Indian Institute of Technology Kharagpur
Department of Computer Science and Engineering

PacMan Problem Solver

Mayank Roy *(13CS30021)*
V Santosha Pradeep Chandra *(13CS30037)*
Amandeep Singh *(13CS30047)*

Under the Guidance of Prof. Sudeshna Sarkar

A report
Submitted in fulfilment for the Term Project in

Artificial Intelligence (CS60045)

Submitted on 30th November 2016

Abstract

For the voluntary term project for the AI Course, we have decided to focus on implementations of concepts that we have learned throughout the entire course. An excellent platform for this was fount to be the Pac-Man Projects [1] created by Dan Klein and John DeNero. The projects provide a game environment to implement all the basics learned in this course through small, completely contained simulators where modules for path planning, adversarial search games and reinforcement learning modules for Markov Decision Processes and Q-Learning can be implemented without worrying about details for setting up the simulator system which doesn't come into consideration while working on the core AI tasks.

Contents

Chapter 1

Pac-Man Projects

1.1 Motivation and Objectives

The idea behind solving the PacMan Projects was the understanding and implementation of all the concepts we had learned in the course but had not been apply anywhere so we had no experience visualising solutions based on those concepts. Each of the tasks implemented related directly to at least one or more of the topics we've done in class directly. Also, the project provides an autograder that creates randomized testcases to test our implementations on and to provide a check whether our implementations works properly or not.

URL for Repository

The tasks are as follows:

1. **Search**: Involved implementing depth-first, breadth-first, uniform cost, and A* search algorithms. These algorithms are used to solve navigation and traveling salesman problems in the Pacman world.

2. **Multi-Agent Adversarial Search**: Classic Pacman is modeled as both an adversarial and a stochastic search problem. This involved implementing multiagent minimax and expectimax algorithms, as well as designing evaluation functions for each of the heuristics where the adversaries are the ghosts.

3. **Reinforcement Learning**: This involved implementing model-based and model-free reinforcement learning algorithms to Pacman for unsupervised test runs in the game maze of varying

sizes and learning from chosen number of episodes for every test run of the Q-Learned Agent.

4. **Ghostbusters**: Probabilistic inference in a hidden Markov model tracks the movement of hidden ghosts in the Pacman world and we implemented exact inference using the forward algorithm and approximate inference via particle filters.

1.2 Search

1.2.1 Food Search

The first task was to solve the path searching to the food source at a corner of the maze. This problem

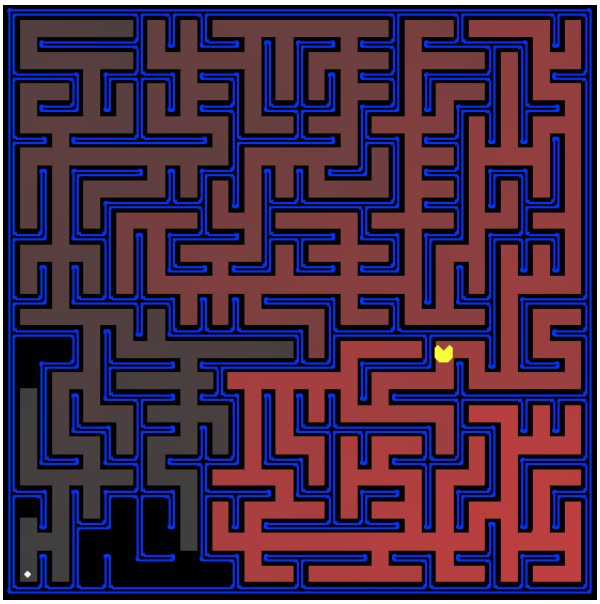

Figure 1.1: The Search Tasks

was solved using an implementation of DFS, BFS and A* search with no limits on the number of nodes expanded.

DFS and BFS

The only difference in the two implementations where we assumed no ghosts being present in the maze to assume a uniform cost function, the DFS implementation used a stack while the BFS implementation used a queue.

A* Search

The A* search[2] worked almost equivalently in speed to the BFS and DFS since the distance was the only metric we needed to solve using to get the food spot. To complicate this, we implement the next part.

Corner Finding Problem

This problem was a bit more complicated since now there are food spots on all the 3 corners other than the one our Pac-Man agent starts from. The task is to cover all the spots by expanding the least number of nodes. Further, we implement a sub-optimal solution to implement the search in a faster

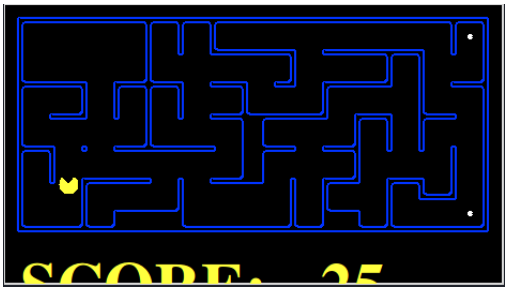

Figure 1.2: A* Corners Solver

manner with the expansion of fewer nodes (ID A* Search).

1.2.2 Food Eating With Optimized Search

Now, instead of just a single or three or four food sources, we implemented a solution for finding a fast path solution for eating multiple food sources. This increases the problem of expansion of nodes in

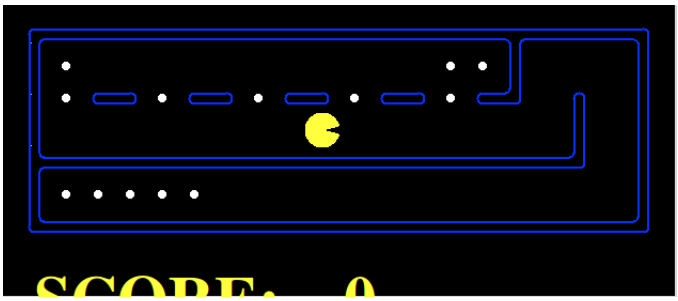

Figure 1.3: Problem State for the Food Eating Problem

the search of one food source exponentially because of which our optimal solution using full expansion

of all the nodes till each food spot resulted in the run taking close to 220 seconds.

Therefore, the suboptimal solution was tested next to see if we could complete the task in less time:
Time taken: 0.74s

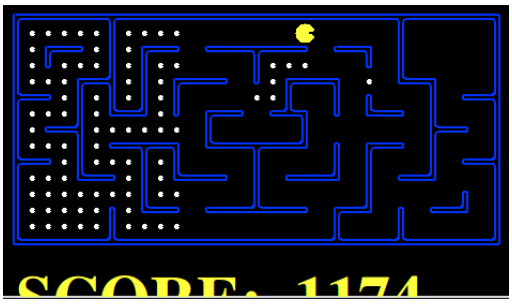

Figure 1.4: Suboptimal Search

1.3 Multi-Agent Adversarial Search

For this problem, it was a food capturing situation with the ghosts as our adversaries where they are actively on our agent's tails in the entire run. The first part was implementing a simple reflex agent

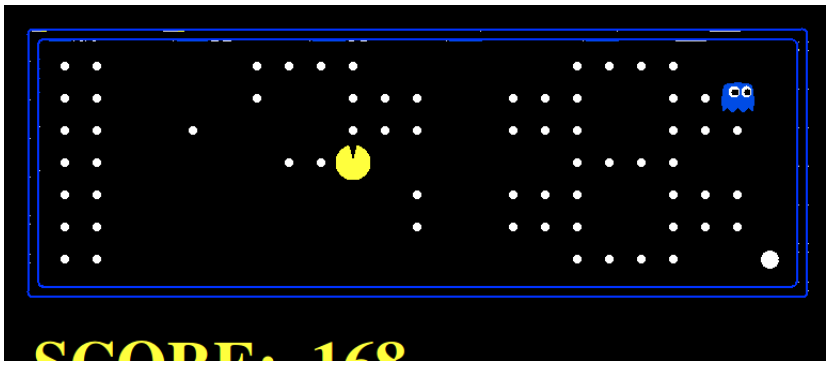

Figure 1.5: Simple Reflex Solver

based on distance from food sources and ghosts. The next part was implementing a minimax agent with subsequent parts extending it to alpha-beta and then to expecti-minimax[2] where the only difference between minimax and alpha-beta was that the latter had an appreciably faster runtime. However, on the expectimax run, it depended on our heuristic function how well our agent moved compared to the ghosts for food sources that were at a greater distance from the current agent location.

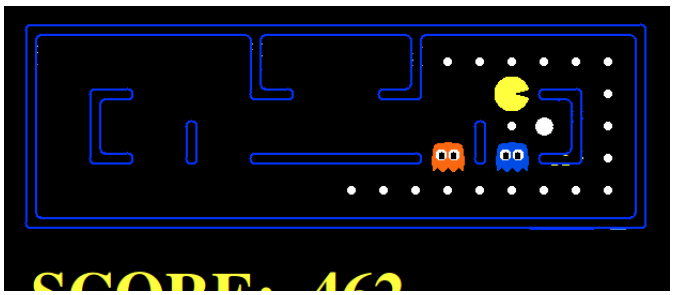

Figure 1.6: Minimax Solver

1.4 Reinforcement Learning

In this part of the project, we implemented a value iteration agent for solving known MDPs. This solver was used then in the Q-Learning part where we trained our implementation on 20000 runs or episodes and then tested our learned agent on 500 test games. The output resulted in wins for our agent close to 90% of the time suggesting it had learned very well the state transitions for various situations.

This however is a caution because our grid was very small sized and any increase in the size would

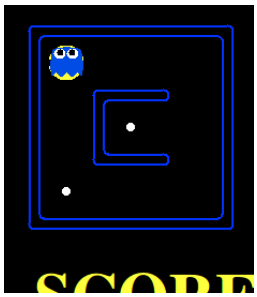

Figure 1.7: Q-Learning Agent for a Small Grid

result in an exponential increase in the number of states and hence a huge increase in the time taken to reach solutions. Hence, we solved for an approximate Q-Learning agent next which enabled us to solve for larger grids but would still take a lot of time because of the still significant increase in the size of the grid.

1.5 Ghostbusters

This part of the project was focused on solving for current ghost locations based on a noisy distance sensor model which decreases exponentially with block distance from the actual ghost location.

We implemented algorithms for performing both exact and approximate inference using Bayes' Nets.

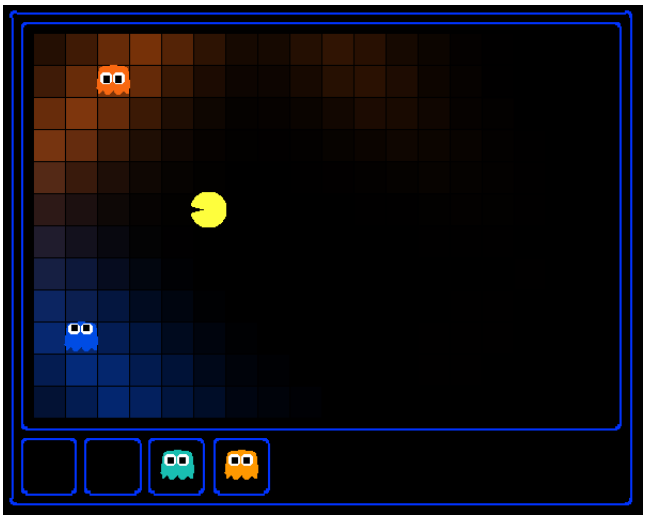

Figure 1.8: The Ghostbusting Problem

G denotes the current ghost location and E represents the estimate of the ghost location at the time

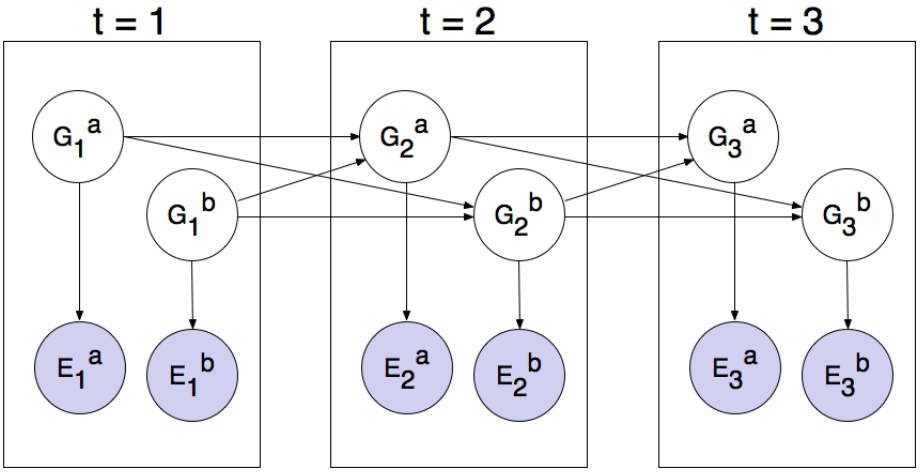

Figure 1.9: The Bayes Net for the Inference Problem

instant. It takes into account the current location and the previous inference returned by our querying as its belief state. The problem was solved in two parts:

1. Solving for Exact inferences of current ghost locations
2. Solving for Approximate inferences to implement a faster solution

Exact Inference is first trained for inference values and their transitions.

The knowledge that our agent has about the ghosts also involves the ways in which it can move and we needed to take into account all of these to create our solution. After training, the testing was done

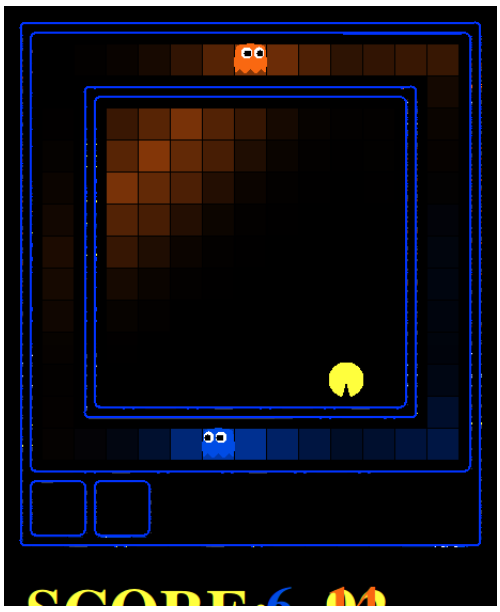

Figure 1.10: Training for Exact Inference

on a medium sized grid for the following case: The approximate versions of the solutions worked on

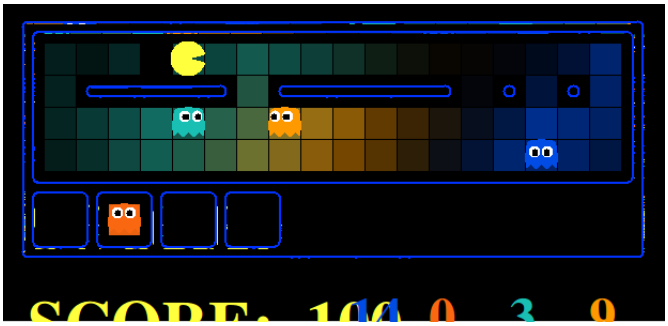

Figure 1.11: Exact Inference

similar training and test sized grids with the only benefit being reduced time for training the inference model for the Bayes Net.

1.6 Conclusion

This project has helped us provide a visualisation for each of the concepts we've learned in class as well the real world challenges that come in using them. We will try to improve upon our implemented work in this project later on our own so that we can remove the problems we faced in some places such as the IDA* implementation and the Q-Learning training time with use of advanced algorithms to make our computations faster.

Bibliography

[1] John DeNero and Dan Klein. *The Pac-Man Projects*. University of California Berkeley, CS188 Intro to AI, November 2016.

[2] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.