Points System

0 means that both the rabbit and the wolves have equal chances to win from the current board state.

positive number means that rabbit has advantage from current state.

negative number means that wolves have advantage from current state.

Actions System

the ai will have functions. Each function returns a number. every function checks a specific thing in the current board state. if it returns positive number, rabbit got advantage. negative - wolfes got advantage. zero - nothing.

not all functions will be able to return both positive and negative values because sometimes actions maybe take advantage from one side, but don’t give to the other side.

First AI - lets start simple

functions:

a) rabbit gets 1 point for each direction he can move(for example if near wall will get only 2 points, in center 4 points).

b) rabbit will get points for each open hole that he can pass threw the wolves. we should include distance from the hole in the calculations. possible example is if rabbit already 1 cell from the hole that he can pass the wolves threw he will get 15 points, and if 5 cells from it only 2.(its just for example).

I want the amount of points rabbit gets to be reduced polynomial with the distance. Lets start with implementing the formula:

if(rabbit above hole) return 0

SOME\_CONSTANT = 15

POINT\_REDUCE\_PER\_CELL = 3

float points = SOME\_CONSTANT - distance\_from\_hole \* POINT\_REDUCE\_PER\_CELL

if(points < 3) points = 3

return points

end

AI efficiency

lets talk about how many steps forward we should look for. in each step its or wolf or rabbit, in worst case rabbit have 4 options(bot left, bot right, top left, top right) and wolves 8 options(2 for each of the 4). so number of nodes is:

8 \* 4 \* 8 \* 4 \* 8 … = somewhere around 8^(n/2) \* 4^(n/2)

when n is number of turns we look forward.

8^(n/2) \* 4^(n/2) = (2^3)^(n/2) \* (2^2)^(n/2) = (2^5)^(n/2) = 2^(2.5n)

lets try to start so we will have less then 100k nodes:

2^(2.5n)=100,000

2.5n = log2(100,000)

2.5n = 16.609640474 suppose 16

2.5n = 16

n = 6.4 suppose 6

so we will start by looking 6 steps forward.

algorithms

to check \*a\* is easy complexity O(1) - just check 4 directions.

\*b\* is problematic a bit. first lets talk about what holes are, give them formal definitions.

**Hole - the most bottom cell of each column where if rabbit were located now he would win - if he were located there in the current turn wolves would have no opportunity to block him anymore in the future.**

and how do we check this? i don’t know right now. the best thing i can think of is that we can recursively find all ways to the last row by running function from current rabbits cell, and the hole will be the closest cell to rabbit with only one recursive path going forward. but it wont work in all cases. so for AI one i will implement only the first simple option.

demo virtual environment:

we will want to make tests and steps while building and iterating the decision tree, but we cant do it with the main cells array and other references to objects. we will create a copy of them. the problem is if we create a copy for each node of the board we will run out of memory, so i will try to create one copy of objects and use the same ones in all the ai iterations.

**AI V-1.0:**

\* rabbit gets 1 point for each direction he can move(for example if near wall will get only 2 points, in center 4 points).