Q-Learning Report

Kouame Kouame
KSSOU001

04 November 2016

Contents

1-Pseudo-Code and Parameters

1.1- Pseudo-Code

1.1.1-Initialisation

```
void CQLearningController::InitializeLearningAlgorithm(void)
{	//for each sweeper, create a Q table of states (cell)
	for (int sweeper = 0; sweeper < CParams::iNumSweepers; ++sweeper) {
		//Use vector to avoid resource management
		std::vector<State> QTable;
		//each table has row * column states
		for (int i = 0; i < _grid_size_x*_grid_size_y; ++i) {
		//a state is a struct of int[4] for the 4 actions and an int for the last action
			State state = { {0, 0, 0, 0}, -1};
			QTable.push_back(state);
		}
		QTables.push_back(QTable);
	}
}
```

1.1.2-Reward Function

```
/**The immediate reward function. This computes a reward upon achieving the goal state of collecting all the
mines.*/
double CQLearningController::R(uint x,uint y, uint sweeper_no){
	//get index for the Q table
	int cur_cell_index = ((y / CParams::iGridCellDim) * _grid_size_x) + x / CParams::iGridCellDim;
	int maxQ_ = maxQ(sweeper_no, cur_cell_index); //get maxQ of the next state
	//see if it's found a mine
	int GrabHit = m_vecSweepers[sweeper_no]->CheckForObject(m_vecObjects, CParams::dMineScale);
	if (GrabHit < 0) return MOVEMENT_REWARD + GAMMA * maxQ_; //punish for finding nothing -1
	//we have discovered a mine so increase give 10 reward
	else if (m_vecObjects[GrabHit]->getType() == CDiscCollisionObject::Mine) {
		++mine; return MINE_REWARD + GAMMA * maxQ_;
	}
	//we have hit a supermine so punish with -100
	else if (m_vecObjects[GrabHit]->getType() == CDiscCollisionObject::SuperMine) {
		++mine; return SUPERMINE_REWARD + GAMMA * maxQ_;
	}
}
```

1.1.3-Q-Learning Core Algorithm

```
//core implementation of Q-Learning: removed unneeded
bool CQLearningController::Update(void)
{	//For each sweeper, get current position, then choose best action based on its experience
	for (uint sw = 0; sw < CParams::iNumSweepers; ++sw){
		SVector2D<int> position = m_vecSweepers[sw]->Position(); //1:::Observe the current state:
		//compute corresponding index of the Q table
		int cell_index = ((position.y / CParams::iGridCellDim) * _grid_size_x) + position.x /
		CParams::iGridCellDim;
		int action = bestAction(sw, cell_index); //Select action with highest historic
		m_vecSweepers[sw]->setRotation( (ROTATION_DIRECTION)action); //Set direction
		QTables[sw][cell_index].last_action = action; //Store action performed for future use
	}
	CDiscController::Update();  //now call the parents update, so all the sweepers fulfill their chosen action
```

```
    // or each sweeper, update the Q table based on the outcome of  the action
    for (uint sw = 0; sw < CParams::iNumSweepers; ++sw){
        SVector2D<int> prev_pos = m_vecSweepers[sw]->PrevPosition();//get previous position
        int prev_cell_index = ((prev_pos.y / CParams::iGridCellDim) * _grid_size_x) +
         prev_pos.x / CParams::iGridCellDim; //get previous index for the Q table
        SVector2D<int> position = m_vecSweepers[sw]->Position();//3:::Observe new state:
        int action = QTables[sw][prev_cell_index].last_action; //get stored action
        int reward_ = R(position.x, position.y, sw);  //get immediate reward
        QTables[sw][prev_cell_index].actions[action] = reward_; //Update _Q_s_a accordingly
    }
}
```

1.2- Parameters and Justification

Below are the defined parameters:

```
#define GAMMA 0.9   //The discount factor of the MaxQ value
#define MOVEMENT_REWARD -1 //The immediate reward for finding nothing
# MINE_REWARD 10 //The immediate reward after sweeping a mine
#define SUPERMINE_REWARD -100 //The immediate reward after hitting a supermine
```

The following clarifies why they were chosen as such.

- GAMMA was chosen to be 0.9 for two reasons.
    1. The goal state being collecting all the mines, the next action matters, hence the value being close to 1.
    2. 0.9 was chosen by simulating while varying GAMMA from 0 to 1. 0.9 was the optimal value for the three test environments.

- MOVEMENT_REWARD chosen as -1 to encourage exploration.

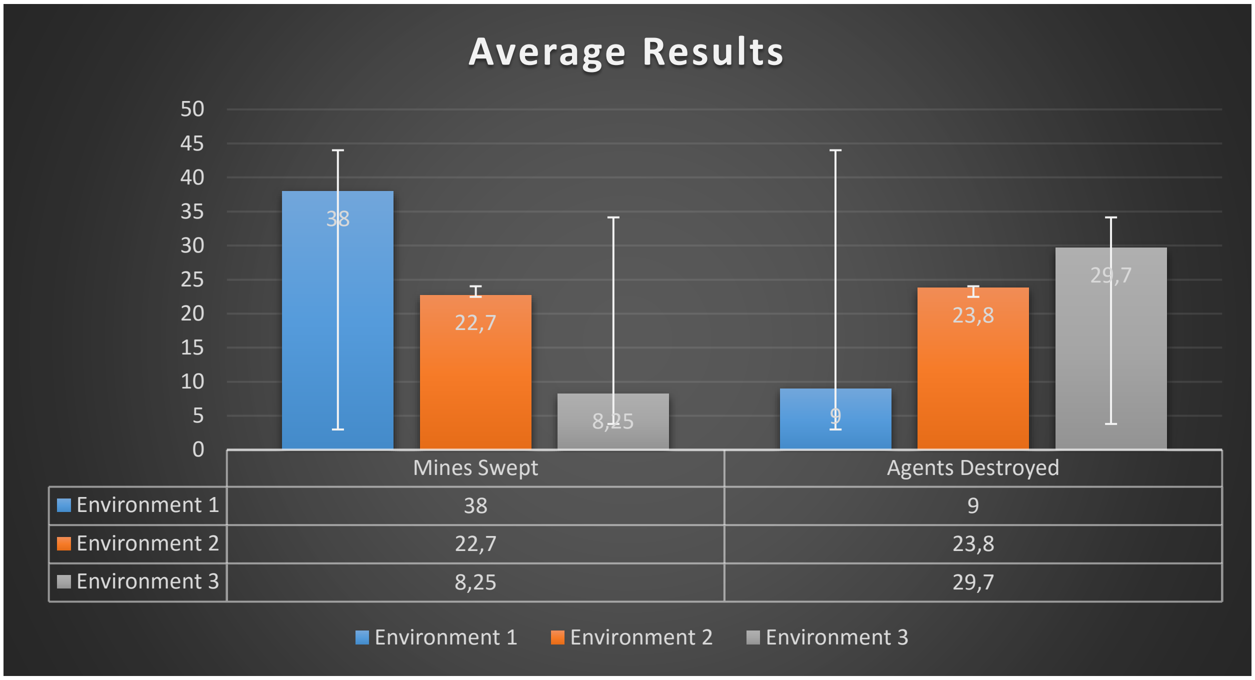
  The initial Q values are 0, punishing the agent for finding nothing, encourages exploration of unvisited cells. The small value of -1 favours an empty cell over a supermine cell with -100 reward.

- SUPERMINE_REWARD chosen as -100. Naturally, hitting a supermine should be punished enough to negatively reinforce learning. Any value smaller than -1 could perfectly do the job.

- MINE_REWARD chosen as 10 to encourage collection of mines. In fact, any positive value could have achieved the same goal.

2-Histogram and Discussion on Results

2.1-Histogram



Note that I used the number of mines swept, not the average. To get the average just divide each number by 30.

2.2-Discussion on Results

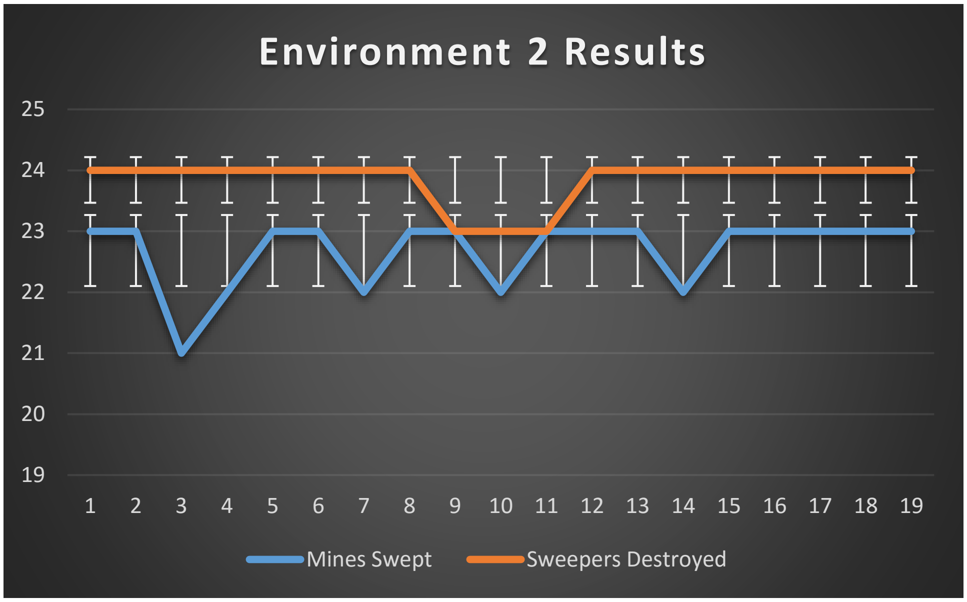
As general comment, the results of the 20 runs were fairly constant because, at every iteration, the mines and the agents are randomly re-placed in the grid. This makes it impossible to learn over the iterations since the Q table of the previous iteration does not reflect the current environment. So, the agents could not optimally make use of the past to reinforce their knowledge of the environment.

- Environment 1: 30 mine-sweepers, 40 mines and 10 super-mines

  On average, only 2 ± 0 mines were uncollected and 21 ± 0 agents stayed alive. This is because just 10 super-mines were available and the agents learnt to collect the mines which were in high density quickly. Note, that the standard deviation was 0 because the simulation resulted in a constant mines swept. However, the most mine gathered varied from 3 to 7. The graph below shows the results of the 20 runs.

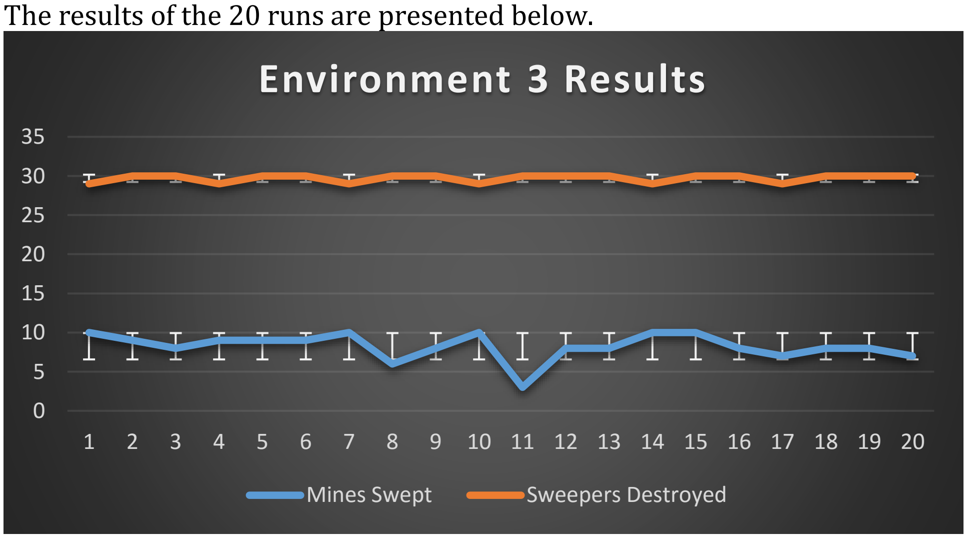
- Environment 2: 30 mine-sweepers, 25 mines and 25 super-mines

  Here, 1.2 ± 0.4 sweepers stayed alive on average but only 1 ± 0.6 mines were not collected.
  This is a fair results given that we had equal number of mines and super-mines.
  Below are the results of the 20 runs.



- Environment 3: 30 mine-sweepers, 10 mines and 40 super-mines

  This environment is very hostile because it hides 40 super-mines and only 10 mines.
  This explains why only 0.3 ± 0.4 agents stayed alive and 8.25 ± 1.6 were collected.
  The results of the 20 runs are presented below.



2.3- Improvements

- The environment should not change over the iteration, this makes the history useful. It furthermore, illustrates the concept of reinforcement learning better.
- Allow the agents to share information among themselves. This will shorten the exploration phase and make them more effective: an agent A will not have to visit a given cell that has already been visited by another agent B to have knowledge about it.