Dokumentacja do projektu AgentChess

Michał Bobowski, Zofia Abramowska, Jakub Meller

16 kwietnia 2014

1 Założenia wstępne

Niniejszy projekt jest realizowany w ramach przedmiotu SAG. Jego przedmiotem jest stworzenie prostego, bazującego na agentowym podejściu, modelu sztucznej inteligencji służącej do gry w szachy. Silnik gry i warstwa graficzna zostaną oparte na istniejących rozwiązaniach. Całość zostanie zrealizowana w języku programowania Scala.

1.1 Figury

Dla uporządkowania wiedzy dziedzinowej posługujemy się nazwami figur w wesji polskiej i angielskiej. Są to kolejno:

- Pion (ang. Pawn) - porusza się o jedno pole naprzód (ew. o dwa pola na początku), bije tylko po skosie.
- Skoczek (ang. Knight) - porusza się systemem dwa pola naprzód i jedno do boku w dowolnym kierunku.
- Goniec (ang. Bishop) - porusza się po skosie.
- Wieża (ang. Rook) - porusza się w pionie lub poziomie.
- Hetman (ang. Queen) - łączy ruchy wieży i gońca.
- Król (ang. King) - porusza się o jedno pole w dowolnym kierunku.

Figury są identyfikowane wewnątrz programu przy pomocy małej (białe) bądź wielkiej (czarne) litery. Dla większości figur identyfikatorem jest pierwsza litera angielskiej nazwy. Wyjątek stanowi skoczek, który posługuje się drugą literą (n).

2 Opis algorytmu

Główną cechą niniejszego algorytmu jest jego zorientowanie na agentowość. Bezcelowe jest porównywanie go do algorytmów budujących drzewo rozwiązań, gdyż na pewno są one wydajniejsze. Sukcesem projektu będzie stworzenie grywalnego, działającego w rozsądnym czasie programu.

2.1 Architektura agentów

Agenci są umieszczeni w architekturze wertykalnej. Głównym mózgiem przedsięwzięcia jest super-agent, którego zdaniem jest wysyłanie żądań do agentów podrzędnych i przeprowadzanie wnioskowania dotyczącego wyboru ruchu.

Na niższym poziomie w hierarchii znajdują się agenci reprezentujący figury obsługiwane przez komputer. Można rozważyć również stworzenie agentów dla figur obsługiwanych przez gracza, ale nie wydaje się to w tym momencie zasadne. W kolejnych częściach opisany jest szczegółowy protokół komunikacji między agentami.

2.2 Struktury pomocnicze algorytmu

W tej sekcji zawarty jest opis pewnych abstrakcyjnych bytów, na podstawie których przeprowadzane jest wnioskowanie.

2.2.1 Ocena heurystyczna

Każda z figur biorących udział w grze posiada heurystyczną ocenę swojej przydatności, oznaczaną dalej jako H. Nie czuję się specjalistą, ale wikipedia proponuje następującą wycenę:

- H=1 dla piona
- H=3 dla skoczka i gońca
- H=5 dla wieży
- H=9 dla hetmana

Do kompletu dodany zostaje król z wyceną równą np. H=20.

2.2.2 Mapa ruchu wroga

Mapa ruchu wroga ma za zadanie pokazać w łatwo dostępnej formie, gdzie i jakimi figurami może ruszyć się przeciwnik w pewnej konkretnej sytuacji na planszy. Zakładam, że bicie jest traktowane w tym przypadku jako normalny ruch. Od strony programistycznej mapa ruchu wroga jest mapą, w której kluczem jest pole planszy, a wartością lista identyfikatorów figur, które mogą wykonać ruch na dane pole.

2.2.3 Mapa wsparcia wroga

Wprowadzam relację wsparcia zdefiniowaną w następujący sposób: figura A wspiera figurę B, jeśli w przypadku zbicia B, A może w kolejnym ruchu dokonać bicia na polu zajmowanym wcześniej przez B. Mapa wsparcia ma za zadanie zagregować informacje dotyczące wzajemnego położenia figur wroga. Kluczem mapy jest identyfikator figury wspieranej, a wartością lista identyfikatorów figur wspierających.

2.3 Faza pierwsza - odpytanie o ruchy

Algorytm rozpoczyna super-agent, który zadaje każdemu agentowi podrzędnemu pytanie o możliwe ruchy. Odpowiedzią na żądanie jest lista ruchów, które mogą być wykonane zgodnie z zasadami gry. W tym przypadku bicie jest również traktowane jako normalny ruch.

Struktura danych reprezentująca ruch powinna zawierać następujące dane:

- Identyfikator figury
- Pole początkowe (source)
- Pole końcowe (destination)
- Wstępną ocenę heurystyczną ruchu

Wstępna ocena heurystyczna ma domyślnie wartość 0. Jedynie w przypadku bicia przyjmuje wartość równą wartości ofiary.

2.4 Faza druga - ocena ruchów

Każdy ruch podlega ocenie przez wszystkich agentów, wliczając w to agenta zmieniającego położenie. Faza oceny pojedynczego ruchu przebiega według następującego schematu:

1. Super-agent odtwarza stan planszy po wykonaniu ruchu

2. Super-agent buduje dla nowego stanu mapę ruchu wroga i mapę wsparcia wroga

3. Super-agent wysyła do agentów podrzędnych pytanie o subiektywną ocenę swojego położenia

Agenci podrzędni wyrażają swoją ocenę sytuacji w sposób egoistyczny tzn. nie interesują się pozostałymi agentami. Ocena sytuacji jest wyrażana liczbowo i jest sumą dwóch składowych: ofensywnej i defensywnej.

Składowa ofensywna jest równa najwyższej ocenie figury wroga, znajdującej się w zasięgu rażenia. Jeżeli figura wroga jest wspierana przez jakąkolwiek inną figurę, to składową ofensywną pomniejszamy o wartość naszej figury.

Składowa defensywna jest równa ujemnej wartości naszej figury, w przypadku gdy może ona zostać zbita (wykorzystujemy mapę ruchu wroga). Jeśli zagrożenie nie występuje, to składowa defensywna ma wartość 0.

2.5 Faza trzecia - wybór ruchu

Po skompletowaniu ocen cząstkowych, super-agent podejmuje decyzję o wyborze ruchu. Całkowita ocena ruchu jest sumą ocen z faz pierwszej i drugiej. Wybrany zostaje ruch z najwyższą oceną sumaryczną lub jeden z najlepszych ruchów.

2.6 Opcjonalne modyfikacje

Trudno na etapie planowania ocenić całkowity czas oczekiwania na ruch, ale na pewno nie powinien on przekraczać minuty. Jeśli w podstawowym wariancie nie będzie to możliwe, należy ograniczyć liczbę ruchów, podlegających ocenie w fazie drugiej. Na przykład po fazie pierwszej można pozostawić 5 najlepszych i 5 losowych ruchów.

Drugą opcjonalną modyfikacją jest dodanie do oceny w fazie trzeciej, średniej z ocen składowych wszystkich agentów. Może to spowodować poprawę działania algorytmu, ale równie dobrze może coś popsuć.

3 Uwagi techniczne - przeczytaj zanim zaczniesz kodować

Chciałbym zwrócić w tej sekcji uwagę na kilka ważnych problemów z którymi miałem do czynienia. Po pierwsze - istnieje duża różnica między bibliotekami Scala Actors i Scala Akka, choć na pierwszy rzut oka wyglądają

tak samo. My korzystamy TYLKO z biblioteki Scala Akka, która od wersji języka 2.11 zastąpi Scala Actors. Ponieważ najnowszym wydaniem stabilnym języka Scala jest wersja 2.10 to musiałem ręcznie dodać sporo plików jar do java build path (cały folder /lib).