NARZĘDZIA I TWORZENIE TREŚCI

Programowanie stanowi stosunkowo niewielki wysiłek w grze na rynek masowy. Większość czasu rozwoju poświęca się na tworzenie treści, tworzenie modeli, tekstur, środowisk, dźwięków, muzyki i animacji — od grafiki koncepcyjnej po dopracowanie poziomu. W ciągu ostatnich 15 lat programiści jeszcze bardziej ograniczyli wysiłek programistyczny, ponownie wykorzystując technologię w wielu tytułach, tworząc lub licencjonując silnik gry, na którym można uruchomić kilka gier. Dodanie do silnika wszechstronnego zestawu sztucznej inteligencji jest naturalnym rozszerzeniem. Jednak większość programistów nie jest zadowolona, aby na tym poprzestać. Ponieważ wysiłek związany z tworzeniem treści jest tak duży, proces tworzenia treści również musi być ustandaryzowany i narzędzia wykonawcze muszą być płynnie zintegrowane z narzędziami programistycznymi. Od ponad dekady te kompletne łańcuchy narzędzi są niezbędne do tworzenia dużych gier. Wraz z eksplozją popularności silnika Unity, stały się kluczowe dla istnienia małych studiów, niezależnych deweloperów i hobbystów. W rzeczywistości trudno przecenić znaczenie łańcucha narzędzi we współczesnym tworzeniu gier. W czasie pierwszego wydania tej książki łańcuch narzędzi był postrzegany jako główny czynnik decydujący o decyzjach wydawcy o wsparciu projektu. Teraz jest prawie wszechobecny. To naprawdę rzadki i odważny programista, który tworzy nową grę od podstaw bez sprawdzonego zestawu narzędzi. Częściowo wynika to z ich dostępności. Renderware Studio było głównym punktem sprzedaży graficznego oprogramowania pośredniczącego Criterion na początku 2000 roku. Ale licencjonowanie było kosztowne i wymagało szczegółowej umowy z dostawcą. Teraz pobranie Unity lub Unreal Engine z sieci, eksperymentowanie na prostej licencji użytkownika końcowego i dostęp do szeroko obsługiwanego i wydajnego zestawu narzędzi jest teraz banalnie proste. Narzędzia te nie są darmowe, ale łatwość ich użycia zmieniła branżę. Oprócz tych dwóch liderów rynku, inne systemy, takie jak open source Godot i Amazon's Lumberyard (widelec Crytek's Cryengine), również podkreślają ten sam styl tworzenia gier, z łańcuchem narzędzi i niestandardowym edytorem w centrum.

LIMIT ŁAŃCUCHÓW NARZĘDZI AI

Znaczenie łańcuchów narzędzi nakłada ograniczenia na sztuczną inteligencję. Zaawansowane techniki, takie jak sieci neuronowe, algorytmy genetyczne i planowanie działań zorientowane na cel (GOAP) nie były szeroko stosowane w komercyjnych tytułach. W pewnym stopniu dzieje się tak, ponieważ są one naturalnie trudne do odwzorowania w narzędziu do edycji poziomów. Wymagają specjalnego programowania dla postaci, co ogranicza szybkość tworzenia nowych poziomów i ponownego wykorzystania kodu między projektami. Większość narzędzi projektowych przeznaczonych dla sztucznej inteligencji dotyczy technik „chleba i masła": maszyn skończonych lub drzew zachowań, ruchu i odnajdywania ścieżek. Podejścia te opierają się na prostych procesach i znaczącej wiedzy. Łańcuchy narzędzi są naturalnie lepsze w umożliwianiu projektantom modyfikowania danych, a nie kodu, dlatego wzmacniane jest stosowanie tych klasycznych technik.

SKĄD POCHODZI WIEDZA AI

Dobra sztuczna inteligencja wymaga dużej wiedzy. Jak wielokrotnie widzieliśmy, posiadanie dobrej i odpowiedniej wiedzy na temat środowiska gry pozwala zaoszczędzić ogromną ilość czasu przetwarzania. A w czasie wykonywania, gdy gra ma wiele rzeczy do śledzenia, czas przetwarzania jest kluczowym zasobem. Wiedza wymagana przez algorytmy AI zależy od środowiska gry. Na przykład poruszająca się postać potrzebuje pewnej wiedzy o tym, gdzie i jak można się poruszać. Mogą to zapewnić programiści, przekazując AI bezpośrednio potrzebne dane. Jednak gdy poziom gry się zmieni, programista musi dostarczyć nowe zestawy danych. Nie promuje to ponownego wykorzystania między wieloma grami i utrudnia wprowadzanie prostych zmian na poziomach. Podejście typu toolchain do tworzenia gry nakłada na zespół tworzący treść obowiązek zapewnienia niezbędnej wiedzy o sztucznej

inteligencji. Proces ten może być wspomagany przez przetwarzanie offline, które automatycznie tworzy bazę wiedzy na podstawie informacji na poziomie surowym. Od lat zespół zajmujący się tworzeniem treści dostarcza wiedzę AI do poruszania się i odnajdywania ścieżek. Albo wprost, poprzez oznaczenie samego poziomu, albo przez automatyczne wygenerowanie takich danych z treści, które tworzą. Niedawno do łańcucha narzędzi włączono również podejmowanie decyzji i funkcje sztucznej inteligencji wyższego poziomu, zwykle za pomocą niestandardowych narzędzi zintegrowanych z aplikacją edytora.

WIEDZA DOTYCZĄCA PATHFINDINGU I PUNKTÓW TRASY

Algorytmy odnajdywania ścieżek działają na grafie ukierunkowanym: podsumowaniu poziomu gry w formie optymalnej dla algorytmu odnajdywania ścieżek. W rozdziale 4 omówiono kilka sposobów, w jakie geometrię środowiska wewnętrznego lub zewnętrznego można podzielić na regiony do wykorzystania w odnajdywaniu ścieżek. Ten sam rodzaj struktury danych jest używany w niektórych taktycznych AI. Na szczęście te same wymagania dotyczące narzędzi do odnajdywania ścieżki dotyczą taktyk z punktami orientacyjnymi. Podział geometrii poziomu na węzły i połączenia może być wykonany ręcznie przez projektanta poziomu lub może być wykonany automatycznie w procesie offline. Ponieważ ręczne tworzenie wykresu odnajdywania ścieżki może być czasochłonnym procesem (i trzeba go powtarzać za każdym razem, gdy zmienia się geometria poziomu), wielu programistów używa procesów automatycznych, przynajmniej do wstępnego szkicu. Wyniki czysto automatycznych procesów są zazwyczaj mieszane, a wraz ze wzrostem stopnia złożoności pojawia się więcej problemów. Aby uzyskać optymalne wyniki, zwykle wymagany jest pewien nadzór człowieka.

RĘCZNE TWORZENIE DANYCH REGIONU

Istnieją trzy elementy grafu odnajdywania ścieżek, które należy utworzyć: rozmieszczenie węzłów grafu (i wszelkie powiązane informacje o lokalizacji), połączenia między tymi węzłami oraz koszty związane z połączeniami. Cały wykres można utworzyć za jednym razem, ale często każdy element jest tworzony oddzielnie przy użyciu różnych technik. Projektant poziomów może ręcznie umieszczać węzły na poziomie gry. Połączenia można następnie obliczyć na podstawie informacji o zasięgu wzroku, a koszty można obliczyć podobnie algorytmicznie. Do pewnego stopnia koszt i połączenia między węzłami można łatwo obliczyć algorytmicznie. Prawidłowe umieszczanie węzłów obejmuje zrozumienie struktury poziomu i docenienie wzorców ruchu, które mogą wystąpić. Ta ocena jest znacznie łatwiejsza dla operatora niż algorytmu. W tej sekcji omówiono problemy związane z ręcznym określaniem wykresów (głównie węzłów wykresu). W poniższej sekcji omówiono automatyczne obliczanie wykresów, w tym połączeń i kosztów. Aby wesprzeć ręczne tworzenie węzłów wykresu, możliwości narzędzia do edycji poziomów zależą od używanej reprezentacji świata.

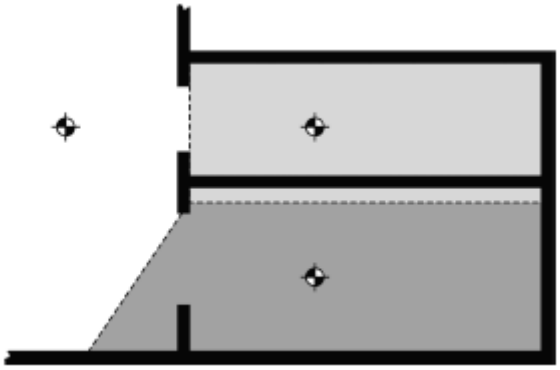
Wykresy kafelkowe

Wykresy kafelkowe zwykle nie wymagają od projektantów ręcznego określania jakichkolwiek danych w narzędziu do modelowania. Układ poziomu jest zwykle ustalony (na przykład gra RTS zwykle opiera się na stałej siatce, często o ograniczonej liczbie różnych rozmiarów). Należy również określić funkcje kosztów związane z odnajdywaniem ścieżki. Większość funkcji kosztowych bazuje na odległości i nachyleniu, modyfikowanych parametrami właściwymi dla danego typu znaku. Wartości te mogą być zazwyczaj generowane automatycznie (gradienty mogą być obliczane bezpośrednio z wartości np. wysokości). Modyfikatory specyficzne dla znaków są zwykle dostarczane w danych znakowych. Na przykład jednostka artylerii może ponieść dziesięciokrotnie większy koszt gradientu niż lekka jednostka rozpoznania. Często narzędzie do projektowania poziomów w grze opartej na kafelkach może zawierać dane AI za kulisami. Na przykład umieszczenie skrawka lasu może automatycznie zwiększyć koszt ruchu przez ten kafelek. Projektant poziomów nie musi wyraźnie określać zmiany kosztów ani nawet

wiedzieć, że dane AI są obliczane. W rezultacie do obsługi odnajdywania ścieżek na wykresach opartych na kafelkach nie jest wymagana dodatkowa infrastruktura. Jest to jeden z powodów, dla których nadal są one tak intensywnie wykorzystywane w sztucznej inteligencji w grach, które wymagają dużo odnajdywania ścieżki (takich jak RTS), nawet gdy grafika odsunęła się od kafelków sprite'ów.

Domeny Dirichleta.

Domeny Dirichleta są użyteczną reprezentacją świata w wielu gatunkach. Mają one zastosowanie (w formie punktów orientacyjnych) do wszystkiego, od gier samochodowych, przez strzelanki, po gry strategiczne. Edytor poziomów musi tylko umieścić zestaw punktów na poziomie gry, aby określić węzły wykresu. Region powiązany z każdym punktem to objętość, która jest najbliżej tego punktu niż do jakiegokolwiek innego. Większość narzędzi do edycji poziomów i wszystkie narzędzia do modelowania trójwymiarowego (3D) pozwalają użytkownikowi na dodanie niewidocznego obiektu pomocniczego w punkcie. Można to odpowiednio oznaczyć i wykorzystać jako węzeł na wykresie. Jak omówiono w rozdziale 4, z domenami Dirichleta wiążą się pewne problemy. Rysunek przedstawia dwie domeny Dirichleta w dwóch sąsiednich korytarzach.



Pokazane są regiony powiązane z każdym węzłem. Zauważ, że krawędź jednego korytarza jest niepoprawnie zgrupowana z następnym korytarzem. Postać, która zabłąka się w ten obszar, pomyśli, że znajduje się w zupełnie innym obszarze poziomu. Dlatego jego zaplanowana ścieżka będzie błędna. Podobne problemy z grupowaniem regionów występują w pionie, gdzie jedna trasa przechodzi przez drugą. Problemy komplikują się, gdy z każdym z nich można powiązać różne „wagi" węzła (więc większa objętość jest przyciągana do jednego węzła niż do drugiego). Rozwiązanie tego rodzaju błędnej klasyfikacji może wiązać się z wieloma testami zabawy i frustracją ze strony projektanta poziomów. Dlatego ważne jest, aby narzędzia wspierały wizualizację regionów powiązanych z każdą domeną. Jeśli projektanci poziomów są w stanie zobaczyć zbiór lokalizacji skojarzonych z każdym węzłem, mogą szybciej przewidywać i diagnozować problemy. Wielu problemów można całkowicie uniknąć, projektując poziomy, na których obszary żeglowne nie sąsiadują ze sobą. Poziomy z cienkimi ścianami, przejściami przez pomieszczenia i dużą ilością ruchu w pionie trudno właściwie podzielić na domeny Dirichleta. Oczywiście zmiana stylu gry nie jest możliwa tylko ze względu na narzędzie do oznaczania AI.

Siatki nawigacyjne

Ta sama siatka wielokątów używana do renderowania może być używana jako siatka nawigacyjna do wyszukiwania ścieżek. Każdy wielokąt piętra jest węzłem na wykresie, a łączność między węzłami jest określona przez łączność między wielokątami. To podejście wymaga od edytora poziomów określenia wielokątów jako części „podłogi". Najczęściej osiąga się to za pomocą materiałów: pewien zestaw materiałów jest uważany za podłogi. Każdy wielokąt, do którego nałożony jest jeden z tych materiałów, jest częścią podłogi. Niektóre narzędzia 3D i edytory poziomów pozwalają użytkownikowi powiązać

dodatkowe dane z wielokątem. Można to również wykorzystać do ręcznego oflagowania każdego wielokąta kondygnacji. W obu przypadkach przydatne może być zaimplementowanie narzędzia, dzięki któremu edytor poziomów może szybko zobaczyć, które wielokąty są częścią podłogi. Częstym problemem jest posiadanie zestawu tekstur dekoracyjnych na środku pokoju, który jest błędnie oznaczony jako „niepodłogowy" i uniemożliwia nawigację po pomieszczeniu. Można to łatwo zobaczyć, jeśli można łatwo zwizualizować wielokąty podłogi. Siatki nawigacyjne mają reputację niezawodnego sposobu reprezentowania świata podczas wyszukiwania ścieżek. Są one obsługiwane po wyjęciu z pudełka zarówno przez Unity, jak i Unreal Engine, dzięki czemu stały się najczęściej stosowaną techniką. Nie są jednak bez problemu. Jak omówiono w rozdziale dotyczącym odnajdywania ścieżek, niektóre geometrie poziomów mogą prowadzić do nieoptymalnych wykresów. Po raz kolejny bardzo przydatna jest możliwość nadpisania przez człowieka automatycznie generowanego wykresu.

Regiony ograniczone

Najbardziej ogólną formą grafu odnajdywania ścieżek jest taka, w której projektant poziomów może umieścić dowolne struktury ograniczające, aby utworzyć węzły grafu. Wykres można następnie zbudować bez ograniczania się do problemów domen Dirichleta lub ograniczeń wielokątów podłogi. Arbitralne regiony ograniczające są skomplikowane w obsłudze w narzędziu do projektowania poziomów lub modelowania. To podejście jest zatem zwykle uproszczone do umieszczania arbitralnie wyrównanych ramek ograniczających. Projektant poziomów może przeciągnąć ramkę ograniczającą nad regionami poziomu gry, aby określić, że zawartość tego pola powinna być liczona jako jeden węzeł na wykresie planowania. Węzły można następnie łączyć ze sobą, a ich koszty ustawiać ręcznie lub generować z właściwości geometrycznych skrzynek węzłowych.

AUTOMATYCZNE TWORZENIE WYKRESÓW

W przypadku wielu poprzednich podejść do obliczenia kosztów związanych z połączeniami w grafie można użyć algorytmu. Podejścia oparte na ręcznie określonych punktach widoczności lub domenach Dirichleta również wykorzystują algorytmy do określania łączności między węzłami. Automatyczne umieszczanie węzłów na pierwszym miejscu jest znacznie trudniejsze. Dla ogólnych poziomów w pomieszczeniach nie ma jednej optymalnej techniki. Z mojego doświadczenia wynika, że programiści, którzy polegają na automatycznym rozmieszczaniu węzłów, będą musieli ostatecznie dysponować mechanizmem pozwalającym projektantowi poziomów wywierać pewien wpływ i ręcznie poprawiać wynikowy wykres. Techniki automatycznego umieszczania węzłów można podzielić na dwa podejścia: analizę geometryczną i eksplorację danych.

ANALIZA GEOMETRYCZNA

Techniki analizy geometrycznej działają bezpośrednio na geometrii poziomu gry. Analizują strukturę poziomu gry i obliczają odpowiednie elementy grafu odnajdywania ścieżki. Analiza geometryczna jest również wykorzystywana w innych obszarach tworzenia gier, takich jak obliczanie potencjalnie widocznej geometrii, wykonywanie obliczeń globalnego radiosity i zapewnianie, że budżety renderowania są spełnione.

Obliczanie kosztów

W przypadku danych wyszukiwania ścieżek większość analiz geometrycznych oblicza koszt połączeń między węzłami. Jest to stosunkowo prosty proces, do tego stopnia, że rzadko można znaleźć grę, której koszt wykresu został ustalony ręcznie. Większość kosztów połączeń jest obliczana na podstawie odległości. Odnajdywanie ścieżek zwykle wiąże się ze znalezieniem krótkiej ścieżki, dlatego naturalną miarą jest odległość. Odległość między dwoma punktami można obliczyć trywialnie. W przypadku

reprezentacji, w których węzły są traktowane jako punkty, odległość połączenia może być traktowana jako odległość między dwoma punktami. Reprezentacja siatki nawigacyjnej zwykle ma koszty połączenia oparte na odległości między środkami sąsiednich trójkątów. Reprezentacje obszarów granicznych mogą podobnie wykorzystywać punkty środkowe regionów do obliczania odległości.

Obliczanie połączeń

Obliczanie, które węzły są połączone, jest również powszechną aplikacją. Najczęściej jest to wykonywane przez sprawdzanie linii wzroku między punktami.

Reprezentacje punktowe

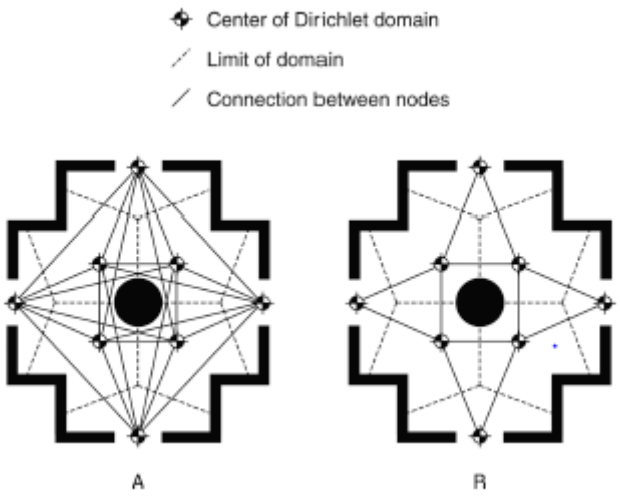
Reprezentacje węzłów oparte na punktach (takie jak domeny Dirichleta i reprezentacje punktów widoczności) wiążą każdy węzeł z jednym reprezentatywnym punktem. Pomiędzy każdą parą takich punktów można sprawdzić linię wzroku. Jeśli między punktami znajduje się linia wzroku, następuje połączenie między węzłami. Takie podejście może prowadzić do ogromnej liczby połączeń na wykresie. Rysunek 12.2 pokazuje dramatyczną złożoność wykresu opartego na widoczności dla stosunkowo prostego pomieszczenia. Z tego powodu często pojawiają się obawy dotyczące wydajności wykresów opartych na widoczności. Ale takie obawy są ciekawe, ponieważ prosty etap przetwarzania końcowego może łatwo naprawić sytuację i wygenerować użyteczne wykresy:

1. Każde połączenie jest rozpatrywane po kolei.

2. Połączenie zaczyna się w jednym węźle i kończy w innym. Jeśli połączenie przechodzi po drodze przez węzły pośrednie, to połączenie jest usuwane.

3. Tylko pozostałe połączenia stanowią część grafu odnajdywania ścieżki.

Ten algorytm szuka par węzłów, które są w zasięgu wzroku, ale nie ma między nimi bezpośredniej trasy. Ponieważ postać będzie musiała przejść po drodze przez inne węzły, nie ma sensu utrzymywać połączenia. Druga część rysunku pokazuje efekt zastosowania algorytmu do oryginalnego wykresu.



Arbitralne regiony graniczne

Dowolne obszary ograniczające są zwykle połączone w podobny sposób do punktów. W każdej parze regionów wybierany jest wybór punktów próbkowania i przeprowadzane są kontrole linii wzroku. Połączenie jest dodawane, gdy przechodzi pewna część kontroli linii wzroku. Poza używaniem wielu

sprawdzeń dla każdej pary regionów, proces jest taki sam, jak w przypadku reprezentacji punktowej. Często proporcja wymaganych przejść jest ustawiona na zero; połączenie jest dodawane, jeśli którakolwiek z kontroli pola widzenia zakończy się pomyślnie. W większości przypadków, jeśli jakakolwiek kontrola pola widzenia zakończy się pomyślnie, większość z nich to zrobi. Gdy tylko jedna kontrola przejdzie, możesz przestać sprawdzać i po prostu dodać połączenie. W przypadku regionów, które są od siebie bardzo oddalone, kilka sprawdzeń linii wzroku może przejść przez przeciskanie się przez drzwi, rozwarte narożniki pod kątem, wzniesienia i tak dalej. Te pary regionów nie powinny być połączone. Zwiększenie proporcji wymaganych przejść może rozwiązać problem, ale może znacznie wydłużyć czas potrzebny na analizę połączenia. Dodanie powyższego algorytmu przetwarzania końcowego wyeliminuje prawie wszystkie błędne połączenia, ale nie wyeliminuje fałszywych połączeń, które nie mają pośredniego zestawu regionów nawigowanych (takich jak w przypadku dużej pionowej przerwy między regionami). Połączenie obu rozwiązań poprawi sytuację, ale moje doświadczenie pokazało, że nadal będą problemy, które trzeba będzie rozwiązać ręcznie.

Ograniczenia podejść do widoczności

Podstawowym problemem związanym z podejściami na linii wzroku jest żeglowność. Tylko dlatego, że dwa regiony na poziomie są widoczne od siebie, nie oznacza to, że możesz się między nimi poruszać. Ogólnie rzecz biorąc, nie ma prostego testu sprawdzającego, czy możesz poruszać się między dwoma lokalizacjami w grze. W przypadku przygodowych gier akcji z perspektywy trzeciej osoby dotarcie do określonej lokalizacji może wymagać złożonej kombinacji dokładnych ruchów. Przewidywanie takich sekwencji ruchów jest trudne geometrycznie. Na szczęście postacie AI w takich grach rzadko muszą wykonywać takie sekwencje akcji. Zwykle ograniczają się one do poruszania się po łatwo dostępnych obszarach. To otwarte pytanie badawcze, czy analiza geometryczna może generować dokładne wykresy w złożonych środowiskach. Zespoły, którym się to udało, osiągnęły to poprzez ograniczenie nawigowalności poziomów, a nie przez poprawę złożoności algorytmów analizy. Reprezentacje siatkowe pozwalają uniknąć niektórych problemów, ale wprowadzają własne (w szczególności przeskakiwanie jest trudne do włączenia). Jak dotąd eksploracja danych jest najbardziej obiecującym podejściem do tworzenia wykresów odnajdywania ścieżek na poziomach o złożonej nawigacji.

Reprezentacje siatki

Reprezentacje siatki jawnie dostarczają informacji o połączeniu wymaganych do znajdowania ścieżek. Reprezentacja siatki oparta na trójkątach ma każdy trójkąt podłogi powiązany z węzłem wykresu. Trójkąt można opcjonalnie połączyć wzdłuż każdego z jego trzech boków z sąsiednim trójkątem podłogowym. Dlatego dla każdego węzła są do trzech połączeń. Połączenia można łatwo wyliczyć na podstawie danych geometrycznych: dwa trójkąty są połączone, jeśli mają wspólne dwa wierzchołki i oba są oznaczone jako trójkąty podłogi. Możliwe jest również łączenie trójkątów, które spotykają się w punkcie (tj. dzielą tylko jeden wierzchołek). Zmniejsza to ilość ruchów, które postać odnajdująca ścieżki będzie wyświetlać podczas poruszania się po gęstej siatce, ale może również powodować problemy z postaciami próbującymi skrócić rogi.

Obliczanie węzłów

Obliczenie położenia i geometrii węzłów za pomocą analizy geometrycznej jest bardzo trudne. Większość programistów unika tego wszystkiego razem. Jak dotąd jedynym (pół-)praktycznym rozwiązaniem było zastosowanie redukcji wykresów.

Redukcja grafów to szeroko badany temat w matematycznej teorii grafów. Zaczynając od bardzo złożonego wykresu z tysiącami lub milionami węzłów, tworzony jest nowy wykres, który oddaje „istotę” większego wykresu. W Części 4 przyjrzeliśmy się procesowi tworzenia grafu hierarchicznego.

Aby zastosować to podejście, geometria poziomu jest zalana milionami węzłów wykresów. Często można to zrobić po prostu za pomocą siatki: na przykład węzły wykresu są umieszczane co pół metra na całym poziomie. Węzły siatki znajdujące się poza obszarem gry (w ścianie lub nieosiągalne z ziemi) są usuwane. Jeśli poziom jest podzielony na sekcje (co jest powszechne w silnikach korzystających z portali do renderowania wydajności), węzły siatki można dodawać na zasadzie sekcja po sekcji. Ten wykres jest następnie połączony i wyceniony przy użyciu technik, które analizowaliśmy do tej pory. Wykres na tym etapie jest ogromny i bardzo gęsty. Średni poziom może mieć dziesiątki milionów węzłów i setki milionów połączeń. Zazwyczaj tworzenie tego wykresu zajmuje bardzo dużo czasu przetwarzania i pamięci. Wykres można następnie uprościć, tworząc wykres z rozsądną liczbą węzłów - na przykład kilkoma tysiącami. Struktura poziomu wyrażona wyraźnie na wysokim poziomie szczegółowości zostanie do pewnego stopnia ujęta na uproszczonym wykresie. Chociaż brzmi to dość prosto, wykresy tworzone przez to podejście są często niezadowalające bez wprowadzania poprawek. Często upraszczają kluczowe informacje, które człowiek uznałby za oczywiste. Trwają badania nad lepszymi technikami upraszczania, ale zespoły, które używają tej metody w swoim łańcuchu narzędzi, niezmiennie liczą na możliwość wizualizacji i poprawiania wynikowych wykresów.

EKSPLOATACJA DANYCH

Metody eksploracji danych do tworzenia wykresów znajdują węzły, analizując dane dotyczące ruchu postaci w świecie gry. Budowane jest środowisko gry i tworzona jest geometria poziomu. Postać jest następnie umieszczana na poziomie. Postać może być pod kontrolą gracza lub może być zautomatyzowana. Gdy postać porusza się po poziomie, jej pozycja jest stale rejestrowana. Zarejestrowane dane o pozycji można następnie przeszukiwać w celu uzyskania interesujących danych. Jeśli postać wystarczająco się poruszała, większość legalnych lokalizacji na poziomie gry będzie znajdować się w pliku dziennika. Ponieważ postać w silniku gry będzie mogła wykorzystać wszystkie możliwe ruchy (skoki, latanie itd.), nie ma potrzeby wykonywania skomplikowanych obliczeń aby określić, gdzie postać może się dostać.

Obliczanie węzłów

Lokacje, w których postać często się znajduje, będą prawdopodobnie składać się z skrzyżowań i arterii na poziomie gry. Można je zidentyfikować i ustawić jako węzły w grafie odnajdywania ścieżki. Plik dziennika jest agregowany, więc pobliskie punkty dziennika są scalane w pojedyncze lokalizacje. Można to wykonać za pomocą algorytmu kondensacji z rozdziału 4 lub śledząc liczbę punktów logarytmicznych nad każdym wielokątem kondygnacji i używając punktu środkowego wielokąta (tj. za pomocą siatki nawigacji opartej na wielokątach). Chociaż może być używany z siatkami nawigacyjnymi, eksploracja danych jest zwykle używana w połączeniu z reprezentacją poziomu w domenie Dirichleta. W tym przypadku węzeł można umieścić w każdym obszarze szczytowym gęstości ruchu. Zazwyczaj wykresy mają stały rozmiar (liczba węzłów dla wykresu jest określona z góry). Algorytm następnie wybiera z wykresu taką samą liczbę lokalizacji gęstości szczytowej, tak aby żadne dwie lokalizacje nie były zbyt blisko siebie.

Obliczanie połączeń

Wykres można następnie wygenerować z tych węzłów, stosując podejście oparte na punktach widoczności lub dalszą analizę danych z pliku dziennika. Podejście oparte na punktach widoczności jest szybkie, ale nie ma gwarancji, że wybrane węzły będą znajdować się w bezpośredniej linii wzroku. Dwa obszary o dużej gęstości mogą znajdować się za rogiem od siebie. Podejście „line-of-sight" będzie błędnie zakładać, że nie ma połączenia między dwoma węzłami. Lepszym rozwiązaniem jest użycie danych połączenia z pliku dziennika. Dane z pliku dziennika można dalej analizować i obliczać trasy między różnymi węzłami. Dla każdego wpisu w pliku dziennika można obliczyć odpowiedni węzeł (przy

użyciu normalnej lokalizacji; więcej szczegółów znajduje się w Rozdziale 4). Połączenia można dodawać między węzłami, jeśli plik dziennika pokazuje, że znak został przeniesiony bezpośrednio między nimi. Daje to solidny zestaw połączeń dla grafu.

Ruch postaci

Aby zaimplementować algorytm eksploracji danych, potrzebny jest mechanizm poruszania postacią po poziomie gry. Może to być tak proste, jak sterowanie postacią przez człowieka lub granie w wersję beta gry. Jednak w większości przypadków potrzebna jest w pełni automatyczna technika. W tym przypadku postać jest kontrolowana przez AI. Najprostszym podejściem jest użycie kombinacji zachowań kierowania, aby losowo wędrować po mapie. Może to być tak proste, jak zachowanie „wędrówki", ale zwykle obejmuje dodatkowe przeszkody i omijanie ścian. W przypadku postaci, które potrafią skakać lub latać, zachowanie sterowania powinno umożliwiać postaci korzystanie z pełnego zakresu opcji ruchu. W przeciwnym razie plik dziennika będzie niekompletny, a wykres odnajdywania ścieżki nie obejmie dokładnie całego poziomu. Stworzenie tego rodzaju eksplorującej postaci jest samo w sobie trudnym zadaniem AI. Idealnie, postać będzie mogła eksplorować wszystkie obszary poziomu, nawet te, do których trudno jest dotrzeć. W rzeczywistości automatyczne eksplorowanie postaci często może utknąć i wielokrotnie eksplorować niewielki obszar poziomu. Zazwyczaj automatyczne postacie są eksplorowane tylko przez stosunkowo krótki czas (maksymalnie kilka minut gry). Aby stworzyć dokładny dziennik poziomu, postać jest za każdym razem restartowana z losowej lokalizacji. Błędy spowodowane zablokowaniem się postaci są zminimalizowane, a połączone pliki dziennika z większym prawdopodobieństwem pokrywają większość poziomu.

Ograniczenia

Minusem takiego podejścia jest czas. Aby upewnić się, że żadne regiony poziomu nie zostaną przypadkowo pozostawione niezbadanym i aby upewnić się, że wszystkie możliwe połączenia między węzłami są reprezentowane w pliku dziennika, postać będzie musiała się poruszać przez bardzo długi czas. Dzieje się tak szczególnie wtedy, gdy postać porusza się w sposób losowy lub jeśli są obszary poziomu, które wymagają precyzyjnych sekwencji skoków i innych ruchów, aby dotrzeć. Zazwyczaj przeciętny poziom gry (przejście postaci poruszającej się z pełną prędkością zajmuje około 30 sekund) wymaga zarejestrowania milionów punktów dziennika. Pod kontrolą gracza wymagana jest mniejsza liczba próbek. Gracz może wykonywać kombinacje ruchów dokładnie i wyczerpująco eksplorować wszystkie obszary poziomu. Niestety to podejście jest ograniczone czasowo: graczowi zajmuje dużo czasu, aby przejść przez wszystkie możliwe obszary poziomu we wszystkich kombinacjach. Podczas gdy postać zautomatyzowana może robić to przez całą noc, jeśli jest to wymagane (i zwykle tak jest), używanie do tego ludzkiego gracza jest marnotrawstwem. Szybciej byłoby ręcznie utworzyć w pierwszej kolejności wykres odnajdywania ścieżki. Niektórzy deweloperzy eksperymentowali z podejściem hybrydowym: automatyczne wędrowanie postaci w połączeniu z tworzonymi przez graczy plikami dziennika dla trudnych obszarów. Aktywnym obszarem badań jest zaimplementowanie wędrującej postaci, która wykorzystuje dane z poprzednich plików dziennika, aby systematycznie badać słabo zarejestrowane obszary, próbując nowatorskich kombinacji ruchów, aby dotrzeć do miejsc, które nie są obecnie badane. Dopóki nie zostanie osiągnięta wiarygodna eksploracja sztucznej inteligencji, ograniczenia tego podejścia będą oznaczać, że optymalizacja rąk będzie nadal potrzebna do konsekwentnego tworzenia użytecznych wykresów.

Inne Reprezentacje

Do tej pory opisywałem eksplorację danych w odniesieniu do reprezentacji grafów punktowych. Reprezentacje oparte na siatce nie wymagają metod eksploracji danych; węzły są wyraźnie zdefiniowane jako wielokąty w siatce. Pytaniem otwartym jest, czy za pomocą eksploracji danych

można zidentyfikować ogólne regiony ograniczające. Problem dopasowania ogólnego regionu do mapy gęstości danych dziennika jest z pewnością bardzo trudny i może być niemożliwy do wykonania w rozsądnych skalach czasowych. Dotychczas znane mi praktyczne narzędzia do eksploracji danych opierały się na reprezentacjach punktowych.

WIEDZA DLA RUCHU

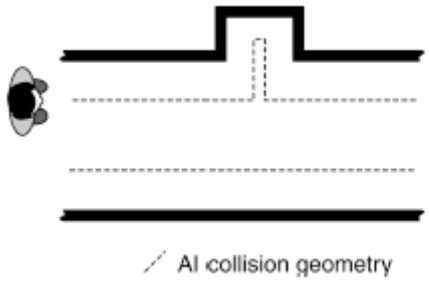
Podczas gdy taktyka odnajdywania drogi i punktów orientacyjnych stanowi najczęstszą i najtrudniejszą presję w łańcuchu narzędzi, uzyskanie danych o ruchu jest bardzo ważne.

PRZESZKODY

Sterowanie to prosty proces, gdy odbywa się na płaskiej, pustej płaszczyźnie. W środowisku wewnętrznym istnieje zazwyczaj wiele różnych ograniczeń dotyczących ruchu postaci. Postać AI musi zrozumieć, gdzie leżą ograniczenia i być w stanie odpowiednio dostosować sterowanie. Informacje te można obliczyć w czasie wykonywania, badając geometrię poziomu. W większości przypadków jest to marnotrawstwo, a do zbudowania reprezentacji specyficznej dla AI do sterowania wymagany jest etap wstępnego przetwarzania.

Ściany

Przewidywanie kolizji ze ścianami nie jest trywialnym zadaniem. Zachowania związane z kierowaniem traktują postacie jako cząsteczki bez szerokości, ale postacie nieuchronnie muszą zachowywać się tak, jakby były stałym obiektem w grze. Obliczenia kolizji można wykonać, dokonując wielu kontroli geometrii poziomu (na przykład kontroli z prawego i lewego krańca postaci). Ale może to powodować problemy ze sterowaniem i blokowanie postaci. Rozwiązaniem jest zastosowanie osobnej geometrii AI dla poziomu przesuniętego ze wszystkich ścian o promień postaci (zakładając, że postać może być reprezentowana jako kula lub cylinder). Ta geometria umożliwia obliczenie wykrywania kolizji z lokalizacjami punktów i obniża koszt przewidywania i unikania kolizji. Obliczenie tej geometrii odbywa się zwykle automatycznie za pomocą algorytmu geometrycznego. Niestety, te algorytmy często mają efekt uboczny polegający na wprowadzaniu bardzo małych wielokątów w rogach lub szczelinach, które mogą uwięzić postać. Rysunek przedstawia przypadek, w którym geometria może spowodować powstanie drobnej szczeliny, która prawdopodobnie spowoduje problemy dla agenta.

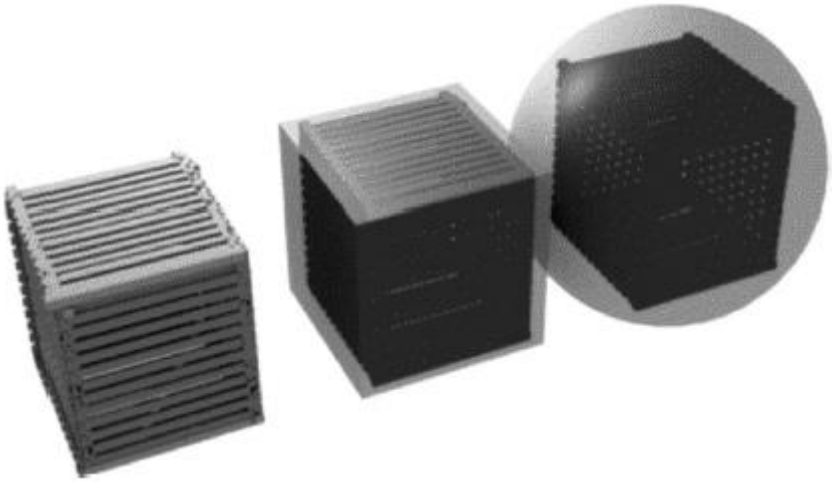


W przypadku bardzo złożonych geometrii poziomu może być wymagana wstępna uproszczona geometria kolizji lub wsparcie dla wizualizacji i modyfikacji geometrii AI w pakiecie modelowania.

Reprezentacja przeszkód

AI nie działa wydajnie z surową geometrią wielokątów poziomu. Wykrywanie przeszkód poprzez szukanie ich geometrycznie to czasochłonne zadanie, które zawsze działa słabo. Geometria kolizji jest często uproszczoną wersją geometrii renderowania. Wielu programistów korzysta ze sztucznej

inteligencji, która wyszukuje w oparciu o geometrię kolizji. Często do przeszkody należy zastosować dodatkową geometrię AI, aby można ją było łatwo ominąć. Skomplikowane kontury przedmiotu nie mają znaczenia dla postaci, która stara się go całkowicie uniknąć. Tak jak na rysunku, wystarczyłaby otaczająca całość sfera.



W miarę jak środowisko staje się bardziej złożone, wzrastają ograniczenia dotyczące ruchu postaci. Podczas gdy poruszanie się po pomieszczeniu, w którym znajduje się jedna skrzynia, jest łatwe (bez względu na to, gdzie jest skrzynia), znalezienie drogi przez pomieszczenie usiane skrzyniami jest trudniejsze. Mogą istnieć trasy przez geometrię, które są wykluczone, ponieważ sfery ograniczające nakładają się na siebie. W takim przypadku wymagana jest bardziej złożona geometria AI.

ETAP NA WYSOKIM POZIOMIE

Chociaż pierwotnie zaprojektowany do użytku w przemyśle filmowym, inscenizacja AI jest coraz częściej wykorzystywana do efektów w grach. Inscenizacja polega na koordynowaniu wydarzeń w grach opartych na ruchu. Zazwyczaj projektant poziomów umieszcza na poziomie gry wyzwalacze, które włączają lub wyłączają określone postacie. AI postaci zacznie wtedy sprawiać, by postacie działały poprawnie. Historycznie było to często widoczne dla gracza (postacie nagle ożywają, gdy gracz się zbliża), ale teraz jest to ogólnie lepiej ukryte przed wzrokiem. Inscenizacja idzie o jeden krok dalej i pozwala projektantowi poziomów ustawić akcje wysokiego poziomu dla postaci w odpowiedzi na wyzwalacze. Zwykle dotyczy to sytuacji, w których na scenie występuje wiele różnych postaci AI (takich jak rój pająków lub oddział żołnierzy). Tak ustawione działania są w przeważającej mierze związane z ruchem. Jest to zaimplementowane jako stan w narzędziu decyzyjnym postaci, w którym wykona ona ruch parametryczny (zwykle „przenieś się do tej lokalizacji", przy czym lokalizacja jest parametrem). Ten parametr można następnie ustawić w narzędziu do inscenizacji, bezpośrednio lub w wyniku wyzwalacza podczas gry. Bardziej wyrafinowana inscenizacja wymaga bardziej złożonych zestawów decyzji. Może być wspierany przez bardziej kompletne narzędzie do projektowania AI, zdolne do modyfikowania procesu decyzyjnego postaci. Zmiany stanu wewnętrznego postaci można następnie zażądać w wyniku wyzwalaczy na poziomie.

WIEDZA DO PODEJMOWANIA DECYZJI

Na najprostszym poziomie podejmowanie decyzji może być realizowane wyłącznie poprzez odpytywanie świata gry o informacje. Na przykład postać, która musi uciekać w obliczu

niebezpieczeństwa, może rozglądać się za niebezpieczeństwem przy każdej klatce i uciekać, jeśli test się sprawdzi. Ten poziom podejmowania decyzji był powszechny w grach do przełomu wieków.

RODZAJE OBIEKTÓW

Większość współczesnych gier używa pewnego rodzaju systemu przekazywania wiadomości do moderowania komunikacji. Postać będzie stać w pobliżu, dopóki nie usłyszy, że widzi niebezpieczeństwo, po czym ucieknie. W tym przypadku decyzja o tym, „co jest niebezpieczne" nie zależy od postaci; jest to właściwość gry jako całości. Pozwala to programiście zaprojektować poziom, w którym tworzone są zupełnie nowe obiekty, oznaczane jako niebezpieczne i pozycjonowane. Postać poprawnie zareaguje na te obiekty i ucieknie, bez konieczności dodatkowego programowania. Algorytm przekazywania wiadomości i AI postaci są stałe. Łańcuch narzędzi musi obsługiwać tego rodzaju dane specyficzne dla obiektu. Projektant poziomów będzie musiał oznaczyć różne obiekty, aby sztuczna inteligencja mogła zrozumieć ich znaczenie. Często nie jest to proces związany z AI. Wzmocnienie w grze platformowej, na przykład, musi być oznaczone jako kolekcjonerskie, aby gra poprawnie umożliwiła graczowi przejście do niego (w przeciwieństwie do uczynienia go nie do przebicia i odbicia się od niego). Ta „kolekcjonerska" flaga może być używana przez AI: postać może być ustawiona tak, aby broniła wszelkich pozostałych przedmiotów kolekcjonerskich przed graczem. Większość łańcuchów narzędzi opiera się na danych: pozwalają użytkownikom dodawać dodatkowe dane do definicji obiektu. Dane te mogą być wykorzystywane do podejmowania decyzji.

DZIAŁANIA ZESPOLONE

W niektórych grach akcje dostępne dla gracza zależą od obiektów w pobliżu gracza – na przykład możliwość wciśnięcia przycisku lub pociągnięcia dźwigni. W grach z bardziej złożonym procesem decyzyjnym postać może korzystać z szeregu gadżetów, technologii i przedmiotów codziennego użytku. Postać może na przykład użyć stołu jako tarczy lub spinacza do papieru, aby otworzyć zamek. Podczas gdy większość gier wciąż rezerwuje ten poziom interakcji dla gracza, gry symulacyjne ludzi prowadzą tendencję do szerszego przyjmowania postaci o szerokich kompetencjach. Aby to wspierać, przedmioty muszą powiedzieć postaci, jakie działania są w stanie wspierać. Przycisk można tylko nacisnąć. Na stół można się wspinać, popychać, rzucać lub pozbawiać nóg i używać jako tarczy. W najprostszym przypadku można to osiągnąć za pomocą dodatkowych elementów danych: na przykład wszystkie obiekty mogą mieć flagę „można wypchnąć". Wtedy postać może po prostu sprawdzić flagę. Ale ten poziom podejmowania decyzji jest zwykle związany z zachowaniem zorientowanym na cel, gdzie działania są wybierane, ponieważ postać wierzy, że pomogą one osiągnąć cel. W tym przypadku wiedza, że można naciskać zarówno przyciski, jak i tabele, nie pomaga. Postać nie rozumie, co się stanie, gdy taka akcja zostanie wykonana, więc nie może wybrać akcji, która przyczyni się do realizacji jej celów. Naciśnięcie przycisku w windzie działa zupełnie inaczej niż wciśnięcie stołu pod dziurę w dachu; osiągają bardzo różne cele. Aby wspierać zachowanie zorientowane na cel lub jakikolwiek rodzaj planowania działań, obiekty muszą komunikować znaczenie działania wraz z samym działaniem. Najczęściej znaczenie to jest po prostu listą celów, które zostaną osiągnięte (i tych, które zostaną zagrożone) w przypadku podjęcia działania . Łańcuchy narzędzi do gier ze sztuczną inteligencją zorientowaną na cel muszą traktować działania jako konkretne obiekty. Z akcją, taką jak obiekt gry w zwykłej grze, mogą być powiązane dane. Dane te obejmują zmianę stanu świata, która będzie wynikać z przeprowadzenia akcji, wraz z wymaganiami wstępnymi, informacjami o czasie i animacjach do odtworzenia. Akcje są następnie kojarzone z obiektami na poziomie.

ŁAŃCUCH NARZĘDZI

Do tej pory nakreśliłem wymagania narzędziowe poszczególnych technik AI. Aby złożyć całą grę, te poszczególne funkcje muszą zostać połączone, aby można było utworzyć wszystkie dane gry i przenieść

je do skompilowanej gry. Proces ujawniania tych różnych funkcji edycyjnych i łączenia wyników razem jest znany jako „łańcuch narzędzi”: łańcuch narzędzi wymaganych do stworzenia gotowej gry. Historycznie, aż do początku XXI wieku programiści najczęściej używali do integracji szeregu różnych narzędzi w połączeniu ze skryptami i popularnymi formatami plików. Producenci silników gier zaczęli zdawać sobie sprawę, że łańcuch narzędzi stanowi kluczowy problem dla programistów i zaczęły pojawiać się rozwiązania obejmujące zintegrowane edytory. Dwa najbardziej znane silniki gier — Unreal Engine i Unity — są obecnie zdecydowanie najczęściej używanymi łańcuchami narzędzi w branży i oba zapewniają zintegrowane aplikacje do edycji z rozszerzalnym wsparciem dla narzędzi zdefiniowanych przez programistów. Dla większości programistów silnik gry i edytor tworzą zintegrowany zestaw narzędzi. To znaczy większość, ale nie wszystkie. Wciąż istnieją studia, które wolą budować razem zestaw narzędzi z różnych narzędzi, aby napędzać własne silniki gier. Nie każdy chce używać silnika takiego jak Unity, a jeśli silnik nie jest używany, nie ma powodu, aby używać oprzyrządowania. Ta sekcja zawiera krótki przegląd elementów kompletnego łańcucha narzędzi związanych ze sztuczną inteligencją, zarówno dla użytkowników znanych silników gier, jak i tych, którzy zdecydują się pracować samodzielnie. Uwzględnia szereg rozwiązań, od niestandardowych rozszerzeń dla edytorów silników gier, przez niezależne narzędzia do edycji zachowań, po wtyczki do oprogramowania do modelowania 3D.

ZINTEGROWANE SILNIKI DO GRY

Na początku rozdziału krótko wspomniałem o Renderware Studio, wczesnym, kompletnym narzędziu i aplikacji do edycji dla licencjonowanego silnika Renderware (która została zakupiona jako technologia wewnętrzna i wycofana z rynku przez Electronic Arts w 2004 roku). Jego główni konkurenci, Gamebryo i Unreal Engine, dodali także własne zintegrowane edytory, a także wiele zintegrowanych systemów u większych deweloperów. Ale te silniki zdolne do tworzenia najnowocześniejszych gier były sprzedawane w uznanych studiach deweloperskich, z odpowiednimi warunkami licencyjnymi i kosztami. To Unity, począwszy od 2005 roku, obniżyło bariery w korzystaniu i przyniosło te same korzyści większemu rynkowi studiów, ostatecznie wspierając rozwijającą się branżę niezależnych twórców gier i hobbystów. Chociaż pierwotnie była przeznaczona tylko dla komputerów Mac i mierzyła się z konkurencją ze strony innych zintegrowanych narzędzi do gier zorientowanych na komputery PC, Unity niezaprzeczalnie wygrał. Jest to obecnie najczęściej używany silnik gier na świecie, z milionami programistów (prawdopodobnie dziesiątki milionów, aktualne liczby są trudne do zdobycia), a jego model rozszerzalnego edytora poziomów ma stać się de facto standardem.

Wbudowane narzędzia

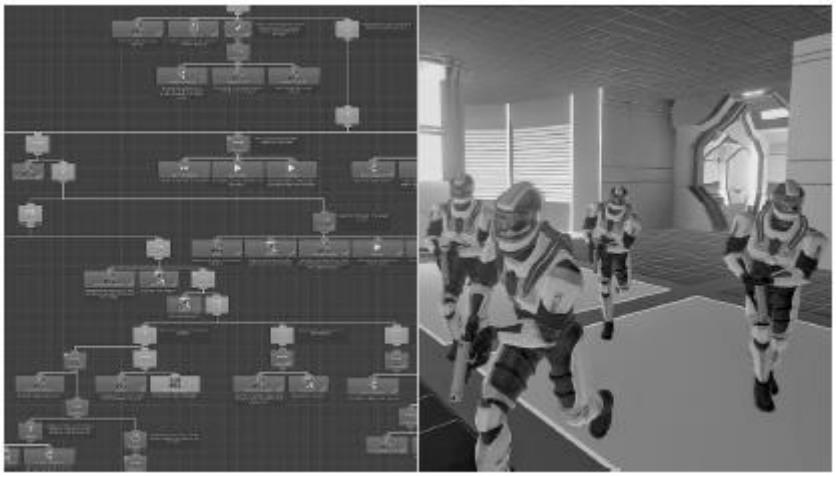
Unity i podobne silniki (najpopularniejszy jest Unreal Engine) zapewniają podstawowy zestaw narzędzi po wyjęciu z pudełka. aplikacje edytora. W najprostszym przypadku możliwe jest powiązanie dodatkowych danych z obiektami na poziomie. Można zdefiniować niestandardowe przedziały danych, a autor poziomu może wypełnić te przedziały danymi. Może to obejmować wartość odbioru, trudność nawigacji na powierzchni lub umiejscowienie punktu osłony. Wszystkie silniki obsługują obiekty, które nie są widoczne dla użytkownika, które mogą być używane do przechowywania danych globalnych, aby wszystkie znaki miały dostęp; lub umieszczone niewidocznie na scenie, aby reprezentować punkt orientacyjny lub inną zlokalizowaną informację. Te narzędzia to zwykle proste listy par klucz-wartość, w których autor poziomu może edytować wartość. Zarówno Unity, jak i Unreal Engine zapewniają wbudowaną obsługę wyszukiwania ścieżek i nawigacji. I obaj ujawniają to w edytorze, umożliwiając wizualizację i modyfikację danych wyszukiwania ścieżek.

Ponadto, zwykle udostępniany jest niestandardowy edytor automatów stanów do precyzyjnego dostrajania animacji. Uzyskane automaty stanów mogą być dostępne programowo i mogą być

używane jako podstawa sztucznej inteligencji automatu stanów. Chociaż, ponieważ nie są zoptymalizowane do tego celu, mogą być zbyt nieporęczne w porównaniu z dedykowaną wtyczką (patrz poniżej). Dodatkowe narzędzia są dostępne dla aplikacji innych niż AI, takich jak efekty cząsteczkowe, miksowanie dźwięku i obsługa czcionek, ale zazwyczaj nie zapewnia się pierwszorzędnej obsługi narzędzi dla AI poza nawigacją. W tym celu potrzebna jest wtyczka.

Wtyczki edytora

Sprzedawcy silników gier nie mogą przewidzieć wszystkich wymagań każdej gry zbudowanej przy użyciu ich systemu. Zamiast tego obsługują systemy wtyczek, w których niestandardowy kod może nakładać dane na widok głównego poziomu lub mieć wyłączny dostęp do okna, w którym można narysować własny interfejs użytkownika. Wymagania dotyczące narzędzi w dalszej części tej sekcji są zazwyczaj implementowane jako wtyczki do modyfikacji zachowania edytora. Zarówno Unreal Engine, jak i Unity mają duży sklep internetowy, w którym deweloperzy mogą sprzedawać swoje wtyczki. Są szanse, że w sklepie jest już dostępne narzędzie, które przybliża potrzeby Twojej gry. Chociaż może wymagać poprawek, może być przynajmniej wykorzystany do szybkiego prototypowania i jako inspiracja dla bardziej spersonalizowanego rozwiązania. Rysunek przedstawia zrzut ekranu takiego narzędzia skoncentrowanego na tworzeniu i uruchamianiu drzew zachowań.



Redakcja skryptów

Duża część mocy zintegrowanych edytorów pochodzi z ich języka skryptowego. Zintegrowane skrypty unikają potrzeby implementowania całej logiki gry w języku niskiego poziomu, takim jak C++. Jak opisano w następnym rozdziale, możliwość pisania fragmentów kodu zmniejsza potrzebę stosowania prostszych technik sztucznej inteligencji do podejmowania decyzji. Zamiast drzewa decyzyjnego, zbudowanego przy użyciu narzędzia do edycji drzewa decyzyjnego, ten sam efekt może mieć skrypt z listą instrukcji if. Skrypty napisane w kodzie źródłowym wymagają do napisania programisty. To może być w porządku dla niezależnego studia, w którym projektant gry również ma za zadanie programować grę, ale w większych zespołach o większej specjalizacji może to być niewykonalne. Na szczęście istnieją bardziej przyjazne alternatywy w zasięgu technicznie myślących nie-programistów. Po wyjęciu z pudełka, Unreal Engine (począwszy od wersji 4) obsługuje Blueprints, wizualny język programowania, z dużą elastycznością narzędzia do tworzenia maszyn stanów AI, drzew zachowań lub drzew decyzyjnych. Podobne narzędzia są dostępne w sklepie zasobów Unity. W chwili pisania tego tekstu najpopularniejszy jest Playmaker firmy Hutong Games LLC, choć istnieje kilka alternatyw.

NIESTANDARDOWE EDYTORY OPARTE NA DANYCH

Deweloperzy korzystający z własnego silnika muszą uwzględnić tworzenie niestandardowych edytorów w celu przygotowania swoich danych. Sztuczna inteligencja nie jest jedynym obszarem, w którym do osiągnięcia poziomu gry wymagana jest ogromna ilość dodatkowych danych. Logika gry, fizyka, sieć i dźwięk wymagają własnych zestawów informacji. Możliwe jest stworzenie kilku oddzielnych narzędzi do generowania tych danych, ale większość programistów podąża za modelem Unity lub Unreal i przenosi wszystkie funkcje edycyjne do jednej aplikacji. Można to następnie wykorzystać ponownie we wszystkich ich grach. Posiadanie takiego narzędzia zapewnia elastyczność implementacji złożonych funkcji edycyjnych ściśle zintegrowanych z ostatecznym kodem gry, w sposób, który byłby trudny w istniejącym edytorze. Mój były klient przyjął to podejście, ponieważ ich silnik gry opierał się na danych poziomu mapowanych w pamięci, w których wszystkie informacje wymagane do odtworzenia poziomu (modele, tekstury, sztuczna inteligencja, dźwięk itp.) były przechowywane w pamięci dokładnie w sposób, w jaki ich kod C++ tego potrzebował, skracając czas ładowania do punktu, w którym nowe sekcje poziomu mogą być ładowane w tle podczas gry. Obecni redaktorzy silników byli zbyt zdeterminowani, jak przechowują swoje dane, więc firma opracowała własne od podstaw. Bez względu na motywację, ten rodzaj kompletnego pakietu do edycji poziomów jest często nazywany „napędzanym danymi" lub „zorientowanym obiektowo". Z każdym obiektem w świecie gry powiązany jest zestaw danych. Ten zestaw danych kontroluje zachowanie obiektu — sposób, w jaki jest traktowany przez logikę gry. W tym kontekście stosunkowo łatwo jest wspierać edycję danych AI. Często jest to kwestia dodania kilku dodatkowych typów danych dla każdego obiektu (takich jak oznaczenie niektórych obiektów jako „do uniknięcia", a innych jako „do zebrania"). Tworzenie tego typu narzędzi to duży projekt rozwojowy i nie jest opcją dla małych studiów, zespołów self-publishingu lub hobbystów. Nawet w przypadku zespołów dysponujących takim narzędziem istnieją ograniczenia w podejściu opartym na danych. Tworzenie sztucznej inteligencji postaci to nie tylko kwestia ustawienia zestawu wartości parametrów. Różne postacie wymagają różnej logiki podejmowania decyzji i umiejętności łączenia kilku różnych zachowań w celu wybrania właściwego we właściwym czasie. Wymaga to specjalnego narzędzia do projektowania AI (chociaż takie narzędzia są często zintegrowane z edytorem opartym na danych).

NARZĘDZIA PROJEKTOWANIA AI

W ten sam sposób, w jaki współczesne silniki gier zapewniają pewne narzędzia w standardzie (a mianowicie możliwość oznaczania poziomu i narzędzia nawigacyjne), najpopularniejsze narzędzia niestandardowe umożliwiają sztucznej inteligencji lepsze zrozumienie poziomu gry i uzyskanie dostępu do potrzebnych informacji podejmować rozsądne decyzje. Wraz ze wzrostem złożoności technik sztucznej inteligencji, a projektanci poziomów mają za zadanie implementować więcej zachowań, programiści szukają sposobów, aby umożliwić projektantom poziomów dostęp do sztucznej inteligencji umieszczanych przez nich postaci. Na przykład projektanci poziomów tworzący scenariusz w laboratorium wewnętrznym mogą potrzebować stworzyć kilka różnych postaci strażników. Będą musieli zapewnić im różne trasy patrolowe, różne zdolności wyczuwania intruzów i różne zestawy zachowań, gdy wykryją gracza. Umożliwienie projektantom poziomów posiadania tego rodzaju kontroli wymaga specjalistycznych narzędzi do projektowania AI. Bez narzędzia projektant musi polegać na programistach, którzy wprowadzają modyfikacje AI i ustawiają postacie z ich odpowiednimi zachowaniami.

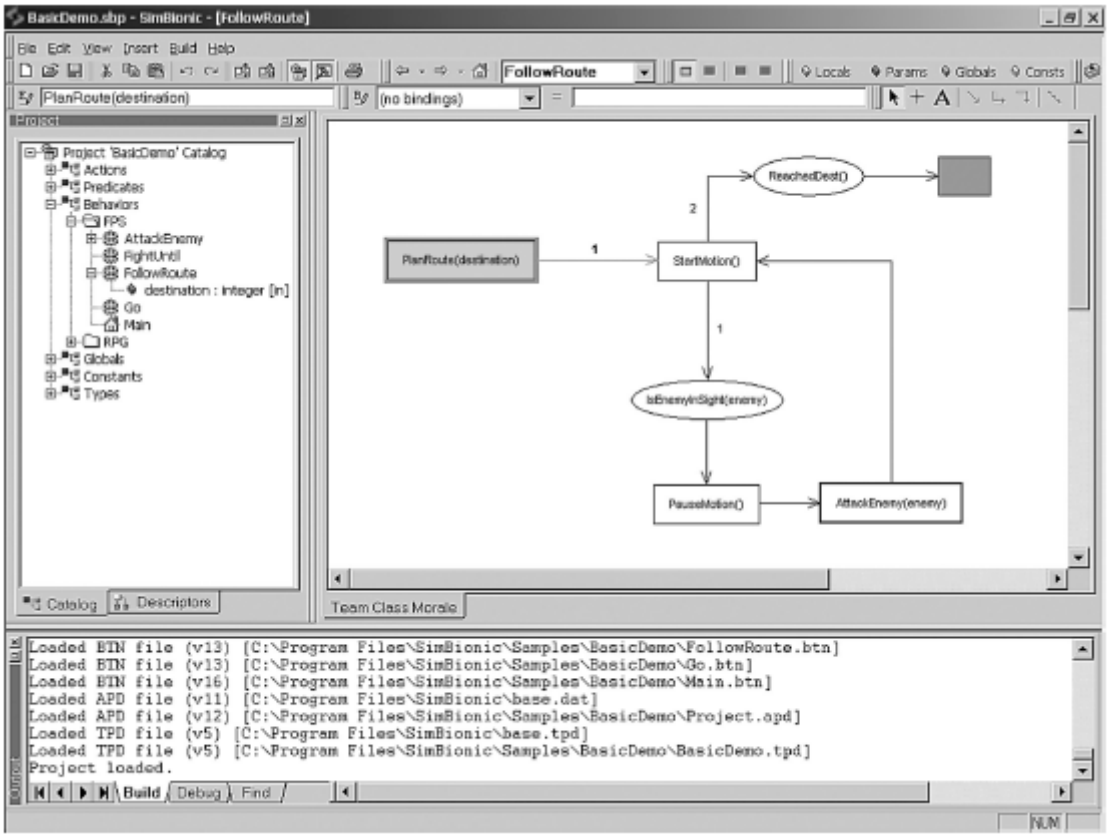
Narzędzia do tworzenia skryptów

Pierwsze narzędzia wspierające ten rodzaj rozwoju były oparte na językach skryptowych. Skrypty można edytować bez ponownej kompilacji i często można je łatwo przetestować. Wiele silników gier

obsługujących skrypty zapewnia mechanizmy edycji, debugowania i przechodzenia przez skrypty. Zostało to wykorzystane głównie do opracowania logiki na poziomie gry (takiej jak otwieranie drzwi w odpowiedzi na naciśnięcia przycisków itp.). Jednak wraz z ewolucją sztucznej inteligencji z tego poziomu, języki skryptowe zostały rozszerzone, aby ją obsługiwać. Języki skryptowe borykają się z problemem bycia językami programowania. Nietechniczni projektanci poziomów mogą mieć trudności z tworzeniem złożonych skryptów do kontrolowania sztucznej inteligencji postaci.

Projektanci maszyn stanowych

W połowie 2000 roku narzędzia wspierające łączenie gotowych zachowań stały się powszechnie dostępne. Do tej kategorii należą niektóre komercyjne narzędzia pośredniczące, takie jak wsparcie AI Havok, a także kilka narzędzi typu open source i narzędzi wewnętrznych stworzonych przez dużych programistów i wydawców. Narzędzia te umożliwiają projektantowi poziomów łączenie palety zachowań AI. Postać może potrzebować patrolować trasę, dopóki nie usłyszy alarmu i na przykład nie przeprowadzi dochodzenia. Zachowania typu „patrol" i „badania" zostałyby stworzone przez zespół programistów i wystawione na działanie narzędzia AI. Projektant poziomów wybierałby je następnie i łączył z procesem podejmowania decyzji, który zależy od stanu syreny. Akcje wybrane przez projektanta poziomów to często niewiele więcej niż sterowanie zachowaniami. Jak omówiono w rozdziale 3, często jest to wszystko, co jest wymagane do większości zachowań postaci w grze. Proces podejmowania decyzji preferowany przez to podejście to maszyny stanów, przy czym drzewa zachowań stanowią bardziej wyrafinowaną opcję mniejszościową. Rysunek przedstawia zrzut ekranu narzędzia SimBionic FSM, pierwotnie komercyjnej oferty oprogramowania pośredniczącego, teraz open source.



Najlepsze narzędzia tego typu zawierają obsługę debugowania języka skryptowego, umożliwiając edytorom poziomów przechodzenie przez działanie FSM, wizualnie widząc aktualny stan postaci i będąc w stanie ręcznie ustawić ich wewnętrzne właściwości.

ZDALNE DEBUGOWANIE

Wydobycie informacji z gry w czasie wykonywania ma kluczowe znaczenie dla diagnozowania rodzaju problemu AI, który nie pojawia się w izolowanych testach. Zazwyczaj programiści dodają kod debugowania, aby zgłaszać wewnętrzny stan gry podczas jej odtwarzania. Można to wyświetlić na ekranie lub zapisać w pliku i przeanalizować pod kątem źródła błędów. Działając na komputerze, stosunkowo łatwo jest dostać się do działającej gry. Narzędzia do debugowania mogą dołączać się do gry i zgłaszać szczegóły jej stanu wewnętrznego. Na konsolach i urządzeniach mobilnych istnieją narzędzia do zdalnego debugowania umożliwiające połączenie komputera deweloperskiego ze sprzętem testowym. Chociaż można wiele zrobić za pomocą tego rodzaju inspekcji, bardziej wyrafinowane narzędzia do debugowania są prawie zawsze wymagane do wszystkiego, poza najprostszymi grami. Analiza lokalizacji pamięci lub wartości zmiennych jest przydatna w przypadku niektórych zadań debugowania. Jednak trudno jest ustalić, jak reaguje złożona maszyna stanów, i niemożliwe jest zrozumienie działania sieci neuronowej. Narzędzie do debugowania może dołączyć do uruchomionej gry, aby odczytywać i zapisywać dane dla sztucznej inteligencji (oraz wszelkie inne działania w grze, jeśli o to chodzi). Jednym z najczęstszych zastosowań zdalnego debugowania jest wizualizacja wewnętrznego stanu sztucznej inteligencji, czy to decyzji drzewa zachowań, wyjścia sieci neuronowej, czy też testów linii wzroku wykonywanych przez zarządzanie zmysłami. Często w połączeniu z narzędziami, które edytują te dane, pozwala to programiście widzieć i modyfikować postacie w grze podczas ich działania, a czasem nawet wprowadzać określone zdarzenia do mechanizmu zarządzania zdarzeniami. Zdalne debugowanie wymaga, aby aplikacja debugująca była uruchomiona na komputerze PC, komunikowała się z grą przez sieć lub była uruchomiona na innym komputerze, urządzeniu mobilnym lub konsoli (lub czasami ten sam komputer). Ta komunikacja sieciowa może powodować problemy z niezawodnością i synchronizacją danych (stan gry mógł odbiegać od tego, na co patrzy deweloper). Chociaż poprawiło się to w ciągu ostatniej dekady wraz z przejściem wszystkich platform na stałą łączność z Internetem, niektóre starsze urządzenia nie obsługują ogólnej komunikacji sieciowej odpowiedniej dla tego rodzaju narzędzia. Platformy połączone z Internetem dają również możliwość dostępu do informacji debugowania działających na komputerach graczy po wydaniu gry. Mogą pojawić się prawne problemy z prywatnością, ale stało się to standardowym narzędziem, którego wielu programistów używa do optymalizacji swojej gry w kolejnych łatkach. Dostępnych jest kilka gotowych rozwiązań do analizy w grze, szczególnie na urządzeniach mobilnych.

WTYCZKI

Chociaż niestandardowe narzędzia do edycji poziomów są teraz prawie wszechobecne, projektowanie 3D, modelowanie i teksturowanie są nadal w przeważającej mierze wykonywane w wysokiej klasy pakiecie do modelowania, takim jak Autodesk 3ds Max lub Maya, z otwartym pakietem Blender popularnym wśród mniejszych niezależnych programistów i hobbystów. Każde z tych narzędzi ma zestaw programistyczny (SDK), który umożliwia nowe funkcjonalność do zaimplementowania w postaci narzędzi typu plug-in. Dzięki temu programiści mogą dodawać narzędzia w postaci wtyczek do przechwytywania danych AI. Wtyczki napisane za pomocą SDK są pakowane w biblioteki. Czasami są napisane w C/C++, ale częściej w języku skryptowym dostarczonym przez pakiet (na przykład MAXScript dla 3DS Max i Python dla Blendera). Wewnętrzne działanie każdego pakietu oprogramowania nakłada znaczne ograniczenia na architekturę wtyczek. Integracja z istniejącymi narzędziami w aplikacji jest trudna, a połączenie z systemem cofania, interfejsem użytkownika

oprogramowania i wewnętrznym formatem danych może być nieintuicyjne. Ponieważ każde narzędzie jest tak radykalnie różne, a pakiety SDK mają bardzo odmienną architekturę, to, czego się uczysz, tworząc dla jednego, często nie zostanie przetłumaczone. W przypadku wsparcia AI kandydaci do tworzenia wtyczek są tacy sami, jak funkcjonalność wymagana w narzędziu do edycji poziomów. Ponieważ nastąpiła tak znaczna zmiana w kierunku dedykowanych narzędzi do edycji poziomów, mniej programistów tworzy wtyczki AI do oprogramowania do modelowania 3D.