Tomasz Murawski
Szymon Szymborski

Dokumentacja projektu gry „Czwórki”

1. Ogólne informacje o projekcie

Projekt implementuje funkcyjną postać aplikacji do gry w czwórki w standardowej wersji z wykorzystaniem algorytmu min-max z obcinaniem alfa-beta. Aplikacja umożliwia grę gracza z algorytmem sztucznej inteligencji. Aplikacja posiada uproszczony interfejs konsolowego wejścia-wyjścia.

2. Przyjęte założenia

Projekt napisany został w języku Scala za pomocą implementacji funkcyjnej. Na projekt składają się trzy pliki źródłowe:

- Game.scala
- AI.scala
- Gui.scala

3. Opis rozwiązania

Zadany problem algorytmu grającego w grę w „czwórki” rozwiązaliśmy przy pomocy algorytmu min-max z obcinaniem alfa-beta.

Za realizację wyżej wymienionego algorytmu odpowiadają funkcje klasy AI, będącej programową realizacją sztucznej inteligencji za pomocą języka funkcyjne-

go. W momencie, gdy w grze następuje ruch gracza będącego sztuczną inteligencją, wywoływana jest funkcja *makeMove()*, która dla każdego możliwego kolejnego ruchu wywołuje funkcję *alphabeta()*, a na koniec zwraca wybraną przez program kolumnę, do której chciałaby wrzucić swój krążek.

Wybór odpowiedniej kolumny realizowany jest przez funkcję *alphabeta()*, która wywołuje się rekurencyjnie dla d kolejnych ruchów (gdzie d – głębokość drzewa) przyjmując jako parametry między innymi współczynniki alfa i beta, będące odpowiednio maksymalnym wynikiem gracza AI oraz maksymalnym wynikiem jego przeciwnika. Algorytm bierze pod uwagę wszystkie możliwości, lecz również odrzuca te, dla których współczynnik alfa jest większy lub równy współczynnikowi beta. W przypadku ruchu gracza AI funkcja zwraca maksymalną wartość spośród wartości wszystkich jego możliwych ruchów (jego najlepszy ruch), w przypadku przeciwnika minimum spośród wszystkich możliwych ruchów (ruch najlepszy dla przeciwnika, najgorszy dla AI), a w razie dojścia do zadanej głębokości d lub końca gry – wartość danego węzła określaną przez funkcję *evaluate()*.

Funkcja *evaluate()* oblicza liczbową „wartość" danego ułożenia krążków na planszy z perspektywy gracza o kolorze podanym jako argument wywołania funkcji. Postanowiliśmy, że przyporządkujemy każdej liczbie od 1 do 4 sąsiadujących krążków tego samego koloru pewną wartość liczbową. W przypadku krążków w kolorze badanego koloru dla czwórek, trójek, dwójek i jedynek są to odpowiednio wartości: 1000/100/10/1, a w przypadku krążków przeciwnika: -10000/-1000/-10/-1.

Funkcja przegląda wszystkie możliwe czwórki pól na planszy, sąsiadujące pionowo, poziomo lub na skos, w których znajdują się krążki nieblokowane przez krążki drugiego gracza (czyli takie czwórki pól, które mają szansę być zapełnione przez krążki tego samego koloru, tj. dać graczowi wygraną w przyszłości). Dla każdej znalezionej czwórki spełniającej wymaganie, funkcja dodaje do łącznej sumy jej wartość.

Dodatkowo, jeśli w pierwszych poziomach drzewa zostanie wykryte zwycięstwo któregoś z graczy, łącza suma zostaje zwiększona (bądź zmniejszona) o zwie-

lokrotnioną wartość zwycięskiej czwórki krążków, by lepiej zapobiegać przegranej lub agresywniej dążyć do wygranej.

4. Raport z testów

Przeprowadziliśmy serię testów rozgrywek za pomocą testowania manualnego z wykorzystaniem mniej i bardziej doświadczonych testerów. Test polegały na grze z algorytmem z wykorzystaniem różnych głębokościach drzew, zaczynając od zaledwie jednopoziomowego drzewa, kończąc na drzewach o poziomach w których algorytm miał pewność wygranej od pierwszego ruchu.

W przypadku głębokości drzewa $d = 1$ wygranie z komputerem nie stanowiło większego problemu, gdyż algorytm rozważał jedynie swój następny ruch. Przy zwiększeniu głębokości drzewa do $d = 2$ komputer był w stanie przewidywać następny ruch przeciwnika, a co za tym idzie blokować oczywiste wygrane, co znacznie podniosło poziom trudności.

Podczas zwiększania głębokości drzewa zaobserwować można było nie tylko uporczywe blokowanie ruchów gracza, ale także coraz większe przemyślenie kolejnych stawianych krążków i dążenie do nieoczywistego zwycięstwa przez sztuczną inteligencję.

Po kilkudziesięciu grach o $d = 7$ udało nam się wygrać rozgrywkę, jednak tylko dzięki doprowadzeniu do sytuacji, w której została jedna niezapełniona kolumna, więc AI było zmuszone podstawić się pod nasz wygrywający ruch.

Wraz ze zwiększaniem głębokości drzewa, od wartości $d = 7$ wyraźnie można zauważyć coraz powolniejsze ruchy sztucznej inteligencji. Próby dalszego zwiększania głębokości drzewa są jednak bezcelowe z powodu pewnego zwycięstwa komputera.