Artificial Intelligence Report 5th Week-15th Problem

Le Trung Kien
03-120291, 3rd Year
Department of Mechano-Informatics
The University of Tokyo

Professor: Hitoshi Iba

December 2, 2012

Note

- I originally wanted to spend more time on this project, but finally I decided to finish it on time.
- Fully tested on Ubuntu 32/64 bit and Macbook with Opengl library installed.
- Opengl library is installed by default on Mac, and it can be installed on Ubuntu by typing the following line in terminal:

  $ sudo apt-get install mesa-common-dev

1 Pacman

Pacman is classic game which was very popular in the 1980's. It is also a good benchmark to implement AI's algorithms, especially good for testing adversarial search algorithms. The original game's rules are quite complicated, and due to the time constraint, I will present its simpler version in this report:

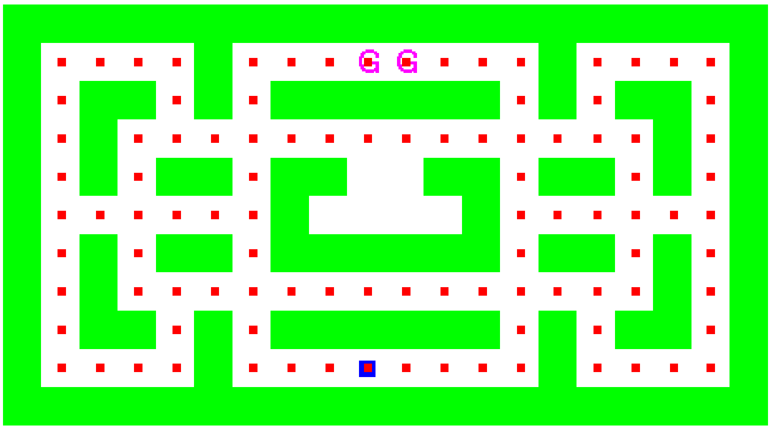

Figure 1: Start field of Pacman game (G is ghost, Blue dot is Pacman)

- Size of the board: 9×18
- Thers are two ghosts and one Pacman on the field.
- Pacman wins when eating up foods.
- Pacman looses when being eated by ghosts.
- The ghosts are always in chasing mode.

- Turn-taking game: Pacman first, then ghosts. Even though it looks like Pacman and Ghost simuntaneously make their moves, but in my implementation, pacman moves first, then ghosts choose their corresponding actions.

Normally, the task is to train Pacman to win the game, but I choose the opposition that is to make AI's ghosts. The two tasks have their own difficulty and appealing.

- AI's Pacman: Pacman wins only if he can eat all the foods without being eaten by ghosts. Therefore, when accessing a state from Pacman's point of view, we have to consider number of food left, distances from ghosts, distance to the nearest food.

- AI's Ghosts: Ghosts, on the other hand, do not care about foods, so their only concern is to chase Pacman. However, ghosts are not alone, so this is a **collaborative task**, when two or more ghosts have to act together to win their game.

In short, the most interesting point of making AI's ghosts are their collaborative behaviour toward the same goal, kill Pacman.

2 Adversarial Search

As the name suggests, adversarial search algorithms are in demand when adversary agents present. An agent cannot control its opponents or even its teammates' actions, that is why when making a move, it has to consider all possible outcomes of its and other agents' combined actions. The next following algorithms endevour to deal with a apecialized kind of games, which are deterministic, turn-taking.

2.1 Minimax

In Pacman game, Pacman is the Max player, while ghosts are Min players. The prototype for two main methods of MinimaxAgent class are:

```
virtual double Evaluate(const State &state, int depth, int player);
vector<Action> ChooseCombinedGhostAction(const State &state, int depth, double *v = NULL);
```

There is an ChoosePacmanAction method, but because my focus is to make AI's ghosts, so I will not present it here.

- **Evaluate State**: The state space of Pacman is very big, so we have to access a state even if it is not a terminal one. Basically, this function will define the efficiency of the algorithm. The better it evaluates a state, the stronger the chosen move will be.

- **Choose Ghost Action**: After get all legal ghosts combined moves, those combinations are evaluated and the one that minimize the outcome is chosen.

The mimimax algorithm performs a complete depth-first exploration of the game tree. If the maximum depth of the tree is m and there are b legal moves at each point, the the time complexity of the minimax algorithm is $\Theta(b^m)$. This is obviously infeasible for Pacman game, when b can be around 3, and m is very large. That is the reason why we need **Evaluate** method, instead of **Utility** method which only access the terminal state. The **depth** argument is used to limit the search space, that means, whenever the method reaches a terminal state or a state having the depth bound, it return a value.

2.2 Alphabeta Pruning

Alphabeta pruning is an enhanced version of Minimax algorithm in term of complexity. Therefore, AlphaBetaAgent class inherits almost all methods from MinimaxAgen class, except for the **Evaluation** method whose prototype is:

```
double Evaluate(const State &state, int depth, int player, double alpha, double beta);
```

- Alpha is the highest value we have found so far at any choice point along the path for MAX.
- Beta is the lowest value we have found so far at any choise point along the path for MIN.

Alphabeta pruning algorithm produces the same results as Minimax does, but its complexity is $\Theta(b^{\frac{m}{2}})$, which is far better.

2.3 Evaluation Function

Evaluation function is the one that decides Minimax or Alphabeta pruning algorithms' efficiency. Serveral features that I have tested are:

- + Pacman to ghosts Manhanttan distances: Ghosts want to minimize this distances
- - Ghosts to ghosts Manhattan distances: Ghosts should not be too far and too close to each other. If they are separated, they will not be able to combine to kill Pacman. If they are too close, they become one Ghost and which is an advantage for Pacman.
- + Pacman to ghosts graph distances:
- - Ghosts to ghosts graph distances
  + means the bigger the better for MAX, while - means the smaller the better for MIN

At first, I used Manhattan distances as features and quickly recognized that they are not suitable for Pacman game because not every grids pair is connected. Therefore, I chose the latter two, which are the real distances between two grids. In order to do this, I used **A*** algorithm to calculate all the shortest between all possible pairs of grids in advanced. The interesting point is I was able to utilize the same Gsearch class I wrote for 8-15 puzzle in previous report. It emphasizes the reusablity of my code.
Many references suggest that the evaluating result should be a linear combination of features. However, by doing so, we also ignore all dependency among features. That is not a good idea. For example, when the

3 Experiments

4 Discussion

References

[1] Stuart Russell, Peter Norvig *Aritifical Intelligence A Modern Approach*, Prentice Hall 3rd Edition, December 2009

[2] George F. Luger *Artificial Intelligence, Structure and Strategies for Complex Problem Solving*, Addison Wesley Longman 3rd Edition, 2009

[3] Robert Sedgewick, Kevein Wayne *Algorithms* Addison Wesley 4th Edition, January, 2012