GRY AI

Zanim przejdziemy do szczegółów poszczególnych technik i algorytmów, warto poświęcić trochę czasu na zastanowienie się, czego potrzebujemy od sztucznej inteligencji naszej gry. W tej części przyjrzymy się ogólnym kwestiom związanym ze sztuczną inteligencją w grach: jakie rodzaje podejść działają, co należy wziąć pod uwagę i jak można je wszystkie połączyć.

BŁĄD ZŁOŻONOŚCI

Powszechnym błędem jest myślenie, że im bardziej skomplikowana sztuczna inteligencja w grze, tym lepiej postacie będą wyglądać dla gracza. Tworzenie dobrej sztucznej inteligencji polega na dopasowaniu wymagań gry do właściwych zachowań i odpowiednich algorytmów, które je produkują. W tym tekście jest oszałamiający wachlarz technik, a właściwa nie zawsze jest najbardziej oczywistym wyborem. Niezliczone przykłady trudnej do wdrożenia, złożonej sztucznej inteligencji zaowocowały złym, a nawet głupio wyglądającym zachowaniem. Podobnie, bardzo prosta technika może być doskonała, jeśli jest dobrze stosowana.

KIEDY PROSTE RZECZY WYGLĄDAJĄ DOBRZE

W ostatniej części wspomniałem o Pac-Manie, jednej z pierwszych gier z dowolną postacią AI postaci. AI ma trzy stany: jeden normalny, kiedy gracz zbiera pestki; drugi stan, kiedy gracz zjadł power-up i chce się zemścić; i stan końcowy, który wyzwala się w określonych odstępach czasu, aby duchy trochę się cofnęły. We wszystkich trzech stanach każdy z czterech duchów ma cel. Porusza się w linii prostej aż do skrzyżowania, a następnie wybiera trasę najbliższą jego celowi. Nie próbuje planować całej trasy, ani nawet sprawdzać, czy jej cel może zostać osiągnięty, po prostu zbliża się do niego. W stanie pogoni, podczas polowania na gracza, każdy duch ma swój własny, prosty fragment kodu, aby wybrać cel. Blinky (czerwony duch) zawsze celuje w pozycję gracza. Pinky (oczywiście różowy) celuje w kwadrat cztery pola przed graczem, nawet jeśli znajduje się on wewnątrz lub po drugiej stronie ściany. Inky (jasnoniebieski) używa zmodyfikowanego offsetu własnego i pozycji gracza. Clyde (pomarańczowy) celuje w gracza, jeśli jest daleko, lub w róg planszy, jeśli jest blisko. Wszystkie te procedury kierowania można zaimplementować w jednym lub dwóch wierszach kodu. To jest tak proste, jak możesz sobie wyobrazić sztuczną inteligencję dla poruszającej się postaci. Każdy prostszy, a duchy byłyby albo bardzo przewidywalne (jeśli zawsze się w nich znajdowały), albo całkowicie losowe. Na własną rękę strategię duchów można łatwo przewidzieć; ich sztuczna inteligencja nie stanowi wyzwania. Ale razem, różne zachowania każdego ducha wystarczą, aby stworzyć znaczącą przeciwną siłę - tak bardzo, że sztuczna inteligencja do dziś otrzymuje pochlebne komentarze. Na przykład ten komentarz pojawił się niedawno na stronie internetowej: „Aby nadać grze trochę napięcia, w grze zaprogramowano sprytną sztuczną inteligencję. Duchy grupowały się, atakowały gracza, a następnie rozpraszały. Każdy duch miał swoją własną sztuczną inteligencję.” Inni gracze zgłaszali strategie wśród duchów: „Czwórka z nich jest zaprogramowana do zastawienia pułapki, a Blinky prowadzi gracza w zasadzkę, w której czają się pozostali trzej”. Prosta sztuczna inteligencja, wykonana dobrze, może wydawać się graczowi znacznie bardziej inteligentna niż jest w rzeczywistości. To samo zostało zgłoszone przez wielu innych twórców własnych gier. Jako skrajny przykład: kilka lat temu Chris Kingsley z Rebellion wspomniał o nieopublikowanym tytule Game Boy na Nintendo, w którym poszczególne wrogie postacie po prostu trafiają na gracza, ale dla odmiany, omijają w losowych odstępach czasu, gdy posuwają się do przodu. Gracze zgłosili, że te postacie „przewidziały” ich wzorce strzelania i uniknęły z drogi. Sztuczna inteligencja niczego nie przewidziała i tylko czasami udało jej się uniknąć zbiegu okoliczności. Jednak unik w odpowiednim momencie w kluczowym momencie pozostał w umyśle gracza i ukształtował jego postrzeganie sztucznej inteligencji.

KIEDY ZŁOŻONE RZECZY WYGLĄDAJĄ ŹLE

Oczywiście łatwo może się zdarzyć coś odwrotnego. Gra, na którą z niecierpliwością czekano, Herdy Gerdy, jedna z gier startowych, które Sony wykorzystywały do reklamowania nowych możliwości rozgrywki w chipie „silnika emocji" w swoim sprzęcie PlayStation 2. Gra jest grą pasterską. Na poziomie gry obecny jest ekosystem postaci. Gracz musi zaganiać osobniki różnych gatunków do odpowiednich zagród. Herding był używany wcześniej i od tego czasu jest elementem większej gry, ale w Herdy Gerdy stanowił całą rozgrywkę. Niestety, ruch postaci AI nie był w stanie sprostać wyzwaniu, jakim było bogate projektowanie poziomów. Łatwo było ich złapać w scenerii, a wykrycie kolizji mogło sprawić, że utknęli w nieodwracalnych miejscach. Byłoby to frustrujące (choć nierzadkie) w każdej grze. Ale wchodził w interakcję z pasterską sztuczną inteligencją w sposób, który sprawiał, że niektóre postacie wydawały się niespodziewanie nieinteligentne. Recenzje były mieszane, a sprzedaż słaba. W przeciwieństwie do Herdy Gerdy firma Black & White osiągnęła znaczący sukces sprzedażowy. Ale w niektórych miejscach cierpiała również z powodu świetnej sztucznej inteligencji, która wyglądała źle. Gra polega na nauczeniu postaci, co ma robić, łącząc przykład i informacje zwrotne. Kiedy ludzie po raz pierwszy grają w tę grę, często nieumyślnie uczą stworzenie złych nawyków, a w najgorszych przypadkach może ono nie być w stanie wykonać nawet najbardziej podstawowych czynności. Zwracając większą uwagę na to, jak działa sztuczna inteligencja stworzenia, gracze są w stanie lepiej nim manipulować, ale iluzja uczenia prawdziwego stworzenia może zniknąć. Większość skomplikowanych rzeczy, które widziałem, a które wyglądały źle, nigdy nie dotarła do finałowej gry. Jest to odwieczna pokusa dla programistów, aby korzystać z najnowszych technik i najbardziej rozreklamowanych algorytmów do implementacji sztucznej inteligencji postaci. Na późnym etapie rozwoju, kiedy ucząca się sztuczna inteligencja nadal nie może nauczyć się kierować samochodem po torze bez odjeżdżania na każdym rogu, na ratunek przychodzą prostsze algorytmy i wprowadzają je do gry. Wiedza, kiedy być złożonym, a kiedy prostym, jest najtrudniejszym elementem sztuki programisty AI w grze. Najlepsi programiści AI to ci, którzy potrafią użyć bardzo prostej techniki, aby stworzyć iluzję złożoności. Jest to łatwiejsze, gdy istnieje ścisła pętla sprzężenia zwrotnego między implementacją a projektowaniem gry. Niewielka modyfikacja wymagań może oznaczać możliwość zastosowania lepszej techniki AI, co prowadzi do lepszej gry. Czasami oznacza to uproszczenie wymaganego zachowania, aby było znacznie bardziej niezawodne. Niestety, przy tak dużej liczebności zespołu zajmującego się masowymi grami na komputery PC i konsole, programiście trudno jest mieć duży wpływ. Gry niezależne i mobilne, których zespoły są znacznie mniejsze, choć nie tak małe jak kilka lat temu, wciąż mają więcej możliwości.

OKNO PERCEPCJI

O ile twoja sztuczna inteligencja nie kontroluje wszechobecnego pomocnika lub wroga jeden na jednego, istnieje szansa, że twój gracz natknie się na postać tylko przez krótki czas. Dla jednorazowych strażników, których celem życiowym jest rozstrzelanie, może to być znacznie krótki czas. Trudniejsi wrogowie mogą być na ekranie przez kilka minut, gdy ich upadek jest planowany i wykonywany. Kiedy staramy się zrozumieć kogoś w prawdziwym życiu, naturalnie stawiamy się na jego miejscu. Przyglądamy się ich otoczeniu, informacjom, które czerpie ze swojego otoczenia i czynnościom, które wykonują. To samo dzieje się z postaciami z gry. Strażnik stojący w ciemnym pokoju słyszy dźwięk: „Nacisnąłbym włącznik światła", myślimy. Jeśli strażnik tego nie zrobi, możemy założyć, że jest głupi. Jeśli dostrzeżemy kogoś tylko przez krótką chwilę, nie mamy wystarczająco dużo czasu, aby zrozumieć jego sytuację. Jeśli zobaczymy strażnika, który usłyszał hałas, nagle odwrócił się i powoli ruszył w przeciwnym kierunku, zakładamy, że sztuczna inteligencja jest uszkodzona. Strażnik powinien był przejść przez pokój w kierunku hałasu. Gdybyśmy obserwowali ich dłużej, żeby zobaczyć strażnika kierującego się do włącznika światła przy wyjściu, ich działanie byłoby zrozumiałe. Z drugiej strony strażnik może mimo wszystko nie pstryknąć włącznikiem światła i możemy to uznać za oznakę złego wdrożenia. Ale strażnik może wiedzieć, że światło nie działa lub może czekał, aż kolega wsunie

papierosy pod drzwi, i pomyślał, że hałas jest predefiniowanym sygnałem. Gdybyśmy wiedzieli to wszystko, wiedzielibyśmy, że akcja była mimo wszystko inteligentna. Ta sytuacja bez wygranej jest oknem percepcji. Musisz upewnić się, że sztuczna inteligencja postaci odpowiada jej celowi w grze i uwadze, jaką otrzyma od gracza. Dodanie większej ilości sztucznej inteligencji do przypadkowych postaci może przyciągnąć cię do rzadkiego gracza, który spędza godziny na analizowaniu każdego poziomu, sprawdzaniu dziwnych zachowań lub błędów, ale wszyscy inni (w tym wydawca i prasa) mogą pomyśleć, że twoje programowanie było niechlujne.

ZMIANY ZACHOWANIAM

Okno percepcji to nie tylko czas. Pomyśl ponownie o duchach w Pac-Manie. Mogą nie sprawiać wrażenia odczuwania, ale nie robią niczego nie na miejscu. Dzieje się tak, ponieważ rzadko zmieniają zachowanie (najbardziej zauważalna jest ich transformacja, gdy gracz zjada power-up). Za każdym razem, gdy postać w grze zmienia zachowanie, zmiana ta jest znacznie bardziej widoczna niż samo zachowanie. W ten sam sposób, gdy zachowanie postaci powinno się oczywiście zmienić, a nie, zwraca uwagę. Jeśli dwóch strażników stoi i rozmawia ze sobą, a ty zestrzelisz jednego, drugi strażnik nie powinien kontynuować rozmowy! Zmiana zachowania prawie zawsze następuje, gdy gracz jest w pobliżu lub został zauważony. Tak samo jest w grach platformowych, jak w strategii czasu rzeczywistego. Dobrym rozwiązaniem jest zachowanie tylko dwóch zachowań przypadkowych postaci — normalnej akcji i akcji zauważonej przez gracza.

RODZAJ AI W GRACH

Gry zawsze były krytykowane za to, że są źle zaprogramowane w sensie inżynierii oprogramowania: używają sztuczek, tajemnych optymalizacji i niesprawdzonych technologii, aby uzyskać dodatkową szybkość lub zgrabne efekty. Chociaż silniki gier mogą być ponownie wykorzystywane, kod rozgrywki zwykle nie jest (lub przynajmniej nie jest pisany z myślą o tym), a silna presja czasu oznacza, że programiści często robią wszystko, czego potrzebują, aby ukończyć grę. Sztuczna inteligencja w grze nie jest inna. Istnieje duża przepaść między tym, co kwalifikuje się jako sztuczna inteligencja w grach (tj. za co odpowiada programista AI), a tym, co reszta branży programistycznej lub środowiska akademickiego uważa za sztuczną inteligencję. Z mojego doświadczenia wynika, że sztuczna inteligencja w grze to w równym stopniu hakowanie (rozwiązania ad hoc i zgrabne efekty), heurystyka (zasady praktyczne, które działają tylko w większości, ale nie we wszystkich przypadkach) i algorytmy („właściwe" rzeczy). Większość tekstu jest skierowana do tej ostatniej grupy, ponieważ są to techniki, które możemy uogólnić, zbadać analitycznie, użyć w wielu grach i wbudować w silnik AI. Ale rozwiązania ad hoc i heurystyki są równie ważne i mogą tchnąć w postaci tyle samo życia, co najbardziej skomplikowany algorytm.

HACKI

Jest takie powiedzenie: „Jeśli wygląda jak ryba i pływa jak ryba, to prawdopodobnie jest ryba". Rozumiemy zachowanie, replikując je z wystarczającą dokładnością. Jako podejście psychologiczne ma pewien rodowód (jest związany ze szkołą psychologii behawiorystycznej), ale zostało w dużej mierze zastąpione. Ten upadek z mody wpłynął również na psychologiczne podejście do sztucznej inteligencji. W pewnym momencie poznawanie ludzkiej inteligencji przez tworzenie maszyny do jej replikacji było całkiem akceptowalne, ale teraz jest to uważane za kiepską naukę. I nie bez powodu; w końcu budowanie maszyny do umiejętnej gry w szachy wymaga algorytmów, które oceniają miliony pozycji na szachownicy. Istoty ludzkie po prostu nie są do tego zdolne. Z drugiej strony, inżynierom AI nie płaci się za zainteresowanie naturą rzeczywistości lub umysłu; chcemy postaci, które wyglądają dobrze. W większości przypadków oznacza to rozpoczęcie od ludzkich zachowań i próbę wypracowania najłatwiejszego sposobu ich implementacji w oprogramowaniu. Dobra sztuczna inteligencja w grach

zwykle działa w tym kierunku. Deweloperzy rzadko budują świetne nowe algorytmy, a następnie zadają sobie pytanie „więc co mogę z tym zrobić?" Zamiast tego zaczynasz od projektu postaci i stosujesz najbardziej odpowiednie narzędzie, aby uzyskać wynik. Oznacza to, że to, co kwalifikuje się jako sztuczna inteligencja w grze, może być nierozpoznawalne jako technika sztucznej inteligencji. Prosty generator liczb losowych zastosowany rozsądnie może dać wiele wiarygodności. Generowanie liczby losowej nie jest techniką AI jako taką. W większości języków istnieją wbudowane funkcje do uzyskania liczby losowej, więc z pewnością nie ma sensu podawać do tego algorytmu! Ale może działać w zaskakująco wielu sytuacjach. Innym dobrym przykładem kreatywnego rozwoju sztucznej inteligencji jest The Sims. Chociaż pod powierzchnią dzieją się dość skomplikowane rzeczy, wiele zachowań postaci jest komunikowanych za pomocą animacji. Usuń animację postaci, a sztuczna inteligencja wyglądałaby znacznie mniej imponująco. W Gwiezdnych wojnach: Odcinek 1 - Racer postacie, które są zirytowane, dały mały cios innym postaciom. Quake II wprowadził polecenie „gest", które jest teraz używane w szerokiej gamie gier pierwszoosobowych, w których postacie (i gracze) mogą odwrócić wroga. Wszystko to nie wymaga znaczącej infrastruktury AI. Nie potrzebują skomplikowanych modeli poznawczych, uczenia się ani sieci neuronowych. Potrzebują tylko kilku linijek kodu, które zaplanują animację we właściwym czasie. Zawsze szukaj prostych rzeczy, które mogą dać złudzenie inteligencji. Jeśli chcesz angażować postacie emocjonalne, czy jest możliwe dodanie do gry kilku animacji emocji (np. sfrustrowanego pocierania świątyni lub tupnięcia stopą)? Wyzwalanie ich we właściwym miejscu jest znacznie łatwiejsze niż próba przedstawienia stanu emocjonalnego postaci poprzez jej działania. Czy masz portfolio zachowań, z których wybierze postać? Czy wybór będzie wymagał złożonego ważenia wielu czynników? Jeśli tak, warto wypróbować wersję sztucznej inteligencji, która wybiera zachowanie całkowicie losowo (być może z różnym prawdopodobieństwem dla każdego zachowania). Możesz być w stanie odróżnić, ale Twoi klienci mogą nie; więc wypróbuj go, zanim zakodujesz złożoną wersję.

HEURYSTYKA

Heurystyka to praktyczna zasada, przybliżone rozwiązanie, które może działać w wielu sytuacjach, ale jest mało prawdopodobne we wszystkich. Istoty ludzkie cały czas używają heurystyk. Nie staramy się wypracować wszystkich konsekwencji naszych działań. Zamiast tego opieramy się na ogólnych zasadach, które sprawdziły się w przeszłości (lub w równym stopniu nas nauczono, a nawet poddano nas praniu mózgu). Może to być coś tak prostego, jak „jeśli coś stracisz, prześledź swoje kroki, aby tego poszukać" po heurystyki, które rządzą naszymi wyborami życiowymi, takie jak „nigdy nie ufaj sprzedawcy używanych samochodów". Heurystyki zostały skodyfikowane i włączone do niektórych algorytmów w tej książce, a powiedzenie „heurystyka" programiście sztucznej inteligencji często wywołuje wyobrażenie o odnajdywaniu ścieżki lub zachowaniach zorientowanych na cel. Poza tym wiele technik opisanych w tej książce opiera się na heurystyce, która nie zawsze jest jednoznaczna. Istnieje kompromis między szybkością a dokładnością w obszarach takich jak podejmowanie decyzji, ruch i myślenie taktyczne (w tym sztuczna inteligencja w grach planszowych). Kiedy poświęca się dokładność, zwykle zastępuje się wyszukiwanie poprawnej odpowiedzi heurystyką. Szeroki zakres heurystyk można zastosować do ogólnych problemów AI, które nie wymagają konkretnego algorytmu. W naszym odwiecznym Pac-Manie, duchy poruszają się, obierając trasę na skrzyżowaniu, które prowadzi do ich aktualnego celu. Nie próbują obliczyć najlepszej trasy: najkrótszej lub najszybszej. Może to być dość skomplikowane i wymagać odwrócenia się od siebie, a ostatecznie może być zbędne, ponieważ pozycja gracza wciąż się zmienia. Ale praktyczna zasada (ruch w aktualnym kierunku celu) działa przez większość czasu i zapewnia graczowi wystarczającą kompetencję, aby zrozumieć, że duchy w swoim ruchu nie są czysto przypadkowe. W Warcraft (i wielu innych grach RTS, które nastąpiły później) istnieje heurystyka, która przesuwa postać nieco do przodu, aby mogła zaatakować wroga znajdującego się ułamek poza zasięgiem postaci. Chociaż to działało w większości przypadków, nie

zawsze była to najlepsza opcja. Wielu graczy było sfrustrowanych, gdy rozległe struktury obronne zaczęły się poruszać, gdy wrogowie się zbliżyli. Później gry RTS pozwalały graczowi wybrać, czy to zachowanie zostało włączone, czy nie. W wielu grach strategicznych, w tym w grach planszowych, różnym jednostkom lub pionom przypisywana jest pojedyncza wartość liczbowa, aby pokazać, jak „dobre" są. W szachach pionki często otrzymują jeden punkt, gońce i skoczki trzy, wieże pięć, a hetman osiem. To jest heurystyka; zastępuje złożone obliczenia dotyczące możliwości jednostki jedną liczbą. A numer może być wcześniej zdefiniowany przez programistę. Sztuczna inteligencja może określić, która strona jest przed nami, po prostu dodając liczby. W RTS może znaleźć najlepszą ofensywną jednostkę do zbudowania, porównując liczbę z kosztem. Wiele przydatnych efektów można osiągnąć po prostu manipulując liczbą. Nie ma na to algorytmu ani techniki. I nie znajdziesz tego w opublikowanych badaniach AI. Ale to chleb powszedni pracy programisty AI.

Wspólna heurystyka

W kółko pojawia się garść heurystyk. Są dobrym punktem wyjścia przy początkowym rozwiązywaniu problemu.

Najbardziej ograniczone

Biorąc pod uwagę obecny stan świata, należy wybrać jeden przedmiot z zestawu. Wybrany element powinien być tym, który byłby opcją dla najmniejszej liczby stanów. Na przykład oddział postaci atakuje grupę wrogów. Jeden z wrogów nosi rodzaj zbroi, którą może przebić tylko jeden karabin. Jeden członek oddziału ma ten karabin. Kiedy wybierają kogo zaatakować, w grę wchodzi najbardziej ograniczona heurystyka; tylko jeden członek oddziału może zaatakować tego wroga, więc to jest akcja, którą powinni podjąć. Nawet jeśli ich broń byłaby potężniejsza przeciwko innym wrogom, ich koledzy z drużyny powinni poradzić sobie z innymi.

Najpierw zrób najtrudniejszą rzecz

Najtrudniejsza rzecz do zrobienia często ma konsekwencje dla wielu innych działań. Lepiej zrobić to najpierw, niż stwierdzić, że łatwe rzeczy dobrze się układają, ale ostatecznie są zmarnowane. Na przykład armia ma dwa oddziały z pustymi miejscami. Komputer planuje stworzenie pięciu wojowników orków i ogromnego kamiennego trolla. Chce skończyć ze zrównoważonymi składami. Jak powinien przydzielać jednostki do oddziałów? Kamienny Troll wymaga najwięcej slotów, więc jest najtrudniejszy do przydzielenia. Powinien być umieszczony jako pierwszy. Gdyby Orkowie zostali przydzieleni jako pierwsi, byliby zrównoważeni między dwiema drużynami, zostawiając miejsce dla połowy trolla w każdej drużynie, ale trolle nie miały dokąd się udać.

Najpierw wypróbuj najbardziej obiecującą rzecz

Jeśli istnieje wiele opcji dostępnych dla sztucznej inteligencji, często możliwe jest przyznanie każdemu z nich naprawdę surowego i gotowego wyniku. Nawet jeśli ten wynik jest dramatycznie niedokładny, wypróbowanie opcji w malejącej kolejności wyników zapewni lepszą wydajność niż próbowanie rzeczy wyłącznie losowo.

ALGORYTMY

I tak dochodzimy do ostatniej trzeciej części pracy programisty AI: budowania algorytmów wspierających interesujące zachowanie postaci. Hacki i heurystyki zaprowadzą Cię daleko, ale poleganie na nich oznacza, że będziesz musiał ciągle odkrywać koło na nowo. Ogólne elementy sztucznej inteligencji, takie jak ruch, podejmowanie decyzji i myślenie taktyczne, korzystają z wypróbowanych i przetestowanych metod, które można bez końca wykorzystywać ponownie. Ten

tekst dotyczy tego rodzaju technik, a kilka następnych sekcji przedstawia ich dużą liczbę. Pamiętaj tylko, że w każdej sytuacji, w której najlepszym sposobem jest złożony algorytm, prawdopodobnie jest kilka innych, w których prostszy hack lub heurystyka załatwią sprawę.

OGRANICZENIA PRĘDKOŚCI I PAMIĘCI

Największym ograniczeniem pracy programisty AI są fizyczne ograniczenia maszyny. Sztuczna inteligencja gier nie może pozwolić sobie na luksus wielodniowego czasu przetwarzania i terabajtów pamięci. Nie mamy nawet luksusu korzystania z całego procesora i pamięci komputera, na którym działa gra. Inne zadania, takie jak grafika, dźwięk, sieć i dane wejściowe, wymagają miejsca i czasu. W zespołach, w których różne grupy programistów muszą równolegle pracować nad swoimi specjalnościami, zostanie ustalony budżet szybkości i pamięci. Jednym z powodów, dla których techniki sztucznej inteligencji z badań akademickich lub komercyjnych nie znajdują szerokiego zastosowania, jest ich czas przetwarzania lub wymagania dotyczące pamięci. Algorytm, który może być atrakcyjny w prostym demie, może spowolnić grę produkcyjną. W tej sekcji omówiono problemy sprzętowe niskiego poziomu związane z projektowaniem i konstruowaniem kodu AI. Większość tego, co jest tutaj zawarte, to ogólne porady dotyczące całego kodu gry. Jeśli jesteś na bieżąco z aktualnymi problemami związanymi z programowaniem gier i chcesz po prostu zapoznać się ze sztuczną inteligencją, możesz bezpiecznie pominąć tę sekcję.

PROCESY Z PROCESORAMI

Najbardziej oczywistym ograniczeniem wydajności gry jest szybkość procesora, na którym działa. Pierwotnie wszystkie maszyny do gier miały jeden główny procesor, który odpowiadał również za grafikę. Większość sprzętu do gier ma teraz kilka procesorów (zazwyczaj kilka rdzeni przetwarzających na tym samym kawałku krzemu) i dedykowane procesory graficzne do przetwarzania grafiki. Z reguły procesory są szybsze i bardziej elastyczne, podczas gdy GPU jest bardziej równoległy. Kiedy zadanie można podzielić na wiele prostych podzadań, z których wszystkie działają w tym samym czasie, dziesiątki do tysięcy rdzeni przetwarzających na GPU może być o rząd wielkości szybciej niż to samo zadanie uruchamiane sekwencyjnie na procesorze. Sterowniki kart graficznych miały kiedyś „stałe funkcje" potoków, w których kod graficzny był wbudowany w sterownik i można go było modyfikować tylko w ramach wąskich parametrów. Nie można było zrobić wiele poza grafiką na karcie graficznej. Teraz sterowniki obsługują technologie takie jak Vulkan, DirectX 11, CUDA i OpenCL, które umożliwiają wykonywanie kodu ogólnego przeznaczenia na GPU. W rezultacie więcej funkcji zostało przeniesionych na GPU, uwalniając więcej mocy obliczeniowej procesora. Udział czasu przetwarzania przeznaczonego na sztuczną inteligencję wzrastał zrywami i rozpoczynał się w ciągu ostatnich dwóch dekad. W niektórych przypadkach większość obciążenia procesora jest teraz obciążona, a na GPU działa sztuczna inteligencja. Wraz ze wzrostem prędkości procesora jest to oczywiście dobra wiadomość dla programistów AI, którzy chcą stosować bardziej skomplikowane algorytmy, szczególnie w podejmowaniu decyzji i opracowywaniu strategii. Ale chociaż przyrostowe ulepszenia czasu procesora pomagają odblokować nowe techniki, nie rozwiązują podstawowego problemu. Uruchomienie wielu algorytmów AI zajmuje dużo czasu. Kompleksowy system odnajdywania ścieżek może zająć dziesiątki milisekund na każdą postać. Oczywiście w RTS-ie z 1000 znaków nie ma szans na uruchomienie każdej klatki. Złożona sztuczna inteligencja, która działa w grach, musi zostać podzielona na komponenty o niewielkich rozmiarach, które można rozłożyć na wiele klatek. Rozdział poświęcony zarządzaniu zasobami pokazuje, jak to osiągnąć. Zastosowanie tych technik do wielu długotrwałych algorytmów sztucznej inteligencji może wprowadzić je w sferę praktyczności.

Obawy niskiego poziomu

Jedną z dużych zmian w branży w ciągu ostatnich 10 lat było odejście od C++, które ma hegemonię nad programowaniem gier. Teraz zachowanie postaci, logika gry i sztuczna inteligencja są często pisane w językach wyższego poziomu, takich jak C#, Swift, Java, a nawet w językach skryptowych. Jest to istotne, ponieważ języki te zapewniają programiście mniejsze możliwości mikrozarządzania charakterystyką wydajności ich kodu. W C++ nadal pracują programiści AI, którzy nadal potrzebują dobrej znajomości charakterystyki wydajności procesorów „gołego metalu", ale z mojego niedawnego doświadczenia wynika, że programiści tacy są zwykle specjalistami niskiego poziomu pracującymi nad silnikami AI: portfolio zaprojektowanych funkcji do ponownego wykorzystania w wielu grach. SIMD (pojedyncza instrukcja, wiele danych) to zestaw rejestrów na nowoczesnym sprzęcie wystarczająco duży, aby pomieścić kilka liczb zmiennoprzecinkowych. Do tych rejestrów można zastosować operatory matematyczne, co skutkuje równoległym uruchomieniem tego samego kodu na wielu fragmentach danych. Może to radykalnie przyspieszyć działanie kodu, w szczególności rozumowanie geometryczne. Chociaż procesory mają dedykowane rejestry dla SIMD, zapewniają one największe przyspieszenie kodu, który zwykle pasuje do GPU. Optymalizacja pod kątem SIMD na CPU jest często nadmiarowa, gdy kod można przenieść na GPU. Procesory superskalarne mają jednocześnie aktywnych kilka ścieżek wykonania. Kod jest dzielony na części, które mają być wykonywane równolegle, a wyniki są następnie ponownie łączone w wynik końcowy. Gdy wynik jednego potoku zależy od drugiego, może to obejmować albo czekanie, albo zgadywanie, jaki może być wynik, i ponawianie pracy, jeśli okaże się ona błędna (tzw. przewidywanie rozgałęzień). W ostatniej dekadzie wielordzeniowe procesory, w których kilka niezależnych procesorów umożliwia równoległe działanie różnych wątków, stały się niemal wszechobecne. Chociaż każdy rdzeń może nadal być superskalarny, jest to obecnie w dużej mierze traktowane jako szczegół zakulisowy, nieistotny dla programistów AI. Lepsze przyspieszenie można osiągnąć, koncentrując się na umożliwieniu zrównoleglenia kodu AI, zamiast martwić się szczegółami przewidywania gałęzi. Kod AI może skorzystać z tego paralelizmu, uruchamiając sztuczną inteligencję dla różnych postaci w różnych wątkach lub uruchamiając całą sztuczną inteligencję w innym wątku do innych systemów gier. Te wątki będą następnie wykonywane równolegle na różnych rdzeniach, jeśli będą dostępne. Wiele wątków wykonujących to samo (na przykład jedna postać uruchamiająca swoją sztuczną inteligencję w każdym wątku) jest często bardziej wydajny, ponieważ łatwiej jest upewnić się, że wszystkie procesory są używane z taką samą wydajnością, i bardziej elastycznym, ponieważ skaluje się bez ponownego równoważenia do sprzęt o różnej liczbie rdzeni. Kiedy sztuczna inteligencja rutynowo musiała uważać na każdy używany cykl procesora, powszechne było unikanie klas wirtualnych w C++, z ich narzutem na wywołanie funkcji wirtualnych. Oznaczało to unikanie polimorfizmu obiektowego, gdy tylko było to możliwe. Wywołanie funkcji wirtualnej przechowuje lokalizację pamięci, w której funkcja jest zaimplementowana w zmiennej (w strukturze zwanej tabelą funkcji lub vtable). Wywołanie funkcji wymaga zatem wyszukania zmiennej w czasie wykonywania, a następnie wyszukania lokalizacji określonej przez zmienną. Chociaż to dodatkowe wyszukiwanie zajmuje niewiele czasu, może znacząco wchodzić w interakcje z predyktorem rozgałęzień i pamięcią podręczną procesora. Z tego powodu funkcje wirtualne, a co za tym idzie polimorfizm, miały złą reputację. Reputacja, która w ciągu ostatnich 10 lat w dużej mierze osłabła. Teraz silniki gier, takie jak Unity, Unreal, Lumberyard i Godot, zakładają, że logika gry będzie polimorficzna.

OBAWY PAMIĘCI

Większość algorytmów AI nie wymaga dużej ilości pamięci RAM, często zaledwie kilku, nawet kilkudziesięciu megabajtów. To niewielkie wymaganie w zakresie pamięci masowej, łatwo osiągalne na skromnych urządzeniach mobilnych, jest wystarczające dla ciężkich algorytmów, takich jak analiza terenu i odnajdywanie ścieżek. Gry online dla wielu graczy (MMOG) zazwyczaj wymagają znacznie więcej pamięci dla swoich większych światów, ale są uruchamiane na farmach serwerów, na których można zainstalować wystarczającą ilość pamięci (nawet wtedy jesteśmy tylko mówiącymi gigabajtami

pamięci RAM, rzadko więcej). Ogromne światy są zwykle podzielone na osobne sekcje, w przeciwnym razie postacie są ograniczone do określonych obszarów, co dodatkowo zmniejsza wymagania dotyczące pamięci AI. Zatem to nie ilość pamięci jest zwykle czynnikiem ograniczającym, ale sposób jej wykorzystania. Alokacja i spójność pamięci podręcznej to problemy związane z pamięcią, które wpływają na wydajność. Oba mogą wpływać na implementację algorytmu AI.

Przydział i wywóz śmieci

Alokacja to proces żądania pamięci, w której można umieścić dane. Kiedy ta pamięć nie jest już potrzebna, mówi się, że jest zwalniana lub cofana. Alokacja i cofnięcie alokacji są stosunkowo szybkie, o ile dostępna jest pamięć. Języki niskiego poziomu, takie jak C, wymagają od programisty ręcznego zwalniania pamięci. Języki takie jak C++ i Swift, gdy pamięć jest przydzielana dla określonego obiektu, zapewniają „liczenie referencji". Przechowuje liczbę miejsc, które wiedzą o istnieniu obiektu. Gdy obiekt nie jest już przywoływany, licznik spada do 0, a pamięć jest zwalniana. Niestety, oba te podejścia mogą oznaczać, że pamięć, która powinna być uwolniona, nigdy nie jest uwalniana. Albo programista zapomni zwolnić ręcznie, albo istnieje cykliczny zestaw odniesień, tak że ich liczniki nigdy nie spadają do 0. Wiele języków wyższego poziomu implementuje wyrafinowane algorytmy do zbierania tych „śmieci", tj. wolnej pamięci, która nie jest już przydatna . Niestety wywóz śmieci może być kosztowny. W językach takich jak C#, szczególnie w środowisku uruchomieniowym mono, w którym działa silnik gry Unity, wyrzucanie elementów bezużytecznych może być wystarczająco wolne, aby opóźnić ramkę renderowania, powodując zacinanie się obrazu. Jest to nie do przyjęcia dla większości programistów. W rezultacie implementacja algorytmów sztucznej inteligencji dla języków wyższego poziomu często wiąże się z próbą nieprzydzielania i cofania alokacji obiektów, gdy poziom jest uruchomiony. Dane wymagane dla całego poziomu są rezerwowane na początku poziomu i udostępniane dopiero po jego zakończeniu. Niektóre algorytmy zakładają, że w dowolnym momencie można utworzyć nowe obiekty, które znikną, gdy nie będą już potrzebne. Na platformie z czasochłonnym wyrzucaniem elementów bezużytecznych może być ważna modyfikacja tych implementacji. Na przykład: kilka algorytmów wyszukiwania ścieżek tworzy i przechowuje dane dla każdej lokalizacji na mapie, gdy ta lokalizacja jest brana pod uwagę po raz pierwszy. Po ukończeniu ścieżki nie są potrzebne żadne dane dotyczące lokalizacji pośredniej. Implementacja przyjazna dla zbierania śmieci może utworzyć pojedynczy obiekt odnajdujący ścieżki, zawierający dane dla każdej lokalizacji na mapie. Ten sam obiekt jest wywoływany za każdym razem, gdy wymagane jest odnajdywanie ścieżek, i wykorzystuje wstępnie przydzielone dane o lokalizacji, których potrzebuje, a resztę ignoruje. Ta implementacja sama w sobie będzie nieco bardziej skomplikowana i może być znacznie bardziej złożona, jeśli wiele znaków, które potrzebują znajdowania ścieżek, musi stać w kolejce, aby użyć jednego obiektu do znajdowania ścieżek. Aby uniknąć komplikowania podstawowego algorytmu, ta książka przedstawia je w najprostszej formie: bez względu na alokację.

Pamięć podręczna

Sam rozmiar pamięci nie jest jedynym ograniczeniem wykorzystania pamięci. Czas potrzebny na uzyskanie dostępu do pamięci z pamięci RAM i przygotowanie jej do użycia przez procesor jest znacznie dłuższy niż czas, w którym procesor wykonuje swoje operacje. Gdyby procesory musiały polegać na głównej pamięci RAM, byłyby stale w zastoju w oczekiwaniu na dane. Wszystkie współczesne procesory wykorzystują co najmniej jeden poziom pamięci podręcznej: kopię pamięci RAM przechowywanej w procesorze, którą można bardzo szybko manipulować. Pamięć podręczna jest zwykle pobierana na stronach; cała sekcja pamięci głównej jest przesyłana strumieniowo do procesora. Można nim wtedy dowolnie manipulować. Gdy procesor wykona swoją pracę, pamięć podręczna jest przesyłana z powrotem do pamięci głównej. Procesor zazwyczaj nie może działać w pamięci głównej: cała potrzebna pamięć musi znajdować się w pamięci podręcznej. System operacyjny może dodać do

tego dodatkową złożoność, ponieważ żądanie pamięci może musieć przejść przez procedurę systemu operacyjnego, która tłumaczy żądanie na żądanie pamięci rzeczywistej lub wirtualnej. Może to wprowadzić dalsze ograniczenia, ponieważ dwa bity pamięci fizycznej o podobnym zmapowanym adresie mogą nie być dostępne w tym samym czasie (tzw. awaria aliasingu). Wiele poziomów pamięci podręcznej działa tak samo, jak pojedyncza pamięć podręczna. Duża ilość pamięci jest pobierana do pamięci podręcznej najniższego poziomu, jej podzbiór jest pobierany do każdej pamięci podręcznej wyższego poziomu, a procesor zawsze działa tylko na najwyższym poziomie. Jeśli algorytm wykorzystuje dane rozproszone po pamięci, jest mało prawdopodobne, aby w pamięci podręcznej co chwilę znajdowała się właściwa pamięć. Te chybienia w pamięci podręcznej są kosztowne z czasem. Procesor musi pobrać do pamięci podręcznej cały nowy fragment pamięci na jedną lub dwie instrukcje, a następnie przesłać to wszystko z powrotem i zażądać kolejnego bloku. Dobry system profilowania pokaże, kiedy występują błędy w pamięci podręcznej. Z mojego doświadczenia wynika, że nawet w językach, które nie dają kontroli nad układem pamięci, dramatyczne przyspieszenie można osiągnąć, upewniając się, że wszystkie dane potrzebne do jednego algorytmu są przechowywane w tym samym miejscu, w tych samych kilku obiektach. Tutaj, dla ułatwienia zrozumienia, użyłem stylu zorientowanego obiektowo do rozmieszczenia danych. Wszystkie dane dotyczące konkretnego obiektu gry są przechowywane razem. Może to nie być najbardziej wydajne rozwiązanie w zakresie pamięci podręcznej. W grze z 1000 znaków może być lepiej trzymać wszystkie ich pozycje razem w tablicy, więc algorytmy wykonujące obliczenia na podstawie lokalizacji nie muszą ciągle przeskakiwać pamięci. Podobnie jak w przypadku wszystkich optymalizacji, profilowanie jest wszystkim, ale ogólny poziom wydajności można osiągnąć, programując z myślą o spójności danych.

PLATFORMY

Wraz z centralizacją branży wokół kilku silników gier, różnice między platformami mają mniejszy wpływ na projektowanie AI niż kiedyś. Na przykład programiści grafiki mogą nadal martwić się o konsolę lub urządzenia mobilne. Ale programowanie AI wydaje się być bardziej ogólne. W tej sekcji omówię każdą z głównych platform gier, podkreślając wszelkie problemy specyficzne dla kodu AI.

PC

Komputery PC mogą być najpotężniejszymi maszynami do gier, a zagorzali gracze kupują drogi sprzęt wysokiej klasy. Ale mogą być frustrujące dla programistów z powodu braku spójności. Tam, gdzie konsola ma stały sprzęt (lub przynajmniej stosunkowo niewiele odmian), istnieje oszałamiająca liczba różnych konfiguracji dla komputerów PC. Istnieje ogromna różnica między maszyną z parą najwyższej klasy kart graficznych, napędami SSD i szybką pamięcią, a niedrogim komputerem PC ze zintegrowaną grafiką. Sprawy są prostsze niż były: deweloperzy niskiego poziomu polegają na interfejsach programowania aplikacji (API), takich jak Vulkan i DirectX, aby odizolować je od większości specyfiki sprzętu, ale gra nadal musi wykrywać obsługę funkcji i szybkość oraz odpowiednio dostosowywać. Deweloperzy pracujący w silnikach takich jak Unity i Unreal mają to jeszcze łatwiejsze, ale nadal mogą potrzebować korzystać z wbudowanego wykrywania funkcji, aby upewnić się, że ich gra działa dobrze na wszystkich systemach. Praca z komputerami polega na tworzeniu oprogramowania, które można skalować od ograniczonego systemu zwykłego gracza po najnowocześniejszy sprzęt dla zagorzałych fanów. W przypadku grafiki to skalowanie może być dość modułowe; na przykład w przypadku maszyn o niskiej specyfikacji wyłączamy zaawansowane funkcje renderowania. Można użyć prostszego algorytmu cieni lub fizycznie oparte shadery można zastąpić prostym mapowaniem tekstur. Zmiana zaawansowania grafiki zwykle nie zmienia rozgrywki. Sztuczna inteligencja jest inna. Jeśli sztuczna inteligencja ma mniej czasu na pracę, jak powinna zareagować? Może próbować wykonywać mniej pracy. Jest to w rzeczywistości równoznaczne z posiadaniem głupszej sztucznej inteligencji i może wpłynąć na poziom trudności gry. Łatwiejsza gra na maszynach o niższej specyfikacji jest

prawdopodobnie nie do przyjęcia. Podobnie, jeśli spróbujemy wykonać tę samą ilość pracy, może to potrwać dłużej. Może to oznaczać niższą liczbę klatek na sekundę lub więcej klatek między postaciami podejmującymi decyzje. Postacie, które wolno reagują, są często łatwiejsze do grania i mogą powodować te same problemy z QA. Rozwiązaniem używanym przez większość programistów jest kierowanie sztucznej inteligencji do najniższego wspólnego mianownika: maszyny o minimalnej specyfikacji wymienionej w dokumencie projektu technicznego. Czas AI w ogóle nie skaluje się z możliwościami maszyny. Szybsze maszyny po prostu wykorzystują proporcjonalnie mniej swojego budżetu przetwarzania na sztuczną inteligencję. Istnieje jednak wiele gier, w których skalowalna sztuczna inteligencja jest możliwa. Wiele gier wykorzystuje sztuczną inteligencję do sterowania postaciami otoczenia: przechodniami spacerującymi chodnikiem, członkami tłumu wiwatującego na wyścigu lub stadami ptaków rojących się na niebie. Ten rodzaj sztucznej inteligencji jest dowolnie skalowalny: można użyć większej liczby znaków, gdy dostępny jest czas procesora.

Konsola

Konsole mogą być prostsze w obsłudze niż komputer. Znasz dokładnie maszynę, na którą się kierujesz, i zazwyczaj możesz zobaczyć kod działający na maszynie docelowej. Nie trzeba się martwić o przyszłość nowego sprzętu lub ciągle zmieniających się wersji interfejsów API. Deweloperzy pracujący z technologią nowej generacji często nie mają dokładnych specyfikacji ostatecznej maszyny lub niezawodnej platformy sprzętowej (wstępne zestawy deweloperskie to często niewiele więcej niż dedykowany emulator), ale większość deweloperów konsol ma dość ustalony cel. Proces listy kontrolnej wymagań technicznych (TRC), za pomocą którego producent konsoli określa minimalne standardy działania gry, służy do naprawy takich rzeczy, jak liczba klatek na sekundę (chociaż różne terytoria mogą się różnić - na przykład PAL i NTSC). Oznacza to, że budżety AI można zablokować w postaci stałej liczby milisekund. To z kolei znacznie ułatwia ustalenie, jakich algorytmów można użyć i ustalenie celu optymalizacji (pod warunkiem, że budżet nie zostanie zmniejszony na ostatnim etapie, aby zrobić miejsce dla najnowszej techniki graficznej stosowanej w grze konkurencji). Te same silniki gier, które były używane w konsolach przeznaczonych do tworzenia na PC, dzięki czemu tworzenie międzyplatformowe jest znacznie łatwiejsze niż w przeszłości. Na szczęście niewielu twórców AI tworzących gry pracuje teraz z niskopoziomowymi szczegółami konkretnej konsoli. Prawie cały kod niskopoziomowy jest obsługiwany przez silniki lub oprogramowanie pośredniczące.

Urządzenia mobilne

Apple wypuścił iPhone'a w 2007 roku, zapoczątkowując rewolucję w grach tak wielką, jak wszystko inne od konsol domowych z lat 80-tych. W 2006 roku, gry mobilne składały się z dedykowanych konsol przenośnych, takich jak PlayStation Portable (PSP) i postęp GameBoy firmy Nintendo. Teraz prawie 100% rynku to telefony i tablety. W przestrzeni dwie platformy: Apple z urządzeniami iOS (iPhone, iPad, iPod Touch) oraz Android. Do niedawna były one bardzo różne i wymagały kodowania gier dla każdego z osobna. Chociaż oba mogą używać języków niskiego poziomu, takich jak C i C ++, w przypadku języków wyższego poziomu Apple zachęca do używania Swift (wcześniej używał Objective-C) i Android Java (lub języków, które kompilują się do kodu bajtowego Java, takich jak Kotlin) . Zarówno główne silniki gier pod względem udziału w rynku (Unreal i Unity), jak i wielu mniejszych konkurentów (np. Godot) obsługują platformy mobilne tym samym kodem gry, dzięki czemu wdrożenie specyficzne dla platformy jest niepotrzebne. Nastąpiła duża zmiana w kierunku programistów mobilnych pracujących na wielu platformach, korzystających z tych narzędzi. Biorąc pod uwagę platformę Steam jako opłacalny rynek gier mobilnych działających na PC, myślę, że nie ma wątpliwości, że ten trend wkrótce stanie się niemal wszechobecny. Smartfony zdolne do uruchamiania gier to potężne maszyny, porównywalne z konsolami ostatniej generacji i komputerami PC w wieku 5-10 lat. Nie ma już żadnej praktycznej różnicy między rodzajami sztucznej inteligencji, które można uruchomić na komputerze PC

lub konsoli, a tymi, które można uruchomić na urządzeniach mobilnych. Telefony mogą wymagać prostszej grafiki lub mniejszych rozmiarów tłumu, ale jeśli chodzi o algorytmy, teraz obowiązują te same rzeczy.

Rzeczywistość wirtualna i rozszerzona

Na początku 2019 r., rzeczywistość wirtualna i rozszerzona są niezwykle rozreklamowane i stanowią niewielką część rynku gier. Technologia i rynek podlegają szybkim zmianom i poza ogólnikami, niewiele, co można by teraz powiedzieć, byłoby prawdą za dwa lata. Rzeczywistość wirtualna (VR) próbuje zanurzyć gracza w świecie gry, zapewniając stereoskopowy punkt widzenia 3D. W zależności od sprzętu ruch gracza może również zostać wykryty i włączony jako ruch w grze. VR wymaga renderowania oddzielnych widoków sceny dla każdego oka, a aby uniknąć choroby lokomocyjnej, zazwyczaj celem jest uzyskanie większej liczby klatek na sekundę (na przykład 90 kl./s). Do tego momentu większość urządzeń rzeczywistości wirtualnej było wyświetlanych przywiązanych do istniejących maszyn do gier, takich jak komputer PC (Oculus Rift i Vive), konsola (PlayStation VR) lub telefon (Gear VR). Firmy zaczynały wypuszczać samodzielne produkty VR, oparte na procesorach mobilnych, o mniej więcej podobnej wydajności do telefonów z wyższej półki. Rzeczywistość rozszerzona (AR) wykorzystuje półprzezroczyste wyświetlacze do dodawania elementów generowanych komputerowo do rzeczywistego świata. Chociaż firma Microsoft wydała zestaw deweloperski na początku 2016 r., wersja konsumencka nie pojawiła się jeszcze. Magic Leap wypuścił swój produkt w 2018 roku, ale popyt był ograniczony. Rzeczywistość rozszerzona może odnosić się również do gier wykorzystujących kamerę w telefonie komórkowym i dodawania elementów generowanych komputerowo do uchwyconych obrazów. W tym sensie Pokémon Go , na przykład, jest uważany za grę z rozszerzoną rzeczywistością, ale nie wymaga specjalistycznego sprzętu. Podczas gdy wizualna prezentacja gier VR może być niekonwencjonalna, logika gry rzadko jest taka. Większość komercyjnych silników gier obsługuje VR, AR na urządzeniach mobilnych za pośrednictwem kamery i może oferować sprzętową obsługę AR, gdy produkty zostaną ogłoszone. Gry VR i AR są na tyle podobne w konstrukcji, że nie potrzebują nietypowych algorytmów AI. Okaże się, czy platformy te otwierają nowe możliwości projektowe. Zobaczymy również, czy platformy te staną się znaczącą częścią branży.

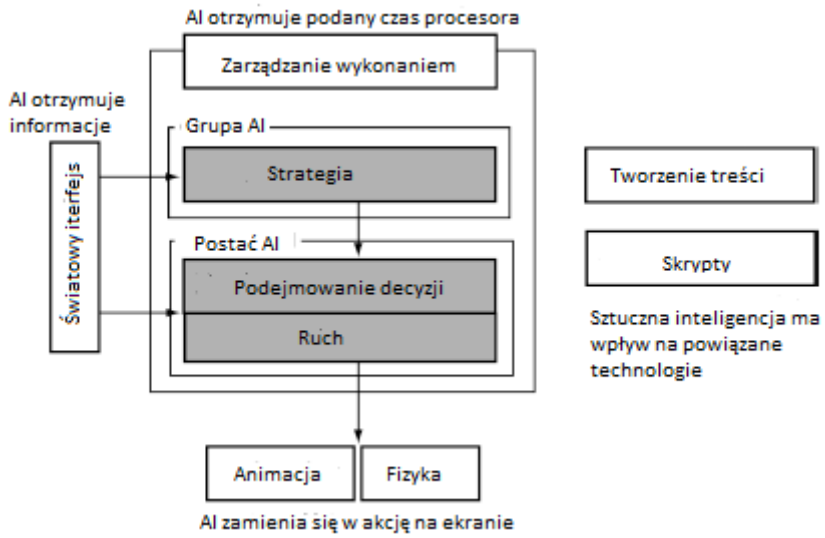
SILNIK AI

Kiedyś gra była w większości budowana od podstaw. Niektóre fragmenty kodu zostały przeciągnięte z poprzednich projektów, a niektóre zostały przerobione i ponownie użyte, ale większość była nowa. Kilka firm używało tego samego podstawowego kodu do pisania wielu gier, o ile gry miały podobny styl i gatunek. Na przykład silnik SCUMM firmy LucasArts był stopniowo ewoluującym silnikiem gier używanym do zasilania wielu gier przygodowych typu „wskaż i kliknij”. Od tego czasu silniki gier stały się wszechobecne, spójną platformą techniczną, na której zbudowano wiele gier. Procedury niskiego poziomu (takie jak rozmowa z systemem operacyjnym, ładowanie tekstur, formaty plików modeli itp.) są wspólne dla wszystkich tytułów, zestaw narzędzi jest dostępny dla szerokiej gamy gier (np. grafika 2D, grafika 3D, praca w sieci), a na koniec zapewniono interfejsy, aby dodać kod specyficzny dla gry na górze. Początkowo silniki te należały do pojedynczych firm, ale z czasem tylko największe firmy mogły sobie pozwolić na unowocześnianie swoich silników. Obecnie wielcy programiści często licencjonują silniki komercyjne. Zmienił się również sposób rozwoju sztucznej inteligencji. Początkowo sztuczna inteligencja była napisana dla każdej gry, a czasem dla każdej postaci. Obecnie rośnie tendencja do posiadania ogólnych procedur sztucznej inteligencji, wbudowanych w silnik gry, dostępnych do licencjonowania jako komercyjne dodatki lub tworzonych i ponownie używanych przez programistę. Pozwala to na projektowanie poszczególnych postaci przez edytorów poziomów, projektantów gier lub artystów technicznych. Struktura silnika jest stała, a sztuczna inteligencja każdej postaci łączy elementy w odpowiedni sposób. Algorytmy opisane w tej książce są udostępniane niespecjalistom za

pośrednictwem graficznych interfejsów. Przeciąganie pól i linii pozwala każdemu na stworzenie skończonej maszyny stanów lub drzewa zachowań. Tak więc budowanie silnika gry obejmuje tworzenie narzędzi AI, które można łatwo ponownie wykorzystać, połączyć i zastosować w interesujący sposób. Aby to wesprzeć, potrzebujemy struktury AI, która ma sens w wielu gatunkach.

STRUKTURA SILNIKA AI

Iistnieje kilka podstawowych udogodnień, które muszą być dostępne dla ogólnego systemu sztucznej inteligencji. Są one zgodne z modelem AI przedstawionym na rysunku .



Po pierwsze, musimy mieć pewien rodzaj infrastruktury w dwóch kategoriach: ogólny mechanizm zarządzania zachowaniami sztucznej inteligencji (decydowanie, które zachowanie zostanie uruchomione, kiedy itd.) oraz system połączeń światowych do pobierania informacji do sztucznej inteligencji. Każdy algorytm AI musi honorować te mechanizmy. Po drugie, musimy mieć środki, aby zmienić wszystko, co chce zrobić sztuczna inteligencja, w działanie na ekranie. Składa się on ze standardowych interfejsów do ruchu i kontrolera animacji, który może zmieniać żądania, takie jak „pociągnij dźwignię 1” lub „podejdź ukradkiem do pozycji x; y” do działania. Po trzecie, standardowa struktura zachowania musi służyć jako łącznik między nimi. Jest prawie pewne, że będziesz musiał napisać jeden lub dwa algorytmy AI dla każdej nowej gry. To, że cała sztuczna inteligencja jest dostosowana do tej samej struktury, ogromnie w tym pomaga. Nowy kod może być opracowywany, gdy gra jest uruchomiona, a nowa sztuczna inteligencja może po prostu zastąpić zachowania zastępcze, gdy będzie gotowa. Wszystko to należy wcześniej przemyśleć. Struktura musi być na swoim miejscu, zanim zagłębisz się w kodowanie AI. Później omówimy technologie wsparcia, które są pierwszą rzeczą, którą należy zaimplementować w silniku AI. Poszczególne techniki mogą się wtedy włączyć. Silniki gier robią to za ciebie, ale nie wszystkie. Każdy silnik ma swój własny mechanizm zapewniający działanie kodu, często w postaci klas bazowych, z których należy czerpać. Ale może być konieczne zapewnienie bardziej precyzyjnej kontroli: nie każda postać potrzebuje swojej sztucznej inteligencji w każdej klatce. Zapewniają również standardowe mechanizmy planowania animacji, ale rzadziej ruchu postaci. I mogą dostarczać podstawowych narzędzi do ustalenia, która postać wie co (takich jak wbudowana linia wzroku lub sprawdzanie stożka wzroku), ale będą potrzebować niestandardowych implementacji dla czegokolwiek bardziej złożonego. O ile nie zamierzasz używać bardzo prostych technik, będziesz musiał stworzyć trochę infrastruktury i ewentualnie kilka narzędzi w edytorze do obsługi. Istnieją techniki, które omówię, które mogą działać samodzielnie, a wszystkie algorytmy są dość niezależne. W przypadku wersji demonstracyjnej lub prostej gry może wystarczyć użycie tej techniki. Ale dobra

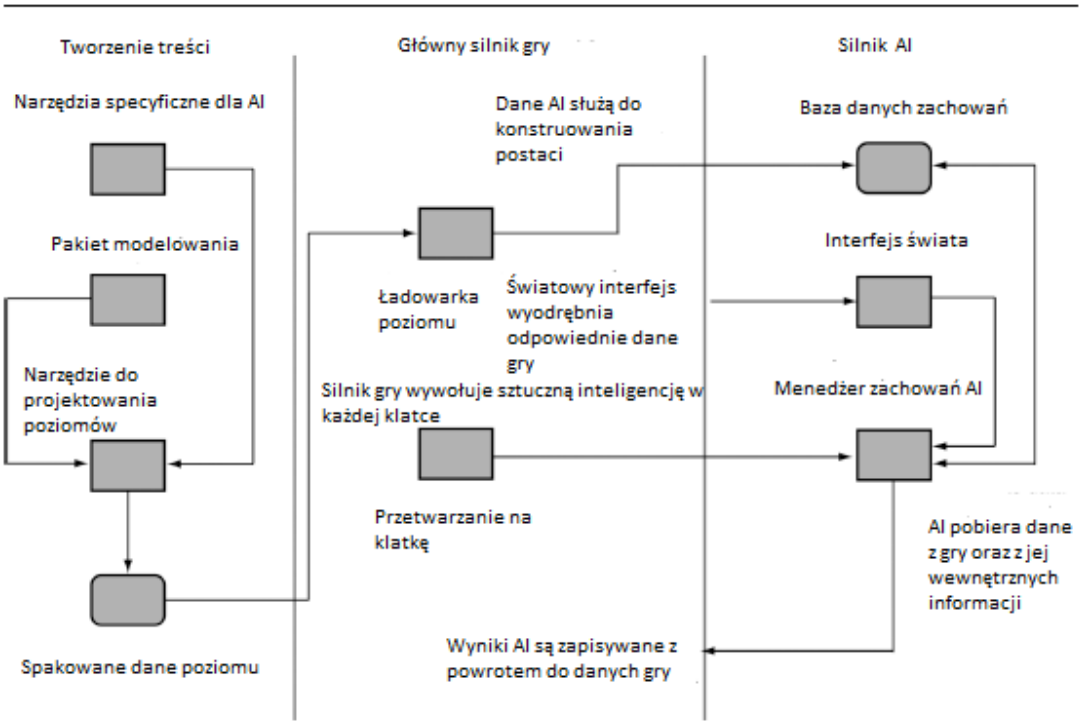
struktura sztucznej inteligencji pomaga promować ponowne użycie i skraca czas debugowania i programowania.

PROBLEMY Z NARZĘDZIAMI

Kompletny silnik AI będzie miał centralną pulę algorytmów AI, które można zastosować do wielu postaci. Definicja sztucznej inteligencji konkretnej postaci będzie zatem składać się z danych (które mogą obejmować skrypty w niektórych językach skryptowych), a nie ze skompilowanego kodu. Dane określają, w jaki sposób postać jest składana: jakie techniki będą używane oraz w jaki sposób te techniki są parametryzowane i łączone. Te dane muszą skądś pochodzić. Dane można tworzyć ręcznie, ale nie jest to lepsze niż ręczne pisanie AI za każdym razem. Elastyczne narzędzia zapewniają, że artyści i projektanci mogą tworzyć zawartość w łatwy sposób, jednocześnie umożliwiając wstawianie zawartości do gry bez ręcznej pomocy. Są one zwykle tworzone jako niestandardowe tryby w edytorze silnika gry: narzędzie do ustawiania zasad podejmowania decyzji przez postać lub nakładka na poziom służąca do oznaczania taktycznych lokalizacji lub miejsc, których należy unikać. Konieczność ujawnienia narzędzi wyszukiwania ma swój wpływ na wybór technik AI. Łatwo jest ustawić zachowania, które zawsze działają w ten sam sposób. Zachowania sterujące (omówione w rozdziale 3) są dobrym przykładem: wydają się być bardzo proste, łatwo je sparametryzować (z fizycznymi możliwościami postaci) i nie zmieniają się z postaci na postać. Trudniej jest używać zachowań, które mają wiele warunków, w których postać musi ocenić szczególne przypadki. Wiele z tych, o których mowa w rozdziale 5, działa w ten sposób. Te oparte na drzewie (drzewa decyzyjne, drzewa zachowań) są łatwiejsze do przedstawienia wizualnego. Z drugiej strony system oparty na regułach musi mieć zdefiniowane skomplikowane reguły dopasowywania. Kiedy są one obsługiwane w narzędziu zazwyczaj wyglądają jak kod programu, ponieważ język programowania jest najbardziej naturalnym sposobem ich wyrażenia.

SKŁADAJĄC TO WSZYSTKO RAZEM

Ostateczna struktura silnika AI może wyglądać podobnie do poniższego rysunku. Dane są tworzone w narzędziu (modelowanie lub edytor poziomów), które są następnie pakowane do wykorzystania w grze. Po załadowaniu poziomu zachowania AI gry są tworzone na podstawie danych poziomu i rejestrowane w silniku AI. Podczas rozgrywki główny kod gry wywołuje silnik sztucznej inteligencji, który aktualizuje zachowania, pozyskuje informacje z interfejsu świata i ostatecznie stosuje ich wyniki do danych gry. Konkretne stosowane techniki zależą w dużej mierze od gatunku rozwijanej gry. W tej książce zobaczymy szeroki wachlarz technik dla wielu różnych gatunków. Rozwijając sztuczną inteligencję w grze, musisz zastosować podejście mieszane i dopasowujące, aby uzyskać zachowania, których szukasz. Ostatnia część książki daje kilka wskazówek na ten temat; przygląda się, jak sztuczna inteligencja w grach z głównych gatunków jest łączona kawałek po kawałku.


Tworzenie treści
Główny silnik gry
Silnik AI
Narzędzia specyficzne dla AI
Dane AI służą do konstruowania postaci
Baza danych zachowań
Pakiet modelowania
Interfejs świata
Ładowarka poziomu
Światowy interfejs wyodrębnia odpowiednie dane gry
Narzędzie do projektowania poziomów
Silnik gry wywołuje sztuczną inteligencję w każdej klatce
Menedżer zachowań AI
Przetwarzanie na klatkę
AI pobiera dane z gry oraz z jej wewnętrznych informacji
Wyniki AI są zapisywane z powrotem do danych gry
Spakowane dane poziomu