[설계 교과목]

알고리즘 및 실습 Report

<Term Project>

과목 : 알고리즘 및 실습

분반 : 03분반

학번 : 2008160056

학부 : 컴퓨터공학부

조 : 16조

이름 : 문 찬 호

담당교수 : 한 연 희 교수님

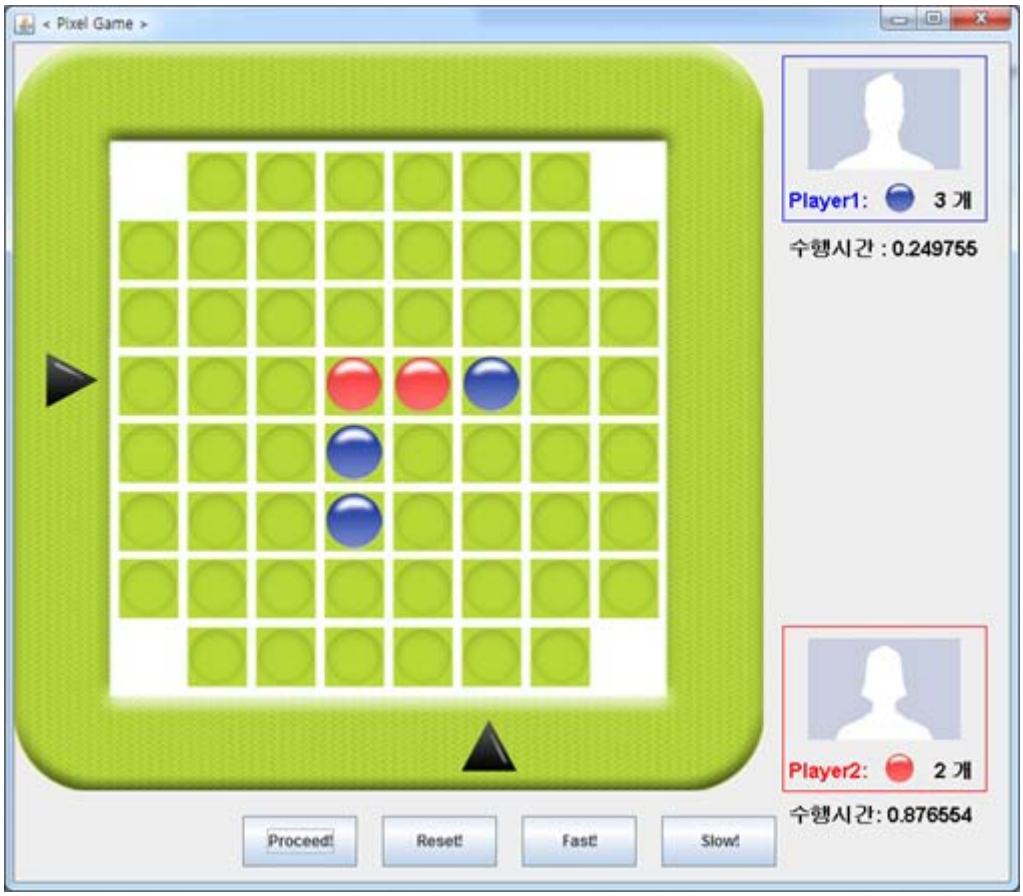
제출일자 : 2013.06.15.(토)

1. 서론

① Pixel 게임의 알고리즘을 작성하고 실행해본다.

② Pixel 게임?

픽셀은 슬라이더를 움직여 좌표가 맞는 자리에 게임칩을 내려놓는 보드게임입니다.
오직, 상하 좌우중 한쪽 슬라이더만 움직여서 일치되는 좌표에 칩을 내려놓을수 있습니다.



서로 돌을 두는데 돌을 가로, 세로 대각선 중 4개를 연속해서 놓으면 승리하는 게임입니다.

③ 목적 : 인공지능 지식 함양, 알고리즘 지식 활용

2. 본론

① 승리 전략!

⒜ 공격 : 기존의 오목과는 다르게 Pixel에서는 가로나 세로 슬라이드 중 하나의 슬라이드를 움직인다는 것입니다. 하지만 공격에 있어서는 오목하고 비슷한 생각을 하게 되었습니다. 공격은 가장 최근의 돌의 가로 좌표와 세로 좌표를 검사해서 가로, 세로에 상대방의 돌이 어떻게 배열되어 있나를 판별한 후, 자신이 공격 가능하다면 공격을 하는 것입니다. 예를 들어서 상대방에게 끝낼 방법이 없고, 나 자신은 공격이 가능하다면 공격하는 것입니다. 1개에서 2개를 만들 수 있고, 2개에서 3개를 만들 수 있습니다. 만약에 자신이 현재 슬라이드 한 개를 움직여서 놓을 수 있고, 4개를 놓을 수 있는 점이 있다면 그 점을 우선적으로 선택해야 합니다. 이와 같은 생각으로 공격 전략을 짜게 되었습니다.

⒝ 방어 : 방어에 있어서는 기존 오목보다도 방법이 많아졌다고 생각합니다. 일단 가장 최근의 돌의 위치를 검색하고 그 좌표의 가로나 세로를 탐색하여 상대방의 돌이 나란히 놓여있다면 두가지의 선택이 가능하게 됩니다.

1) 두 개의 돌이 나란히 놓인 경우
- 이 경우에는 자신의 돌을 그 두 돌의 옆에 놓으면 됩니다.

2) 두 개가 돌이 한 칸 띄워져서 놓인 경우
- 이 경우에는 자신의 돌을 그 두 돌의 사이가 비어있다면 그 사이에 놓고 그게 아니라면 무시하면 됩니다.

3) 세 개의 돌이 나란히 놓인 경우
- 이 경우에는 무조건 그 나란히 놓인 슬라이드가 아닌 다른 축의 슬라이드를 움직여서 게임을 승리하지 못하게 해야 합니다.

② 승리 절차!

⒜ 우선적으로 공격만으로 승리하기는 어렵습니다. 선 방어 후에 공격의 빈틈이 있을 경우에 공격을 하여 승리하는 것이 가장 이상적인 절차라고 생각합니다.

⒝ 방어적으로 경기를 풀게 되면 최소한 비길 수 있기 때문에 방어적으로 경기하는 것이 가장 좋습니다. 물론, 알고리즘이 올바르게 동작해야 이 방법은 통하게 됩니다.

⒞ Greedy한 방법을 통해서 Attack 변수와 Danger변수를 지정해서 위에 제시된 방법을 실천하기 위해 그 변수를 통해 전략을 결정할 수 있습니다.

⒟ 이 방법이 그 상황 내에서 최선일 수는 없지만, 그나마 가장 올바르다고 생각할 수 있습니다.

③ 승리 방법!

⒜ 공격 : 가로줄을 우선 검사하여 자신의 돌이 있는 지를 확인하고, 그 돌이 연속되어 있는지를 확인합니다. 연속할 경우에는 attack변수를 증가시켜서 공격이 가능한지를 표현했습니다. 공격지수가 일정 숫자 이상이라면 공격이 가능합니다.

그 뒤 가로 공격을 검사한 후, 세로도 같은 방법으로 검사하여 확인합니다. 그 결과 공격할 점이 있다면, 공격 가능한 위치인 빈 곳을 검사하여 돌을 놓습니다.

탐욕적인 방법이라고 생각합니다.

attack 지수가 증가하면서

```
   // 가로 공격 (선공일땐 방어)
                                        if ( map[i][y] == 1 && ( i == n || y ==
m)) {
                                          if (i < 7 && (map[i+1][y] == 1)) {
                                                attack1++; // 자신의 돌이 있다면
attack 지수 증가
                                              }
                                          if (i < 7 && (map[i+1][y] == 2)) {
                                                attack1--; // 상대의 돌이 있다면
attack 지수 하락
                                             }
                                          }

                                        // attack 지수가 1이상이면서
                                        if (attack1 >= 1) {
                                            if (i > 0 && map[i-1][y] == 0) { // 옆
의 좌표에 돌이 없다면 돌을 둔다.
                                               nextPosition = new Point(i-1, y);

System.out.println(nextPosition); //
                                                        return nextPosition;

                                          }
                                          else if (i < 7 && map[i+1][y] == 0) {
                                             nextPosition = new Point (i+1,y);
                                             System.out.println(nextPosition);
                                                        return nextPosition;
                                                        }
                                                      }

// 세로 공격(선공일땐 방어)
                                         if ( map[x][i] == 2 && ( i == n || y ==
m)) {
                                          if (i < 7 && (map[x][i+1] == 2)) {
                                               attack2++;
                                               }
                                        if (i < 7 && (map[x][i+1] == 1)) {
                                               attack2--;
                                               }
                                          }

                                        if (attack2 >= 1) {
                                            if (i > 0 && map[x][i-1] == 0) {
                                              nextPosition = new Point(x, i-1);

System.out.println(nextPosition);
                                                        return nextPosition;

                                          }
                                          else if (i < 7 && map[x][i+1] == 0) {
                                             nextPosition = new Point (x, i+1);
                                             System.out.println(nextPosition);
                                                        return nextPosition;
                                                        }
                                                      }
```

⑹ 방어 : 가로줄을 우선 검사하여 상대의 돌이 있는 지를 확인하고, 그 돌이 연속되어 있는지를 확인합니다. 연속할 경우에는 danger변수를 증가시켜서 방어를 해야하는 지 표현했습니다. 위험지수가 일정 숫자 이상이라면 방어를 해야 합니다.
그 뒤 가로 방어을 검사한 후, 세로도 같은 방법으로 검사하여 확인합니다. 그 결과 방어해야 할 점이 있다면, 방어 가능한 위치인 빈 곳을 검사하여 돌을 놓습니다.

```
// 위험지수 (선공일땐 공격)                    // 상대의 돌을 발견하고 지수를 검사해서 해결
if ( map[i][y] == 2 && ( i == n || y == m)) {
  if (i < 7 && (map[i+1][y] == 2)) {
      danger1++;
      }
  if (i < 7 && (map[i+1][y] == 1)) {
      danger1--;
      }
  }

if (danger1 >= 1) {
   if (i > 0 && map[i-1][y] == 0) {
     nextPosition = new Point(i-1, y);
System.out.println(nextPosition);
                          return nextPosition;
  }
  else if (i < 7 && map[i+1][y] == 0) {
    nextPosition = new Point (i+1,y);
    System.out.println(nextPosition);
                    return nextPosition;
                    }

 // 세로 방어(선공일땐 공격)
if ( map[x][i] == 2 && ( i == n || y == m)) {
  if (i < 7 && (map[x][i+1] == 2)) {
      danger2++;
      }
if (i < 7 && (map[x][i+1] == 1)) {
      danger2--;
      }
  }

if (danger2 >= 1) {
   if (i > 0 && map[x][i-1] == 0) {
     nextPosition = new Point(x, i-1);
System.out.println(nextPosition);
                          return nextPosition;
  }
  else if (i < 7 && map[x][i+1] == 0) {
    nextPosition = new Point (x, i+1);
    System.out.println(nextPosition);
                    return nextPosition;
                    }
                  }
```

④ 소스 코드

```
import java.awt.*;
import java.util.Random;

public class PixelPlayer16 extends Player {
        PixelPlayer16(int[][] map) {  //이 생성자 이름은 자신의 클래스명과 동일하게 변경할 것
                super(map);
        }

        public Point nextPosition(Point lastPosition) {
```

```
                int x = (int)lastPosition.getX(), y = (int)lastPosition.getY();
                int cx = (int)currentPosition.getX(), cy =
(int)currentPosition.getY();
                int direction, count = 0;
                int myNum =
map[(int)currentPosition.getX()][(int)currentPosition.getY()];
                int checkPositionX;
                int checkPositionY;
                int danger1, danger2; // 위험요인 변수
                int attack1, attack2; // 공격 변수
                Point nextPosition;
                Random random = new Random();


                for( int n = 0; n < 8; n++ )
                {
                        for( int m = 0; m < 8; m++ )
                        {
                                if( myNum == map[n][m] )
                                {
                                        for(int i = 0; i <
PixelTester.SIZE_OF_BOARD; i++)
                                        {
                                          attack1 = 0;
                                          danger1 = 0;

                                          // 가로 공격 (선공일땐 방어)
                                          if ( map[i][y] == 1 && ( i == n || y ==
m)) {
                                            if (i < 7 && (map[i+1][y] == 1)) {
                                                  attack1++; // 자신의 돌이 있다면
attack 지수 증가
                                                }
                                            if (i < 7 && (map[i+1][y] == 2)) {
                                                  attack1--; // 상대의 돌이 있다면
attack 지수 하락
                                                }
                                          }

                                          // attack 지수가 1이상이면서
                                          if (attack1 >= 0) {
                                             if (i > 0 && map[i-1][y] == 0) { //
옆의 좌표에 돌이 없다면 돌을 둔다.
                                               nextPosition = new Point(i-1, y);

System.out.println(nextPosition); //
                                                        return nextPosition;

                                          }
                                          else if (i < 7 && map[i+1][y] == 0) {
                                            nextPosition = new Point (i+1,y);
                                            System.out.println(nextPosition);
                                                        return nextPosition;
                                                        }
                                                      }


                                          // 위험지수 (선공일땐 공격)
                                          // 상대의 돌을 발견하고 지수를 검사해서
해결
                                          if ( map[i][y] == 2 && ( i == n || y ==
m)) {
                                            if (i < 7 && (map[i+1][y] == 2)) { //
상대의 돌이 있다면 위험지수 증가
                                                danger1++;
                                                }
                                            if (i < 7 && (map[i+1][y] == 1)) { //
자신의 돌이 있다면 위험지수 하락
                                                danger1--;
                                                }
                                          }

                                          if (danger1 >= 0) {
```

```
                if (i > 0 && map[i-1][y] == 0) {
                  nextPosition = new Point(i-1, y);
System.out.println(nextPosition);
                        return nextPosition;
                }
                else if (i < 7 && map[i+1][y] == 0) {
                  nextPosition = new Point (i+1,y);
                  System.out.println(nextPosition);
                        return nextPosition;
                        }
                      }
            }

            for(int i = 0; i < PixelTester.SIZE_OF_BOARD; i++)
            {
              attack2 = 0;
               danger2 = 0;
               // 세로 공격(선공일땐 방어)
               if ( map[x][i] == 1 && ( i == n || y == m)) { // 자신의 돌이 있다면 attack 증가
                if (i < 7 && (map[x][i+1] == 1)) {
                    attack2++;
                    }
               if (i < 7 && (map[x][i+1] == 2)) { // 상대의 돌이 있다면 attack 감소
                    attack2--;
                    }
              }

              if (attack2 >= 0) {
                 if (i > 0 && map[x][i-1] == 0) {
                   nextPosition = new Point(x, i-1);
System.out.println(nextPosition);
                        return nextPosition;
                }
                else if (i < 7 && map[x][i+1] == 0) {
                  nextPosition = new Point (x, i+1);
                  System.out.println(nextPosition);
                        return nextPosition;
                        }
                      }

               // 세로 방어(선공일땐 공격)
              if ( map[x][i] == 2 && ( i == n || y == m)) {
                 if (i < 7 && (map[x][i+1] == 2)) { // 상대의 돌이 있다면 위험지수 증가
                    danger2++;
                    }
              if (i < 7 && (map[x][i+1] == 1)) {  // 자신의 돌이 있다면 위험지수 감소
                    danger2--;
                    }
              }

              if (danger2 >= 0) {
                 if (i > 0 && map[x][i-1] == 0) {
                   nextPosition = new Point(x, i-1);
System.out.println(nextPosition);
                        return nextPosition;
                }
                else if (i < 7 && map[x][i+1] == 0) {
                  nextPosition = new Point (x, i+1);
                  System.out.println(nextPosition);
                        return nextPosition;
```

```
                                                    }
                                                }

                                            }
                                        }
                                    }
                                }

                                nextPosition = new Point(x, y);
                                return nextPosition;
                            }
}
```

3. 결론

이번 텀 프로젝트는 해결이 잘 되지 못했습니다. 대전을 하게 되면 하위를 기록 할 것 같습니다만, 알고리즘 시간에 배운 알고리즘을 생각 하여 전략을 짤 수는 있었습니다. 단. 코드 짜는 실력이 부족하여 명확하게 구현하지는 못했습니다. 방학 때에 부족한 코드 실력을 보완해서 위 전략을 보다 완벽하게 구현할 수 있도록 노력하겠습니다. 한 학기 동안 알고리즘 수업을 통해 많은 것을 배웠습니다. 앞으로 코드 짤 때 많은 전략을 생각해서 좋은 효율의 알고리즘으로 프로그램을 짤 수 있도록 노력할 것입니다. 한학기 동안 많은 것을 가르쳐 주셔서 감사합니다.