Analysis of Q Learning Performance on Water World

Ivan Remus
Faculty of Computer Science
Alexandru Ioan Cuza University of Iasi

***Abstract*—In this paper we show the policies that that a Double Deep Q Learning Agent is able to learn in the environment Water World as well as how it compares with a Random and Heuristic Agent.We show that the Q Learning Agents is able to come up with good policies and get better rewards than a Random agent as well as being capable of learning the optimal policy for the case of having only 1 circle in the environment and compete with the Heuristic Agent.**

I. Introduction

A. Problem

Water World is a learning environment comprised of a rectangle 2D map.In this map,we have our agent designated as a blue circle/molecule along with green and red circles.The task of the agent is to move around this map capturing as many green circles while avoiding red circles.After a circle is captured,another one will spawn at a random position and a random color.

The game is over when all the green circles have been captured or whether we have exceeded a threshold such as a maximum number of frames.

The agent can either move Left,Right,Up,Down or do nothing at each time step/frame.The game starts with the agents in the middle and circles spawned randomly on the map.

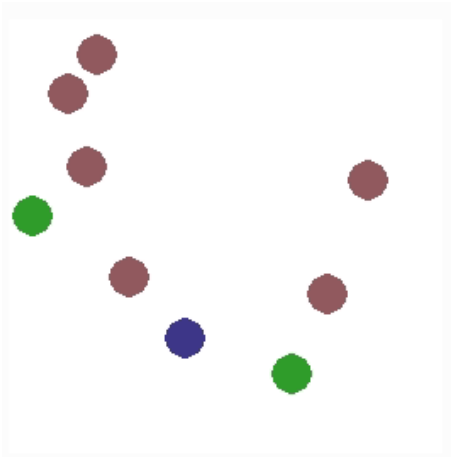

Fig. 1. Game State 1

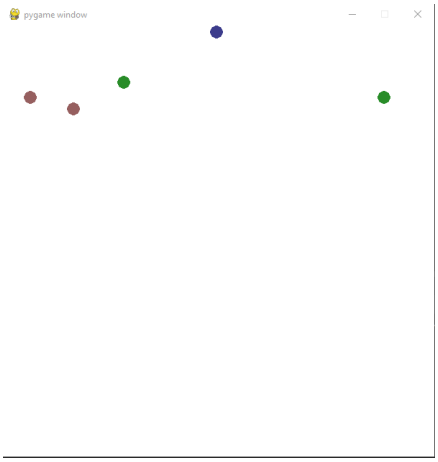

Fig. 2. Game State 2

B. Our Solution

We implemented a Double Deep Q Learning agent with Epsilon Greedy and Experience Replay Buffer using some state representations and episode definitions ending up with an agent that is able to learn the optimal policy when there is only one circle(follows the green circle and captures it) and some worse strategies for more complex environments,such as when we increase the number of circles.

II. Methods description

In this section we will describe the methods used and for the benchmark performance as well as other ideas that we're used.

A. State Representation

The environment has the internal game state representation as the players position,velocity and for each circle its position on the map.For the representation of the game state,the most convenient was the one that established for each circle the label,distance from the player and the angle of the circle consider the player at coordinates $(0, 0)$.

With the exception of always giving the players x and y position and the player x and y velocity we can have the following state representation:

1) Type 1:Represent each circle as (label,position x and y of circle,distance from player) with the green circles being placed the earliest and the red ones after

2) Type 2:Same as Type 1 but we with a randomized order

3) Type 3:Represent each circle as (label,angle of player towards this circle,distance) by having the green circles information given before the red ones.Intuitively,we expect this to help the agent associate its actions even better with the circles position.

4) Type 4:Same as type 3 where we randomize the order in order to further make the agent generalize.

5) Type 5:Use Type4 representation and the information about the circles is given non-decreasing based on the distance from the player.We expect this to make the agent put more focus the first closest circles without losing too much generalization.

6) Type 6:Use multiple previous states in order extract more information about the circles.For instance,the agent receives the last previous 2 states from which it can also extract every circles direction,which it can't tell from just one game state.

B. Game Proxy

In order allow for more episode definitions,we implemented a game proxy which allows us to use another episode definition without changing the code of the environment itself.This game proxy also keeps the history of a game as the previous game which is used for the Type 6 game state representation.

C. Episode Definition

We have 2 episode definition,which mostly refer to when the game is considered over(without time/frame limits):

1) Definition 1:The game ends when there are no more green circles

2) Definition 2:The game ends at the first captured circles,either red or green

The reason we are interested in another episode definition is because the agent sometimes gets stuck in the corners of the map or the agent is not yet good enough to capture all the green circles in order to end the game faster,which leads to an increased execution time.

We found that the second definition can significantly speed up the execution time while also offering the agents experiences that it can learn from.

D. Agents

For the agents we only allow the moves of Up,Down,Left and Right ,excluding the action of doing nothing.This is in order to force the Q Learning agent to also have to learn how to 'balance' its momentum.

1) Random Agent: This agent chooses at random between the 4 possible actions.Although it is simple,it can be observed to sometimes(but rarely) catch multiple green circle in succession as if it has a strategy,meaning the optimal policy for certain time steps is achievable by chance.

2) Heuristic Agent: This agent uses the Type 4 Game State Representation in order to use the angle between the agent and a circle and it works on the following 2 ideas:

- Go to the closest green circle in the 4 possible directions
- Go to the farthest red circle in order to avoid other red circles,when no green circle in sight

We give a range of degrees for each one of the 4 actions as:

- Up:$(90° - 45°, 90° + 45°)$
- Left:$(180° - 45°, 180° + 45°)$
- Down:$(270° - 45°, 270° + 45°)$
- Right:$(-45°, 45°)$

For each range,we find the closest circle to the player and remember its label.We choose the actions that gets us to the closest green circle.If no green circle is closest to the agent,we go in the direction of the farthest red circle in order to avoid the closer red circles.

Algorithm 1 Heuristic Agent

```
gameState=state
for move in moves do
    for circleInfo in circles do
        if circle position in range(move) then
            if circle is Green then
                Update closest green circle
            else
                Update farthest red circle
            end if
        end if
    end for
end for
if Found at least 1 green circle then
    Choose Move towards the closest green circle
else
    Choose Move towards farthest red circle
end if
```

3) Q Learning Agent: For the architecture of our model for the Q Learning Agent,we have 2 hidden layers of 512 and 256 units using Batch Norm and Leaky ReLU. As an optimizer we also used Adam and for some models we also used weight decay.The criterion used is Huber Loss.

Algorithm 2 Double Deep Q Learning Agent with Experience Replay and ϵ-greedy

Initialize Game Proxy
Initialize Game State Adaptor ϕ
Initialize Q Model and Q Target Model
Initialize Experience Replay Buffer D
for episode=1,noEpisodes **do**
    $s_1 = gameProxy.getGameState()$
    $\phi_1 = \phi(s_1)$
    for t=1,noMaxFrames **do**
        if currentFrame≥noRandomFrames **then**
            Select a_t at random
        else
            With probability $epsilon$ select action a_t
            Otherwise select $a_t = \underset{a}{max}\ Q(\phi_t, a)$
        end if
        Execute action a_t
        Get reward r_t and state s_{t+1}
        $s_t = s_{t+1}$
        $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
        if updateModel **then**
            Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
            Performan update of $QModel$ using gradient descent with prediction $Q(\phi_j)$ and target $r_j + \gamma \cdot \underset{a}{max} Q(\phi_{j+1}, a)$
        end if
        if updateTarget **then**
            Q Target=Q Model
        end if
    end for
end for

III. Benchmark performance

In this section we present the training metrics for the Q Learning models and the results of comparison with the other agents.

A. Training For Q Learning Agents

We define:

$$runningReward_i = 0.95 \cdot runningReward_{i-1} + 0.05 \cdot reward_i$$

For the agents A3T which used 4000 episodes to train but with Definition 2 this are the plots:

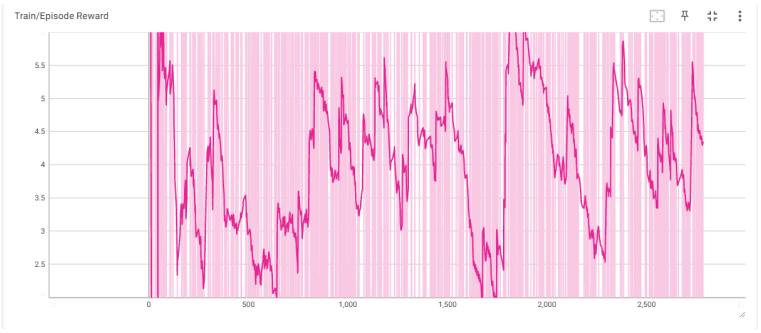

Fig. 3. Rewards per Episode

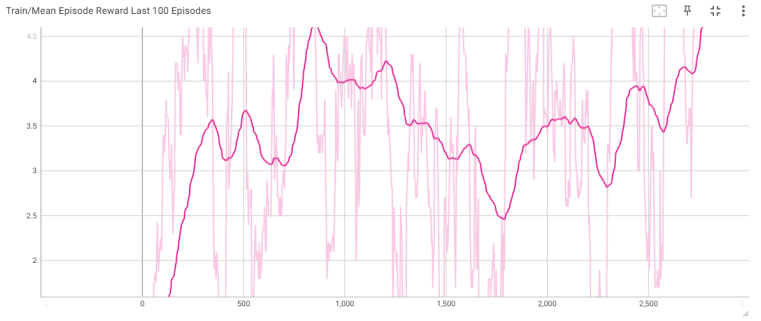

Fig. 4. Mean Average Over The Last 100 Episodes

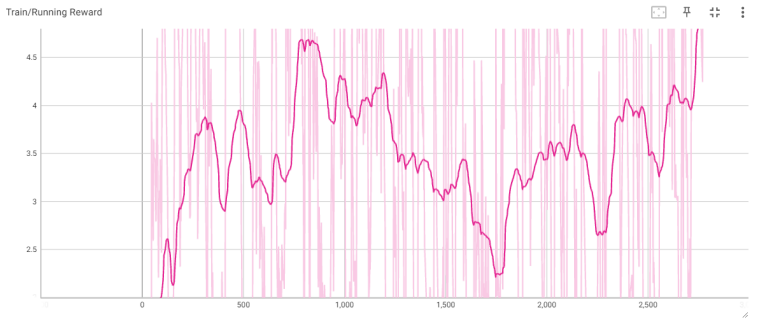

Fig. 5. Running Reward

Although its hard to see from the rewards per episode,the other 2 metrics tell us that after the random and epsilon greedy exploration is over,the agent slowly started to get better and better as it was developing its policy.

B. Performance Comparison

Firstly,we will describe some policies found by our agents:

- A1L manages to follow head right to the green circle in most occasions,but sometimes it just manages to stay close to it but not catch it,almost as if balancing
- A4L goes up and down in order to try to get closer to the circle,while also sometimes camping

- A6L tries to balance itself closer to the circle instead of running around

- Other strategies consist of either going randomly in order to catch any circle in the hope it is green or they go wait at an edge and go up and down when a circle gets close to the edge

We present the performance of the Random,Heuristic and some models with 1 ,2 and 4 circles. We set the width and height of the window to be 600x600 with the following rewards:

- Green Circle:10
- Red Circle:-10
- Win(no red circle remaining):100

The maximum number of frames per episode is 1000,so we expect the agents to perform their best in that number of frames.We set the number of episodes to be 250 and for each episode we run each seed for a fair comparison.

For the first table,the models that were trained used the episode definition 1 and their number of episodes in order is 1500,500,200,200 showing how the increase of the number of episodes also increases their performance.However,we can also see that A6L has better results than A4L ,which shows that we can find agents that learn fairly well with a smaller budget.This also shows the fact that policies obtained by Q Learning are very sensitive to the hyper parameters given.

Agent Name/No Frames	250	1000	2000
Random	6.56	31.12	49.76
Heuristic	114.08	120.88	120.48
Q Learning A1L	73.92	109.32	115.72
Q Learning A4L	6.96	36.2	57.4
Q Learning A6L	26.68	65.72	86.32
Q Learning A7L	6.28	23.52	46.72

TABLE I
RESULTS FOR 1 CIRCLE

Agent Name/No Frames	250	1000	2000
Random	7.32	28.12	36.68
Heuristic	85.28	122.24	126.32
Q Learning A8L	19.8	63.2	89.88
Q Learning A10L	18.24	59.64	91.28
Q Learning A14L	2.92	7.64	12.28
Q Learning A16L	11.76	28.56	58.36

TABLE II
RESULTS FOR 2 CIRCLE

Agent Name/No Frames	250	1000	2000
Random	1.6	4.4	20.52
Heuristic	59.32	116.8	130.90
Q Learning A1T	0.52	1.32	6.92
Q Learning A2T	2.76	6.44	16.08
Q Learning A3T	3.4	8.56	10.84
Q Learning A4F	1.6	4	13.36

TABLE III
RESULTS FOR 4 CIRCLE

from the table we can see that the Heuristic agent is the best when we increase the number of circles but for the number of 1 circle ,the Q Learning agent isn't far behind,which shows its potential to find even better policies with further improvements.The Q Learning agent is able to beat the Random Agent in all environments,showing that is is a better choice than randomly choosing actions and that it learned something.

IV. ABLATION STUDY

1) Tried Game State Representations: Besides the different state representation shown ,we also have a Type 6 which would describe the normalized/standardized game state according to each max value(for example,the angle $\in [0, 360]$).Although we used this Type 6 in combination with Type 4,it didn't show any improvements in the learning and even showed worse results for more difficult environment.

Also,the Type 5 representation also didn't yield any promising results,probably due to the agent not being able to learn that well even the optimal policy of chasing the circles.Thus,it struggled alot more to learn a strategy with this representation,even if it contains a lot more useful information.

2) Learning Rate Scheduler: Some models,usually the models that had a low number of episodes(200-700) were able to learn a strategy and converge when the epsilon was at around 0.2,but then started to diverge and started staying stuck in a corner of the map not doing anything.

From what i found,this is called Compounding Errors which happens due to Overfitting.In order to combat this,a learning rate scheduler StepRL and ExponentialRL was attempted which helped,but needed further ajustment to γ in order to stop and minimize the diverge of the agent instead of making the agent not learn anything.Due to this,no model presented uses it.

3) Experience Replay and 2 Q Models: Without Experience Replay to account for highly correlated states that the agent would learn and without trying to stabilize the Q network via using a current model for training and a target model,we found that models would perform significantly worse,even to the point of barely being able to learn anything.

4) Episode Definition: The second episode definition proved to be incredibly useful,since it speed up the execution time but made it harder for the agents to learn.Due to this,if we were to apply a better scheme for the Experience Replay than the one presented(that being,we eliminate the oldest pair of action and introduce the new one) might be able to improve the learning capablities of the agent.Even without this,the agent can still learn using the second definition,which is also possible due to how we can consider this definition as a subset of the first definition.

V. Future Work

A. Using The Heuristic Agent

Based on the results on performance,we could attempt to further improve the Q Learning agent by adding a new ϵ greedy policy.We establish ϵ_1 as the probability to take a random action and ϵ_2 as the probability to take an action made by the Heuristic Agent.In this way,we start the policy with the moves made by the random agent and slowly increase the moves made by the Heuristic Agent. After some time,we also decrease ϵ_1 allowing the agent to stop observing and learn by making its own actions.

B. Early Stopping and Learning Rate Scheduler

We could add a early stopping criteria given by the running reward when its value gets below a certain threshold in order to prevent the agent into overfitting.Of course,we can also use this in order to adjust the learning rate in order to make it more difficult to get stuck in a local optima ,as well as saving this model.

C. Experience Replay

Using a better policy for what experiences to keep such as having a lower or higher value obtained could also help improve the networks performance.Additionally,we could use a clustering algorithm in order to attempt to prioritize states that are 'rare' and have a higher reward value which should improve the exploitation of the agent in its training.

D. State Representation Clustering

A clustering algorithm could be used for the Experience Replay Buffer but it can also be used to potentially generate new episodes from states that are rare and directly influence the exploration of the agent.

We can also use this in order to train a model to cluster the states observed generated either by the Random Agent or the Heuristic Agent following up with starting the training with this model,which should also significantly help the agent learn faster.

E. Curriculum Learning

We can apply curriculum learning by simply starting with 1 circle and slowly adding more and more.To make the task easier at the beginning,we can also restrict the moves to left and right (or up and down) following to adding the other moves once it learns well on this easier task.Since the game state representations are similar(only adding more circles information) this makes it even easier to add.

VI. Conclusion

Q Learning methods show that it is possible to obtain better models than random actions as we increased the difficulty and they can be competitive if not better than heuristic methods on getting good scores on this environment which shows how it could be applied in the future to more difficult problems.

References

[1] Yan Duan et al. "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *CoRR* abs/1604.06778 (2016). arXiv: 1604.06778. URL: http://arxiv.org/abs/1604.06778.

[2] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[3] Sanmit Narvekar and Peter Stone. "Learning curriculum policies for reinforcement learning". In: *arXiv preprint arXiv:1812.00285* (2018).

[4] Sanmit Narvekar et al. "Curriculum learning for reinforcement learning domains: A framework and survey". In: *The Journal of Machine Learning Research* 21.1 (2020), pp. 7382–7431.