INTERFEJS ZE ŚWIATEM

Jedną z najtrudniejszych rzeczy do naprawienia jako programista AI jest interakcja między sztuczną inteligencją a światem gry. Każda postać musi we właściwym czasie uzyskać informacje, które w miarę możliwości zna lub odbiera ze świata gry, aby mogła na nich działać. Ponadto niektóre algorytmy muszą mieć informacje ze świata reprezentowane we właściwy sposób, aby mogły prawidłowo przetwarzać.

Aby zbudować uniwersalny system sztucznej inteligencji, potrzebujemy infrastruktury, która ułatwi pozyskiwanie właściwych informacji do właściwych bitów kodu sztucznej inteligencji we właściwym formacie we właściwym czasie. Dzięki sztucznej inteligencji specjalnego przeznaczenia, przeznaczonej do pojedynczej gry, może nie być linii podziału między interfejsem świata a kodem sztucznej inteligencji: jeśli sztuczna inteligencja potrzebuje pewnych informacji, może udać się i znaleźć je tam i wtedy. Jednak w silniku zaprojektowanym do obsługi sztucznej inteligencji w wielu grach jest to niezbędne do stabilności i możliwość ponownego wykorzystania w celu posiadania jednego centralnego systemu interfejsów świata. Nawet w jednej grze może znacznie wspomóc debugowanie, aby wszystkie informacje przepływały przez centralne centrum, umożliwiając ich wizualizację, rejestrację i kontrolę. W tej Części przyjrzymy się tworzeniu solidnych i wielokrotnego użytku interfejsów świata przy użyciu dwóch różnych technik: przekazywania zdarzeń i odpytywania. System przekazywania zdarzeń zostanie rozszerzony o symulację percepcji sensorycznej, która jest gorącym tematem w sztucznej inteligencji gier obecnej generacji.

KOMUNIKACJA

Łatwo jest zaimplementować postać, która zajmuje się własnymi sprawami, nie zwracając uwagi na otaczający ją świat i inne postacie w grze: strażnicy mogą podążać trasami patrolowymi, jednostki wojskowe mogą poruszać się bezpośrednio tam, gdzie im powiedzą, a postacie niezależne może zignorować gracza. Ale to nie wyglądałoby zbyt realistycznie ani nie byłoby zabawne. Rzeczy, które dzieją się w świecie gry, wymagają prawidłowego działania, a agenci muszą wiedzieć, co dzieje się z nimi samymi, ich kolegami i wrogami. Najprostszym sposobem na wydobycie informacji ze świata gry jest ich poszukanie. Jeśli postać musi na przykład wiedzieć, czy w pobliżu słychać syrenę, kod AI postaci może bezpośrednio zapytać o stan syreny i dowiedzieć się. Podobnie, jeśli postać musi wiedzieć, czy zderzy się z inną postacią, może spojrzeć na pozycje każdej postaci i obliczyć jej trajektorię. Porównując tę trajektorię z jej własną, postać może określić, kiedy nastąpi kolizja i podjąć kroki, aby jej uniknąć.

ODPYTANIE

Szukanie interesujących informacji nazywa się ankietowaniem. Kod AI odpytuje różne elementy stanu gry, aby określić, czy jest coś interesującego, nad czym musi działać. Ten proces jest bardzo szybki i łatwy do wdrożenia. Sztuczna inteligencja dokładnie wie, czym się interesuje i może się o tym od razu dowiedzieć. Nie ma specjalnej infrastruktury ani algorytmu między danymi a sztuczną inteligencją, która ich potrzebuje. Jednak wraz ze wzrostem liczby potencjalnie interesujących rzeczy sztuczna inteligencja spędza większość czasu na sprawdzaniu, które zwrócą wynik negatywny. Na przykład, syrena jest prawdopodobnie bardziej wyłączona niż włączona, a postać prawdopodobnie nie będzie kolidować z więcej niż jednym innym znakiem na klatkę. Odpytywanie może gwałtownie wzrosnąć w wymaganiach dotyczących przetwarzania dzięki czystym liczbom, mimo że każda kontrola może być bardzo szybka. W przypadku kontroli, które trzeba przeprowadzić między postacią a wieloma podobnymi źródłami informacji, czas szybko się mnoży. Dla poziomu ze 100 postaciami, 10 000 kontroli trajektorii byłyby potrzebne do przewidzenia wszelkich kolizji. Najprostsza implementacja odpytywania polega na tym, że każdy algorytm AI żąda informacji, których potrzebuje, kiedy tylko tego potrzebuje. Ponieważ każda postać żąda własnych informacji w sposób (ad hoc), ankiety mogą utrudnić

śledzenie, gdzie informacje przepływają w grze. Próba debugowania gry, w której informacje docierają do wielu różnych lokalizacji, może być trudna.

Lokale wyborcze

Istnieją sposoby na ułatwienie utrzymania technik odpytywania. Stacja wyborcza może służyć jako centralne miejsce żądań i odpowiedzi w celu debugowania. Może być również używany do buforowania danych, dzięki czemu złożone kontrole nie muszą być powtarzane dla każdego żądania. W dalszej części przyjrzymy się dokładniej lokalom wyborczym.

WYDARZENIA

Istnieje wiele sytuacji, takich jak przykład pojedynczej syreny, w których podejście odpytujące może być optymalne. Jednak w przykładzie kolizji każda para znaków jest sprawdzana dwukrotnie: raz z perspektywy pierwszej postaci i raz z perspektywy drugiej. Nawet jeśli zostały one zapisane w pamięci podręcznej, odpytywanie działa nieoptymalnie. Istnieją szybsze sposoby na rozważenie całej grupy postaci za jednym razem i wygenerowanie wszystkich kolizji naraz, zamiast przechodzenia przez agenta listy przez agenta. W takich przypadkach chcemy centralnego systemu sprawdzającego, który może powiadomić każdą postać, gdy wydarzy się coś ważnego. To jest mechanizm przekazywania zdarzeń. Centralny algorytm wyszukuje interesujące informacje i podaje fragmenty kodu, które mogą na tym skorzystać, gdy coś znajdzie. W przykładzie syreny można zastosować mechanizm zdarzenia. W każdej klatce, gdy syrena brzmi, kod kontrolny przekazuje zdarzenie do każdego znaku, który znajduje się w zasięgu słuchu. To podejście jest stosowane, gdy chcemy bardziej szczegółowo zasymulować percepcję postaci, jak zobaczymy w dalszej części. Mechanizm zdarzeń nie jest w zasadzie szybszy niż odpytywanie. Odpytywanie ma złą reputację ze względu na szybkość, ale w wielu przypadkach przekazywanie zdarzeń będzie równie nieefektywne. Aby ustalić, czy zdarzenie miało miejsce, należy przeprowadzić kontrole. Mechanizm zdarzeń nadal wymaga sprawdzenia, tak samo jak w przypadku odpytywania. Korzyść pojawia się, ponieważ w wielu przypadkach mechanizm zdarzeń może zmniejszyć wysiłek, wykonując jednocześnie wszystkie kontrole. Jeśli jednak nie ma możliwości udostępnienia wyników, zajmie to tyle samo czasu, co sprawdzenie każdego znaku. W rzeczywistości, dzięki dodatkowemu kodowi przekazującemu wiadomości, podejście do zarządzania zdarzeniami będzie wolniejsze. Wyobraź sobie sztuczną inteligencję na przykładzie syreny. Event manager musi wiedzieć, że postać interesuje się syreną. Gdy syrena wyje, menedżer zdarzeń wysyła zdarzenie do postaci. Postać prawdopodobnie nie wykonuje dokładnie tego fragmentu kodu, który musi wiedzieć o syrenie, więc przechowuje zdarzenie. Kiedy dotrze do kluczowej sekcji, znajduje zapisane zdarzenie i odpowiada na nie. Wiele przetwarzania zostało dodane przez wysłanie zdarzenia. Jeśli postać odpytywała syrenę, otrzymałaby potrzebne informacje dokładnie wtedy, gdy jej potrzebowała. Warto pamiętać, że wydarzenia nie są panaceum. Gdy nie możesz udostępnić wyników kontroli, przekazywanie zdarzeń może być znacznie wolniejsze.

Menedżerowie wydarzeń

Przekazywanie zdarzeń jest zwykle zarządzane za pomocą prostego zestawu procedur, które same sprawdzają zdarzenia lub mogą być powiadamiane o ich wystąpieniu za pomocą kodu gry, a następnie wysyłają je do zainteresowanych stron. Menedżerowie wydarzeń tworzą scentralizowany mechanizm, przez który przechodzą wszystkie wydarzenia. Śledzą zainteresowania postaci (więc otrzymują tylko zdarzenia, które są dla nich przydatne) i mogą kolejkować zdarzenia w wielu klatkach, aby płynnie korzystać z procesora. Scentralizowane przekazywanie zdarzeń ma znaczące zalety w zakresie modułowości kodu i debugowania. Ponieważ wszystkie warunki są sprawdzane w centralnej lokalizacji, łatwo jest przechowywać rejestr wykonanych kontroli i ich wyników. Zdarzenia przekazywane do

każdego znaku można łatwo wyświetlić lub zarejestrować, co znacznie ułatwia debugowanie złożonych decyzji.

OKREŚLANIE KTÓRE PODEJŚCIA WYKORZYSTAĆ

Jak we wszystkich sprawach, tutaj należy dokonać kompromisu. Z jednej strony odpytywanie może być bardzo szybkie, ale nie skaluje się dobrze. Z drugiej strony przekazywanie zdarzeń ma dodatkowy kod do napisania i jest przesadą w prostych sytuacjach. Dla samej szybkości wykonania podejście, które zapewni najlepszą wydajność, zależy od aplikacji. Trudno przewidzieć z góry. Zgodnie z ogólną zasadą, jeśli wiele podobnych postaci musi znać tę samą informację, często szybciej jest użyć zdarzeń. Jeśli postacie potrzebują tylko od czasu do czasu znać informacje (na przykład, gdy są w określonym stanie), sondowanie będzie szybsze. Chociaż połączenie niektórych sondowań i przekazywania niektórych zdarzeń jest często najszybszym rozwiązaniem, ma to wpływ na tworzenie kodu. Informacje są gromadzone i wysyłane na wiele sposobów i może być trudno ustalić, co i gdzie jest robione. Niezależnie od szybkości, niektórzy programiści uważają, że łatwiej jest zarządzać informacjami o grze tylko za pomocą wydarzeń. Możesz na przykład wydrukować wszystkie zdarzenia na ekranie i użyć ich do debugowania. Możesz skonfigurować specjalne naciśnięcia klawiszy w grze, aby ręcznie uruchamiać zdarzenia i sprawdzać, czy sztuczna inteligencja reaguje poprawnie. Dodatkowa elastyczność oraz fakt, że kod jest często łatwiejszy do zmiany i aktualizacji oznacza, że zdarzenia są często preferowane, nawet jeśli nie są najszybszym podejściem. Ogólnie jednak wymagane jest zwykle przeprowadzanie ankiet, aby uniknąć przeskakiwania przez głupie obręcze w celu uzyskania informacji do sztucznej inteligencji. Jeśli wszystkie pozostałe odpytywanie można przekierować przez stację wyborczą, można uzyskać znaczną poprawę szybkości i debugowania.

MENEDŻEROWIE WYDARZEŃ

Podejście do komunikacji oparte na zdarzeniach jest scentralizowane. Istnieje centralny mechanizm sprawdzający, który powiadamia dowolną liczbę znaków, gdy dzieje się coś interesującego. Kod, który to robi, nazywa się menedżerem zdarzeń. Event manager składa się z czterech elementów:

1. Silnik sprawdzający (może być opcjonalny)

2. Kolejka wydarzeń

3. Rejestr odbiorców wydarzenia

4. Dyspozytor zdarzeń

Zainteresowane postacie, które chcą otrzymywać zdarzenia, są często nazywane „słuchaczami”, ponieważ nasłuchują, jak coś się wydarzy. Nie oznacza to, że interesują ich tylko symulowane dźwięki. Zdarzenia mogą reprezentować wzrok, łączność radiową, określone godziny (na przykład postać wraca do domu o 17:00) lub inne dane z gry. Silnik sprawdzający musi określić, czy wydarzyło się coś, czym może zainteresować się jeden z jego słuchaczy. Może po prostu sprawdzić wszystkie stany gry pod kątem rzeczy, które mogą zainteresować dowolną postać, ale może to być zbyt dużo pracy. Wydajniejsze silniki sprawdzające uwzględniają zainteresowania słuchaczy. Silnik sprawdzający często musi współpracować z innymi usługami dostarczanymi przez grę. Jeśli postać musi wiedzieć, czy wpadła na ścianę, silnik sprawdzający może potrzebować użyć silnika fizyki lub detektora kolizji, aby uzyskać wynik. Istnieje wiele możliwych rzeczy do sprawdzenia, a wiele z nich jest sprawdzanych na różne sposoby: syrenę można sprawdzić, patrząc na pojedynczą wartość logiczną (włączone lub wyłączone), kolizje mogą być przewidywane przez algorytm geometryczny i mowę silnik rozpoznawania może wymagać skanowania poleceń głosowych gracza w celu uzyskania poleceń. Z tego powodu wyspecjalizowane menedżery zdarzeń sprawdzają tylko niektóre rodzaje informacji (takie jak kolizje,

dźwięk lub stan przełączników na poziomie. W wielu przypadkach w ogóle nie trzeba sprawdzać. Na przykład w drużynie wojskowej postacie mogą zdecydować, że powiedzą sobie nawzajem, kiedy są gotowe do bitwy. Jeśli postacie zostaną zaimplementowane przy użyciu automatów skończonych, ich „stan bitwy" stanie się aktywny i będą mogły bezpośrednio wysłać zdarzenie „gotowości do bitwy" do menedżera wydarzeń. Zdarzenia te są umieszczane w kolejce zdarzeń i jak zwykle wysyłane do odpowiednich detektorów. Powszechne jest również oddzielanie mechanizmu sprawdzania od menedżera zdarzeń. Oddzielny fragment kodu sprawdza co kilka ramek, a jeśli dojdzie do sprawdzenia, wysyła zdarzenie bezpośrednio do menedżera zdarzeń. Menedżer zdarzeń następnie przetwarza je w normalny sposób. Ten mechanizm sprawdzania polega na odpytywaniu stanu gry (w taki sam sposób, w jaki postać może odpytywać stan gry) i udostępnianiu jego wyników zainteresowanym postaciom. Implementacja menedżera zdarzeń obejmuje metodę, którą można wywołać w celu bezpośredniego umieszczenia zdarzenia w kolejce zdarzeń. Oprócz kontroli zdarzenia mogą również wynikać z kodu rozgrywki. Na przykład, gdy wynik jest zwiększany, kod wykonujący tę inkrementację może już wiedzieć, że postać może teraz awansować. Nie ma sensu przeprowadzać oddzielnej kontroli w tym przypadku. Przyrost wyniku może zamiast tego wysłać wydarzenie bezpośrednio do menedżera wydarzeń.

W przypadku kolejki zdarzeń po powiadomieniu menedżera zdarzeń o zdarzeniu (poprzez bezpośrednie przekazanie lub sprawdzenie), należy je wstrzymać do czasu, aż będzie można je rozesłać. Zdarzenie będzie reprezentowane jako struktura danych zdarzenia, a poniżej przyjrzymy się jego implementacji. Prosty menedżer zdarzeń wyśle każde zdarzenie w momencie jego wystąpienia, pozostawiając detektorom zdarzeń odpowiednią reakcję. Jest to podejście najczęściej stosowane w menedżerach zdarzeń: nie ma narzutu na przechowywanie w celu utrzymania kolejki zdarzeń i nie wymaga złożonego kodu zarządzania kolejką. Bardziej złożone menedżery zdarzeń mogą śledzić zdarzenia w kolejce i wysyłać je do słuchaczy w najlepszym momencie. Dzięki temu menedżer zdarzeń może działać jako algorytm w dowolnym momencie (patrz Część 10), wysyłając zdarzenia tylko wtedy, gdy AI ma czas pozostały w swoim budżecie przetwarzania. Jest to szczególnie ważne przy transmitowaniu wielu wydarzeń do wielu postaci. Jeśli powiadomienia nie można podzielić na wiele ramek, niektóre ramki będą miały znacznie większe obciążenie AI niż inne.

Oparte na czasie kolejkowanie wydarzeń może być bardzo złożone i obejmować wydarzenia o różnych priorytetach i terminach dostawy. Powiadomienie postaci, że zabrzmi syrena, może zostać opóźnione o kilka sekund, ale powiadomienie postaci, że została zastrzelona, powinno być natychmiastowe (zwłaszcza jeśli kontroler animacji polega na tym zdarzeniu, aby rozpocząć animację „kostki"). Rejestr detektorów umożliwia menedżerowi zdarzeń przekazywanie prawidłowych zdarzeń do właściwych detektorów. W przypadku menedżerów zdarzeń, które mają wyspecjalizowany cel (np. określanie kolizji), słuchacze mogą być zainteresowani każdym zdarzeniem, które menedżer jest w stanie wygenerować. W przypadku innych (np. dowiadywanie się, kiedy jest czas do domu), słuchacz może mieć szczególne zainteresowanie (tj. konkretną godzinę), a inne zdarzenia mogą być bezużyteczne. Żołnierze, którzy muszą wiedzieć, kiedy nadszedł czas, aby wyjść do koszar, nie są zainteresowani informowaniem o godzinie co klatkę (jest 12:01, nadal jest 12:01, ...). Rejestr można utworzyć, aby zaakceptować opis zainteresowań słuchacza. Może to pozwolić kontrolerowi na ograniczenie tego, czego szuka, a dyspozytorowi może wysyłać tylko odpowiednie zdarzenia, zmniejszając nieefektywność niepotrzebnych kontroli i komunikatów. Format używany do rejestracji zainteresowań może być tak prosty, jak pojedynczy kod zdarzenia. Postacie mogą na przykład rejestrować swoje zainteresowanie wydarzeniami „wybuchowymi". Lepszy stopień kontroli można wspierać dzięki postaciom, które mogą rejestrować bardziej skoncentrowane zainteresowania, takie jak „wybuchy granatów w promieniu 50 metrów od mojej obecnej pozycji". Bardziej rozróżniająca rejestracja pozwala silnikowi sprawdzającemu bardziej skoncentrować się na tym, czego szuka, i zmniejsza liczbę przekazywanych

niepożądanych zdarzeń. Z drugiej strony, decyzja, czy zarejestrowany słuchacz powinien zostać powiadomiony, czy nie, zajmuje więcej czasu, co sprawia, że kod jest bardziej złożony, bardziej specyficzny dla gry (ponieważ rodzaje rzeczy, którymi należy się interesować, często zmieniają się w zależności od gry) i mniej wielokrotnego użytku między grami.

Ogólnie rzecz biorąc, większość programistów korzysta z prostego procesu rejestracji opartego na kodzie zdarzenia, a następnie stosuje pewnego rodzaju podejście narrowcasting w celu ograniczenia niechcianych powiadomień. Dyspozytor zdarzeń wysyła powiadomienie do odpowiednich detektorów w przypadku wystąpienia zdarzenia. Jeśli rejestr zawiera informacje o zainteresowaniach każdego słuchacza, dyspozytor może sprawdzić, czy słuchacz musi wiedzieć o zdarzeniu. Działa to jak filtr, usuwając niechciane zdarzenia i poprawiając wydajność. Najczęstszym sposobem powiadamiania detektora o zdarzeniu jest wywołanie funkcji. W językach obiektowych jest to często metoda klasy. Funkcja jest wywoływana, a informacje o zdarzeniu można przekazać w jej argumentach. W systemach zarządzania zdarzeniami, które obsługują większość systemów operacyjnych, sam obiekt zdarzenia jest często przekazywany do odbiornika. Interfejs nasłuchiwania formularza:

1 class Listener:

2 function notify(event: Event)

jest bardzo powszechny.

WDROŻENIE

Jesteśmy teraz gotowi, aby zebrać wszystkie elementy, aby uzyskać implementację menedżera zdarzeń.

Pseudo kod

1 class EventManager:

2 # Holds data on one registered listener. The same listener may be

3 # registered multiple times.

4 class ListenerRegistration:

5 interestCode: id

6 listener: Listener

7

8 # The list of registered listeners.

9 listeners: Listener[]

10

11 # The queue of pending events.

12 events: Event[]

13

14 # Check for new events, and add them to the queue.

15 function checkForEvents()

```
16
17 # Schedule an event to be dispatched as soon as possible.
18 function scheduleEvent(event: Event):
19 events.push(e)
20
21 # Add a listener to the registry.
22 function registerListener(listener: Listener, code: id):
23 # Create the registration structure.
24 lr = new ListenerRegistration()
25 lr.listener = listener
26 lr.code = code
27
28 # And store it.
29 listeners.push(lr)
30
31 # Dispatch all pending events.
32 function dispatchEvents():
33 # Loop through all pending events.
34 while events:
35 # Get the next event, and pop it from the queue.
36 event = events.pop()
37
38 # Go through each listener.
39 for listener in listeners:
40 # Notify if they are interested.
41 if listener.interestCode == event.code:
42 listener.notify(event)
43
44 # Call this function to run the manager (from a scheduler for
45 # example).
46 function run():
```

47 checkForEvents()

48 dispatchEvents()

Struktury danych i interfejsy

Detektory zdarzeń powinny zaimplementować interfejs EventListener, aby mogły zarejestrować się w menedżerze zdarzeń i otrzymywać prawidłowe powiadomienia. Postacie potrzebują informacji o zaistniałym zdarzeniu. Jeśli postać zgłasza swojej drużynie obserwację wroga, należy podać lokalizację i status wroga. W powyższym kodzie założyłem, że istnieje struktura Event. Podstawowa struktura wydarzenia musi tylko być w stanie się zidentyfikować. Użyliśmy w tym celu członka danych kodu:

1 class Event:

2 code: id

Jest to mechanizm używany w wielu zestawach narzędzi okienkowych do powiadamiania aplikacji o komunikatach myszy, okna i naciśnięcia klawisza. Klasę Event można podzielić na podklasy, aby utworzyć rodzinę różnych typów zdarzeń z ich własnymi dodatkowymi danymi. Jest to mechanizm używany w wielu zestawach narzędzi okienkowych do powiadamiania aplikacji o komunikatach myszy, okna i naciśnięcia klawisza. Klasę Event można podzielić na podklasy, aby utworzyć rodzinę różnych typów zdarzeń z ich własnymi dodatkowymi danymi:

1 class CollisionEvent:

2 code: id = 0x4f2ff1438f4a4c99

3 character1: Character

4 character2: Character

5 collisionTime: int

6

7 class SirenEvent:

8 code: id = 0x9c5d7679802e49ae

9 sirenId: id

W systemie zarządzania zdarzeniami opartym na języku C ten sam efekt można osiągnąć, umieszczając w strukturze danych zdarzenia void*. Można to następnie wykorzystać do przekazania wskaźnika do dowolnej innej struktury danych, jako danych specyficznych dla zdarzenia:

1 typedef class event_t

2 {

3 unsigned long long eventCode;

4 void *data;

5 } Event;

Wydajność

Menedżer zdarzeń jest ustawiony na czas O(nm), gdzie n to liczba zdarzeń w kolejce, amis liczba zarejestrowanych detektorów. Jest to O(n +m) w pamięci. Nie uwzględnia to czasu ani pamięci wymaganej przez odbiornik do obsługi zdarzenia. Zazwyczaj przetwarzanie w słuchaczach zdominuje czas potrzebny do uruchomienia tego algorytmu.

Uwagi dotyczące implementacji

Do tej klasy można wprowadzić szereg udoskonaleń. Najwyraźniej dobrze byłoby umożliwić słuchaczowi otrzymywanie więcej niż jednego kodu zdarzenia. Można to zrobić za pomocą powyższego kodu, rejestrując słuchacza kilka razy z różnymi kodami. Bardziej elastyczna metoda może wykorzystywać kody zdarzeń, które są potęgami dwójki i interpretować zainteresowanie słuchacza jako maskę bitową.

RZUCANIE WYDARZENIA

Istnieją dwie różne filozofie stosowania zarządzania zdarzeniami. Możesz użyć kilku bardzo ogólnych menedżerów wydarzeń, z których każdy wysyła wiele wydarzeń do wielu słuchaczy. Słuchacze są odpowiedzialni za ustalenie, czy są zainteresowani wydarzeniem. Możesz też skorzystać z wielu wyspecjalizowanych menedżerów wydarzeń. Każdy będzie miał tylko kilku słuchaczy, ale ci słuchacze prawdopodobnie będą zainteresowani większą liczbą generowanych przez nie zdarzeń. Detektory nadal mogą ignorować niektóre zdarzenia, ale więcej zostanie dostarczonych poprawnie. Podejście rozproszone nazywa się nadawaniem, a podejście ukierunkowane nazywa się nadawaniem wąskim. Oba podejścia rozwiązują problem ustalenia, którzy agenci wysłać jakie zdarzenia. Transmisja rozwiązuje problem, wysyłając im wszystko i pozwalając im ustalić, czego potrzebują. Narrowcasting nakłada odpowiedzialność na programistę: sztuczna inteligencja musi być zarejestrowana z dokładnie odpowiednim zestawem odpowiednich menedżerów wydarzeń.

Nadawanie

Przyjrzeliśmy się dodawaniu dodatkowych danych do rejestru, aby zachowania mogły pokazywać ich zainteresowania. Nie jest to prosty proces do uogólnienia. Trudno jest zaprojektować system rejestracji, który ma wystarczająco dużo szczegółów, aby można było zidentyfikować słuchaczy o bardzo specyficznych potrzebach. Na przykład sztuczna inteligencja może potrzebować wiedzieć, kiedy uderzy w ściany wykonane z jednego z zestawu materiałów do skakania. Aby to wspierać, rejestr musiałby przechowywać wszystkie możliwe materiały dla wszystkich obiektów w świecie gry, a następnie sprawdzać poprawną listę materiałów dla każdego uderzenia. Byłoby łatwiej, gdyby sztuczna inteligencja została poinformowana o wszystkich kolizjach i mogłaby odfiltrować te, którymi nie była zainteresowana. Takie podejście nazywa się rozgłaszaniem. Menedżer zdarzeń transmisji wysyła do swoich słuchaczy wiele zdarzeń. Zazwyczaj służy do zarządzania wszelkiego rodzaju wydarzeniami, dlatego też ma wielu słuchaczy.

Nadawane są programy telewizyjne. Są przesyłane kablowo lub drogą radiową, niezależnie od tego, czy ktoś jest zainteresowany ich oglądaniem. Twój salon jest cały czas bombardowany tymi wszystkimi danymi. Możesz wyłączyć telewizor i zignorować go lub obejrzeć program, który chcesz oglądać. Nawet jeśli oglądasz telewizję, zdecydowana większość informacji docierających do Twojego telewizora nie jest wyświetlana. Nadawanie jest procesem marnotrawnym, ponieważ przesyłanych jest wiele danych, które są bezużyteczne dla odbiorców. Zaletą jest elastyczność. Jeśli postać otrzymuje i wyrzuca dużo danych, może nagle zainteresować się i wiedzieć, że właściwe dane są dostępne natychmiast. Jest to szczególnie ważne, gdy sztuczna inteligencja postaci jest uruchamiana przez skrypt, w którym oryginalni programiści nie są świadomi, jakich informacji twórca skryptu może chcieć użyć. Wyobraź sobie, że mamy postać z gry, która wędruje po grządce z grzybami i zbiera grzyby. Zależy nam na tym,

aby postać wiedziała, czy gracz ukradnie jeden z jej grzybów. Nie interesuje nas wiedza, czy drzwi na tym poziomie zostały otwarte. Postać jest tak rozwinięta, że ignoruje wszystkie zdarzenia związane z otwieraniem drzwi, ale reaguje na zdarzenia związane ze skradzionym grzybem. Później w procesie rozwoju projektant poziomów dodaje dom grzybiarza do poziomu i chce edytować skrypt AI, aby reagował, gdy gracz wejdzie do domu. Jeśli event manager transmituje wydarzenia, nie będzie to trudne. Skrypt mógł reagować na zdarzenia związane z otwieraniem drzwi. Gdyby menedżer zdarzeń zastosował podejście narrowcast, projektant poziomów musiałby zatrudnić programistę do zarejestrowania postaci w słuchaczu otwartych drzwi. Oczywiście można to obejść. Na przykład możesz uczynić proces rejestracji częścią skryptu (chociaż możesz oczekiwać, że projektanci poziomów zbyt wiele będą manipulować kanałami zdarzeń). Ale elastyczność zawsze będzie wyższa przy podejściu do transmisji.

Zawężanie

Narrowcasting rozwiązuje problem z wiedzą, która sztuczna inteligencja jest zainteresowana jakimi wydarzeniami, wymagając od programisty wielu rejestracji do wyspecjalizowanych menedżerów wydarzeń. Jeśli zespoły jednostek w grze RTS muszą dzielić się informacjami, każda z nich może mieć własnego menedżera wydarzeń. Z jednym menedżerem wydarzeń na grupę, wszelkie wydarzenia będą kierowane tylko do właściwych osób. Jeśli na mapie są setki zespołów, muszą być setki event managerów. Ponadto zespoły te mogą być zorganizowane w większe grupy. Te większe grupy mają swoich własnych menadżerów wydarzeń, którzy dzielą się informacjami na temat batalionu. Ostatecznie po każdej stronie istnieje jeden menedżer zdarzeń, który służy do globalnego udostępniania informacji.

Narrowcasting to bardzo efektywne podejście. Niewiele jest zmarnowanych wydarzeń, a informacje kierowane są dokładnie do właściwych osób. Nie musi istnieć żaden zapis zainteresowań słuchacza. Każdy event manager jest tak wyspecjalizowany, że wszyscy słuchacze będą prawdopodobnie zainteresowani wszystkimi wydarzeniami. To ponownie poprawia prędkość. Podczas gdy prędkość w grze jest zoptymalizowana przy użyciu podejścia wąskiego, konfigurowanie postaci jest znacznie bardziej złożone. Jeśli istnieją setki menedżerów zdarzeń, musi istnieć znaczna ilość kodu konfiguracyjnego, który określa, które detektory należy podłączyć do których menedżerów zdarzeń. Sytuacja staje się jeszcze bardziej złożona, jeśli postacie zmieniają się w czasie. W przykładzie RTS większość drużyny może zginąć w bitwie. Pozostałych członków należy umieścić w nowym zespole. Oznacza to dynamiczną zmianę rejestracji. W przypadku prostej hierarchii menedżerów wydarzeń jest to nadal możliwe. W przypadku bardziej złożonych „zup" menedżerów wydarzeń, z których każda kontroluje inny zestaw niepowiązanych ze sobą wydarzeń, może to wymagać więcej wysiłku niż jest to warte.

Kompromis

W rzeczywistości istnieje kompromis, który należy osiągnąć między organizatorami wydarzeń, którzy mają złożone informacje rejestracyjne, a tymi, którzy w ogóle nie mają wyraźnych interesów. Podobnie istnieje powiązany kompromis między nadawaniem wąskim a nadawaniem. W rzeczywistości programiści mają tendencję do używania prostych informacji o zainteresowaniach, które można bardzo szybko przefiltrować. W przykładowej implementacji użyliśmy kodu zdarzenia. Jeśli kod zdarzenia odpowiada zainteresowaniu słuchacza, słuchacz zostanie o tym powiadomiony. Kod zdarzenia może być używany do reprezentowania dowolnego rodzaju informacji o zainteresowaniach, a menedżer zdarzeń nie musi wiedzieć, co oznacza kod w grze. Umożliwia to wykorzystanie tej samej implementacji event managera w dowolnej liczbie sytuacji. Kompromis między rozgłaszaniem a nadawaniem wąskim zależy w większym stopniu od aplikacji, w szczególności od liczby zdarzeń, które

mogą zostać wygenerowane. Często nie ma wystarczającej liczby zdarzeń AI, aby nadawanie było zauważalnie spowolnione. Zalecam korzystanie z metody transmisji, gdy gra jest w fazie rozwoju. Dzięki temu możesz łatwiej bawić się zachowaniami postaci. Jeśli okaże się, że system zdarzeń jest powolny w miarę postępu prac rozwojowych, można go zoptymalizować za pomocą wielu menedżerów narrowcastingu przed wydaniem. Wyjątkiem od tej reguły są menedżerowie wydarzeń z bardzo specyficznymi funkcjami. Menedżer wydarzeń, który powiadamia postacie w określonym czasie gry (na przykład, aby powiedzieć żołnierzom, kiedy mają odczekać), byłby trudny do włączenia do menedżera transmisji wraz z innymi rodzajami wydarzeń.

KOMUNIKACJA MIĘDZY AGENCJAMI

Podczas gdy większość informacji, których potrzebuje sztuczna inteligencja, pochodzi z działań gracza i środowiska gry, w grach coraz częściej pojawiają się postacie, które współpracują lub komunikują się ze sobą. Na przykład oddział strażników powinien współpracować, aby otoczyć intruza. Gdy lokalizacja intruza jest znana, strażnicy mogą osłaniać wszystkie wyjścia, czekając, aż ich koledzy z drużyny znajdą się na swoich pozycjach, zanim przystąpią do ataku. Algorytmy koordynujące tego rodzaju działania zostały omówione w Części 6. Jednak niezależnie od zastosowanych technik postacie muszą zrozumieć, co robią inni i co zamierzają zrobić. Można to osiągnąć, pozwalając każdej postaci na zbadanie wewnętrznego stanu innych postaci lub przez odpytywanie ich o ich intencje. Chociaż jest to szybkie, jest podatne na błędy i może wymagać wielu przepisywania przy każdej zmianie AI postaci. Lepszym rozwiązaniem jest użycie mechanizmu zdarzeń, aby umożliwić każdej postaci poinformowanie innych o swoim zamiarze. Możesz myśleć o tym menedżerze wydarzeń jako zapewniającym bezpieczne łącze radiowe między członkami zespołu AI. Podstawowy mechanizm zdarzeń w tym Części jest wystarczający do obsługi wspólnego przekazywania komunikatów. Korzystanie z menedżera zdarzeń wąskich transmisji dla każdego oddziału zapewnia, że dane szybko docierają do właściwych postaci i nie mylą członków innego oddziału.

OBIEKTY WYBORCZE

Są sytuacje, w których sondowanie jest oczywiście bardziej efektywne niż wydarzenia. Postać, która musi otworzyć drzwi, podchodzi do nich i sprawdza, czy są zamknięte. Nie ma sensu, aby drzwi wysyłały komunikaty „Jestem zamknięty” w każdej klatce. Czasami jednak kontrole są czasochłonne. Jest to szczególnie ważne, gdy kontrola obejmuje geometrię poziomu gry. Strażnik patrolujący może od czasu do czasu sprawdzać stan panelu kontrolnego od wejścia do pokoju kontrolnego. Jeśli gracz przesunie pudełko przed panel, linia wzroku zostanie zablokowana. Obliczanie linii wzroku jest drogie. Jeśli jest więcej niż jeden strażnik, dodatkowa kalkulacja jest marnowana. W systemie opartym na zdarzeniach kontrolę można przeprowadzić raz na zawsze. W systemie odpytywania każdy znak sprawdza z osobna. Na szczęście jest kompromis. Kiedy głosowanie jest najlepszym podejściem, ale kontrole są czasochłonne, możemy użyć struktury zwanej lokalem wyborczym. Lokal wyborczy ma dwa cele. Po pierwsze, jest to po prostu pamięć podręczna informacji odpytywania, z której może korzystać wiele znaków. Po drugie, działa jako pośrednik między sztuczną inteligencją a poziomem gry. Ponieważ wszystkie żądania przechodzą przez to jedno miejsce, można je łatwiej monitorować, a sztuczna inteligencja debugować. Można użyć kilku mechanizmów buforowania, aby upewnić się, że dane nie są przeliczane zbyt często. Przykład pseudokodu używa licznika liczby klatek do oznaczania danych jako przestarzałych. W razie potrzeby dane są przeliczane raz na każdą ramkę. Jeśli dane nie są wymagane w ramce, nie zostaną ponownie przeliczone.

PSEUDO-KOD

Konkretny lokal wyborczy możemy zrealizować w następujący sposób:

```
1 class PollingStation:
2 # The cache for a boolean property of the game.
3 class BoolCache:
4 value: bool
5 lastUpdated: int
6
7 # The cache value for one topic.
8 isFlagSafe: BoolCache[MAX_TEAMS]
9
10 # Updates the cache, when required.
11 function updateIsFlagSafe(team: int)
12 isFlagSafe[team].value = # ... query game state ...
13 isFlagSafe[team].lastUpdated = getFrameNumber()
14
15 # Query the cached topic.
16 function getIsFlagAtBase(team: int) -> bool:
17 # Check if the topic needs updating.
18 if isFlagSafe[team].lastUpdated < getFrameNumber():
19 # Only update if the cache is stale.
20 updateIsFlagSafe(team)
21
22 # Either way, return its value.
23 return isFlagSafe[team].value
24
25 # ... add other polling topics ...
26
27 # A polling topic without a cache.
28 function canSee(fromPos: Vector, toPos: Vector) -> bool:
29 # ... always query game state ...
30 return not raycast(from, to)
```

11.3.2 WYDAJNOŚĆ

Lokal wyborczy jest O(1) zarówno pod względem czasu, jak i pamięci dla każdego obsługiwanego tematu odpytywania. Wyklucza to wykonanie samego odpytywania.

UWAGI DO WDRAŻANIA

Powyższa implementacja dotyczy konkretnego lokalu wyborczego, a nie ogólnego systemu. Pokazuje dwa różne tematy odpytywania: getIsFlagAtBase i canSee. Pierwszy z nich pokazuje wzór wyniku z pamięci podręcznej, a drugi jest obliczany za każdym razem, gdy jest potrzebny. Buforowanie części kodu opiera się na istnieniu funkcji getFrameNumber do śledzenia nieaktualnych elementów. W pełnej implementacji byłoby kilka dodatkowych klas pamięci podręcznej podobnych do BoolCache dla różnych zestawów typów danych. Często lokal wyborczy również upraszcza sztuczną inteligencję. W powyższym kodzie znak wystarczy, aby wywołać funkcję canSee lokalu wyborczego. Nie musi przeprowadzać samego sprawdzenia. W takim przypadku funkcja zawsze przelicza kontrolę wzroku; jego wartość nie jest buforowana. Sztuczna inteligencja nie dba o to, czy wynik poprzedniego połączenia jest przechowywany, czy też musi zostać przeliczony. Nie obchodzi go również, jak pobierany jest wynik. Pozwala to programistom na późniejszą zmianę i optymalizację implementacji bez przepisywania dużej ilości kodu.

ODBIERANIE STRESZCZENIA

Powyższa lista to najprostsza forma lokalu wyborczego. Często tego rodzaju metody można dodać do światowej klasy gry jako standardowy interfejs. Mają tę wadę, że są trudne do wydłużenia. W końcu lokal wyborczy będzie bardzo duży i będzie zawierał dużo danych. Lokal wyborczy można ulepszyć, dodając centralną metodę żądania, do której kierowane są wszystkie ankiety. Ta metoda żądania pobiera kod żądania, który sygnalizuje, które sprawdzenie jest potrzebne. Ten abstrakcyjny model odpytywania umożliwia rozbudowę lokalu wyborczego bez zmiany jego interfejsu i bez zmiany innego kodu, który się na nim opiera. Pomaga także narzędziom debugowania i rejestrowania, ponieważ wszystkie żądania odpytywania są kierowane za pomocą metody centralnej. Z drugiej strony istnieje dodatkowy krok tłumaczenia, aby ustalić, które żądanie jest dostarczane, co spowalnia wykonanie. Ta implementacja lokalu wyborczego rozszerza ideę o krok dalej, umożliwiając „wtyczkowe” odpytywanie. Instancje zadania odpytywania mogą być rejestrowane w stacji, z których każda reprezentuje jedną możliwą część danych, które można odpytywać. Logika kontroli pamięci podręcznej jest taka sama dla wszystkich tematów (takie samo buforowanie na podstawie liczby ramek, jak poprzednio).

```
1 # Abstract class; the base for any pollable topic.

2 class PollingTask:

3 taskCode: id

4 value: any

5 lastUpdated: int

6

7 # Checks if the cache is out of date.

8 function isStale() -> bool:

9 return lastUpdated < getFrameNumber()

10
```

```
11 # Update the value in the cache - implement in subclasses.
12 function update()
13
14 # Get the correct value for the polling task.
15 function getValue() -> any:
16 # Update the internal value, if required.
17 if isStale():
18 update()
19
20 # Return it.
21 return value
22
23 class AbstractPollingStation:
24 # The tasks registered as a hash-table indexed by code.
25 tasks: HashTable[id -> PollingTask]
26
27 function registerTask(task: PollingTask):
28 tasks[task.code] = task
29
30 function poll(code: id) -> value:
31 return tasks[code].getValue()
```

W tym momencie jesteśmy prawie w złożoności systemu zarządzania wydarzeniami, a kompromis między nimi staje się zamazany. W praktyce niewielu deweloperów polega na lokalach wyborczych tej złożoności.

ZARZĄDZANIE ZMYSŁAMI

Do tej pory omówiliśmy techniki przekazywania odpowiedniej wiedzy w ręce postaci, które mogą być zainteresowane. Naszą troską było upewnienie się, że postać otrzymuje informacje, których chce, aby móc podejmować odpowiednie decyzje. Ale, jak wszyscy wiemy, pragnienie czegoś to nie to samo, co zdobycie tego! Musimy również zadbać o to, aby postać była w stanie zdobyć interesującą ją wiedzę. Środowiska gry przynajmniej do pewnego stopnia symulują świat fizyczny. Postać uzyskuje informacje o swoim otoczeniu za pomocą zmysłów. Dlatego warto sprawdzić, czy postać może fizycznie wyczuć informacje. Jeśli w grze generowany jest głośny dźwięk, możemy określić, które postacie go słyszały: postać po drugiej stronie poziomu może tego nie robić, podobnie jak postać za dźwiękoszczelnym oknem. Wróg może przechodzić przez środek pokoju, ale jeśli światła są wyłączone lub postać zwrócona jest w złym kierunku, wróg nie będzie widoczny. Symulacja percepcji sensorycznej jest

stosunkowo nowa, a wyrafinowane podejścia wciąż nie są powszechne (co najwyżej przeprowadza się kontrolę rzucania promieni, aby określić, czy istnieje linia wzroku). Jednak w niektórych grach percepcja stanowi kluczową część rozgrywki. Jest to najczęstsze w przypadku percepcji wzrokowej. W grach takich jak Splinter Cell , Thief: The Dark Project i Metal Gear Solid zdolność postaci AI do widzenia tylko oświetlonych obiektów stanowi podstawę rozgrywki z ukrycia. Wszystko wskazuje na to, że ten trend się utrzyma. Oprogramowanie sztucznej inteligencji wykorzystywane w przemyśle filmowym (takie jak Weta's Massive) i symulacje wojskowe wykorzystują bardziej wszechstronne modele percepcji do kierowania bardzo wyrafinowanymi zachowaniami grupowymi.2 Wydaje się prawdopodobne, że percepcja zmysłowa przeniknie inne projekty gier, aby stać się integralną częścią gier strategicznych, gry fabularne i platformówki, a także gry akcji.

UDAWANIE

Oczywiście staramy się iść na skróty, kiedy tylko jest to możliwe. Nie ma sensu symulować sposobu, w jaki dźwięk przemieszcza się ze słuchawek na głowie postaci do kanału słuchowego. Trochę wiedzy, którą możemy po prostu przekazać postaci. Nawet w przypadku wątpliwości, czy wiedza przeniknie, możemy skorzystać z metod omówionych wcześniej w tym Części: na przykład użyć event managera na pokój. Dźwięki, które pojawiają się w pokoju, mogą być przekazywane wszystkim postaciom znajdującym się w tym pokoju i rejestrowane w menedżerze wydarzeń. Tutaj menedżer wydarzeń jest używany w nieco inny sposób niż opisany wcześniej. Zamiast korzystać z jego mocy dystrybucyjnej, polegamy na tym, że informacje nie przekazane event managerowi nie mogą zostać pozyskane przez jego słuchaczy. Niekoniecznie tak jest, jeśli znaki odpytują o dane (chociaż możemy dodać kod filtrujący, aby ograniczyć dostęp do lokalu wyborczego, aby uzyskać ten sam efekt). Aby menedżer wydarzeń działał z powiadomieniami dźwiękowymi, musimy upewnić się, że postacie zmieniają menedżerów wydarzeń lub dostosowują wydarzenia, których chcą słuchać, za każdym razem, gdy poruszają się między pokojami. Może to działać w określonych sytuacjach, takich jak określony styl poziomu gry lub bardzo prosty projekt gry. Nie jest to jednak realistyczny model, ponieważ głośne dźwięki mogą być słyszalne w korytarzu, ale delikatne dźwięki mogą być niesłyszalne z odległości metra. A co robimy z innymi zmysłami? Wizja jest często implementowana za pomocą rzutowania promieni, aby sprawdzić linię wzroku, ale może to szybko wymknąć się spod kontroli, jeśli wiele postaci próbuje zobaczyć wiele różnych rzeczy. Jeśli martwimy się o dokładność, w końcu będziemy potrzebować dedykowanego kodu symulacji zmysłów.

CO WIEMY?

Postać ma dostęp do różnych źródeł wiedzy w grze. Na początku Części 5 przyjrzeliśmy się pokrótce wiedzy, dzieląc ją na dwie kategorie: wiedza wewnętrzna i wiedza zewnętrzna. Wewnętrzna wiedza postaci mówi o niej samej: jej aktualnym stanie zdrowia, wyposażeniu, stanie umysłu, celach i ruchu. Wiedza zewnętrzna obejmuje wszystko inne w środowisku postaci: pozycję wrogów, otwarte drzwi, dostępność ulepszeń lub liczbę pozostałych przy życiu członków drużyny. Wiedza wewnętrzna jest zasadniczo darmowa, a postać powinna mieć do niej bezpośredni i nieskrępowany dostęp. Wiedza zewnętrzna jest dostarczana postaci na podstawie stanu gry. Wiele gier pozwala postaciom być wszechwiedzącymi; na przykład zawsze wiedzą, gdzie jest gracz. Aby zasymulować pewien stopień tajemniczości, zachowanie postaci można zaprojektować tak, aby wydawało się, że nie ma wiedzy. Postać może na przykład stale patrzeć na pozycję gracza. Gdy gracz zbliży się dostatecznie blisko, postać nagle rozpoczyna akcję pościgu. Graczowi wydaje się, że postać nie widziała gracza, dopóki nie zbliżył się wystarczająco. Jest to cecha projektowania AI, a nie sposób, w jaki postać zdobywa wiedzę. Bardziej wyrafinowane podejście wykorzystuje menedżerów wydarzeń lub lokali wyborczych, aby przyznać dostęp tylko do informacji, które może znać prawdziwa osoba w środowisku gry. Na ostatnim krańcu znajdują się menedżerowie zmysłów rozprowadzający informacje w oparciu o fizyczną symulację

świata. Nawet w grze z wyrafinowanym zarządzaniem zmysłami sensowne jest stosowanie podejścia mieszanego. Wiedza wewnętrzna jest zawsze dostępna, ale do wiedzy zewnętrznej można uzyskać dostęp na trzy sposoby: bezpośredni dostęp do informacji, powiadamianie tylko o wybranych informacjach i symulacja percepcji. Pozostała część tej sekcji skupi się tylko na ostatnim elemencie: zarządzaniu zmysłami. Pozostałe elementy zostały omówione do tej pory w tej Części.

Ponowne odpytywanie i powiadamianie

Chociaż teoretycznie możliwe jest wdrożenie systemu zarządzania sensorycznego opartego na ankietach, nigdy nie widziałem tego w praktyce. Moglibyśmy na przykład testować proces sensoryczny za każdym razem, gdy stan odpytywania otrzyma żądanie informacji, przekazując dane tylko wtedy, gdy test zakończy się pomyślnie. Nie ma nic złego w tym podejściu, ale to nie jest to, które proponuję. Percepcja sensoryczna wydaje się bardziej naturalnie mapować na powiadomienia: postać odkrywa informacje, postrzegając je, zamiast szukać wszystkiego i nie dostrzegać większości z nich. Uruchamianie zarządzania sensami w strukturze sondowania oznacza, że ogromna większość żądań sondowania kończy się niepowodzeniem — co jest dużym marnotrawstwem wydajności. Odtąd będę używał wyłącznie modelu opartego na zdarzeniach dla naszych narzędzi do zarządzania zmysłami. Do menedżera zmysłów wprowadzana jest wiedza ze stanu gry, a postacie, które są w stanie ją dostrzec, zostaną o tym powiadomione. Mogą wtedy podjąć odpowiednie działania, takie jak przechowywanie ich do późniejszego wykorzystania lub natychmiastowe działanie.

MODALNOŚCI SENSORYCZNE

W grze można używać czterech naturalnych zmysłów człowieka: wzroku, dotyku, słuchu i węchu, w mniej więcej malejącej kolejności użycia. Smak to piąty ludzki zmysł, ale jeszcze nie widziałem (a nawet nie wymyśliłem) gry, w której postacie wykorzystują smak, aby zdobywać wiedzę o swoim świecie. Rozważmy kolejno każdą modalność sensoryczną. Ich osobliwości stanowią podstawowe wymagania kierownika zmysłu.

Widok

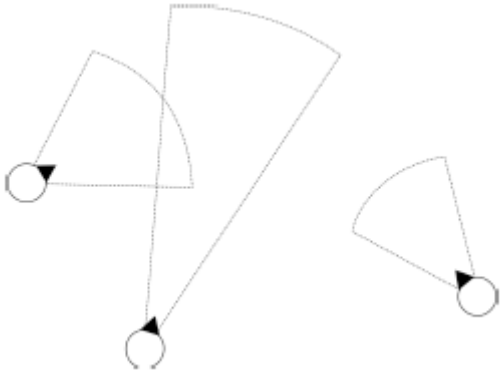
Wzrok to najbardziej oczywisty zmysł. Ponieważ jest to tak oczywiste, gracze mogą stwierdzić, czy jest źle symulowany. To z kolei oznacza, że będziemy musieli ciężej pracować, aby opracować przekonujący model wzroku. Spośród wszystkich metod, które będziemy wspierać, wzrok wymaga największej infrastruktury. Cały szereg czynników może wpływać na naszą zdolność widzenia czegoś.

Prędkość

Światło przemieszcza się z prędkością prawie 300 milionów ms-1. O ile twoja gra nie obejmuje bardzo dużych odległości w przestrzeni, światło będzie podróżować przez twój poziom gry w mniej niż klatkę. Możemy traktować wizję jako natychmiastową.

Stożek wzroku

Po pierwsze, mamy stożek celowniczy. Nasza wizja ogranicza się do kształtu stożka przed nami, jak pokazano na rysunku.



Jeśli głowa jest nieruchoma, ma stożek wzroku o kącie pionowym około 120◦ i kącie poziomym około 220◦. Jesteśmy w stanie widzieć w dowolnym kierunku 360◦, poruszając szyją i oczami, jednocześnie utrzymując resztę ciała w bezruchu. Jest to możliwy stożek widzenia dla postaci, która szuka informacji. Ludzie zajmujący się normalnym biznesem koncentrują się na bardzo małej części swojego pola widzenia. Świadomie jesteśmy w stanie monitorować stożek o zaledwie kilku stopniach, ale ruchy gałek ocznych przesuwają ten stożek szybko, dając złudzenie szerszego pola widzenia. Badania psychologiczne wskazują, że ludzie bardzo słabo zauważają rzeczy, których specjalnie nie szukają. W rzeczywistości jesteśmy gorsi w zauważaniu rzeczy, których szukamy, niż możesz sobie wyobrazić. Jeden z eksperymentów obejmował nagranie wideo z treningu koszykówki, przez który przechodził ktoś w puszystym kostiumie zwierzęcym. Poproszony o policzenie liczby podań wykonanych przez koszykarzy, większość widzów nie zauważyła osoby w stroju stojącej na środku kortu, machając rękami. Aby uzyskać posmak tych ograniczeń, często używa się stożka wzroku około 60◦. Uwzględnia normalny ruch gałek ocznych, ale skutecznie oślepia postać na dużą powierzchnię, którą widzi, ale raczej nie zwraca na nią uwagi.

Linia wzroku

Najbardziej charakterystyczną cechą Vision jest nieumiejętność pokonywania zakrętów. Aby coś zobaczyć, musisz mieć to w linii wzroku. Chociaż jest to oczywiste, nie jest to do końca prawda. Jeśli postać stanie na jednym końcu zaciekłego, ciemnego korytarza, nie będzie w stanie dostrzec wroga z drugiego końca. Ale gdy tylko wróg wystrzeli z karabinu, postać zobaczy odbity błysk lufy. Zdarzenia, które emitują światło, zachowują się inaczej niż te, które go nie emitują, jeśli chodzi o symulację. Wszystkie powierzchnie w pewnym stopniu odbijają światło, co pozwala mu dość łatwo odbijać się od narożników. Jeden system zarządzania zmysłami, z którym pracowałem, miał tę funkcję. Niestety efekt był tak subtelny, że nie warto było go symulować (okazało się, że wydawca uznał, że cała gra nie jest warta zachodu i została skasowana przed publikacją). Pomimo swoich wad, symulacja zmysłów w tej grze przewyższała wszystko, co widziałem, i będę odnosić się do kilku jej funkcji w dalszej części tej sekcji. Na potrzeby tego Częśću przyjmę, że widzenie dzieje się tylko w liniach prostych. Aby zasymulować radiosity lub efekty takie jak lustra, będziesz musiał rozszerzyć ramy.

Dystans

W skalach modelowanych na średnich poziomach gry, ludzie nie mają ograniczenia odległości wzroku. Efekty atmosferyczne (na przykład mgła lub zamglenie) i krzywizna Ziemi ograniczają naszą zdolność widzenia na bardzo duże odległości, ale ludzie nie mają problemu z widzeniem przez miliony lat świetlnych, jeśli nic nie stoi na przeszkodzie. Istnieje jednak niezliczona ilość gier, w których odległość jest wykorzystywana jako ograniczenie widzenia. To nie zawsze jest to zła rzecz. Na przykład w grze platformowej Jak and Daxter: The Precursor Legacy nie chcielibyśmy zakradać się za każdy róg, aby znaleźć wrogów z drugiej strony polany, który nas widzi i nadchodzi. W grach często stosuje się

konwencję, że wrogowie zauważają gracza tylko wtedy, gdy ten znajdzie się na pewną odległość. Celowo daje postaciom gorszy wzrok, niż mieliby w innym przypadku. Gry, które nie stosują się do tego ograniczenia, takie jak Tom Clancy's Ghost Recon , wymagają innej taktyki gry, zwykle obejmującej znacznie więcej skradania się.

Tam, gdzie odległość ma znaczenie, jest rozmiar oglądanej rzeczy. Wszystkie zwierzęta mogą rozpoznawać obiekty tylko wtedy, gdy wydają się wystarczająco duże (ignorując na chwilę jasność i wzory tła). W skali ludzkiej dla większości poziomów gry nie stanowi to problemu. Możemy na przykład rozróżnić człowieka oddalonego o ponad pół mili. Podobnie jak w przypadku stożków wzroku, istnieje różnica między zdolnością a prawdopodobieństwem. Chociaż możemy rozróżnić istoty ludzkie w odległości setek metrów, jest mało prawdopodobne, że zauważymy osobę z tej odległości, chyba że mamy specjalne zadanie ich szukania. Nawet w grach, które nie ograniczają odległości widzianej przez postać, zalecany jest pewien próg odległości do zauważenia małych obiektów.

Jasność

Polegamy na fotonach docierających do naszego oka, aby widzieć rzeczy. Światłoczułe komórki w oku stają się podekscytowane, gdy uderza w nie foton, i stopniowo rozluźniają się w ciągu następnych kilku milisekund. Jeśli wystarczająca ilość fotonów dotrze do komórki, zanim się rozluźnia, będzie ona coraz bardziej podekscytowana i ostatecznie wyśle swój sygnał do mózgu. Uważamy, że bardzo trudno jest nam widzieć w słabym świetle. Splinter Cell wykorzystuje tę cechę ludzkiego wzroku z dobrym skutkiem, pozwalając graczowi ukrywać się w cieniu i unikać wykrycia przez strażników (mimo że postać gracza ma trzy jasnozielone pochodnie przypięte do czoła). W rzeczywistości rzadko przebywamy w wystarczająco ciemnych warunkach, aby wrażliwość naszych oczu na światło była czynnikiem ograniczającym nasze widzenie. Zdecydowana większość naszego problemu z widzeniem w słabym świetle to nie brak fotonów, to problem różnicowania.

Różnicowanie

Ludzki wzrok ewoluował w oparciu o nasze potrzeby przetrwania. To, co widzimy okiem umysłu jako obraz świata zewnętrznego, jest w rzeczywistości iluzją zrekonstruowaną z wielu różnych sygnałów i wielu oczekiwań. Wszystkie bodźce wzrokowe są filtrowane i przetwarzane. Kiedy na przykład przechylasz głowę, obraz, który widzisz, nie przechyla się; mamy wyspecjalizowane komórki w naszym systemie wizualnym, które są przeznaczone do wyszukiwania pionu. Wszystkie wyniki z reszty naszego układu wzrokowego są następnie wewnętrznie obracane z powrotem, zanim ich wynik trafi do naszego świadomego mózgu. Większość z nas fizycznie nie widzi w przechyleniu (jest to jeden z powodów, dla których większość gier samochodowych nie przechyla kamery tak, jak na zakrętach samochodu, a te, które rzadko przechylają się do głów większości kierowców, a nie często w tym samym kierunku). Najważniejszą adaptacją do zarządzania zmysłami są nasze detektory kontrastu. Posiadamy całą gamę komórek dedykowanych do identyfikacji obszarów, w których zmieniają się kolory lub odcienie. Niektóre z tych komórek są przeznaczone do znajdowania wyraźnych linii pod różnymi kątami, a inne do znajdowania plam, w których Wydajność uje zmiana kontrastu. Ogólnie rzecz biorąc, trudno jest nam coś zobaczyć bez wystarczającej zmiany kontrastu. Zmiana kontrastu może wystąpić tylko w jednym składniku koloru. To jest podstawa tych testów na daltonizm; jeśli nie możesz wykryć różnicy między czerwienią a zielenią, nie możesz wykryć jednoczesnej, ale przeciwnej zmiany intensywności czerwieni i zieleni, a zatem nie możesz zobaczyć liczby. Oznacza to, że nie możemy zobaczyć obiektów, które nie kontrastują z ich tłem, i jesteśmy bardzo dobrzy w widzeniu obiektów, które kontrastują. Wszystkie kamuflaże działają na tej zasadzie; stara się upewnić, że nie ma zmiany kontrastu między krawędzią czegoś a jego tłem. Powodem, dla którego nie możemy zobaczyć rzeczy w słabym świetle, jest to, że nie ma wystarczającego kontrastu, aby to zobaczyć, a nie dlatego, że fotony nie docierają do

naszych oczu. Gry Ghost Recon mają dobrą implementację kamuflażu tła. Jeśli twoja drużyna znajduje się na wojskowej zieleni, leżąc w gąszczu listowia, wrogie postacie ich nie zobaczą. W tym samym mundurze, stojąc przed ceglaną ścianą, siedzą kaczki. Z drugiej strony Splinter Cell, słusznie chwalony za chowanie się w cieniu, nie bierze pod uwagę tła. Sam Fisher (postać, którą grasz) może stać w cieniu w połowie bardzo jasno oświetlonego korytarza, a wróg na jednym końcu korytarza go nie zobaczy. W rzeczywistości, oczywiście, wróg zobaczyłby ogromną czarną sylwetkę na jasnym tle, a Sam by się poruszył. (Szczerze mówiąc, projektanci poziomów ciężko pracują, aby ta sytuacja nie Wydajność ewoluowała zbyt często.)

Nasłuchiwanie

Słuch nie jest ograniczony liniami prostymi. Dźwięk rozchodzi się w postaci fal kompresji przez dowolne medium fizyczne. Fala potrzebuje czasu, aby się poruszyć, a gdy się porusza, rozprzestrzenia się i podlega tarciu. Oba czynniki służą do zmniejszania intensywności (głośności) dźwięku wraz z odległością. Dźwięki o niskim tonie ulegają mniejszemu tarciu (ponieważ ich wibracje są wolniejsze) i dlatego przemieszczają się dalej niż dźwięki o wysokim tonie. Dźwięki o niskim tonie są również w stanie łatwiej omijać przeszkody. Dlatego dźwięk dobiegający zza przeszkody brzmi mętnie i nisko. Słonie emitują infradźwiękowe szczekanie poniżej poziomu ludzkiego słuchu, aby komunikować się z innymi członkami swojego stada oddalonego o kilka mil przez zarośla. Natomiast nietoperze używają wysokich dźwięków do postrzegania ciem; dźwięki o niskiej częstotliwości po prostu zginałyby się wokół ofiary. Te różnice są prawdopodobnie zbyt subtelne, aby włączyć je do sztucznej inteligencji w grze. Wszystkie dźwięki potraktujemy jednakowo: równomiernie zmniejszają głośność wraz z odległością, aż przekroczą pewien próg. Pozwalamy różnym postaciom na wyczuwanie różnych głośności dźwięków, aby symulować ostry słuch lub głuchotę wynikającą na przykład z pobliskiego wybuchu bomby. Jeśli chodzi o sztuczną inteligencję, dźwięk bez problemu przemieszcza się w powietrzu za rogiem, niezależnie od jego wysokości. Technologie dźwięku środowiskowego, używane do przygotowania trójwymiarowego (3D) dźwięku dla odtwarzacza, mają bardziej wszechstronne możliwości symulowania okluzji. Kiedy gracz słucha, efekty są znaczące. Jednak przy ustalaniu, czy postać coś wie, efekty nie są znaczące. W rzeczywistości wszystkie materiały w pewnym stopniu przenoszą dźwięk. Gęstsze i sztywniejsze materiały przenoszą dźwięk szybciej. Na przykład stal przenosi dźwięk szybciej niż woda, a woda szybciej niż powietrze. Z tego samego powodu powietrze o wyższej temperaturze przenosi dźwięk szybciej. Prędkość dźwięku w powietrzu wynosi około 345 metrów na sekundę.

Jednak w implementacji gier zazwyczaj wszystkie materiały dzielimy na dwie kategorie: te, które nie przekazują dźwięku i te, które to przekazują. Wszystkie materiały przenoszące dźwięk są traktowane jak powietrze. Ponieważ poziomy gry są zwykle dość małe, prędkość dźwięku jest często wystarczająco duża, aby nie zostać zauważonym. Wiele gier symuluje dźwięk, pozwalając mu zachowywać się jak światło, błyskawicznie podróżując. Na przykład Metal Gear Solid nie ma dostrzegalnej prędkości dźwięku, podczas gdy Conflict: Desert Storm [159] tak. Jeśli zamierzasz wykorzystać prędkość dźwięku, warto ją spowolnić. W typowej grze z perspektywy trzeciej lub pierwszej osoby prędkość dźwięku około 100 metrów na sekundę daje „realistyczny” i zauważalny efekt.

Dotyk

Dotyk to zmysł, który wymaga bezpośredniego kontaktu fizycznego. Najlepiej zaimplementować to w grze wykorzystującej wykrywanie kolizji: postać jest powiadamiana, jeśli zderzy się z inną postacią. W grach z ukrycia jest to część gry. Jeśli dotkniesz postaci (lub zbliżysz się do niej w niewielkiej, ustalonej odległości), poczuje cię tam, niezależnie od tego, czy widzi lub słyszy cię w inny sposób. Ponieważ łatwo jest zaimplementować dotyk za pomocą wykrywania kolizji, opisany tutaj system zarządzania zmysłami

nie obejmuje dotyku. Wykrywanie kolizji wykracza poza zakres tej książki. W tej serii znajdują się dwie książki [14] [73] z obszernymi szczegółami. W systemie produkcyjnym korzystne może być włączenie dotykania do struktury menedżera zmysłów. Po wykryciu kolizji między dotykającymi się znakami wysyłane jest specjalne zdarzenie dotykania. Przekierowanie tego przez menedżera zmysłów pozwala postaci na odbieranie wszystkich informacji o zmysłach jedną drogą, nawet jeśli dotyk jest inaczej obsługiwany za kulisami.

Zapach

Zapach jest stosunkowo niezbadanym zmysłem w grach. Zapachy są spowodowane dyfuzją gazów w powietrzu. Jest to proces powolny i ograniczony na odległość. Szybkość dyfuzji sprawia, że efekty wiatru są bardziej widoczne. Podczas gdy dźwięk może być przenoszony przez wiatr, jego szybki ruch sprawia, że nie zauważamy, że porusza się z wiatrem szybciej niż z wiatrem. Z wiatrem zapachy są znacznie bardziej wyczuwalne. Zazwyczaj zapachy, które nie są związane ze skoncentrowanymi chemikaliami (takie jak zapach wroga) przemieszczają się tylko na kilkadziesiąt metrów. Zwierzęta o lepszej wrażliwości na zapach mogą wykrywać ludzi ze znacznie większej odległości, przy odpowiednich warunkach wiatrowych. Gry myśliwskie są zazwyczaj jedynymi, które model pachnie. Istnieją inne potencjalne zastosowania zapachu. Wspomniana przeze mnie wcześniej gra (modelująca pośrednią transmisję światła) wykorzystywała zapach do reprezentowania dyfuzji trujących gazów. Granat gazowy mógł zostać zdetonowany na przykład poza posterunkiem straży. Menadżer zmysłów sygnalizował strażnikom, że mogą poczuć zapach gazu. W tym przypadku odpowiedzieli na zapach śmiercią. Jednym z najlepszych zastosowań symulacji zapachu jest Alien vs. Predator. Tutaj kosmici wyczuwają obecność gracza za pomocą zapachu. Gdy zapach się rozchodzi, kosmici podążają szlakiem o rosnącej intensywności, aby znaleźć lokalizację gracza. Daje to początek zgrabnej taktyce. Jeśli postać przez długi czas stoi w dobrym miejscu na zasadzkę, a następnie szybko schowa się za osłoną, kosmici podążą tropem do intensywnego miejsca zapachu, w którym wcześniej stała, dając graczowi inicjatywę do ataku.

Moda na fantazję

Oprócz wzroku, słuchu i węchu istnieje wiele zastosowań, do których można wykorzystać menedżera zmysłów. Chociaż ograniczymy symulację do tych trzech modalności, powiązane z nimi parametry oznaczają, że możemy symulować inne fikcyjne zmysły. Zmysły fantasy, takie jak aura lub magia, mogą być reprezentowane za pomocą zmodyfikowanej wizji; Telepatia może być zmodyfikowaną wersją słyszenia, a strach, reputacja lub urok mogą być zmodyfikowanym zapachem. Za pomocą menedżera zmysłów można nadawać całą gamę efektów zaklęć: ofiary zaklęcia zostaną powiadomione przez menedżera zmysłów, eliminując potrzebę przeprowadzania całej partii specjalnych testów w kodzie specyficznym dla zaklęć.

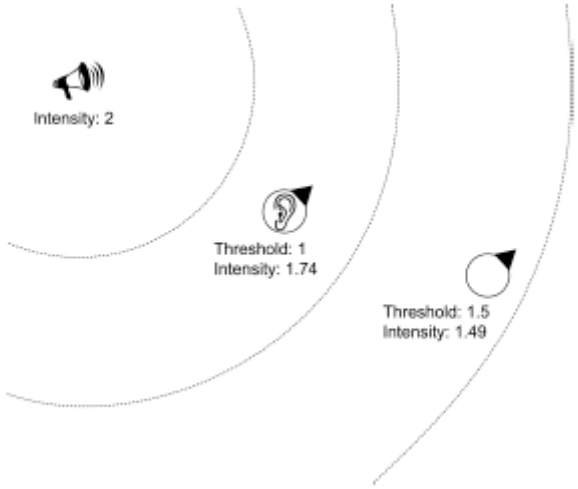
MENEDŻER REGIONU ZMYSŁU

Opiszę dwa algorytmy zarządzania zmysłami. Pierwsza to prosta technika wykorzystująca sferyczny obszar wpływu, ze stałymi prędkościami dla każdej modalności. Odmiana tej techniki jest używana w większości gier z symulacją zmysłów. Jest to również podejście preferowane przez oprogramowanie symulacyjne do animacji (takie jak Massive) i wojsko.

Algorytm

Algorytm działa w trzech fazach: czujniki potencjału znajdują się w fazie agregacji; czujniki potencjału są sprawdzane pod kątem przejścia sygnału w fazie testowania; a sygnały, które przechodzą, są wysyłane w fazie powiadamiania. Postacie rejestrują swoje zainteresowanie menedżerowi zmysłów

wraz ze swoją pozycją, orientacją i zdolnościami sensorycznymi. Jest on przechowywany jako czujnik, równoważny strukturze słuchacza w menedżerze zdarzeń. W praktycznym zastosowaniu pozycja i orientacja są zwykle podawane jako wskaźnik do danych pozycyjnych postaci, więc postać nie musi stale aktualizować menedżera zmysłów podczas ruchu. Zdolności sensoryczne składają się z wartości progowej dla każdej modalności, którą postać może wyczuć. Menedżer zmysłów może obsługiwać dowolną liczbę modalności. Z każdą modalnością związany jest współczynnik tłumienia, maksymalny zasięg i odwrotna prędkość transmisji. Menedżer zmysłów akceptuje sygnały: komunikaty, które wskazują, że coś się wydarzyło na poziomie gry (odpowiednik wydarzeń w menedżerze wydarzeń). Sygnały są podobne do zdarzeń używanych w menedżerze zdarzeń, ale zawierają trzy dodatkowe dane: sposób, w jaki sygnał powinien być przesyłany, intensywność sygnału u jego źródła oraz położenie źródła. Współczynnik tłumienia odpowiadający każdej modalności określa, jak głośność dźwięku lub intensywność zapachu spada wraz z odległością. Dla każdej jednostki odległości intensywność sygnału jest mnożona przez współczynnik tłumienia. Algorytm przestaje przetwarzać transmisję poza maksymalny zasięg. Gdy intensywność sygnału spadnie poniżej wartości progowej postaci, postać nie jest w stanie go wyczuć. Oczywiście maksymalny zasięg modalności powinien być tak dobrany, aby był na tyle duży, aby dotrzeć do dowolnych postaci, które byłyby w stanie odbierać odpowiednie sygnały. Rysunek 11.2 przedstawia ten proces dla sygnału dźwiękowego.



Menedżer zmysłów ma zarejestrowane tłumienie dźwięku 0,9. Sygnał o natężeniu 2 jest emitowany z pokazanego źródła. W odległości 1 jednostki od źródła natężenie dźwięku wynosi 1,8, w odległości 2 jednostek 1,62 i tak dalej. Znak A ma próg dźwięku równy 1. W odległości 1,5 jednostki dźwięk ma natężenie około 1,7, a znak A jest powiadamiany o dźwięku. Znak B ma próg 1,5. W odległości 2,8 jednostki dźwięk ma natężenie 1,49, a znak B nie jest powiadamiany. Odwrotna prędkość transmisji wskazuje, ile czasu zajmie sygnałowi pokonanie jednej jednostki odległości. Nie używamy nieodwróconej prędkości, ponieważ chcemy być w stanie poradzić sobie z nieskończoną prędkością związaną z widzeniem. Podstawowy algorytm działa w ten sam sposób dla każdej modalności. Kiedy sygnał zostaje wprowadzony do menedżera sensu, natychmiast znajduje on wszystkie znaki w maksymalnym promieniu odpowiedniej modalności (faza agregacji). Dla każdej postaci oblicza intensywność sygnału, kiedy dotrze do postaci i czas, w którym to nastąpi. Jeśli intensywność jest poniżej progu postaci, jest ignorowana. W przypadku pozytywnego wyniku testu intensywności algorytm może wykonać dodatkowe testy, w zależności od rodzaju modalności. Jeśli wszystkie testy zakończą się pomyślnie, w kolejce umieszczana jest prośba o powiadomienie postaci. To jest faza testowania. Rekordy kolejki przechowują sygnał, czujnik do powiadomienia, intensywność i czas dostarczenia wiadomości (liczony od czasu wyemitowania sygnału i czasu, w którym sygnał dotrze do postaci). Przy każdym uruchomieniu menedżera sensu sprawdza on kolejkę pod kątem komunikatów,

których czas minął, i dostarcza je. To jest faza powiadamiania. Algorytm ten ujednolica sposób działania zapachów i dźwięków (dźwięki to po prostu szybko poruszające się zapachy). Żaden z nich nie wymaga dodatkowych testów; test intensywności jest wystarczający. Modalności oparte na wzroku wymagają dwóch dodatkowych testów w fazie testowania. Najpierw testowane jest źródło sygnału, aby upewnić się, że znajduje się w aktualnym stożku widzenia postaci. Jeśli ten test przejdzie pomyślnie, wykonywany jest rzut promienia, aby upewnić się, że istnieje linia wzroku. Jeśli chcesz wesprzeć kamuflaż lub ukrywanie się w cieniu, możesz dodać tutaj dodatkowe testy. Te rozszerzenia są omówione poniżej głównego algorytmu. Zauważ, że ten model pozwala nam mieć postacie o stałych odległościach widzenia: pozwalamy na osłabianie sygnałów wizualnych wraz z odległością i nadajemy różnym postaciom różne progi. Jeżeli natężenie sygnału wizualnego jest zawsze takie samo (rozsądne założenie), to próg narzuca maksymalny promień widzenia wokół postaci

Pseudo kod

Menadżer zmysłów można wdrożyć w następujący sposób:

```
1 class RegionalSenseManager:
2 # A record in the notification queue, ready to notify the sensor
3 # at the correct time.
4 class Notification:
5 time: int
6 sensor: Sensor
7 signal: Signal
8
9 # The list of sensors.
10 sensors: Sensor[]
11
12 # A queue of notifications waiting to be honored.
13 notificationQueue: Notification[]
14
15 # Introduces a signal into the game. This also calculates the
16 # notifications that this signal will be needed.
17 function addSignal(signal: Signal):
18 # Aggregation phase.
19 validSensors: Sensor[] = []
20
21 for sensor in sensors:
22 # Testing phase.
```

```
23
24 # Check the modality first.
25 if not sensor.detectsModality(signal.modality):
26 continue
27
28 # Find the distance of the signal and check range.
29 distance = distance(signal.position, sensor.position)
30 if signal.modality.maximumRange < distance:
31 continue
32
33 # Find the intensity of the signal and check
34 # the threshold.
35 intensity = signal.strength *
36 pow(signal.modality.attenuation, distance)
37 if intensity < sensor.threshold:
38 continue
39
40 # Perform additional modality specific checks.
41 if not signal.modality.extraChecks(signal, sensor):
42 continue
43
44 # Notification phase.
45
46 # We're going to notify the sensor, work out when.
47 time = getCurrentTime() +
48 distance * signal.modality.inverseTransmissionSpeed
49
50 # Create a notification record and add it to the queue.
51 notification = new Notification()
52 notification.time = time
53 notification.sensor = sensor
```

```
54 notification.signal = signal
55 notificationQueue.add(notification)
56
57 # Send signals, in case the current signal is ready to
58 # notify immediately.
59 sendSignals()
60
61 # Flush notifications from the queue, up to the current time.
62 function sendSignals():
63 # Notification Phase.
64 currentTime: int = getCurrentTime()
65
66 while notificationQueue:
67 notification: Notification = notificationQueue.peek()
68
69 # Check if the notification is due.
70 if notification.time < currentTime:
71 notification.sensor.notify(notification.signal)
72 notificationQueue.pop()
73
74 # If we are beyond the current time, then stop
75 # (assuming the queue is sorted).
76 else:
77 break
```

Kod zakłada funkcję getCurrentTime, która zwraca bieżący czas gry. Zakłada również istnienie funkcji matematycznej pow. Należy zauważyć, że funkcja sendSignals powinna być wywoływana w każdej ramce, niezależnie od tego, czy wprowadzono jakiekolwiek sygnały, aby upewnić się, że buforowane powiadomienia są prawidłowo wysyłane.

Struktury danych i interfejsy

Ten kod zakłada interfejs dla modalności, czujników i sygnałów. Sposoby zgodne z interfejsem:

```
1 class Modality:
2 maximumRange: float
```

3 attenuation: float

4 inverseTransmissionSpeed: float

5

6 function extraChecks(signal: Signal, sensor: Sensor) -> bool

gdzie extraChecks przeprowadza kontrole specyficzne dla modalności w fazie testowania. Będzie to realizowane inaczej dla każdej konkretnej modalności. Niektóre modalności zawsze mogą zdać ten test. Dla wzroku możemy mieć:

1 class SightModality:

2 function extraChecks(signal: Signal, sensor: Sensor) -> bool

3 if not checkSightCone(signal.position,

4 sensor.position,

5 sensor.orientation):

6 continue

7 if not checkLineOfSight(signal.position,

8 sensor.position):

9 continue

gdzie checkSightCone i checkLineOfSight przeprowadzają poszczególne testy; oba zwracają prawdę, jeśli zdają. Czujniki posiadają interfejs:

1 class Sensor:

2 position: Vector

3 orientation: Quaternion

4

5 function detectsModality(modality: Modality) -> bool

6 function notify(signal: Signal)

gdzie detectsModality zwraca prawdę, jeśli czujnik może wykryć daną modalność; modalność jest instancją modalności. Metoda powiadamiania jest taka sama, jak w przypadku zwykłego zarządzania zdarzeniami: powiadamia czujnik o sygnale. Sygnały mają interfejs:

1 class Signal:

2 strength: float

3 position: Vector

4 modality: Modality

Oprócz tych trzech interfejsów kod zakłada, że notyfikacjaQueue jest zawsze posortowana według czasu. Ma strukturę:

1 class NotificationQueue:

2 function add(notification: Notification)

3 function peek() -> Notification

4 function pop() -> Notification

gdzie metoda add dodaje dane powiadomienie we właściwe miejsce w kolejce. Ta struktura danych jest kolejką priorytetową, aby czas. Ze względu na jego zastosowanie w algorytmie A* Część 4 zawiera więcej szczegółów na temat efektywnej implementacji kolejek priorytetowych.

Wydajność

Regionalny menedżer zmysłów to O(nm) w czasie, gdzie n to liczba zarejestrowanych czujników, amis liczba sygnałów. Przechowuje tylko oczekujące sygnały, więc jest to O(p) w pamięci, gdzie p jest liczbą oczekujących sygnałów. W zależności od prędkości sygnałów może to zbliżyć się do O(m) w pamięci, ale w większości przypadków będzie znacznie mniejsze.

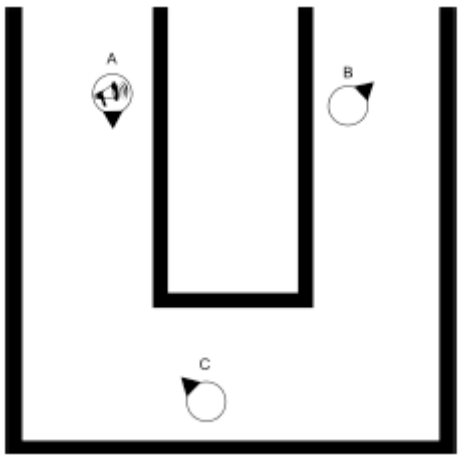
Kamuflaż i Cienie

Aby wesprzeć kamuflaż, możemy dodać dodatkowy test dla wizualnych modalności w klasie SightModality. Po rzuceniu promienia, aby sprawdzić, czy sygnał znajduje się w linii wzroku z postacią, wykonujemy jeden lub więcej dodatkowych promieni zarzucanych poza postać. Znajdujemy materiały związane z pierwszym obiektem, który przecina każdy promień.

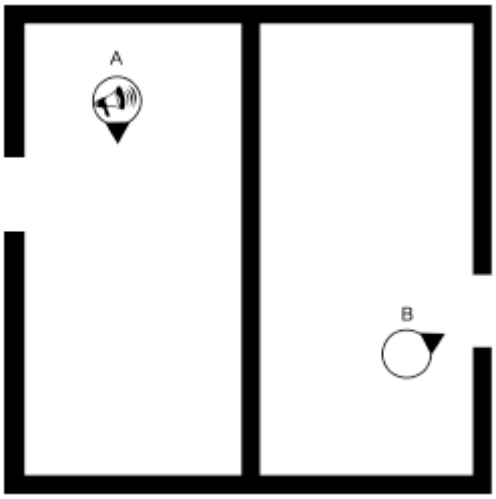
Zazwyczaj projektant poziomów oznacza każdy materiał zgodnie z jego rodzajem wzoru. Możemy mieć na przykład dziesięć typów wzorów, w tym cegła, liście, kamień, trawa, niebo i tak dalej. W oparciu o rodzaje materiałów tła obliczany jest dodatkowy współczynnik tłumienia. Załóżmy, że postać ma na sobie zielony kamuflaż. Projektant może zdecydować, że tło liści daje dodatkowe tłumienie 0,1, podczas gdy niebo daje dodatkowe tłumienie 1,5. Dodatkowe tłumienie jest mnożone przez siłę sygnału i przekazywane tylko wtedy, gdy wynik jest wyższy niż próg znaku. Możemy użyć podobnego procesu, aby wesprzeć ukrywanie się w cieniu. Łatwiejszą metodą jest po prostu uczynienie początkowej siły sygnału proporcjonalną do światła padającego na jego emiter. Jeśli postać jest w pełnym świetle, wysyła silne sygnały „Jestem tutaj” do menedżera zmysłów. Jeśli postać znajduje się w cieniu, siła sygnału będzie niższa, a postacie z progiem wysokiej intensywności mogą ich nie zauważyć.

Słabości

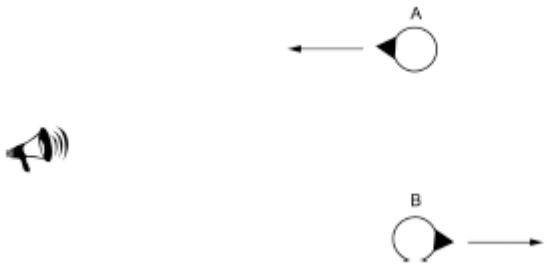
Rysunek przedstawia sytuację, w której załamuje się implementacja prostego menedżera sensu.



Dźwięk emitowany przez znak A jest słyszany jako pierwszy przez znak C, mimo że C jest dalej od źródła niż znak B. Transmisja jest zawsze obsługiwana przez odległość i nie uwzględnia geometrii poziomu, z wyjątkiem testów linii wzroku . Ta niewielka rozbieżność w czasie może nie być zbyt zauważalna. Rysunek przedstawia poważniejszą sytuację.



Tutaj postać B słyszy dźwięk, mimo że B nie znajduje się w pobliżu źródła dźwięku i jest izolowana dużą barierą. Dotychczasowy kod zakładał również, że znaki są zawsze nieruchome. Weźmy przypadek z rysunku.



Dwie postacie zaczynają się w tej samej odległości od dźwięku. Jedna postać porusza się szybko w kierunku źródła. Realistycznie rzecz biorąc, postać A usłyszy dźwięk wcześniej niż znak B w punkcie zaznaczonym na schemacie. W naszym modelu jednak słyszą dźwięk razem. Zwykle nie jest to zauważalne w przypadku dźwięków, ponieważ mają tendencję do poruszania się znacznie szybciej niż postacie. Jednak w przypadku zapachów może mieć duże znaczenie. Ten algorytm zarządzania

zmysłami jest bardzo prosty, szybki i wydajny. Doskonale sprawdza się w przypadku poziomów na wolnym powietrzu lub w pomieszczeniach, w których grubość ścian jest większa niż odległość, jaką może pokonać sygnał. Jednak w środowiskach powszechnych w grach akcji z perspektywy pierwszej i trzeciej osoby może to powodować nieprzyjemne artefakty. Korzystanie z tego rodzaju menedżera zmysłów ułatwia rozszerzenie gry o dodatkowe testy, kod na specjalne przypadki i heurystykę, aby sprawić wrażenie omijania ograniczeń algorytmu. Zamiast tracić czas na łatanie podstawowego systemu (co jest poprawnym planem, o ile łaty nie wymagają zbyt dużego wysiłku wdrożeniowego), opiszę bardziej kompleksowe rozwiązanie. Należy jednak pamiętać, że wraz ze wzrostem zaawansowania pojawią się odpowiednio większe wymagania dotyczące przetwarzania.

MENEDŻER CZUJNIKÓW MODELI ELEMENTÓW SKOŃCZONYCH

Dokładne modelowanie wzroku, słuchu i węchu wymagałoby poważnego wysiłku rozwojowego. W eksperymentach kodowania przeprowadzonych w mojej firmie zajmującej się oprogramowaniem pośredniczącym AI na początku 2000 roku, przyjrzeliśmy się tworzeniu geometrycznie dokładnej symulacji zmysłów. Zadanie jest ogromne i wtedy było całkowicie niepraktyczne. Jestem przekonany, że nie ma praktycznego sposobu, aby to zrobić nawet w przypadku obecnej generacji sprzętu. Zamiast tego opracowaliśmy mechanizm oparty na modelach elementów skończonych, który działa dobrze i może być dość wydajny. Później dowiedzieliśmy się, że była to technika opracowana niezależnie przez co najmniej dwóch innych programistów (nic dziwnego, biorąc pod uwagę jej podobieństwo do innych algorytmów gier).

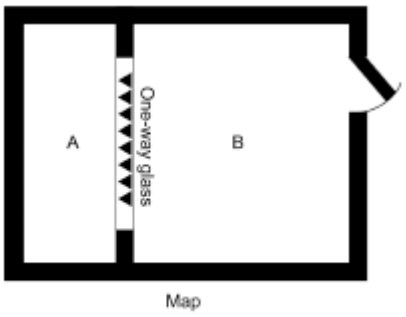
Modele elementów skończonych

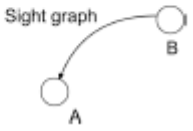
Model elementów skończonych (MES) dzieli ciągły problem na skończoną liczbę elementów dyskretnych. Zastępuje trudność rozwiązania problemu dla nieskończonej liczby lokalizacji w ciągłym świecie problemem dla skończonej liczby lokalizacji. Chociaż odnajdywanie ścieżek nie wykorzystuje ściśle MES, stosuje bardzo podobne podejście. Dzieli ciągły problem na elementy skończone, w taki sam sposób, w jaki będziemy musieli to zrobić dla naszego algorytmu. (Nie jest to ściśle MES, ponieważ nie stosuje algorytmu do każdego regionu równolegle; stosuje algorytm jednorazowy do całego modelu). Dzieląc ciągły problem na regiony, prostsze algorytmy mogą być stosowane. W odnajdywaniu ścieżek zamieniamy trudny problem znajdowania najszybszej trasy przez dowolną geometrię 3D z prostszym problemem przemierzania wykresu. Za każdym razem, gdy używasz MES do rozwiązania problemu, dokonujesz uproszczonego przybliżenia. Nie rozwiązując prawdziwego problemu, ryzykujesz odzyskanie tylko przybliżonego rozwiązania. Dopóki aproksymacja jest dobra, model działa.

Opisałem ten proces aproksymacji zarówno dla celów wyszukiwania ścieżki, jak i analizy taktycznej, wraz ze wskazówkami, jak podzielić poziom na regiony, aby powstałe ścieżki były wiarygodne, a taktyka spójna. Podobnie, gdy używamy MES do modelowania percepcji w grze, musimy ostrożnie wybierać regiony, aby upewnić się, że uzyskany wzór percepcji postaci jest wiarygodny.

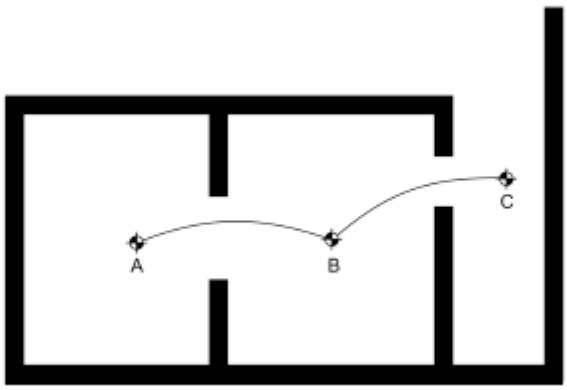
Wykres zmysłów

W taki sam sposób, jak w przypadku odnajdywania ścieżki, przekształcamy poziom gry w ukierunkowany acykliczny wykres do zarządzania zmysłami. Każdy węzeł na wykresie reprezentuje region poziomu gry, w którym sygnały mogą przepływać bez przeszkód. Dla każdej modalności opartej na zapachu węzeł zawiera wartość rozpraszania, która wskazuje, ile zapachu zaniknie na sekundę. Na przykład rozproszenie 0,5 oznacza, że zapach traci co sekundę o połowę swoją intensywność. Dla wszystkich modalności węzeł zawiera wartość tłumienia, która wskazuje, jak zanika sygnał dla każdej jednostki przebytej odległości. Połączenia są tworzone między parami węzłów, gdzie jedna lub więcej modalności może przechodzić między odpowiednimi regionami. Rysunek pokazuje przykład.

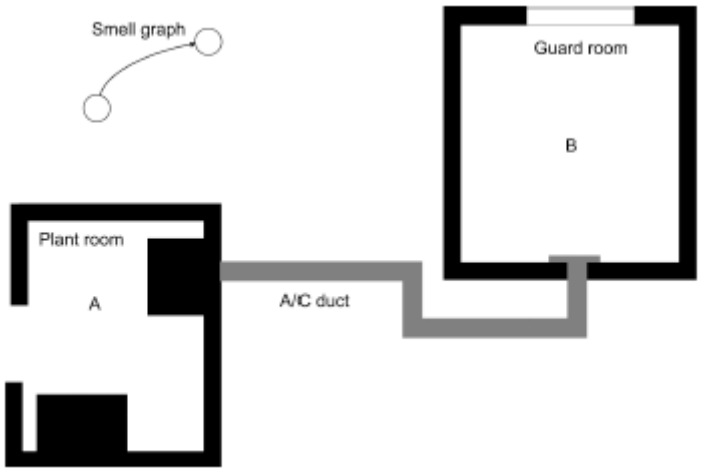




Dwa oddzielne pokoje oddzielone są dźwiękoszczelnym, jednokierunkowym oknem. Wykres zmysłów zawiera dwa węzły, po jednym dla każdego pomieszczenia. Sala A jest połączona z salą B, ponieważ bodźce wzrokowe mogą przechodzić w tym kierunku, chociaż dźwięk i zapach nie mogą. Sala B nie jest jednak połączona z salą A, ponieważ żadne bodźce nie mogą przejść w tym kierunku. Dla każdej modalności połączenie ma odpowiedni współczynnik tłumienia i odległość. To pozwala nam obliczyć ilość przesyłanego sygnału. W powyższym przykładzie połączenie będzie miało tłumienie 0 zarówno dla zapachu, jak i dźwięku (nie przepuszcza żadnego). Ma współczynnik tłumienia 0,9 dla widzenia, aby symulować fakt, że okno jest przyciemnione. Odległość wzdłuż połączenia jest podana jako 1, dla uproszczenia (więc całkowite tłumienie przez okno wyniesie 0,9). Głównym powodem posiadania zarówno tłumienia, jak i odległości, jest umożliwienie wolno poruszającym się sygnałom (mianowicie zapachom) czasu na przemieszczanie się wzdłuż połączenia. Połączenia mają również skojarzoną pozycję 3D dla obu ich końców, jak pokazano na rysunku.



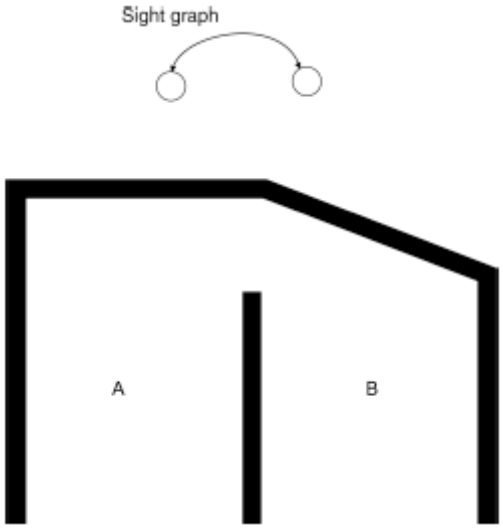
Pozycja połączenia służy do ustalenia, w jaki sposób sygnał przesyłany jest przez węzeł z połączenia przychodzącego. Ponieważ węzły zwykle graniczą ze sobą, często punkt początkowy i końcowy połączenia znajdują się w tej samej pozycji: algorytm poradzi sobie z tą sytuacją. Odległość skojarzona z połączeniem nie musi być taka sama jak odległość 3D między jego punktem początkowym i końcowym. Algorytm zajmuje się nimi całkowicie oddzielnie. Nie ma powodu, aby połączenia ograniczały się do pobliskich regionów poziomu. Rysunek ilustruje połączenie na duże odległości, które umożliwia tylko wąchanie.



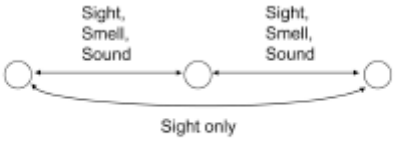
To przykład z nieszczęsnej gry opartej na zmysłach, którą przedstawiłem wcześniej. Połączenie reprezentuje kanał klimatyzacyjny, kluczową zagadkę w grze. Rozwiązanie polega na zdetonowaniu granatu z trującym gazem w pokoju A i przepuszczeniu go przez kanał klimatyzacji, aby zabić strażnika stojącego w pokoju B. Kanał jest jedynym połączeniem między dwoma pomieszczeniami. Innym przypadkiem może być pokój kontrolny z łączami wideo do kilku pomieszczeń na poziomie; mogą istnieć wizualne powiązania między salą konferencyjną a badanymi obszarami, nawet jeśli znajdują się one w pewnej odległości. Strażnicy w dyspozytorni byliby powiadamiani i reagowali na zdarzenia uchwycone przez kamerę.

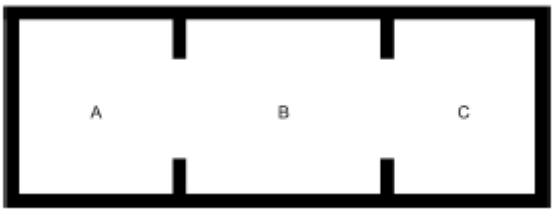
Widzenie

Wzrok zasługuje na szczególną wzmiankę. Połączenie między dwoma węzłami powinno umożliwiać przekazywanie sygnałów wzrokowych, jeśli dowolna lokalizacja w węźle docelowym jest widoczna z dowolnego miejsca w źródle. Ogólnie rzecz biorąc, w węźle docelowym będzie wiele lokalizacji, których nie można wyświetlić z wielu lokalizacji w źródle. Jak zobaczymy, te przypadki zostaną uwięzione przez testy linii wzroku w algorytmie. Ale testy linii wzroku nie będą brane pod uwagę, jeśli węzły nie są połączone. Rysunek pokazuje połączenie między dwoma pokojami, mimo że tylko bardzo mały obszar Pokoju B jest widoczny z Pokoju A, i to tylko po przesunięciu się w róg Pokoju A.



Inną konsekwencją poniższego algorytmu jest to, że wszystkie pary węzłów które mają połączone linie widzenia, muszą mieć połączenie. W przeciwieństwie do odnajdywania ścieżki, nie możemy polegać na węzłach pośrednich do przenoszenia informacji. Nie dotyczy to modalności innych niż wzrok. Rysunek przedstawia poprawny wykres zmysłu dla serii trzech pomieszczeń.





Zwróć uwagę, że istnieją połączenia wzrokowe między pokojami A i C, mimo że pomieszczenie B jest na drodze. Nie ma jednak połączeń zapachowych ani dźwiękowych między pokojami A i C. Menedżerowie Sense korzystający z tego modelu czasami używali oddzielnego wykresu dla wzroku, ponieważ jest on wyspecjalizowany. Szczególnie solidna implementacja wykorzystuje dane potencjalnie widocznego zestawu (PVS) z silnika renderującego do obliczenia wykresu widoku. Potencjalnie widoczny zestaw to nazwa nadana szeregowi technik graficznych używanych do zmniejszenia ilości geometrii, która musi być renderowana w każdej klatce. Jest to standardowa funkcja wszystkich nowoczesnych silników renderujących. W poniższym algorytmie użyję jednego wykresu dla wszystkich zmysłów, ale ponieważ każdy zmysł jest obsługiwany nieco inaczej, stosunkowo prostą modyfikacją jest zastąpienie jednego wykresu dwoma lub więcej.

Algorytm

Algorytm działa w tych samych trzech fazach, co poprzednio: agregując czujniki, które mogą zostać powiadomione, testując je w celu sprawdzenia, czy są prawidłowe i powiadamiając o sygnale. Tak jak poprzednio, menedżer zmysłów jest powiadamiany o sygnałach z zewnętrznego kodu (często jakiegoś mechanizmu odpytywania), który nie jest częścią algorytmu. Sygnały są dostarczane wraz z ich lokalizacją, intensywnością, modalnością i wszelkimi dodatkowymi danymi, które muszą zostać przekazane. Menedżer zmysłów przechowuje również listę czujników: detektorów zdarzeń zdolnych do wykrywania jednej lub więcej modalności. Ponownie zawierają one listę modalności i wartości progowych intensywności. Zostaną powiadomieni o każdym sygnale, który są w stanie wykryć. Algorytm otrzymuje również graf sensu wraz z pewnym mechanizmem kwantyzacji lokalizacji w świecie gry na węzły w grafie sensu. Zarówno czujniki, jak i sygnały muszą zostać skwantowane w węźle, zanim algorytm będzie mógł działać. Ta kwantyzacja może być wykonana dokładnie tak, jak kwantyzacja ze znajdowaniem ścieżki. Więcej szczegółów znajdziesz w Części 4 o odnajdywaniu ścieżek. Wewnętrznie menedżer czujników przechowuje czujniki dla każdego węzła, dzięki czemu może szybko znaleźć czujniki obecne w danym węźle. W zależności od rodzaju modalności algorytm zachowuje się nieco inaczej. W celu zwiększenia złożoności obrazy, dźwięki i zapachy są obsługiwane przez różne podalgorytmy.

Widzenie

Celowniki to najprostsze sygnały do obsługi. Po wprowadzeniu celownika algorytm otrzymuje listę potencjalnych czujników: jest to faza agregacji. Ta lista zawiera wszystkie czujniki w tym samym węźle

co sygnał oraz wszystkie czujniki w węzłach, które są połączone z tym węzłem. Przestrzegany jest tylko jeden zestaw połączeń; nie pozwalamy, aby sygnały wizualne rozprzestrzeniały się po całym poziomie. Jeśli chcesz zasymulować radiosity, jak wspomniano wcześniej, dwa zestawy połączeń można śledzić wtedy i tylko wtedy, gdy sygnał wizualny emituje światło. Algorytm przechodzi następnie do fazy testowania. Lista potencjalnych czujników jest testowana dokładnie tak, jak w regionalnym menedżerze wyczuwania. Sprawdza się, czy interesują ich bodźce wzrokowe, czy sygnał będzie miał wystarczające natężenie, czy sygnał znajduje się w stożku wzroku i czy jest na linii wzroku. Kontrast tła można również sprawdzić, dokładnie tak jak poprzednio. Dane dotyczące czasu i intensywności są obliczane na podstawie danych dotyczących pozycji, transmisji i odległości w każdym połączeniu. To samo dotyczy wszystkich trzech modalności i jest szczegółowo opisane poniżej. Jeśli czujnik przejdzie wszystkie testy, wówczas kierownik ustala, kiedy należy go powiadomić, na podstawie jego odległości od bodźca (obliczonej jako odległość euklidesowa w trzech wymiarach, w przeciwieństwie do innych modalności poniżej). Powiadomienie jest następnie dodawane do kolejki powiadomień, dokładnie tak jak poprzednio. Jeśli w twojej grze wzrok jest zawsze natychmiastowy, możesz pominąć ten krok i natychmiast powiadomić czujnik.

Dźwięk

Podobnie traktuje się dźwięk i zapach, ale z jednym zasadniczym wyróżnieniem. Z czasem zapachy utrzymują się w danym regionie. Dźwięki w naszym modelu nie (nie bierzemy pod uwagę np. echa, chociaż można je modelować wysyłając świeże dźwięki co kilka klatek). Dźwięk traktujemy jak falę, rozchodzącą się od źródła i coraz bardziej słabnącą. Kiedy osiągnie swój minimalny limit intensywności, znika na zawsze. Oznacza to, że dźwięk może być odbierany tylko wtedy, gdy fala Cię mija. Jeśli fala dźwiękowa dotarła do krawędzi pomieszczenia, dźwięk nie jest już słyszalny w pomieszczeniu. Modelowanie dźwięków zaczynamy od węzła, w którym znajduje się źródło dźwięku. Algorytm wyszukuje wszystkie czujniki w tym węźle. Oznacza węzeł jako odwiedzony. Następnie podąża za połączeniami oznaczonymi jako dźwięk, zmniejszając intensywność o wartość określoną przez połączenie. Kontynuuje ten proces tak daleko, jak to możliwe, pracując od węzła do węzła za pośrednictwem połączeń i oznaczając każdy odwiedzany węzeł. Jeśli dotrze do węzła, w którym już był, nie przetwarza go ponownie. Węzły są przetwarzane w porządku odległości (który jest równy porządkowi czasu, jeśli założymy, że dźwięk rozchodzi się ze stałą prędkością). W każdym odwiedzanym węźle gromadzona jest lista potencjalnych czujników. Jeśli intensywność dźwięku jest poniżej minimalnej intensywności, nie są przetwarzane żadne węzły. Intensywność oblicza się w ten sam sposób dla każdej modalności i opisano poniżej. W fazie testowania dla każdego czujnika wykonywane są kontrole intensywności: te, które są w stanie odebrać sygnał, mają dodawane żądanie powiadomienia do kolejki gotowej do fazy wysyłki.

Zapach

Zapach zachowuje się w sposób bardzo podobny do dźwięku. Dźwięk śledzi każdy węzeł, przez który przeszedł, i odmawia przetwarzania poprzednich węzłów. Zapach zastępuje to przechowywaną intensywnością i powiązaną informacją o czasie. Każdy węzeł może mieć dowolną utrzymującą się intensywność zapachu, więc przechowuje wartość intensywności zapachu. Aby upewnić się, że ta wartość jest dokładnie aktualizowana, przechowywana jest również wartość czasu. Wartość czasu wskazuje, kiedy intensywność była ostatnio aktualizowana. Za każdym razem, gdy algorytm jest uruchamiany, rozsyła swój zapach do sąsiadów w oparciu o transmisję i odległość interweniujących połączeń. Nie rozchodzi się, jeśli intensywność źródła lub nowego miejsca docelowego jest poniżej progu minimalnej intensywności lub jeśli sygnał nie może dotrzeć do miejsca docelowego w czasie symulowanym przez menedżera czujników. Ten czas symulacji zwykle odpowiada okresowi między wywołaniami menedżera wykrywania (być może ramka). Ograniczenie go przez czas w ten sposób

zapobiega szybszemu rozprzestrzenianiu się zapachu na wykresie zmysłowym niż na poziomie. Zapach w pojedynczym węźle wygasa w oparciu o parametr rozpraszania węzła. Aby uniknąć wielokrotnego aktualizowania węzła na iterację menedżera wykrywania, przechowywany jest znacznik czasu. Węzeł jest przetwarzany tylko wtedy, gdy jego znacznik czasu jest mniejszy niż czas bieżący. W każdej iteracji agreguje czujniki z każdego węzła, w którym Wydajność uje intensywność większa niż wartość minimalna. Są one następnie testowane w fazie testowej pod kątem zainteresowania modalnością i progiem intensywności. Prośby o powiadomienia są zaplanowane dla tych, które przechodzą w normalny sposób.

Obliczanie intensywności od węzła do węzła

Aby obliczyć intensywność i czas podróży bodźca niewizualnego w miarę przemieszczania się od węzła do węzła, podzieliliśmy podróż na trzy sekcje: podróż od źródła do początku połączenia, podróż wzdłuż połączenia oraz podróż od końca połączenia do czujnika (lub do początku następnego połączenia, jeśli podróż jest wielostopniowa). Całkowity czas jest wyrażony jako prędkość modalności podzielona przez całkowitą odległość: odległość od sygnału do początku połączenia (odległość euklidesowa 3D), odległość wzdłuż połączenia (przechowywana jawnie) oraz odległość do czujnika (inna odległość 3D).

Całkowite tłumienie jest określane przez współczynnik tłumienia każdego komponentu: tłumienie dla węzła, w którym znajduje się źródło, tłumienie połączenia i tłumienie węzła czujnika.

Algorytm iteracyjny

Do tej pory zakładaliśmy, że cała propagacja wzroku i dźwięku jest obsługiwana w jednym przebiegu menedżera zmysłów. Z zapachem, ponieważ pełza i stopniowo się rozprasza, należy postępować iteracyjnie. Wzrok działa tak szybko, że natychmiast musimy przetworzyć wszystkie jego efekty. Dźwięk może zajmować pośrednią pozycję. Jeśli porusza się wystarczająco wolno, może skorzystać na traktowaniu go jak zapachu: jest rozprowadzany przez kilka węzłów i połączeń za każdym razem, gdy uruchamiany jest menedżer zmysłów. Ten sam znacznik czasu używany do aktualizacji zapachu może być używany do aktualizacji dźwięku, o ile nie szukasz doskonałej dokładności w odniesieniu do sposobu rozszerzania się fali dźwiękowej. (Idealnie chcielibyśmy przetwarzać węzły od źródła na zewnątrz, ale użycie tylko jednego znacznika czasu oznacza, że nie możemy tego zrobić dla każdego źródła). Oprogramowanie pośredniczące menedżera zmysłów, które zbudowali moi koledzy przy użyciu tego algorytmu, pozwalało na spowolnienie dźwięku tego rodzaju. W praktyce jednak nigdy nie był potrzebny. Jeśli dźwięk był obsługiwany natychmiast, był równie wiarygodny.

Wysyłka

Na koniec algorytm wysyła wszystkie zdarzenia bodźcowe do czujników, które zostały zagregowane i przetestowane. Czyni to w oparciu o czas, dokładnie tak, jak w przypadku kierownika ds. wyczucia regionu. W przypadku zapachów lub wolno poruszających się dźwięków generowane są tylko powiadomienia na najbliższą przyszłość. Jeśli dźwięk jest obsługiwany w jednej iteracji, kolejka może przechowywać powiadomienie przez kilka milisekund lub sekund.

Uwagi dotyczące implementacji

Jeśli zapachy są wykluczone, ten algorytm zachowuje się podobnie do regionalnego menedżera zmysłów. Korzystanie z reprezentacji opartej na wykresie skutecznie przyspiesza wykrywanie czujników kandydujących (faza agregacji) i zatrzymuje dodatkowe przypadki, w których oryginalny algorytm dawał błędne wyniki (takie jak modalności przechodzące przez ściany). Jest stosunkowo wolny od stanów (musi tylko przechowywać, które węzły zostały sprawdzone pod kątem transmisji dźwięku). Dodawanie zapachów lub sprawdzanie dźwięku podzielone na wiele iteracji zmienia go w

zupełnie inną bestię. Potrzeba znacznie więcej stanu, a zapachy przechodzące w przód iw tył między węzłami mogą radykalnie zwiększyć liczbę potrzebnych obliczeń. Chociaż zapach ma swoje zastosowania i może umożliwić nową, świetną rozgrywkę, radzę wdrożyć go tylko wtedy, gdy tego potrzebujesz.

Słabości

Gdy dźwięk jest przetwarzany w całości w jednej klatce, algorytm ten ma te same słabości, co menedżer wykrywania regionu: potencjalnie możemy zostać powiadomieni w niewłaściwym czasie. W przypadku bardzo szybko poruszających się postaci może to być zauważalne. Algorytm ten usunął problem związany z zapachem i może go całkowicie rozwiązać, jeśli dźwięk jest obsługiwany iteracyjnie (oczywiście kosztem dodatkowej pamięci i czasu).

Tworzenie treści

Algorytm ten zapewnia wiarygodną symulację zmysłów i radzi sobie z naprawdę interesującymi projektami poziomów: szkło jednokierunkowe, jednostki klimatyzacyjne, kamery wideo, wietrzne korytarze i tak dalej. Zarządzanie zmysłami FEM i podobne algorytmy są najnowocześniejszym rozwiązaniem w symulacji zmysłów w grach. Jak często w tej książce, stan wiedzy jest tutaj synonimem kompleksu. Najtrudniejszym elementem tego algorytmu są dane źródłowe; dokładne określenie wykresów sensu wymaga dedykowanego wsparcia narzędziowego. Projektant poziomów musi być w stanie zaznaczyć, dokąd mogą pójść różne modalności. Zgrubne przybliżenie można uzyskać za pomocą geometrii poziomu, wystrzeliwując promienie dookoła, ale to nie poradzi sobie z efektami specjalnymi, takimi jak szklane okna, kanały lub telewizja przemysłowa. Na razie symulacja zmysłów jest luksusem, a jeśli twoja gra nie zawiera jej funkcji, lepszym rozwiązaniem jest prostsze rozwiązanie, takie jak regionalne zarządzanie zmysłami lub menedżer zdarzeń waniliowych. Ale tendencja do zwiększania wszechobecności symulacji zmysłów, szczególnie w grach akcji z perspektywy pierwszej i trzeciej osoby z elementami skradanki (nie są one tak ważne w mniej realistycznych gatunkach). Może nie minąć dużo czasu, zanim spodziewana jest złożona symulacja zmysłów.