Co to jest AI?

Sztuczna inteligencja polega na umożliwieniu komputerom wykonywania zadań myślowych, do których zdolni są ludzie i zwierzęta. Możemy już zaprogramować komputery, aby miały nadludzkie zdolności rozwiązywania wielu problemów: arytmetyki, sortowania, wyszukiwania i tak dalej. Możemy nawet zmusić komputery do grania w niektóre gry planszowe lepiej niż jakikolwiek człowiek (na przykład Reversi lub Connect 4). Wiele z tych problemów pierwotnie uważano za problemy AI, ale ponieważ zostały one rozwiązane w bardziej wszechstronny sposób, wymknęły się one z domeny programistów AI. Ale jest wiele rzeczy, w których komputery nie są dobre, w których uważamy, że są trywialne: rozpoznawanie znajomych twarzy, mówienie własnym językiem, podejmowanie decyzji, co robić dalej i kreatywność. Oto domena sztucznej inteligencji: próba ustalenia, jakie rodzaje algorytmów są potrzebne do wyświetlenia tych właściwości. W środowisku akademickim niektórzy badacze AI są motywowani filozofią: rozumieniem natury myśli i natury inteligencji oraz budowaniem oprogramowania do modelowania sposobu działania myślenia. Niektóre są motywowane przez psychologię: zrozumienie mechaniki ludzkiego mózgu i procesów umysłowych. Inne motywowane są inżynierią: budowaniem algorytmów do wykonywania czynności podobnych do ludzkich. To potrójne rozróżnienie leży u podstaw akademickiej sztucznej inteligencji, a różne nastawienia są odpowiedzialne za różne subpozycje przedmiotu. Jako twórców gier interesuje nas przede wszystkim inżynieria: budowanie algorytmów, dzięki którym postacie w grze wyglądają jak ludzie lub zwierzęta. Deweloperzy zawsze czerpali z badań akademickich, które pomagają im w wykonywaniu pracy. Warto szybko zapoznać się z pracą AI wykonaną w środowisku akademickim, aby zorientować się, co istnieje w temacie i co może być warte plagiatowania. Nie mamy miejsca (ani zainteresowania i cierpliwości), aby dać pełny spacer - poprzez sztuczną inteligencję akademicką, ale pomocne będzie przyjrzenie się, jakie techniki kończą się w grach.

<u>Akademicka AI</u>

Ogólnie rzecz biorąc, akademicką sztuczną inteligencję można podzielić na trzy okresy: wczesne dni, era symboliczna i współczesna. Jest to oczywiście znaczne uproszczenie, a te trzy w pewnym stopniu się pokrywają, ale uważamy, że jest to użyteczne rozróżnienie.

Wczesne dni

Do pierwszych dni należą czasy przed komputerami, w których filozofia umysłu czasami robiła wyprawy do sztucznej inteligencji z takimi pytaniami, jak: „Co wytwarza myśl?" „Czy możesz ożywić przedmiot nieożywiony?" „Jaka jest różnica między zwłokami a człowiekiem, jakim był wcześniej?" Styczną z tym był popularny smak robotów mechanicznych, szczególnie w wiktoriańskiej Europie. Na przełomie XIX i XX wieku powstały modele mechaniczne, które pokazały takie animowane zachowania przypominające zwierzęta, które obecnie zatrudniamy w pakiecie do modelowania. W wysiłkach wojennych lat czterdziestych potrzeba przełamania kodów wroga i wykonania obliczeń niezbędnych do działań wojennych spowodowała rozwój pierwszych programowalnych komputerów. Biorąc pod uwagę, że maszyny te były używane do wykonywania obliczeń, które w innym przypadku byłyby wykonywane przez osobę, programiści byli zainteresowani sztuczną inteligencją. Kilku pionierów komputerowych (takich jak Turing, vonNeumann i Shannon) byli również pionierami wczesnej sztucznej inteligencji. W szczególności Turing stał się adoptowanym ojcem w tej dziedzinie, w wyniku pracy filozoficznej opublikowanej w 1950 r.

Era symboliczna

Od końca lat 50. do wczesnych lat 80. głównym kierunkiem badań nad sztuczną inteligencją były systemy „symboliczne". System symboliczny to taki, w którym algorytm jest podzielony na dwa

elementy: zbiór wiedzy (reprezentowany jako symbole, takie jak słowa, liczby, zdania lub obrazy) oraz algorytm rozumowania, który manipuluje tymi symbolami, aby stworzyć nowe kombinacje symboli, które, miejmy nadzieję, reprezentują rozwiązania problemów lub nową wiedzę. System ekspercki, jeden z najczystszych przejawów tego podejścia, jest najbardziej znaną techniką sztucznej inteligencji. Ma dużą bazę wiedzy i stosuje zasady do wiedzy w celu odkrywania nowych rzeczy. Inne symboliczne podejścia mające zastosowanie do gier obejmują architektury tablic, wyszukiwanie ścieżek, drzewa decyzyjne, automaty stanów i algorytmy sterowania. Wspólną cechą systemów symbolicznych jest kompromis: przy rozwiązywaniu problemu im więcej posiadasz wiedzy, tym mniej pracy musisz wykonać przy rozumowaniu. Często algorytmy wnioskowania polegają na wyszukiwaniu: próbowaniu różnych możliwości uzyskania najlepszego rezultatu. To prowadzi nas do złotej zasady sztucznej inteligencji: wyszukiwanie i wiedza są ze sobą nierozerwalnie związane. Im więcej masz wiedzy, tym mniej potrzebujesz odpowiedzi; im więcej możesz przeprowadzić wyszukiwania (tzn. im szybciej możesz wyszukiwać), tym mniej wiedzy potrzebujesz. Naukowcy Newell i Simon zasugerowali w 1976 r., że właśnie w ten sposób powstają wszystkie inteligentne zachowania. Niestety, pomimo wielu solidnych i ważnych cech, teoria ta została w dużej mierze zdyskredytowana. Wiele osób z niedawnym wykształceniem w zakresie sztucznej inteligencji nie zdaje sobie sprawy, że jako kompromis inżynieryjny wiedza kontra wyszukiwanie jest nieunikniona. Ostatnie prace nad matematyką rozwiązywania problemów udowodniły to teoretycznie, a inżynierowie AI zawsze to znali.

Era współczesna

Stopniowo w latach 80. i na początku lat 90. XX wieku narastała frustracja związana z symbolicznym podejściem. Frustracja pochodziła z różnych stron. Z technicznego punktu widzenia wczesne sukcesy w prostych problemach nie zdawały się rozszerzać na trudniejsze problemy ani radzić sobie z niepewnością i złożonością prawdziwego świata. Wydawało się, że łatwo jest rozwijać sztuczną inteligencję, która rozumie (lub wydaje się, że rozumie) proste zdania, ale rozwijanie zrozumienia pełnego ludzkiego języka nie wydawało się bliższe. Był też wpływowy argument filozoficzny, który sprawił, że podejście symboliczne nie było biologicznie wiarygodne. Zwolennicy argumentowali, że nie można zrozumieć, w jaki sposób człowiek planuje trasę za pomocą symbolicznego algorytmu planowania trasy, tak samo jak nie można zrozumieć, jak działają ludzkie mięśnie, badając wózek widłowy. Efektem było przejście do naturalnego przetwarzania danych: technik inspirowanych biologią lub innymi systemami naturalnymi. Techniki te obejmują sieci neuronowe, algorytmy genetyczne i symulowane wyżarzanie. Warto jednak zauważyć, że niektóre techniki, które stały się modne w latach 80. i 90. XX wieku, zostały wynalezione znacznie wcześniej. Na przykład sieci neuronowe poprzedzają erę symboliczną; po raz pierwszy zasugerowano je w 1943 r. Niestety obiektywne wykonanie niektórych z tych technik nigdy nie pasowało do ewangelizacyjnej retoryki ich najbardziej zagorzałych zwolenników. Stopniowo badacze AI z głównego nurtu zdali sobie sprawę, że kluczowym składnikiem tego nowego podejścia było nie tyle połączenie ze światem przyrody, ale umiejętność radzenia sobie z niepewnością i wagę, jaką przykłada do rozwiązywania rzeczywistych problemów. Zrozumieli, że techniki takie jak sieci neuronowe można wyjaśnić matematycznie za pomocą rygorystycznych ram probablistycznych i statystycznych. Wolny od konieczności jakiejkolwiek naturalnej interpretacji, szkielet probablistyczny mógłby zostać rozszerzony, aby znaleźć rdzeń współczesnej statystycznej sztucznej inteligencji, która obejmuje sieci Bayesa, maszyny wektora wspomagającego (SVM) i procesy Gaussa.

Inżynieria

Całkowita zmiana w akademickiej sztucznej inteligencji to coś więcej niż moda. Dzięki temu sztuczna inteligencja stała się kluczową technologią istotną przy rozwiązywaniu rzeczywistych problemów. Na przykład technologia wyszukiwania Google opiera się na nowym podejściu do sztucznej inteligencji. To

nie przypadek, że Peter Norvig jest zarówno dyrektorem ds. badań Google, jak i współautorem (wraz ze swoim byłym doradcą, profesorem Stuartem Russellem) kanonicznego odniesienia do współczesnej sztucznej inteligencji akademickiej. Niestety przez pewien czas istniała tendencja do wyrzucania dziecka z kąpielą i wiele osób kupowało szum, że symboliczne podejścia nie żyją. Rzeczywistość praktycznego zastosowania sztucznej inteligencji polega na tym, że nie ma darmowego lunchu, a późniejsza praca wykazała, że żadne pojedyncze podejście nie jest lepsze niż jakiekolwiek inne. Jedynym sposobem, w jaki dowolny algorytm może przewyższyć inne, jest skupienie się na określonym zestawie problemów. Im węższa dziedzina problemowa, na której się skupisz, tym łatwiej będzie algorytmowi zabłysnąć - co w okrągły sposób przywraca nam złotą zasadę sztucznej inteligencji: wyszukiwanie (wypróbowanie możliwych rozwiązań) to druga strona medalu do wiedzy (wiedza na temat problemu jest odpowiednikiem zawężenia liczby problemów, do których odnosi się twoje podejście). Obecnie niektórzy z najlepszych statystycznych badaczy AI podejmują wspólne wysiłki, aby stworzyć ujednolicone ramy dla obliczeń symbolicznych i probabilistycznych. Ważne jest również, aby zdawać sobie sprawę, że w inżynieryjnych zastosowaniach obliczeń statystycznych zawsze stosuje się technologię symboliczną. Na przykład program do rozpoznawania głosu konwertuje sygnały wejściowe przy użyciu znanych wzorów na format, w którym sieć neuronowa może je dekodować. Wyniki są następnie przekazywane przez szereg algorytmów symbolicznych, które patrzą na słowa ze słownika i sposób łączenia słów w języku. Algorytm stochastyczny optymalizujący kolejność linii produkcyjnej będzie zawierał reguły dotyczące produkcji zakodowane w jej strukturze, więc nie może sugerować nielegalnego harmonogramu: wiedza jest wykorzystywana do zmniejszenia wymaganej liczby wyszukiwań. W tej książce przyjrzymy się kilku technikom statystyki statystycznej, przydatnym przy określonych problemach. Mamy wystarczające doświadczenie, aby wiedzieć, że w przypadku gier są one często niepotrzebne: ten sam efekt można często osiągnąć lepiej, szybciej i przy większej kontroli dzięki prostszemu podejściu. Choć się zmienia, w przeważającej mierze sztuczna inteligencja używana w grach jest nadal technologią symboliczną.

Gry AI

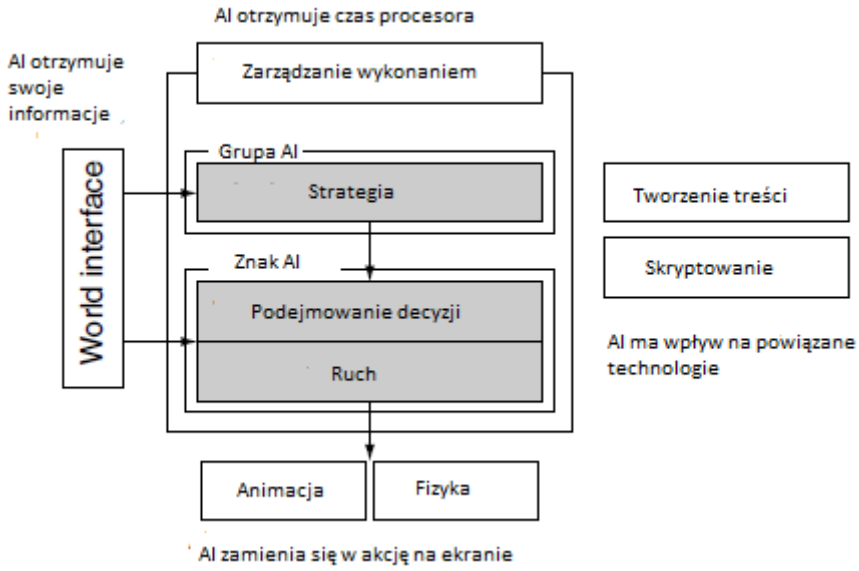
Pac-Man [Midway Games West, Inc., 1979] był pierwszą grą, jaką wielu ludzi pamięta, grając z raczkującą sztuczną inteligencją. Do tego momentu istniały klony Ponga z nietoperzami kontrolowanymi przez przeciwników (które zasadniczo podążały za piłką w górę i w dół) i niezliczonymi strzelcami w formie Space Invaders. Ale Pac-Man miał zdecydowane wrogie postacie, które wydawały się spiskować przeciwko tobie, poruszały się na tym samym poziomie, co ty, i utrudniały życie. Pac-Man polegał na bardzo prostej technice sztucznej inteligencji: maszynie stanów (którą omówimy w dalszej części). Każdy z czterech potworów (zwanych później duchami po katastrofalnie migoczącym porcie na Atari 2600) albo cię ścigał, albo uciekał. Dla każdego stanu wybrali pół losową trasę na każdym skrzyżowaniu. W trybie pościgowym każda z nich miała inną szansę ścigania gracza lub wyboru losowego kierunku. W trybie ucieczki albo uciekli, albo wybrali losowy kierunek. Wszystko bardzo proste i bardzo z 1979 roku. Sztuczna inteligencja w grze nie zmieniła się aż do połowy lat 90. Większość wcześniej sterowanych komputerowo postaci była mniej więcej tak wyrafinowana jak duch Pac-Mana. Weź klasykę taką jak Golden Axe [SEGA Entertainment, Inc., 1987] osiem lat później. Wrogie postacie stały w bezruchu (lub cofały się na krótką odległość), aż gracz zbliżył się do nich, po czym zasiedliły gracza. Golden Axe miał ciekawą innowację z wrogami, którzy pędzili obok gracza, a następnie przełączali się na tryb bazowania, atakując od tyłu. Wyrafinowanie AI to tylko mały krok od Pac-Mana. W połowie lat 90. sztuczna inteligencja stała się punktem sprzedaży gier. Gry takie jak Beneath a Steel Sky [Revolution Software Ltd., 1994] nawet wspomniały o sztucznej inteligencji z tyłu pudełka. Niestety, bardzo rozbudowany system sztucznej inteligencji „Virtual Theater" po prostu pozwalał bohaterom chodzić do tyłu i do przodu w grze - prawie nie jest to postęp. Goldeneye 007 [Rare Ltd., 1997] prawdopodobnie zrobił najwięcej, aby pokazać graczom, co AI może zrobić, aby poprawić

rozgrywkę. Wciąż polegając na postaciach z małą liczbą dobrze zdefiniowanych stanów, Goldeneye dodał system symulacji zmysłu: postacie widziały swoich kolegów i zauważyliby, że zostaną zabici. Symulacja zmysłu jako temat chwili, w której Thief: The Dark Project [Looking Glass Studios, Inc., 1998] i Metal Gear Solid [Konami Corporation, 1998] opierają cały projekt na tej technice. W połowie lat 90. zaczęły się też pojawiać gry strategiczne czasu rzeczywistego (RTS). Warcraft [Blizzard Entertainment, 1994] był jednym z pierwszych przypadków, gdy wyszukiwanie ścieżek było szeroko zauważane w akcji (faktycznie było używane kilka razy wcześniej). Badacze AI pracowali z emocjonalnymi modelami żołnierzy w symulacji wojskowego pola bitwy w 1998 roku, kiedy zobaczyli Warhammer: Dark Omen [Mindscape, 1998] robiąc to samo. Był to także jeden z pierwszych przypadków, gdy ludzie zobaczyli silny ruch formacyjny w akcji. Ostatnio rosnąca liczba gier uczyniła AI punktem gry. Creatures [Cyberlife Technology Ltd., 1997] zrobiły to w 1997 r., Ale gry takie jak The Sims [Maxis Software, Inc., 2000] i Black and White [Lionhead Studios Ltd., 2001] nadal działają na pochodnię. Stworzenia wciąż mają jeden z najbardziej złożonych systemów AI w grze, z mózgiem opartym na sieci neuronowej dla każdego stworzenia (co prawda często może wyglądać głupio w działaniu). Teraz mamy ogromną różnorodność AI w grach. Wiele gatunków wciąż korzysta z prostej sztucznej inteligencji z 1979 roku, ponieważ to wszystko, czego potrzebują. Boty w strzelankach FPS cieszyły się większym zainteresowaniem ze strony akademickiej sztucznej inteligencji niż jakikolwiek inny gatunek. Gry RTS kooptowały wiele AI używanych do budowy symulatorów szkoleniowych dla wojska (do tego stopnia, że Full SpectrumWarrior [Pandemic Studios, 2004] rozpoczął życie jako symulator szkolenia wojskowego). W szczególności gry sportowe i gry kierownicze mają własne wyzwania AI, z których niektóre pozostają w dużej mierze nierozwiązane (na przykład dynamiczne obliczanie najszybszej drogi na torze wyścigowym), podczas gdy gry RPG ze złożonymi interakcjami postaci są nadal realizowane jako konwersacja drzewa spóźniają się na lepszą AI. W wielu wykładach i artykułach w ciągu ostatnich pięciu lub sześciu lat wprowadzono brzydkie ulepszenia, które nie pojawiły się jeszcze w grach produkcyjnych. Sztuczna inteligencja w większości współczesnych gier zaspokaja trzy podstawowe potrzeby: zdolność do przemieszczania postaci, zdolność do decydowania o tym, gdzie się przenieść, oraz zdolność do myślenia taktycznego lub strategicznego. Mimo, że przeszliśmy od korzystania ze sztucznej inteligencji opartej na stanie wszędzie (nadal są one używane w większości miejsc) do szerokiej gamy technik, wszystkie one spełniają te same trzy podstawowe wymagania.

Model Gry AI

Wykorzystamy rozległe zoo technik. Łatwo byłoby się zgubić, dlatego ważne jest, aby zrozumieć, jak pasują do siebie bity.

Aby pomóc, zastosowaliśmy spójną strukturę, aby zrozumieć sztuczną inteligencję używaną w grze. To nie jest jedyny możliwy model i nie jest to jedyny model, który skorzystałby z technik opisanych w tej książce. Aby jednak wyjaśnić dyskusje, pomyślimy, że każda technika pasuje do ogólnej struktury tworzenia inteligentnych postaci w grze. Rysunek poniżej ilustruje ten model.



Dzieli zadanie AI na trzy sekcje: ruch, podejmowanie decyzji i strategię. Dwie pierwsze sekcje zawierają algorytmy, które działają na zasadzie znak po znaku, a ostatnia sekcja działa dla całego zespołu lub drużyny. Wokół tych trzech elementów AI znajduje się cały zestaw dodatkowej infrastruktury.

Nie wszystkie aplikacje do gier wymagają wszystkich poziomów AI. Gry planszowe, takie jak Szachy lub Ryzyko, wymagają jedynie poziomu strategii; postacie w grze (nawet jeśli można je tak nazwać) nie podejmują własnych decyzji i nie muszą się martwić, jak się poruszać. Z drugiej strony w wielu grach nie ma żadnej strategii. Postacie w grze platformowej, takiej jak Jak i Daxter, lub pierwsza gra Oddworld [Oddworld Inhabitants, Inc., 1997] są całkowicie reaktywne, podejmując własne proste decyzje i działając na ich podstawie. Brak koordynacji, która zapewni, że wrogie postacie wykonają najlepszą robotę udaremniania gracza.

Ruch

Ruch odnosi się do algorytmów, które zamieniają decyzje w pewien rodzaj ruchu. Gdy postać wroga bez broni musi zaatakować gracza w grze Super Mario Sunshine [Nintendo Entertainment, Analysis and Development, 2002], najpierw kieruje się bezpośrednio w stronę gracza. Gdy jest wystarczająco blisko, może wykonać atak. Decyzja o ataku jest podejmowana przez zestaw algorytmów ruchu, które znajdują się w lokalizacji gracza. Dopiero wtedy można odtworzyć animację ataku i wyczerpać zdrowie gracza. Algorytmy poruszania się mogą być bardziej złożone niż po prostu zasiedlanie domu. Postać może potrzebować omijać przeszkody na drodze lub nawet przedzierać się przez szereg pomieszczeń. Strażnik na niektórych poziomach Splinter Cell [Ubisoft Montreal Studios, 2002] zareaguje na pojawienie się gracza, podnosząc alarm. Może to wymagać nawigacji do najbliższego zamontowanego na ścianie punktu alarmowego, który może znajdować się w dużej odległości i może obejmować złożoną nawigację wokół przeszkód lub korytarzy. Wiele działań jest przeprowadzanych przy użyciu animacji bezpośrednio. Jeśli Sim, w The Sims, siedzi przy stole z jedzeniem przed sobą i chce przeprowadzić akcję jedzenia, to po prostu odtwarzana jest animacja jedzenia. Gdy AI zdecyduje, że postać powinna jeść, nie jest już potrzebna (sztuczna technologia animacji nie jest opisana w tej książce). Jeśli ta sama postać jest przy tylnych drzwiach, kiedy chce jeść, ruch AI musi poprowadzić go do krzesła (lub innego pobliskiego źródła jedzenia).

Podejmowanie decyzji

Podejmowanie decyzji polega na tym, że postać zastanawia się, co dalej. Zazwyczaj każda postać ma szereg różnych zachowań, które może wybrać: atakować, stać w miejscu, ukrywać się, badać, patrolować i tak dalej. System decyzyjny musi ustalić, które z tych zachowań jest najbardziej odpowiednie w każdym momencie gry. Wybrane zachowanie można następnie wykonać za pomocą sztucznej inteligencji ruchu i technologii animacji. W najprostszym przypadku postać może mieć bardzo proste zasady wyboru zachowania. Zwierzęta hodowlane na różnych poziomach gier Zelda będą stać w miejscu, chyba że gracz zbliży się zbyt blisko, po czym odejdą na niewielką odległość. Z drugiej strony, wrogowie w Half-Life 2 [Valve, 2004] wykazują złożone podejmowanie decyzji, w ramach których próbują dotrzeć do gracza na wiele różnych sposobów: łącząc razem pośrednie działania takie jak rzucanie granatami i stawianie ognia tłumiącego w celu osiągnięcia ich celów. Niektóre decyzje mogą wymagać AI ruchu w celu ich wykonania. Atak wręcz wymaga od postaci zbliżenia się do ofiary. Inne są obsługiwane wyłącznie przez animację (na przykład Sim jedzący) lub po prostu przez aktualizację stanu gry bezpośrednio, bez żadnego wizualnego sprzężenia zwrotnego (gdy krajowa AI w SidMeier's Civilization III [Firaxis Games, 2001] decyduje się na badanie nowej technologia, na przykład, po prostu dzieje się bez wizualnego sprzężenia zwrotnego).

Strategia

Możesz przejść długą drogę od ruchu AI do podejmowania decyzji AI, a większość trójwymiarowych gier akcji (3D) wykorzystuje tylko te dwa elementy. Aby jednak koordynować cały zespół, wymagana jest strategiczna sztuczna inteligencja. W kontekście nas, strategia odnosi się do ogólnego podejścia stosowanego przez grupę postaci. W tej kategorii znajdują się algorytmy AI, które nie kontrolują tylko jednego znaku, ale wpływają na zachowanie całego zestawu znaków. Każda postać w grupie może (i zwykle będzie miała) własne algorytmy podejmowania decyzji i ruchów, ale ogólnie na ich decyzje wpływać będzie strategia grupy. W oryginalnym Half-Life [Valve, 1998] wrogowie pracowali jako zespół, aby otoczyć i wyeliminować gracza. Często mijamy gracza, by zająć pozycję flankującą. Zostało to zaobserwowane w nowszych grach, takich jak Ghost Recon Toma Clancy'ego [Red Storm Entertainment, Inc., 2001], z rosnącym wyrafinowaniem w strategicznych działaniach, które może wykonać zespół wrogów.

Infrastruktura

Jednak same algorytmy AI to tylko połowa historii. Aby zbudować sztuczną inteligencję dla gry, potrzebujemy całego zestawu dodatkowej infrastruktury. Prośby o ruch muszą zostać przekształcone w akcję w grze za pomocą animacji lub, w coraz większym stopniu, symulacji fizyki. Podobnie AI potrzebuje informacji z gry, aby podejmować rozsądne decyzje. Czasami nazywa się to „percepcją" (szczególnie w akademickim AI): sprawdzanie, jakie informacje zna postać. W praktyce jest znacznie szerszy niż tylko symulacja tego, co każda postać może zobaczyć lub usłyszeć, ale obejmuje wszystkie interfejsy między światem gry a AI. Ten światowy interfejs często stanowi dużą część pracy wykonanej przez programistę AI, a z naszego doświadczenia wynika, że jest to największy odsetek wysiłków związanych z debugowaniem AI. Wreszcie cały system AI musi być zarządzany, aby zużywał odpowiednią ilość czasu procesora i pamięci. Chociaż dla każdego obszaru gry zwykle istnieje pewien rodzaj zarządzania wykonywaniem (na przykład algorytmy renderowania szczegółowe), zarządzanie AI podnosi cały zestaw własnych technik i algorytmów. Każdy z tych elementów może być uznany za niezależny od twórcy sztucznej inteligencji. Czasami tak jest (w szczególności system animacji jest prawie zawsze częścią silnika graficznego), ale są one tak ważne, aby AI działało, że nie można ich całkowicie uniknąć. W tej książce omówiliśmy każdy element infrastruktury oprócz animacji.

AI oparte na agentach

Nie używamy terminu „agenci”, mimo że model, który opisaliśmy, jest modelem opartym na agentach. W tym kontekście sztuczna inteligencja oparta na agentach polega na wytwarzaniu autonomicznych postaci, które pobierają informacje z danych gry, określają, jakie działania należy podjąć na podstawie tych informacji, i wykonują je. Można to postrzegać jako projekt oddolny: zaczynasz od ustalenia, jak zachowa się każda postać, i wdrożenia sztucznej inteligencji potrzebnej do jej obsługi. Ogólne zachowanie całej gry jest po prostu funkcją współdziałania zachowań poszczególnych postaci. Dwa pierwsze elementy używanego przez nas modelu AI, ruch i podejmowanie decyzji, tworzą AI dla agenta w grze. Natomiast sztuczna inteligencja nie oparta na agentach stara się ustalić, jak wszystko powinno działać od góry do dołu, i buduje jeden system do symulacji wszystkiego. Przykładem jest symulacja ruchu i pieszych w miastach Grand Theft Auto 3 [DMA Design, 2001]. Całkowity ruch i ruch pieszych są obliczane na podstawie pory dnia i regionu miasta i są przekształcane w pojedyncze samochody i osoby tylko wtedy, gdy gracz je widzi. Rozróżnienie jest jednak niejasne. Przyjrzymy się poziomowi technik szczegółowych, które są bardzo z góry na dół, podczas gdy większość postaci AI jest oddolna. Dobry programista AI będzie łączyć i dopasowywać wszelkie niezawodne techniki, które wykonają zadanie, niezależnie od podejścia. To pragmatyczne podejście jest tym, za którym zawsze podążamy. Dlatego w tej książce unikamy terminologii opartej na agentach. Wolimy rozmawiać o postaciach w ogóle, jednak mają one strukturę

Będziemy odwoływali się do tego modelu sztucznej inteligencji, wskazując, gdzie się mieści. Model ten jest przydatny do zrozumienia, jak rzeczy się do siebie pasują i jakie techniki są alternatywami dla innych. Ale linie podziału nie zawsze są ostre; ma to być ogólny model, a nie zwykły żakiet. W ostatecznym kodzie gry nie ma połączeń. Cały zestaw technik AI z każdej kategorii, a także duża część infrastruktury, będą działały bezproblemowo razem. Wiele technik spełnia role w więcej niż jednej kategorii. Na przykład odnajdywanie ścieżek może być zarówno ruchem, jak i techniką podejmowania decyzji. Podobnie, niektóre algorytmy taktyczne, które analizują zagrożenia i szanse w środowisku gry, mogą być użyte jako decydenci dla jednej postaci lub do określenia strategii całego zespołu.

Algorytmy, struktury danych i reprezentacje

Istnieją trzy kluczowe elementy wdrażania technik jakie opisujemy: sam algorytm, struktury danych, od których zależy algorytm oraz sposób reprezentacji świata gry do algorytmu (często zakodowanego jako odpowiednia struktura danych). Każdy element jest omawiany osobno

Algorytmy

Algorytmy są procesami krok po kroku, które generują rozwiązanie problemu sztucznej inteligencji. Przyjrzymy się algorytmom generującym trasy przez poziom gry, aby osiągnąć cel, algorytmy, które ustalają, w którym kierunku się poruszać, aby przechwycić uciekającego wroga, algorytmy które dowiadują się, co zrobi gracz, i wiele innych. Struktury danych są drugą stroną medalu w stosunku do algorytmów. Przechowują dane w taki sposób, że algorytm może szybko nimi manipulować w celu znalezienia rozwiązania. Często struktury danych muszą być szczególnie dostrojone dla jednego konkretnego algorytmu, a ich szybkość wykonywania jest wewnętrznie powiązana. Musisz znać zestaw elementów do implementacji i dostrajania algorytmu, a są one omówione krok po kroku w tekście:

* Problem, który algorytm próbuje rozwiązać

* Ogólny opis działania rozwiązania, w tym schematy tam, gdzie są potrzebne

* Prezentacja algorytmu w pseudokodzie

* Wskazanie struktur danych wymaganych do obsługi algorytmu, w tym w razie potrzeby pseudokod

* Poszczególne węzły implementacyjne

* Analiza wydajności algorytmu: szybkość jego wykonywania, wielkość pamięci i skalowalność

* Słabości w podejściu

Często prezentowany jest zestaw algorytmów, który staje się coraz bardziej wydajny. Prostsze algorytmy zostały przedstawione, aby pomóc Ci zrozumieć, dlaczego złożone algorytmy mają swoją strukturę. Odskocznię opisano nieco bardziej szkicowo niż pełny system. Niektóre kluczowe algorytmy w AI mają dosłownie setki odmian. Ta książka nie ma nadziei na skatalogowanie i opisanie ich wszystkich. Kiedy opisano kluczowy algorytm, często dajemy szybki przegląd głównych różnic w terminach skrótowych.

Charakterystyka wydajności

W największym możliwym stopniu próbowaliśmy w każdym przypadku uwzględnić właściwości wykonania algorytmu. Szybkość wykonywania i zużycie pamięci często zależą od wielkości rozważanego problemu. Zastosowaliśmy standardową notację O(), aby wskazać kolejność najbardziej znaczącego elementu w tym skalowaniu. Algorytm można opisać jako O(n log n) podczas wykonywania i O(n) pamięć, gdzie n jest zwykle pewnego rodzaju składnikiem problemu, takim jak liczba innych znaków w obszarze lub liczba mocy wzloty na poziomie. Każdy dobry tekst na temat ogólnego projektu algorytmu da pełne matematyczne podejście do sposobu, w jaki osiąga się wartości O() i ich wpływu na rzeczywistą wydajność algorytmu. My pominiemy pochodne; nie są przydatne do praktycznego wdrożenia. Zamiast tego będziemy polegać na ogólnym wskazaniu. Jeśli pełne wskazanie złożoności jest zbyt skomplikowane, podamy przybliżony czas działania lub pamięć w tekście, zamiast próbować uzyskać dokładną wartość O(). Niektóre algorytmy mają mylące charakterystyki wydajności. Można skonfigurować bardzo nieprawdopodobne sytuacje, w których celowo powodują one słabe wyniki. Przy regularnym użyciu (i na pewno przy każdym użyciu, który prawdopodobnie będziesz mieć w grze), będą one miały znacznie lepszą wydajność. W takim przypadku staraliśmy się wskazać zarówno oczekiwane, jak i najgorsze wyniki. Prawdopodobnie możesz bezpiecznie zignorować najgorszą wartość przypadku.

Pseudo kod

Nasze algorytmy są przedstawione w pseudokodzie dla zwięzłości i prostoty. Pseudokod jest fałszywym językiem programowania, który wycina wszelkie szczegóły implementacji dotyczące jednego języka programowania, ale opisuje algorytm wystarczająco szczegółowo, aby jego implementacja stała się prosta. Nasz pseudo-kod ma więcej wyczucia w języku programowania niż niektóre w książkach z czystym algorytmem (ponieważ zawarte tu algorytmy są często ściśle powiązane z otaczającymi fragmentami oprogramowania w sposób, który jest bardziej naturalnie uchwycony za pomocą idiomów programowania). W szczególności wiele algorytmów AI musi współpracować ze stosunkowo zaawansowanymi strukturami danych: listami, tabelami i tak dalej. W C ++ struktury te są dostępne tylko jako biblioteki i są dostępne za pośrednictwem funkcji. Aby wszystko było wyraźniejsze, pseudo-kod traktuje te struktury danych w sposób przejrzysty, znacznie upraszczając kod. Tworząc pseudo-kod w tej książce, trzymaliśmy się tych konwencji, tam gdzie to możliwe:

* Wcięcie wskazuje na strukturę bloku i zwykle jest poprzedzone dwukropkiem. Nie ma w tym nawiasów klamrowych ani instrukcji „end”. To sprawia, że znacznie prostszy kod, z mniej redundantnymi liniami do nadmuchiwania list. Dobry styl programowania zawsze wykorzystuje wcięcia, jak również inne znaczniki bloków, więc równie dobrze możemy po prostu użyć wcięcia.

* Funkcja wprowadza słowo kluczowe def, a klasa wprowadza słowo kluczowe class lub struct. Klasy odziedziczone podano po nazwie klasy w nawiasach. Podobnie jak w C ++, jedyną różnicą między

klasami i strukturami jest to, że struktury mają mieć bezpośredni dostęp do swoich zmiennych składowych.

* Konstrukcje zapętlające są a i a dla b. Pętla for może iterować dowolną tablicę. Może także iterować po szeregu liczb (w stylu C ++), używając składni dla a w 0..5. Ten ostatni element składni to zakres.

* Przedziały zawsze zawierają najniższą wartość, ale nie najwyższą, więc 1..4 obejmuje tylko liczby (1, 2, 3). Zakresy mogą być otwarte, takie jak 1 .., czyli wszystkie liczby są większe lub równe 1; lub ..4, który jest identyczny z 0..4. Zakresy mogą się zmniejszać, ale zauważ, że najwyższa wartość wciąż nie mieści się w zakresie: 4..0 jest ustawionym (3, 2, 1, 0) .1

* Wszystkie zmienne są lokalne dla funkcji lub metody. Zmienne zadeklarowane w definicji klasy, ale nie w metodzie, są zmiennymi instancji klasy.

* Pojedynczy znak równości „=" jest operatorem przypisania, podczas gdy podwójny znak równości „==" jest testem równości.

* Operatory logiczne to „i", „lub" i „nie".

* Dostęp do metod klasowych można uzyskać po nazwie, używając kropki między zmienną instancji a metodą - na przykład instance.variable ().

* Symbol „#" wprowadza komentarz do pozostałej części wiersza.

* Elementy tablicy podano w nawiasach kwadratowych i mają indeksy zerowe (tzn. Pierwszy element tablicy a to [0]). Podtablica jest oznaczona zakresem w nawiasach, więc [2..5] to podtablica składająca się z 3 do 5 elementów tablicy a. Obowiązują formularze otwartego zakresu: a [1 ..] to pod-tablica zawierająca wszystkie oprócz pierwszego elementu.

* Zasadniczo zakładamy, że tablice są równoważne listom. Możemy zapisywać je jako listy oraz dowolnie dodawać i usuwać elementy: jeśli tablica a ma wartość [0,1,2] i napiszemy + = 3, to a będzie miała wartość [0,1,2,3].

* Wartości logiczne mogą być „prawdziwe" lub „fałszywe".

Przykładowo poniższy przykład to pseudo-kod dla prostego algorytmu do wyboru najwyższej wartości z nieposortowanej tablicy:

```
def maximum(array):

max = array[0]

for element in array[1..]:

if element > max: max = element

return max
```

Czasami wyjaśniona zostanie specyficzna dla algorytmu część składni, gdy pojawi się ona w tekście. Ludzie wszzechstronni programistycznie prawdopodobnie zauważą, że pseudo-kod ma więcej niż przejściowe podobieństwo do języka programowania Python, z okazjonalnie wyskakującymi strukturami podobnymi do Ruby i przyprawianiem Lua. Jest to celowe, o ile Python jest językiem łatwym do odczytania. Niemniej jednak są one nadal pseudokodami, a nie implementacjami w języku Python, i wszelkie podobieństwa są nie powinno sugerować stronniczości języka ani implementacji

Reprezentacje

Informacje w grze często muszą zostać przekształcone w odpowiedni format do wykorzystania przez AI. Często oznacza to konwersję do innej reprezentacji lub struktury danych. Gra może przechowywać poziom jako zestawy geometrii, a pozycje postaci jako lokalizacje 3D na świecie. AI często musi przekonwertować te informacje na formaty odpowiednie do wydajnego przetwarzania. Ta konwersja jest procesem krytycznym, ponieważ często gubi informacje (o to chodzi: aby uprościć nieistotne szczegóły) i zawsze istnieje ryzyko utraty niewłaściwych fragmentów danych. Reprezentacje są kluczowym elementem AI, a niektóre kluczowe reprezentacje są szczególnie ważne w AI. Kilka algorytmów zawartych w książce wymaga przedstawienia gry w określonym formacie. Chociaż jest bardzo podobna do struktury danych, często nie martwimy się bezpośrednio o sposób implementacji reprezentacji, ale zamiast tego skupimy się na interfejsie, który przedstawia kodowi AI. Ułatwia to integrację technik sztucznej inteligencji z grą, po prostu tworząc odpowiedni kod kleju, aby przekształcić dane gry w reprezentację wymaganą przez algorytmy. Wyobraźmy sobie na przykład, że chcemy się dowiedzieć, czy postać czuje się zdrowa, czy nie, jako część niektórych algorytmów określających jego działania. Możemy po prostu wymagać przedstawienia znaku za pomocą metody, którą możemy wywołać:

```
class Character:

# Returns true if the character feels healthy,

# and false otherwise.

def feelsHealthy()
```

Następnie możesz to zaimplementować, sprawdzając wynik zdrowotny postaci, utrzymując logiczną „zdrową” wartość dla każdej postaci, a nawet uruchamiając cały algorytm, aby określić stan psychiczny postaci i jej postrzeganie własnego zdrowia. Jeśli chodzi o procedurę decyzyjną, nie ma znaczenia, w jaki sposób generowana jest wartość. Pseudokod definiuje interfejs (w sensie obiektowym), który można zaimplementować w dowolny sposób. Kiedy przedstawienie jest szczególnie ważne lub trudne (a jest ich kilka), opiszemy możliwe implementacje dogłębnie.