COSE361(03) - Final Project 2/2 Report
: Algorithm for Multi-Player Capture-The-Flag Pacman

2021320117 Bae Min-Seong

Abstract

This document provides a explanation for algorithm and implemented code for the Final Project 2/2, the multi-player Capture-The-Flag variant of Pacman game. The main algorithm is MiniMax-search tree with alpha-beta pruning and evaluation functions with features.

1. Introduction

1.1. Description for Final Project 2/2

Game Structure: Basically, the game for this project is based on Pacman Rule, but there is a big difference that there are two teams, blue and red and two teams are adversarially competing for trying to eat food in opponent's region and disturbing opponents to eat food in own region.

Key Points: This adversarial structure of the game represents that this game is a multi-agent game, so we can apply many multi-agent game tree algorithms.

1.2. Brief Description for Implementation

The implementation of agents in the game is basically based on **Minimax-Search Tree** algorithm with **alpha-beta pruning**. For the agents of red team(player's team), the MAX node represents them and for the agents of blue team(opponent's team), the MIN node represents them. Also the alpha-beta pruning method reduce the computation for calculating value of tree nodes. In addition, by using evaluation function for some features from states of the game, it search the next action in the limited depth.

2. Methods: Details for Implemented Agents

2.1. 2021320117.py (Minimax with pruning + Evaluation function)

This file is the main algorithm for agent implementation. Basically, this code is implemented using CaptureAgent class from captureAgents.py. First, we implemented CaptureMinimaxAgent for Minimax-Search tree structure. It is basically implemented with my code in the second assignment. By max-value and min-value function, each value of the node is calculated and we choose the action with highest value. Also, in the process, alpha-beta pruning is used for reducing computation. With this CaptureMinimaxAgent, we implemented two agents : OffensiveMiniMaxAgent and DefensiveMinimaxAgent using inheritance. Each agent has difference in features and weights of the evaluation function.

2.1.1. Features for OffensiveMinimaxAgent

1. foodNum : number of remaining opponent's food
2. capsuleNum : number of remaining opponent's capsule
3. score : score of current game state
4. distanceToFood : distance to the closest food
5. scaredGhostNum : number of scared ghost
6. distanceTononScaredGhost : distance to the closest non-scared ghost
7. distanceToScaredGhost : distance to the closest scared ghost
8. distanceToHome : distance to start position to current position
9. distanceToCapsule : distance to the closest opponent's capsule
10. isTrapped : whether it is trapped by the enemy ghost or not

2.1.2. Features for DefensiveMinimaxAgent

1. score : score of current game state
2. foodNum : number of remaining player's food
3. distanceToInvader : distance to the closest invading pacman opponent
4. invaderNum : number of invading pacman opponents
5. distanceToCapsule : distance to the closest player's capsule

2.2. Other Baseline Codes

Each baseline code is implemented with variants of the main algorithm code and provided baseline code. The first baseline code is almost same to main code, but there is a difference between the depth limit of search tree. The main tree has depth-limit 2 but the first baseline has depth-limit 1. The second one and third one is based on the provided baseline code, but second one has same features and weights from the offensive agent from the main code and third one

has same features and weights from the defensive agent. Other parts are same as provided baseline code.

3. Result

We can check the table for the result of capture.py with baseline codes and main code.

output

	your_best(red)
	<Average Winning Rate>
your_base1	0.4
your_base2	1.0
your_base3	1.0
baseline	0.9
Num_Win	4.0
Avg_Winning_Rate	0.825
	<Average Scores>
your_base1	2.2
your_base2	4.6
your_base3	15.4
baseline	3.1
Avg_Score	6.325

4. Conclusion & Free Discussion

From the result, we can conclude that our main algorithm is better than provided baseline code and other baseline code.

Q. Why the main algorithm is better than other baseline algorithms?

First, the main algorithm is better than the baseline algorithm that it is almost same as main minimax-search algorithm but the depth-limit is smaller than the main one because the evaluation function is an approximation for the values of deeper nodes so it can be more accurate for calculating values.

In addition, the second and third algorithm are algorithms based on baseline algorithm and only replacing one of offensive / defensive agent with main offensive / defensive agent. So, the main algorithm would be better than these modified algorithms because it has more various and specific features to evaluate values of nodes in the limit of the depth.