Margarita Chirillova

Nr albumu 18017

Connect 4

Opis

Connect 4 to prosta gra, polega na połączeniu kamyków w taki sposób, by kamyki jednego koloru połączyły się w linię z 4 sztuk pionowo bądź poziomo. Kto pierwszy osiągnie taki wynik - wygrywa. W naszym przypadku 1 graczem jest komputer a drugim - człowiek.

Wymagania systemowe

Java must be installed, other than that, there should not be any problem running this game.

Detale implementacji

Implementacja

Struktura mvc była wybrana przez prostotę, mimo to nie pokazuje kompletnie możliwości mvc, bo projekt uczelniany jest bardzo prosty.

Gui jest maksymalnie proste rysowane znakami w oknie terminalu.

Największe skupienie realizacji gry poświęcone było realizacji algorytmu MiniMax. Wykorzystane zostały obie wersje algorytmu - podstawowa redukujący alpha-beta.

Minimax to algorytm którego celem jest minimalizacja strat. Moim zdaniem idealnie pasuje dla koncepcji wybranej gry. Ułatwia on podejmowanie decyzji w przypadku niepewności, więc pasuje nie tylko do gier, ale również może mieć zastosowanie w życiu realnym dla ludzi, które mają problem przy podejmowaniu decyzji - np ja.

Zalegałam z napisaniem tej dokumentacji ze względu właśnie na trudność w podejmowaniu decyzji która ruinuje życie bo skutkiem nie podjęcia decyzji może być zaleganie z pracą domową bądź oblania innych spraw życiowych. Narzędzie które pomaga dokonać wyboru w 21 wieku jest bezcenne. Dla tego np rozwiązania mądrych reklam kontekstowych jest dobrą drogą rozwoju internetu.

Wracając do projektu: zaimplementowane zostały klasy Gracza, oraz dziedziczące po nich klasy “człowieka” oraz “komputera”. Zostały również zaimplementowane klasy dwóch wersji algorytmów Minimax gracz oraz Aplha-Beta gracz.
Wszystkie gracze “spotykają się” w klasie planszy gdzie odbywa się gra.
Posiadając funkcję oceniającą wartość stanu gry w dowolnym momencie czyli gracz min chce ten stan zminimalizować, a gracz max zmaksymalizować. Obliczane jest drzewo wszystkich możliwych stanów w grze do pewnej głębokości, ograniczonej zazwyczaj przez naszą moc obliczeniową, zakładamy że rozgałęzienie drzewa stanów jest stałe i wynosi x - czyli na każdy ruch można odpowiedzieć x innych ruchów, a głębokość y - tyle ruchów do przodu symulujemy algorytmem minimax, to mamy stanów końcowych, dla których obliczamy wartość stanu gry naszą funkcją.

Zaczynamy przeglądanie od stanów końcowych, symulując optymalne wybory dla obu graczy, tak aby na głębokość y w liściach drzewa była dla nich optymalna liczba - stan gry po wykonaniu x ruchów. Tak więc gracz min zawsze wybiera ruch, który prowadzi do mniejszej wartości końcowej, a gracz max – przeciwnie. Po przeprowadzeniu tej symulacji gracz, który znajduje się w korzeniu drzewa - aktualnie wykonujący ruch, ma pewność, że jego ruch jest optymalny w kontekście informacji o stanie gry z przeprowadzonej symulacji algorytmem minimax na głębokość y.

Natomiast **algorytm alpha-beta** jest w pewnym sensie rozszerzeniem i usprawnieniem algorytmu minimax.
Usprawnienie to polega na tym, że niektóre gałęzie drzewa przeszukiwania mogą zostać odcięte. Czas przeszukiwania ograniczony zostaje do przeszukania najbardziej obiecujących poddrzew, w związku z czym możemy zejść głębiej w tym samym czasie. Współczynnik rozgałęzienia jest dwukrotnie mniejszy niż w metodzie min-max. Algorytm staje się wydajniejszy, gdy węzły rozwiązywane są układane w porządku optymalnym lub jemu bliskim.

Jeśli porządek wykonywania ruchów jest optymalny, czyli najlepsze ruchy są przeszukiwane jako pierwsze, liczba przeszukiwanych pozycji wyniesie O(b*1*b*1*...*b) dla nieparzystej głębokości i odpowiednio O (b*1*b*1*...*1), gdy głębokość będzie parzysta. W późniejszych przypadkach efektywny współczynnik rozgałęzienia jest redukowany do pierwiastka lub przeszukiwanie może odbywać się dwukrotnie głębie. b*1*b*1*... bierze się stąd, że wszystkie pierwsze ruchy gracza muszą zostać sprawdzone w celu znalezienia ruchu najlepszego, ale dla każdego kolejnego tylko najlepszy ruch gracza jest potrzebny, aby odrzucić wszystkie ruchy poza pierwszym, najlepszym ruchem – alfa-beta dba o to, że żaden inny ruch drugiego gracza nie musi być brany pod uwagę.
Normalnie w trakcie wykonywania algorytmu alfa-beta poddrzewa są tymczasowo zdominowane przez przewagę pierwszego gracza lub vice versa. Ta przewaga może się wiele razy powtórzyć w trakcie poszukiwań, jeśli porządek ruchów jest niewłaściwy – za każdym razem prowadzi do marnotrawstwa. Jako że liczba pozycji zmniejsza się wykładniczo dla każdego ruchu początkowego, warto zastanowić się nad sortowaniem pierwszych ruchów. Zastosowanie

sortowania na każdej głębokości wykładniczo zredukuje liczbę przeszukiwanych pozycji, ale sortowanie wszystkich pozycji na głębokości bliższej korzeniowi jest relatywnie tańsze z powodu ich niewielkiej liczby. W praktyce porządkowanie ruchów jest określane przez wyniki wcześniejszych mniejszych poszukiwań, np iteracyjnych.
Algorytm utrzymuje dwie wartości alfa i beta, które reprezentują minimalny wynik gracza MAX i maksymalny wynik gracza MIN. Początkowo alfa jest -nieskończonością, a beta +nieskończonością. W miarę postępowania rekursji przedział (alfa; beta) staje się mniejszy i kiedy beta staje się mniejsze niż alfa, oznacza to, że obecna pozycja nie może być wynikiem najlepszej gry przez obu graczy i wskutek tego nie ma potrzeby przeszukiwania głębiej.