# Search and Destroy

Jerry Yang

March 2021

## 1 Abstract

In this project, we worked on several AIs that plays a simple search and destroy game. The AI is placed in a landscape represented by cells which can be "flat", "hilly", "forested", or "cave"/"tunnels". These states of the cells all have different difficulties in terms of how hard it is for the AI to find the target. The AI will need to search the cells, and update its belief as it goes along and try to find the target in as few searches as possible. This project is done in Python and Pygames is used for the GUI. The algorithms below and of the Search and Destroy game and AI in general can be found in the searchdestroyai.py file. The searchdestroyui.py is used mainly for GUI purposes and does not encompass any of the algorithms below.

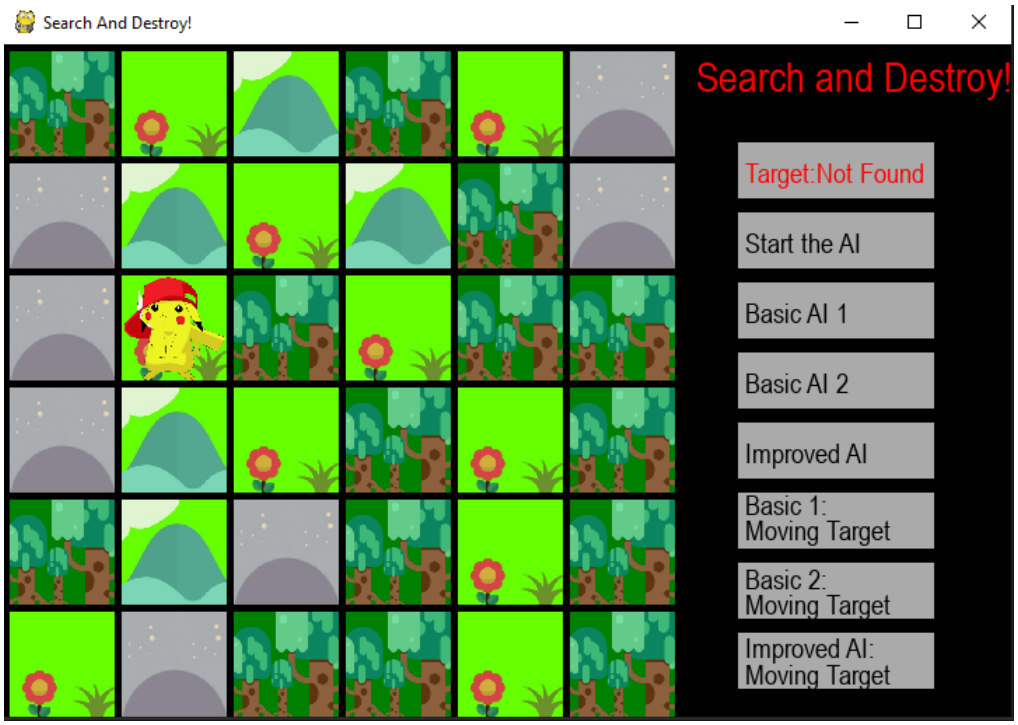


Figure 1: Random Snapshot of a given gamestate

## 2 Bayesian Beliefs: Which Cell is the Target in?

We can use Bayes Theorem and effectively update our belief system with the following. What we are trying to calculate is $P(TargetinCell_{ij} \mid Observations_t and Failure in Cell_j)$, The first thing we need to mention is our belief system. By definition, our belief system is $P(TargetinCell_{ij} \mid Observations_t)$, which you might notice, is part of our goal. We can then rewrite this as the following using Bayes Theorem.

$$P(A \mid B) = P(A)P(B \mid A)/P(B)$$

where A is that the current Cell contains the target, and B is the fact the search failed.

From here, we will split updating our belief into two cases. The first is the case we are updating the belief about the cell that we are searching (and return a failure). The second is the case that we are updating the cells that are not the cells we are searching. In the first case, we can start by plugging in the numbers. $P(A)$ or the belief the current cell contains the target is nothing more than our previous belief on the target (that we will be updating). Note that this will already be based on past observations. The next element we need to look at is $P(B \mid A)$, which can also be seen as the probability that we failed to find the target in the Cell given that A, the target is in the Cell is true. In this case, since we the cell we are searching is the one we are updating, the probability we will find the target is just the rate based on the terrain of the Cell. The last part we need to look at for this case is $P(B)$. We can calculate this from what we have with marginalization and split this into the following:

$$P(B) = P(A)P(B \mid A) + P(\neg A)P(B \mid \neg A)$$

With this, we can start plugging in what we know once again. $P(A)$, $P(B \mid A)$, we already know from the plugging into the theorem previously. $P(\neg A)$ We can calculate simply with $1 - P(A)$. $P(B \mid \neg A)$ Can be seen as the probability that we will fail to find the target given that the Cell is not in the the Cell we are updating. This is obviously guaranteed to happen so that would be a 1. With this, we then have everything we need to calculate $P(B)$ which in turn means we have everything we need to calculated our belief on the target or $P(A \mid B)$ for this case.

The next case is in the case that the cell we searched is not the cell we are updating. We are once again looking for the following:

$$P(A \mid B) = P(A)P(B \mid A)/P(B)$$

where A is that the current Cell contains the target, and B is the fact the search failed. For $P(A)$ and $P(B)$, the same rules apply as above but $P(B \mid A)$ is different as previously, it is the probability we fail to find the target in the searched cell given that the target is in the Cell we are updating. The probability we will fail is guaranteed as the target is not in the searched cell so this is just 1. With this, we then have everything to solve to update the beliefs of all cells given a failure, and these beliefs will stay up to date, making it so that there is no need to record the previous beliefs.

# 3 Bayesian Beliefs: Finding the Cell

We already have our belief state from the previous question and it captures the current probability the target is in the given Cell. In looking for $P(TargetFoundinCell \mid Observations)$, our belief system already takes the given observations into account. In this case, the probability we would find the target in a given cell is simply our belief times the rate given the terrain of the Cell. In other words, we can sum it up as the following.

$$P(Targetfoundin Cell_{ij} \mid Observations_t) = BeliefofCell_{ij} * RateOfCell_{ij}.terrain$$

# 4 Basic Agent vs Basic Agent

We created two Basic Agents. The first travels to the cell with the highest probability of containing the target, and then searches that cell and repeats this process. The second basic agent traveled to the cell with the highest probability of finding the target, searches the cell, and repeats the process. The results are shown below.

| AI 1 Distance | AI 1 Searches | AI 2 Distance | AI 2 Searches | AI 1 Total Score | AI 2 Total Score |
|---|---|---|---|---|---|
| 24408 | 5323 | 7346 | 1191 | 29731 | 8537 |
| 171781 | 17703 | 2723 | 1070 | 189484 | 3793 |
| 2417 | 2418 | 675 | 260 | 4835 | 935 |
| 5421 | 3303 | 7615 | 1645 | 8724 | 9260 |
| 410 | 411 | 3955 | 1529 | 821 | 5484 |
| 7132 | 3825 | 4718 | 1226 | 10957 | 5944 |
| 130020 | 14518 | 2332 | 859 | 144538 | 3191 |
| 540 | 541 | 3895 | 1032 | 1081 | 4927 |
| 14 | 15 | 122464 | 11075 | 29 | 133539 |
| 123713 | 14574 | 457363 | 30375 | 138287 | 487738 |

The average of the results are the first AI agent receiving a total average score of 52,848.7 and the second agent receiving a total average score of 66,334.8. From this we can say that the first AI agent performed better on average. However, this statement is actually very misleading because of the standard deviation that comes with these maze runs. If we look at the total score, the number of times the second Agent defeated the first was actually equal. However, in the last run in particular, the agent got insanely unlucky and took over 400,000 runs to find the target. Up til that point, if we just look at the first 8 trials, the second AI was clearly outperforming the former almost every run. If we think about it, this actually makes sense as in the case that the target is hiding in a cave, the agent will search the cell with the highest probability of finding the target. This means the agent will search the other cells extensively with the caves being lower on their priority list. The data is only based upon 10 maps, but from this data, it looks as if the second agent performs better on average, but will have very bad runs, especially if the target lies in a cave.

# 5 The Improved Agent

There are two points to our Improved Agent that we worked on. The first is that we wanted to keep the consistency of Basic Agent 2, more specifically, when the target is in a flat or hilly

area. At the same time, we want to alleviate its weakness in that the target being in a cave is like a nightmare for the agent. Because of this, we added a simulated annealing aspect to it and so there are cases when the Improved AI will "make a mistake". When this happens, the Agent will follow the rules of Basic Agent 1. One thing to note is it is quite likely that with the rules of the second agent, the belief the target is in one of the caves will tend to be higher. The basic idea is that now, as a result, Basic Agent 1 will choose the closest of these, and give the AI a chance to get out of the nightmare.

The second point is the fact that there are times when we will have to travel far to get to our designated Cell. When this happens, we pass by many Cells. In this ruleset where every Cell moved is the same cost as searching a cell, there may be Cells that are worth searching as they are along the way to our designated target. With that in mind we made a heuristic for for the Agent to follow. The Agent will take his belief of every Cell along with the chance of finding it, and generate a heuristic based on the 95th percentile of these values. As the agent passes each cell, if the chances of finding the cell is greater than the 95th percentile of the cells on the board, the agent will search that cell as he makes his way to the target cell.

The results of this are seen below.

| Improved AI Distance | Improved AI Searches | AI 1 Total Score | AI 2 Total Score | Improved AI Total Score |
|---|---|---|---|---|
| 382 | 129 | 29731 | 8537 | 511 |
| 3322 | 913 | 189484 | 3793 | 4235 |
| 107092 | 9700 | 4835 | 935 | 116792 |
| 4188 | 1642 | 8724 | 9260 | 5830 |
| 3778 | 1078 | 821 | 5484 | 4856 |
| 4861 | 2074 | 10957 | 5944 | 6935 |
| 4171 | 1682 | 144538 | 3191 | 5853 |
| 126400 | 11765 | 1081 | 4927 | 138165 |
| 1582 | 259 | 29 | 133539 | 1841 |
| 49612 | 6095 | 138287 | 487738 | 55707 |

If you recall, the averages of the first two Basic AIs were 52848.7 and 66334.8 respectively. The average for our Improved Agent came out to be a total score of 34072.5, which defeats the first two Agents with a decent margin. Of course, with the standard deviation being what it is and the data only being based off of ten runs, it's quite difficult to gauge the actual level of improvement. However, in theory, it should not only alleviate the second Basic Agent's weaknesses, but the total score should improve as well thanks to the heuristic.

If we were given enough time there are a few changes we would make. The first is to find the ideal rate for simulated annealing. We would run millions of trials or however many necessary to get a very good result and see where the ideal rate for our improved agent would lie. Likewise we would also move that onto figuring out what the ideal percentile is for our improved agent as a heuristic.

# 6 Additional notes

*How to run Code:*
Simply run searchdestroyai.py to play! If you want an AI to make a move, click Basic AI or Improved AI at the right.