także czy środowisko gry jest dwuwymiarowe czy jednak świat przedstawiony jest w trzech wymiarach. Do stworzenia siatki w środowiskach dwuwymiarowych służą obszary przybierające kształty trójkątne, kwadratowe lub sześciokątne, natomiast w trójwymiarowych terenach wykorzystywana jest siatka składająca się z sześcianów (Abd Algfoor i in., 2015).

Siatki nieregularne w odróżnieniu od tych regularnych, nie dzielą mapy równomiernie. Wyznacznikiem, który definiuje rozmiar oraz kształt podzielonych obszarów jest obecność przeszkód oraz ich usytuowanie. Sąsiadujące ze sobą przeszkody określają parametry danego obszaru. Nawet gdy znajdują się daleko od siebie, nie ma potrzeby dzielenia obszaru na jeszcze mniejsze fragmenty, ponieważ wewnątrz danego sektora nie istnieje żadna przeszkoda uniemożliwiająca ruch. Za nieregularne siatki uważane są (Abd Algfoor i in., 2015): grafy punktów widoczności, siatki nawigacyjne wypukłe a także grafy punktów orientacyjnych.

Na rozmieszczenie punktów widoczności używanych do stworzenia grafu mają wpływ wszystkie przeszkody znajdujące się na mapie, ponieważ punkty usytuowane są w pobliżu ich wierzchołków. Punkty te jednak muszą znajdować się w odpowiedniej odległości od tych wierzchołków, tak aby nie dochodziło do kolizji między nimi (Stout, 1999). Oznacza to, że odległość między wierzchołkiem a punktem musi być na tyle duża, aby postać w momencie osiągnięcia jakiegokolwiek punktu nawigacyjnego nie wchodziła w żadną kolizję z przeszkodą oraz nie została przez nią w żaden sposób zablokowana. Ponadto sąsiadujące punkty połączone ze sobą muszą się wzajemnie widzieć (Stout, 1999). Między nimi nie może znajdować się żaden element, który uniemożliwi przemieszczenie się z jednego punktu do drugiego po linii prostej. Jedną z zalet tej metody jest duża szybkość działania. Dzięki temu może być używana nie tylko w środowiskach statycznych ale także może zostać użyta w czasie rzeczywistym, w środowisku z przemieszczającymi się przeszkodami (Kulbiej, 2018).

Siatki nawigacyjne wypukłe są jedną z najpopularniejszych metod podziału mapy a swoje zastosowanie znajdują w grach, w których świat jest trójwymiarowy (Verma i in., 2015). Mapa przedstawiona za pomocą tej metody jest podzielona na wielokątne, wypukłe obszary po których mogą poruszać się postacie (Cui i Shi, 2011). W momencie gdy punkt docelowy znajduje się w innym wielokącie niż postać próbująca się do niego dostać, wyszukiwany jest obszar, który sąsiaduję z zajmowanym aktualnie przez postać sektorem, którego zdobycie pozwoli nam zbliżyć się do miejsca docelowego. Dopiero gdy wszystkie kolejne obszary aż do tego, w którym znajduje się punkt docelowy zostaną zbadane, następuje wytyczenie trasy, po której postać przemieści się od punktu startowego prosto do swojego celu (Verma i in., 2015).

Kolejnym sposobem reprezentowania mapy z wykorzystaniem nieregularnej siatki nawigacyjnej jest wykres punktów orientacyjnych. Wykres ten składa się z wielu punktów, zadaniem których jest określenie położenia w przestrzeni fizycznej (Cui i Shi, 2011). Każdy punkt musi być połączony z co najmniej jednym innym punktem tworząc w ten sposób jedną spójną siatkę. Osiągnięcie wybranego przez nas celu musi być możliwe z dowolnego miejsca znajdującego się na siatce. Najczęściej twórcy gier sami określają wszystkie punkty orientacyjne na mapie. Celem takiego zabiegu jest stworzenie jak najwydajniejszej reprezentacji mapy (Graham i in., 2003). Stworzenie dużej siatki, łączy się z ustanowieniem wielu punktów orientacyjnych a to może okazać się bardzo czasochłonne. Jednak czym więcej odniesień zdefiniujemy, tym ruch będzie wydawał się naturalniejszy, ponieważ postać będzie przechodzić dokładnie przez te punkty. Wykorzystanie tej metody nie jest możliwe w dynamicznych środowiskach. Statyczne światy są jednak obszarem, w których grafy punktów orientacyjnych sprawdzają się bardzo dobrze (Graham i in., 2003). Podobnie jest ze światami przedstawionymi w dwóch wymiarach, gdzie wykorzystanie tej metody daje bardzo zadowalające efekty. Jednak w trójwymiarowych mapach nie jest to najlepsze rozwiązanie, ze względu na złożoność otoczenia trójwymiarowego. By osiągnąć chociaż trochę zadowalający efekt, potrzebne byłoby określenie bardzo wielu punktów orientacyjnych (Cui i Shi, 2011).

2.2. Charakterystyka poszczególnych algorytmów wyszukiwania drogi

Wykorzystanie algorytmów stworzonych do wyszukiwania najkrótszej trasy pomiędzy dwoma punktami w świecie gry jest kolejnym etapem w procesie wyszukiwania drogi. W swoim działaniu uwzględnia dokonany wcześniej podział mapy, tak więc podczas

wdrażania jakiegokolwiek algorytmu trzeba pamiętać o dostosowaniu go do wykorzystanej metody reprezentacji mapy.

Mapa przedstawiona w postaci siatki może być przeszukiwana za pomocą wielu algorytmów opracowanych w tym celu. Algorytmy te stosują różne techniki, jednak każdy dąży wykonania jednego zadania, jakim jest znalezienie najkrótszej trasy łączącej ze sobą dwa punkty: startowy i docelowy. Metody przeszukiwania dostępnych tras zasadniczo dzielą się na dwa typy (Dudek, 2014):

- nieinformowane (ślepe), które w procesie przeszukiwania nie wykorzystują żadnych dodatkowych informacji;
- informowane (heurystyczne), wykorzystujące w procesie przeszukiwania dodatkowe informacje.

Ze względu na brak informacji w strategiach nieinformowanych, przeszukiwanie odbywa się w sposób nieukierunkowany. Oznacza to, że algorytm zaczyna swoje działanie w punkcie startowym a następnie na ślepo bada kolejne punkty z nadzieją, że w końcu dotrze do punktu docelowego. Skutkiem takiego działania są sytuacje, w których algorytm bada trasę prowadzącą w całkowicie innym kierunku niż nasz punkt docelowy. Najbardziej popularnymi strategiami ślepymi są (Graham i in., 2003): przeszukiwanie wszerz (Breadth-first search) oraz przeszukiwanie w głąb (Depth-first search).

Przeszukiwanie wszerz polega na przeszukaniu węzłów poziom po poziomie (Dudek, 2014). Pierwszym krokiem w tej metodzie jest rozwinięcie węzła w punkcie startowym. Następnie sprawdzane są punkty dostępne z jego poziomu. Gdy wszystkie te punkty zostaną sprawdzone, dla każdego z nich zostają generowani potomkowie. Ponownie dopiero po sprawdzeniu wszystkich tych potomków tworzona zostaje kolejna warstwa. Proces ten jest powtarzany aż do momentu znalezienia rozwiązania. Przedstawiona zasada działania odpowiada modelowi kolejki FIFO (First In, First Out), która charakteryzuje się umieszczaniem nowego węzła potomnego powstałego z rozwinięcia bieżącego węzła na końcu kolejki określającej, który węzeł zostanie rozwinięty jako następny (Kołodziejczyk, 2017). Gdy do celu prowadzi więcej niż jedna droga, możemy mieć pewność, że znajdzie tę, która posiada najmniej punktów łączących punkt startowy z docelowym. Jednak wyszukiwanie to może być czasochłonne, ponieważ na zmianę sprawdzane są kolejne odcinki, wszystkich możliwych tras wychodzących z punktu startowego.

Druga metoda zwana przeszukiwaniem w głąb jest przeciwieństwem do wcześniej opisywanego przeszukiwania wszerz (Graham i in., 2003). Punkt startowy stanowi początek

naszego drzewa. Algorytm nie skupia się od razu na wszystkich jego potomkach, lecz początkowo zajmuje się tylko jednym z nich. Rozwija go, ale w dalszej części procesu sprawdzania trasy ponownie bierze pod uwagę tylko jeden węzeł dla którego potomek ten jest rodzicem. Działanie to powtarzane jest do momentu, w którym sprawdzany punkt nie posiada żadnego potomka lub punkt ten okazał się punktem docelowym. Jeśli okaże się, że rozwinięcie danego punktu jest niemożliwe z powodu braku potomków, algorytm wraca do najbliżej znajdującego się węzła i ponawia procedurę związaną z rozwijaniem jednego potomka. W tej metodzie stosowany jest model kolejki LIFO (Last In, First Out), w której nowo utworzony węzeł potomny umieszczany jest od razu na samym początku kolejki (Kołodziejczyk, 2017). Przeszukiwanie w głąb może występować w formie standardowej, ale także w formie ograniczonej lub pogłębianej iteracyjnie (Dudek, 2014). W obydwóch tych modyfikacjach zasada działania się nie zmienia, jednak w wersji ograniczonej przeszukiwanie odbywa się tylko do pewnej głębokości, po osiągnięciu której algorytm nie rozwija już aktualnie sprawdzanego węzła lecz wraca się do jego rodzica w celu sprawdzenia kolejnego potomka. Takie ograniczenie sprawia, że algorytm może nie znaleźć drogi do punktu docelowego. Aby tak się nie stało można zastosować pogłębienie iteracyjne, które jest rozszerzeniem przeszukiwania ograniczonego (Dudek, 2014). Gdy w obrębie podanej głębokości nie została znaleziona trasa do punktu docelowego, zostaje dodany jeden poziom węzłów potomnych. Czynność ta jest powtarzana aż do osiągnięcia celu.

W trakcie przeszukiwania nieinformowanego algorytmy miały za zadanie znalezienie trasy łączącej dwa punkty bez zwracania uwagi na koszt przejścia przez poszczególne punkty aż do celu. W strategiach informowanych koszt ten wyliczany jest na bieżąco w trakcie obliczania dostępnych tras. Do wyliczania kosztów poszczególnych tras można użyć (Graham i in., 2003):

- przeszukiwania o jednolitym koszcie (uniform cost search);
- przeszukiwania heurystycznego.

Metoda przeszukiwania o jednolitym koszcie służy do wyszukiwania drogi pomiędzy punktem startowym i docelowym, posiadającej najniższy koszt całkowity (Jagga, 2020). Swoja pracę zaczyna od punktu startowego i ocenia koszt każdego połączenia z sąsiadującymi punktami. Koszty te zostają zapamiętane, nawet jeśli dany węzeł nie zostanie w tym momencie wybrany. Wybierany jest węzeł, którego koszt jest najmniejszy, a następnie liczony jest koszt wszystkich nieobliczonych jeszcze połączeń z tym właśnie punktem w celu określenia ich odległości od punktu startowego. Koszt tych tras zostaje zaktualizowany

i ponownie wybierany jest węzeł o najmniejszym koszcie trasy rozpoczynającej się od punktu startowego a sprawdzane są wszystkie aktualnie otwarte węzły, również te które zostały odrzucone podczas wcześniejszej selekcji. Gdy punkt docelowy zostanie znaleziony, trasa łącząca go z punktem startowym oraz jej koszt zostają zapamiętane, jednak działanie algorytmu nie zostaje przerwane. Zakończenie jego pracy nastąpi dopiero wtedy, gdy koszt wszystkich otwartych węzłów przekroczy koszt tej trasy.

Przykładem algorytmu, który wykorzystuje lekko zmodyfikowaną metodę przeszukiwania o jednolitym koszcie jest algorytm Dijkstry, stworzony przez holenderskiego naukowca i informatyka Edsgera Dijkstrę oraz opublikowany w 1959 roku (Cassingena Navone, 2020). Zasadniczą różnicą pomiędzy wyszukiwaniem o jednolitym koszcie a algorytmem Dijkstry jest ilość sprawdzonych węzłów oraz końcowa zawartość wykresu definiującego znalezione drogi (Wu, 2021). Przeszukiwanie o jednolitym koszcie sprawdza tylko te punkty, które mogą być przydatne do wyznaczenia najkrótszej trasy, natomiast algorytm Dijkstry oblicza koszt trasy pomiędzy punktem startowym a wszystkimi węzłami znajdującymi się w grafie. Dzięki temu, wykres sporządzony przy pomocy przeszukiwania o jednolitym koszcie może być tylko częściowy, natomiast algorytm Dijkstry tworzy całkowity wykres wszystkich tras. Wiąże się to oczywiście z różnicą w wykorzystaniu pamięci oraz czasu.

Metody heurystyczne charakteryzują się większa ilością danych wejściowych, które są informacjami na temat ścieżki, którą chcemy odnaleźć (Kołodziejczyk, 2017). Wykorzystanie tych informacji pozwala na zwiększenie wydajności algorytmów stosujących te metody. Zadaniem funkcji heurystycznej jest zmniejszenie ilości sprawdzanych tras, tak aby badane były tylko te, które są korzystne. W tym celu szacowany jest koszt trasy w linii prostej od każdego punktu do punktu docelowego (Cui i Shi, 2011). Jednak oszacowany koszt rzadko pokrywa się z rzeczywistym kosztem trasy, która musi zostać przebyta. Wystarczy, że na mapie pojawi się jakaś przeszkoda, która uniemożliwia zdobycie punktu docelowego bez konieczności ominięcia jej. Wtedy może okazać się, że punkt, który teoretycznie jest dalej od naszego punktu docelowego będzie jednak częścią naszej najkrótszej trasy, ponieważ pozwoli ominąć przeszkodę mniejszym kosztem.

Najpopularniejszymi przykładami informowanych algorytmów wykorzystujących funkcje heurystyczne są (Kołodziejczyk, 2017):

- zachłanne przeszukiwanie metodą "pierwszy najlepszy" (Greedy Best-first search);
- algorytm A* (A-star).

Zachłanne przeszukiwanie metodą "pierwszy najlepszy" w procesie badania ewentualnej użyteczności trasy wykorzystuje tylko funkcję heurystyczną (Kołodziejczyk, 2017). Oznacza to, że wpływ na decyzję o kolejności rozwijania kolejnych węzłów ma tylko i wyłącznie szacowany koszt drogi od punktu badanego do punktu docelowego. Węzeł, który w danej chwili posiada najmniejszy szacowany koszt jest rozwijany jako następny. Dzięki temu kontrolowane jest to, czy dana trasa zmierza w kierunku naszego celu. Zastosowanie heurystyki zdecydowanie zwiększa skuteczność oraz wydajność tego algorytmu, jednak nie jest on bez wad. Największą wadą takiego przeszukiwania jest brak obliczenia przebytego już dystansu
z uwzględnieniem kosztowności terenu (Stout, 1999). Może się zdarzyć, że droga wyznaczona przy użyciu opisywanej metody (mająca najmniejszy heurystyczny koszt) będzie przebiegać przez wymagający, kosztowny teren. W związku z tym, ominięcie tego terenu dłuższą, lecz mniej kosztowną drogą może docelowo sprawić, że całkowity koszt trasy omijającej dany teren będzie niższy niż trasy wybranej. Niestety ze względu na zasadę działania metody pierwszy najlepszy, realne koszty ścieżek nie zostaną obliczone, a my zostaniemy poprowadzeni kosztowniejszą drogą. To pokazuje, że ta metoda nie jest optymalna (nie zawsze znajdzie najkrótszą trasę) (Kołodziejczyk, 2017).

Algorytm A* (A-star) jest jednym z najpopularniejszych algorytmów stosowanych do wyszukiwania drogi w grach (Graham i in., 2003). W swoim działaniu łączy techniki stosowane w dwóch wcześniej scharakteryzowanych metodach: algorytmie Dijkstry oraz zachłannego przeszukiwania pierwszy najlepszy. Jest bardzo udanym oraz kompletnym algorytmem, ponieważ łączy wszystkie dobre cechy tych algorytmów, jednocześnie nie powielając ich wad (Verma i in., 2015). Stosuje technikę przeszukiwania o jednolitym koszcie oraz heurystykę (Graham i in., 2003). Przy wyznaczaniu kolejności rozwijania węzłów wykorzystuje sumę obliczeń z tych dwóch metod. Oblicza koszty drogi jaką musimy przebyć od punktu startowego do miejsc aktualnie sprawdzanych a następnie dodaje do nich heurystyczne koszty trasy od tych miejsc aż do punktu docelowego. To powoduje wstępne odrzucenie tras pozornie dłuższych, jednak w przeciwieństwie do przeszukiwania pierwszy najlepszy istnieje możliwość powrotu do nich. Dzięki temu w znajdowaniu najkrótszej trasy jest tak dobry jak algorytm Dijkstry, jednak dzięki funkcji heurystycznej jego szybkość działania jest większa, zbliżona do metody pierwszy najlepszy. Kolejnym argumentem przemawiającym na korzyść algorytmu A* jest jego optymalność. Zawsze znajdzie najkrótszą trasę do celu (Kołodziejczyk, 2017).

2.3. Metody wyszukiwania drogi stosowane w grach

Do wyszukiwania drogi w grach opracowanych zostało wiele metod różniących się od siebie parametrami a także specyfiką działania. Ze względu na te różnice użyteczność poszczególnych algorytmów może zależeć od typu gry, w której mają zostać użyte. Jednak nie jest to jedyny problem towarzyszący twórcom gier w trakcie wyboru konkretnego rozwiązania. Muszą oni także wziąć pod uwagę strukturę gry oraz jej środowisko. Jak najlepsze wdrożenie wybranej techniki jest kolejnym ważnym krokiem w procesie tworzenia inteligentnego ruchu postaci niezależnych w grze.

Studio Grand Cru stworzyło grę o nazwie Supernauts, w której świat gry składa się z układanych przez gracza w dowolny sposób klocków (Aske, 2014). Do reprezentacji mapy w grze została użyta siatka nawigacyjna wypukła. Jednak po jej wdrożeniu oraz po przeprowadzeniu testów została ona zmodyfikowana, poprzez podzielenie jej na większą ilość mniejszych siatek w celu zaoszczędzenia zasobów w trakcie inicjalizacji świata oraz jego aktualizacji (Hatinen, 2014). W grze został użyty algorytm A*, jednak musiał on zostać w odpowiedni sposób zmodyfikowany, tak aby dopasować go do zastosowanego sposobu reprezentacji mapy. Ze względu na zastosowanie wielu siatek nawigacyjnych jednocześnie, algorytm oprócz wyszukiwania drogi w danej siatce musiał oceniać połączenia między znajdującymi się w niej węzłami, a sąsiednimi węzłami, które znajdowały się już w innej siatce (Hatinen, 2014). Dzięki temu możliwe było poruszanie się z obszaru jednej siatki na obszar innej siatki.

Gra PlayStation Move Heroes, która początkowo nosiła nazwę Heroes on the Move została wydana w 2011 roku. Twórcy gry do reprezentacji terenu także zastosowali siatkę nawigacyjną wypukłą, która podzieliła środowisko gry na trójkątne obszary. Jednak w tym przypadku przy jej pomocy stworzyli dwa grafy (Anhalt i in., 2011): niskopoziomowy oraz wysokopoziomowy. Pierwszy z nich jest zwykłym grafem, który łączy poszczególne węzły znajdujące się w przestrzeni gry. Natomiast wysokopoziomowy graf zwany także grafem abstrakcyjnym łączy poszczególne grupy mniejszych obszarów w większy teren, po którym gracz powinien poruszać się w trakcie rozgrywki. Na każde połączenie między większymi terenami przypadają dwa węzły łączące je. Twórcy przy zastosowaniu abstrakcyjnego grafu chcieli osiągnąć efekt gładkiego, płynnego ruchu (Anhalt i in., 2011). Algorytm, który został użyty do wyszukiwania drogi nie został przez twórców ujawniony. Jednak podzielili się informacją na temat ogólnego sposobu wykorzystanego w tym procesie. Wykorzystali stopniowe wyszukiwanie drogi (incremental pathfinding), który polega na podzieleniu

problemu wyznaczenia trasy na większą ilość mniejszych problemów oraz późniejsze pojedyncze ich rozwiązywanie (Anhalt i in., 2011).

Stworzona przez studio The Molasses Flood gra o nazwie The Flame in the Flood jest kolejną produkcją, w której w procesie wyszukiwania drogi zastosowane zostały wcześniej opisane metody. Jest to gra survivalowa, w której znajdujemy się na wyspie, jednak oprócz niej występują także inny wyspy, do których możemy się udać. Tym razem do reprezentacji mapy zastosowana została regularna, kwadratowa siatka, a w procesie obliczania tras użyty został algorytm Dijkstry (Abercrombie i Isla, 2016). Ze względu na zasadę działania podanego algorytmu, obliczana jest odległość między naszą postacią a każdym dostępnym punktem na mapie.

W popularnej grze strategicznej czasu rzeczywistego jaką jest Age of Empires, także zastosowana została regularna, kwadratowa siatka do reprezentacji przestrzeni gry, lecz do wyznaczania tras został wdrożony inny algorytm niż w poprzednio omawianym tytule, a mianowicie algorytm A* (Cui i Shi, 2011). Pomimo popularności tego algorytmu oraz częstego wykorzystywania go w grach komputerowych, w tej produkcji twórcy gier nie osiągnęli jednak zadowalającego efektu. Wielu graczy skarżyło się na bardzo słabo wdrożony system wyszukiwania drogi (Cui i Shi, 2011).

W kolejnej grze tego typu, która nosi nazwę Civilization V, także została zastosowana regularna siatka, tym razem jednak o kształcie sześciokąta. Niestety informacja na temat wykorzystywanego przez twórców algorytmu odpowiedzialnego za poruszanie się postaci nie została ujawniona, jednak podobnie jak w poprzednim przypadku, wyszukiwanie drogi także było na bardzo słabym poziomie (Cui i Shi, 2011).

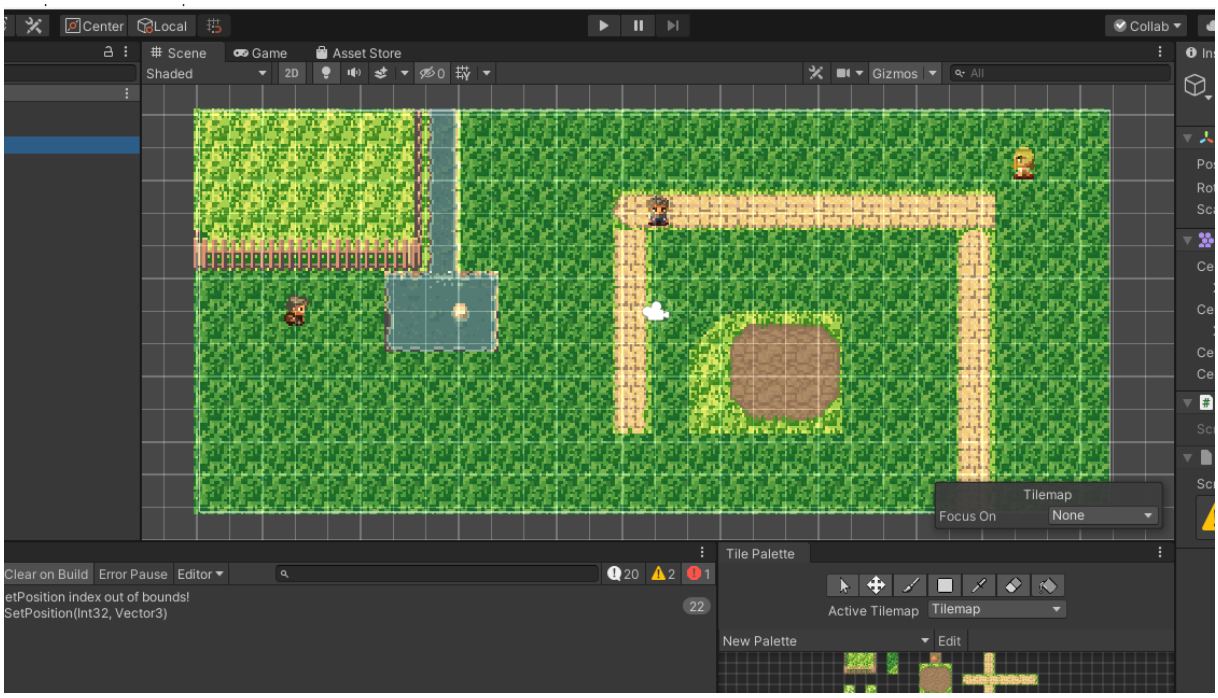
Jeśli chodzi o sposób podziału mapy, najczęściej wybierany jest jeden, przy użyciu którego zostaje zaprezentowane całe środowisko gry. Jednak w celu nawigacji jednostek twórcy gier czasami w jednej grze używają dwóch lub nawet większej ilości algorytmów wyszukiwania drogi, tak jak zostało to rozwiązane w bardzo popularnej grze strategicznej czasu rzeczywistego Starcraft II. Mapa przedstawiona jest za pomocą siatki nawigacyjnej wypukłej, podzielonej na trójkątne obszary (Anhalt i in., 2011). Wyszukiwanie drogi odbywa się za pomocą hybrydowej metody, łączącej dwa algorytmy: A* oraz algorytmu stadnego (Hagelbäck, 2016). Użycie odpowiedniego algorytmu zależy od sytuacji, w której dana jednostka się znajduje. Jeśli w zasięgu tej jednostki nie znajdują się żadne inne obiekty (przeciwnicy, budynki przeciwnika) sterowana jest ona za pomocą algorytmu A*. Jednak gdy, któryś z tych obiektów pojawi się w najbliższym otoczeniu jednostki kontrolę nad jej ruchem

przejmuje algorytm stadny. Odpowiedzialny jest on za sterowanie ruchem z uwzględnieniem pozycji wrogich obiektów oraz utrzymywanie od nich odpowiedniej odległości, jednocześnie nie oddalając się od swojej grupy. Odległość ta jest różna dla każdego typu jednostki, którą steruje i jest ona definiowana na podstawie jej maksymalnego zasięgu ataku.

Drugi rozdział zawierał informacje dotyczące wyszukiwania drogi w grach. W pierwszej jego części wyszczególnione zostały metody wykorzystywane w procesie podziału środowiska gry oraz sposób reprezentacji tego środowiska. Kolejna część poświęcona była najbardziej popularnym metodom przeszukiwania oraz wyznaczania trasy wraz z ich charakterystyką. W ostatniej części rozdziału znajdują się informacje na temat wykorzystywania różnych technik wyszukiwania drogi w kilku przykładowych grach. Do tej pory w niniejszej pracy znajdowały się teoretyczne informacje oraz przykładowe gry wykorzystujące wcześniej opisane techniki. Natomiast w trzecim rozdziale znajduje się opis autorskiego projektu dwuwymiarowej gry, który został wykonany na potrzeby pracy. Celem projektu jest pokazanie, jak w praktyce wygląda cały proces tworzenia algorytmu odpowiedzialnego za wyszukiwanie drogi w grze. Jednak algorytm nie został stworzony tylko w celu wyznaczania najkrótszej trasy. W jego działaniu zostały zaprogramowane dodatkowe funkcjonalności, które pozwalają na omijanie wroga oraz znajdowanie optymalnej ścieżki, uwzględniając podłoże, po którym porusza się nasz bohater.

3. Wdrożenie algorytmu wyszukiwania drogi

Celem zaprojektowanej gry jest znajdowanie trasy z uwzględnieniem podłoża oraz zasięgu wroga przy wykorzystaniu algorytmu wyznaczania trasy. Zaprojektowana gra dwuwymiarowa wykorzystuje to, że plansza rozgrywki złożona jest z kwadratowej siatki. W praktyce dzięki temu możliwym jest określanie położenia wroga, drogi, wody czy innych elementów terenu na konkretnym polu w siatce. Gra obsługuje ruch gracza za wskazaniem kliknięcia przycisku myszy oraz rysowanie linii, wskazującej najkorzystniejszą trasę od miejsca, w którym znajduje się gracz do punktu docelowego (księżniczki).



Rysunek 1 Projekt w Unity

W strukturze świata gry została określona jego wysokość i szerokość oraz zawarta została tablica dwuwymiarowa, w której znajdują się wszystkie pola w grze.

```
public class World
{
    public Field[,] Fields;
    public float Width;
    public float Height;

    public World(Field[,] fields, float width, float height)

        this.Fields = fields;
        this.Width = width;
        this.Height = height;
    }
```

Każde pole zawiera szereg właściwości określających jego nazwę, informacje czy jest przeszkodą, położenie X oraz Y, wartość (dla drogi lub trawy), wartości wykorzystywane przez algorytm opisane dalej oraz powiązanie z rodzicem, wykorzystywane do odwzorowania ścieżki.

```
public class Field
{
    public string TileName { get; set; }
    public bool isAvailableToWalk { get; private set; }
    public int X { get; private set; }
    public int Y { get; private set; }
    public float Value { get; private set; }

    public float GCost { get; private set; }
    public float HCost { get; private set; }
    public float FCost => GCost  * HCost;

    public Field ParentField { get; set; }


    public Field(string tileName, int x, int y, bool isAvailableToWalk, float
value)
    {
        this.isAvailableToWalk = isAvailableToWalk;
        this.X = x;
        this.Y = y;
        this.TileName = tileName;
        this.Value = value;
    }
```

W niniejszej grze wykorzystano heurystyczny algorytm A*, który oblicza ścieżkę pomiędzy punktem źródłowym oraz docelowym. W przypadku gry 2d wybór pól odbywa się przy wykorzystaniu poniższego równania:

$$f(x) = g(x) + h(x)$$

Gdzie:

$f(x)$ – koszt całkowity ruchu,

$g(x)$ – koszt ruchu z punktu startowego do aktualnie rozpatrywanego,

$h(x)$ –heurystyczny koszt przejścia pomiędzy punktem aktualnie rozpatrywanym oraz punktem docelowym.

W praktyce algorytm A* analizuje trasę uwzględniając powyższy wzór szacując koszty przejścia pomiędzy danymi punktami przyległymi w kolejnych iteracjach.

3.1. Wyliczanie najkrótszej trasy

Implementacja algorytmu A* została zaprezentowana poniżej. Algorytm bazuje na polach przejrzanych (zamkniętych) oraz nieprzejrzanych (otwartych). W tym celu inicjowane są dwie kolekcje, które będą przechowywać informacje o tych polach. Dodatkowo przez inicjalizacją pierwszej iteracji do listy otwartych zostaje dodany punkt startowy. Czyli będzie to aktualne położenie gracza.

```
ISet<Field> closedSet = new HashSet<Field>();
IList<Field> openSet = new List<Field>();
openSet.Add(fieldStart);
```

Następnie do aktualnego rozpatrywanego punktu (w pierwszej iteracji jest to punkt startowy) przeszukiwane są pola przyległe. Celem jest znalezienie pola o jak najmniejszym koszcie F. W czasie iteracji po polach przyległych sprawdzanym jest czy jest to pole dozwolone do przechodzenia – dla pola, które stanowi przeszkodę. W wyniku obliczeń dodawany jest również odpowiedni koszt pola – dzięki temu możliwym będzie wprowadzanie wag danych pól. W konsekwencji w grze przejście drogą będzie korzystniejsze niż poruszanie się po trawie. Oprócz wag do obliczania kosztu H (koszt heurystyki) pól przyległych wykorzystano odległość Manhattan ze współczynnikiem 10 dla pól w pionie i poziomie oraz 14 dla pól po skosie.

```
IList<Field> neighbours = world.GetNeighbours(currentNode);

            foreach (var neighbour in neighbours)
            {
                if (!neighbour.isAvailableToWalk) continue;

                if (closedSet.Contains(neighbour)) continue;

                int costToNeighbour = (int)currentNode.GCost + (GetDistance(cu
rrentNode, neighbour) *  neighbour.GetValue());
                if (costToNeighbour < neighbour.GCost || !openSet.Contains(nei
ghbour))
                {
                    neighbour.SetGCost(costToNeighbour);
                    neighbour.SetHCost(GetDistance(neighbour, fieldTarget));
                    neighbour.SetParentField(currentNode);

                    if (!openSet.Contains(neighbour))
                        openSet.Add(neighbour);
                }
            }
```

W kolejnej iteracji poprzednie pole staje się rodzicem aktualnego i zostaje dodane do listy zamkniętych i nie będzie analizowane jako pole przyległe.

```
            openSet.Remove(currentNode);
            closedSet.Add(currentNode);
```

Algorytm sprawdza również czy na liście otwartych nie znajduje się już korzystniejszy element ścieżki. W każdej iteracji sprawdzanym jest dodatkowo otoczenie poprzedniego rodzica. Jeżeli jest tam korzystniejszy element drogi to zostaje on dodany jako pole aktualnie rozpatrywane.

```
             for (int i = 1; i < openSet.Count; i++)
            {
                if (openSet[i].FCost < currentNode.FCost
                    || openSet[i].FCost == currentNode.FCost
                    && openSet[i].HCost < currentNode.HCost)
                {
                    currentNode = openSet[i];
                }
            }
```

Jeżeli w ciągu działania aktualnie rozpatrywane pole jest polem docelowym, to za pomocą powiązania z rodzicem odtwarzana jest ścieżka i zwracana jest do gry.

```
if (currentNode == fieldTarget)
            {
                List<Field> path = new List<Field>();

                while (fieldTarget != fieldStart)
                {
                    path.Add(fieldTarget);
                   fieldTarget = fieldTarget.ParentField;
                }
                path.Reverse();
                return path;
            }
```

Cały proces rozpoczyna się w momencie, kiedy użytkownik kliknie myszą na pole gry. Wówczas realizowana jest poniższa metoda, która znajduje się w implementacji zachowania głównego gracza. Innymi słowy pole kliknięte zostaje określone jako pole docelowe do którego ma udać się główny bohater gry.

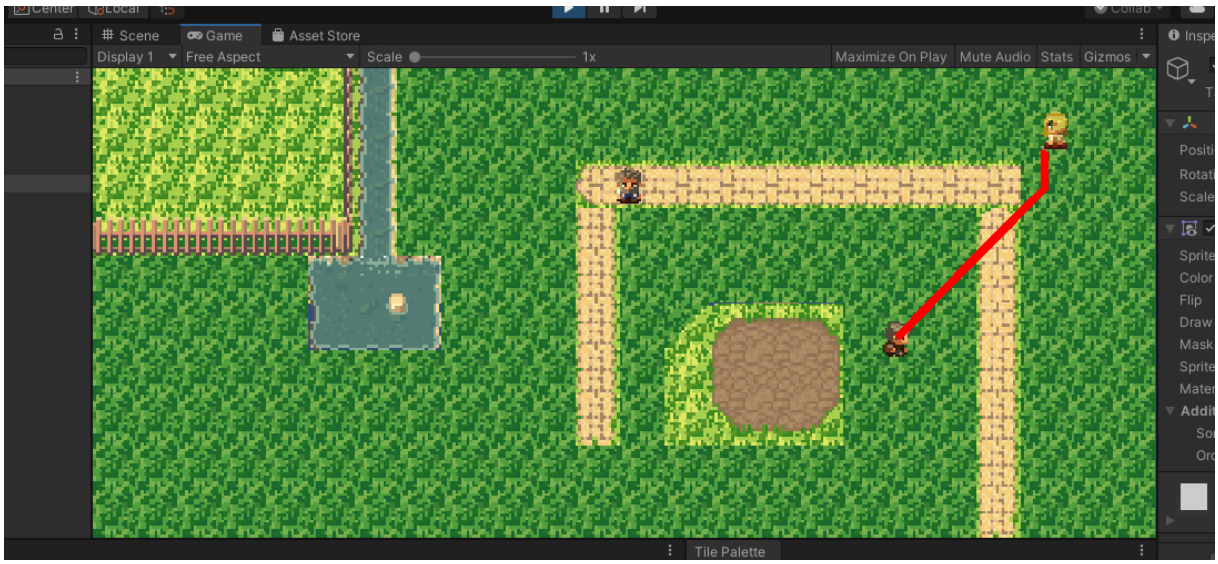
```
void OnGUI()
    {
        Event currentEvent = Event.current;
        Vector2 mousePos = new Vector2();
        Vector2 point = new Vector2();

        // compute where the mouse is in world space
        mousePos.x = currentEvent.mousePosition.x;
        mousePos.y = cam.pixelHeight - currentEvent.mousePosition.y;
        point = cam.ScreenToWorldPoint(new Vector3(mousePos.x, mousePos.y, 0.0
f));

        if (Input.GetMouseButtonDown(0))
        {
            target = point;

        }
    }
```

Następnie główny bohater zostaje płynnie przemieszczony przy wykorzystaniu poniższej metody *Update()*, zgodnie z cyklem życia klasy dziedziczącej po *MonoBeahaviour*.

W międzyczasie również za pomocą klasy *PathCalculator* oraz metody *CalculateRoadToPrincess* zwracanym jest lista punktów, które zostały wyszczególnione przez algorytm. Oprogramowanie dynamicznie rysuje linię co zostało zaprezentowane na rysunku. W praktyce metoda CalculateRoadToPrincess odpowiednio przekształca położenie gracza na osi XY na konkretne pole. Tak, że możliwym jest wykorzystanie opisywanej wyżej implementacji Algorytmu A*. Zatem po kliknięciu myszą główny bohater przemieszcza się oraz rysowana jest najkorzystniejsza ścieżka pomiędzy nim, a obiektem docelowym.

```
 void Update()
    {
        float step = speed * Time.deltaTime;

        // move sprite towards the target location
        transform.position = Vector2.MoveTowards(transform.position, target, s
tep);

        if (transform.position.x != target.x && transform.position.y != target
.y)
        {
            var lineRender = (LineRenderer)GameObject.Find("LineRenderer").Get
Component<LineRenderer>();
```

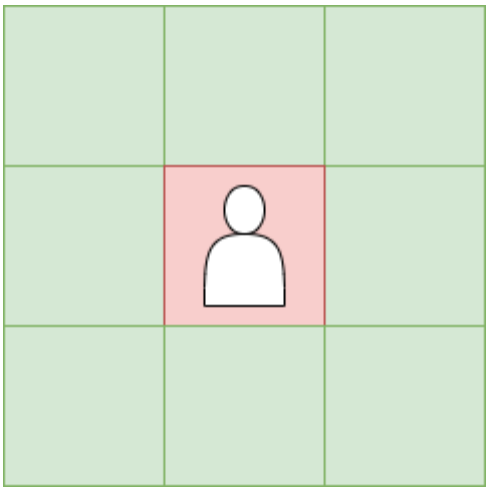
```
            lineRender.material = new Material(Shader.Find("Sprites/Default"))
;
            this.vectors = this.pathCalculator.CalculateRoadToPrincess(this.tr
ansform.position);
            lineRender.material.color = Color.red;
            lineRender.positionCount = this.vectors.Count;

            lineRender.SetPosition(0, transform.position);
            for (int i = 1; i < this.vectors.Count; i++)
            {
                lineRender.SetPosition(i, this.vectors[i]);
            }
        }
    }
```



Rysunek 2 Rysowanie ścieżki do punktu docelowego zależnie od pozycji gracza

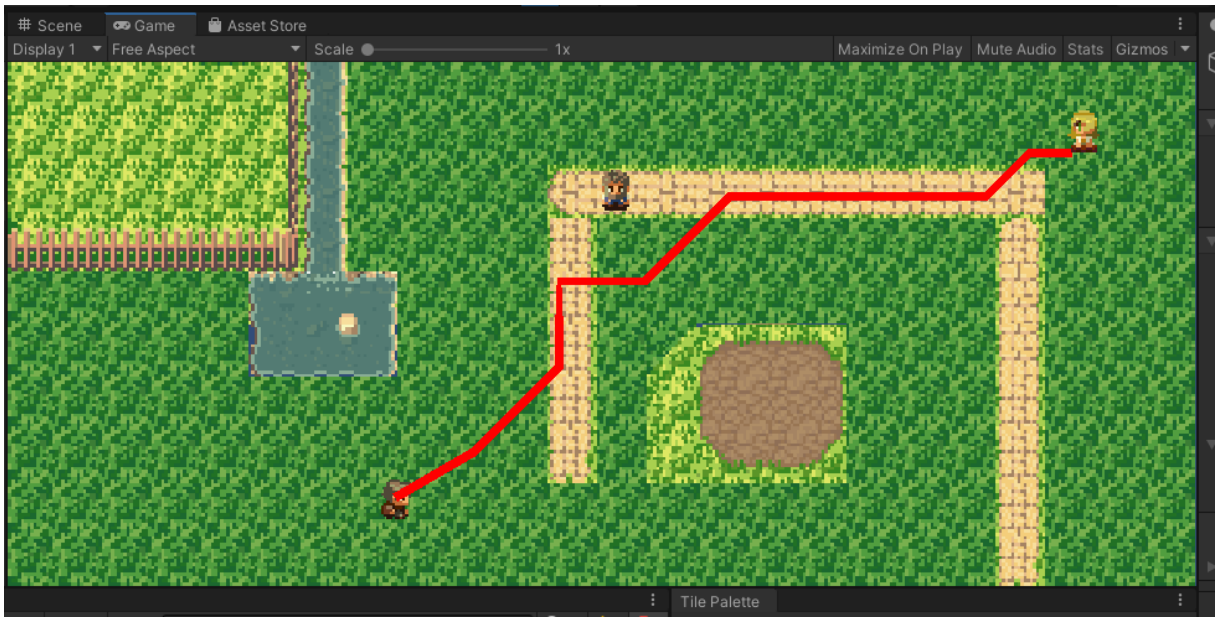
3.2. Znajdowanie ścieżki poza zasięgiem wroga

W niniejszej implementacji znajdowania ścieżki poza zasięgiem wroga zdecydowano się na modyfikację opisanego wcześniej algorytmu o nie uwzględnianie ścieżki, która będzie przebiegała przez pola znajdujące się w sąsiedztwie wroga. Idea sąsiedztwa została zaprezentowana na rysunku 3. W praktyce algorytm unika sprawdzania pól znajdujących się wokoło wroga jak i polu, na którym on się znajduje.



Rysunek 3 Sąsiedztwo obiektu jako jego zasięg

Efekt działania omijania wroga został zaprezentowany na rysunku 4.



Rysunek 4 Ścieżka poza zasięgiem wroga

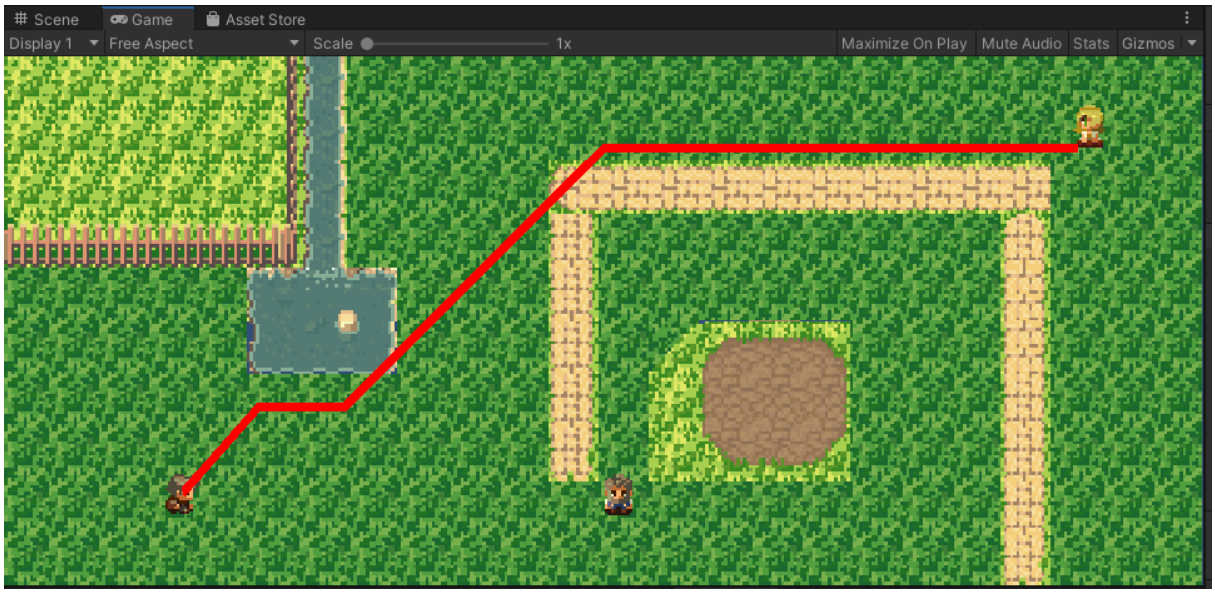
Funkcjonalność ta została zrealizowana przy wykorzystaniu poniższego kodu źródłowego. Po obliczeniu pól i utworzeniu obiektu świata gry, wartość poszczególnych pól podlega aktualizacji. Kod ten wprowadzono do osobnej metody, która początkowo znajduje wroga po nazwie obiektu. Następnie pozycja obiektu jest konwertowana na pole. Następnie z uwagi na świat gry dodawane są odpowiednie wartości (Unity oblicza położenie od środka, świat gry opisany w poprzedniej sekcji rozpoczyna położenie pól od lewego dolnego rogu). W kolejnej części realizowana jest metoda pobierająca sąsiadów danego elementu planszy.

Następnie za pomocą pętli ustawiane jest wartość pola poprzez wywołanie metody ustawiającą zmienną typu bool na polu.

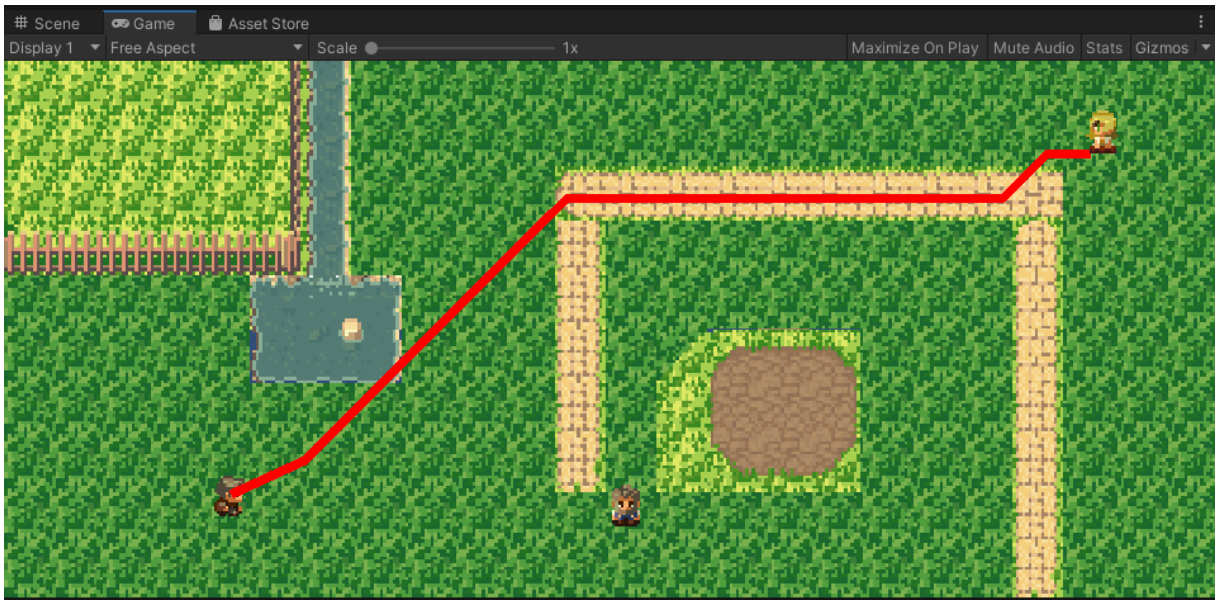
```
   public void AddEnemiesPosition()
        {
            var enemy = GameObject.Find("Enemy").GetComponent<SpriteRenderer>().bo
unds.center;
            var vs = grid.WorldToCell(enemy);
            string tile = $"{vs.x + 12} {vs.y + 5}";
            var enemytile = this.world.FindFieldByName(tile);
            var neighbours = this.world.GetNeighbours(enemytile);
            foreach (var item in neighbours)
            {
                this.world.FindFieldByName(item.TileName).SetIsAvailableToWalk(fal
se);
            }
        }
```

3.3. Wpływ podłoża na szybkość poruszania się

Do algorytmu A* opisanego w pierwszej sekcji dodano możliwość uwzględnienia wag danego elementu ścieżki. Dzięki temu algorytm traktuje konkretne pola na planszy jako bardziej korzystne do budowy ścieżki. Odbywa się to poprzez odpowiednie równanie generujące koszt trasy. W praktyce dzięki temu możliwym jest dodanie możliwości wartości podłoża, zatem w niniejszej grze podłożem, które daje lepszy rezultat będzie droga kosztem trawy. Na rysunkach 5 oraz 6 zaprezentowano przykład wygenerowania najlepszej ścieżki dla braku uwzględnienia drogi jako podłoża korzystniejszego oraz z uwzględnieniem. Jak można zauważyć algorytm dla zbliżonej lokalizacji bohatera obliczył dwie różne ścieżki.

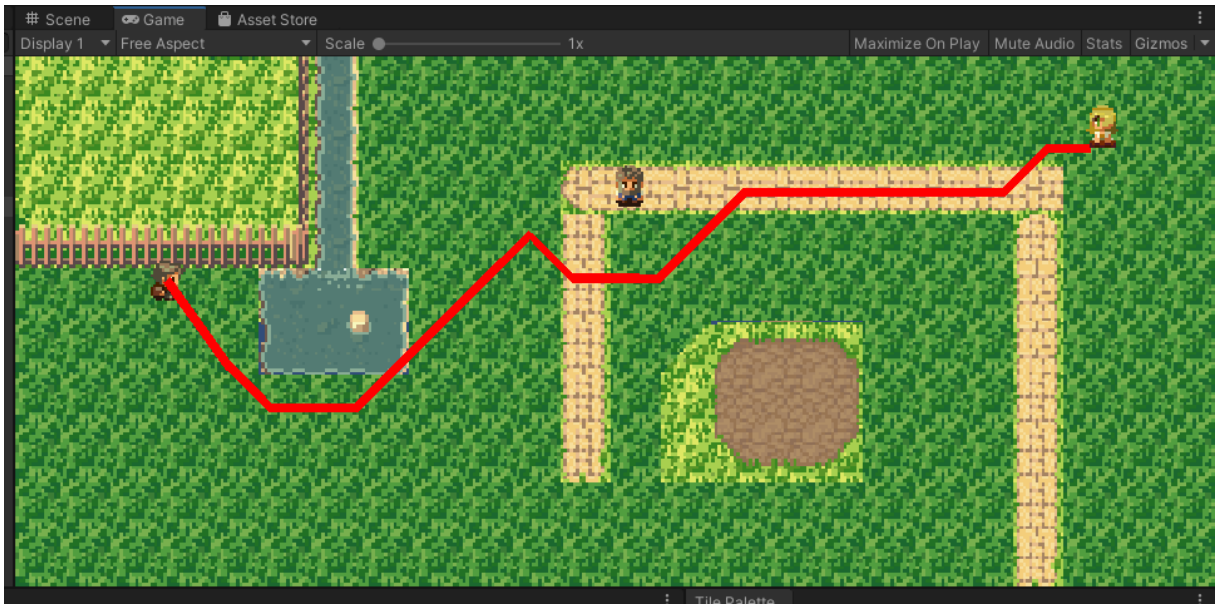


Rysunek 5 Brak uwzględnienia drogi jako korzystniejsze podłoże



Rysunek 6 Uwzględnienie drogi jako korzystniejsze podłoże

W projekcie zaimplementowano również możliwość generowania trasy, która omija wodę na planszy. Na rysunku 7 zaprezentowano położenie gracza w takim punkcie, w którym najkrótszą ścieżką byłaby tak przechodząca przez wodę. Jednakże z uwagi na reguły zaimplementowane w kodzie źródłowym ścieżka odpowiednio omija przeszkodę.



Rysunek 7 Uwzględnienie drogi i zasięgu wroga do ustalenia najkorzystniejszej ścieżki

W obu przypadkach wykorzystywana jest wartość wag, która jest dostarczana z wykorzystaniem poniższej klasy odpowiadającą za kalkulację wartości pola. Klasa ta zawiera zmienne stałe określające czy dane pole ma normalną wartość, większą czyli stanowi drogę na mapie lub jest niemożliwe do przejścia. Następnie za pomocą metody statycznej Calculate, przyjmującą jako parametry współrzędne pola, obliczane jest położenie danego pola względem mapy. W metodzie zadeklarowano reguły, które określają w jakim punkcie znajduje się woda oraz droga w grze. Dzięki temu metoda zwraca odpowiednią wartość, która jest pobierana w momencie generowania świata gry. Oczywiście w praktyce do sprawdzenia czy dane pole ma wpływ na poruszanie się gracza można wykorzystać inne reguły bazujące na chociażby nazwie tekstury.

```
public class FieldCalculator
{
    private const float NORMAL_FIELD = 1.0f;
    private const float ROAD_FIELD = 20.0f;
    private const float NOT_WALKABLE_FIELD = 0.0f;


    public static float Calculate(int x, int y)
    {
        if ((x>= 11 && x<=21) && (y==9) || ((x==12) && (y>=1 || y<=9))) return
 ROAD_FIELD;
```

```
        if ((x <5 && y>=6) || ((x>=4 && x<=6) && y>=4 && y<=6)) return NOT_WAL
KABLE_FIELD;

        return NORMAL_FIELD;
    }
}
```

W niniejszym projekcie wykorzystano heurystyczny algorytm A*, którego celem jest odnalezienie jak najkrótszej ścieżki. Testy pokazały, że algorytm ten może z sukcesem być wykorzystywany jako element sztucznej inteligencji. Algorytm zwróci informacje o ścieżce jeżeli ona w rzeczywistości istnieje. W niniejszych testach wykorzystano również modyfikację wag poszczególnych elementów trasy. Innymi słowy algorytm A* może być zmodyfikowany w taki sposób, że w ramach analizy przyległych pól do aktualnie analizowanego może danego pola nie uwzględniać, traktując je jako przeszkodę lub określając większą wagę dla pola aktualnie sprawdzanego. W ten sposób wykazano, że możliwym jest wprowadzenie wpływu podłoża na poruszanie się oraz uwzględnienie zasięgu wroga.

Oprócz samej implementacji algorytmu w przypadku realizacji niniejszego projektu problematycznym była obsługa odpowiednich zdarzeń w Unity. Zaprezentowana gra dwuwymiarowa zawiera podstawowe mechanizmy w zakresie poruszania się bohatera oraz rysowania ścieżki niemniej bazowanie na konkretnych elementach silnika Unity wymaga odpowiedniej wiedzy w zakresie obsługi obiektów. Szczególnie problematycznym było obliczanie położenia obiektu bohatera w stosunku do położenia podłoża. W ten sposób weryfikowano czy gracz znajduje się chociażby na ścieżce.

Zakończenie

Niniejsza praca opisuje zagadnienie z bardzo dużej, rozbudowanej oraz wszechstronnej dziedziny nauki jaką jest sztuczna inteligencja. W pracy został poruszony temat jednego z jej wielu zastosowań a mianowicie wykorzystywania sztucznej inteligencji w grach komputerowych. To właśnie informacje związane z wszelkimi, zróżnicowanymi obszarami wykorzystywania w grach znalazły się w pierwszym rozdziale. Opisanych zostało wiele elementów rozgrywki, za działanie których odpowiedzialna jest sztuczna inteligencja. Jednak jej poprawne wdrożenie wymaga wiele pracy oraz możliwe jest przy pomocy wielu metod. To właśnie informacje opisujące poszczególne metody sztucznej inteligencji zostały ujęte w kolejnej części rozdziału. Jednak w celu lepszego zobrazowania jej wykorzystywania przez twórców gier, przedstawionych zostało kilka gier, w których z wielkim sukcesem wdrożone zostały algorytmy sztucznej inteligencji. Opisane zostało działanie tych algorytmów, to jakie zachowania generowały oraz jak dobrze odwzorowywały inteligentne zachowania człowieka. Tytuły te były bardzo ważne w procesie rozwoju opisywanej w pracy dziedziny nauki w branży gier komputerowych. Po przybliżeniu wielu zróżnicowanych funkcjonalności sztucznej inteligencji w grach, autor skupił się na jednej konkretnej, a mianowicie na wyszukiwaniu drogi w grach. To właśnie ta tematyka została szerzej opisana w drugim rozdziale. Przedstawiony został cały proces odpowiedzialny za inteligentne poruszanie się postaci niezależnych w środowisku gry, począwszy od sposobów podziału tego środowiska a kończąc na algorytmach przeszukujących, odpowiedzialnych za znalezienie odpowiedniej trasy. Udało się przybliżyć zasady działania poszczególnych podziałów oraz to jak reprezentowana była mapa po ich dokonaniu. Aby lepiej zrozumieć działanie poszczególnych algorytmów, opisane zostało dokładnie w jaki sposób algorytmy przeszukują dostępne trasy lub czym kierują się podczas wyboru kolejnego potencjalnie najlepszego ruchu w celu zbadania czy faktycznie taki jest. W kolejnej części autor wskazał kilka autentycznych przykładów ukazujących jakie techniki wyszukiwania drogi zastosowali twórcy gier w swoich produkcjach. Nie było to łatwe zadanie, ponieważ studia tworzące gry rzadko dzielą się takimi informacjami.

W dwóch początkowych rozdziałach znajdowały się teoretyczne informacje z zakresu badanego tematu. W pierwszym rozdziale odnośnie ogólnego wykorzystywania sztucznej inteligencji w grach, a w drugim odnośnie procesu wyszukiwania drogi. Znajdowały się w nich przykłady zastosowań oraz opisy w jaki sposób teoretycznie działają algorytmy, jednak nie zostało to w żaden sposób ukazane w praktyce. Na tym właśnie zadaniu skupiony

był trzeci rozdział, na potrzeby którego został stworzony autorski projekt dwuwymiarowej gry, ukazującej jak w praktyce wygląda wdrożenie algorytmu wyszukiwania drogi w odpowiednio podzielonym środowisku gry. Autor przedstawił w jaki sposób można przedstawić świat gry i jak dostosować do tego wykorzystywany algorytm w celu znajdowania najkrótszej ścieżki, uwzględniając wszystkie przeszkody znajdujące się na mapie. W kolejnej części projektu przedstawione zostało w jaki sposób można modyfikować algorytm tak, aby spełniał funkcjonalności, które sobie zaplanowaliśmy. Ukazane zostało w jaki sposób można osiągnąć omijanie wroga, tak aby dotrzeć do celu nie przechodząc przez pola znajdujące się w jego zasięgu. Kolejną modyfikacją było określenie wpływu podłoża na szybkość poruszania się oraz uwzględnienie go w trakcie obliczania najkorzystniejszej trasy.

Cele określone na początku pracy zostały zrealizowane. Czytelnikowi przybliżony został temat sztucznej inteligencji w grach nie tylko pod względem teoretycznym ale także pod względem praktycznym, wykorzystując w tym celu stworzony przez autora na potrzeby pracy projekt gry. Jednak całe zagadnienie, którego część została omówiona w niniejszej pracy, jest niezwykle obszerne, w związku z tym praca stanowi tylko wprowadzenie w wielki świat sztucznej inteligencji.

Bibliografia

Abd Algfoor, Z., Sunar, M. S., i Kolivand, H. (2015). A comprehensive study on pathfinding techniques for robotics and video games. *International Journal of Computer Games Technology*, *2015*. https://doi.org/10.1155/2015/736138

Abercrombie, J., i Isla, D. (2016). *AI For Generated Worlds*. https://www.youtube.com/watch?v=BZh8dVSrulk

Anhalt, J., Kring, A., i Sturtevant, N. (2011). *AI Navigation: It's Not a Solved Problem - Yet*. https://www.gdcvault.com/play/1014514/AI-Navigation-It-s-Not

Argasiński, J. (2012). *Sztuczna Inteligencja w grach wideo*. https://ruj.uj.edu.pl/xmlui/bitstream/handle/item/55252/argasinski_sztuczna_inteligencja_w_grach_wideo_2012.pdf?sequence=1&isAllowed=y

Aske. (2014). *Supernauts – nowa konkurencja dla Minecrafta?* https://www.tabletowo.pl/supernauts-nowa-konkurencja-dla-minecrafta/

Cao, L., Liu, J., i Tang, X. (2006). Degen generalized cylinders and their properties. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3951 LNCS*(June 2014), 83–94. https://doi.org/10.1007/11744023_7

Cassingena Navone, E. (2020). *Dijkstra's Shortest Path Algorithm - A Detailed and Visual Introduction*. https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/

Champandard, A. J. (2007). *Top 10 Most Influential AI Games*. http://www.effecthub.com/topic/207

Cui, X., i Shi, H. (2011). A * -based Pathfinding in Modern Computer Games. *International Journal of Computer Science and Network Security*, *11*(1), 125–130.

Dias, J., Mascarenhas, S., i Paiva, A. (2014). FAtiMA modular: Towards an agent architecture with a generic appraisal framework. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8750*, 44–56. https://doi.org/10.1007/978-3-319-12973-0_3

Drabik, E. (2018). *Zastosowanie teorii gier w tworzeniu sztucznej inteligencji*. Oficyna Wydawnicza Politechniki Warszawskiej.

Duarte, F. F., Lau, N., Pereira, A., i Reis, L. P. (2020). A survey of planning and learning in games. *Applied Sciences (Switzerland)*, *10*(13), 1–55. https://doi.org/10.3390/app10134529

Dudek, G. (2014). *Sztuczna inteligencja wykład 12. Przeszukiwanie*. https://gdudek.el.pcz.pl/files/SI/SI_wyklad12.pdf

Dudzic, M. (2009). *Typy sztucznych sieci neuronowych używanych jako narzędzie prognozy w procesach zarządzania*. http://www.pszw.edu.pl/images/publikacje/t021_pszw_2009_dudzic_-_typy_sztucznych_sieci_neuronowych_uzywanych_jako_narzedzie_prognozy_w_procesach_zarzadzania.pdf

E Shummon Maass, L. (2019). *Artificial Intelligence in Video Games*. https://towardsdatascience.com/artificial-intelligence-in-video-games-3e2566d59c22

Gault, M. (2020). *How Artificial Intelligence Could Help Video Gamers Create the Exact Games They Want to Play*. https://time.com/5779217/artificial-intelligence-video-games/

Graham, R., Mccabe, H., i Sheridan, S. (2003). Pathfinding in computer games. *ITB Journal*, *4*(8), 57–81. https://doi.org/10.21427/D7ZQ9J

Hagelbäck, J. (2016). Hybrid Pathfinding in StarCraft. *IEEE Transactions on Computational Intelligence and AI in Games*, *8*(4), 319–324. https://doi.org/10.1109/TCIAIG.2015.2414447

Hagelbäck, J., i Johansson, S. J. (2008). Dealing with fog of war in a real time strategy game environment. *2008 IEEE Symposium on Computational Intelligence and Games, CIG 2008*, *08*, 55–62. https://doi.org/10.1109/CIG.2008.5035621

Hatinen, H. (2014). *Advanced Real-time Pathfind in Dynamic Environment in Supernauts*. https://drive.google.com/file/d/1b30cx62uwYH7o3FOoXELR7iuVgDXuzuO/view

Horti, S. (2017). *Why F.E.A.R.'s AI is still the best in first-person shooters*. https://www.rockpapershotgun.com/why-fears-ai-is-still-the-best-in-first-person-shooters

Jagga, S. (2020). *Uniform Cost Search*. https://www.educba.com/uniform-cost-search/

Kobiałka, D. (2019). *5 najlepszych programów do tworzenia gier*. https://www.komputerswiat.pl/artykuly/redakcyjne/5-najlepszych-programow-do-tworzenia-gier/9g5xh3v

Kołodziejczyk, J. (2017). *Programowanie gier. Wykład 4. Pathfinder*. https://wikizmsi.zut.edu.pl/uploads/4/44/PG_W3.pdf

Kulbiej, E. (2018). Autonomous Vessels' Pathfinding Using Visibility Graph. *Proceedings - 2018 Baltic Geodetic Congress, BGC-Geomatics 2018*, *June 2018*, 107–111. https://doi.org/10.1109/BGC-Geomatics.2018.00026

Kuźmińska-Sołśnia, B., i Siwiec, T. (2015). *Zastosowanie algorytmów sztucznej inteligencji*

w grach komputerowych. http://www.bks.pr.radom.pl/publikacje/SI w grach.pdf

Lara-Cabrera, R., Nogueira-Collazo, M., Cotta, C., i Fernández-Leiva, A. J. (2015). Game artificial intelligence: Challenges for the scientific community. *CEUR Workshop Proceedings*, *1394*(January), 1–12.

Lochiatto, J. (2020). *5 Genre-Defining Horror Games You Don't Remember*. https://www.cbr.com/genre-defining-horror-video-games-underrated/

Marecki, J. (2001). *Metody sztucznej inteligencji*. Wyższa Szkoła Informatyki i Zarządzania.

Millington, I., i Funge, J. (2009). *Artificial Intelligence for Games* (Second Edition). CRC Press.

Pluta, E. (2020). *Sekretne życie sieci neuronowych. Sztuczna inteligencja a neuronauka*. https://www.swps.pl/strefa-psyche/blog/relacje/22346-sekretne-zycie-sieci-neuronowych-sztuczna-inteligencja-a-neuronauka?dt=1620063689814

Reeves, B., i Nass, C. (1996). *The media equation: How people treat computers, television, and new media like real people and places*. https://www.academia.edu/2585264/The_media_equation_How_people_treat_computers_television_and_new_media_like_real_people_and_places

Rutkowski, L. (2006). *Metody i techniki sztucznej inteligencji*. Wydawnictwo Naukowe PWN.

Sanocki, Ł., i Gołda, A. (2005). *Wstęp do sieci neuronowych*. Katedra Elektroniki AGH. http://galaxy.uci.agh.edu.pl/~vlsi/AI/wstep/

Schreiner, T. (2009). *Artificial Intelligence in Game Design*. https://web.archive.org/web/20110810144013/http://ai-depot.com/GameAI/Design.html

Statt, N. (2019). *How Artificial Intelligence Will Revolutionize the Way Video Games Are Developed and Played*. https://www.theverge.com/2019/3/6/18222203/video-game-ai-future-procedural-generation-deep-learning

Stewart, J., Lizzy, I., All, A., Mariën, I., Schurmans, D., Looy, V., Jacobs, A., Willaert, K., i Grove, F. De. (2013). *Digital Games for Empowerment and Inclusion of Groups at Risk of Social and Economic Exclusion: Evidence and Opportunity for Policy*. https://doi.org/10.2791/88148

Stout, B. (1999). *Smart Move: Intelligent Path-Finding*. https://www.gamasutra.com/view/feature/131724/smart_move_intelligent_.php

Verma, S., Mehta, P., Shah, H., Shukla, S., Professor, A., Nmims, M., Sturtevant, N. R., Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., i Thrun, S. (2015). A Review on

Algorithms for Pathfinding in Computer Games. *IEEE Transactions on Computational Intelligence and AI in Games*, *March 2015*, 262–271.

Wawrzyński, P. (2014). *Podstawy sztucznej inteligencji*. Oficyna Wydawnicza Politechniki Warszawskiej.

Westera, W., Prada, R., Mascarenhas, S., Santos, P. A., Dias, J., Guimarães, M., Georgiadis, K., Nyamsuren, E., Bahreini, K., Yumak, Z., Christyowidiasmoro, C., Dascalu, M., Gutu-Robu, G., i Ruseti, S. (2020). Artificial intelligence moving serious gaming: Presenting reusable game AI components. *Education and Information Technologies*, *25*(1), 351–380. https://doi.org/10.1007/s10639-019-09968-2

Winiarski, M. (2016). *Historia sztucznej inteligencji w grach komputerowych*. http://mwin.pl/historia-sztucznej-inteligencji-grach-komputerowych/

Winiarski, M. (2018). *Jak powstaje sztuczna inteligencja w grach komputerowych*. https://mwin.pl/jak-powstaje-sztuczna-inteligencja-w-grach-komputerowych/

Wu, G. (2021). *Comparison Between Uniform-Cost Search and Dijkstra's Algorithm*. https://www.baeldung.com/cs/uniform-cost-search-vs-dijkstras

Yannakakis, G. N. (2012). Game AI revisited. *CF '12 - Proceedings of the ACM Computing Frontiers Conference*, 285–292. https://doi.org/10.1145/2212908.2212954

Zabłocki, M. (2018). *Historia sztucznej inteligencji w grach komputerowych*. https://pclab.pl/art76450.html

Spis ilustracji