Dokumentacja do projektu AgentChess

Michał Bobowski, Zofia Abramowska, Jakub Meller

23 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.

Pion po dojściu do końca planszy może zamienić się w inną figurę. Nie obsługujemy od strony AI roszady i bicia w przelocie (chyba, że ktoś ma ogromną potrzebę to może dopisać).

2 Uwagi techniczne - przeczytaj zanim zaczniesz kodować

Chciałbym zwrócić w tej sekcji uwagę na kilka 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 w repozytorium).

Korzystałem z IDE Eclipse z wtyczką do języka Scala. Nie mogę powiedzieć, żeby ten zestaw działał oszałamiająco dobrze, ale na potrzeby tego projektu wystarcza. Trzeba się tylko przyzwyczaić, że czasem pokazuje błędy tam gdzie ich nie ma i dopiero *Project/Clean* załatwia sprawę.

Długo zastanawiałem się jak najlepiej wykorzystać istniejące szachy z książki Grzegorza Balcerka. Architektura, którą przyjął autor nie była dla nas z wielu względów korzystna np. logika ruchu poszczególnych figur nie znajdowała się bezpośrednio w klasach tych figur. Z drugiej strony głupio było od nowa pisać interfejs graficzny. Ostatecznie zdecydowałem się na użycie podstawowych klas (Figure, Field, Game), ale logikę ruchu napisałem jeszcze raz w klasach agentów. W ten sposób ruchy gracza są wykonywane tak jak w oryginale.

Ciekawe jest to, że gra w szachy wcale nie kończy się zbiciem króla. Ruchy, które pozostawiają własnego króla w sytuacji zagrożenia, nie są w ogóle akceptowane jako poprawne. Dlatego nasze AI ogłasza przegraną, kiedy nie może wykonać żadnego poprawnego ruchu.

3 Architektura agentów

Agenci są umieszczeni w architekturze wertykalnej. Najważniejsze decyzje podejmowane są przez super-agenta, którego zdaniem jest wysyłanie żądań do agentów podrzędnych.

Na niższym poziomie w hierarchii znajdują się agenci reprezentujący figury obsługiwane przez nasze AI. Można rozważyć również stworzenie agentów dla figur obsługiwanych przez gracza, ale nie wydaje się to w tym momencie zasadne.

W trakcie kodowania okazało się, że potrzebna jest jeszcze jedna warstwa, łącząca super-agenta z dotychczasowym szkieletem gry. Jest nią klasa *Listener*, będąca klasą wewnętrzną *ChessGui*.

4 Struktury pomocnicze algorytmu

W tej sekcji zawarty jest opis pewnych abstrakcyjnych bytów, biorących udział we wnioskowaniu.

4.1 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.

4.2 Mapa bicia wroga

Mapa bicia wroga pokazuje, które figury AI mogą zostać zbite w następnym wykonywanym przez gracza ruchu. Kluczem mapy jest pole planszy, a wartością zmienna boolowska, która jest równa *true* wtedy i tylko wtedy gdy:

1. Pole jest zajmowane przez figurę należącą do AI.
2. Którakolwiek z figur gracza może w następnym ruchu zbić figurę z punktu 1.

4.3 Mapa wsparcia wroga

Wprowadzam relację wsparcia zdefiniowaną w następujący sposób: figura A wspiera figurę B, jeśli w przypadku zbicia B, A może w następnym ruchu tego samego gracza, dokonać bicia na polu zajmowanym wcześniej przez B. Kluczem mapy jest pole planszy, a wartością zmienna boolowska, która jest równa *true* wtedy i tylko wtedy gdy:

1. Pole jest zajmowane przez figurę należącą do gracza.
2. Którakolwiek z pozostałych figur gracza wspiera figurę z punktu 1.

5 Aktualizacja stanu

Na początku gry tworzeni są agenci odpowiadający za każdą z figur AI. Życie agentów kończy się w bliżej nieokreślonej przyszłości, po tym jak zostaną oni zbici przez gracza. Z tego powodu, działanie algorytmu rozpoczyna się od sprawdzenia, czy poprzedni ruch gracza nie zakończył się na polu zajmowanym przez jedną z figur AI. Odpytanie jest wykonywane przy pomocy komunikatów. W przypadku stwierdzenia swojego zgonu, agent figury przesyła stosowny komunikat do super-agenta i kończy swoje działanie.

6 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. Bicie jest traktowane jako normalny ruch.

Struktura danych reprezentująca ruch zawiera następujące dane:

- 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.

7 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 wypadkową dwóch składowych: ofensywnej i defensywnej.

Jeżeli nasza figura jest w danej sytuacji zagrożona, to ocena jest równa ujemnej wartości H naszej figury. W przeciwnym przypadku ocena jest równa najwyższej ocenie figury wroga, znajdującej się w zasięgu rażenia naszej figury. Jeżeli figura wroga jest wspierana, to składową ofensywną pomniejszamy o wartość naszej figury.

Końcowa ocena ruchu jest sumą następujących składowych:

1. Oceny z fazy pierwszej
2. Najniższej z ocen agentów
3. Średniej ze wszystkich ocen agentów (być może przemnożonej przez jakąś stałą np. 0.5)

8 Faza trzecia - wybór ruchu

Po skompletowaniu ocen cząstkowych, super-agent przesyła do obiektu nasłuchującego, posortowaną względem sumarycznej oceny listę ruchów. Następnie wykonywany jest najlepszy z ruchów, który nie pozostawia króla w pozycji zagrożenia. Jeżeli żaden z ruchów nie spełni tego warunku, to AI ogłosi swoją porażkę.

Po wyborze ruchu następuje jego ogłoszenie do wiadomości wszystkich agentów. Agent, który jest źródłem ruchu dokonuje aktualizacji swojego położenia.