SW 페스티벌
육목 경진대회

AI Connect6 (육목)
유뇨윤효 팀
21700589 이찬효 & 21800741 최윤서

SW페스티벌 육목AI

목차

1-1. 육목의 배경

- 대만의 우이청(吳毅成) 교수가 만든 게임으로 바둑판 위에 먼저 여섯 개의 돌을 방해 없이 일렬로 늘어놓으면 이기는 게임. 오목은 먼저 시작하는 흑이 절대적으로 유리하지만 육목의 경우 항상 상대보다 돌이 1개씩 많기 때문에 꽤나 공평하고 단순한 규칙을 가지고 있다.

1-2. 육목의 규칙

i. 게임의 랜덤성을 부여하기 위해 시작하기 전에 6개 이상의 적돌을 임의의 위치에 배치한다.
ii. 흑돌이 1개로 시작하여 이후로는 각 턴마다 2개의 돌을 착수한다.
iii. 여섯 개의 돌을 가장 먼저 배치하는 사람이 승리하는데, 7목 등 장목은 허용되지 않는다.

2-1. 알고리즘

*다음과 같은 순서로 수의 우선 순위를 파악하였음

i. 이번 두 수로 육목을 완성할 수 있는가?

ii. 없다면 이번 두 수로 꼭 저지해야하는 상대의 육목이 있는가?

iii. 없다면 가장 영향력이 높은 움직임이 어디인가?

2-2. 객체지향

*다음과 같이 객체와 기능을 분류하였음

i. 각 자리를 의미하는 Tile과 그 Tile에 위치할 Stone

ii. 육목 게임을 컨트롤하는 GameController

iii. 모든 타일의 현재상황을 갖고 있는 CheckBoard

iv. 두 수를 묶어 가지고 있는 Moves

v. 기타 메인&팝업 프레임, 사운드패키지 등 존재

vi. 현재 CheckBoard의 상황을 기반으로 최적의 수를 판단하는 Decision (*여기서는 인공지능) 존재

3-1. Reference

블로그

- https://blog.encrypted.gg/
- 참고했던 블로그 링크를 첨부합니다.

소스코드

- https://github.com/encrypted-def/Connect6AI_SamsungDSCompetition/blob/master/AI_Algorithm_Code.cpp
- 참고했던 소스코드의 링크를 첨부합니다.
- 상세한 설명은 이후 메소드 설명부분에서 덧붙이겠습니다.

3-2. 메소드 설명

public static void **find_best()**

- 최고의 수를 실행할 수 있도록 하는 함수로, 각 단계의 로직을 병합
- 알고리즘의 우선순위에 해당하는 함수를 순서대로 불러와 적용

public static void **ai_click(Point first_move)**

- 마우스 이벤트를 발생시켜 주어진 좌표를 클릭

public static boolean **IsOutOfBounds(int x, int y)**

- 주어진 인덱스 값의 배열의 최소, 최대 범위 판단

public static Vector<Point> **get_allPossibleMoves()**

- 현재 보드의 모든 빈 포인트를 벡터 구조에 담아 반환

public static double **calculateDoubleMove_score(Moves myMoves, int player)**

- 두 수의 점수 최댓값을 계산하는 함수
- 처음 착수하는 돌을 보드에 반영시켜 두번째 착수의 점수를 계산

3-2. 메소드 설명

public static double **calculateSingleMove_score(Point myMove, int player)**

- 한 움직임의 최댓값을 계산하는 함수
- 6칸에 속한 상대와 나의 돌, 추가로 적돌이 있을 때 가중치를 계산
- 4개, 5개 이상일 때는 상위 함수에서 걸러지기 때문에 가중치를 적용할 필요가 없음

❖ 가중치는 3.1에서 언급한 블로그의 가중치를 가져왔음

❖ 해당 가중치는 블로그 작성자가 최적화한 가중치로, 강화학습에 대한 지식이 부족하여 직접 구현하기 어렵다고 판단하여 이를 채택하였음

public static Point **Find_BestSingleMove(int [] [] board)**

- 모든 빈칸에 수를 두었을 때 점수가 가장 높은 지점을 찾는 함수

private static Moves **defend_threat()**

- 위협이 되는 자리를 최대의 효율로 막는 움직임을 반환하는 함수
- 첫 수를 보드에 반영한 뒤, 다시 위협을 탐색하여 한 수만으로도 위협이 사라지면 나머지 한 수는 자유롭게 둘 수 있도록함

private static Vector<Point> **find_threat(int [] [] board, int x, int y)**

- 기준점에 대해 4방향을 탐색한 후 상대의 승리조건에 해당하는 위치들을 반환
- 6개의 칸이 보드 안에 유효하게 들어올 경우에만 지속하여 탐색
- 6개의 칸을 문자열 형태로 변환하여 해시맵 키와 비교 후, 키에 있으면 빈칸을 반환

public static Vector<Point> **find_pattern(int x, int y)**

- 기준점에 대해 4방향을 탐색한 후 나의 승리조건에 해당하는 위치들을 반환
- 6개의 칸이 보드 안에 유효하게 들어올 경우에만 지속하여 탐색
- 6개의 칸을 문자열 형태로 변환하여 해시맵 키와 비교 후, 키에 있으면 빈칸을 반환

public static HashMap<String, int[]> **makeConnect6_hashmap(int player)**

- 4목이 완성되었을 때 승리 패턴을 키 값으로 하고 빈칸을 value값으로 가지는 해시맵 생성
- 6C2가지의 경우의 수가 있음, 플레이어에 따라 키 값을 다르게 생성

public static HashMap<String, int[]> **makeConnect6_5_hashmap(int player)**

- 5목이 완성되었을 때 승리 패턴을 키 값으로 하고 빈칸을 value값으로 가지는 해시맵 생성
- 6C1가지의 경우의 수가 있음, 플레이어에 따라 키 값을 다르게 생성

public static void **update_HashMap()**

- Player에 맞는 해시맵 두 쌍 생성