COSE361-02 인공지능

김현철 교수님

2012130888 김인호

**Four In a Row 인공지능 보고서**

**소스 코드의 기본 구조 및 실행 방법**

TURN이라는 변수를 사용해서 AI가 자신의 턴인지 상대의 턴인지 구분한다. TurnChecker라는 스레드를 통해 TURN을 계속 검사하다가 AI가 둘 차례가 되면 WhereToPut이라는 Class를 이용하여 어디에 돌을 두어야 하는지 판단하고 수를 둔다. Main 함수는 Connect4.java 파일에 정의되어 있다. Main 함수에서 Connect4 Class의 instance를 생성하고 해당 instance 내에 정의된 init이라는 메서드를 호출하면 게임이 실행된다. Init 메서드의 인자로 1을 제공하면 AI가 먼저 시작하게 되고 2를 제공하면 플레이어가 먼저 시작하게 된다. AI가 먼저 시작할 때는 3, 4, 5번 Column에는 두지 않고 랜덤으로 2 또는 6번 Column에 두고 시작하게 된다. 소스 코드에 대한 상세한 설명은 주석으로 일일이 달아 두었다.

**GUI환경**

GUI는 직접 구현하였으며, Java의 awt를 이용하였다. Board 상단의 1~7 사이의 숫자가 적인 버튼을 누르면 해당 Column에 돌을 둔다. Refresh은 가끔 AI가 순식간에 돌을 두어서 Board 상에 돌이 제대로 그려지지 않는 현상 때문에 만들었다. Refresh를 누르면 repaint() 메서드를 호출해서 현재 Board의 상태가 제대로 그려지도록 해준다.

**Minimax 알고리즘과 Alpha-Beta pruning**

노드들에 대한 접근을 용이하기 위해 GameTree와 StateNode라는 별도의 Class를 만들어서 트리의 level별로 접근을 할 수 있도록 하였다. AI의 차례가 되었을 때, WhereToPut이라는 Class에서 Board의 상태를 Root Node로 하여 트리를 생성하고 Minimax 알고리즘을 수행하여 어디에 돌을 둘지 결정한다. WhereToPut Class의 Minimax 메서드는 Alpha-Beta pruning을 추가하지 않은 것으로, 간단하게 level별로 Node들을 탐색하여 Minimize를 할지, Maximize를 할지 판단한다. ABPruningMinimax 메서드는 Alpha-Beta pruning을 추가한 것으로, Recursive method를 통해 Depth First Search 방식으로 노드들을 탐색한다. 수행 시간을 적절히 Control하기 위해 Generate되는 트리의 Depth가 놓여진 돌의 개수에 따라 점점 증가하도록 하였다.

**평가 함수 (Evaluation Function)**

Evaluation은 ScoreChecker.java 파일의 getScore 메서드에서 이루어진다.

1. 기본 구조

기본적으로 수업시간에 다뤘던 Winning line의 개념을 활용하여 Heuristic value를 계산한다. AI는 판(board)의 상태가 주어졌을 때 Heuristic value의 초기값을 0으로 초기화한 뒤, 자신의 Winning line에 대한 점수를 구해서 더한다. 그리고 이와 동시에 상대방의 Winning line에 대한 점수를 구해서 뺀다. 이 과정을 수식으로 간단히 표현하면 다음과 같다.

**E(n) = M(n) – O(n)**

State n에 대하여,

**E(n)**: Total evaluation

**M(n)**: 자신의 Winning line에 대한 evaluation 값들의 합

**O(n)**: 상대의 Winning line에 대한 evaluation 값들의 합

1. Winning line에 대한 evaluation

앞서 언급했듯이 평가 함수는 단순히 Winning line을 count하는 것이 아니라, 이에 더해서Winning line에 대해 특정한 evaluation을 수행해서 점수를 계산한다. 이에 대해 상세히 서술해보겠다.

우선, evaluation은 Winning line을 만들 수 있는 공간은 한정되어 있다는 basic idea로부터 출발한다. 가로선이나 세로선 같은 경우는 board의 모든 공간을 활용하여 만들 수 있지만, 대각선의 경우는 한정된 공간에서만 만들 수 있다. 예를 들어, 좌측상단에서 우측하단으로 뻗어나가는 대각선(RL Diagonal)의 경우는 다음과 같은 공간(색칠된 부분)에서만 한정적으로 만들어질 수 있다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

[Figure ] RL Diagonal Case에서의 Winning lines

이를 통해 검색 범위를 조금이나마 줄일 수 있다. 평가 함수는 이 한정된 범위 내에서 만들어질 수 있는 모든 Winning line 각각에 대해서 Heuristic value를 계산한다. 왼쪽에서부터 오른쪽으로 검색을 한다고 가정할 때, Winning line의 시작점이 될 수 있는 점들은 [Figure 1]의 주황색 부분이므로, Winning line의 수는 12개라고 할 수 있다. 이 과정을 가로선, 세로선, 좌측상단에서 우측하단으로 뻗어나가는 대각선, 우측상단에서 좌측하단으로 뻗어나가는 대각선에 대해 모두 수행한다.

하나의 Winning line을 평가하는 기준은 바로 놓여있는 돌의 개수이다. 이때, 자신의 Heuristic value를 구하는 것이라면, 상대의 돌이 Winning line에 놓여있는 경우 아무 점수도 얻지 못한다. 즉, 점수에 합산되는 고려 대상은 자신의 돌과 비어있는 공간 두 가지뿐이다. 예를 들어, 자신의 돌이 놓여있는 공간을 1, 빈 공간을 0, 상대의 돌이 놓여있는 공간을 2라고 한다면 돌 한 개로 점수를 얻을 수 있는 경우는 1000, 0100, 0010, 0001 네 가지 경우이다. 돌 두 개로 점수를 얻을 수 있는 경우는 1100, 0110, 0011, 1010, 0101, 1001 여섯 가지 경우이다. 돌 세 개로 점수를 얻을 수 있는 경우는 0111, 1011, 1101, 1110으로 네 가지 경우이다. 그리고, 네 개의 돌로 점수를 얻을 수 있는 경우는 1111 한 가지밖에 없다. 1211처럼 2가 포함되어 있다면 아무런 점수도 얻을 수 없다.

각 경우에 대한 점수는 ScoreChecker.java 파일에서 다음과 같이 정의되어 있다.

돌 네 개: ∞

돌 한 개: 100점

돌 두 개: 500점

돌 세 개: 1400점

그 이유는 중복된 계산을 고려해서이다. 돌이 한 개 놓였을 때를 가정해보면, 다음의 경우 (Figure 2) 돌이 놓인 곳은 파란색 부분이라고 하고 수평선 계산을 했을 때 총 네 번 점수를 얻고 (시작점 1, 2, 3, 4) 수직선으로 한 번 점수를 얻기 때문에 (시작점 5) 총 500점의 점수를 얻게 된다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| 1 | 2 | 3 | 4,5 |  |  |  |

[Figure ] 돌이 한 개 놓였을 때

그런데, 상식적으로 돌이 한 개 Winning line에 놓였을 때보다 두 개가 놓였을 때 더욱 유리한 상황이기 때문에 이 점이 반영되어야 한다. 그런데, Figure 2는 주변에 상대의 돌이 아무 것도 없는 이상적인 경우이므로, 상대의 돌이 다양한 공간에 놓여있는 것을 고려했을 때 단방향의 Winning line에 대해서만 계산하는 것이 현실적이라고 판단하였다. 따라서 수직선에 대한 점수 100점은 무시하고 400점이라고 가정한 뒤 돌이 두 개 놓인 경우는 400점보다 높은 500점을 부여하였다.

또한, 돌이 두 개 놓인 경우를 가정해보면, 다음의 경우 (Figure 3) 돌이 두 개 놓인 것에 대한 점수는 수평으로 시작점 1, 2, 3의 경우에서 3번 얻고, 수평으로 시작점 4의 경우와 수직으로 시작점 5, 6의 경우에서는 돌이 한 개 놓인 것에 대한 점수를 얻는다. 따라서 Figure 3의 총 점수는 500\*3 + 100\*3으로 1800점이다. 근데 이 경우에도 현실적인 board의 상태를 고려함과 동시에 돌이 두 개 놓인 것에 대한 점수만으로 계산하여 총 1500점이라고 판단하여 이를 돌 세 개에 대한 점수를 정의하는 데 반영하였다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| 1 | 2 | 3,5 | 4,6 |  |  |  |

[Figure ] 돌이 두 개 놓였을 때

그런데, 당장 한 줄에 세 개의 돌을 놓으면 당연히 상대에게 쉽게 막히기 때문에, 노련한 플레이어라면 세 개의 돌을 한 줄에 놓는 것보다 두 개의 돌을 포함한 여러 줄을 만드는 것이 후에 막을 수 없는 공격을 하는 데 유용하다는 것을 생각하기 마련이다. 따라서 후자에 더 비중을 두어서 세 개의 돌이 놓였을 때의 경우는 1500보다 조금 작은 1400점을 부여하였다.

그리고 당연하게도 네 개가 한 줄에 놓였을 때는 가장 큰 점수를 얻어야 하기 때문에 무한대의 점수를 얻게 된다.

1. Evaluation function에 대한 평가

Heuristic value를 어떻게 도출해내야 더 유리할지를 고민할 때, 흔히 상대방보다 높은 위치에 돌을 둘 때와 가운데 쪽 공간에 둘 때가 승리할 확률이 높다는 것을 고려한다. 즉, 상대보다 먼저 가운데 위쪽의 공간에 많은 돌로 선점하는 것이 승률을 높여주는 것이다. 그런데 이 조건들이 고려되는 이유는 결국 높은 곳 혹은 가운데에 가까운 곳에 둘수록 만들 수 있는 Winning line의 수가 많아지기 때문이다. 따라서 앞서 살펴본 Evaluation function에서는 이 조건들이 자동으로 고려되는 것이라고 할 수 있다. 어차피 가운데 위쪽 공간에 놓인 돌이 Winning line을 많이 만들어서 더욱 높은 점수를 받을 확률이 높아지기 때문에 AI는 자동적으로 evaluation function에 의해서 가운데 위쪽 공간에 두려고 할 것이다. 따라서 따로 가운데 위쪽에 공간에 대한 가중치를 두는 것은 오히려 정확한 Winning line에 대한 평가를 저해할 것이라고 판단하여 포함시키지 않았다.

**개발 환경**

첨부된 zip파일은 Eclipse Java EE IDE for Web Developers. Version: Mars.2 Release (4.5.2)에서 Java 언어로 작성된 Project이며, Archive File로 Export된 형태이다. Eclipse환경에서 Archive File로부터 Import를 하면 온전하게 실행이 가능하다. UTF-8 인코딩은 고려하지 않았기 때문에 한글로 쓰여진 주석은 깨질 수가 있음에 유의해야 한다.

**References**

GameTree의 전체적인 구조와 Minimax 알고리즘

<http://software-talk.org/blog/2012/01/connect4/#download%20connect4>

Winning line에 대한 Heuristic value를 구하는 단순 반복 루틴

<https://github.com/mattnenterprise/LWJGLConnectFour>

Alpha-Beta Pruning 구현

<https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning> (위키피디아의 Pseudocode)