Jishu Pro Report
AI's Pacman

By:
Le Trung Kien

03-120291
Department of Mechano-Informatics
The University of Tokyo

February 19, 2013

Contents

1 Pacman **2**

2 Adversarial Search **3**
- 2.1 Minimax . . . 3
- 2.2 Alphabeta Pruning . . . 3
- 2.3 Evaluation Function . . . 3
- 2.4 Polynomial Combination . . . 3
- 2.5 Neural Network & Optimization . . . 5

3 Experiments **5**
- 3.1 Execution . . . 5
- 3.2 Minimax vs Alphabeta Pruning . . . 6

4 Game State **7**

5 Discussion **8**

1 Pacman

Pacman is classic game which was very popular in the 1980's. It is also a good benchmark to implement algorithms related to Artificial Intelligence and Machine Learning, especially good for testing adversarial search algorithms. The game I created here is a bit different from the original one due to the complexity of the game. Some notes about the game rules:

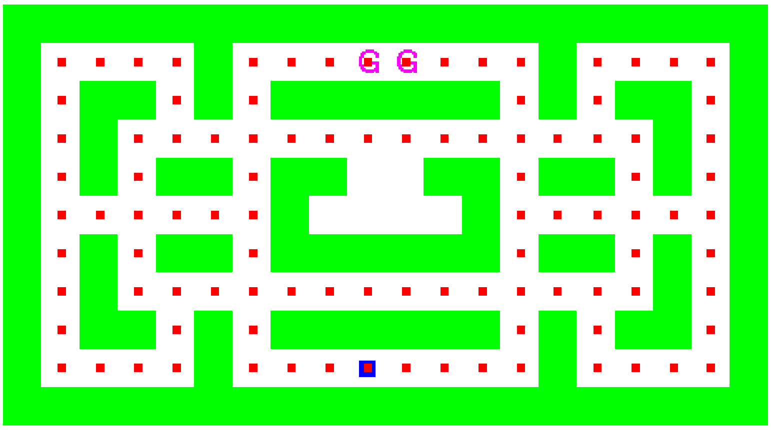

Figure 1: Start field of Pacman game (G is ghost, Blue dot is Pacman)

- Size of the board: 9×18 or larger.
- There are two ghosts and one Pacman on the field.
- Pacman wins when eating up foods.
- Pacman looses when being eated by ghosts.
- Ghosts cannot reverse directions, or stop.
- The ghosts are normally in chasing mode but will turn to scared mode when Pacman eats speacial food known as power pellets.
- Scared ghosts will return to normal after a specific number of moves.
- After a ghost is eaten, a new ghost is born out of the center of the board.
- 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. This make the implementation simpler.

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. We will see the difference in practice later in this report.

2.3 Evaluation Function

2.4 Polynomial Combination

Evaluation function is the one that decides Minimax or Alphabeta pruning algorithms' performance. 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. Finally, I came up with the following evaluation functions:

$$v = c_1 * f_1 + c_2 * f_2 + c_3 * f_1 * f_2$$

In which, f_1 is the total distance between Pacman and the two ghosts, and f_2 is the distance between two ghosts. Very few features here, but it turns out to be very powerful. The effect of this function will be discussed in the next section.
The actual implementation is:

```
double Utility::Evaluate(const State & state){
    vector<double> features(NUMFEATURES);
    features[0] = PacmanToGhostDistance(state);
    features[1] = GhostToGhostDistance(state);
    features[2] = IsFinal(state);
    features[3] = NumFood(state);
    // features[3] = PacmanToNearestFood(state);

    features[4] = NumGhostKilled(state);
    // features[6] = GhostDirection(state);
    double value = 0;
    // If terminal state
    if(features[2])
        return INFINITY*features[2];
    // If the two ghosts are next to each other
    // and close to Pacman
    if(features[0] <6  && features[1]==1){
        value += 100;
    }
    // If the two ghosts are far away from each other
    if(features[1]>=4)
        features[1] = 0;
    // If the two ghosts are overlap
    if(features[1] == 0)
        value += 100;

    value += coeff[0]*features[0] + coeff[1]*features[1]+coeff[2]*features[0]*features[1]+
        features[3]*coeff[3];
    return value;
```

Listing 1: Evaluate method (utility.cpp)

Because when Pacman is very close to Ghosts, the ghosts do not want to be next to each other. If they do so, they can not attack Pacman from different directions. Moreover, the two ghosts should never overlap each other too much. Therefore in those cases, I added an quite heavy weight to evaluation function. And when ghosts are far away from each other, I set it to Zero because at that point, all they should care is to get as close as possible to Pacman.

2.5 Neural Network & Optimization

The above method of evaluating state, though, is reasonably good, has its litmitation. It is very hard to find good basic function to fully ultilize relationship among features. In classification task, we have exactly the same problem when using polynominal features to form features's space. Another way to evaluate a state is to use neural network. Because three layer neural network can approximate any function, we can evaluate any state through its features. Additionally, by using genetic algorithm, neural network can be trained to gain the optimal weights for accessing game's states. In order to do this, both backpropagatation neural network and genetic algorithm for mutating neural network were created. Pacman agent and Ghosts' agent compete against each other and theoretically they should both get better over time.

3 Experiments

Performance of this game depends on which algorithm is used and which evaluation's method is utilized. Comparison between minimax and alphabeta pruning with in term of complexity will be ilustrated as following. Numerous experiments are conducted to train the neural work with genetic algorithm. However, the results was not good enough compared with evaluation's function by polynomial combination of features. Therefore, only the latter will be mentioned here.

3.1 Execution

Please visit

`https://www.youtube.com/watch?v=9TLatwHJ27E`

to see numerous experiments I have done to test the algorithm with graphics user interface. I also asked some of my friends to play Pacman role, and they all easily lost to the Ghosts. Until now, I and my friends have not won a game against the Ghosts (algorithm = alphabeta pruning, depth = 14, coefficient=10,-5,5). Please give it a try if you have time, and if you win, please send the playing log to me.

Both GUI and CLI interfaces are provided. At any point of the game, if the Min value is $-1e6$ (INFINITY), the ghosts already know that they are going to win. An example of running my application in command line.

- To run the game

./main algorithm depth c_1 c_2 c_3

	algorithm	depth	c_1	c_2	c_3
description	minimax or alphabeta	depth bound	1st coeff	2nd coeff	3rd coeff
default	a	12	10	-5	5

- Press Up, Down, Left, Right arrow keys to move Pacman. Press Home to stop Pacman.
- Press **'r'** to reset the game.
- Press **Esc** to end the game.

```
@:~/Dropbox/git/Pacman$ make
g++ -Wall -g -O2 -c -o state.o state.cpp
g++ -Wall -g -O2 -c -o common.o common.cpp
g++ -Wall -g -O2 -c -o minimaxAgent.o minimaxAgent.cpp
g++ -Wall -g -O2 -c -o utility.o utility.cpp
g++ -Wall -g -O2 -c -o alphabetaAgent.o alphabetaAgent.cpp
g++ -Wall -g -O2 -c -o main.o main.cpp
g++ -Wall -g -O2 -o main state.o common.o minimaxAgent.o utility.o alphabetaAgent.o
```

```
@:~/Dropbox/git/Pacman$ ./main a 12 10 -5 5
XXXXXXXXXXXXXXXXXXXX
X----X---GG---X----X
X-XX-X-XXXXXX-X-XX-X
X-X--------------X-X
X-X-XX-XX  XX-XX-X-X
X------X    X------X
X-X-XX-XXXXXX-XX-X-X
X-X--------------X-X
X-XX-X-XXXXXX-X-XX-X
X----X---P----X----X
XXXXXXXXXXXXXXXXXXXX
Number of food left: 100
Number of Evaluate: 10247
Minvalue: 250
Ghost Minimax Move: (L, R, )
XXXXXXXXXXXXXXXXXXXX
X----X--G--G--X----X
X-XX-X-XXXXXX-X-XX-X
X-X--------------X-X
X-X-XX-XX  XX-XX-X-X
X------X    X------X
X-X-XX-XXXXXX-XX-X-X
X-X--------------X-X
X-XX-X-XXXXXX-X-XX-X
X----X----P---X----X
XXXXXXXXXXXXXXXXXXXX
Number of food left: 99
.........................................................
Number of Evaluate: 1499
Minvalue: 70
Ghost Minimax Move: (L, L, )
XXXXXXXXXXXXXXXXXXXX
X----X--------X----X
X-XX-X-XXXXXX-X-XX-X
X-X---G----------X-X
X-X-XX-XX  XX-XX-X-X
X-----PX    X------X
X-X-XX XXXXXX-XX-X-X
X-X--- G      ---X-X
X-XX-X-XXXXXX X-XX-X
X----X----    X----X
XXXXXXXXXXXXXXXXXXXX
Number of food left: 85
Number of Evaluate: 2266
Minvalue: -1e+06
Ghost Minimax Move: (D, L, )
Pacman: LOSE
```

3.2 Minimax vs Alphabeta Pruning

The following table shows comparision between Minimax and Alphabeta Pruning in term of number of evaluated nodes. The depth is set as 12. Clearly Alphabeta pruning is much more efficient than Minimax. Notice that number of evaluated nodes varies a lot due to the different number of legal moves. Alphabeta pruning performance also depends on the order of examined nodes. The sooner the

Pacman Move	Ghost Moves	Minimax	Alphabeta Pruning
r	l,r	5824	1936
r	l,r	5299	1554
r	l,r	10514	1478
r	d,d	22021	2182
u	d,d	61080	1855
u	r,d	286179	20064
r	r,d	49025	1119
r	r,d	109956	3288
r	r,d	66916	5059
u	r,r	66804	9269
u	r,r	17848	2193
l	r,r	13621	1660
l	d,u	17056	1673

best move is found, the more nodes can be pruned.

4 Game State

The State class, which represents the Game state take me lots of hours to complete. It was hard to think from how to represent the game state to how to make transition from a state to another. The class declaration is shown in the following Listing.

```
class State{
private:
    // Food matrix
    vector<int> food;
    // Wall matrix
    bool *wall;
    // Number of rows, cols
    int rows, cols;
    // Number of food
    int numFood;
    // Number of move
    int startScared;
    int numMove;
    // Pacman's position
    Position pacmanPos;
    // Ghosts' position
    vector<Position> ghostPos;
    // Previous ghost actions
    vector<Action> previousGhostAction;
    // Gostscared
    vector<bool> ghostScared;
    // Make ghost scared
    void MakeGhostScared(bool scared);
public:
    // Constructor
    State();
    State(int rows, int cols, bool *wall, int *food);
    // Initialization
    void Initialize(int rows, int cols, bool *wall, int *food);
    // Get next states
    State GetNextState(Action pacmanAction, const vector<Action> &ghostAction);
    State GetNextState(Action pacmanAction);
    State GetNextState(const vector<Action> &ghostAction);
    // Get legal actions
    vector<Action > GetLegalGhostAction(int ghostIndex) const;
    vector<vector<Action> > GetLegalCombinedGhostAction(int ghostIndex) const;
    vector<vector<Action> > GetLegalCombinedGhostAction() const;
    vector<Action > GetLegalPacmanAction() const;
    // Check if legal actions
```

```
    bool IsLegalPacmanAction(Action pacmanAction) const;
    bool IsLegalGhostAction(Action ghostAction, int ghostIndex) const;
    bool IsLegalCombinedGhostAction(vector<Action> ghostAction) const;
    /*Element Access*/
    int Food(int i, int j) const;
    int &Food(int i, int j);
    int Food(Position pos) const;
    int &Food(Position pos);
    bool Wall(int i, int j) const;
    bool Wall(Position pos) const;
    bool Wall(int i) const;
    int NumFood() const;
    int NumAction() const;
    Result IsFinal() const;
    int Rows() const;
    int Cols() const;
    int NumGhost() const;
    int GhostScared(int ghostIndex) const;
    bool GhostKilled(int ghostIndex) const;
    Position PacmanPosition() const;
```

Listing 2: state class (state.hpp)

Please refer to the source code **state.cpp** for more details about this class.

5 Discussion

Firstly, I want to talk about the Evaluation function. As mentioned above, this function is the most important aspect of Minimax and Alphabeta pruning algorithm. In a limited time, I think I considered a very simple version of it. In order to optimize the Evaluation function, we can add more features to it, and try to combine features in more sophisticated way such as Neural Network. I think it is possible to combine Neural Network and Genetic Algorithm to find the optimal evaluation function to this problem. That is maybe the theme for my next report.
Secondly, Alphabeta pruning is much more efficient than Minimax algorithm. In real applications, probably minimax will never be used. The game now is a deterministic one, however, if uncertainty is taken into account, we have an other algorithm called Expectimax. In this case, opponents do not always take their best moves, they sometimes take other moves which produce different outcomes.
Finally, I admit I regret not to make hardware for Jishu Pro, since it was probably the last chance for me to touch hardware. I am seriously interested in Artificial Intelligence and Machine Learning, so I decided to use Pacman as a platform to implement those algorithms I have learned. However, there would have been much more fun if I had made some hardware, such as a Pacman and a ghost fighting against each other.

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

[4] Christopher M. Bishop *Pattern Recognition and Machine Learning*, Springer 1st Edition, 2006