

Jacek Śliwerski (rzyjontko)

Designing network protocols for multiplayer games

18 marca 2003 roku

Streszczenie

Niniejszy tekst opisuje najważniejsze zagadnienia związane z projektowaniem protokołów gier sieciowych. Koncentrować się będą na zagadnieniach związanych z grami o wartkiej akcji, ale większość poniższych wskazówek dotyczy wszystkich gier sieciowych.

Niestety tekst ten jest w dużej mierze niekompletny. Napisałem go aby lepiej przygotować się do poprowadzenia wykładu na seminarium, więc sporo może tu być niedopowiedzeń i niejasności. Brakuje ilustracji, a niektóre zagadnienia zostały pominięte bądź skwitowane zaledwie kilkoma zdaniem.

Spis treści

Streszczenie	1
1 Kilka słów na wstępie	1
2 Gra sieciowa to nie gra lokalna	2
3 Gracze oszukują	2
4 Dobrze jest ufać, lepiej kontrolować	3
5 Serwer i klient	4
6 Dylematy spowodowane niewiedzą	4
7 Wybór odpowiedniej warstwy transportowej	4
Literatura	4

1. Kilka słów na wstępie

*Longum iter est per praecepta, breve et efficax per exempla!*¹ Na początku pisania tego tekstu postanowiłem sobie, że trzymać się będę tego wspaniałego rzymskiego powiedzenia. W dalszej części artykułu rozwijać będziemy przykład jednej, konkretnej gry. Jako przykład przyjmijmy znaną chyba wszystkim rozgrywkę pochodzącą z filmu „Tron”. Polega ona na tym, że po zamkniętej przestrzeni poruszają się pojazdy zostawiające za sobą „ślady”, przez który nie można przejechać. Gra kończy się dla zawodnika, który wpadnie na okalającą ścianę, lub ślad pozostawiony przez jednego z graczy. „Tron” ma kilka istotnych właściwości, które będziemy mieć cały czas na uwadze:

¹Długa droga przez zasady i rozkazy, krótsza i efektywniejsza przez zestaw przykładów. [wolne tłumaczenie autora]

- tempo gry jest względnie duże - pojazdy poruszają się z dużymi prędkościami i mogą w każdej chwili zmienić kierunek jazdy,
- stan gry można opisać niewielką ilością współczynników (o czym przekonamy się później),
- bolidy mogą poruszać się w dowolnym kierunku (jest to odstępstwo od oryginału).

2. Gra sieciowa to nie gra lokalna

W środowisku programistów tworzących gry panuje powszechna opinia, że najgorszą decyzją dotyczącą istniejącego produktu jest dodanie do niego obsługi wielu graczy przez sieć. Dlaczego tak się dzieje? Programiści bardzo często nieświadomie korzystają z pewnych założeń, które nie są prawdziwe w grze sieciowej. Takie fragmenty programu trudno jest odnaleźć i usunąć, bez wprowadzania dodatkowych błędów.

Przyjrzyjmy się (hipotetycznej) grze „Tron” przeznaczonej dla jednego gracza. Główna pętla programu składa się z następujących części:

1. Program sprawdza, czy gracz wygenerował zdarzenie wejścia (np. wcisnął klawisz bądź poruszył myszką).
2. Program weryfikuje poprawność wygenerowanego zdarzenia.
3. Program generuje odpowiednie wejścia dla przeciwników gracza.
4. System fizyki oblicza nowe parametry obiektów gry.
5. Odświeżany jest widok gry.

Punkty 4 i 5 wykonywane są bez względu na to, czy użytkownik wygenerował jakieś nowe zdarzenie, czy nie. Jeśli pojazd się poruszał, to system fizyki musi obliczyć jego nowe położenie, a system grafiki odpowiednio przygotować obraz, który zostanie wyświetlony.

Na pierwszy rzut oka rozszerzenie tego schematu jest bardzo proste. Wystarczy przecież tylko dodać punkt:

- 3a. Program wysyła innym swój stan oraz odbiera stany gry innych graczy.

Niestety takie rozwiązanie ma wiele wad, które będziemy rozważać w kolejnych punktach.

3. Gracze oszukują

Ludzie zawsze próbowali oszukiwać i będą to robić aż po wsze czasy. Gracz, który oszukuje grę lokalną (zmieniając plik wykonywalny, pliki danych gry, a nawet zmieniając zawartość obszaru pamięci, w którym znajduje się gra) oszukuje tylko siebie. Uczestnik rozgrywki sieciowej oszukuje innych graczy. Zepsucie innym zabawy to jeszcze niewielka strata. Przypomnijmy sobie jednak, jak kończyła się porażka w filmie „Tron”. Gdyby grę rozszerzyć o istnienie świata, w którym gracze mogą „tracić życie”, to niewielkie oszustwo może mieć dużo dalej idące konsekwencje. Aby nie być gołosłownym to wspomnę grę „Diablo”, w której gracze sprzedawali swoje postacie za ciężkie pieniądze (prawdziwe, nie wirtualne!!).

Uczestnik gry sieciowej ma ułatwione zadanie. Nie musi grzebać w plikach binarnych, wystarczy dokładna obserwacja danych przesyłanych przez interfejs sieciowy, a następnie podmienianie ich na takie, które zagwarantują zwycięstwo.

Skoro stan gry „Tron” można opisać zaledwie kilkoma współczynnikami, to kuszącym rozwiązaniem byłoby przesyłanie pozycji pojazdu do pozostałych graczy. Konsekwencje tego będą jednak tragiczne. Nieuczciwy zawodnik może napisać sobie prosty skrypt, który będzie analizował przychodzące pozycje pozostałych pojazdów, a następnie obliczał sobie pozycję, w której należy się ustawić, aby spowodować kolizję.

4. Dobrze jest ufać, lepiej kontrolować

Jednym z rozwiązań jest zastosowanie protokołu typu „zamówienie – odpowiedź”. Klient, przed wykonaniem każdej akcji, wysyła do serwera zamówienie na wykonanie pewnej akcji. Serwer sprawdza poprawność otrzymanych danych i możliwość wykonania danej akcji, a następnie odsyła żądającemu potwierdzenie. Jednocześnie zmienia u siebie stan gry zgodnie z żadaną akcją i rozsyła pozostałym klientom informację o zmianie stanu gry.

Niestety to rozwiązanie jest nie do przyjęcia w grze „Tron”. Jak przekonamy się później, bardzo trudno jest utrzymywać spójny stan gry na wszystkich komputerach w sytuacji, kiedy pojazdy poruszają się z tak dużymi prędkościami. Dodatkowy etap potwierdzania może nałożyć dodatkowe opóźnienia w synchronizacji, a w efekcie powodować „skoki” w obrazie generowanym przez program gry.

Zamiast więc stosować się do maksymy Józefa Stalina, zmodyfikujemy ją tak, by dla oszusta protokół wyglądał jak „zamówienie – odpowiedź”, a dla uczciwego gracza jak protokół informowania serwera o wykonaniu akcji. W tabeli 1 zestawilem kroki, jakie podejmują uczciwe strony podczas gry.

klient	serwer
Wysyła aktualną pozycję swojego pojazdu wraz z danymi opisującymi dynamikę pojazdu.	
Wysłane dane przyjmuje jako wstępnie poprawne i kontynuuje działanie tak samo, jak ma to miejsce w grze lokalnej. Symuluje przy tym zachowanie swojego samochodu oraz zachowanie samochodów rywali.	<p>Dokonuje weryfikacji poprawności danych, tj:</p> <ol style="list-style-type: none"> 1. sprawdza możliwość zmiany dynamiki pojazdu, 2. porównuje przysłaną pozycję z pozycją, którą sam miał obliczoną na daną chwilę. <p>Jeśli weryfikacja przeszła pomyślnie, to odsyła klientowi potwierdzenie chwili wraz z pozycjami pozostałych samochodów.</p>
Porównuje przysłane pozycje rywali ze swoimi i koryguje w miarę potrzeb.	

Tabela 1: Protokół wymiany pozycji

A co zrobić z oszustem? Józef Stalin kazałby pewnie rozstrzelać, albo wysłać na roboty

na Syberię. My jednak musimy być ostrożni. Możliwe jest, że dane przychodzące od klienta będą docierały z dużymi opóźnieniami. Serwer powinien potrafić ocenić odchylenie żądanej akcji od poprawnej. Jeśli odległość przekracza pewną ustaloną wartość, klient powinien zostać wyeliminowany z dalszej gry, a kolejne jego żądania ignorowane. Jeśli jednak żądanie mieści się w granicy, to serwer powinien odesłać niezbędną korektę.

5. Serwer i klient

Do tej pory dość często posługiwałem się pojęciami serwera i klienta, nie precyzując dokładnie, o co mi chodzi. Popularnym podejściem do tego zagadnienia jest przyjęcie, że jeden z komputerów uczestniczących w grze pełni rolę serwera (jest to tzw *player hosted client – server*). Ta jedna instancja gry musi umożliwiać innym klientom nawiązanie połączenia i rozsyłać odpowiednie pakiety do wszystkich uczestników rozgrywki. Takie podejście jest dosyć intuicyjne dla gracza, a przy okazji daje się łatwo zaimplementować. Ma jednak swoje wady. Należy pamiętać, że osoba, która uruchamia usługę serwera musi sobie zdawać sprawę ze zwiększonego obciążenia sieciowego.

Ponieważ kontrola poprawności ruchów odbywa się na serwerze, to rozwiązanie to pociąga za sobą niebezpieczeństwo malwersacji ze strony osoby prowadzącej usługę serwera. Jeśli wyniki gry miałyby mieć duże znaczenie dla graczy, to znacznie lepszym pomysłem jest uruchomienie dedykowanego serwera.

6. Dylematy spowodowane niewiedzą

Jest jeszcze jeden poważny problem do rozważenia. Co robią przeciwnicy, kiedy nie dostajemy informacji o ich pozycji? Gra nie może zmieniać pozycji pojazdu innego gracza tylko w tych chwilach, kiedy otrzymuje odpowiednie dane. Powodowałoby to „skakanie” tych pojazdów z miejsca na miejsce.

Na nasze szczęście z problemem tym uporała się już amerykańska armia, która opracowała odpowiednie protokoły do symulacji bitwy. Technikę likwidowania skutków opóźnień związanych z przesyłaniem danych przez sieć nazwano „dead reckoning” i opracowano w ramach projektu DARPA Advanced Distribution Simulation (w skrócie DIS).

Program gry symuluje zachowanie wszystkich pojazdów (łącznie ze swoim). Jeśli znane są prędkość i przyspieszenie pojazdu, to można określać jego kolejne położenia w kolejnych chwilach gry. W przypadku własnego pojazdu należy jednak za każdym razem, czy właściwa pozycja nie różni się zbyt od tej symulowanej. Zakłada się przy tym pewną granicę tolerancji. Jeśli pojazd przekroczył granicę tolerancji, to wysyła się poprawkę pozycji.

7. Wybór odpowiedniej warstwy transportowej

Podobno nie ma gier sieciowych, które korzystałyby z protokołu TCP. Wynika to głównie z tego, że w grach nie trzeba się martwić o dostarczenie każdego datagramu. Tym bardziej, że retransmitowany pakiet jest już bezużyteczny.

Literatura

[1] Aronson J.: *Dead Reckoning: Latency Hiding for Networked Games*. 1997.

- [2] Pritchard M.: *How to Hurt the Hackers*. 2000.
- [3] Kirmse A., Kirmse C.: *Security in Online Games*. 1997.
- [4] Yu-Shen Ng: *Designing Fast-Action Games For The Internet*. 1997.
- [5] Coulouris G., Dollimore J., Kindberg T.: *Systemy rozproszone podstawy i projektowanie*. WNT, Warszawa 1998.
- [6] Comer D.: *Sieci komputerowe TCP/IP*. WNT, Warszawa 1998.
- [7] Mills D.: *Internet delay experiments*. RFC 889, 1983.
- [8] Postel J.: *Transmission Control Protocol*. RFC 793, 1981.