**Team Fruitbots**

Aaron Bockmiller

Shubham Mishra

Exploration into resource-limited adversarial AI stratagems

**Introduction**

The site <http://fruitbots.org/> is a game platform where user-programmed AI bots compete in a simple game to collect various fruits spread out across a randomly-generated 2D board. Bots can only move within the board by 1 space horizontally or vertically. Taking a fruit requires a full action as well. Fruits are divided into different types with varying amounts. Your goal is to collect fruits so that at the end, you have more fruits than your opponent in as many categories as possible. The game ends when the outcome can no longer change based on further fruit pickups. This website hosts thousands of bots to compete against and obtain a ranking. One further restriction is that there are both memory and time constraints placed on the bots, so any algorithms used will need to be optimized to operate within those specs.

We proposed to focus on exploring the effectiveness of various algorithms/strategies in a resource-limited environment. We strived to create the best-performing bots through comparing, combining, and optimizing various known search and competitive algorithms. Performance was to be measured based on ranking among the 1300+ bots already uploaded to the platform.

Note: During our project, the website went down, which had a number of effects. First, we could no longer use our original performance metrics or compete with other bots directly. We had a test environment in which we could compete against a baseline AI bot. Second, this test environment did not have the same constraints as the site, so this would need to be artificially implemented. Lastly, the only bots we could use to compare our own too were those that were written in Javascript and publicly available on [GitHub](https://github.com/scribd/robot-fruit-hunt/network/members). Copies of the important webpages were obtained from the WayBackMachine and saved for reference within our project folder.

**Design**

The first step of our design process was to research the different types of applicable algorithms that could be implemented into these bots. We found that they broadly fell under the following categories: random, logical, naive search, adversarial search, knowledge inference, and reinforcement learning/neural networks. Random consists of uninformed, random movements. Logic-based algorithms took into account the rules of the game and used those to optimize predetermined strategies to use. However, these are aware of the game board and all positions to use when implementing those strategies. Naive or simple search algorithms completely ignore the presence of the opponent. On the other hand, adversarial search algorithms do factor in the opponent's moves as well to determine the best move. Knowledge inference builds a KB which can be used to check logical statements for where to move. Finally, reinforcement learning utilizes neural networks that have been repeatedly trained on large sets of data to solve the problem. From there, we had to decide which of these categories we wanted to represent, and the specific algorithms to use.

Random was a completely basic design that would be quick to implement and demonstrate that our bots were working. They always failed to achieve any significant score or progress towards fruit collection. These always resulted in complete defeats.

For the logical algorithms, there were 3 that we used. First, the game provides an incredibly simple algorithm as a baseline for testing called SimpleBot. This AI simply searches for the closest fruit and moves towards it. The second bot was of our own design and used a utility function to weight the fruits based on distance, rarity, and if contested to select the highest rank. The third AI was an "opponent" bot that was publicly available. This logical algorithm was not as advanced as ours. It favored close fruit that had not yet been won or lost but did not factor in the rarity. It was called "themonkeytoucher\_bot1" based on the [author](https://github.com/themonkeytoucher/robot-fruit-hunt).

We found that the simple search algorithms did not sufficiently differ from the baseline bot or logical bots, and we did not expect good performance. Therefore, it was not implemented.

Adversarial search seemed to be the most promising choice to produce the best results in bot performance. As we know, the Minimax algorithm can often guarantee a win or tie in zero-sum, deterministic environments such as the Fruitbots game. Consequently, a proper implementation of this algorithm should always win. However, that is where the constraints come in. Calculation resources of time and memory are limited, so we cannot expect the Minimax algorithm to always finish. If time runs out without a move produced, your turn is skipped. Taking this into consideration, we chose to use the ExpectiMax algorithm for our bot. We thought this was the best choice as it not only assumes that the adversary is not making optimal decisions (which we know the SimpleBot baseline is not), but we can limit the depth of the search to ensure that the calculations are never too large. This algorithm can 'take a risk’ and end up in a state with a higher utility. During our implementation of this algorithm, we varied the depth to see its effects on results and try to match approximately what we expect the site constraints would have done. The other opponent bot that we explored (by [dima42](https://github.com/dima42/fruitbot)) did use Minimax. However, they limited the time that algorithm had to run in order to stay within the constraints. It would run Minimax until it finished, or time ran out, and give the best option it is able to find so far. We found that in practice, this generally gives a depth of around 10 moves in advance, but this can vary a lot based on system performance and complexity of the board.

The last type of algorithms to consider for implementation were reinforcement learning and neural networks. We found that using these was unfeasible within the scope of this project. Not only would it require a vast amount more work and research than the others but may not even work due to the time and memory constraints of the game. For instance, training the bot may be next to impossible if it continues to time out most turns without collecting any fruit to reinforce the model with. Lastly, the game seemed simple enough that better results couldn't be achieved with these algorithms. We were only able to find one example of a trained neural network bot, but its code was not publicly available to compete against. For these reasons, we forewent implementing either of these.

**Testing, Results, and Analysis**

As stated earlier, we were only able to test the bots performance against a standard baseline bot, rather than direct competition. However, we still feel that this produced useful results, as the testing methodology remained consistent across all AI that we tested. By specifying the random seed used in board creation, we tested each bot on the same selection of boards that represented varying sizes of boards and item amounts. We found that across these criteria, results were largely the same and did not vary within each category, so that is how we broke up the data representation below.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Many items | | | Medium items | | | Few items | | |
| Bot / Board | *Small board* | *Medium board* | *Large board* | *Small board* | *Medium board* | *Large board* | *Small board* | *Medium board* | *Large board* |
| Random | **L** | **L** | **L** | **L** | **L** | **L** | **L** | **L** | **L** |
| LogicBot | **W** | L | W | W | W | W | T | W | T |
| themonkeytoucher\_bot1 | W | L | T | W | L | T | L | L | **L** |
| ExpectiMax (depth=8) | W | L | L | T | L | **L** | T | L | **L** |
| ExpectiMax (depth=10) | W | W | L | T | L | L | T | T | L |
| Minimax | W | W | W | W | T | T | W | W | W |

*\* Bolded items represent the most dominant wins or losses, respectively*

From the data, we can see quite a number of interesting trends that correspond with the various traits of the algorithms. Going down the list, we start with the random bot, which completely loses all games. This is to be expected as the baseline SimpleBot has some logic to allow it to move around the board with the goal in mind.

From there, we come to the two bots following game logic-based algorithms. Our LogicBot performs very well compared to the baseline. It rarely loses or ties and by analyzing the boards, we can see that this seems to only happen in specific scenarios. When, by chance, the SimpleBot will take off towards a favorable cluster of fruit at the outset and the highest ranked fruit to go for first is far away from this, so the SimpleBot pulls into an insurmountable lead. This effect is mitigated by its high-ranking of rare fruits which are more valuable to the score. However, without this factor in the utility function, the effect is amplified in "themonkeytoucher\_bot1". As items grow fewer and make this phenomenon more likely, this algorithm confuses its planning and is simply outcompeted.

The adversarial search algorithms performed relatively well but were obviously subject to the constraints. ExpectiMax dealt best with a higher density (meaning more items per board-space). This is because it could only look so far ahead, which effectively limited its vision of the board. For instance, it could not plan to move 7 spaces away to collect an item, as this was beyond its depth. As the depth increased, the "vision" did as well and it could plan better, leading to improved results. Additionally, we know that Expectimax does not produce 'optimal' play, which is all but a guarantee that it will sometimes lose. Minimax, which was time-limited, rather than depth-limited, performed the best of the adversarial search bots. Often, only one or two crucial calculations at a higher depth (~9-14) was necessary to secure a win, so if one of these happened to finish or be in the finished part of the search, then it could be played and yield dividends.

**Conclusion**

From our tests and observations, we can come to several conclusions about the use of AI for this game. Overall, creating a logical algorithm to follow based on game rules and strategy was the best approach when working within constraints. We demonstrated that adversarial search algorithms improve with greater depth. Minimax produced great results, but is subject to varying depth for each turn, which becomes problematic with increasing board size. It can essentially be "luck" whether the best solutions come within the early parts of the calculation. Lastly, it is theoretically possible for better results with reinforcement learning, especially on larger boards, but this would be very complicated to implement.