AI Gry

Zanim przejdziemy do szczegółów z konkretnymi technikami i algorytmami, warto poświęcić trochę czasu na zastanowienie się nad tym, czego potrzebujemy od sztucznej inteligencji naszej gry. W tej części omówimy ogólne problemy związane z AI gry: jakie podejścia działają, co muszą wziąć pod uwagę i jak je wszystkie połączyć.

Złożoność

Powszechnym błędem jest myślenie, że im bardziej złożona sztuczna inteligencja w grze, tym lepiej postacie będą wyglądać dla gracza. Tworzenie dobrej sztucznej inteligencji polega na dopasowaniu właściwych zachowań do właściwych algorytmów. Znajdziesz się oszałamiający zestaw technik, a właściwa nie zawsze jest najbardziej oczywistym wyborem. Niezliczone przykłady trudnych do wdrożenia, złożonych sztucznej inteligencji wyglądają głupio. Równie dobrze zastosowana bardzo prosta technika może być idealna.

Kiedy proste rzeczy wyglądają dobrze

Wspomnieliśmy o Pac-Manie [Midway Games West, Inc., 1979], jednej z pierwszych gier o dowolnej postaci AI. AI ma dwa stany: jeden normalny, gdy gracz zbiera pipsy, i drugi stan, gdy gracz zjadł energię i jest gotów do zemsty. W swoim normalnym stanie każdy z czterech duchów (lub potworów) porusza się w linii prostej, aż dojdzie do skrzyżowania. Na skrzyżowaniu pół losowo wybierają trasę, którą należy przejść do następnej. Każdy duch decyduje się albo wybrać trasę, która jest w kierunku gracza (obliczona na podstawie prostego przesunięcia do lokalizacji gracza, bez szukania ścieżki w pracy), albo wybrać losową trasę. Wybór zależy od ducha: każdy ma inne prawdopodobieństwo zrobienia jednego lub drugiego. Jest to tak proste, jak tylko możesz sobie wyobrazić AI. Prościej, a duchy byłyby albo bardzo przewidywalne (gdyby zawsze się znajdowały), albo czysto losowe. Połączenie tych dwóch zapewnia świetną rozgrywkę. W rzeczywistości różne uprzedzenia każdego ducha wystarczają, aby uczynić z nich wszystkich czterech znaczącą siłę przeciwstawną - do tego stopnia, że AI do dziś otrzymuje komentarze.

Kiedy złożone rzeczy wyglądają źle

Oczywiście może się zdarzyć coś przeciwnego. Grą, na którą wielu z niecierpliwością czekało, była Herdy Gerdy [Core Design Ltd., 2002], jedna z gier Sony, w której reklamowano nową rozgrywkę i możliwości sprzętu PlayStation 2 przed jego uruchomieniem. Ta gra jest grą pasterską. Ekosystem postaci jest obecny na poziomie gry. Gracz musi gromadzić osobniki różnych gatunków w odpowiednich kojcach. Herding był używany wcześniej i od tego czasu stanowi element większej gry, ale w Herdy Gerdy stanowiła całą rozgrywkę. Niestety postacie zaniedbały podstawy ruchu sztucznej inteligencji. Łatwo było ich złapać w scenerii, a wykrycie kolizji mogło sprawić, że utknęli w nieodwracalnych miejscach. Rzeczywisty efekt był frustracją.

W przeciwieństwie do Herdy Gerdy, Black and White [Lionhead Studios Ltd., 2001] osiągnęły znaczący sukces sprzedażowy. Ale miejscami cierpiał również z powodu złej sztucznej inteligencji. Gra polega na uczeniu postaci, jak postępować poprzez kombinację przykładów i informacji zwrotnych. Kiedy ludzie po raz pierwszy grają w tę grę, często nieumyślnie uczą stworzenia złych nawyków, co kończy się niemożnością wykonania nawet najbardziej podstawowych działań . Zwracając większą uwagę na to, jak działa stworzenie, gracze mogą lepiej nim manipulować, ale złudzenie uczenia prawdziwego stworzenia może zniknąć. Większość złożonych rzeczy, które widzieliśmy źle, nigdy nie dotarło do ostatecznej wersji gry. Odwieczna pokusa dla deweloperów do korzystania z najnowszych technik i najbardziej wysublimowanych algorytmów do implementacji ich postaci AI. W późnej fazie rozwoju,

kiedy ucząca się sztuczna inteligencja wciąż nie może nauczyć się kierować samochodem po torze bez odjeżdżania na każdym zakręcie, prostsze algorytmy niezmiennie przychodzą na ratunek i trafiają do gry. Wiedza, kiedy być złożonym, a kiedy pozostać prostym, jest najtrudniejszym elementem sztuki programisty AI. Najlepsi programiści AI to ci, którzy potrafią zastosować bardzo prostą technikę, aby dać złudzenie złożoności.

Okno postrzegania

O ile twoja sztuczna inteligencja nie kontroluje zawsze obecnego pomocnika lub wroga jeden na jednego, istnieje szansa, że twój gracz spotka postać tylko przez krótki czas. Może to być bardzo krótki czas dla strażników jednorazowych, których celem życiowym jest zastrzelenie. Trudniejsi wrogowie mogą znajdować się na ekranie przez kilka minut, gdy ich upadek jest planowany i wykonywany. Kiedy oceniamy kogoś w prawdziwym życiu, naturalnie stawiamy się w jego butach. Patrzymy na otoczenie, informacje, które zbierają z ich otoczenia i działania, które przeprowadzają. Strażnik stojący w ciemnym pokoju słyszy hałas: „Pstryknąłbym włącznikiem światła", myślimy. Jeśli strażnik tego nie zrobi, uważamy, że jest głupi. Jeśli tylko przez chwilę ujrzymy kogoś, nie będziemy mieli czasu, aby zrozumieć jego sytuację. Jeśli widzimy, że strażnik, który usłyszał hałas, nagle odwraca się i porusza powoli w przeciwnym kierunku, zakładamy, że AI jest wadliwa. Strażnik powinien był przejść przez pokój w kierunku hałasu. Jeśli zatrzymamy się trochę dłużej i zobaczymy, jak strażnik zbliża się do włącznika światła przy wyjściu, zrozumiemy jego działanie. Z drugiej strony strażnik może nie pstryknąć włącznikiem światła, a my traktujemy to jako oznakę złej realizacji. Ale strażnik może wiedzieć, że światło nie działa, albo czekał, aż kolega wsadzi papierosy pod drzwi i pomyślał, że hałas to z góry określony sygnał. Gdybyśmy to wszystko wiedzieli, wiedzielibyśmy, że akcja była inteligentna. Ta sytuacja, w której nie ma wygranej, jest oknem postrzegania. Musisz upewnić się, że AI postaci pasuje do jej celu w grze i uwagi, jaką zwróci od niej gracz. Dodanie większej liczby sztucznej inteligencji do przypadkowych postaci może cię zrazić do rzadkiego gracza, który gra na każdym poziomie przez kilka godzin, sprawdzając ciekawe zachowanie lub błędy, ale wszyscy inni (w tym wydawca i prasa) mogą myśleć, że twoje oprogramowanie było niechlujne.

Zmiany w zachowaniu

W oknie percepcji nie chodzi tylko o czas. Pomyśl jeszcze raz o duchach w Pac-Manu. Mogą nie sprawiać wrażenia wrażliwości, ale nie robią nic nie na miejscu. Dzieje się tak, ponieważ rzadko zmieniają zachowanie (jedyną okazją jest ich transformacja, gdy gracz spożywa dopalacz). Ilekroć postać w grze zmienia zachowanie, zmiana jest znacznie bardziej zauważalna niż samo zachowanie. W ten sam sposób, gdy zachowanie postaci powinno się oczywiście zmienić i nie zmienia się, rozlegają się dzwonki ostrzegawcze. Jeśli dwóch strażników stoi i rozmawia ze sobą, a ty zestrzelisz jednego, drugi strażnik nie powinien kontynuować rozmowy! Zmiana zachowania prawie zawsze występuje, gdy gracz jest w pobliżu lub został zauważony. To samo dotyczy gier platformowych, jak i strategii czasu rzeczywistego. Dobrym rozwiązaniem jest zachowanie tylko dwóch zachowań przypadkowych postaci - normalnej akcji i akcji wykrytej przez gracza.

Rodzaj AI w grach

Gry zawsze były krytykowane za słabo zaprogramowane (w sensie inżynierii oprogramowania): wykorzystują sztuczki, tajemne optymalizacje i niesprawdzone technologie, aby uzyskać dodatkową szybkość lub zgrabne efekty. Gra AI nie jest inna. Jedną z największych barier między ludźmi AI a naukowcami AI jest to, co kwalifikuje się jako AI. Z naszego doświadczenia wynika, że AI w grze to hakowanie równych części (rozwiązania ad hoc i fajne efekty), heurystyka (praktyczne zasady, które działają tylko w większości, ale nie we wszystkich przypadkach) i algorytmy („właściwe" rzeczy). Większość tego tekstu jest skierowana do ostatniej grupy, ponieważ to rzeczy, które możemy

analizować analitycznie, mogą być używane w wielu grach i mogą stanowić podstawę silnika AI. Ale dwie pierwsze kategorie są równie ważne i mogą tchnąć życie w postacie jak najbardziej skomplikowany algorytm.

Hacki

Jest takie powiedzenie: „Jeśli wygląda jak ryba i pachnie jak ryba, prawdopodobnie jest to ryba". Korelacją psychologiczną jest behawioryzm. Badamy zachowanie, a rozumiejąc, w jaki sposób jest ono budowane, rozumiemy wszystko, co możemy na temat tego, co się zachowuje. Jako podejście psychologiczne ma swoich zwolenników, ale zostało w dużej mierze zastąpione (szczególnie wraz z pojawieniem się neuropsychologii). Ten spadek mody wpłynął również na sztuczną inteligencję. W pewnym momencie akceptacja wiedzy o ludzkiej inteligencji przez zbudowanie maszyny do jej replikacji była akceptowalna, obecnie uważa się ją za słabą naukę. I nie bez powodu; w końcu zbudowanie maszyny do gry w szachy wymaga algorytmów, które wyglądają na dziesiątki ruchów do przodu. Ludzie po prostu nie są zdolni do tego. Z drugiej strony, w przypadku sztucznej inteligencji w grze często należy stosować behawioryzm. Nie interesuje nas natura rzeczywistości ani umysłu; chcemy postaci, które wyglądają dobrze. W większości przypadków oznacza to rozpoczęcie od ludzkich zachowań i próbując znaleźć najprostszy sposób na ich wdrożenie w oprogramowaniu. GoodAI w grach zwykle działa w tym kierunku. Programiści rzadko budują świetny nowy algorytm, a następnie zadają sobie pytanie: „Co więc mogę z tym zrobić?" Zamiast tego zacznij od projektu postaci i zastosuj najbardziej odpowiednie narzędzie, aby uzyskać wynik. Oznacza to, że to, co kwalifikuje się jako sztuczna inteligencja gry, może być nierozpoznawalne jako technika sztucznej inteligencji. W poprzedniej części przyjrzeliśmy się sztucznej inteligencji dla duchów Pac-Mana - prostego generatora liczb losowych jako stosowane rozsądnie. Generowanie liczby losowej nie jest techniką AI jako taką. W większości języków są wbudowane funkcje pozwalające uzyskać losową liczbę, więc na pewno nie ma sensu podawać algorytmu! Ale może działać w zaskakującej liczbie sytuacji. Innym dobrym przykładem kreatywnego rozwoju AI jest The Sims [Maxis Software, Inc., 2000]. Podczas gdy pod powierzchnią dzieją się dość skomplikowane rzeczy, wiele zachowań postaci komunikuje się z animacją. W grze Star Wars: Episode 1 Racer [LucasArts Entertainment Company LLC, 1999] postacie, które są zirytowane, będą nieco przesuwać się po bokach innych postaci. Quake II [id Software, Inc., 1997] ma polecenie „gest", w którym postacie (i gracze) mogą odwrócić wroga. Wszystko to nie wymaga znaczącej infrastruktury AI. Nie potrzebują skomplikowanych modeli poznawczych, uczenia się ani algorytmów genetycznych. Potrzebują tylko odrobiny kodu, który wykonuje animację we właściwym miejscu. Zawsze wypatruj prostych rzeczy, które mogą dać iluzję inteligencji. Jeśli chcesz angażować emocjonalne postacie, czy możesz dodać do animacji kilka animacji emocji (sfrustrowany pocieranie świątyni lub piętno stopy)? Wywołanie ich we właściwym miejscu jest znacznie łatwiejsze niż próba przedstawienia stanu emocjonalnego postaci poprzez jej działania. Czy masz wiele zachowań, z których postać wybierze? Czy wybór będzie obejmował złożone ważenie wielu czynników? Jeśli tak, warto wypróbować wersję AI, która wybiera zachowanie wyłącznie losowo (być może z innym prawdopodobieństwem dla każdego zachowania). Być może będziesz w stanie powiedzieć różnicę, ale Twoi klienci mogą nie; wypróbuj to na facecie z działu kontroli jakości.

Heurystyka

Heurystyka to ogólna zasada, przybliżone rozwiązanie, które może działać w wielu sytuacjach, ale jest mało prawdopodobne, aby działało we wszystkich. Ludzie cały czas używają heurystyki. Nie próbujemy obliczyć wszystkich konsekwencji naszych działań. Zamiast tego opieramy się na ogólnych zasadach, które sprawdziły się w przeszłości (lub które zostały poddane praniu mózgu, jednakowo). Może to być coś tak prostego, jak „jeśli coś zgubisz, a następnie cofnij swoje kroki" do heurystyki, która rządzi naszymi życiowymi wyborami, na przykład „nigdy nie ufaj sprzedawcy samochodów używanych".

Heurystyka została skodyfikowana i włączona do niektórych algorytmów zawartych u nas, a powiedzenie „heurystycznego" programistom AI często wywołuje obrazy zachowań polegających na szukaniu ścieżki lub zorientowaniu na cel. Mimo to wiele technik opisanych opiera się na heurystyce, które nie zawsze są jednoznaczne. Istnieje kompromis między szybkością i dokładnością w obszarach takich jak podejmowanie decyzji, ruch i myślenie taktyczne (w tym sztuczna inteligencja gry planszowej). Kiedy poświęca się dokładność, tak jest zwykle zastępując poszukiwanie poprawnej odpowiedzi heurystyką.

Szeroki zakres heurystyki można zastosować do ogólnych problemów AI, które nie wymagają określonego algorytmu. W naszym odwiecznym przykładzie Pac-Man, duchy wracają do gracza, wybierając trasę na skrzyżowaniu prowadzącym do jego aktualnej pozycji. Droga do gracza może być dość skomplikowana; może polegać na odwróceniu się od siebie i może być bezowocny, jeśli gracz będzie się nadal poruszał. Ale zasada kciuka (ruch w bieżącym kierunku gracza) działa i zapewnia graczowi wystarczające kompetencje, aby zrozumieć, że duchy nie są przypadkowe w ruchu. W InWarcraft [Blizzard Entertainment, 1994] (i wielu innych grach RTS, które nastąpiły później) istnieje heurystyka, która przesuwa postać do przodu w kierunku zasięgu broni dystansowej, jeśli wróg znajduje się ułamek poza zasięgiem postaci. Chociaż działało to w większości przypadków, nie zawsze była to najlepsza opcja. Wielu graczy było sfrustrowanych, gdy obszerne struktury obronne poszły na piechotę, gdy wrogowie się zbliżyli. Później gry RTS pozwoliły graczowi wybrać, czy to zachowanie zostanie włączone, czy nie. W wielu grach strategicznych, w tym w grach planszowych, różne jednostki lub elementy otrzymują jedną wartość liczbową, która reprezentuje ich „dobro". To jest heurystyka; zastępuje złożone obliczenia dotyczące możliwości jednostki jednym numerem. Liczba ta może zostać wcześniej zdefiniowana przez programistę. AI może ustalić, która strona jest do przodu, po prostu dodając liczby. W RTS może znaleźć najlepszą jednostkę ofensywną do zbudowania, porównując liczbę z jej kosztem. Wiele przydatnych efektów można osiągnąć, manipulując liczbą. Nie ma do tego algorytmu ani techniki. Nie znajdziesz go w opublikowanych badaniach AI. Ale to chleb powszedni pracy programisty AI.

Wspólna heurystyka

Garść heurystyki pojawia się w kółko w AI i oprogramowaniu w ogóle. Stanowią one dobry punkt wyjścia przy początkowym rozwiązywaniu problemu.

Najbardziej ograniczony

Biorąc pod uwagę obecny stan świata, należy wybrać jeden element w zestawie. Wybrany element powinien być tym, który byłby opcją dla najmniejszej liczby stanów. Na przykład grupa postaci napotyka zasadzkę. Jeden z zasadzek nosi fazowaną zbroję pola siłowego. Tylko nowy i rzadki karabin laserowy może go przebić. Jedna postać ma ten karabin. Kiedy wybierają, kogo zaatakować, w grę wchodzi najbardziej ograniczona heurystyka; rzadko zdarza się być w stanie zaatakować tego wroga, dlatego należy podjąć odpowiednie działania.

Najpierw wykonaj najtrudniejszą rzecz

Najtrudniejsza rzecz często ma wpływ na wiele innych działań. Lepiej to zrobić najpierw, niż odkryć, że łatwe rzeczy idą dobrze, ale ostatecznie się marnują. To jest przypadek najbardziej ograniczonej heurystyki, powyżej. Na przykład armia ma dwie drużyny z pustymi polami. Komputer planuje utworzenie pięciu wojowników orków i ogromnego Kamiennego Trolla. Chce skończyć z wyważonymi drużynami. Jak powinien przypisać jednostki do oddziałów? Kamienny Troll jest najtrudniejszy do przypisania, więc należy to zrobić najpierw. Gdyby Orkowie zostali przydzieleni jako pierwsi, byliby w

równowadze między dwiema drużynami, pozostawiając miejsce dla połowy Trolla w każdej drużynie, ale Troll nie miał dokąd pójść.

Najpierw wypróbuj najbardziej obiecującą rzecz

Jeśli istnieje wiele opcji dostępnych dla AI, często można dać każdej z nich naprawdę trudny i gotowy wynik. Nawet jeśli ten wynik jest dramatycznie niedokładny, wypróbowanie opcji w malejącej kolejności wyników zapewni lepszą wydajność niż wypróbowywanie rzeczy wyłącznie losowo.

Algorytmy

I tak dochodzimy do ostatniej trzeciej części pracy programisty AI: budowania algorytmów wspierających ciekawe zachowanie postaci. Haki i heurystyka zapewnią ci długą drogę, ale poleganie na nich oznacza tylko, że będziesz musiał ciągle wymyślać koło. Ogólne elementy sztucznej inteligencji, takie jak ruch, podejmowanie decyzji i myślenie taktyczne, korzystają ze sprawdzonych metod, które można bez końca wykorzystywać. Ta książka dotyczy tego rodzaju techniki, a następna część przedstawia dużą ich liczbę. Pamiętaj tylko, że w każdej sytuacji, w której najlepszym rozwiązaniem jest skomplikowany algorytm, prawdopodobnie będzie co najmniej pięć, gdzie prostszy hack lub heurystyka wykona zadanie.

Szybkość i pamięć

Największym ograniczeniem pracy programisty sztucznej inteligencji są fizyczne ograniczenia maszyny do gry. Sztuczna inteligencja w grze nie ma luksusu dni przetwarzania i gigabajtów pamięci. Deweloperzy często pracują nad szybkością i budżetem pamięci dla swojej sztucznej inteligencji. Jednym z głównych powodów, dla których nowe techniki sztucznej inteligencji nie są szeroko stosowane, jest ich czas przetwarzania lub wymagania dotyczące pamięci. To, co może wyglądać jak atrakcyjny algorytm w prostym demo, może spowolnić produkcję gry do zastoju. W tej sekcji omówiono problemy sprzętowe niskiego poziomu związane z projektowaniem i budową kodu AI. Większość tego, co jest tutaj zawarte, to ogólne porady dotyczące całego kodu gry. Jeśli jesteś na bieżąco z aktualnymi problemami z programowaniem gier i chcesz po prostu zapoznać się z AI, możesz

Problemy z procesorem

Najbardziej oczywistym ograniczeniem wydajności gry jest szybkość procesora, na którym jest uruchomiona. Wraz z rozwojem technologii graficznej rośnie tendencja do przenoszenia funkcji graficznych na sprzęt graficzny. Typowe działania związane z procesorem, takie jak animacja i wykrywanie kolizji, są współdzielone między GPU i CPU lub całkowicie przenoszone do układów graficznych. Uwalnia to znaczną ilość mocy obliczeniowej dla sztucznej inteligencji i innych nowych technologii (przede wszystkim fizyki, chociaż dźwięk środowiskowy jest teraz również bardziej widoczny). Udział czasu przetwarzania przeznaczonego na sztuczną inteligencję wzrósł i zaczął się w ciągu ostatnich pięciu lat w wielu przypadkach do około 20%, a w niektórych do ponad 50%. To oczywiście dobra wiadomość dla twórców sztucznej inteligencji, którzy chcą stosować bardziej skomplikowane algorytmy, szczególnie do podejmowania decyzji i strategicznych działań. Ale chociaż stopniowe wydłużanie czasu procesora pomaga odblokować nowe techniki, nie rozwiązuje podstawowego problemu. Uruchomienie wielu algorytmów sztucznej inteligencji zajmuje dużo czasu. Wszechstronny system odnajdywania ścieżek może zająć dziesiątki milisekund na znak. Oczywiście w RTS-ie z 1000 znaków nie ma szans na uruchomienie każdej klatki przez wiele lat. Złożona sztuczna inteligencja, która działa w grach, musi zostać podzielona na komponenty o niewielkich rozmiarach, które można rozprowadzać w wielu klatkach. Zastosowanie tych technik do dowolnego algorytmu sztucznej inteligencji może wprowadzić go w sferę użyteczności.

SIMD

Oprócz szybszego przetwarzania i rosnących budżetów sztucznej inteligencji, nowoczesne procesory do gier mają dodatkowe funkcje, które przyspieszają pracę. Większość z nich ma dedykowane przetwarzanie SIMD (pojedyncza instrukcja, wiele danych), technikę programowania równoległego, w której pojedynczy program jest stosowany do kilku elementów danych w tym samym czasie, tak jak się wydaje. Tak więc, jeśli każda postać musi obliczyć odległość euklidesową do swojego najbliższego wroga i kierunek ucieczki, AI można zapisać w taki sposób, że wiele znaków (zwykle cztery na obecnym sprzęcie) może wykonywać obliczenia w tym samym czasie. W tej książce jest kilka algorytmów, które odniosły ogromne korzyści z implementacji SIMDimplementation (algorytmy sterowania są najbardziej oczywiste). Ale ogólnie rzecz biorąc, możliwe jest przyspieszenie prawie wszystkich algorytmów dzięki rozsądnemu wykorzystaniu SIMD. Na konsolach SIMD można wykonać w koncepcyjnie oddzielnej jednostce przetwarzającej. W takim przypadku komunikacja między głównym CPU a jednostkami SIMD, a także dodatkowy kod synchronizujący ich działanie, często mogą wyeliminować

Zaletą szybkości z równoległości sekcji kodu. Użycie SIMD jest bardzo zależne od tego, czy kilka postaci robi to samo w tym samym czasie. Dane dla każdego zestawu znaków muszą być przechowywane razem (zamiast gromadzić wszystkie dane dla każdego znaku razem, co jest normalne), aby jednostki SIMD mogły je znaleźć jako całość. Prowadzi to do radykalnej restrukturyzacji kodu i znacznego spadku czytelności wielu algorytmów. Ponieważ ta książka dotyczy technik, a nie kodowania niskopoziomowego, pozostawimy równoległość jako ćwiczenie implementacyjne, jeśli Twoja gra tego potrzebuje.

Przetwarzanie wielordzeniowe i technologia Hyper-Threading

Nowoczesne procesory mają jednocześnie kilka aktywnych ścieżek wykonywania. Kod jest przekazywany do procesora, dzieląc go na kilka potoków, które są wykonywane równolegle. Wyniki z każdego potoku są następnie łączone w ostateczny wynik oryginalnego kodu. Kiedy wynik jednego potoku zależy od wyniku innego, może to obejmować cofanie i powtarzanie zestawu instrukcji. W procesorze znajduje się zestaw algorytmów, które sprawdzają, jak i gdzie podzielić kod i przewidują prawdopodobny wynik pewnych zależnych operacji; nazywa się to prognozowaniem gałęzi. Ten projekt procesora nazywa się superskalarnym. Zwykłe wątkowanie to proces pozwalający na jednoczesne przetwarzanie różnych bitów kodu.

Ponieważ w komputerze szeregowym nie jest to możliwe, jest to symulowane przez szybkie przełączanie do tyłu i do przodu między różnymi częściami kodu. Na każdym przełączniku (zarządzanym przez system operacyjny lub ręcznie wdrażanym na wielu konsolach) wszystkie istotne dane muszą również zostać przełączone. To przełączanie może być powolnym procesem i może spalić cenne cykle. Hyper-Threading to znak towarowy firmy Intel dotyczący wykorzystywania superskalarnego charakteru procesora do wysyłania różnych wątków w różnych potokach. Każdemu potokowi można przypisać inny wątek do przetworzenia, co pozwala na rzeczywiste równoległe przetwarzanie wątków. Procesory w konsolach obecnej generacji (PlayStation 3, Xbox 360 itd.) Są wszystkie wielordzeniowe. Nowsze procesory do komputerów PC wszystkich dostawców również mają tę samą strukturę. Procesor wielordzeniowy faktycznie ma wiele oddzielnych systemów przetwarzania (każdy może być dodatkowo superskalarny). Różne wątki mogą być przypisane do różnych rdzeni procesorów, dając ten sam rodzaj przyspieszenia w stylu hiperwątkowości (w rzeczywistości jest to większe, ponieważ istnieje jeszcze mniej zależności między potokami). W obu przypadkach kod AI może skorzystać z tej równoległości, uruchamiając sztuczną inteligencję dla różnych znaków w różnych wątkach, które mają być przypisane do różnych ścieżek przetwarzania. Na niektórych platformach (na przykład komputerach z procesorami Intel) wymaga to po prostu dodatkowego wywołania funkcji do skonfigurowania. Na innych (na

przykład PlayStation 3) należy o tym pomyśleć wcześnie i odpowiednio uporządkować cały kod AI. Wszystko wskazuje na to, że w przyszłych platformach sprzętowych będzie coraz większy stopień równoległości, szczególnie w obszarze konsoli, gdzie tańsze będzie wykorzystanie mocy obliczeniowej przy użyciu wielu prostszych procesorów, a nie jednego gigantycznego procesora. Nie będzie to nazywane hiperwątkowością (inaczej niż przez Intel), ale technika ta pozostanie tutaj i będzie kluczowym elementem tworzenia gier na wszystkich platformach przynajmniej do końca dekady.

Funkcje wirtualne / kierunek

Jest jeden szczególny kompromis, który jest mocno odczuwalny wśród programistów AI: kompromis między elastycznością a wykorzystaniem pośrednich wywołań funkcji. W konwencjonalnym wywołaniu funkcji kod maszynowy zawiera adres kodu, w którym funkcja jest implementowana. Procesor przeskakuje między lokalizacjami w pamięci i kontynuuje przetwarzanie w nowej lokalizacji (po wykonaniu różnych czynności, aby upewnić się, że funkcja może powrócić we właściwe miejsce). Superskalarna logika procesora jest do tego zoptymalizowana i może w pewnym stopniu przewidywać, w jaki sposób nastąpi skok. Pośrednie wywołanie funkcji jest trochę inne. Przechowuje lokalizację kodu funkcji w pamięci. Procesor pobiera zawartość lokalizacji pamięci, a następnie przeskakuje do określonej lokalizacji. W ten sposób realizowane są wywołania funkcji wirtualnych w C ++: lokalizacja funkcji jest sprawdzana w pamięci (w tabeli funkcji wirtualnych) przed wykonaniem. To dodatkowe obciążenie pamięci dodaje trywialną ilość czasu do przetwarzania, ale powoduje spustoszenie w predyktorze rozgałęzień na procesorze (i ma negatywny wpływ na pamięć podręczną, jak zobaczymy poniżej). Ponieważ procesor nie jest w stanie przewidzieć, dokąd będzie zmierzał, często zatrzymuje się, czeka, aż wszystkie jego potoki zakończą to, co robią, a następnie podejmuje pracę w miejscu, w którym zostało przerwane. Może to również wymagać uruchomienia dodatkowego kodu czyszczącego w procesorze. Taktowanie na niskim poziomie pokazuje, że pośrednie wywołania funkcji są zazwyczaj znacznie bardziej kosztowne niż bezpośrednie wywołania funkcji.

Tradycyjną mądrością w tworzeniu gier jest unikanie niepotrzebnych wywołań funkcji wszelkiego rodzaju, szczególnie wywołań funkcji pośrednich, ale wywołania funkcji wirtualnych sprawiają, że kod jest znacznie bardziej elastyczny. Pozwalają na opracowanie algorytmu, który działa w wielu różnych sytuacjach. Na przykład zachowanie pościgowe nie musi wiedzieć, do czego dąży, o ile może łatwo uzyskać lokalizację celu. W szczególności sztuczna inteligencja czerpie ogromne korzyści z możliwości dostosowywania się do różnych zachowań. Nazywa się to polimorfizmem w języku zorientowanym obiektowo: pisanie algorytmu używającego ogólnego obiektu i pozwalanie na wprowadzenie szeregu różnych implementacji. My używamy polimorfizmu i używalimy go w wielu grach. Uznaliśmy, że pokazanie algorytmów w całkowicie polimorficznym stylu jest jaśniejsze, mimo że część elastyczności może zostać zoptymalizowana w kodzie produkcyjnym. Kilka implementacji w kodzie źródłowym na stronie internetowej robi to: usuwa polimorfizm w celu uzyskania zoptymalizowanego rozwiązania dla podzbioru problemów. To kompromis, a jeśli wiesz, z jakimi rodzajami obiektów będziesz pracować w swojej grze, warto spróbować uwzględnić polimorfizm w niektórych algorytmach (szczególnie w wyszukiwaniu ścieżek, widzieliśmy przyspieszenie w ten sposób).Nasz punkt widzenia, który nie jest podzielany przez wszystkich (a może nawet większość) programistów, jest taki, że nieefektywność spowodowana pośrednimi wywołaniami funkcji nie jest warta utraty snu. Jeśli algorytm jest ładnie rozprowadzany za pomocą wielu klatek, to dodatkowe wywołanie funkcji będzie również rozproszone i ledwo zauważalne. Znamy przynajmniej jeden przypadek, w którym programista AI gry został skrytykowany za używanie funkcji wirtualnych, które „spowalniały grę” tylko po to, aby stwierdzili, że profilowanie wykazało, że nie spowodowały one żadnego wąskiego gardła.

Problemy związane z pamięcią

Większość algorytmów AI nie wymaga dużej ilości pamięci. Budżety pamięci dla sztucznej inteligencji wynoszą zwykle około 1 MB na konsolach 32 MB i 8 MB na maszynach 512 MB - wystarczająca ilość miejsca na nawet ciężkie algorytmy, takie jak analiza terenu i znajdowanie ścieżek. Gry online dla wielu graczy (MMOG) zazwyczaj wymagają znacznie więcej pamięci dla swoich większych światów, ale są uruchamiane na farmach serwerów o znacznie większej pojemności (mierzonej w gigabajtach pamięci RAM).

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 potrzebny procesorowi do wykonania swoich operacji. Gdyby procesory musiały polegać na głównej pamięci RAM, byłyby stale w martwym punkcie w oczekiwaniu na dane. Wszystkie nowoczesne procesory używają co najmniej jednego poziomu pamięci podręcznej: kopii 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. Następnie można nim dowolnie manipulować. Gdy procesor wykona swoją pracę, pamięć podręczna jest wysyłana z powrotem do pamięci głównej. Procesor zazwyczaj nie może pracować w pamięci głównej; cała potrzebna pamięć musi znajdować się w pamięci podręcznej. Systemy z systemem operacyjnym mogą dodatkowo komplikować ten problem, ponieważ żądanie pamięci może wymagać przejścia 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 odwzorowanym adresie mogą nie być dostępne w tym samym czasie (co jest określane jako błąd aliasingu). Wiele poziomów pamięci podręcznej działa tak samo, jak pojedyncza pamięć podręczna. Duża ilość pamięci jest pobierana do pamięci podręcznej najniższego poziomu, a jej podzbiór jest pobierany do każdej pamięci podręcznej wyższego poziomu, a procesor zawsze działa tylko na najwyższym poziomie. Jeśli algorytm wykorzystuje dane rozrzucone po pamięci, jest mało prawdopodobne, aby odpowiednia pamięć znajdowała się w pamięci podręcznej od chwili do chwili. Te chybienia w pamięci podręcznej są bardzo kosztowne w czasie. Procesor musi pobrać całą nową porcję pamięci do pamięci podręcznej dla jednej lub dwóch instrukcji, a następnie musi to wszystko przesłać strumieniowo i zażądać kolejnego bloku. Dobry system profilowania pokaże, kiedy występują błędy w pamięci podręcznej. Z naszego doświadczenia wynika, że dramatyczne przyspieszenie można osiągnąć, upewniając się, że wszystkie dane potrzebne do jednego algorytmu są przechowywane w tym samym miejscu. W tej książce, aby ułatwić zrozumienie, użyliśmy stylu obiektowego do rozmieszczenia danych. Wszystkie dane dotyczące konkretnego obiektu gry są przechowywane razem. To może nie być najbardziej wydajne rozwiązanie pamięci podręcznej. W grze zawierającej 1000 znaków może być lepiej trzymać wszystkie ich pozycje razem w tablicy, więc algorytmy wykonujące obliczenia na podstawie ich pozycji nie muszą ciągle przeskakiwać pamięci. Podobnie jak w przypadku wszystkich optymalizacji, profilowanie jest wszystkim, ale ogólny poziom wydajności można uzyskać programując z myślą o spójności danych.

Ograniczenia PC

Komputery PC to zarówno najpotężniejsze, jak i najsłabsze maszyny do gier. Mogą być frustrujące dla programistów ze względu na ich brak spójności. Tam, gdzie konsola ma naprawiony sprzęt, istnieje oszałamiająca liczba różnych konfiguracji dla komputerów PC. Sprawy są prostsze niż były: interfejsy programowania aplikacji (API), takie jak DirectX, uniemożliwiają programistom kierowanie sprzętem, ale gra nadal musi wykrywać obsługę funkcji i szybkość oraz odpowiednio dostosowywać. Praca z komputerami PC obejmuje tworzenie oprogramowania, które można skalować od ograniczonego systemu zwykłego gracza do najnowocześniejszego sprzętu z wbudowanym wentylatorem. W

przypadku grafiki to skalowanie może być dość proste; na przykład w przypadku maszyn o niskiej specyfikacji wyłączamy zaawansowane funkcje renderowania. Można zastosować prostszy algorytm cieni lub moduł cieniujący pikseli można zastąpić prostym mapowaniem tekstury. Zmiana w wyrafinowaniu grafiki zwykle nie zmienia rozgrywki. AI jest inna. Jeśli sztuczna inteligencja ma mniej czasu na pracę, jak powinna zareagować? Może próbować wykonać mniej pracy. W rzeczywistości jest to to samo, co posiadanie głupiej sztucznej inteligencji i może wpływać na poziom trudności gry. Zespół ds. Zapewnienia jakości (QA) lub wydawca prawdopodobnie nie może zaakceptować, aby gra była znacznie łatwiejsza na komputerach o niższych parametrach. Podobnie, jeśli spróbujemy wykonać taką samą ilość pracy, może to potrwać dłużej. Może to oznaczać niższą liczbę klatek na sekundę lub może oznaczać więcej klatek między postaciami podejmującymi decyzje. Postacie o wolnym reagowaniu są często łatwiejsze do gry i mogą powodować te same problemy przy QA. Rozwiązanie stosowane przez większość programistów polega na ukierunkowaniu sztucznej inteligencji na najniższy wspólny mianownik: maszynę o minimalnych specyfikacjach wymienioną w dokumencie projektu technicznego. Czas sztucznej inteligencji w ogóle nie skaluje się z możliwościami maszyny. Szybsze maszyny po prostu wykorzystują proporcjonalnie mniej swojego budżetu przetwarzania na sztuczną inteligencję. Jest jednak wiele gier, w których skalowalna sztuczna inteligencja jest możliwa. Wiele gier wykorzystuje sztuczną inteligencję do kontrolowania postaci z otoczenia: pieszych spacerujących po chodniku, członków tłumu wiwatujących podczas wyścigu lub rojących się na niebie stad ptaków. Ten rodzaj sztucznej inteligencji jest dowolnie skalowalny: można użyć większej liczby znaków, gdy dostępny jest czas procesora. Rozdział poświęcony zarządzaniu zasobami obejmuje kilka technik określających poziom szczegółowości AI, który może poradzić sobie z taką skalowalnością.

Ograniczenia konsoli

Konsole mogą być prostsze w obsłudze niż na komputerze. Znasz dokładnie maszynę, na którą celujesz, i zwykle widzisz kod działający na komputerze docelowym. Nie trzeba się martwić o przyszłe zabezpieczenia dla nowego sprzętu lub ciągle zmieniających się wersji interfejsów API. Programiści pracujący z technologią nowej generacji często nie mają dokładnych specyfikacji ostatecznej maszyny lub niezawodnej platformy sprzętowej (początkowe zestawy rozwojowe dla Xbox 360 były niewiele więcej niż dedykowanym emulatorem), ale większość rozwoju konsoli ma dość ustalony cel . Proces listy kontrolnej wymagań technicznych (TRC), za pomocą którego producent konsoli określa minimalne standardy dotyczące działania gry, służy do naprawiania takich rzeczy, jak liczba klatek na sekundę (chociaż różne terytoria mogą się różnić - na przykład PAL i NTSC). Oznacza to, że budżety sztucznej inteligencji można zablokować na określoną liczbę milisekund. To z kolei znacznie ułatwia ustalenie, jakich algorytmów można użyć i ustalenie celu optymalizacji (pod warunkiem, że budżet nie zostanie obcięty na ostatnim etapie, aby zrobić miejsce dla najnowszej techniki graficznej używanej w grze konkurencji). Z drugiej strony konsole generalnie cierpią z powodu długiego czasu realizacji. Jest możliwe i całkiem istotne, aby skonfigurować projekt rozwoju na PC, aby poprawki do sztucznej inteligencji mogły być kompilowane i testowane bez wykonywania pełnej wersji gry. Dodając nowy kod, można szybko ocenić obsługiwane przez niego zachowanie. Często ma to postać okrojonych miniaplikacji, chociaż wielu programistów korzysta z bibliotek współdzielonych podczas tworzenia, aby uniknąć ponownego łączenia całej gry. Oczywiście możesz zrobić to samo na konsoli, ale podróż w obie strony do konsoli zajmuje więcej czasu. Sztuczna inteligencja ze sparametryzowanymi wartościami, które wymagają wielu poprawek (na przykład algorytmy ruchu są z tego znane) prawie wymaga pewnego rodzaju systemu dostosowywania w grze dla konsoli. Niektórzy programiści idą dalej i pozwalają, aby ich narzędzie do projektowania poziomów lub tworzenia sztucznej inteligencji było bezpośrednio połączone przez sieć z komputera deweloperskiego do uruchomionej gry na konsoli tekstowej. Pozwala to na bezpośrednią manipulację zachowaniem postaci i natychmiastowe testowanie. Infrastruktura potrzebna do tego jest różna, a niektóre platformy (przychodzi na myśl

GameCube firmy Nintendo) znacznie utrudniają życie. We wszystkich przypadkach jest to jednak znaczna inwestycja i znacznie wykracza poza zakres tej książki (nie wspominając o naruszeniu kilku umów o zachowaniu poufności). Jest to obszar, w którym firmy zajmujące się oprogramowaniem pośredniczącym zaczęły osiągać sukcesy, zapewniając solidne narzędzia do debugowania na miejscu i przeglądania treści w ramach swoich pakietów technologicznych.

Praca ze sprzętem renderującym

Największym problemem w przypadku starszych (czyli poprzedniej generacji) konsol jest ich optymalizacja pod kątem grafiki. Grafika jest zwykle czynnikiem technologicznym stojącym za grami, a przy niewielkiej ilości soku do umieszczenia w maszynie, naturalne jest, że sprzedawca konsoli kładzie nacisk na grafikę możliwości. Oryginalna architektura Xboksa była pod tym względem powiewem świeżości, dostarczając pierwszą architekturę konsoli podobną do PC: główny procesor podobny do komputera PC, zrozumiałą (ale nie podobną do PC) magistralę graficzną i znajomy chipset graficzny. Na drugim końcu spektrum, dla tej samej generacji, PlayStation 2 (PS2) zostało bezwstydnie zoptymalizowane pod kątem renderowania grafiki. Aby jak najlepiej wykorzystać sprzęt, konieczne było zrównoleglenie jak największej części renderowania, przez co problemy z synchronizacją i komunikacją były bardzo trudne do rozwiązania. Kilku programistów po prostu zrezygnowało i użyło śmiesznie prostej sztucznej inteligencji w swoich pierwszych grach na PS2. Przez całą iterację konsoli nadal był solą w oku programisty AI pracującego nad tytułem wieloplatformowym. Na szczęście dzięki wielordzeniowemu procesorowi w PlayStation 3 szybkie przetwarzanie AI jest znacznie łatwiejsze do osiągnięcia.

Sprzęt do renderowania działa na modelu potoku. Dane trafiają z jednej strony i są przetwarzane za pomocą wielu różnych prostych programów. Pod koniec potoku dane są gotowe do renderowania na ekranie. Dane nie mogą łatwo przekazać kopii zapasowej potoku, a tam, gdzie jest wsparcie, ilość danych jest zwykle niewielka (na przykład kilkadziesiąt pozycji danych). Sprzęt można skonstruować tak, aby działał bardzo wydajnie; istnieje prosty i logiczny przepływ danych, a fazy przetwarzania nie mają interakcji poza przekształceniem danych wejściowych. AI nie pasuje do tego modelu; jest z natury rozgałęziony, ponieważ różne bity kodu są uruchamiane w różnym czasie. Jest również wysoce autoreferencyjny; wyniki jednej operacji wpływają na wiele innych, a ich wyniki wracają do pierwszego zestawu i tak dalej. Nawet proste zapytania AI, takie jak ustalenie, gdzie postacie będą się zderzać, jeśli będą się poruszać, są trudne do wdrożenia, jeśli cała geometria jest przetwarzana na dedykowanym sprzęcie. Starszy sprzęt graficzny może obsługiwać wykrywanie kolizji, ale przewidywanie kolizji wymagane przez kod sztucznej inteligencji wciąż jest trudne do wdrożenia. Bardziej złożona sztuczna inteligencja jest nieuchronnie uruchamiana na procesorze, ale ponieważ ten układ jest stosunkowo niedostateczny na konsolach ostatniej generacji, sztuczna inteligencja była ograniczona do rodzaju budżetów widocznych na 5- lub nawet 10-letnich komputerach. Historycznie rzecz biorąc, wszystko to miało tendencję do ograniczania ilości sztucznej inteligencji wykonywanej na konsolach w porównaniu z komputerami PC o takiej samej mocy obliczeniowej. Najbardziej ekscytującą częścią uprawiania sztucznej inteligencji w ciągu ostatnich 18 miesięcy była dostępność konsol obecnej generacji z możliwością uruchamiania sztucznej inteligencji bardziej podobnej do komputerów PC.

Konsole ręczne

Konsole przenośne zwykle pozostają w tyle za pełnowymiarowymi konsolami i komputerami o około 5 do 10 lat. Dotyczy to również typowych technologii używanych do tworzenia gier dla nich. Podobnie jak sztuczna inteligencja pojawiła się w połowie lat 90. XX wieku, tak i XXI w. Obserwuje się rozwój komputerów podręcznych zdolnych do zaawansowanej sztucznej inteligencji. Większość technik opisanych w tej książce nadaje się do użycia na urządzeniach przenośnych obecnej generacji

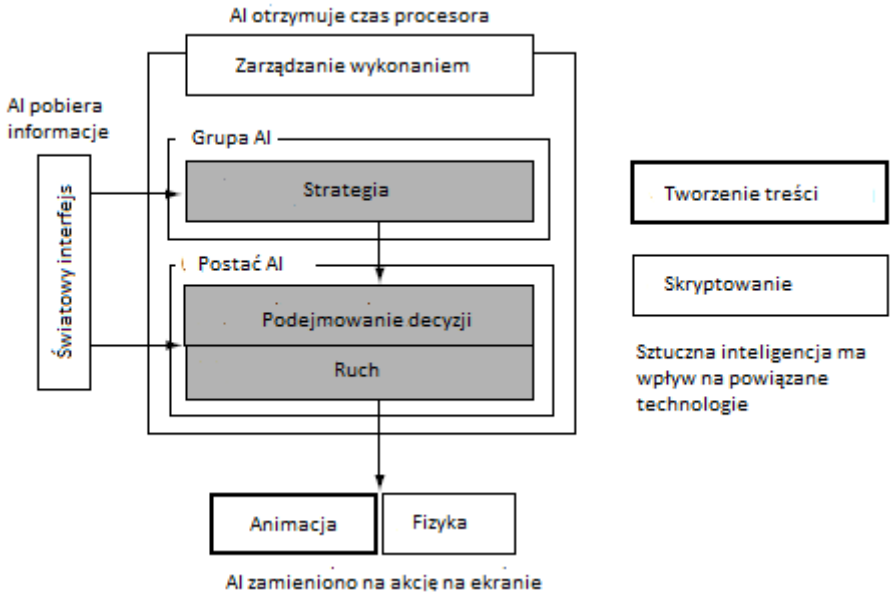
(PlayStation Portable i nie tylko), z takim samym zestawem ograniczeń, jak w przypadku każdej innej konsoli. Na prostszych urządzeniach (telefony komórkowe niezoptymalizowane pod kątem gier, dekodery telewizyjne lub PDA o niskiej specyfikacji) jesteś znacznie ograniczony pamięcią i mocą przetwarzania. W skrajnych przypadkach nie ma wystarczającej mocy w maszynie, aby zaimplementować odpowiednią warstwę zarządzania wykonywaniem, więc każdy używany algorytm sztucznej inteligencji musi być szybki.

Silnik AI

W ciągu ostatnich 20 lat nastąpiła wyraźna zmiana w sposobie tworzenia gier. Kiedyś gra była w większości tworzona od zera. Przeciągnięto niektóre fragmenty kodu z poprzednich projektów, a niektóre fragmenty zostały przerobione i ponownie wykorzystane, ale większość została napisana od podstaw. Kilka firm użyło tego samego podstawowego kodu do napisania wielu gier, o ile były one w podobnym stylu i gatunku. Na przykład silnik SCUMM firmy Lucasarts był stopniowo ewoluującym silnikiem używanym do zasilania wielu gier przygodowych typu „wskaż i kliknij". Od tego czasu wszechobecny stał się silnik gry, spójna platforma techniczna, na której firma tworzy większość swoich gier. Niektóre rzeczy niskiego poziomu (takie jak rozmowa z systemem operacyjnym, ładowanie tekstur, formaty plików modeli itp.) Są wspólne dla wszystkich gier, często z warstwą elementów specyficznych dla gatunku. Przygodowa gra akcji z perspektywy trzeciej osoby i kosmiczna strzelanka mogą nadal korzystać z tego samego podstawowego silnika w obu projektach. Zmienił się także sposób rozwoju sztucznej inteligencji. Początkowo AI została napisana dla każdej gry i dla każdej postaci. Dla każdej nowej postaci w grze byłby blok kodu do wykonania jego sztucznej inteligencji. Zachowanie postaci było kontrolowane przez mały program i nie było potrzeby stosowania algorytmów podejmowania decyzji w tej książce. Obecnie rośnie tendencja do posiadania ogólnych procedur AI w silniku gry i pozwalania na projektowanie postaci przez edytorów poziomów lub artystów technicznych. Struktura silnika jest stała, a SI dla każdej postaci łączy komponenty w odpowiedni sposób. Tak więc budowanie silnika gry wiąże się z budowaniem narzędzi AI, które można łatwo ponownie wykorzystać, łączyć i stosować w interesujący sposób. Aby to wspierać, potrzebujemy struktury sztucznej inteligencji, która ma sens w przypadku wielu gatunków.

Struktura silnika AI

Z naszego doświadczenia wynika, że istnieje kilka podstawowych struktur, które muszą istnieć dla ogólnego systemu SI. Są one zgodne z modelem AI podanym na rysunku



Po pierwsze, musimy mieć jakąś infrastrukturę w dwóch kategoriach: ogólny mechanizm zarządzania zachowaniami sztucznej inteligencji (decydowanie, które zachowanie zostanie uruchomione, kiedy itd.) Oraz system łączący się ze światem w celu pobierania informacji do sztucznej inteligencji. Każdy stworzony algorytm sztucznej inteligencji musi uwzględniać te mechanizmy. Po drugie, musimy mieć środki, aby zmienić wszystko, co chce sztuczna inteligencja, w działanie na ekranie. Składa się ze standardowych interfejsów do ruchu i kontrolera animacji, który może zamienić żądania, takie jak „pociągnij dźwignię 1" lub „idź ukradkiem do pozycji x, y" w działanie. Po trzecie, łącznikiem między nimi musi być standardowa struktura zachowania. Jest prawie pewne, że będziesz musiał napisać jeden lub dwa algorytmy sztucznej inteligencji dla każdej nowej gry. Ogromnie pomaga to, że cała sztuczna inteligencja jest zgodna z tą samą strukturą. Nowy kod może być opracowywany, gdy gra jest uruchomiona, a nowa sztuczna inteligencja może po prostu zastąpić zachowania zastępcze, gdy jest gotowa. Oczywiście wszystko to należy przemyśleć z wyprzedzeniem. Struktura musi być gotowa, zanim zaczniesz dobrze kodować AI. Część III tej książki omawia technologie wspomagające, które są pierwszą rzeczą do zaimplementowania w silniku AI. Następnie można zastosować poszczególne techniki.

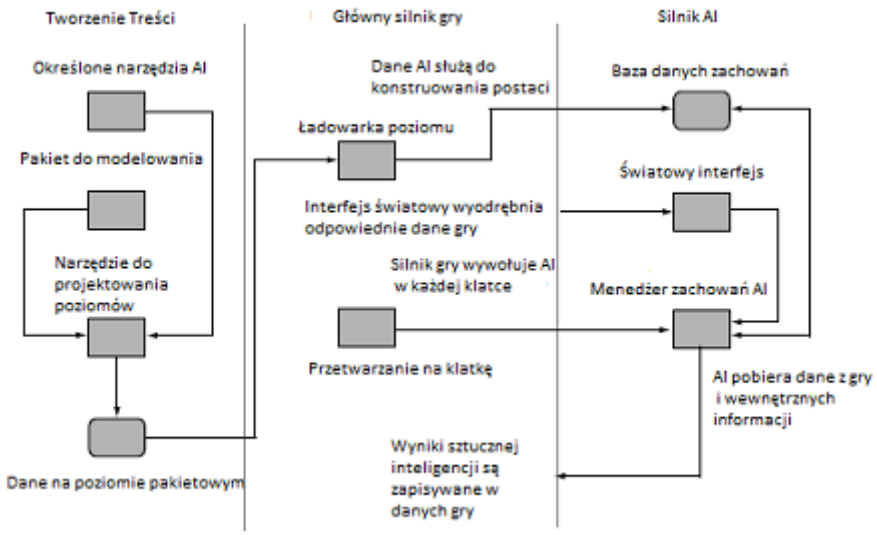
Nie będziemy rozpamiętywać tej struktury. Istnieją techniki, które omówimy, które mogą działać samodzielnie, a wszystkie algorytmy są dość niezależne. W przypadku wersji demonstracyjnej lub prostej gry wystarczy zastosować tę technikę. Kod na stronie internetowej jest zgodny ze standardową strukturą zachowań sztucznej inteligencji: każdemu można nadać czas wykonania, każdy otrzymuje informacje z centralnego systemu przesyłania wiadomości i każdy wyprowadza swoje działania w standardowym formacie. Konkretny zestaw interfejsów, których używaliśmy, pokazuje nasze własne nastawienie programistyczne. Zaprojektowano je jako dość proste, więc algorytmy nie są przeciążone przez kod infrastruktury. Z tego samego powodu istnieją łatwe optymalizacje, które zauważysz, których nie wdrożyliśmy, ponownie ze względu na przejrzystość. Pełnowymiarowy system AI może mieć podobny interfejs do kodu na stronie internetowej, ale z licznymi optymalizacjami szybkości i pamięci. Inne silniki AI na rynku mają inną strukturę, a używany silnik graficzny prawdopodobnie nałoży dodatkowe ograniczenia na twoją własną implementację. Jak zawsze, użyj kodu na stronie jako punktu wyjścia. Dobra struktura sztucznej inteligencji pomaga skrócić czas ponownego użycia, debugowania i programowania, ale tworzenie sztucznej inteligencji dla określonej postaci wymaga połączenia różnych technik we właściwy sposób. Konfigurację postaci można wykonać ręcznie, ale coraz częściej wymaga to jakiegoś narzędzia edycyjnego.

Obawy związane z łańcuchem narzędzi

Kompletny silnik AI będzie miał centralną pulę algorytmów AI, które można zastosować do wielu postaci. Definicja sztucznej inteligencji konkretnego znaku będzie zatem składać się z danych (które mogą obejmować skrypty w jakimś języku skryptowym), a nie skompilowanego kodu. Dane określają, w jaki sposób znak jest składany: jakie techniki będą używane oraz w jaki sposób te techniki są parametryzowane i łączone. Dane muszą pochodzić skądś. Dane można tworzyć ręcznie, ale nie jest to lepsze niż ręczne pisanie AI. Stabilne i niezawodne łańcuchy narzędzi to gorący temat w tworzeniu gier, ponieważ zapewniają artystom i projektantom możliwość łatwego tworzenia zawartości, jednocześnie umożliwiając wstawianie zawartości do gry bez pomocy ręcznej. Coraz więcej firm rozwija komponenty sztucznej inteligencji w swoim łańcuchu narzędzi: edytory do konfigurowania zachowań postaci i udogodnień w edytorze poziomów do oznaczania taktycznych lokalizacji lub miejsc, których należy unikać. Kierowanie się łańcuchem narzędzi ma swój wpływ na wybór technik sztucznej inteligencji. Łatwo jest ustawić zachowania, które zawsze działają w ten sam sposób. Dobrym przykładem są zachowania sterujące: zwykle są bardzo proste, łatwo je sparametryzować (z fizycznymi możliwościami postaci) i nie zmieniają się z postaci na postać. Trudniej jest zastosować zachowania, które mają wiele warunków, w których postać musi ocenić szczególne przypadki. System oparty na regułach musi mieć zdefiniowane skomplikowane reguły dopasowywania, które obsługiwane przez narzędzie zwykle wyglądają jak kod programu, ponieważ język programowania jest najbardziej naturalnym sposobem ich wyrażenia. Wielu programistów udostępnia tego rodzaju konstrukcje programistyczne w swoich narzędziach do edycji poziomów. Projektanci poziomów z pewnymi umiejętnościami programistycznymi mogą pisać proste reguły, wyzwalacze lub skrypty w języku, a edytor poziomów obsługuje przekształcanie ich w dane dla sztucznej inteligencji. Innym podejściem, używanym przez kilka pakietów oprogramowania pośredniego, jest wizualne przedstawienie warunków i decyzji. Na przykład moduł Maya AI-Implant eksponuje złożone warunki boolowskie i maszyny stanu za pomocą graficznych elementów sterujących.

Składanie wszystkiego razem

Ostateczna struktura silnika AI może wyglądać podobnie do tej.



Dane są tworzone w narzędziu (pakiet do modelowania lub projektowania poziomów lub dedykowane narzędzie AI), które jest następnie pakowane do wykorzystania w grze. Po załadowaniu poziomu zachowania AI w grze są tworzone z danych poziomu i rejestrowane w silniku AI. Podczas rozgrywki

główny kod gry wywołuje silnik AI, który aktualizuje zachowania, uzyskuje informacje z interfejsu świata i ostatecznie stosuje ich wyjście do danych gry. Zastosowane techniki w dużej mierze zależą od gatunku tworzonej gry. Omówimy szeroki zakres technik dla wielu różnych gatunków. Rozwijając sztuczną inteligencję w grze, musisz zastosować podejście mieszane i dopasowujące, aby uzyskać zachowania, których szukasz. Ostatni rozdział książki zawiera kilka wskazówek na ten temat; przygląda się, jak sztuczna inteligencja dla gier z głównych gatunków jest składana kawałek po kawałku.