Dyna Blaster

Marcin Witkowski Łukasz Witkowski

6 lutego 2009

1 Zadanie

Naszym zadaniem była implementacja inteligentnego gracza, dla gry „Dyna Blaster". Należało zaimplementować dwa interfejsy ICPlayerFactory, IPlayerFactory, komunikujące się z serwerem gry odpowiednio za pomocą CORBY oraz gniazd (socketów).

2 Implementacja

Interfejsy są implementowane przez klasy, odpowiednio :

- ICPlayerFactory - StubAIPlayerController
- IPlayerFactory - PlayerServant

3 Algorytm sztucznej inteligencji

W tworzeniu algorytmu wykorzystano idee zawarte w [2] oraz [1]. Algorytm możemy podzielić na trzy główne części : obliczanie statystyk graczy (zasięg bomb) i pól (czas do wybuchu bomby), wyszukiwanie bezpiecznej ścieżki, oraz wybór ruchu.

3.1 Obliczanie statystyk

Dla każdej przychodzącej ramki przeglądamy wszystkie pola na planszy w celu odnalezienia zmian, pomiędzy stanem pamiętanym a otrzymanym. Chodzi głównie o zdarzenia pojawienia się na planszy bomby, bonusu, czy płomieni, oraz analogiczne zdarzenie zniknięcia wymienionych rzeczy. Przetworzone dane przekazywane są algorytmowi sztucznej inteligencji w celu dalszej analizy.

3.2 Wyszukiwanie ścieżki

Realizowane jest za pomocą przeszukiwania BFS po planszy. Na wejściu klasa otrzymuje planszę z obliczonymi statystykami i przetwarza ją symulując jej wygląd w przyszłości, oznaczając przy tym odpowiednie pola jako niedostępne lub dostępne w danym momencie czasu. Następnie wywoływane jest przeszukiwanie wszerz po polach planszy. Każdy stan modelowany jako para (pole,czas ramki) i dodawany do kolejki, czasy przejść między polami obliczane są dokładnie co do ramki zgodnie z metryką taksówkową (nasz zawodnik nie ma „ścinać" zakrętów). Jak widać nie jest to przeszukiwanie pełne, ścieżki zostają kumulowane w jedną gdy osiągają dane pole w tym samym czasie.
Przeszukiwane jest zakończone po znalezieniu szukanego elementu (bonus, przeciwnik) lub osiągnięciu momentu czasu w którym wszystkie bomby na planszy wybuchną (wyszukiwanie ścieżki która pozwala przeżyć).

3.3 Wybór ruchu

W centralnym algorytmie wyboru ruchu wzorowano się na modelu warstwowym z [2]. Główny priorytet przypisany jest zdarzeniu postawienia bomby, algorytm wywołuje wtedy wyszukiwanie bezpiecznej ścieżki z parametrem „null" jako pierwsza wartość (zatrzymujemy gracza na jedną ramkę aby miał czas na postawienie bomby). Bomba jest stawiana gdy znajdujemy się w otoczeniu pola typu „CRATE" lub przeciwnika w zasięgu 2 kratek w pionie lub poziomie. Jeżeli nie znajdziemy bezpiecznej ścieżki wywołujemy szukanie bezpiecznej ścieżki nie biorącej pod uwagę stawiania bomby w danym miejscu.
Gdy nie chcemy postawić bomby staramy się wyszukać ścieżki do najbliższego przeciwnika, bonusu lub ściany. Znalezienie bonusu lub przeciwnika przerywa działanie BFS-a (zawsze więc idziemy do najbliższego obiektu), do ściany idziemy tylko wtedy gdy nie mamy nic ciekawszego do zrobienia. Wyszukana ścieżka jest następnie sprawdzana w celu weryfikacji czy jest bezpieczna. Jeżeli nie jest bezpieczną (nie możemy znaleźć bezpiecznej ścieżki o pierwszym kroku równym zaproponowanemu wcześniej) szukamy bezpiecznej ścieżki, lub zwracamy "null" (gdy pole na którym stoimy nie grozi nam utratą życia w przyszłości).

4 Wyniki

Powyżej opisana strategia wpływa na to, że nasz gracz jest bardzo agresywny w swoich działaniach i dąży do konfrontacji. Niestety każda utrata ramek ma duży wpływ na algorytm i często prowadzi do śmierci gracza.
Za dobrze obraną strategią sztucznej inteligencji naszego bota przemawiają najlepsze wyniki względem ilości „fragów" podczas masowych gier, oraz

drugie miejsce w konkursie indywidualnym podczas przeprowadzonego na wykładzie konkursu. Co ciekawe w finałowej rozgrywce nasz zawodnik raz zabił przeciwnika a sam zginął trzykrotnie od własnej bomby (wynik we „fragach" 1-0), przegrał więc nie dzięki sprytowi oponenta tylko błędom w obliczeniach przy stawianiu własnych bomb.

Literatura

[1] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[2] Samuel H Kenyon. Behavioral software agents for real-time games. *IEEE Potentials*, (25), 2006.