UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER SCIENCE

EVALUATION FUNCTIONS FOR MINIMAX/ALPHABETA/EXPECTIMAX

Instructor: Luong Ngoc Hoang

Student: Ha Huy Hoang - 22520460

Class: CS106.O21

Table of Contents

1. Introduction

Pacman is an engaging game where the player navigates a Pacman character through a maze filled with food, capsules, and ghosts. The primary objective is to guide Pacman to consume all the capsules without getting trapped by the ghosts. By employing Adversarial and Random Search techniques, we can optimize the game strategy to achieve the highest possible score and ensure Pacman's victory in this task. This approach allows us to anticipate the ghosts' movements and make the most effective decisions for Pacman.

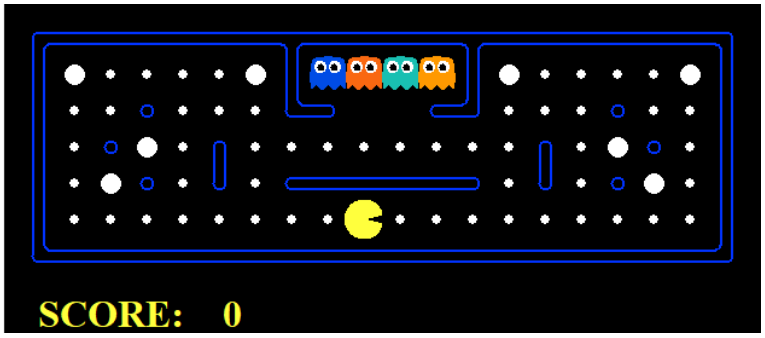

Image 1. An example map in Pacman

2. Idea of betterEvaluationFunction

Evaluating Pacman's intermediate states based solely on the score may not yield the most effective results. In this discussion, I will explore an approach that evaluates each element separately, such as the score, Pacman's proximity to food, and his distance from ghosts, among others.

I will assign a reward value to each feature, and then calculate the score for each step taken towards the nearest object. Specifically we will have four main weights, as follows:

- eatenWeight = 300 (Award much more points to prioritize eating ghosts over food or capsules)
- ghostWeight = 20 (Award additional points to prioritize avoiding ghosts over eating food or capsules)
- foodWeight = 10
- numWeight = 500

The function will evaluate the state through the weights and the number of steps Pacman moves to each corresponding object. We will take **score = currentGameState.getScore()** as the standard.

First, we will iterate through all the ghosts. If a ghost is in the **scaredTimer**, we will add to the score at that state **eatenWeight** divided by the number of steps Pacman takes to that ghost. If the ghost is not in a scared state, we will check if it is the closest ghost to Pacman. If it is, we subtract **ghostWeight** divided by the number of steps Pacman takes to it from the score, if the number of steps is greater than 0. If it equals 0, we subtract -99999999 points

Second, we iterate through the Foods and Capsules, adding to the score an additional **foodWeight** divided by the number of steps to the nearest food or Capsule.

Third, we add to the score an additional **numWeight** divided by the number of remaining food (the fewer the remaining food, the more points are added).

3. Statistics

For convenience in statistics, I have written a file called **run_for_statistic.py**. Due to the limited configuration conditions of my computer, I can only run a maximum of **'depth = 3'**. Specifically, the trickyClassic, openClassic, and originalClassic levels can only run **'depth = 2'**.

In the case of openClassic, because the eval function uses scoreEvaluationFunction, it only estimates based on the score and I can only evaluate within 2 steps. Therefore, Pacman just stands and moves around a corner. So, if using scoreEvaluationFunction in this level, the score would be **-** (undefiend)

This is the best result video I could achieve when running the mediumClassic layout with Expectimax, betterEvaluationFunction, and depth = 3.

capsuleClassic.lay, depth = 3

RandomSeed	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
22520460	-438	-438	-187	-66	-66	972
22520461	-457	-457	-458	-13	-13	783
22520462	-431	-431	-431	-431	-431	-430
22520463	-587	-587	-44	789	789	796
22520464	-448	-448	-232	804	804	1192

contestClassic.lay, depth = 3

RandomSeed	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
22520460	349	349	349	2290	2290	1840
22520461	-21	-21	-21	1836	1836	1534
22520462	-39	-39	908	3153	3153	931
22520463	63	63	1732	1460	1460	2997
22520464	-124	-124	1079	211	211	2922

mediumClassic.lay, depth = 3

RandomSeed	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
22520460	370	370	-2292	1839	1839	2112
22520461	1313	1313	1313	1869	1869	1634
22520462	1679	1679	1679	2003	2003	1934
22520463	-4339	-4339	-7284	913	913	1923
22520464	-1071	-1071	278	2076	2076	2051

minimaxClassic.lay, depth = 3

RandomSeed	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
22520460	-496	-496	-498	-496	-496	-498
22520461	509	509	513	502	502	502
22520462	-496	-496	-496	-496	-496	-497
22520463	514	514	515	514	514	514
22520464	512	512	512	-498	-498	509

openClassic.lay, depth = 2

RandomSeed	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
22520460	-	-	-	1157	1157	1293
22520461	-	-	-	1416	1416	1056
22520462	-	-	-	1352	1352	1273
22520463	-	-	-	1217	1217	1230
22520464	-	-	-	1416	1416	1426

originalClassic.lay, depth = 2

RandomSeed	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
22520460	325	325	325	2479	2479	1641
22520461	364	364	488	-27	-27	3277
22520462	42	42	42	942	942	2852
22520463	71	71	71	855	855	2205
22520464	-45	-45	-45	2094	2094	2245

powerClassic.lay, depth = 3

RandomSeed	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
22520460	3190	3190	472	4528	4528	5801
22520461	3658	3658	2561	2077	2077	3560
22520462	2383	2383	3472	4426	4426	4059
22520463	611	611	891	960	4960	5228
22520464	998	998	3016	5103	5103	4988

smallClassic.lay, depth = 3

RandomSeed	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
22520460	1286	1286	1286	1202	1202	1467
22520461	-718	-718	305	1660	1660	1740
22520462	214	214	212	1665	1665	1741
22520463	1094	1094	1094	1614	1614	150
22520464	1554	1554	-184	1651	1651	1594

testClassic.lay, depth = 3

RandomSeed	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
22520460	524	524	526	564	564	561
22520461	532	532	532	562	562	561
22520462	530	530	536	560	560	564
22520463	522	522	522	564	564	558
22520464	500	500	500	561	561	561

trappedClassic.lay, depth = 3

RandomSeed	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
22520460	-501	-501	-502	-502	-502	-502
22520461	-501	-501	-502	-502	-502	-502
22520462	-501	-501	-502	-502	-502	-502
22520463	-501	-501	-502	-502	-502	-502
22520464	-501	-501	-502	-502	-502	-502

trickyClassic, depth = 2

RandomSeed	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
22520460	980	980	646	1500	1500	1576
22520461	263	263	263	1056	1056	2406
22520462	484	484	483	1359	1359	3325
22520463	1630	1630	-610	2128	2128	2865
22520464	758	758	807	3175	3175	1145

4. Evaluation

As we can see, it's clear that the **betterEvaluationFunction** yields significantly better results compared to the **scoreEvaluationFunction**. Comparing the methods, **MiniMax** and **AlphaBeta Pruning** yield the same results for each game, meaning the effectiveness of both methods is the same. However, the runtime of **AlphaBeta Pruning** is faster than **MiniMax** in some games and depends on how the nodes are expanded. **Expectimax** generally has the highest win rate and average score among all methods, even when using both **scoreEvaluationFunction** and **betterEvaluationFunction**.

If we rank them based on effectiveness and performance, the order would be:

MiniMax →AlphaBeta Pruning → Expectimax

capsuleClassic.lay, depth = 3

	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
Win Rate	0%	0%	0%	20%	20%	40%
Avg Score	-472.2	-472.2	-270.4	216.6	216.6	662.6

contestClassic.lay, depth = 3

	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
Win Rate	0%	0%	0%	20%	20%	40%
Avg Score	45.6	45.6	809.4	1790	1790	2044.8

mediumClassic.lay, depth = 3

	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	AlphaBeta	Expectimax	Minimax	AlphaBeta	Expectimax
Win Rate	40%	40%	40%	80%	80%	100%
Avg Score	-409.6	-409.6	-1261.2	1740	1740	1930.8

minimaxClassic.lay, depth = 3

	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	**AlphaBeta**	**Expectimax**	**Minimax**	**AlphaBeta**	**Expectimax**
Win Rate	60%	60%	60%	40%	40%	60%
Avg Score	108.6	108.6	109.2	-94.8	-94.8	106

openClassic.lay, depth = 3

	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	**AlphaBeta**	**Expectimax**	**Minimax**	**AlphaBeta**	**Expectimax**
Win Rate	0%	0%	0%	100%	100%	100%
Avg Score	-	-	-	1311.6	1311.6	1255.6

originalClassic.lay, depth = 3

	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	**AlphaBeta**	**Expectimax**	**Minimax**	**AlphaBeta**	**Expectimax**
Win Rate	0%	0%	0%	20%	20%	60%
Avg Score	151.4	151.4	176.2	1268.6	1268.6	2444

powerClassic.lay, depth = 3

	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	**AlphaBeta**	**Expectimax**	**Minimax**	**AlphaBeta**	**Expectimax**
Win Rate	60%	60%	60%	80%	80%	60%
Avg Score	2168	2168	2082.4	3418.8	3418.8	4727.2

smallClassic.lay, depth = 3

	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	**AlphaBeta**	**Expectimax**	**Minimax**	**AlphaBeta**	**Expectimax**
Win Rate	60%	60%	40%	100%	100%	80%
Avg Score	686	686	542.6	1558.4	1558.4	1338.4

testClassic.lay, depth = 3

	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	**AlphaBeta**	**Expectimax**	**Minimax**	**AlphaBeta**	**Expectimax**
Win Rate	100%	100%	100%	100%	100%	100%
Avg Score	521.6	521.6	523.2	562.2	562.2	561

trappedClassic.lay, depth = 3

	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	**AlphaBeta**	**Expectimax**	**Minimax**	**AlphaBeta**	**Expectimax**
Win Rate	0%	0%	0%	0%	0%	0%
Avg Score	-501	-501	-502	-502	-502	-502

trickyClassic.lay, depth = 3

	scoreEvaluationFunction			betterEvaluationFunction		
	Minimax	**AlphaBeta**	**Expectimax**	**Minimax**	**AlphaBeta**	**Expectimax**
Win Rate	0%	0%	0%	20%	20%	40%
Avg Score	823	823	317.8	1843.6	1843.6	2263.4