Asteroids! Report

Introduction

Our goal in this project is to design two agents to play Asteroids. Both of our agents stem from the same abstract idea of an efficient agent: one that is both "mobile" in the sense that it attempts to avoid the nearest asteroid at any given time step and one that is additionally "stationary" in the sense that, when the agent does not need to move, it instead targets and eventually fires at the nearest asteroid. Following this notion, our first, and more mobile agent attempts to follow both of these notions. We dub this agent the "Mobile" agent. With the complexity of the mobile agent came a considerable amount of difficulty and problems in its development. And from the sweat and tears produced from this development came the origins of our second agent, the "Stationary" agent. This agent, following the second notion of our abstract idea, acts as a stationary turret that will never move and only shoot at its nearest targets. This agent was much simpler to develop, as the code itself is the mobile agent code, with its mobility portion removed. On average, the mobile agent performed slightly better than the random agent, while the stationary agent performed considerably better than the random agent. We believe this potentially ironic disparity between a simpler agent performing better than a complex one, and thus an agent with a much higher potential to perform well, is due to the great difficulty we had in implementing the mobile agent's features with accuracy.

Methods

i. Observation

The first step we took in making both the mobile and stationary agent was to identify where the asteroids are. We did this by parsing through the RGB space environment and building an array of asteroid center of gravities (centroids.) During the parsing, we also record the position of the ship, which is updated each time step if needed. To find the bounds of an asteroid, we ran a BFS every time we find an asteroid pixel.

We want the mobile agent to move away from asteroids that are too close to the ship, so we define a minimum distance that the ship is comfortable having asteroids in. Whenever an asteroid approaches too close, we use trigonometry to figure out the direct angle away from the centroid of the asteroid. It should be noted that the stationary agent also performs the same process of identifying the asteroid that is closest to the ship at a given time step, however this agent uses this asteroid only as a current target. Here is where problems started to arise: controlling the angle of the ship is exceptionally difficult.

ii. An Angle Problem

Our first attempt of angle determination was based on the idea that the ship is not an equilateral triangle. The ship's centroid and center must be different points and, furthermore, the centroid should revolve around the center as the ship turns. Therefore, if we draw a line through the centroid and center, we should know the angle of the ship. There were several problems with this approach. The first was that the ship is not made up of many pixels, so sometimes the centroid and the center actually occupied the same position, or otherwise produced very inaccurate angles due to them being very close in value. The second problem was that the ship isn't an isosceles triangle when turned to certain angles, so the line projected from the centroid and center doesn't accurately represent the ship's angle.

Our second attempt, which was the one we decided to go with, was to estimate the angle of the ship based on how many frames we press the turn left or right buttons. Our testing reveals that it takes about 30 ship-showing frames to make a full revolution turning. With this information, we formulated a function that maps a specific frame number, which in our source code we call "turn_frames", to an estimated angle in degrees:

$$\phi : \textit{frames spent turning right} \rightarrow \theta \text{, where}$$

$$\phi(x) = 90 - 12x.$$

Additionally, to estimate how many frames the ship will need to turn to be facing a certain asteroid, we also defined and used the inverse of ϕ:

$$\phi^{-1} : \theta \rightarrow \textit{frames spent turning right}, \text{ where}$$
$$\phi^{-1}(\theta) = \left[\frac{90-\theta}{12}\right].$$

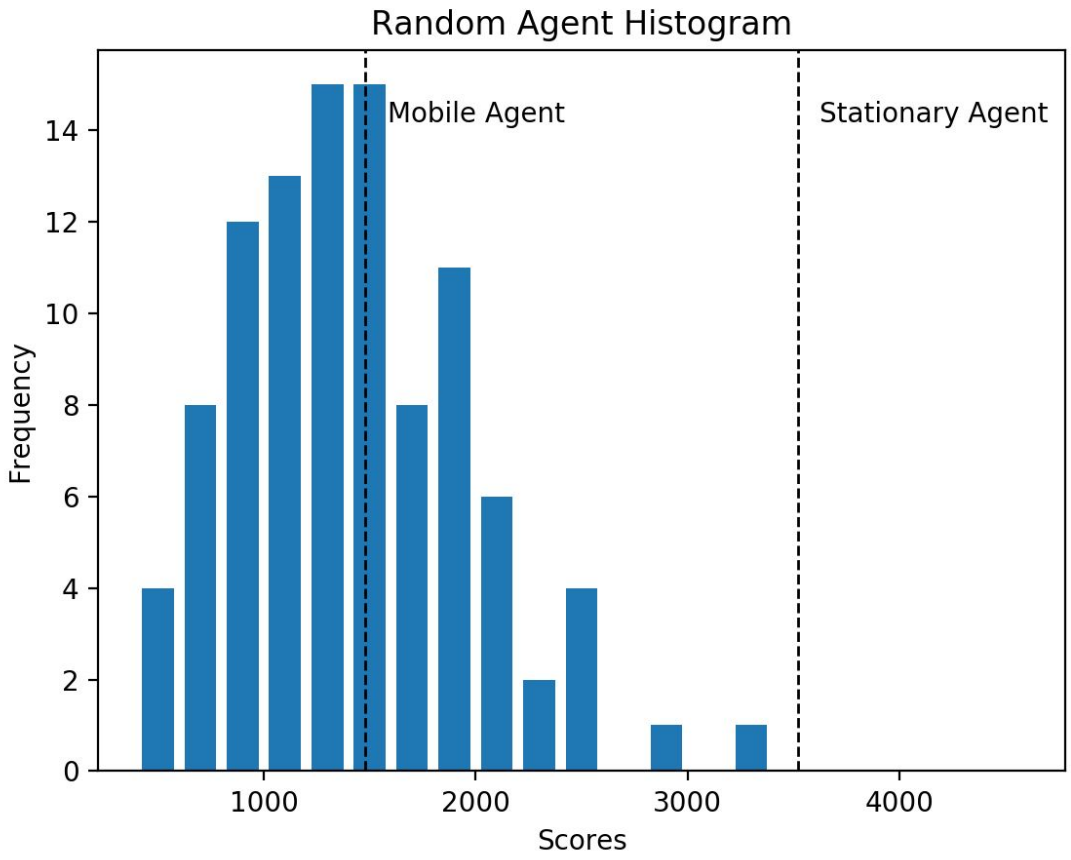
Moreover, this function was primarily used to convert the ship-to-minimum asteroid angle to an estimated target frame that the agent will then try to match by turning left and right based on conditions. We also simplified the dodging task so that the ship only needs to dodge to the left or to the right of the screen. For the early parts of execution, this idea worked very well. Our ship dodged the asteroids, but as estimation errors compound, this method becomes unreliable in the long run, as the agent would start to turn in wrong directions sometimes and even run into asteroids in the painfully worst case. Furthermore, this method of angle and turn frame estimation made shooting accurately spotty. Sometimes it would appear the agents had no idea where they were shooting, while sometimes they were surprisingly accurate.

iii. Run and Gun!

As the title dictates, we made our mobile agent have two states: running or gunning. The agent starts in the gunning state but switches to the running state when an asteroid gets too close, which was dictated by an internal "personal space" radius that the agent carries within its definition. The running state decides to move either right or left to avoid an incoming asteroid, and returning the steps to run away, i.e., the amount of right or left turns it has to perform before using the UP action to move away. To accomplish this, we take the current estimated angle of the ship and compare it to 180 degrees. Following this comparison, we either add or subtract turn frames (and consecutively return an action of RIGHT or LEFT) until we have met the left or right horizontal plane (with an estimated frame amount of ± 7.5, a quarter of the estimated 30 ship-showing frames to perform). In the gunning state, we aim at the asteroid closest to our current ship's angle, calculated by comparing the ship's angle to the angle of the line

connecting the center of the ship to the centroid of the asteroid. This aiming technique works best early on before all the angle estimation errors compound.

Results



The stationary agent scored 3520, while the mobile agent scored 1480. The random agent had a mean score of 1423 and a median score of 1370. The minimum score of the random agent is 410 and the maximum score is 3200.

Conclusions

We see that the stationary agent performs better than the mobile agent, which performs better than the random agent. It seems that staying at the center of the map is a very effective strategy, as it appeared that most asteroids generally spawn on the outskirts of the space map

and generally only move vertically, therefore usually leaving a good majority of the center untouched. It also appears that shooting well is easier to implement than dodging well. In fact, if implemented poorly, dodging can greatly negatively impact the agent's score. This is because in order to shoot well, one only needs to consider the angle the ship makes with asteroids and/or the asteroid's velocity vector. In order to dodge well, one needs to consider the movements of all asteroids, predict future environmental states, as well as have precise control of the ship.

Conversely, constantly shooting, despite sometimes being inaccurate at times much less of potential consequences, as seen in the random agent's high variance of scores, randomly shooting seems to actually benefit more commonly than not. As a result, attempting to implement a mobile agent appears to have only negatively impacted the overall performance of the agent, as the trials and tribulations of experimenting with the emulator's actions and following the pursuit of the scientific method led to a system that was perhaps too complicated for our coding ability with the emulator and our limited time. Despite this, we believe that given enough time and resources to develop a more precise way of measuring frame data and the like, the mobile agent has the potential to outperform the rest.