Assignment: 3

ECSE- 526

Artificial Intelligence

Vraj Patel (261022581)

Introduction:

For this assignment, we design an agent which plays Atari game Mspacman with the help of library Arcade learning Environment (ALE). In the start of the report I discuss about the basic architecture of my game environment. In the following part, there is a brief discussion about the different approaches to exploration of the given state of the game and how to do I generalize my AI agent to reach a certain performance which is satisfactory.

MsPacman:

Arcade learning environment provides you're an environment where we can train our reinforcement learning agent to achieve human level performance. In reinforcement learning agent's main goal is to increase the game reward as much as possible, in this case maximize the game points. In mspacman game, the observation space is 210X160X3 which is RGB channel. For the analysis, I've tried to identify my components of the game with the help of different colors. I did multiple experiments initially to find out those RGB values for different colors and it help me to build a map for my agent from which I can easily identify different ghosts, pellets, fruits, etc. There are four different, types of ghosts in this game, but if pacman has a power up then it can eat those ghosts and collect points. There are four discrete different actions pacman can do at a particular state, which are shown in the table below.

Actions	Value
UP	2
RIGHT	3
LEFT	4
DOWN	5

Action space for MsPacman

Objects	Rewards
Pellet	10
Good Ghost	100
Bad Ghost	-200
Power Up	200
Fruit	100

Reward space for MsPacman

For finding my agent position and other moving objects of the game, I've used RAM values from the "MS_PACMAN.BIN" which was provided in the assignment description. I've used some libraries which converts images from the game play in to history which help me to identify the near objects of my pacman and help to better understand the current state of my game. The first part of my program is preprocessing of my current game state which gives me a more comprehensive game map, which helps my agent to do more precise decision making and helps the AI to achieve its objective which is maximize the game score.

Learning:

1. **TD (Temporal Difference):**
   TD (Temporal Difference) method to get the best possible action for my current state given that we take best possible action in next state.

$$\boldsymbol{U^{\pi}_{(S)} \leftarrow U^{\pi}_{(S)} + \alpha \left(R_{(s)} + \gamma \; U^{\pi}_{(s')} - U^{\pi}_{(S)} \right)}$$

   α = learning rate

   γ = discount factor

   $U^{\pi}_{(S)}$ = utility of a given state, initially it is zero

   $U^{\pi}_{(s')}$ = utility of a next state, initially it is zero

   $R_{(s)}$ = reward at a given state

   Above is the equation for TD method and how do we update the utility(U) of the given state. Here π is a policy which gives us the best possible move in the current state. Initially all the values are zero except reward values but as the game progresses the values of utilities converges to a fixed values and these are call state-value pair (Q-value) of the game.

Exploration:

1. **ε- greedy**
   In the ε- greedy algorithm, agent takes a random action as per value of ε provided. For the exploration part this thing is very important, but if we take random action, it can lead nearer to bad ghost and as a result pacman loses one life.
2. **SARSA:**
   SARSA refers to, State-Action-Reward-State-Action. If our aim is to play defensively this type of exploration is best for any defensive play. Since, in SARSA, we always take that gives the best rewards we take the safe route to the goal instead of exploring more options, we take a predefined path to collect our rewards.
3. **Optimistic Prior:**
   Optimistic prior is also called as (Q-learning) algorithm. In Q-learning, we keep track of those states where our agent previously been, next time we take the least taken state with the expectation of it giving us more rewards than the preferred policy. ε- greedy is too risky meanwhile, SARSA follows a defensive strategy, optimistic prior takes both of their characteristic and gives the best balance between exploration and exploitation.

Generalization:

Often, in the moving environment we can't calculate the Q-values of each state-action pair.

In this scenario, we're using RAM which takes least possible space than other modes of analysis of the game such as RGB and grayscale.
In RAM there are 128 different positions at a particular gametime, each state can have discrete values between [0,255] which means, there are 128^{255} possible states for this game of pacman, we can't calculate the whole Q-table instead we've somehow use some approximate function such that we can generalize this agent behavior which we observed during our training time.

Approach to generalization:

$$\boldsymbol{\widehat{U_\theta}(s) = \theta_1 f(s)_1 + \theta_2 f(s)_2 + \cdots + \theta_n f(s)_n}$$

$\widehat{U_\theta}(s)$ = approximate utility value at a given state
$f(s)_1, f(s)_2, \ldots\ldots, f(s)_n$ = function associated with the different objects of the game
$\theta_1, \theta_2, \ldots, \theta_n$ = parameters associated with the function which are unknown to us.

To solve the unknowns, we apply updated TD algorithm where we use approximate utility values instead of actual utility values,

$$\boldsymbol{\theta_t = \theta_t + \alpha \left[R_{(s)} + \gamma\, \widehat{U_\theta}(s') - \widehat{U_\theta}(s)\right] \frac{\partial \widehat{U_\theta}(s)}{\partial \theta_t}}$$

$\frac{\partial \widehat{U_\theta}(s)}{\partial \theta_t}$ = Gradient of utility with respect to given parameter
$R_{(s)}$ = Actual reward at a particular state
α = learning rate
γ = discount factor

$\widehat{U_\theta}(s)$ = utility of a given state s
$\widehat{U_\theta}(s')$ = utility at a next state s'

Possible ways for Generalization:

I've taken an approach of calculating the distance between my current position of pacman and different objects of the game such as good ghosts, bad ghosts, fruits, pellet, power up. I've also emphasize more on some parameters compared to the others based on the rewards the I would get. Based on that, I've more focused on my agent's distance from good ghost, bad ghost and power up. Since, these three distances would drastically change my game's reward I focused more on this rather than pellet and collecting fruits (which doesn't happen frequently)

Different types of distances:

1. **Manhattan distance:**
   In Manhattan distance we take the absolute value of distance between two points. In this distance, our parameters take more time to converge, or we might not get converged values after certain iterations.
2. **Euclidean distance:**
   Euclidean distance is root mean square distance between two points. Since we want to minimize or maximize our distance from pacman and objects this is the best way to calculate the distance since, our distance space is quadratic it is easier and faster way to converge.

Results:

For my generalization, I’ve implemented TD algorithm with ε- greedy and SARSA for exploration of results, I’ve also experimented with different distances such as Manhattan and Euclidean.

Distance		**Exploration**		**Score**
Manhattan	**Euclidean**	**ε- greedy**	**SARSA**	
Yes	-	-	Yes	760
-	Yes	-	Yes	450
-	Yes	0.25	-	2390
-	Yes	0.40	-	840
-	Yes	0.10	-	3400

Results from different experiments

Acknowledgement:

For this assignment, Alok Patel helped to provide some function which was really helpful to visualize the game map since, it was really hard to interpret with the ALE and get the require data.

References:

1) M. G. Bellemare, Y. Naddaf, J. Veness and M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents, Journal of Artificial Intelligence Research, Volume 47, pages 253-279, 2013.
2) *M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, M. Bowling. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents, Journal of Artificial Intelligence Research, Volume 61, pages 523-562, 2018.*
3) *Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.*
4) *Sridhar Mahadevan and Jonathan Connell. Automatic programming of behavior-based robots using reinforcement learning. Artificial intelligence, 55(2-3):311-365, 1992.*