Projektowanie algorytmów i metody sztucznej inteligencji

Projekt 3-gra

Sebastian Morta 241593

Prowadzący- Dr inż. Łukasz Jeleń

Termin-środa, 11:15

1 Wstęp

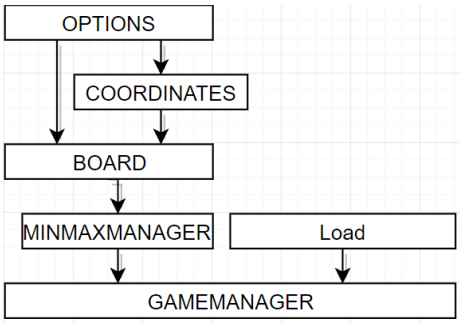
Algorytm MinMax wywodzi się z twierdzenia o grze o sumie stałej. Oznacza to, że jeśli dwie osoby grają przeciwko sobie, to poprawa sytuacji jednego z graczy oznacza proporcjonalne pogorszenie się sytuacji gracza drugiego. Dzięki tej informacji można przy użyciu odpowiedniej funkcji
heurystycznej przypisać danej sytuacji na planszy konkretną wartość (dodatnią, jeśli jest korzystna dla gracza, dla którego ją rozpatrujemy lub ujemną w przeciwnym wypadku). Cięcia alfa i beta pozwalają lepiej wykorzystać moc obliczeniową w celu umożliwienia głębokiego zbadania drzewa gry tam gdzie może to wpłynąć na zmianę wybieranego ruchu, kosztem rezygnacji z rozbudowy i analizy drzewa tam gdzie jej wynik i tak nie wpłynie na wybierany ruch.

2 Opis stworzonej gry

2.1 Opis programu

W kółko i krzyżyk (tic-tac-toe), w podstawowej wersji gra się na planszy 3 na 3 pola i do wygrania potrzebne są 3 takie same znaku ustawione w jednej linii. W założeniu projektowym wymagana była możliwość niezależnego dostosowania wielkości planszy i ilości znaków potrzebnych do odniesienia zwycięstwa. Mój program posiada obsługę gry dla wielkości planszy od 3x3 do 10x10 i odpowiednio ilość pól w rzędzie również może mieć wielkość od 3 do 10. Każde pole może mieć trzy stany: puste(-), zajęte przez gracza o znaku ”X” oraz gracza o znaku O. Algorytm przewiduje możliwe ruchy przeciwnika z maksymalną głębokością 6. Gra odbywa się poprzez wpisywanie w konsoli odpowiednich współrzędnych planszy do wykonania ruchu. Do kontroli błędów wykorzystałem bibliotekę <except>

2.2 Powiązania klas



3 Wykorzystanie techniki AI

3.1 Algorytm MinMax

Algorytm ten jest metodą wybierania ruchu, który pozwala na minimalizację możliwych strat lub maksymalizację zysków. Jeśli dla danej sytuacji na planszy stworzymy drzewo wszystkich możliwych układów dla n-następnych ruchów, to możemy wybrać taki ruch, który daje nam największe korzyści. Algorytm MinMax dostaje jako parametr korzeń drzewa stanów. Następnie sprawdza, czy aktualny gracz dąży do maksymalizacji, czy do minimalizacji zysków i zależnie od tego wybiera ruch o największej / najmniejszej wartości. Aby to uzyskać, wywołuje się rekurencyjnie aż do osiągnięcia warunku podstawowego, dzięki któremu może zwrócić konkretną wartość. Jeśli założymy, że dla każdego ruchu można wykonać n-następnych, to złożoność obliczeniowa algorytmu wyniesie $O(n^m)$, gdzie m to maksymalna głębokość rekurencji. W tic-tac-toe liczba możliwych ruchów spada wraz z ilością wykonanych ruchów

3.2 Cięcia Alpha-Beta

Cięcia αβ nie są osobnym algorytmem, a raczej modyfikacją do algorytmu MinMax. Zakładają
one dodanie dwóch zmiennych, które przechowują minimalną -β i maksymalną – α wartość, jakie MinMax obecnie może zapewnić na danej głębokości drzewa lub wyżej. Przy starcie algorytmu $\alpha = -\infty$ i $\beta = \infty$, wartości te są korygowane w dalszych etapach działania algorytmu. Na ich pod-
stawie, jeśli przy sprawdzaniu kolejnego poddrzewa węzła minimalizującego αβ algorytm może odciąć dane poddrzewo (w tym przypadku cięcie β), ponieważ już wie, że maksymalna wartość te-go poddrzewa przekroczy minimalną, którą węzeł może obecnie zagwarantować. Bardzo korzystnie wpływa to na złożoność algorytmu, pozwalając osiągnąć dużo lepszy czas wyszukiwania optymalne - go ruchu lub zwiększyć głębokość rekursji.

3.3 Pseudokod

```
function minimax(node, depth, isMaximizingPlayer, alpha, beta):
  if node is a leaf node :
    return value of the node
  if isMaximizingPlayer :
    bestVal = -INFINITY
    for each child node :
      value = minimax(node, depth+1, false, alpha, beta)
      bestVal = max( bestVal, value)
      alpha = max( alpha, bestVal)
      if beta <= alpha:
        break
    return bestVal
  else :
    bestVal = +INFINITY
    for each child node :
      value = minimax(node, depth+1, true, alpha, beta)
      bestVal = min( bestVal, value)
      beta = min( beta, bestVal)
      if beta <= alpha:
        break
    return bestVal
```

4 Wnioski

Algorytm radzi sobie bardzo dobrze w rozgrywce. Złożoność algorytmu jest dosyć duża, ze względu na ilość ruchów, które algorytm ma do rozpatrzenia.
Niekiedy jego reakcja wydaje się długa i maksymalnie dochodzi do 5 minut. Ostatecznie nie udało mi się go pokonać, jedynie zremisować.

5 Bibliografia

-https://en.wikipedia.org/wiki/Minimax

-www.encyklopedia.eduteka.pl

-Stephen Prata ”C++”

-https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/