이 문서는 유전자 알고리즘을 이용해 오목 인공지능을 구현한 수도코드를 설명하는 문서이다.

강사님께서 주신 코드로 관찰하면서 보완하고 싶은 내용이 몇가지 생기게 되었다.

1. 같은 색의 돌이 3개나 4개 나열되어 있는 중요한 상황에서(특히, 4개 나열된 상황에서) 그대로 게임을 끝내거나 막는 조치를 우선적으로 하지 않는 현상
2. 자신 색의 돌만 카운트하여 승리 가능성을 구하다보니, (빈칸,흰,흰,흰,검), (빈칸,흰,흰,흰,빈칸) 두 상황이 같은 승리 가능성을 가지게 된다.

이 두가지는 각각 유전자의 결과값을 구하는 함수, 승리 가능성을 구하는 함수의 코드를 의도에 알맞게 수정하였다.

* 이 유전자 알고리즘에 대한 설명 :

한 유전자에 8칸의 양의 정수들의 배열로 이루어져 있다. 유전자를 무작위로 생성하고 그 유전자에 들어있는 값들을 이용해 위치를 정해 그 위치에서의 흑돌과 백돌의 승률을 구할 것이다. 승률이 가장 높은 곳의 위치에 인공지능 자신의 돌을 놓게 될 것이다. 인공지능은 유전자가 가리키는 자리를 기준으로 가로, 세로, 대각선 방향으로 오목판을 체크하며 자신의 돌이 많이 분포해있다면 그 유전자에 대해 높은 결과값을 도출해낼 것이다. 이를 통해 자신의 놓을 자리를 정하게 된다.

다음은 수도코드에 대한 설명이다.

|  |
| --- |
| Init |
| INPUT : data, myid  m\_fg <- 0.1f  m\_ppMap <- data  m\_GeneList.clear()  m\_nMyID <- myid  m\_nMaxgeneration <- 6  m\_nInitGeneCount <- 16 |

Init 함수는 게임이 처음 시작할 때 이 인공지능의 설정값을 정하는 함수이다.

m\_fg는 이 인공지능의 공격 및 수비 성향을 나타낸다. 높은 값을 가질수록 공격적인 성향이다.

m\_ppMap은 정수타입으로 이루어진 13x13 행렬 맵 데이터를 나타내며 각 칸마다 0은 빈칸, 1은 백돌, 2는 흑돌을 의미하게 할 것이다.

m\_GeneList는 유전자를 모두 담은 리스트이다. Clear 함수는 리스트 안에 있는 모든 성분을 지우는 함수이다.

m\_nMyID는 자신이 어떤 색의 돌인지를 나타내는 데이터이다.

m\_nMaxgeneration은 유전자 알고리즘에서 선택-교차 과정을 몇 번 반복할지 나타내는 데이터이다.

m\_nInitGeneCount는 선택-교차 과정을 시작하기 전 처음에 몇 개의 유전자를 무작위로 생성할지 정하는 데이터이다.

|  |
| --- |
| FrameMove |
| INPUT: m\_nMaxgeneration, m\_GeneList  onCreateGene()  FOR n=0 in range(0, m\_nMaxgeneration)  FOR I in range(0, m\_GeneList.size)  GetCheckGene(i)  End FOR  sort(m\_GeneList)  onReleaseGene()  IF n!= m\_nMaxgeneration-1 do  onMatingGene()  End IF  End FOR  p <- Vector(m\_GeneList[0].x,m\_GeneList[0].y)  m\_GeneList.clear()  RETURN p |

FrameMove함수는 인공지능이 자신의 턴마다 실행하는 함수이다. 처음에 유전자를 생성하고, 각 유전자의 결과값들을 계산하여 우성과 열성을 선별하여 교배하는 과정을 반복하는 내용을 담고 있다. 이 과정을 반복하여 제일 우성인 유전자를 채택하여 인공지능 자신이 놓을 자리를 정하게 된다.

onCreateGene 함수는 자신의 턴 처음에 초기 유전자들을 생성하는 함수이다.

GetCheckGene 함수는 한 유전자의 결과값을 도출해내는 함수이다.

onReleaseGene 함수는 이미 내림차순으로 정렬된 유전자리스트 중 열성인 유전자를 버리는 함수이다.

onMatingGene 함수는 우성으로 판별된 유전자를 무작위로 교배하여 다음 세대에서 사용할 유전자를 설정하는 함수이다.

Sort 함수는 유전자 리스트를 결과값에 따른 내림차순으로 정렬하는 함수이다.

|  |
| --- |
| onCreateGene |
| INPUT : m\_GeneList, m\_nInitGeneCount, m\_vUserPos, MAPSIZE  m\_GeneList.clear()  FOR i in range(1, m\_nInitGeneCount)  GENE a  FOR j in range(1, 8)  a.g[j] <- rand()%2  End FOR  m\_GeneList.push\_back(a)  End FOR  FOR x in range(-1, 1)  FOR y in range(-1, 1)  GENE a  dx <- m\_vUserPos.x+x  dy <- m\_vUserPos.y+y  IF dx <= 0 do  dx <- 0  End IF  IF dy <= 0 do  dy <- 0  End IF  IF dx >= MAPSIZE-1 do  dx <- MAPSIZE - 1  End IF  IF dy >= MAPSIZE-1 do  dy <- MAPSIZE - 1  End IF  a.x <- dx  a.y <- dy  FOR i in range(0,3)  a.g[i] <- dx%2  dx <- dx/2  End FOR  FOR i int range(4,7)  a.g[i] <- dy%2  dy <- dy/2  End FOR  m\_GeneList.push\_back(a)  End FOR  End FOR |

onCreateGene 함수는 자신의 턴 처음에 초기 유전자들을 생성하는 함수이다.

GENE 은 유전자 구조체이며 유전자(g[ ]), 결과값(re), 맵의 위치정보(x, y)를 담고있다.

Rand 함수는 랜덤한 양의 정수 값을 가져오는 함수이다.

Push\_back함수는 리스트에 매개변수 값을 추가하는 함수이다.

m\_vUserPos는 인공지능이 이전에 돌을 놓았던 위치벡터를 의미한다.

MAPSIZE는 맵의 한 줄의 크기인 13을 의미한다.

|  |
| --- |
| GetCheckGene |
| INPUT: id, m\_GeneList, MAPSIZE, m\_ppMap, m\_nMyID  x <- m\_GeneList[id].g[0]+m\_GeneList[id].g[1]\*2+m\_GeneList[id].g[2]\*4+m\_GeneList[id].g[3]\*8  y <- m\_GeneList[id].g[4]+m\_GeneList[id].g[5]\*2+m\_GeneList[id].g[6]\*4+m\_GeneList[id].g[7]\*8  m\_GeneList[id].x <- x  m\_GeneList[id].y <- y  IF y<MAPSIZE&&x<MAPSIZE&&x>=0&&y>=0 do  IF m\_ppMap[x][y]== 0 do  tThisWinRate <- GetCheckWinrate(m\_nMyID, x, y)  tOtherWinRate <- GetCheckWinrate(3-m\_nMyID, x, y)  g1 <- 1 - m\_fg  t <- (tThisWinRate \* m\_fg) + (tOtherWinRate \* g1)    IF tThisWinRate == 1 do  p2 <- tThisWinRate  m\_GeneList[id].re <- tThisWinRate  else IF tOtherWinRate == 1 do  p2 <- tOtherWinRate  m\_GeneList[id].re <- tOtherWinRate  else IF tOtherWinRate == 0.75 do  p2 <- tOtherWinRate  m\_GeneList[id].re <- tOtherWinRate  else IF tThisWinRate == 0.75 do  p2 <- tThisWinRate  m\_GeneList[id].re <- tThisWinRate  else do  p2 <- t  m\_GeneList[id].re <- p2  End IF  RETURN p2  End IF  m\_GeneList[id].re <- 0  RETURN 0  else do  m\_GeneList[id].re <- 0  RETURN 0  End IF |

GetCheckGene 함수는 한 유전자의 결과값을 도출해내는 함수이다. 유전자의 값들을 이용해 위치를 지정하고 그 위치에 자신의 돌을 놓았을 때의 자신의 승률과 상대가 상대의 돌을 놓았을 때의 상대의 승률을 계산하여 가중치 계산을 통해 알맞게 유전자의 결과값을 도출하게 된다.

GetCheckWinRate(id, x, y) 함수는 id의 돌이 맵의 x행 y열 자리에 놓였을 때 id가 이길 가능성을 구하는 함수이다.

If ~ else if 문은 다음과 같은 의도대로 작성하였다.

1. 그 자리에 놓으면 무조건 승리하는 경우 가중치 계산을 하지 않고 그 유전자의 결과값을 1로 설정한다.
2. 그 자리에 상대 돌이 놓이면 상대가 승리하는 경우 가중치 계산을 하지 않고 그 유전자의 결과값을 1로 설정한다.
3. 상대의 돌이 3개가 놓이면 가중치 계산을 하지 않고 그 유전자의 결과값을 0.75로 설정한다.
4. 내 돌이 3개가 놓이면 가중치 계산을 하지 않고 그 유전자의 결과값을 0.75로 설정한다.

|  |
| --- |
| onReleaseGene |
| count <- m\_GeneList.size \* 0.7  FOR i in range(0, count)  m\_GeneList.erase(m\_GeneList.end())  End FOR |

onReleaseGene 함수는 이미 내림차순으로 정렬된 유전자리스트 중 결과값이 하위 30퍼센트인 유전자를 버리는 함수이다.

Erase 함수는 리스트에서 특정 원소를 제거하는 함수이다.

End 함수는 리스트에서 마지막 원소를 가져오는 함수이다.

|  |
| --- |
| onMatingGene |
| count = m\_GeneList.size()  FOR i in range(0,count)  FOR j in range(0,count)  IF i != j do  pt = rand() % 8+1  pt1 = 8 - pt  GENE a  a.g[1 ~ pt] <- m\_GeneList[i].g[1 ~ pt]  a.g[(pt+1) ~ 8] <- m\_GeneList[j].g[(pt+1) ~ 8]    r = rand() % 10  IF r == 1 do  t = rand()%8  IF a.g[t]==0 do  a.g[t]=1  else do  a.g[t]=0  End IF  End IF  m\_GeneList.push\_back(a)  End IF  End FOR  End FOR |

onMatingGene 함수는 우성으로 판별된 유전자를 무작위로 교배하여 다음 세대에서 사용할 유전자를 설정하는 함수이다. 두 유전자를 각각 두개로 쪼개서 교차하여 붙인다. 그리고 10퍼센트 확률로 유전자에 돌연변이를 준다.

|  |
| --- |
| GetCheckWinrate |
| count[] = {0,0,0,0}  left <- FALSE  right <- FALSE  //가로  FOR I in range(1,8)  IF right == TRUE && left == TRUE do  break  dx1 <- x - i  dx2 <- x + i  IF m\_ppMap[dx1][y] == 3 - mid || m\_ppMap[dx2][y] == 3 – mid do    count[0]--  IF count[0] <= 0 do  count[0] <- 0  break  End IF  IF m\_ppMap[dx1][y] == mid && left == FALSE && dx1 >= 0 && dx1 < MAPSIZE do    count[0]++  else do  left <- TRUE  End IF  IF m\_ppMap[dx2][y] == mid && right == FALSE && dx2 >= 0 && dx2 < MAPSIZE do    count[0]++  else do  right <- TRUE  End IF  End FOR  left <- FALSE  right <- FALSE  //세로  FOR i in range(1,8)  IF right == TRUE && left == TRUE do  break  dy1 <- y - i  dy2 <- y + i  IF m\_ppMap[x][dy1] == 3 - mid || m\_ppMap[x][dy2] == 3 – mid do    count[1]--  IF count[1] <= 0 do  count[1] <- 0  break  End IF  IF m\_ppMap[x][dy1] == mid && left == FALSE && dy1 >= 0 && dy1 < MAPSIZE do    count[1]++  else do  left <- TRUE  End IF  IF m\_ppMap[x][dy2] == mid && right == FALSE && dy2 >= 0 && dy2 < MAPSIZE do    count[1]++  else do  right <- TRUE  End IF  End FOR  left <- FALSE  right <- FALSE  //우하향 대각선  FOR i in range(1,8)  IF right == TRUE && left == TRUE do  break  dx1 <- x + i  dy1 <- y - i  dx2 <- x - i  dy2 <- y + i  IF m\_ppMap[dx1][dy1] == 3 - mid || m\_ppMap[dx2][dy2] == 3 – mid do    count[2]--  IF count[2] <= 0 do  count[2] <- 0  break  End IF  IF m\_ppMap[dx1][dy1] == mid && left == FALSE && dx1 >= 0 && dx1 < MAPSIZE && dy1 >= 0 && dy1 < MAPSIZE do    count[2]++  else do  left <- TRUE  End IF  IF m\_ppMap[dx2][dy2] == mid && left == FALSE && dx2 >= 0 && dx2 < MAPSIZE && dy2 >= 0 && dy2 < MAPSIZE do    count[2]++  else do  right <- TRUE  End IF  End FOR  left <- FALSE  right <- FALSE  //우상향 대각선  FOR i in range(1,8)  IF right == TRUE && left == TRUE do  break  dx1 <- x + i  dy1 <- y + i  dx2 <- x - i  dy2 <- y - i  IF m\_ppMap[dx1][dy1] == 3 - mid || m\_ppMap[dx2][dy2] == 3 – mid do    count[3]--  IF count[3] <= 0 do  count[3] <- 0  break  End IF  IF m\_ppMap[dx1][dy1] == mid && left == FALSE && dx1 >= 0 && dx1 < MAPSIZE && dy1 >= 0 && dy1 < MAPSIZE do    count[3]++  else do  left <- TRUE  End IF  IF m\_ppMap[dx2][dy2] == mid && left == FALSE && dx2 >= 0 && dx2 < MAPSIZE && dy2 >= 0 && dy2 < MAPSIZE do    count[3]++  else do  right <- TRUE  End IF  End FOR  left <- FALSE  right <- FALSE  FOR I in range(0,3)  IF Max <= count[i] do  Max <- count[i]  End IF  End FOR  p <- Max / 4  RETURN p |

GetCheckWinRate는 특정 색의 돌이 특정한 자리에 놓였을 때 그 색에 해당하는 플레이어가 승리할 가능성을 계산하는 함수이다. (가로, 세로, 대각선) 총 네 방향의 같은 색의 돌의 개수를 세며 그 개수가 가장 많은 라인을 통해 결과값을 도출하게 된다. 같은 라인에 상대의 돌이 있으면 그 결과값이 조금 작아지도록 만들었다. 대체로 자신의 돌 1개마다 결과값이 0.25씩 증가하게 된다.예를 들어, 4개가 연달아 있다면 대체로 1을 반환할 것이다.