**프로그래밍 언어론 과제1**

(AI 4x4 사목게임 구현)

logo.gif

|  |  |
| --- | --- |
| 과목명 | 프로그래밍 언어론 |
| 담당교수 | 양승민 |
| 학과 | 소프트웨어학부 |
| 학년 | 4 |
| 학번 | 20152994 |
| 성명 | 이진영 |
| 제출일 | 2018.4.5 |



**개 요**

1. **Evaluation 함수 동작**
   * 1. **새로 정의한 Evaluation 함수 개요**
     2. **재귀호출 대신 Iteration 방법을 선택**
   1. **Evaluation3 함수 기본 수행 동작** (Pseudo Code로 설명)
   2. **기타 세부사항**
2. **구현 및 수정한 자료구조 및 함수**
   * 1. **구현한 자료구조 및 함수**
     2. **수정한 자료구조 및 함수**

**1. Evaluation 함수 동작**

**① 새로 정의한 Evaluation 함수 개요**

**-** Look-Ahead level 3을 구현한 함수는 Evaluation3,

Look-Ahead level 5를 구현한 함수는 Evaluation5

**-** 매개변수를 통해 root node, 객체변수를 통해 현재 GameBoard를 입력받고,

객체변수를 통해 bestX/bestY(가장 좋아보이는 다음 수), nCount(총 노드개수),

vec(트리출력을 위한 노드 정보) 리턴

**② 함수호출 오버헤드를 줄이기 위해 재귀호출 대신 Iteration 방법을 선택**

**-** Tree의 Node 생성 순서는 Breadth-first search 순서로 Iteration 형식으로 이루어진다.

**-** 각 depth의 max 또는 min 값은

|  |
| --- |
| int max\_1, min\_2, max\_3, min\_4, max\_5; |

의 형태로 max\_1은 depth 1 Node들 중 maximum값, min\_2는 depth 2 Node들 중

minimum값을 저장하며, 하위 노드들 간의 비교를 통해 상위 노드의 max 또는 min 값을

구한다.

**③ Evaluation3 함수 기본 수행 동작** (Pseudo Code로 설명)

|  |
| --- |
| possible = GetPossibleMove(iList) // 현재 둘 수 있는 수를 구해 iList에 저장  둘 수 있는 수의 개수는 possible에 저장  for(현재 둘 수 있는 수의 개수(possible)만큼 loop){  new Node // depth 1 Node 생성  tttBoard.DoMove(iList.x,iList.Y) // Look-Ahaed level 1 진입, GameBoard에 수를 둔다  **[**GetPossibleMove(iList)**]**  **[**for(현재 둘 수 있는 수의 개수(possible)만큼 loop){**]**  **[**new Node**]**  **[**tttBoard.DoMove(iList.x,iList.y) **]** // Look-Ahead level 2 진입  **[[**GetPossibleMove(iList)**]]**  **[[**for(현재 둘 수 있는 수의 개수(possible)만큼 loop){**]]**  **[[**new Node**]]**  **[[**tttBoard.DoMove(iList.x,iList.y)**]]** // Look-Ahead level 3 진입  **[[**Node->eval = EvaluateBoard()**]]** // Terminal Node 이므로 eval값 구함  **[[**tttBoard.UndoMove()**]]**  }  **[**root->next[i]->next[j]->eval = max\_3**]** // level3 Node간의 비교를 통해 구한 값  **[**root->next[i]->next[j]->x = best\_x**]**  **[**root->next[i]->next[j]->y = best\_y**]**  **[**tttBoard.UndoMove()**]**  }  root->next[i]->eval = min\_2 // level2 Node간의 비교를 통해 구한 값  root->next[i]->x = best\_x  root->next[i]->y = best\_y  tttBoard.UndoMove()  }  root->eval = max\_1 // level1 Node간의 비교를 통해 구한 값  bestX = root->x = best\_x  bestX = root->y = best\_y  …  /\*이후 PrintNode를 위한 vector 변수 사용 및 메모리 정리 부분\*/  }  /\* Evaluation5 함수는 위와 동일한 구조에서 depth를 늘림 \*/ |

**④ 기타 세부사항**

**-** 해당 소스코드는 Visual Studio 2015 에서 수정 및 컴파일 함

**-** Look-Ahead level 5에서 사람이 첫 수를 뒀을때 아래와 같은 환경(노트북)에서

AI가 첫 수를 두기까지 최대 약 4분이 걸리고 마지막 수를 둘 때까지 문제없이 진행됨

|  |
| --- |
| 1.PNG |
| 메모리_cpu.PNG |

**-** GetPossibleMove() 내부의 CheckSymmetric() 에서 대칭성 검사 수행

**-** level1 Node간에 evaluation 값이 같을 경우 같은 Node 중에서 랜덤으로 수를 선택

**2. 구현 및 수정한 자료구조 및 함수**

**① 구현한 자료구조 및 함수**

**=구현한 자료구조=**

ⓐ **struct PerLevelNodeInfo** (GameBoard.h)

**-** Tree의 Node 정보를 다이얼로그 상에 보여주기 위한 정보를 저장하는 구조체 변수

**=구현한 함수=**

ⓐ **void Evaluation3**(treeNode\* root) (TictactoeAI.h/c)

**-** AI의 Look-Ahead level3 기능을 구현

ⓑ **void Evaluation5**(treeNode\* root) (TictactoeAI.h/c)

**-** AI의 Look-Ahead level5 기능을 구현

ⓒ **void PrintNode**(vector<nInfo>& info, unsigned int nodeCount) (TictactoDlg.h/c)

**-** 새로 만든 Evaluation 함수는 최선의 eval값/좌표값을 구한 다음

Tree에서 필요없는 Node는 바로 delete한다.

따라서 Tree를 다이얼로그상에 보여주는 데에 필요한 정보를 nInfo 구조체에 따로

저장하여 vector로 묶은 다음 출력한다.

ⓓ **void DeleteBoard**(GameBoard\* board) (TictactoeAI.h/c, TictactoDlg.h/c)

**-** 디버깅 과정 중 아래 코드에서 주로 Memory Leak이 발생하는 것을 확인하였고,

이 생성자 내부에서 생성한 객체들을 한꺼번에 delete 하기위해 만든 함수

|  |
| --- |
| 디버그2.PNG |

ⓔ **OnDestory()** (TictactoDlg.h/c)

**-** 프로그램 종료시 m\_board 객체 내부에서 메모리가 할당된 객체들을 delete 하기위해

해당 함수 내부에서 DeleteBoard(&m\_board) 함수 호출

**② 수정한 자료구조 및 함수**

**=수정한 자료구조=**

ⓐ 3x3 tictacto에 맞춰진 배열 크기 등을 **4x4 사목에 맞게 조정**

ⓑ **iList, possible** 변수를 다차원 배열로 선언

**-** 재귀함수 구조에서 Iteration 구조로 바뀜에 따라 사용하기 적절하도록 다차원 배열로

변경

**=수정한 함수=**

ⓐ **Maximize/Minimize 함수 제거**

**-** 해당 함수를 제거한 대신에 새로 만든 Evaluation 함수 내에 해당 기능을 구현

ⓑ **void PrintTreeNode(Node\* root) 함수 삭제**

**-** 해당 함수 대신 더 간략하게 node정보를 저장/출력할 수 있도록 PrintNode함수 구현