Finding Optimal Policy in Grid World Problem Using TDLearning Reinforcment learning project 2

Technical Report · July 2018

CITATIONS
0

READS
24

1 author:

Sadra Hemmati
Michigan Technological University
17 PUBLICATIONS **1** CITATION

SEE PROFILE

“Finding Optimal Policy in Grid World Problem Using TD-Learning with Eligibility Traces”

Project 2 of the Reinforcement Learning course

Sadra Hemmati

Fall 2015

University of Tennessee, Knoxville

Abstract:

In this project, we implement Temporal Difference Learning (TD(λ)) to enable an agent to learn the shortest path from any start position to the goal in a maze. Specifically, we implement SARSA(λ) algorithm that utilizes eligibility traces to improve the rate of learning for the agent. In our result section, we show how the agent finds the optimal path and the values of the eligibility traces as the agent gets closer to the path. The promising performance of this algorithm gives a measure of strength of this learning paradigm in the specific problem of grid world.

Introduction and Background:

Introduction:

Eligibility traces are one of the basic mechanisms of reinforcement learning. There are two ways to view eligibility traces: One is that these methods are filling the gap from TD to Monte Carlo methods(forward view) and the other is seeing it as a temporary record of an event such as visiting a state or taking an action.

Eligibility traces can be combined with any TD learning method to build more efficient algorithms than other TD-based 01-step method.They are methods that help relate events with state-action pairs and assign rewards to the state-actions depending on the distance they have with the goal state.

Problem Definition:

The problem in this project is defined as the agent in the environment tries to maximize its rewards by reaching the goal state. Here a rectangular grid is used to represent the value functions for a simple finite MDP. Each cell of the grid corresponds to a state and set of possible actions for the agent at each state are defined as: North, South, West and East. After choosing an specific action, the agent deterministically moves one cell in the respective direction on the grid unless the action transfers the agent to outside of the grid or into a wall in which case the location is unchanged. If the agent reaches the goal state, a positive reward of 1 is given to the agent.

We chose to implement Tabular Sarsa(λ) with the addition of 10% exploration as a simple and robust approach to the problem.

In this project we build a reasonably complex grid world of size 20*20 in the excel software and transfer it to MATLAB software. A fixed goal is set up for the mazes and the starts are chosen in a random fashion.An episode is defined as the time in which the agents reaches from the start state to the terminal state.Reaching the goal state has a +1 reward while other states have a zero reward.

At first we made some mazes which depending on the starting point, could form various optimal paths hence making the problem more challenging. Therefore we come up with another set of mazes which have a specific solution ensuring that an optimal policy exists. We added blind alleys to the maze in order to increase the complexity of the maze.

Design:

Design Objective:

Here the high level goal is to implement the Sarsa learning algorithm for the Grid World problem. In order to achieve this, some requirements should be fulfilled which are the design parameters. We summarise these parameters as following:

States:

The state space is represented by a 20 by 20 array. This array includes the wall states but they are never visited. A function was created to determine the next state given a state action pair.

Actions:

As mentioned previously, we have defined four actions for the agent at each step keeping in mind that actions that lead to a wall are not allowed execution. In the table below, the current and next states change in array value after taking different actions is shown below:

State at time t	State at time t+1 after taking action left	State at time t+1 after taking action right	State at time t+1 after taking action up	State at time t+1 after taking action down
(row,col)	(row,col-1)	(row,col+1)	(row+1,col)	(row-1,col)

State at time t	State at time t+1 if action leads outside the maze or into a wall
(row,col)	(row,col)

Rewards:

As explained before, the agent receives a reward of 1 upon reaching the goal and zero for other actions that do not result in reaching the goal.

Implementation:

We implemented SARSA(λ) algorithm based on the Sutton's book and recreated the example following the algorithm description in the book(figure 1). The results of the evolution for the agent with a gamma of 0.9 is shown in figure 2.

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
  $E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
  Initialize S, A
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
    $E(S, A) \leftarrow E(S, A) + 1$ (accumulating traces)
    or $E(S, A) \leftarrow (1 - \alpha)E(S, A) + 1$ (dutch traces)
    or $E(S, A) \leftarrow 1$ (replacing traces)
    For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
      $Q(s, a) \leftarrow Q(s, a) + \alpha\delta E(s, a)$
      $E(s, a) \leftarrow \gamma\lambda E(s, a)$
    $S \leftarrow S'$; $A \leftarrow A'$
  until S is terminal

Figure 7.13: Tabular Sarsa(λ).

Example 7.2: Traces in Gridworld The use of eligibility traces can substantially increase the efficiency of control algorithms. The reason for this is illustrated by the gridworld example in Figure 7.14. The first panel shows the path taken by an agent in a single episode, ending at a location of high reward, marked by the *. In this example the values were all initially 0, and all rewards were zero except for a positive

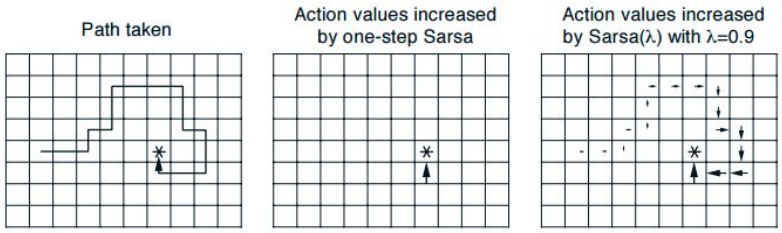
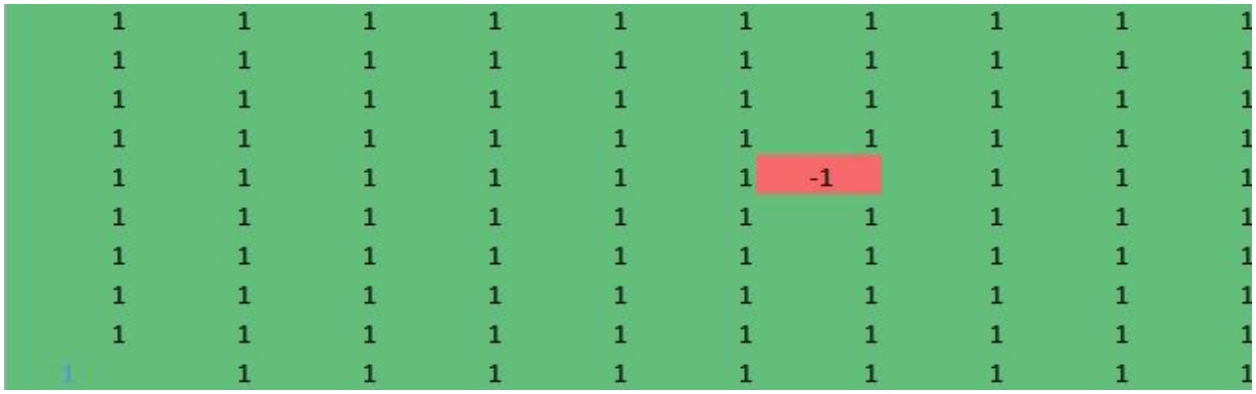


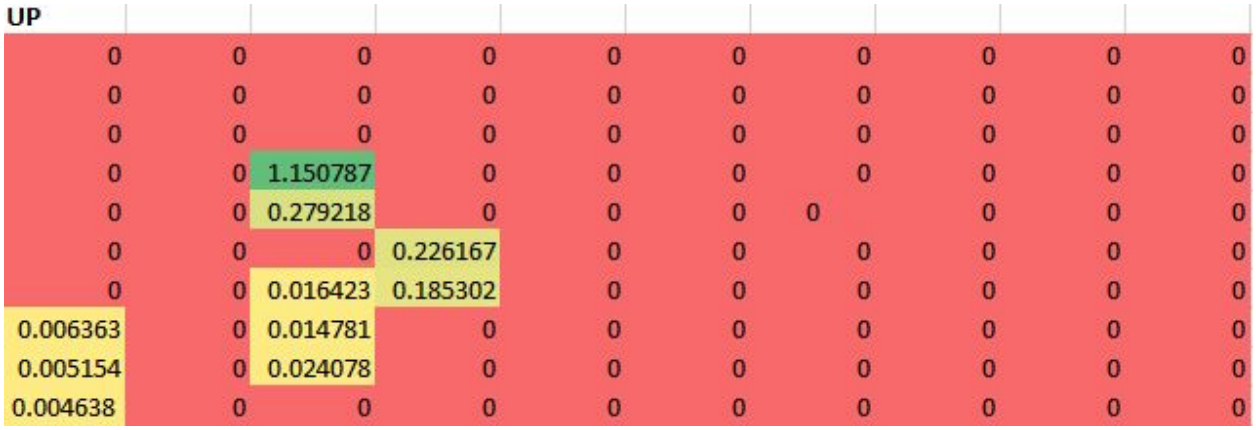
Figure 7.14: Gridworld example of the speedup of policy learning due to the use of eligibility traces.

Reference 1



Initial Trace Example

This example depicts the evolution of Q values in an empty maze as the agent takes different actions and learns by TD method with accumulating eligibility traces. For example in the picture with UP identifier, the state-action values of cells that the agent performs the action of going up has a non-zero value while the other values are zero. As the agent gets closer to the goal, the values increases which is in line with the observation that the 4th row has the highest value since it is closer to the goal state vertically. Similar explanations can be considered for the other actions(down, left, right).



RIGHT

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0.517139	0.147571	0	0	0	0	0	0	0
0	0.313811	0.6561	0.729	0.81	0.9	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0.018248	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0.007855	0	0	0	0	0	0	0	0	0
0	0.009698	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

DOWN

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1.287356	0.121577	0	0	0	0	0	0
0	0	0	0.135085	0	0	1	0	0	0
0	0	0	0.150095	0	0	0	0	0	0
0	0	0	0.166772	0	0	0	0	0	0
0.00707	0	0	0	0	0	0	0	0	0
0	0.008728	0.011973	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0.004175	0	0	0	0	0	0	0	0	0

LEFT

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0.465425	0.042391	0	0	0	0	0	0
0	0	0.28243	0	0	0	0	0	0	0
0	0	0	0.251296	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0.005726	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

AVERAGE*4									
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0.517139	1.900352	0.163968	0	0	0	0	0	0
0	0.313811	2.089316	0.864085	0.81	0.9	1	0	0	0
0	0	0.279218	0.401391	0	0	0	0	0	0
0	0	0.018248	0.392939	0	0	0	0	0	0
0.00707	0	0.016423	0.185302	0	0	0	0	0	0
0.019944	0.008728	0.026753	0	0	0	0	0	0	0
0.005154	0.009698	0.024078	0	0	0	0	0	0	0
0.008813	0	0	0	0	0	0	0	0	0

The above average effectively shows the full trace / path to the goal. This path can be seen by following the progression of the values It is noteworthy that in this figure, the average of cell (4,3) is more than the possible reward of 1. this is the effect of accumulation of traces such that bigger values are assigned to some states. Accumulation trace effects are addressed in challenge 1.

Design Challenges:

There were 3 main challenges when implementing Tabular Sarsa.

1. Accumulating traces leading to perpetually banging into walls.
2. The agent getting lost in endless spirals seeking greed gains
3. Defining lambda for optimal trace length

The first of these was the biggest challenge in running our algorithm because it lead to long run times. In instances where the agent took stationary actions right before reaching the end state, the eligibility of the stationary state action pair accumulated disportionately. This lead to future runs only taking this action and running in an infinite loop. This was solved by changing our eligibility tracing method to replacing traces instead of accumulation. We found this worked well for our simple problem statement

The second problem happened when the agent starts to effectively chase its own tail. This only happened on rare occasions with certain maze designs. This was solved by implementing a 1/10 random chance to take a random action instead of the calculated optimal. However, this problem is maze dependant, and was not an issue with our main maze shown in our result. In that maze, random actions were not used to optimize results.

The last problem was the most open ended. By setting lambda, you effectively determine the length of the traces tail. The main design goal is for it to be long enough to reach out of / into dead ends and optimally reach from any starting point accross the optimal path to the goal. However, this value is set specifically to the maze's unique properties. We decided on a value of .9 after trying to find a value that works robustly with the different kinds of mazes we designed even if it is not 100% optimal for each one.

Technical Approach:

We implemented the following algorithm:

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
$E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Initialize S, A
Repeat (for each step of episode):
Take action A, observe R, S' or rand A with 10% chance
Choose A' from S' using policy derived from Q (e.g., ε-greedy)
$\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
$E(S, A) \leftarrow 1$ (replacing traces)
For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow Q(s,a) + \alpha \delta E(s,a)$
$E(s,a) \leftarrow \gamma \lambda E(s,a)$
$S \leftarrow S'$; $A \leftarrow A'$
until S is terminal

Four scripts were used to implement this. These scripts can be found in the Appendix.

1. **TDLambda.m**

   This script does everything in the above algorithm. It is the master script that runs through all the iterations and stores all the values.

2. **rewardSmall.m**

   This script is a function that returns a reward value when passed a state action pair. It calculates this based on the current maze.

3. **nextStateSmall.m**

   This function returns the next state given a state action pair based on the current maze

4. **getDigiEgg.m**

This script automatically generates a maze based on a dimension (such as 4 by 4 or 20 by 20). It would be more useful for future potential application discussed in the summary.

As discussed earlier in challenge 3, lambda was determined to be 0.9. γ was set to allow lambda full control over the eligibility decay and to fully account for the future state action value because if the value isn't 0 then it for sure is on the path to the exit. Alpha was also set to 1 to fully incorporate delta and the eligibility. These values were found experimentally to solve the maze. For some mazes, adjusting alpha and gamma had an effect because of alternative paths but in all cases the trace length was found to be the most important factor.

The algorithm runs by constantly calculating delta based on the seen reward and the value of the next state. For the majority of the first run, these values remain zero until it hits the single reward for the first time. After that, the values of each station action are updated and the trace value is discounted.

These values could of been chosen with more thought, however the given Maze 1 below was solved after a single iteration so they are considered optimal.

One additional note is that for the results in Maze 1,running with 10% random exploration was not optimal and was not used. However, it was used in the additional results of Maze 2.

Experiments and Results:

Below shows our first full sized maze. The green represents the movable path and the red is the reward state. The cell values are arbitrarily set for matlab interpretation.

1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	-1
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Maze 1

This maze is very effectively solved by our algorithm. After the first completed run, it will optimally run for future starting values. This can be mainly attributed to the eligibility trace reaching the reward and propagating its value outward. This maze was also designed to be scalable and created automatically with an algorithm. In early testing, we ran smaller versions of the maze seen below:

1	1	0	-1
0	1	1	1
0	1	0	0
1	1	1	1

Maze 1 (4x4)

1	1	1	1	1	0	0	0	0	-1
0	0	0	0	1	1	1	1	1	1
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1

Maze 1 (10 x 10)

These smaller versions are useful because of how long the first initial run is.

E1

6.41E-133	1.49E-132	2.04E-132	2.80E-132	5.27E-132	1.22E-131	1.26E-129	3.30E-135	1.47E-126	3.41E-126	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4.21E-126	0.004638398	0.052334763	0.038152042	0.088629381	0.135085172	5.55E-05	0.254186583	0	0.729	0.9
0	0	0	0	0	0	0	0	0	4.43E-06	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3.99E-06	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3.23E-06	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	2.91E-06	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	2.12E-06	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1.72E-06	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	5.30E-08	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3.37E-10	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	8.58E-11	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	5.06E-11	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	2.99E-11	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	2.69E-11	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	6.16E-12	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4.99E-12	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4.49E-12	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4.04E-12	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	2.15E-12	0	0	0	0	0	0	0	0	0	0
1.65E-36	2.23E-37	4.26E-36	6.39E-37	1.98E-38	4.07E-39	4.38E-41	2.16E-39	1.86E-35	1.57E-12	2.57E-14	5.37E-14	1.01E-27	1.23E-14	8.96E-15	5.88E-15	8.05E-22	5.20E-23	2.49E-23	6.60E-83

Eligibility Traces for the moving up

E2

1.34E-132	1.84E-132	2.27E-132	7.23E-132	1.87E-131	7.45E-130	1.92E-129	2.14E-129	1.81E-126	5.20E-126	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	6.08E-06	0.005153775	0.064610819	0.071789799	0.109418989	0.150094635	0.228767925	0.59049	0.6561	0.81	0
0	0	0	0	0	0	0	0	0	1.66E-125	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1.10E-124	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	5.39E-07	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	6.65E-07	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1.91E-06	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1.39E-06	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4.77E-08	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	9.36E-122	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	2.08E-69	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1.59E-11	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1.16E-11	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	2.42E-11	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	5.54E-12	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	2.73E-17	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	6.74E-13	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4.16E-17	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4.42E-13	0	0	0	0	0	0	0	0	0	0
3.11E-36	3.83E-36	4.73E-36	5.26E-36	5.84E-36	6.49E-36	7.22E-36	2.89E-20	1.12E-13	5.70E-17	3.17E-14	4.35E-14	5.21E-16	1.09E-15	1.85E-15	3.47E-15	6.52E-22	9.64E-24	1.19E-23	1.47E-23

Eligibility Traces for the moving right

E3

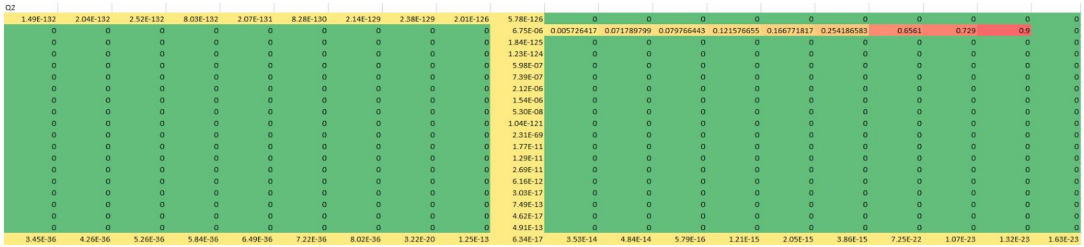
5.77E-133	1.65E-132	2.55E-138	6.51E-132	8.03E-132	6.71E-130	1.40E-129	4.40E-130	1.63E-126	5.78E-126	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1.34E-125	5.69E-127	0.058149737	0.034336838	0.09847709	0.000196627	4.50E-05	0.185302019	0	0.34867844	0
0	0	0	0	0	0	0	0	0	5.87E-125	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	2.86E-07	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	5.98E-07	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	7.39E-07	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1.01E-06	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4.30E-08	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	2.73E-10	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	7.72E-11	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	4.98E-67	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1.77E-11	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1.96E-11	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3.68E-18	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1.61E-17	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3.03E-17	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3.64E-12	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1.93E-12	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1.27E-12	0	0	0	0	0	0	0	0	0	0
2.04E-36	3.45E-36	3.40E-37	4.66E-37	1.97E-45	4.59E-38	3.72E-38	3.70E-59	1.01E-13	1.41E-12	2.86E-14	1.51E-28	1.52E-14	9.08E-21	9.96E-15	6.53E-15	2.81E-15	7.13E-23	2.24E-23	1.32E-23

Eligibility Traces for the moving down

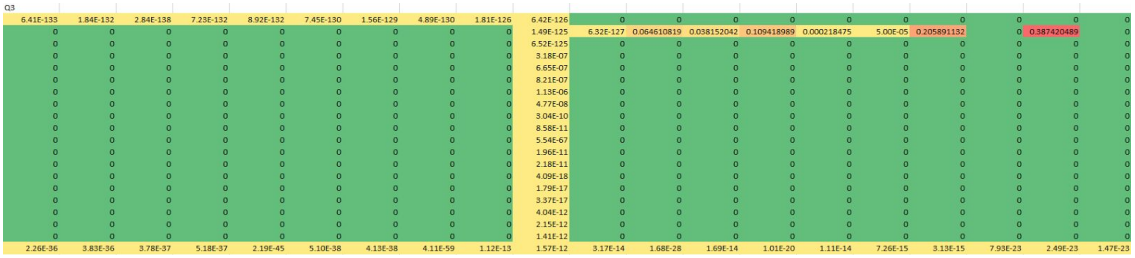
Q1

1.65E-132	2.27E-132	3.11E-132	5.85E-132	1.36E-131	1.40E-129	3.67E-135	1.63E-126	3.79E-126	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	4.68E-126	0.005153775	0.058149737	0.042391158	0.09847709	0.150094635	6.17E-05	0.282429536	0	0.81	1	
0	0	0	0	0	0	0	0	4.92E-06	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	4.43E-06	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	3.59E-06	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	3.23E-06	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	2.35E-06	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	1.91E-06	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	5.89E-08	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	3.75E-10	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	9.53E-11	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	5.63E-11	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	3.32E-11	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	2.99E-11	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	6.84E-12	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	5.54E-12	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	4.99E-12	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	4.49E-12	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	2.39E-12	0	0	0	0	0	0	0	0	0	0	
2.48E-37	4.73E-36	7.11E-37	2.20E-38	4.52E-39	4.87E-41	2.40E-39	2.07E-35	1.74E-12	2.86E-14	5.97E-14	1.12E-27	1.37E-14	9.96E-15	6.53E-15	8.94E-22	5.78E-23	2.76E-23	7.33E-83	

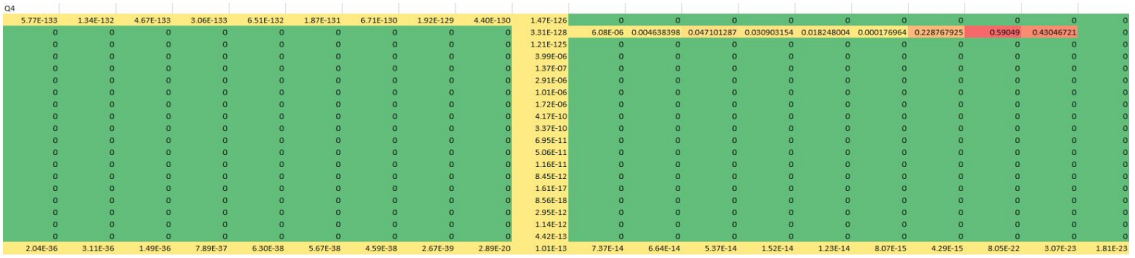
State-action values for the moving Up



State-action values for the moving right



State-action values for the moving down



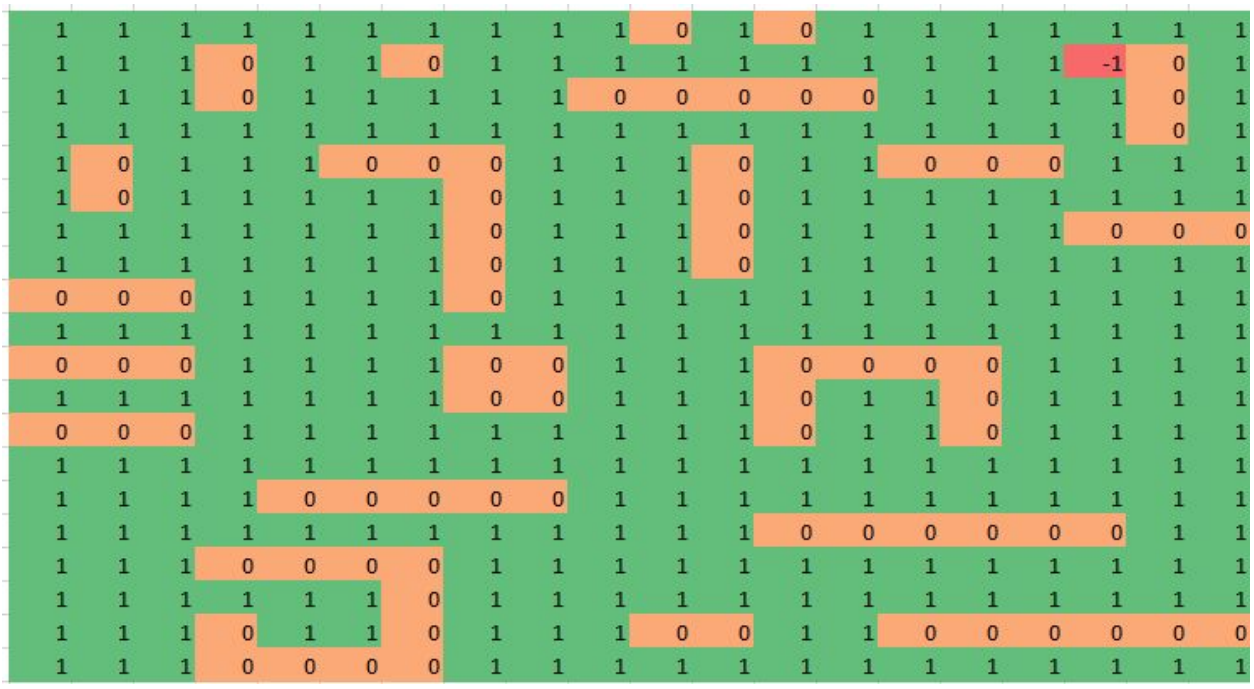
State-action values for the moving left

2	2	2	2	2	2	2	2	2	3	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	2	2	1
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	2	2	2	1	4	4	4	4	4	4	4	4	4	4

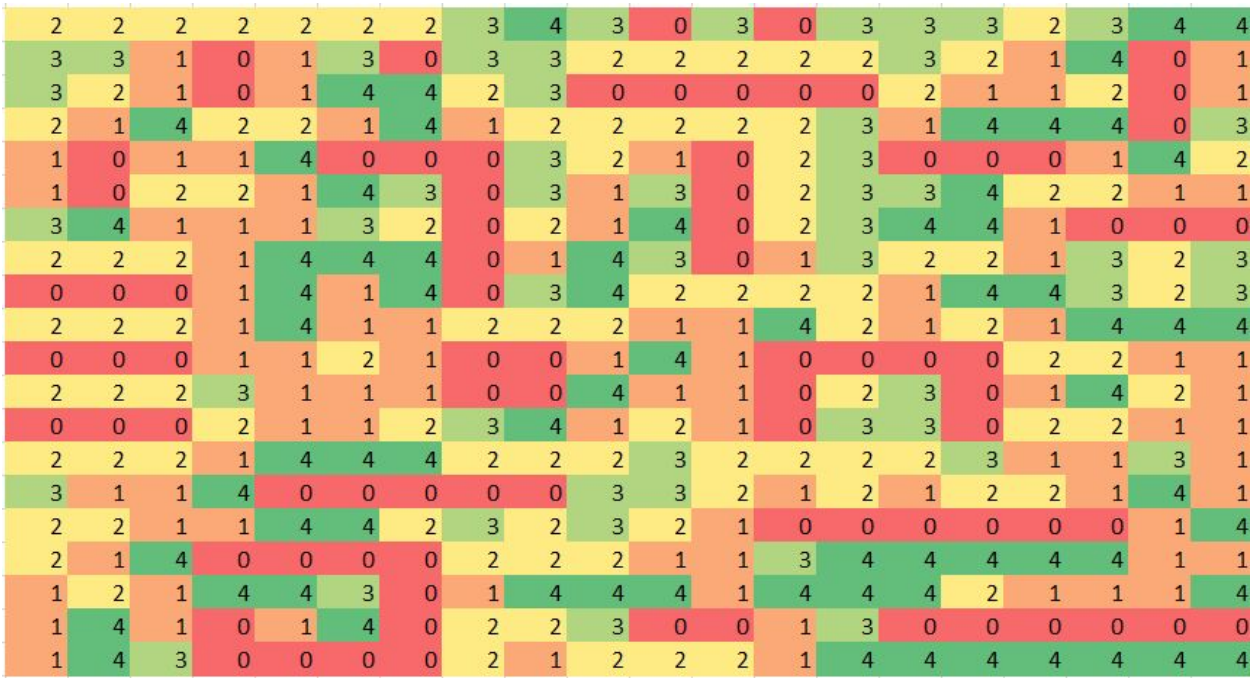
Maze 1 (20 x 20) results after 1 run

Above is a table of the optimal actions calculated after the first iteration of the maze. The actions correspond perfectly to the optimal path that should be taken (1 is up, 2 is right, 3 is down, 4 is left). Every further iteration with random location optimally goes to the goal.

Additionally, we ran our algorithm on other kinds of mazes. However, they do not run as optimally as the above results. They can be seen below:



Maze 2



Maze 2 results after 108 runs

Above it can be seen that the program solves many features of the maze. This can clearly be seen in the bottom right corner. Although we did not calculate it explicitly, the maze runs

functionally well (compared to random navigation). Each run is around 8 to 20 steps which is reasonably fast and moves clearly to the goal. This shows that our solution is robust enough to solve different types of mazes.

Summary

In this project, the formulation of the problem was the first step. In this stage the states and action-state value pairs were mapped to the cells in the maze. The rewards, actions, and factors that affect the learning were initialized in the code. The problem was first addressed by simply implementing eligibility trace TD in an open state space and then iteratively applying it to more complex problems until it fully fulfilled the design scope. Through this process, we learned about many aspects of applied temporal difference learning.

Ultimately, in this project we learned how to implement TD learning with eligibility traces and how to interpret the impact of different parameters (learning, discount, eligibility, etc) on the outcome of the simulation. This knowledge let us overcome a the difficult challenge of not having negative rewards.

Future work could be done in further optimizing the hyper variables to make the most optimal program. With the scalable nature of Maze 1, it would be interesting to scale up the optimal policy with the size of the maze to save on compute time.

Appendix A:

Matlab Code:

TDLambda.m -

```
%Initialize matrixes
clc
clear all
close all
global mazesize
%2 1 2 1 best, avg 9.2
% mazesize = 20;
% maze = getDigiEgg(mazesize);
mazesize = 20;
global mazeOverride
mazeOverride = 1;
global override
override = -2;
%maze = getDigiEgg(mazesize);
maze = getDigiEgg(-2);
exploration = .9;

optSeed = [optA,optG,optL];
oldmaxavgResultVars = zeros(1,3);
maxavgResultVars = ones(1,3);
startRow = mazesize;
startCol = 1;
runs = 1000;
global run;



%Set goalRow and goalCol
for mazerow = 1:mazesize(1)
    for mazecol = 1:mazesize(1)
        if(maze(mazerow,mazecol) == -1)
            goalRow = mazerow;
            goalCol = mazecol;
        end
    end
end

%might need this up here maxavgResultVars = ones(1,3);


Q = zeros(mazesize(1),mazesize(1),4);
E = zeros(mazesize(1),mazesize(1),4);

row = startRow;
col = startCol;
nextRow= startRow;
nextCol= startCol;
maxLimit = 10000;

%New starting position
rowStarts = zeros(1,runs);
colStarts = zeros(1,runs);

for(ij = 1:runs)
```

```
    newRow = randi(mazesize(1),1);
    newCol = randi(mazesize(1),1);
    while (maze(newRow,newCol) == 0 || maze(newRow,newCol) == -1)
        newRow = randi(mazesize(1),1);
        newCol = randi(mazesize(1),1);
    end
    rowStarts(ij) = newRow;
    colStarts(ij) = newCol;
end



moves = zeros(1,runs);
limit = 0
%Run Temporal code
for run = 1:runs
    limit = limit + 1
    limit = 0;
    E = zeros(mazesize(1),mazesize(1),4); %reset E
    moves(run) = 0;
    while ((~(row == goalRow && col == goalCol)) && (limit <maxLimit))
        limit = limit + 1;
        moves(run) = moves(run) + 1;
        %Finds optimal action, picks random if tie
        temp = Q(row,col,:);
        temp2 = find(temp==(max(max(temp))));

        optimalA = temp2(randi([1 length(temp2)],1));
        %if(rand(1) > exploration)
        %    optimalA = (randi([1 4],1))
        %end
        nextReward = rewardSmall(row,col,optimalA);
        [nextRow, nextCol] = nextStateSmall( row,col,optimalA);

        temp = Q(nextRow,nextCol,:);
        temp2 = find(temp==(max(max(temp))));
        optimalAprime = temp2(randi([1 length(temp2)],1));

        delta = nextReward + gamma * Q(nextRow,nextCol,optimalAprime) - Q(row,col,optimalA);
        %E(row,col,optimalA) = E(row,col,optimalA) + 1;
        %E(row,col,optimalA) = (1-alpha)*E(row,col,optimalA) + 1
        E(row,col,optimalA) = 1;
        avgE = E(:,:,1) + E(:,:,2) + E(:,:,3) + E(:,:,4)
        if(limit > maxLimit*.9)
            pause = 1;
            disp(row)
            disp(col)
            disp(optimalA)
            disp(Q(row,col,optimalA))
            disp(optimalAprime)
            disp(Q(nextRow,nextCol,optimalAprime))
        end


        for staterow = 1:mazesize(1)
            for statecol = 1:mazesize(1)
                for action = 1:4
                                                Q(staterow,statecol,action)  =  Q(staterow,statecol,action)  +
alpha*delta*E(staterow,statecol,action);
                    E(staterow,statecol,action) = gamma * lambda * E(staterow,statecol,action);
                end
            end
        end

        row = nextRow;
        col = nextCol;
```

```
    end

    if(limit == maxLimit)
        for iii = run:runs
            moves(iii) = maxLimit*100;
        end

    end

    row = rowStarts(run);
    col = colStarts(run);
end
```

rewardSmall.m -

```
function [ myReward ] = rewardSmall( row,col,action)


nextRow = row;
    nextCol = col;

standStillReward = 0;
goalReward = 1;
% maze = [1 1   0   -1;
% 0 1   1   1;
% 0 1   0   0;
% 1 1   1   1];
global mazesize
global mazeOverride
global override
if(mazeOverride ~= 1)
maze = getDigiEgg(mazesize);
else
maze = getDigiEgg(override);
end
%maze = getDigiEgg(-1); %temp


%1 up, 2 right, 3 down, 4 left
if(action == 1)
    nextRow = nextRow - 1;
end
if(action == 2)
    nextCol = nextCol + 1;
end
if(action == 3)
    nextRow = nextRow + 1;
end
if(action == 4)
    nextCol = nextCol - 1;
end


if(nextRow > mazesize(1) || nextRow < 1 || nextCol > mazesize(1) || nextCol < 1)
    %Illegial move, stand still
    myReward = standStillReward;
elseif(maze(nextRow, nextCol) == 0)
    myReward = standStillReward;
elseif(maze(nextRow, nextCol) == -1)
    myReward = goalReward;
else
    myReward = 0;
end


end
```

nextStateSmall.m -

```
function [ finalNextRow, finalNextCol ] = nextStateSmall( row,col,action)
    nextRow = row;
    nextCol = col;


% maze = [1 1   0  -1;
%         0 1   1   1;
%         0 1   0   0;
%         1 1   1   1];
global mazesize
global mazeOverride
global override
if(mazeOverride ~= 1)
maze = getDigiEgg(mazesize);
else
maze = getDigiEgg(override);
end
%maze = getDigiEgg(-1); %temp
%1 up, 2 right, 3 down, 4 left
if(action == 1)
    nextRow = nextRow - 1;
end
if(action == 2)

    nextCol = nextCol + 1;
end
if(action == 3)
    nextRow = nextRow + 1;
end
if(action == 4)
    nextCol = nextCol - 1;
end


    if(nextRow > mazesize(1) || nextRow < 1 || nextCol > mazesize(1) || nextCol < 1)
        %Illegial move, stand still
        finalNextRow = row;
        finalNextCol = col;
    elseif(maze(nextRow, nextCol) == 0)
        finalNextRow = row;
        finalNextCol = col;
    else
        finalNextRow = nextRow;
        finalNextCol = nextCol;
    end


end
```

getDigiEgg.m -

```
function [ matrix ] = getDigiEgg(evolution)
%     if(evolution == 1)
%         maze = [1 1   0  -1;
%                 0 1   1   1;
%                 0 1   0   0;
%                 1 1   1   1];
%     end
    maze = zeros(evolution,evolution);
    %floor(evolution)
    %bottom slice
    if(evolution>0)
    for(filler = 1:evolution)
        maze(evolution, filler) = 1;
    end
        %collumn
        for(filler = 1:evolution)
            maze(filler, floor(evolution/2)) = 1;
        end
            %top
            for(filler = 1:floor(evolution/2))
                maze(1, filler) = 1;
            end
                %end branch
                for(filler = 1:ceil(evolution/2))
                    maze(2, evolution + 1 - filler) = 1;
                end

                    %goal
                    maze(1, evolution) = -1;
    end
    if(evolution==-1)
    maze = [1   1   1   1   1   1   1   1   1   1;
    1   1   1   1   1   1   1   1   1   1;
    1   1   1   1   1   1   1   1   1   1;
    1   1   1   1   1   1   1   1   1   1;
    1   1   1   1   1   1   -1  1   1   1;
    1   1   1   1   1   1   1   1   1   1;
    1   1   1   1   1   1   1   1   1   1;
    1   1   1   1   1   1   1   1   1   1;
    1   1   1   1   1   1   1   1   1   1;
    1   1   1   1   1   1   1   1   1   1];
    end
    if(evolution==-2)
    maze = [1   1   1   1   1   1   1   1   1   1   0   1   0   1   1   1   1   1   1   1;
    1   1   1   0   1   1   0   1   1   1   1   1   1   1   1   1   1   1   -1  0   1;
    1   1   1   0   1   1   1   1   1   0   0   0   0   0   1   1   1   1   0   1;
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   0   1;
    1   0   1   1   1   0   0   0   1   1   1   0   1   1   0   0   0   1   1   1;
    1   0   1   1   1   1   1   0   1   1   1   0   1   1   1   1   1   1   1   1;
    1   1   1   1   1   1   1   0   1   1   1   0   1   1   1   1   1   0   0   0;
    1   1   1   1   1   1   1   0   1   1   1   0   1   1   1   1   1   1   1   1;
    0   0   0   1   1   1   1   0   1   1   1   1   1   1   1   1   1   1   1   1;
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1;
    0   0   0   1   1   1   1   0   0   1   1   1   0   0   0   0   1   1   1   1;
    1   1   1   1   1   1   1   0   0   1   1   1   0   1   1   0   1   1   1   1;
    0   0   0   1   1   1   1   1   1   1   1   1   1   0   1   1   0   1   1   1   1;
    1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1;
    1   1   1   1   0   0   0   0   0   1   1   1   1   1   1   1   1   1   1   1   1;
    1   1   1   1   1   1   1   1   1   1   1   1   1   0   0   0   0   0   0   1   1;
    1   1   1   0   0   0   0   1   1   1   1   1   1   1   1   1   1   1   1   1   1;
    1   1   1   1   1   1   0   1   1   1   1   1   1   1   1   1   1   1   1   1   1;
    1   1   1   0   1   1   0   1   1   1   0   0   1   1   0   0   0   0   0   0   0;
    1   1   1   0   0   0   0   1   1   1   1   1   1   1   1   1   1   1   1   1   1];
    end
```

```
    if(evolution==-3)
    maze=[1 1 1 1   1   1   1   1   1   1   0   1   1   1   1   1   1   1   1  1;
    1   1   1   1   1   1   1   1   1   1   0   1   1   1   1   1   1   1   -1  1   1;
    1   1   1   1   1   1   1   1   1   1   0   0   0   0   0   0   0   0   1   1   1;
    1   1   1   1   1   1   1   1   1   1   0   1   1   1   1   1   1   0   1   1   1;
    1   1   1   1   0   0   0   0   0   0   0   1   1   1   1   1   1   0   1   1   1;
    1   1   1   1   1   1   1   1   1   1   1   0   1   1   1   1   1   0   1   1   1;
    1   1   1   1   1   1   1   1   1   1   1   0   1   1   1   1   1   0   0   1   1   1;
    1   0   0   0   0   0   1   0   0   0   0   0   0   1   1   1   1   0   1   1   1   1;
    1   1   1   1   1   1   1   0   1   1   1   1   1   1   1   1   1   0   1   1   1   1;
    1   0   1   1   1   1   1   0   1   1   1   1   1   1   1   1   1   0   1   1   1   1;
    1   0   1   1   1   1   1   0   1   1   1   1   1   1   1   1   1   0   1   1   1   1;
    1   0   1   0   0   0   0   0   1   1   1   1   1   1   0   1   1   0   1   1   1   1;
    1   0   1   1   1   1   1   1   1   0   0   0   1   0   1   1   1   0   1   1   1   1;
    1   0   1   1   1   1   1   1   1   1   1   1   0   1   0   1   1   0   1   1   1   1;
    1   0   1   0   0   0   0   1   1   1   1   1   0   1   0   1   1   1   0   0   0   1;
    1   0   1   1   0   1   1   1   1   1   1   0   1   1   1   1   1   1   1   0   1   1;
    1   0   1   1   0   1   1   1   1   1   0   1   0   0   0   0   1   0   1   1;
    1   0   1   1   0   1   1   1   1   1   1   1   1   1   1   1   1   1   0   1   1;
    1   0   1   1   0   1   1   1   1   1   1   1   1   1   1   1   1   1   0   1   1;
    1   1   1   1   0   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1];
    end
    if(evolution==-4)
maze = [1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   -1  0   0;
    1   0   0   1   1   1   1   1   1   1   0   1   1   1   1   1   0   1   0   0;
    1   0   0   1   0   0   1   0   0   1   0   1   0   0   0   1   0   1   0   0;
    1   1   1   1   0   0   1   0   0   0   0   1   0   0   0   1   0   1   1   1;
    0   0   0   1   0   0   1   1   1   1   0   1   0   0   1   1   0   0   0   1;
    0   0   0   1   0   0   0   0   0   1   0   1   1   0   1   0   0   0   0   1;
    0   0   0   1   0   1   0   0   0   1   0   0   1   0   1   0   1   1   1   1;
    0   1   1   1   0   1   1   1   0   1   0   0   1   0   1   0   0   0   0   1;
    0   1   0   1   0   0   0   1   0   1   0   0   1   0   1   1   1   1   1   1;
    0   1   0   1   1   0   0   1   0   1   1   1   1   0   0   0   0   0   0   0;
    0   1   0   0   1   1   0   1   0   0   0   0   0   0   0   0   1   1   0   0;
    0   0   0   0   0   1   1   1   0   0   1   0   1   1   1   1   1   0   0   0;
    0   0   1   1   0   0   1   0   0   0   1   0   1   0   0   0   0   0   1   1;
    0   0   1   0   0   0   1   0   0   1   1   1   1   0   0   0   0   0   1   0;
    0   0   1   1   1   1   1   0   0   0   0   0   1   1   1   1   1   0   1   0;
    0   0   0   0   0   0   1   1   1   0   0   0   1   0   0   0   1   0   1   0;
    0   1   1   1   1   0   0   0   1   0   0   0   0   0   0   0   1   1   1   0;
    0   1   0   0   1   0   1   0   1   1   1   1   1   1   1   0   1   0   0   0;
    0   1   0   0   1   0   1   0   0   0   0   0   0   0   1   1   1   0   0   0;
    1   1   0   0   1   1   1   1   1   1   1   1   1   1   1   0   0   0   0   0];
    end

    matrix = maze;
end
```