Artificial Intelligence – HW2 Report

Submitters:

Yuval Nahon – 206866832

Sahar Cohen – 206824088

Part A: Introduction to the game

3) ReflexPlayer is a greedy agent algorithm for evaluating the best possible move in each turn.

The agent overloads the **getAction** method, which gives our agent, Pacman, the action it deems best to take. This action is determined by evaluating a heuristic value (with the **scoreEvaluationFunction** method) for each possible action that Pacman may take. The agent is unaware of future moves, and thus only chooses the highest heuristic value action for the current turn only.

The scoreEvaulationFunction method is the heuristic that the original ReflexPlayer uses to determine the quality of each action. The heuristic only gives credit to Pacman's **score**: for each action, it returns the score that Pacman would have once the action is taken.

Part B: Improved agent

1) We'll denote the heuristic function by $h : S \to R$.

$$h(s) = score + foodDistancesScore + |\,actions\,| + ghostDists + foodScore + scaredAvg + rand$$

where:

- $\boldsymbol{score}$ = $s.score$ is the score of state s,

- **foodDistancesScore** = $\alpha * \max_{food \in s.food}\{\, s.arenaHeight + s.arenaWidth - manhattanDistance(food.position, pacman.position) \,\}$ increases when Pacman moves towards food (as the manhattan distance from the food decreases),

- **actions** = $s.legalActions$ is a list of the legal actions,

- **ghostDists** = $\beta * \sum_{ghost \in s.ghosts} manhattanDistance(ghost.position, pacman.position)$ increases when Pacman moves away from ghosts,

- **foodScore** = $(s.arenaHeight + s.arenaWidth) * (s.arenaHeight * s.arenaWidth - s.foodAmount)$ increases when there's less food on the board,
- **scaredAvg** = $\gamma * s.scaredGhostsAmount$ increases when there are more scared ghosts on the board,
- **rand** = $N(0,2)$ (normal distribution with mean = 0, variance = 2)

The heuristic also evaluates state s with a value of 0 when there exists a ghost on the board with the same location as Pacman's (which means game over).

2) The motivation for the definition of the heuristic function as presented is to make Pacman more knowledgeable of the game elements:

score parameter serves as a decent indication that the action might be worthwhile. It is not enough to determine a good move, but it's generally a good rule of thumb for Pacman to seek higher scores in each turn. In our heuristic, this serves mostly for eating scared ghosts (as they grant a very high score).

foodDistanceScore parameter rewards Pacman for moving towards food. It is better for that purpose than simply checking the score because Pacman might not be able to eat any food in this turn, but usually should still attempt to get closer to food.

actions parameter gives Pacman a slight bonus for reaching states with a lot of available actions he can take. It serves as a tiebreaker.

ghostDists parameter rewards Pacman for staying away from ghosts. It's a good game plan for Pacman to be far away from ghosts in order to avoid a game over.

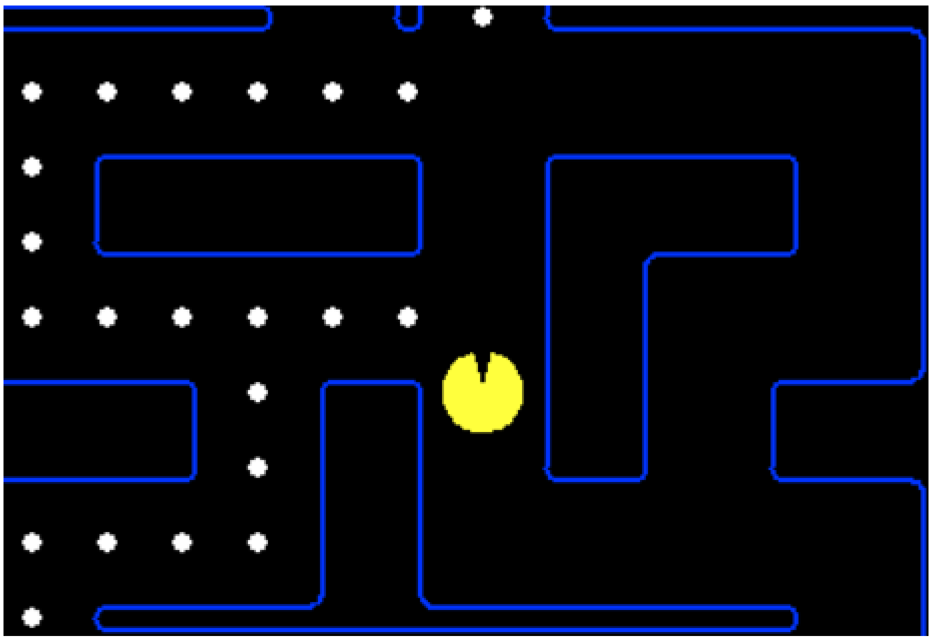
foodScore parameter rewards Pacman for eating food. It is necessary on top of the **score** parameter, because when Pacman eats food: the next closest food might be very far away. So much so that the heuristic would find it better not to eat the last food in that area to avoid such bad evaluation. With this boost: this scenario is migrated.

scaredAvg parameter rewards Pacman for eating capsules when some ghosts on the board would become scared because of it. This is useful because capsules do not grant score, so without this bonus: Pacman would not seek them out at all. However, capsules are extremely useful for Pacman in the long run as they provide protection from the ghosts, and the potential to earn a lot of points.

rand parameter adds a stochastic element: sometimes Pacman might get stuck near a wall and refuse to move towards food without it, because of the manhattan distance increasing while moving towards it along the available path.

We predict that this heuristic function results in a better player than the scoreEvaluationFunction, because it takes more of the game's elements into account. For instance, under the scoreEvaluationFunction heuristic, Pacman would not seek out the capsules because they do not increase his score. But as discussed: eating capsules is very beneficial for Pacman in most scenarios. Our heuristic

Additionally, the agent under scoreEvaluationFunction picks a random move when all the possible moves he could take result in the same score. Consider the probable scenario where all the moves Pacman may take do not result in an increase of score (no food or scared ghosts one block away from Pacman). Picking a random move in this scenario is far from optimal, as it might make Pacman stray away from a chunk of food that's located relatively near him. Illustration:



As we can see, there is food two moves away from Pacman. However, the scoreEvaluationFunction chooses the next move at random (going either up or down), because both actions result in the same score. A possible outcome might be a very long sequence of moves where Pacman keeps going down and up, never closing in on the food. It is more beneficial for Pacman to head towards the food in this scenario, so a heuristic function that highly evaluates a state that's located closer to food than the others is better.

Our evaluation function h does exactly that with the **foodDistanceScore** parameter, which rewards Pacman for heading towards food. **foodScore** serves to achieve that without making Pacman choose not to eat the food when he reaches it – a phenomena that might occur since Pacman prefers being close to food, but should prefer eating the food than just being near it.

Part C: Min-Max agent

1) We assume the following when creating the tree:

- The ghosts always choose the worst possible actions from Pacman's perspective in any given turn: we assume this because the Minimax algorithm will always deem the lowest possible outcome action the best suited for the minimizing players (= ghosts, in our case).
  This assumption is not necessarily right: the ghosts' behavior is unknown (we can't assume anything about them). A ghost's agent might be, for instance, RandomGhost, which picks a random, uniformly distributed decision each turn as where to go next: without paying any attention to Pacman.
- The ghosts' turns are ordered, resulting in knowledge about one another's turns. This might result in different outcomes for some of the ghosts in some scenarios.
  This assumption is not correct, because the ghosts and Pacman all make an action with no knowledge about the others' actions.
- We assume that the ghosts always assume Pacman takes the best possible action in every scenario.
  This assumption is not necessarily correct, because although Pacman does choose the best possible action from the Mini-Max algorithm: the ghosts themselves are unaware of this and correspond to the actions of their own agents.

3) Another possible implementation of the Min-Max algorithm is to create a layer in the tree for all the ghosts' agents at once. This is possible and does not affect the results because according to Pacman's rules: each turn is executed at the same time by all agents. This implementation scales the tree horizontally, which increases the branching factor significantly (each permutation of ghosts' decision should be considered in this layer), rather than vertically.

The advantages of this implementation are:

- Less recursive calls: we don't create a separate layer for each ghost, so in turn we don't call the method recursively as much.

The disadvantages of this implementation are:

- Harder to implement: all permutations must be considered in each stage.
- If one of the ghosts can reach Pacman (terminal state: game over) – the current implementation does not proceed to advance further in the tree to check the moves of the other ghosts. However, this new implementation wouldn't be able to stop the calculation in such a case.

Part D: Alpha-Beta agent

1) The new structure of the tree does not affect the alpha-beta algorithm:
the variant here is that multiple min-layers in succession result in more frequent changes to the β value (which updates every time there's a new minimum). No other changes are made.

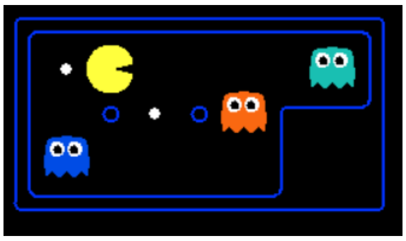
3) The AlphaBetaAgent should have the exact same **decision-making** as MinMaxAgent (solutions' quality is the same), since the pruning is done for sub-trees that can't possibly result in a better outcome for our agent (from AlphaBeta algorithm's correctness). Thus, given the same set-up (layout, ghosts, ghosts' agents), the two agents will tend to the same average score.

However, AlphaBetaAgent is superior in terms of **run-time** because it does not proceed to expand sub-trees that were pruned because they will not result in a better outcome for our agent. This improvement is greatly felt when running the agents side-by-side.

Part E: Expectimax agent for random ghosts

2) The RandomExpectimaxAgent is different than the Min-Max based agents by not assuming that the ghosts always choose the worst possible outcome for Pacman. As we've learned in class, the expectimax algorithm decides on a sequence of moves based on mean values (the random element in our case is the ghosts' decision making, which we assume here follows the RandomGhost agent). This might result in situations where Pacman might be "too afraid" of ghosts when using the Min-Max based agents, as opposed to RandomExpectimaxAgent.

We'll show an example to support this assumption is by running both agent types in the minimaxClassic layout:



This is an extremely small map with an extremely small amount of food (2 only). It follows from the image that the theoretical best possible solution can be achieved in just four steps (if the ghosts play nice).

When we run the Min-Max based agents in this layout, since the heuristic function evaluates losses very badly (as expected), the priority of the agent was to try and survive the worst maneuvers from the ghosts. It didn't necessarily even try to eat the food, because the weight of a loss is too costly when considered (and assumed to happen).

RandomExpectimax by comparison does not assume the worst possible actions from all the ghosts, so it was able to consider the actions of eating the food instead of just focusing on escaping the ghosts. Ironically enough, this led to a much higher win rate, because a win is achieved in this layout by eating just two foods which can be achieved very quickly if attempted at all. Here are the results:

Minimax:

Average score: 222.536
Win Rate: 713/1000

RandomExepctimax:

Average Score: 380.251
Win rate: 867/1000

The algorithms also differ in that RandomExpectimaxAgent assumes that ghosts' decision making each turn is distributed uniformly on equal probabilities (might not always be the case), whereas the Min-Max based agents assume the worst actions as discussed.

Part F: Expectimax agent for non-random ghosts

1) The RandomGhost agent's strategy is as follows:

in each turn, find all the possible moves and choose one at random, with evenly distributed probabilities. The distribution is uniformal.

3) The difference between the two implementations corresponds to the difference between the ghost agents' movement:

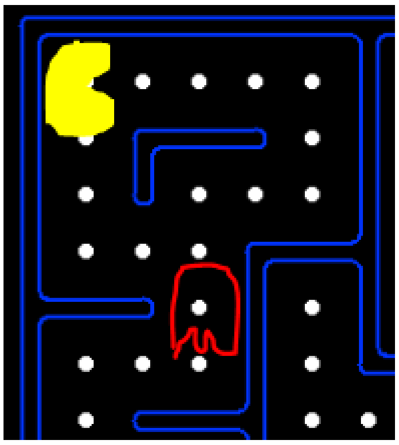
RandomExpectimaxAgent is implemented using the RandomGhost's actions' distribution, whereas DirectionalExpectimaxAgent is implemented using the DirectionalGhost's actions' distributions.

When directional ghosts are not afraid of Pacman, there is a probability of 0.8 that they will choose to move towards Pacman (and 0.2 for the opposite direction). If the ghost is scared: there is a probability of 0.8 that they will choose to move away from Pacman (and 0.2 towards Pacman). In turn, these ghosts are less in favor for Pacman.

The implementation of the two agents differs specifically in the ghosts' layers in the tree: in RandomExpectimaxAgent, the mean is calculated using the probabilities of the uniform distribution of RandomGhost, whereas in DirectionalExpectimaxAgent, the mean is calculated using the said probabilities of the DirectionalGhost's movement.

4) A sophisticated ghost agent might do some of the following:

- Communicate with each other (when there are two ghosts or more in play). A big liability of the ghosts is that they act very similarly to each other (specifically the DirectionalGhost which usually chooses to chase Pacman). This results in ghosts stacking up on top of each other, rendering them unable to utilize their numbers to overwhelm Pacman.
  A solution to this might come in the form of keeping a distance from one another, which will allow the ghosts to chase Pacman from different routes and block him.
- Guard capsules: Pacman's way of scaring the ghosts is by picking up the capsules spread throughout the map. If the ghosts beat Pacman to the capsules: Pacman wouldn't be able to scare them off. A heuristic for Pacman that doesn't credit capsules was shown to have a significantly worse win rate in most situations, so the ghosts might be able to win more often if they do this.
- Sticking to choke points. Consider the following scenario:



  Pacman is trapped in this area. If the ghost continues to chase Pacman: this will allow Pacman to easily flee. Instead, the ghost can make sure that it is always able to go back to this point faster than Pacman in order to safely chase him down without allowing him to escape.

Part G: Zero evaluation proposition

Our **zero-evaluation** is that RandomExpectimaxAgent and AlphaBetaAgent have the same average score. The **alternative** is that they are not equal.

The following table represent every possible scenario:

	H_0 *is false*	H_0 *is true*
Reject H_0	The agents aren’t equivalent, and the experiment's result confirms that	(α) The agents are equivalent, and the experiment's result contradicts it.
Accept H_0	(β) The agents aren't equivalent, and the experiment's result contradicts it	The agents are equivalent, and the experiment's result confirms that

The experiment:

We will run both agents on the same layout 1000 times. Then, we will calculate the amount of times RandomExpectimax won the competition.

If the amount of times RandomExpectimax won is between 450-550 we will conclude that the agents are equivalent in terms of score. Otherwise we will reject the zero-evaluation and conclude that the are different.

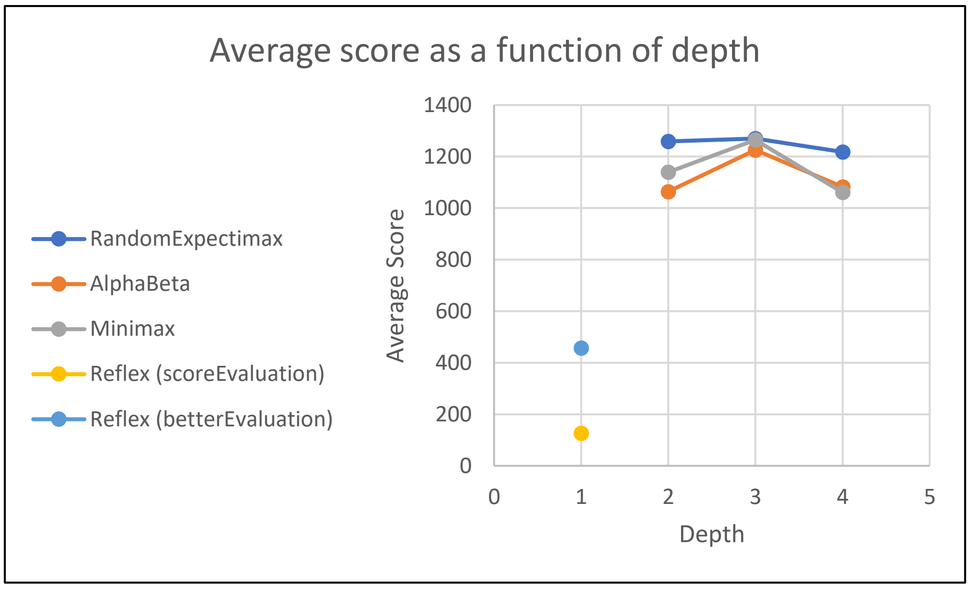
By calculating α we can see that the Pvalue is very low (according to the tutorial's table),

Which means that the possibility for a type 1 error is very low – if we find out that one of the agents is better than the other, the possibility the result is wrong is very small.

$$\alpha = P_{H_0}(X > 550\ or\ X < 450) = 2P_{H_0}(X < 450) = 2P_{H_0}\left(\frac{X - 450}{sqrt(250)} < \frac{450 - 500}{sqrt(250)}\right)$$
$$= 2\phi(-3.16) = 2 * 0.00079 = 0.00158$$

Part H: Experiments, results and conclusions

2) Agents' **scores** as a function of depth:



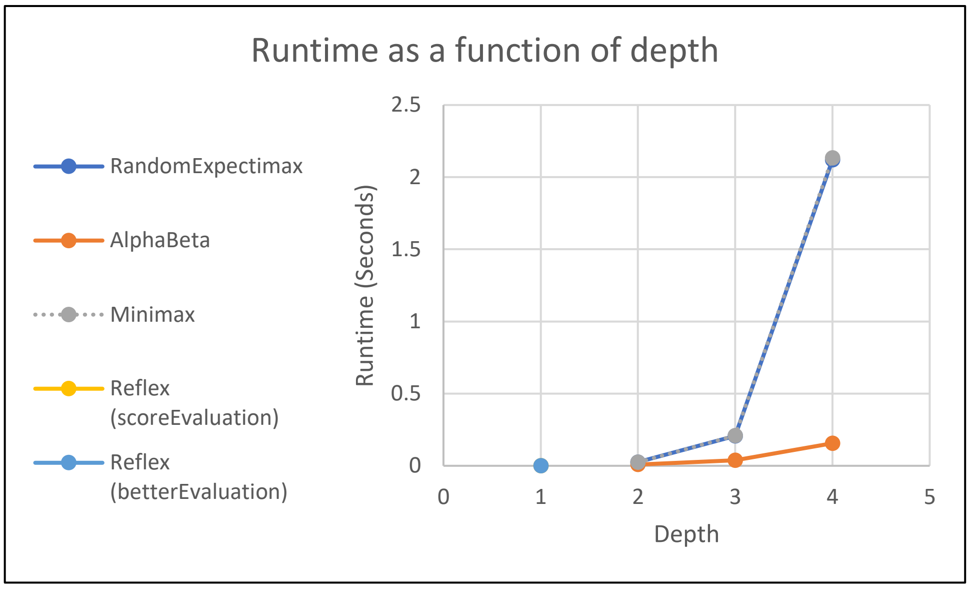
	depth = 1	depth = 2	depth = 3	depth = 4
Reflex (scoreEvaluation)	126.53			
Reflex (betterEvaluation)	457.056			
Minimax		1140.013	1264.685	1060.728
AlphaBeta		1064.243	1225.615	1082.73
RandomExpectimax		1259.285	1270.042	1218.07

3) We can derive the following conclusions from the graph and the table:

First, we can see that our h heuristic function performed a lot better than scoreEvaluation: the reflex agent scored nearly four times higher utilizing our heuristic. That is to be expected, as discussed in **Part B**. This proves our statement (for this dataset).

As we can see, running the algorithm with depth=3 improved our average result as expected (since Pacman could "see" further turns), but as we used depth=4, the score for each of the agents (especially MiniMax and AlphaBeta) has gotten worse. We think the reason for this result is that Minimax and AlphaBeta agents are too **pessimistic**, since they can see one more step ahead and assume the ghosts will always act in the worst possible way for Pacman. In turn, they take less risks than they should (Pacman does not choose to eat the food when ghosts are nearby). We see a slight decrease in the overall performance of the Expectimax agent, probably due to high variance that corresponds to deep search and random ghosts.

4) Agents' **runtime** as a function of depth:



	depth = 1	depth = 2	depth = 3	depth = 4
Reflex (scoreEvaluation)	0.0000643			
Reflex (betterEvaluation)	0.000232			
Minimax		0.023989635	0.209257649	2.132937658
AlphaBeta		0.008715369	0.037370383	0.156004789
RandomExpectimax		0.025652177	0.207825762	2.119351374

5) We can derive the following conclusions from the graph and the table:

First, we can see that the reflex agent that utilizes the scoreEvaluation heuristic chooses an action more than three times faster than the one utilizing our heuristic h. This is to be expected, because the default scoreEvaluation heuristic is an extremely simple function that returns the score and nothing else, whereas our heuristic h gives an evaluation based on many other parameters, which take more time to calculate.

Additionally, we see the difference between the reflex agents' runtime and the runtime of the agents based on multi-agent search algorithms: the reflex agents apply the heuristic function on all the possible actions and return the best one. The multi-agent search agents, however, apply the heuristic function on many states. The algorithms' expanded states have a completely different order of magnitude, which is evident by these runtime differences. That is to be expected, following what we've learned in class.

We can also see that RandomExpectimax and Minimax have nearly the same runtime for all depths, which is also expected, because both algorithms do not prune out any of the sub-trees during the search, and thus expand the same amount of states.

AlphaBeta, on the other hand, is much more efficient thanks to pruning: on depth 4, each turn decided on by the Minimax/RandomExpectimax agents takes more than ten times the time it takes AlphaBeta. This shows us the effectiveness of this optimization.

Finally, the graph shows us the stiff growth in runtime as we increase the search depth for multi-agent algorithms. This follows from what we've learned in class as well and demonstrates that an any-time approach might be better suited for this exercise when a time limit is present for each turn.

6)

	DirectionalExpectimax	RandomExpectimax
Game #1 – RandomGhost	3637 win	2366 win
Game #2 – RandomGhost	2918 win	2940 win
Game #3 – RandomGhost	2383 win	1649 lose
Game #4 – RandomGhost	2410 win	3220 win
Game #5 – RandomGhost	2939 win	2356 win
Game #6 – DirectionalGhost	1170 lose	721 lose
Game #7 – DirectionalGhost	878 lose	2078 win
Game #8 – DirectionalGhost	771 lose	402 lose
Game #9 – DirectionalGhost	1410 lose	784 lose
Game #10 – DirectionalGhost	1132 lose	1747 win

Average score: 2506.2, Win rate: 4/5 - random expectimax & random ghosts

Average score: 2857.4, Win rate: 5/5 - directional expectimax, random ghosts

Average score: 1146.4, Win rate: 2/5 - random expectimax, directional ghosts

Average score: 1072.2, Win rate: 0/5 - directional expectimax, directional ghosts

The results do not meet our expectations. In fact, we expected the exact opposite: we expected that the RandomExpectimax agent would perform better against **random** ghosts, whereas DirectionalExpectimax agent would perform better against **directional** ghosts. This expectation comes from the implementation of both algorithms: RandomExpectimax knows that the ghosts' actions are distributed uniformly, whereas DirectionalExpectimax knows that the ghosts' actions are distributed with 0.8 probability to chase Pacman and 0.2 otherwise (opposite when ghost is afraid). So, it follows that DirectionalExpectimax should model the reality of the games against directional ghosts better.

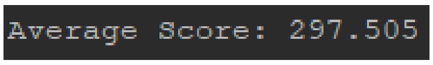
Possible reasons for this result are our heuristic function, which might've led the DirectionalExpectimax to too many losses (it does seem to reach higher scores compared to RandomExpectimax losses, just seems to lose more often), and also large variance (data sample of 5 games per scenario is not big enough).

7) We've covered this situation in **part E** for depth 2 when examining the difference between Minimax and Expectimax.

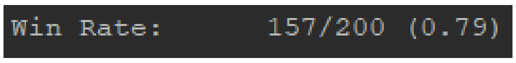
This map has a very big variance because it's very small and has many random ghosts. The outcome is largely determined by the ghosts' actions here. In addition, Pacman may win the game very quickly, too: there are only two pieces of food in the map, theoretically beatable in just four moves.

The .csv file we provided for 7 games was not enough to determine a clear winner. Here are the results of 200 runs with depth 4:

RandomExpectimax:

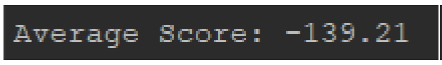


```
Average Score: 297.505
```

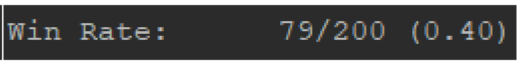


```
Win Rate:      157/200 (0.79)
```

AlphaBeta (=Minimax):



```
Average Score: -139.21
```



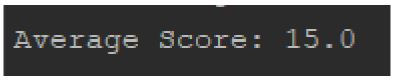
```
Win Rate:      79/200 (0.40)
```

These results show that **under our heuristic h**, Expectimax performs better in this map, because it does not assume the worst possible outcome from each ghost (which would lead to the agent expecting a game over at the very beginning, as AlphaBeta does). Instead, Expectimax is "braver", and chooses to eat the food instead: often resulting in a quick victory.

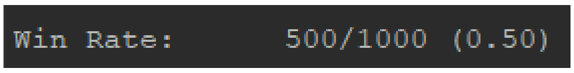
8) Like the previous section: we'll run a larger test (1000 runs).

This map starts with Pacman trapped between two ghosts. There are only four small food pieces at the other hand next to Pacman. Eating them would result in a victory. The ghosts' behavior is random, so it's up to chance if Pacman will be able to win under any circumstances.

RandomExpectimax:

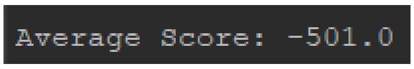


```
Average Score: 15.0
```

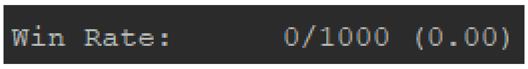


```
Win Rate:      500/1000 (0.50)
```

AlphaBeta (=Minimax):



```
Average Score: -501.0
```



```
Win Rate:      0/1000 (0.00)
```

We'll explain the difference between the two agents:

AlphaBeta searches up to depth 4, assuming all ghosts take the worst possible action every time. In 4 turns: Pacman would certainly lose in this map if the ghosts approached him. So, in order to maximize the score, AlphaBeta/Minimax agents decided to run into the ghosts (purposely lose the game), in order to minimize the amount of turns taken to finish the game (since each turn reduces the score by 1). This behavior, much like the previous section, comes from the fact that this algorithm is pessimistic, and does not see an opportunity to win. Each of the 1,000 runs in this scenario resulted in a score of -501, because Pacman chose to lose instantly.

RandomExpectimax, on the other hand, is aware of the ghosts' random behavior, so it can determine that a win is achievable. Thus, the agent tries to reach the food every single time. The results show that this strategy was much more successful (exactly because of the ghosts' random behavior), as Pacman was able to win 50% of the time and reach a much better average score
(15.0 compared to Minimax's -501.0).

If the ghosts were not choosing their actions at random, and did in fact chose to chase Pacman down: AlphaBeta would be better (would do exactly the same: lose on purpose and have a score of -501.0), since RandomExpectimax would always try to reach the food but would also lose to the ghosts as they reach him faster every single time (and it takes one to two additional turns to lose). This is evident in our results from the fact that each time the RandomExpectimax agent lost: its score was either -502 or -503, but never -501.

9) We'll conclude all the properties that we've seen from all the agents throughout this exercise:

First, the scoreEvaluationFunction heuristic that the original ReflexAgent used was outmatched by our improved betterEvaluationFunction heuristic described in **part B**. The results in experiments.csv support this statement in the general case.

When running the agents in all the layouts, we did not find such layout where our heuristic performed worse (in terms of scores) than scoreEvaluationFunction (utilized by ReflexAgent, as we did not run the score heuristic on the multi-agent search algorithms).

Our heuristic did, however, perform worse in terms of run-time (about three times longer), which is to be expected because it is much more complicated: it takes many of the game's elements into account and performs some relatively long calculations such as finding the minimum value for manhattan distances from all the available food on the layout from Pacman.

When comparing the agents to one another, it is evident that the Reflex agents are orders of magnitude faster compared to the multi-agent search algorithms. Among the multi-agent search algorithms, we saw in our results that AlphaBetaAgent is by far the fastest one: we've come to notice the effects of AlphaBeta pruning.

When playing on different boards, we saw that for higher values of depth and denser maps (small maps with many ghosts): Minimax and AlphaBeta agents' pessimistic approach decreased the average score of the games. In some layouts: Minimax would actively try to lose the game as quickly as possible, because it cannot comprehend any way to win (assumes ghosts always take the worst actions for Pacman). In these cases, the Expectimax agents were evidently better.

The Reflex agents did not handle layouts that occupied many ghosts. That is to be expected, because these agents do not look ahead at all: in turn, they kept getting Pacman into bad situations where he was trapped. Subsequently, when playing against directional ghosts (2 or more ghosts present), these agents performed very poorly.

We've also seen that Minimax and AlphaBeta agents had the same decision-making: their results are consistently tending to each other. This is to be expected, because the AlphaBeta algorithm is an optimization for Minimax that does not change the result of the algorithm, as we've seen in class.

Interestingly, our results show that Minimax/AlphaBeta agents performed **worse** in depth 4 than in depth 3. This is an unexpected result, which might've been different under another heuristic. In our case, we suspect that the reason for this is that looking four turns ahead made these agents "cowardly": their pessimistic behavior denied them from finishing the game in many occasions, due to fearing ghosts. Depth 3 seemed to be the best compromise for these agents as it performed better than depth 2. To support this argument: RandomExpectimax and DirectionalExpecitmax did not demonstrate this behavior.

In most cases, the best performance came from AlphaBeta and RandomExpectimax. DirectionalExpectimax seems to obtain better scores when facing directional ghosts but loses often: perhaps due to pessimistic approach or unsuitable heuristic.

Some interesting layouts that we found when comparing the agents are:

capsuleClassic: relatively dense map (three ghosts, small map). The Expectimax agents showed more consistent results here by "bravely" going after food more often than AlphaBeta/Minimax agents. When running a large sample of tests in this layout: we found that RandomExpectimax had the best results here.

trappedClassic which demonstrates the borderline between Minimax and Expectimax, as discussed in the previous sections. In this layout we've first come to notice the phenomena of Pacman losing on purpose due to pessimistic decision-making.

trickyClassic is a large map with 4 ghosts, which shows just how slow Pacman runs using the multi-agent search algorithms (with depth 4). These games took tens of minutes to finish.

testClassic is an extremely narrow map with 1 ghost and no walls. This map demonstrates the effectiveness of our improved heuristic function: it wins consistently in this map and finished the game rather quickly. The multi-agent search algorithms barely improve the performance here. The reason for this is that the absence of walls makes our manhattan distance-based heuristic a lot more accurate.

smallClassic is a small map with very little freedom for Pacman. When testing our agents in this map, we found that both Reflex agents performed similarly. This showcases that our heuristic function can be improved, specifically to consider the walls present on the board. In the other maps, however, our heuristic was a clear winner (for the Reflex agent at the very least).

When testing our RandomExpectimax agent on a small map with an extremely high depth under the scoreEvaluationFunction heuristic: we found some nice results. This approved our suspicion that given a vast amount of time to calculate all the necessary states: the score maximizing heuristic can perform quite well.

If the exercise limited the run-time of the agents rather than the search depth: an any-time solution would've been better. Our graphs show that the average turn calculation increases in an order of magnitude for each increase of depth. This applies to all the multi-agent search algorithms. So, running these algorithms for lower depths does not have a big toll on runtime, and is favorable under these constraints.

Part I: Competition

The player we chose for the competition is one of the agents that we've already covered in this report. The chosen agent is **AlphaBetaAgent** (from **part D**), under the **betterEvaluationFunction** heuristic h that we've covered in **part B**.

The agents we considered were one of these three: AlphaBeta, RandomExpectimax or DirectionalExpectimax. The Reflex agent is too simple and single-minded (do not look ahead), so it was not considered. The Minimax algorithm has the same performance as AlphaBeta in terms of average score but comes with a much costlier run time – so it was not considered as well.

We chose the AlphaBeta agent for two main reasons:

- It does not assume that the ghosts' agents are either RandomGhost or DirectionalGhost. Instead, it assumes that the ghosts' agents will always choose the worst action in Pacman's perspective, which we felt is a good compromise.
- It utilizes pruning as we've learned from the AlphaBeta algorithm in class. When testing, we saw a vast increase in runtime when running this agent compared to the others. So, we are less concerned about our agent finishing the game in 30 seconds.