LAB 2

Kumar Ayush - 2015eeb1060

Sumit Singh - 2015csb1036

Course Instructor - **Dr. Narayanan C. Krishnan**

Indian Institute of Technology, Ropar

1. Hill Climbing Search

Method

Implemented in C. Used a variant of simulated annealing where our agent took bad moves with decreasing probability. We initialize a random initial state and took best local neighbour each step till local optimum is reached.

Observation

Due to absence of any inbuilt data structures used in C, our implementation runs generally slower. We observed that when basic hill climbing was used, local optimum was reached in generally 10-15 iterations of random restarts. When we added a variant of simulated annealing to our code, it took a little longer, around 20-25 iterations. The probability with which it took bad moves had to be kept decreasing because otherwise it stuck in infinite loop a few times. We initially start with probability of 40% with which it took bad moves but only for a limited number of times.

2. Multi-Agent Pacman

1. Reflex Agent

Method

We evaluate the possible actions rather than states. We gave a weight to food, ghost and scared ghost. Scared ghost weight is a lot higher because we want our agent to chase it as quickly as possible when it appears. Food weight and ghost weight are used respective to each other.

Assessments and Measures

testClassic:

- Score: 564

mediumClassic (1 ghost):

- Score: 1536

mediumClassic(2 ghost):

- Score: 1261

openClassic:

- Score: 1203

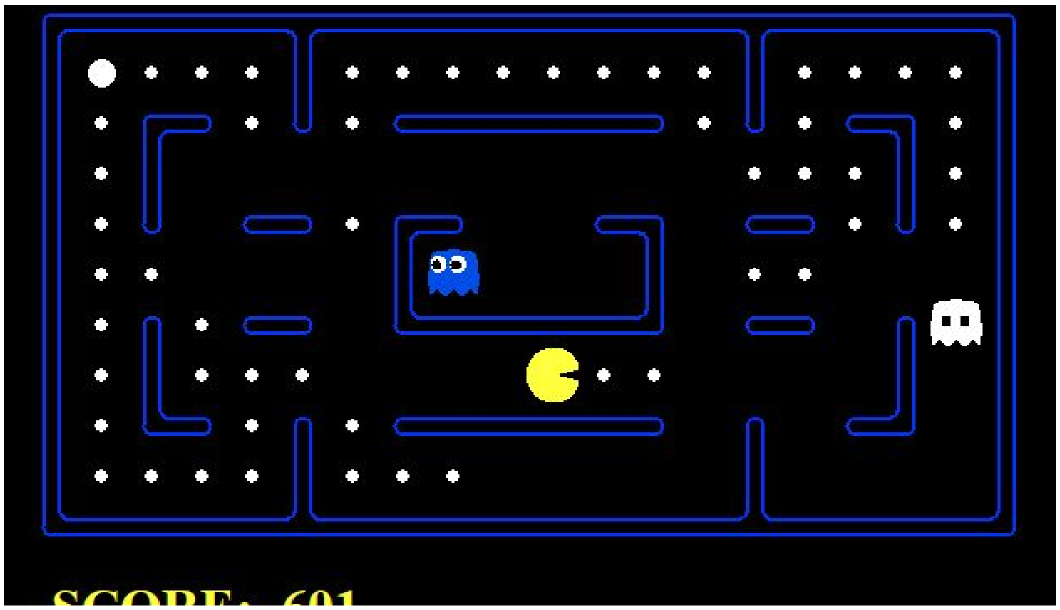


Fig 1: Medium Classic with 2 ghosts

Observation

Using lower value for food weight(< 5) with respect to ghost weight resulted in pacman getting stuck when ghosts are near food even when it should not be. Similarly, using lower value for ghost weight(< 5) resulted in pacman dying more frequently. The agent easily passed *testClassic* and *mediumClassic* with 1 ghost, but, on *mediumClassic* with 2 ghosts, it only passed 1 out of 5 times on an average.

2. Minimax

Method

Implemented using 3 functions. doMinimax decides which node(max or min) belongs to the current agent and the minNode and maxNode are for ghosts and pacman respectively.

Assessments and Measures

Minimax classic:

- Success only 4/10 times on an average.

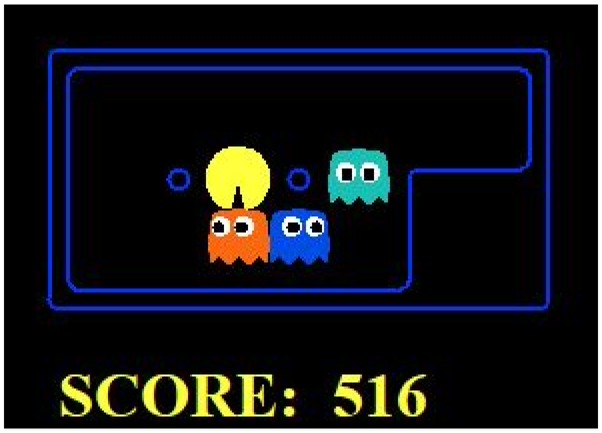


Fig 2: Minimax classic

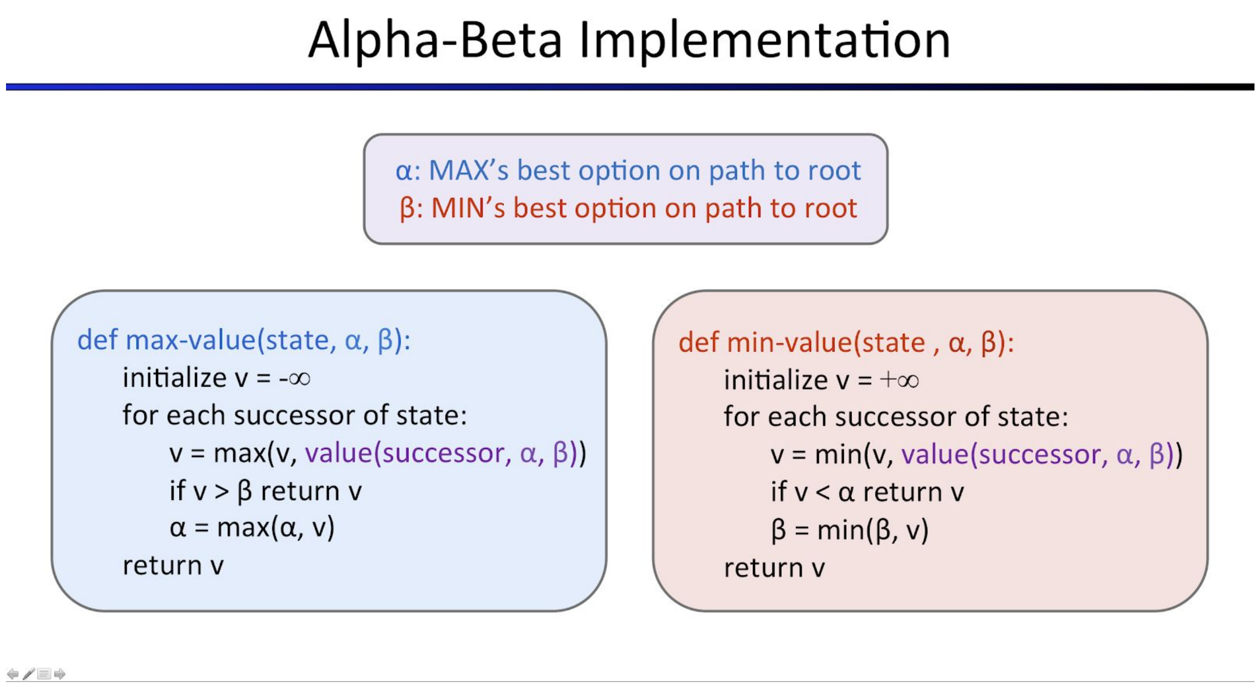
Observation

On minimax classic our agent succeeded only 40% of the time on an average. On trapped classic, our agent just rushed towards the ghost since it was not possible to win anyway, and with time passing, the score reduces. So, it tried to die as soon as possible. On medium classic, pacman often just waits without any progress even when ghosts are far away. It doesn't even take food just in front of it. It's because since we are using manhattan distance as our evaluation function and our ghost agent are assumed to behave as optimal as possible, our pacman assumes the worst. The implementation too is quite slow. Similar problems plague the open classic. On trapped classic, since the final result is to die anyway, our pacman rushes to nearest ghost possible to save time and score.

3. Alpha-Beta pruning

Method

Implemented by adding extra arguments in same functions used in minimax according to following figure:



Alpha-Beta Implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v > β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v < α return v
        β = min(β, v)
    return v
```

Fig 3: Pseudo-code for alpha-beta pruning

Assessments and Measures

Small classic:

- Pacman die 9/10 times

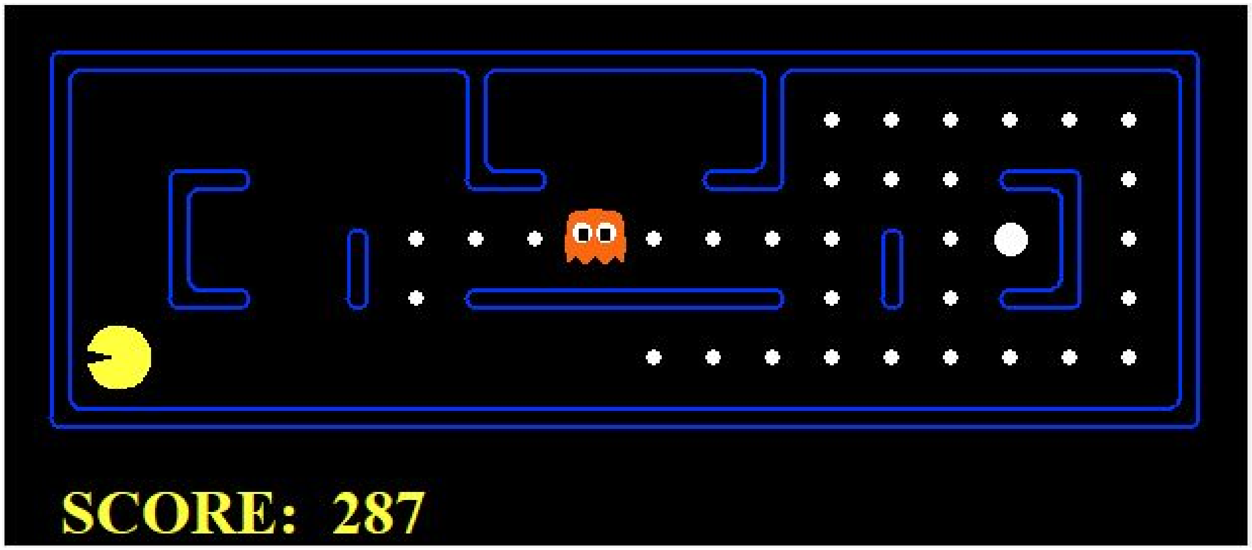


Fig 4: Small classic using alpha-beta pruning

Observation

We observed that in small classic, pacman died 5/10 iterations on an average. Although, it ran few seconds faster than simple minimax on each map.

4. Expectimax

Method

We just changed minNode function to return the average/expected value instead of minimum that it returned before. The action also is the last one in the order returned by *getSuccessor.*

Assessments and Measures

minimaxClassic:

- Succeeded only 3/10 times on an average

trappedClassic:

- Succeeded 4/10 times on an average

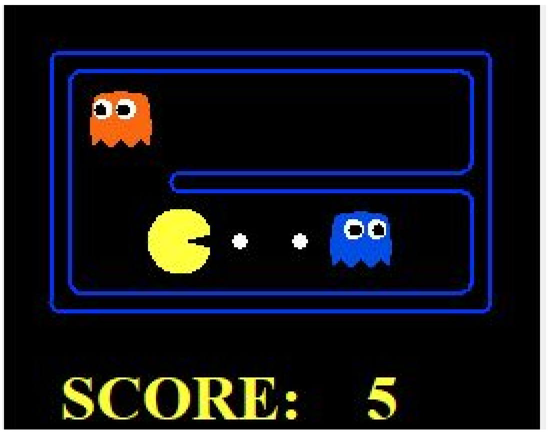


Fig 5: Trapped classic using Expectimax

Observation

Though, on minimax classic, it performed same with respect to alpha beta pruning, we see that it actually performs quite better on trapped classic where our alpha beta pruning miserably failed each time. This time, our pacman agent *does not* assume the worst and try to take as much food possible before dying. Previously, our pacman assumed that it's final state would be to die but this time it's now with only a certain probability (around 50% chance), that means, our pacman assumes that it may stay alive with some probability, and thus, rushes for food.

5. Evaluation function

Method

It's exactly same as what we used in case of reflex agent but this time it does so on state itself, rather than the action. We gave a weight to food, ghost and scared ghost. Scared ghost weight is a lot higher because we want our agent to chase it as quickly as possible when it appears. Food weight and ghost weight are used respective to each other. It's linear combination of those 3.

Observation

Average score is > 1200, so our evaluation function operates quite well as expected.

Conclusion:

Even with evaluation functions, our agent doesn't seems to be perfect as with a small probability, it still loses a few times. It's because there are some cases, where the ghosts traps the pacman with no chance to escape, some cases where the position food and ghost makes the evaluation function nonoptimal and pacman makes ‘incorrect’ move. Still, minimax algorithm works quite well in most of the cases.