AgarAI

TNM095 - Artificial Intelligence for Interactive Media

Kristin Bäck*
Lars Bergman†

November, 2016

Abstract

This report includes a brief summary of the reinforcement learning technique called Q-learning and how it was applied to a simplified version of the game Agar.io. The result is a game where a player meet an agent who gets smarter as the time goes on. The implementation is not perfect, why it is hard to, by testing, see how fast the player learns. To see a video of the result click here: AgarAI - result.

1. Introduction

This paper discusses the theory behind the method and the implementation of the Q-learning algorithm for a simplified version of the game Agar.io. The strengths and weaknesses of the agent will be discussed. The goal with this project was to implement the agent with the Q-learning algorithm and see how quickly it learns to play the game.

2. AgarIO

2.1. The Game

Agar.io is a popular massively multiplayer online action game where the players controls a cell/sphere and the goal is to get as large as possible [2]. The player grows by eating food or by eating smaller players than itself. Once the player itself is eaten, it respawns with the starting size. The camera is zoomed in on the player so it cannot see the whole game area. Agar.io also contains some additional features such as ”splitting up” and ”viruses”. These features are not implemented nor discussed in this report. The start page of the game is shown in Figure 1.

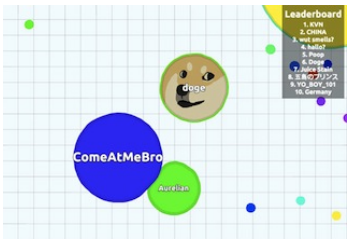

Figure 1: *Agar.io gameplay; this shows only a small fraction of an Agar.io map. There are four cells on this screenshot. Each cell with a name is controlled by a player.*

2.2. Game idea

The game goal is for the player to get bigger than its enemies. In this version Q-learning is applied on a agent playing the game against one player. Both the bot and the player are sphere objects that can move in any direction

*Student in Media Technology at Linköpings University, Sweden, Campus Norrköping,
email: kriba265@student.liu.se
†Student in Media Technology at Linköpings University, Sweden, Campus Norrköping,
email: larbe444@student.liu.se

in the scene. When either the agent or the player comes close to an object smaller than itself it gets eaten. If they are close to an object larger than itself they instead will be eaten. The game terminates/restarts when either the bot eats the player or the opposite.

3. Q-learning

The Q-learning method is a model free reinforcement learning technique based on an action-value-function containing rewards, learning-rate and discount factor [1]. The method gives a rule that given a certain state; choose the most optimal action from that state based on previously made choices.

The Q-learning method consists of a set of states and a set of actions to each state. By performing an action the bot moves between two states. For each action taken the bot get a reward of varying size.

The formula in Equation 1 describes the Q-learning method. QV(t+1)(st , at) is the result of the formula ie. The new Q-value given the old state *st* and old action *at*. α is the learning rate, it can be set to a value between zero and one. By setting it to zero the Q-value will never be updated. Setting it to a value close to one means that the Q-value will be updated more often and the agent will learn quicker. R(st , at) is the reward value given the old state *st* and old action *at*. R is a matrix and are called the *reward-matrix* further on in this report. Vt(st+1) is the learning value given the new state *st*+1. QVt(st , at) is the old Q-value given the old state *st* and old action *a*.

$$QV(t+1)(st,at) = (1-\alpha)QVt(st,at) + \alpha(R(st,at) + Vt(st+1)QVt(st,at)) \quad (1)$$

4. Method

The game is created in Unity, a cross-platform game engine. Unity simplified the modelling of game objects and setting up the scene.

4.1. Game setup

The game world is a 2D plane with the same size as the device screen used when playing the game. Food appears in the world as small circles with different colors and are randomly placed in the world continuously during run time. The players are circles that starts with a greater radius than the food, as can be seen in Figure 2. The size of the players increase when ”eating” food or other players. The food is eaten when the player is placed over the food. Every edge in the world is equipped with a static boundary in order to keep the players inside of the field of view.

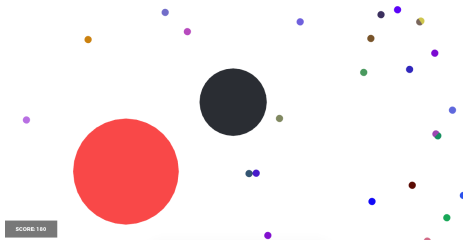

Figure 2: *A snapshot of AgarAI. The bigger red cell is the agent and the black cell is the player. The smalller cells are the food that spawns throughout the game session.*

4.2. Implementation of Q-learning

In order to implement the Q-learning algorithm every state and action had to be set. Below follows the states and actions that were implemented for AgarAI.

Actions

A1: Go to small player

A2: Go to large player

A3: Go to closest food

A4: Flee from player

States

S1: Close to food

S2: Far from food

S3: Close to large player

S4: Close to small player

S5: Far from large player

S6: Far from small player

S7: Close to large player and food

S8: Close to small player and food

S9: Close to corners

S10: Special rewards see section 4.3

The states and actions constitutes the reward matrix R which can be seen in Figure 3, where the actions are the columns and the states are the rows of R. The values of each slot is produced by guesses and trials of the Q-learning method in order to accomplish the wished behaviour of the agent. The values in the reward matrix never change but are instead used to produce the Q-matrix.

		Actions			
	States	**A1**	**A2**	**A3**	**A4**
	S1	0	0	100	0
	S2	100	200	−100	−100
	S3	100	100	0	−100
	S4	0	0	100	−100
R =	**S5**	100	300	−100	−100
	S6	100	100	0	−100
	S7	200	100	0	−100
	S8	100	0	300	−100
	S9	100	300	0	0
	S10	−1000	100	0	0

Figure 3: *The R-matrix with rewards for states and actions. Note that S10 are the special rewards discussed in section 4.3.*

Once the R-matrix was created, it was time to initialize the Q-matrix to zero. In order to during runtime update the matrix with result values based on the Q-learning equation. The current state of the agent needs to be considered and is set to the variable s. Based on this state s, amongst the possible action, one is chosen. The reward for taking the action is r and the resulting new state is set to s. The values of s, r and s are used in the Q-learning equation, 1, in order to update the Q-matrix values. After this the variable s is set to the new state and therefore becomes the current state. The process is repeated until a terminate state is reached. The agent will take the best possible action according to the Q-matrix for the current state.

4.3. Special rewards

Some special rewards/punishments were given to the agent. If the agent for example managed to eat the player given the current state and action performed, it would be rewarded with a bonus of X points. Also the agent was given negative X points if it was eaten for the current Q(s,a). This way the agent will converge to its final values faster and often tend to choose/avoid the action faster.

5. Result

The result is a game in which a bot plays against another player. The agent successively becomes smarter and learns to adapt to the game and the player by applying the Q-learning method. Therefore the agent will act and learn different depending on the playstyle of the player and what has been working and failing during that game session. This made it hard to determine how fast the agent learns to get the ”optimal” performance. However around five minutes into the game session the values tend to converge in the Q-matrix. This was discovered by a simple test where the player stood still while being larger than the agent from start. The simple test can be seen in Figure 4. Nevertheless that was also shown when actually playing against the agent in the Q-matrix after a few minutes.

One improvement to the agent would be to look over the action-functions that were implemented. They were implemented in an early stage of the project and are far from optimal. This was due to the fact that the main focus were on implementing the Q-learning algorithm. An example is the flee function that

only takes the vector of the players movement direction and move the same way. This makes it so the agent can easily be tricked into a bad position such as a corner.

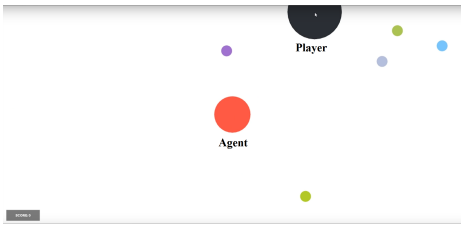

Figure 4: *The simple learning test, the player is the black cell and stands still. The agent is the red cell who starts smaller than the player and needs to eat food and then eat the player.*

6. Discussion

Solving the goal for this game could have been done differently. The Q-learning method worked but an additionally AI-method would improve the speed and accuracy of the bots learning.

Producing the reward matrix came with some troubles as already mentioned in 4.3. The values in the reward matrix were result of several trials. Time showed that the rewards needed to be of great difference for the bot to produce any learning.

The game is dynamic which makes every state and action more complex. The bot can move in any direction. Another ambiguity in the game is the player.

One goal for this project was to try and make the agent unbeatable. However since the player can change the learning of the agent and the interaction between the player and agent this is not the case as of now. Every move the player does affect the bots handling. This means the player can raise the bot in different ways.

If there would be two of the same agent playing against each other it is hard to say what would happen. Probably one of the agents would learn quicker than the other, based on the first random action that is given to the agents at the start of the game. However if one agent would be eaten by the other agent many times in a row, it would learn to dodge the other agent and try to eat food instead. So two identical agents playing against each other, would possibly end up in one agent who is very aggressive and eats food and one agent that is passive and eats food. The Q-matrix values of the two agents would not be the same.

To be able to say if the bot is actually learning fully, the game would have to be running for a long time. It is hard to predict the outcome of for example one week running, but the Q-matrix would probably be fully converged.

7. Conclusion

There are some things that can be further developed in order to get a more accurate result. Improvements to be made mostly relate to efficiency and the functions for the agent. Some functions in the code could definitely be optimized, for example the ”Flee” function. It would also be interesting to try and implement a Neural network or B-trees upon the Q-learning algorithm.

There is no function for testing implemented in this project. In order to accurately say how fast the bot learns by the Q-learning algorithm testing should get more focus. Also it would be interesting to implement more agents in the same game to see how the Q-matrixes for the agents develops.

References

[1] Richard S. Sutton and Andrew G. Barto. MIT Press, 1998 *Reinforcement learning: An Introduction* `https://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html` (Retrieved 2017-02-09)

[2] Fingas, Jon. *Reinforcement learning: An Introduction* https://www.engadget.com/2015/06/01/agar-io/ (Retrieved 2017-02-09)