Rozwój gier żyje we własnym świecie technicznym. Ma własne idiomy, umiejętności i wyzwania. To jeden z powodów, dla których praca nad grami jest tak fajna. Każda gra ma swoje własne zasady, własną estetykę, własne kompromisy, a sprzęt, na którym będzie działać, ciągle się zmienia. Istnieje spora szansa, że będziesz pierwszą osobą, która spotka się i pokona nowe wyzwanie programistyczne. Pomimo licznych wysiłków zmierzających do standaryzacji tworzenia gier, zgodnie z resztą branży oprogramowania (wysiłki sięgające co najmniej 25 lat), styl programowania w grze jest nadal dość wyjątkowy. Nacisk kładziony jest na szybkość, ale różni się od programowania w czasie rzeczywistym dla aplikacji wbudowanych lub sterujących. Koncentruje się na sprytnych algorytmach, ale nie ma takiego samego rygoru, jak inżynieria serwerów baz danych. Czerpie techniki z ogromnej gamy różnych źródeł, ale prawie bez wyjątku modyfikuje je niepodobnie. Aby dodać dodatkową warstwę intrygi, programiści dokonują swoich modyfikacji na różne sposoby, często pod ekstremalną presją czasu i całkowicie dostosowują się do gry, pozostawiając algorytmy nierozpoznawalne od studia do studia lub od projektu do projektu. Choć może to być ekscytujące i trudne, nowym programistom trudno jest uzyskać potrzebne informacje. Dwadzieścia lat temu zdobycie informacji o technikach i algorytmach, które prawdziwi programiści używali w swoich grach, było prawie niemożliwe. W przypadku technik kodowania w najlepszych studiach panowała atmosfera tajemniczości, a nawet alchemii. Potem pojawił się Internet i stale powiększająca się gama stron internetowych, a także książki, konferencje i czasopisma. Teraz łatwiej niż kiedykolwiek można nauczyć się nowych technik tworzenia gier. Ta książka ma na celu pomóc ci opanować jeden element tworzenia gier: sztuczną inteligencję (AI). Opublikowano wiele artykułów na temat różnych aspektów sztucznej inteligencji w grach: strony internetowe poświęcone konkretnym technikom, kompilacje w formie książkowej, kilka tekstów wprowadzających oraz mnóstwo wykładów na konferencjach deweloperskich. Zakładam, że potrafisz programować (chociaż nie zakładam żadnego konkretnego języka), i założę, że masz wiedzę matematyczną na poziomie szkoły średniej. Zanim przejdziemy do samych technik, ten rozdział przedstawia sztuczną inteligencję, jej historię i sposób jej użycia. Przyjrzymy się modelowi sztucznej inteligencji, który pomoże dopasować techniki do siebie, i przedstawię pewne informacje na temat struktury pozostałej części.

CO TO JEST SZTUCZNA INTELIGENCJA?

Sztuczna inteligencja polega na umożliwieniu komputerom wykonywania zadań myślowych, do których zdolni są ludzie i zwierzęta. Możemy zaprogramować komputery tak, aby miały nadludzkie zdolności w rozwiązywaniu wielu problemów: arytmetyki, sortowania, wyszukiwania i tak dalej. Niektóre z tych problemów były pierwotnie uważane za problemy AI, ale ponieważ były rozwiązywane coraz bardziej kompleksowo, wymknęły się z domeny programistów AI. Ale jest wiele rzeczy, w których komputery nie są dobre, a które są dla nas trywialne: rozpoznawanie znajomych twarzy, mówienie we własnym języku, decydowanie, co dalej i bycie kreatywnym. To domena sztucznej inteligencji: próba ustalenia, jakie rodzaje algorytmów są potrzebne do wyświetlenia tych właściwości. Często granica między sztuczną inteligencją a nie-AI jest po prostu trudnością: rzeczy, których nie możemy zrobić, wymagają sztucznej inteligencji, rzeczy, które możemy, to sztuczki i matematyka. Kuszące jest wdanie się w dyskusję na temat tego, czym jest „prawdziwa" sztuczna inteligencja, próba zdefiniowania „inteligencji", „świadomości" lub „myśli". Z mojego doświadczenia wynika, że jest to zadanie niewykonalne, w dużej mierze nieistotne dla biznesu tworzenia gier. W środowisku akademickim niektórzy badacze sztucznej inteligencji są motywowani tymi filozoficznymi pytaniami: zrozumieniem natury myśli i natury inteligencji oraz budowaniem oprogramowania do modelowania, jak może działać myślenie. Innych motywuje psychologia: zrozumienie mechaniki ludzkiego mózgu i procesów umysłowych. A jeszcze innych motywuje inżynieria: budowanie algorytmów do wykonywania zadań podobnych do ludzkich. To potrójne rozróżnienie leży u podstaw akademickiej sztucznej inteligencji, a różne kwestie są odpowiedzialne za różne poddziedziny przedmiotu. Jako twórcy gier jesteśmy

praktycznymi ludźmi; zainteresowany tylko stroną inżynierską. Tworzymy algorytmy, które sprawiają, że postacie w grze wyglądają jak ludzkie lub zwierzęce. Deweloperzy zawsze czerpali z badań akademickich, które pomagają im wykonać zadanie, i ignorowali resztę. Warto zrobić szybki przegląd prac nad sztuczną inteligencją wykonywaną w środowisku akademickim, aby zorientować się, co istnieje w temacie, a co może być warte plagiatu.

AKADEMICKA AI

Aby opowiedzieć tę historię, podzielę akademicką sztuczną inteligencję na trzy okresy: wczesne dni, erę symboliczną oraz erę obliczeń naturalnych i erę statystyczną. Jest to oczywiście rażące uproszczenie i wszystkie one do pewnego stopnia się pokrywają, ale uważam to za użyteczne rozróżnienie

Wczesne dni

Wczesne dni obejmują czasy przed komputerami, gdzie filozofia umysłu od czasu do czasu robiła wypady na sztuczną inteligencję z takimi pytaniami jak: „Co wytwarza myśl?" „Czy mógłbyś ożywić przedmiot nieożywiony?" „Jaka jest różnica między zwłokami a człowiekiem, którym wcześniej był?" Powiązany z tym był popularny gust w automatach, robotach mechanicznych, począwszy od XVIII wieku. Stworzono skomplikowane modele zegarków, które wykazywały rodzaj animowanych, zwierzęcych lub ludzkich zachowań, które teraz zatrudniamy u twórców gier do tworzenia w pakiecie modelowania. W wysiłku wojennym lat 40. konieczność łamania kodów wroga i wykonywania obliczeń potrzebnych do prowadzenia wojny atomowej zmotywowała do opracowania pierwszych programowalnych komputerów . Biorąc pod uwagę, że maszyny te były używane do wykonywania obliczeń, które w innym przypadku byłyby wykonywane przez człowieka, naturalne było, że programiści interesowali się sztuczną inteligencją. Kilku pionierów komputerowych (takich jak Turing, von Neumann i Shannon) było również pionierami wczesnej sztucznej inteligencji. Zwłaszcza Turing stał się adoptowanym ojcem pola, w wyniku pracy filozoficznej, którą opublikował w 1950 r.

Era symboliczna

Od późnych lat pięćdziesiątych do wczesnych lat osiemdziesiątych 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 tworzyć nowe kombinacje, które, miejmy nadzieję, reprezentują problem rozwiązania lub nowa wiedza. System ekspercki, jeden z najczystszych przejawów tego podejścia, jest jedną z najbardziej znanych technik sztucznej inteligencji. Jeśli dziś wszystkie nagłówki dotyczące sztucznej inteligencji mówią o „głębokim uczeniu się", w latach osiemdziesiątych, zrzucają nazwę „systemów eksperckich". System ekspercki posiada dużą bazę wiedzy i stosuje zbiór reguł do wyciągania wniosków lub odkrywania nowych rzeczy. Inne symboliczne podejścia mające zastosowanie do gier obejmują architektury tablicowe, odnajdywanie ścieżek, drzewa decyzyjne i maszyny stanowe. Wspólną cechą systemów symbolicznych jest kompromis: przy rozwiązywaniu problemu im więcej masz wiedzy, tym mniej pracy musisz wykonać w rozumowaniu. Często algorytmy rozumowania polegają na wyszukiwaniu: próbowaniu różnych możliwości, aby uzyskać najlepszy wynik. To prowadzi nas do złotej zasady AI, którą zobaczymy w różnych formach:

Poszukiwanie i wiedza są ze sobą nierozerwalnie związane. Im więcej masz wiedzy, tym mniej potrzebujesz odpowiedzi; im więcej możesz przeprowadzić wyszukiwań (tj. im szybciej możesz wyszukiwać), tym mniej potrzebujesz wiedzy.

Naukowcy Newell i Simon w 1976 roku zasugerowali, że wyszukiwanie oparte na wiedzy (znane jako „poszukiwanie heurystyczne") jest sposobem, w jaki powstają wszystkie inteligentne zachowania. Niestety, pomimo posiadania kilku solidnych i ważnych cech, teoria ta została w dużej mierze zdyskredytowana jako opis wszelkiej inteligencji. Niemniej jednak wiele osób z niedawną edukacją w zakresie sztucznej inteligencji nie zdaje sobie sprawy, że jako kompromis inżynieryjny, wiedza kontra wyszukiwanie jest nieuniknione. Ostatnie prace nad matematyką rozwiązywania problemów dowiodły tego teoretycznie. Na poziomie praktycznym inżynierowie AI od zawsze to wiedzieli.

Naturalna era informatyki/statystyki

W latach 80. i na początku lat 90. narastała frustracja związana z podejściem symbolicznym. Frustracja płynęła z różnych stron. Z inżynierskiego punktu widzenia wczesne sukcesy w prostych problemach nie wydawały się skalować do trudniejszych problemów. Na przykład: wydawało się, że łatwo jest rozwinąć sztuczną inteligencję, która rozumiała (lub wydawała się rozumieć) proste zdania, ale rozwijanie zrozumienia pełnego ludzkiego języka nie wydawało się bliższe. To było spotęgowane przez szum: kiedy sztuczna inteligencja reklamowana jako „następna wielka rzecz" nie spełniła swoich rachunków, zaufanie do całego sektora spadło. Pojawił się również wpływowy argument filozoficzny, że podejścia symboliczne nie były biologicznie wiarygodne. Nie możesz zrozumieć, jak człowiek planuje trasę, używając symbolicznego algorytmu planowania trasy, tak samo jak nie możesz zrozumieć, jak pracują ludzkie mięśnie, studiując wózek widłowy. Efektem było przejście w kierunku przetwarzania naturalnego: technik inspirowanych biologią lub innymi systemami naturalnymi. Techniki te obejmują sieci neuronowe, algorytmy genetyczne i symulowane wyżarzanie. Wiele naturalnych technik obliczeniowych istnieje od dawna. Na przykład sieci neuronowe poprzedzają erę symboliczną; po raz pierwszy zaproponowano je w 1943 roku. Jednak w latach 80. do początku XXI wieku większość prac badawczych poświęcono im. w latach 90-tych. trudno było znaleźć miejsca badawcze np. w systemach eksperckich. Pomimo swojego pochodzenia jako korelatu biologii, sztuczna inteligencja bada w dużym stopniu matematykę stosowaną, w szczególności prawdopodobieństwo i statystykę, w celu zrozumienia i optymalizacji naturalnych technik obliczeniowych. Zdolność do radzenia sobie z całą niepewnością i bałaganem danych rzeczywistych, w przeciwieństwie do czystych i sztywnych granic podejść symbolicznych, doprowadziła do rozwoju szerokiej gamy innych technik probabilistycznych, takich jak sieci Bayesa, maszyny wektorów nośnych (SVM) i procesy Gaussa. Największa zmiana w sztucznej inteligencji w ostatniej dekadzie nie wynika z przełomu w środowisku akademickim. Żyjemy w czasach, gdy sztuczna inteligencja znów pojawia się w gazetach: samojezdne samochody, głębokie podróbki, programy mistrza świata Go i domowi wirtualni asystenci. To era głębokiego uczenia się. Chociaż wykorzystuje się wiele innowacji akademickich, systemy te nadal są zasadniczo zasilane przez sieci neuronowe, które teraz stają się praktyczne dzięki zwiększeniu mocy obliczeniowej.

Inżynieria

Chociaż nagłówki gazet i głośne aplikacje rozkwitły w ciągu ostatnich pięciu lat, sztuczna inteligencja od dziesięcioleci jest kluczową technologią istotną w rozwiązywaniu rzeczywistych problemów. Systemy nawigacyjne w samochodach, planowanie pracy w fabrykach, rozpoznawanie i dyktowanie głosu oraz wyszukiwanie na dużą skalę mają ponad 20 lat. Na przykład technologia wyszukiwania Google jest od dawna wspierana przez sztuczną inteligencję. To nie przypadek, że Peter Norvig jest zarówno dyrektorem ds. badań w Google, jak i współautorem (wraz ze swoim byłym doradcą absolwentów, profesorem Stuartem Russellem) kanonicznego podręcznika do studiów licencjackich dla nowoczesnej akademickiej sztucznej inteligencji. Kiedy coś jest gorące, kuszące jest założenie, że to jedyna rzecz, która ma znaczenie. Kiedy naturalne techniki komputerowe zajęły centralne miejsce, istniała tendencja do zakładania, że podejścia symboliczne są martwe. Podobnie, mówiąc wszędzie o głębokim uczeniu się, możesz otrzymać wybaczenie, że myślisz, że właśnie tego należy użyć. Ale zawsze

wracamy do tego samego kompromisu: wyszukiwanie vs wiedza. Głębokie uczenie jest szczytem wyszukiwania intensywnego obliczeniowo, AlphaGo Zero (trzecia iteracja oprogramowania AlphaGo) otrzymał bardzo minimalną wiedzę na temat reguł gry, ale wymagał nadzwyczajnego czasu przetwarzania, aby wypróbować różne strategie i nauczyć się najlepszych. Z drugiej strony postać, która musi użyć apteczki, gdy zostanie ranna, może wyraźnie powiedzieć:

JEŚLI jest ranny, TO użyj pakietu zdrowotnego

Wyszukiwanie nie jest wymagane. Jedyny sposób, w jaki dowolny algorytm może przewyższyć inny, to zużywanie większej mocy obliczeniowej (więcej wyszukiwania) lub optymalizacja pod kątem określonego zestawu problemów (większa wiedza o problemie). W praktyce inżynierowie pracują z obu stron. Na przykład program do rozpoznawania głosu konwertuje sygnały wejściowe przy użyciu znanych formuł na format, w którym sieć neuronowa może je zdekodować. Wyniki są następnie zasilane przez szereg algorytmów symbolicznych, które analizują słowa ze słownika i sposób łączenia słów w języku. Algorytm statystyczny optymalizujący kolejność linii produkcyjnej będzie miał reguły dotyczące produkcji zakodowane w swojej strukturze, więc nie może sugerować nielegalnego harmonogramu: wiedza jest wykorzystywana do zmniejszenia ilości wymaganych poszukiwań. Niestety gry są zazwyczaj projektowane do działania na sprzęcie konsumenckim. I chociaż sztuczna inteligencja jest ważna, grafika zawsze pobierała większość mocy obliczeniowej. Wydaje się, że nie ma niebezpieczeństwa zmiany. W przypadku sztucznej inteligencji zaprojektowanej do działania na urządzeniu podczas gry często wygrywają podejścia o niskim poziomie obliczeń/wysokiej wiedzy. A te są bardzo często symboliczne: podejścia zapoczątkowane w środowisku akademickim w latach 70. i 80. XX wieku. W tej książce przyjrzymy się kilku technikom obliczeń statystycznych, przydatnych w przypadku konkretnych problemów. Ale mam wystarczające doświadczenie, aby wiedzieć, że w grach często są one niepotrzebne: ten sam efekt często można osiągnąć lepiej, szybciej i z większą kontrolą, stosując prostsze podejście. Nie zmieniło się to znacząco od pierwszego wydania tej książki w 2004 roku. W przeważającej mierze sztuczna inteligencja wykorzystywana w grach jest nadal technologią symboliczną.

GRY AI

Pac-Man był pierwszą grą, którą wiele osób pamięta, grając z raczkującą sztuczną inteligencją. Do tego momentu istniały klony Ponga z kijami kontrolowanymi przez przeciwnika (podążające za piłką w górę iw dół) i niezliczonymi strzelcami w formie Space Invaders. Ale Pac-Man miał określone postacie wroga, które wydawały się spiskować przeciwko tobie, poruszały się po poziomie tak jak ty i utrudniały życie. Pac-Man polegał na bardzo prostej technice sztucznej inteligencji: maszynie stanów. Każdy z czterech potworów (później nazwanych duchami po katastrofalnie migoczącym porcie na Atari 2600) zajmował jeden z trzech stanów: goniący, rozpraszający się (kierujący się do zakrętów w określonych odstępach czasu) i przestraszony (gdy Pac-Man zjada power up).). Dla każdego stanu wybierają płytkę jako cel i zwracają się w jej kierunku na każdym skrzyżowaniu. W trybie pościgu każdy duch wybiera cel zgodnie z nieco inną, zakodowaną na sztywno regułą, nadając mu swoją osobowość. Sztuczna inteligencja gry niewiele się zmieniła do połowy lat 90. XX wieku. Większość sterowanych komputerowo postaci przedtem była mniej więcej tak wyrafinowana jak duch Pac-Mana. Weźmy taki klasyk jak Golden Axe osiem lat później. Postacie wroga stały nieruchomo (lub chodziły tam i z powrotem na niewielką odległość), dopóki gracz nie zbliżył się do nich, po czym skierowały się na gracza. Złoty Topór miał fajną innowację, ponieważ wrogowie przechodzili w stan biegania, by przemknąć obok gracza, a następnie wrócić do trybu naprowadzania, atakując od tyłu. Otoczenie gracza wygląda imponująco, ale bazowa sztuczna inteligencja nie jest bardziej złożona niż Pac-Man. W połowie lat 90. sztuczna inteligencja zaczęła być punktem sprzedaży gier. Gry takie jak Beneath a Steel Sky wspomniały nawet o sztucznej inteligencji z tyłu pudełka. Niestety, jego rozreklamowany system sztucznej inteligencji „Wirtualny

teatr" po prostu pozwalał postaciom poruszać się w przód iw tył przez grę - co nie jest prawdziwym postępem. Goldeneye 007 prawdopodobnie zrobił najwięcej, aby pokazać graczom, co może zrobić sztuczna inteligencja, aby poprawić rozgrywkę. Wciąż polegając na postaciach z niewielką liczbą dobrze zdefiniowanych stanów, Goldeneye dodał system symulacji zmysłów: postacie mogły widzieć swoich kolegów i zauważać, gdyby zostali zabici. Symulacja zmysłów była w tej chwili tematem, a Thief: The Dark Project i Metal Gear Solid oparli cały projekt gry na tej technice. W połowie lat 90. gry strategiczne czasu rzeczywistego (RTS) również zaczęły nabierać tempa. Warcraft był jednym z pierwszych przypadków, w których pathfinding został powszechnie zauważony w akcji (chociaż był używany kilka razy wcześniej). Badacze AI pracowali z emocjonalnymi modelami żołnierzy w wojskowej symulacji pola bitwy w 1998 roku, kiedy zobaczyli Warhammer: Dark Omen robiący to samo. Był to również jeden z pierwszych przypadków, w których ludzie zobaczyli silny ruch formacji w akcji. Halo wprowadziło drzewa decyzyjne ,teraz standardową metodę, dzięki której postacie decydują, co mają zrobić. F.E.A.R. w tym samym celu stosował planowanie działań zorientowane na cel ,Wraz z sukcesem AlphaGo, głębokie uczenie stało się gorącym tematem, choć nadal jest możliwe do zrealizowania tylko w trybie off-line. Niektóre gry są zaprojektowane w oparciu o sztuczną inteligencję. Creatures zrobiły to w 1997 roku, ale gry takie jak The Sims i ich kontynuacje, czy Black & White nadal trwają. Creatures wciąż ma jeden z najbardziej złożonych systemów AI widzianych w grze, z symulowanym systemem hormonalnym i mózgiem opartym na sieci neuronowej dla każdego stworzenia. Gry takie jak Half Life i The Last of Us używają postaci kontrolowanych przez AI do współpracy z graczem, co oznacza, że są na ekranie znacznie dłużej, a wszelkie błędy są znacznie bardziej zauważalne. Strzelanki FPS i gry RTS zostały poddane znaczącym badaniom naukowym (na przykład coroczny konkurs na sztuczną inteligencję Starcrafta). Gry RTS wykorzystują techniki sztucznej inteligencji wykorzystywane w symulacji wojskowej (do tego stopnia, że Full Spectrum Warrior zaczynał życie jako symulator szkolenia wojskowego). Gry sportowe i gry samochodowe w szczególności mają swoje własne wyzwania AI, z których niektóre pozostają w dużej mierze nierozwiązane (na przykład dynamiczne obliczanie najszybszego sposobu poruszania się po torze wyścigowym byłoby pomocne dla zespołów sportów motorowych), podczas gdy gry fabularne (RPG) ze złożonymi interakcjami postaci wciąż zaimplementowanymi, ponieważ drzewa rozmów wydają się spóźnione na coś lepszego (ciekawa i wyrafinowana sztuczna inteligencja konwersacyjna została zaimplementowana w grach takich jak Façade i Blood and Laurels ,jedyna wydana gra wykorzystująca krótkotrwały silnik gry Versu). Z pewnością przebyliśmy długą drogę. Ale chociaż mamy ogromną różnorodność sztucznej inteligencji w grach, wiele gatunków nadal używa prostej sztucznej inteligencji z 1979 roku, ponieważ to wszystko, czego potrzebują. Sztuczna inteligencja w większości gier odpowiada na trzy podstawowe potrzeby: zdolność do poruszania postaciami, zdolność do podejmowania decyzji o tym, gdzie się poruszać oraz umiejętność myślenia taktycznego lub strategicznego. Chociaż mamy szeroki zakres podejść, wszystkie spełniają te same trzy podstawowe wymagania.

MODEL GRY AI

Znajdziesz tu się ogromne zoo algorytmów i technik. Łatwo byłoby się zgubić, dlatego ważne jest, aby zrozumieć, jak te elementy do siebie pasują. Aby pomóc, użyłem spójnej struktury, aby kontekstualizować sztuczną inteligencję używaną w grze. Nie jest to jedyny możliwy model i nie jest to jedyny model, który skorzystałby na technikach tu opisanych. Aby jednak dyskusje były jaśniejsze, pokażę, jak każda technika pasuje do ogólnej struktury tworzenia inteligentnych postaci w grze. Dzielimy zadanie AI na trzy sekcje: ruch, podejmowanie decyzji i strategia. Pierwsze dwie sekcje zawierają algorytmy, które działają na zasadzie znak po znaku, a ostatnia sekcja działa w zespole lub po stronie. Wokół tych trzech elementów AI znajduje się cały zestaw dodatkowej infrastruktury. Nie wszystkie aplikacje do gier wymagają wszystkich poziomów sztucznej inteligencji. Gry planszowe, takie jak Szachy czy Warcaby, wymagają tylko poziomu strategii; postacie w grze (jeśli w ogóle można je tak

nazwać) nie podejmują własnych decyzji i nie muszą się martwić o to, jak się poruszać. Z drugiej strony w wielu grach nie ma żadnej strategii. Postacie niezależne w grach platformowych, takich jak Hollow Knight czy Super Mario Bros., są czysto reaktywne, podejmując własne, proste decyzje i na ich podstawie. Nie ma koordynacji, która zapewniłaby, że wrogie postacie najlepiej wykonają zadanie udaremnienia gracza.

RUCH

Ruch odnosi się do algorytmów, które zamieniają decyzje w pewien rodzaj ruchu. Kiedy postać wroga bez ataku pociskiem musi zaatakować gracza w Super Mario Sunshine, najpierw kieruje się bezpośrednio do gracza. Kiedy jest wystarczająco blisko, może faktycznie atakować. Decyzja o ataku jest podejmowana przez zestaw algorytmów ruchu, które skupiają się na lokalizacji gracza. Dopiero wtedy można odtworzyć animację ataku i wyczerpać zdrowie gracza. Algorytmy ruchu mogą być bardziej złożone niż zwykłe samonaprowadzanie. Postać może potrzebować omijania przeszkód po drodze, a nawet przedzierania się przez szereg pomieszczeń. Strażnik na niektórych poziomach Splinter Cell zareaguje na pojawienie się gracza, podnosząc alarm. Może to wymagać nawigowania do najbliższego naściennego punktu alarmowego, który może znajdować się w dużej odległości i może wiązać się ze skomplikowaną nawigacją wokół przeszkód lub przez korytarze. Wiele działań przeprowadza się bezpośrednio przy użyciu animacji. Jeśli Sim w The Sims siedzi przy stole z jedzeniem przed sobą i chce wykonać akcję jedzenia, po prostu odtwarzana jest animacja jedzenia. Gdy sztuczna inteligencja zdecyduje, że postać powinna jeść, nie jest już potrzebna sztuczna inteligencja (używana technologia animacji nie jest opisana w tej książce). Jeśli jednak ta sama postać jest przy tylnych drzwiach, kiedy chce zjeść, ruch AI musi poprowadzić ją do krzesła (lub do innego pobliskiego źródła pożywienia).

PODEJMOWANIE DECYZJI

Podejmowanie decyzji polega na tym, że postać zastanawia się, co dalej. Zazwyczaj każda postać ma szereg różnych zachowań, które może wykonać: atakowanie, stanie w miejscu, ukrywanie się, eksploracja, patrolowanie i tak dalej. System podejmowania decyzji musi ustalić, które z tych zachowań jest najbardziej odpowiednie w każdym momencie gry. Wybrane zachowanie można następnie wykonać za pomocą technologii ruchu AI i animacji. Najprościej rzecz ujmując, postać może mieć bardzo proste zasady wyboru zachowania. Zwierzęta gospodarskie na różnych poziomach gier Zelda będą stać w miejscu, chyba że gracz zbliży się zbyt blisko, po czym odsuną się na niewielką odległość. Z drugiej strony, wrogowie w Half-Life 2 [194] podejmują złożone decyzje, w których będą próbować różnych strategii, aby dotrzeć do gracza: łącząc ze sobą pośrednie działania, takie jak rzucanie granatów i podłożenie ognia tłumiącego w celu osiągnięcia ich cele. Niektóre decyzje mogą wymagać SI ruchu, aby je wykonać. Walka wręcz lub atak będzie wymagał od postaci zbliżenia się do ofiary. W ciężkich grach bojowych, takich jak Dark Souls, podejmowanie decyzji przesuwa postać w kierunku celu, a także określa, jaki atak, a tym samym, jaka animacja zostanie wykonana. W innych grach, po podjęciu decyzji, z góry określona animacja jest odtwarzana bez żadnego dodatkowego ruchu (na przykład jedzący Sim) lub stan gry jest bezpośrednio modyfikowany bez jakiejkolwiek wizualnej informacji zwrotnej (gdy SI kraju w Sid Meier's Civilization VI postanawia na przykład badać nową technologię, po prostu dzieje się to bez wizualnej informacji zwrotnej dla gracza).

STRATEGIA

Możesz przejść długą drogę z ruchu AI i podejmowania decyzji AI, a większość gier trójwymiarowych (3D) opartych na akcji wykorzystuje tylko te dwa elementy. Ale aby koordynować cały zespół, potrzebna jest strategiczna sztuczna inteligencja. Strategia odnosi się do ogólnego podejścia stosowanego przez grupę postaci. W tej kategorii znajdują się algorytmy AI, które nie kontrolują tylko

jednej postaci, ale wpływają na zachowanie całego zestawu postaci. Każda postać w grupie może (i zwykle będzie) mieć własne algorytmy podejmowania decyzji i ruchu, ale ogólnie na ich podejmowanie decyzji będzie miała wpływ strategia grupy. W oryginalnym Half-Life wrogowie pracowali jako zespół, aby otoczyć i wyeliminować gracza. Często mijano gracza, by zająć pozycję oskrzydlającą. Zostało to śledzone w nowszych grach, takich jak ewoluujący silnik AI w serii Medal of Honor. Z biegiem czasu zaobserwowaliśmy coraz większe wyrafinowanie w rodzajach działań strategicznych, które może przeprowadzić zespół wrogów.

INFRASTRUKTURA

Jednak same algorytmy AI to tylko połowa sukcesu. Aby faktycznie zbudować sztuczną inteligencję do gry, potrzebujemy całego zestawu dodatkowej infrastruktury. Żądania ruchu muszą zostać przekształcone w działanie w grze za pomocą animacji lub, w coraz większym stopniu, symulacji fizyki. Podobnie sztuczna inteligencja potrzebuje informacji z gry, aby podejmować rozsądne decyzje. Nazywa się to czasem „percepcją" (zwłaszcza w akademickiej sztucznej inteligencji): ustalenie, jakie informacje zna postać. W praktyce jest to znacznie szersze niż tylko symulowanie tego, co każda postać może zobaczyć lub usłyszeć, ale obejmuje wszystkie interfejsy między światem gry a sztuczną inteligencją. Ten światowy interfejs to często znaczna część pracy wykonywanej przez programistę AI, a z mojego doświadczenia wynika, że jest to duża część wysiłku związanego z debugowaniem AI. Wreszcie, cały system AI musi być zarządzany, aby wykorzystywał odpowiednią ilość czasu procesora i pamięci. Chociaż dla każdego obszaru gry istnieje zwykle pewien rodzaj zarządzania wykonaniem (na przykład algorytmy poziomu szczegółowości do renderowania), zarządzanie sztuczną inteligencją wymaga całego zestawu własnych technik i algorytmów. Każdy z tych komponentów może być uważany za będący poza zakresem kompetencji programisty AI. Czasami tak jest (w szczególności system animacji jest często częścią silnika graficznego lub coraz częściej ma własnych dedykowanych programistów), ale są tak kluczowe dla działania sztucznej inteligencji, że nie można ich całkowicie uniknąć. W tej książce szczegółowo omówiłem każdy element infrastruktury z wyjątkiem animacji.

AI OPARTA NA AGENTACH

Nie używam zbyt często terminu „agenci", mimo że opisany przeze mnie model jest modelem agentowym. W tym kontekście sztuczna inteligencja oparta na agentach polega na tworzeniu autonomicznych postaci, które pobierają informacje z danych gry, określają, jakie działania należy podjąć w oparciu o te informacje, i wykonują te działania. Można to postrzegać jako projekt oddolny: zaczynasz od ustalenia, jak będzie się zachowywać każda postać i zaimplementowania sztucznej inteligencji potrzebnej do tego. Ogólne zachowanie gry jest zatem funkcją tego, jak ze sobą współgrają zachowania poszczególnych postaci. Pierwsze dwa elementy modelu AI, którego użyjemy, ruch i podejmowanie decyzji, składają się na sztuczną inteligencję agenta w grze. W przeciwieństwie do tego, 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 w Grand Theft Auto 3. Ogólny przepływ ruchu i pieszych jest obliczany na podstawie pory dnia i regionu miasta i jest przekształcany w pojedyncze samochody i osoby tylko wtedy, gdy gracz je widzi. Jednak rozróżnienie jest mgliste. Przyjrzymy się technikom poziomu szczegółowości, które są bardzo odgórne, podczas gdy większość sztucznej inteligencji postaci jest oddolna. Dobry programista AI będzie mieszać i dopasowywać wszelkie niezawodne techniki, które pozwolą wykonać zadanie, niezależnie od podejścia. Zawsze kieruję się tym pragmatycznym podejściem. Dlatego w tej książce unikam terminologii opartej na agentach i wolę mówić ogólnie o postaciach w grze, niezależnie od ich struktury.

ALGORYTMY I STRUKTURY DANYCH

Istnieją trzy kluczowe elementy implementacji opisanych technik: sam algorytm, struktury danych, od których algorytm zależy, oraz sposób, w jaki świat gry jest przedstawiany algorytmowi (często zakodowany jako odpowiednia struktura danych). Każdy element jest omówiony osobno w tekście.

ALGORYTMY

Algorytmy to procesy, które krok po kroku generują rozwiązanie problemu AI. Przyjrzymy się algorytmom, które generują trasy przez poziom gry, aby osiągnąć cel, algorytmom, które określają kierunek ruchu, aby przechwycić uciekającego wroga, algorytmom, które uczą się, co gracz powinien zrobić dalej i wielu innym. Struktury danych to druga strona medalu algorytmów. Przechowują dane w taki sposób, że algorytm może szybko nimi manipulować, aby znaleźć rozwiązanie. Często struktury danych muszą być dostrojone do jednego konkretnego algorytmu, a szybkość ich wykonywania jest nierozerwalnie związana. Aby zaimplementować i dostroić algorytm, musisz znać zestaw elementów, które są omówione krok po kroku w tekście:

• Problem, który algorytm próbuje rozwiązać

• Ogólny opis działania rozwiązania, w tym diagramy tam, gdzie są potrzebne

• Pseudokodowa prezentacja algorytmu

• Wskazanie struktur danych wymaganych do obsługi algorytmu, w tym pseudokodu, jeśli jest to wymagane

• Doradztwo wdrożeniowe, w razie potrzeby

• Analiza wydajności algorytmu: szybkość jego wykonywania, zużycie pamięci i skalowalność

• Słabości w podejściu

Często przedstawiam zestaw algorytmów, które stają się coraz bardziej złożone i potężne. Przedstawiono prostsze algorytmy, aby pomóc Ci wyczuć, dlaczego złożone algorytmy mają swoją strukturę. Odskocznie są opisane nieco bardziej szkicowo niż cały system. Niektóre kluczowe algorytmy sztucznej inteligencji w grze mają dosłownie setki odmian. Kiedy opisywany jest kluczowy algorytm, często podam krótki przegląd głównych wariantów w krótszych terminach.

Charakterystyka wydajności

W miarę możliwości starałem się w każdym przypadku uwzględnić właściwości wykonawcze algorytmu. Szybkość wykonywania i zużycie pamięci często zależą od rozmiaru rozważanego problemu. Użyłem standardowej notacji O(), aby wskazać kolejność najbardziej znaczącego elementu w tym skalowaniu. Algorytm można opisać jako O(n log n) w wykonaniu i O(n) w pamięci, gdzie n jest zwykle jakimś składnikiem problemu, takim jak liczba innych znaków w obszarze lub liczba power-upy na poziomie. Dobry tekst na temat ogólnych algorytmów dla informatyki zapewni pełne matematyczne podejście do sposobu, w jaki wartości O() są uzyskiwane, oraz implikacji, jakie mają one dla wydajności algorytmu w świecie rzeczywistym. W tej książce pomijam wyprowadzenia lub dowody. Gdy pełne wskazanie złożoności jest zbyt skomplikowane, wskazuję przybliżony czas działania lub pamięć w tekście, zamiast próbować uzyskać dokładną wartość O(). Niektóre algorytmy mają wprowadzające w błąd charakterystyki wydajności. Możliwe jest tworzenie wysoce nieprawdopodobnych sytuacji, aby celowo sprawić, by działały słabo. W regularnym użyciu (i na pewno w każdym użyciu, które prawdopodobnie będziesz mieć w grze), będą miały znacznie lepszą wydajność. W takim przypadku starałem się wskazać zarówno oczekiwane, jak i najgorsze wyniki. O ile nie zaznaczono inaczej, prawdopodobnie możesz bezpiecznie zignorować wartość najgorszego przypadku.

Pseudo kod

Algorytmy u nas są przedstawione w pseudokodzie dla zwięzłości i prostoty. Pseudo-kod to wyimaginowany język programowania, który wycina wszelkie szczegóły implementacji specyficzne dla dowolnego prawdziwego języka programowania. Powinien opisywać algorytm wystarczająco szczegółowo, aby można go było zaimplementować w wybranym języku. Nasz pseudokod przypomina bardziej proceduralny język programowania w porównaniu z pseudokodem, który można znaleźć w bardziej teoretycznych książkach o algorytmach. Ponadto używam pseudokodu do opisywania struktur danych i interfejsów, a także algorytmów, ponieważ algorytmy nasze są często ściśle związane z otaczającymi go bitami oprogramowania w sposób, który jest bardziej naturalnie uchwycony za pomocą kodu. Wiele algorytmów AI musi pracować ze stosunkowo wyrafinowanymi strukturami danych: listami, tabelami, kolejkami priorytetowymi, tablicami asocjacyjnymi i tak dalej. Niektóre języki zapewniają te wbudowane, inne udostępniają je jako biblioteki lub dostęp za pośrednictwem funkcji. Aby wyjaśnić to, co się dzieje, pseudokod traktuje te struktury danych tak, jakby były częścią języka, znacznie upraszczając kod. Tworząc pseudokod w tej książce, w miarę możliwości trzymałem się tych konwencji:

• Wcięcie wskazuje na strukturę bloku i jest zwykle poprzedzone dwukropkiem. Nie ma nawiasów klamrowych ani stwierdzeń „koniec”. To sprawia, że kod jest znacznie prostszy, z mniejszą liczbą zbędnych linii, które rozdęłyby wykazy. Dobry styl programowania zawsze używa wcięć tak samo, jak innych znaczników bloków, więc równie dobrze możemy po prostu użyć wcięcia.

• Funkcje są wprowadzane przez słowo kluczowe function, a klasy są wprowadzane przez słowo kluczowe class. Klasa dziedziczona określa swoją klasę nadrzędną po rozszerzeniu słowa kluczowego. Pseudo-kod nie będzie zawierał seterów i getterów, chyba że są one znaczące, wszystkie zmienne składowe będą dostępne.

• Parametry funkcji są umieszczane w nawiasach, zarówno gdy funkcja jest zdefiniowana, jak i gdy jest wywoływana. Dostęp do metod klas uzyskuje się po nazwie, używając kropki między zmienną instancji a metodą - na przykład instance.variable().

• Typy mogą być podane po nazwie zmiennej lub nazwie parametru, oddzielone dwukropkiem. Typ zwracany z funkcji następuje ->.

• Wszystkie zmienne są lokalne w zakresie, w którym są zadeklarowane, zwykle funkcja lub metoda. 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. Modyfikatory przypisania, takie jak +=, istnieją dla wszystkich operacji matematycznych.

• Konstrukcje zapętlone to a i for a in b. Pętla for może iterować po dowolnej tablicy. Może również iterować po serii liczb, używając składni dla a in 0..5.

• Zakres jest oznaczony przez 0..5. Zakresy zawsze zawierają ich najniższą wartość, ale nie najwyższą, więc 1..4 zawiera tylko liczby (1; 2; 3). Zakresy mogą być otwarte, na przykład 1.., co oznacza wszystkie liczby większe lub równe 1; lub ..4, co jest identyczne z 0..4. Zakresy mogą się zmniejszać, ale zauważ, że najwyższa wartość nadal nie znajduje się w zakresie: więc 4..0 to zestaw (3; 2; 1; 0).2

• Operatory logiczne to i, lub i nie. Wartości logiczne mogą być prawdziwe lub fałszywe.

• Symbol # wprowadza komentarz do pozostałej części wiersza.

• Elementy tablicy są podane w nawiasach kwadratowych i są indeksowane przez zero (tj. pierwszym elementem tablicy a jest a[0]). Podtablica jest oznaczona zakresem w nawiasach, więc a[2..5] jest podtablicą składającą się z 3 do 5 elementów tablicy a. Formy z otwartym zakresem są poprawne: a[1..] jest podtablicą zawierającą wszystkie elementy oprócz pierwszego elementu a.

• Generalnie zakładam, że tablice są równoważne listom. Możemy pisać je jako listy oraz dowolnie dodawać i usuwać elementy.

Jako przykład poniższa próbka jest pseudokodem prostego algorytmu wyboru najwyższej wartości z nieposortowanej tablicy:

```
1 function maximum(array:float[]) -> float:

2 max: float = array[0]

3 for element in array[1..]:

4 if element > max:

5 max = element

6 return max
```

Od czasu do czasu zostanie wyjaśniony fragment składni specyficzny dla algorytmu, który pojawia się w tekście. Programiści prawdopodobnie zauważą, że pseudokod ma więcej niż przelotne podobieństwo do języka programowania Python, ze strukturami podobnymi do Ruby wyskakującymi od czasu do czasu i przyprawą Lua. Jest to celowe, ponieważ Python jest językiem łatwym do odczytania. Niemniej jednak podobieństwo jest tylko powierzchowne, listingi wciąż są pseudokodem, a nie implementacjami Pythona. Podobieństwo nie ma sugerować stronniczości językowej lub implementacyjnej

REPREZENTACJA

Informacje w grze często muszą zostać przekształcone w odpowiedni format do wykorzystania przez sztuczną inteligencję. Często oznacza to przekształcenie go w inną reprezentację lub strukturę danych. Gra może przechowywać poziom jako siatki geometrii 3D, a pozycje postaci jako (x; y; z) lokacje na świecie. Konwersja między tymi reprezentacjami jest procesem krytycznym, ponieważ często powoduje utratę informacji (o to chodzi: uprościć nieistotne szczegóły) i zawsze istnieje ryzyko utraty niewłaściwych fragmentów danych. Wybór właściwej reprezentacji jest kluczowym elementem implementacji AI, a niektóre reprezentacje są szczególnie ważne w grze AI. Kilka algorytmów zawartych w książce wymaga, aby gra została im przedstawiona w określonym formacie. Chociaż bardzo podobna do struktury danych, często nie będziemy martwić się bezpośrednio o to, jak zaimplementowana jest reprezentacja, ale zamiast tego skupimy się na interfejsie, który przedstawia kodowi AI. Ułatwia to integrację technik sztucznej inteligencji w grze, po prostu tworząc odpowiedni kod kleju, aby przekształcić dane gry w reprezentację wymaganą przez algorytmy. Na przykład wyobraź sobie, że chcemy sprawdzić, czy postać czuje się zdrowa, czy nie, w ramach jakiegoś algorytmu określającego jej działania. Możemy po prostu wymagać reprezentacji postaci za pomocą metody, którą możemy wywołać:

```
1 class Character:

2 # Zwróć prawdę, jeśli postać czuje się zdrowa, a w przeciwnym razie fałsz.
```

3 function feelsHealthy() -> bool

Możesz to następnie wdrożyć, sprawdzając wynik zdrowia postaci, utrzymując „zdrową” wartość logiczną dla każdej postaci, a nawet uruchamiając cały algorytm w celu określenia stanu psychicznego postaci i jej postrzegania własnego zdrowia. Jeśli chodzi o rutynę podejmowania decyzji, nie ma znaczenia, w jaki sposób generowana jest wartość. Pseudokod definiuje interfejs (w sensie obiektowym), który można zaimplementować w dowolny sposób. Gdy reprezentacja jest szczególnie ważna lub trudna (a jest ich kilka), opiszę szczegółowo możliwe implementacje.

IMPLEMENTACJA

Jeszcze dziesięć lat temu większość programistów używała C++ do swojego kodu AI. Dekadę wcześniej znaczna liczba opierająca się na rozwoju C. Games jest teraz znacznie bardziej zróżnicowana. Swift i Java (dla platform mobilnych), C# dla silnika gry unity, JavaScript w sieci. Istnieje wiele innych języków używanych tu i ówdzie przy tworzeniu gier: Lisp, Lua lub Python, zwłaszcza jako języki skryptowe; ActionScript dla kilku pozostałych programistów Flash. Osobiście pracowałem ze wszystkimi tymi językami w takim czy innym momencie, więc starałem się być jak najbardziej niezależny od języka, jednocześnie udzielając porad dotyczących implementacji. Spośród nich C i C++ są nadal używane w kodzie, który absolutnie musi działać tak szybko, jak to możliwe. W niektórych miejscach część dyskusji na temat struktur danych i optymalizacji skupi się na C++, ponieważ optymalizacje są specyficzne dla C++.