Comparative Analysis of Search Algorithms for Autonomous Sailboat Navigation in a Game Environment

1st Jasmine Owens
Northeastern University
Portland, U.S.
owens.ja@northeastern.edu

2nd Josh Nougaret
Northeastern University
Portland, U.S.
nougaret.j@northeastern.edu

3rd Srikanth Bonkuri
Northeastern University
Portland, U.S.
bonkuri.s@northeastern.edu

4th Yue Yao
Northeastern University
Portland, U.S.
yao.yue2@northeastern.edu

***Abstract*—The Auto Sailor project aims to enhance gameplay of a browser-based sailing game by enabling autonomous vessels to be added to the game world. This paper will analyze a set of created search algorithms for autonomous sailboat navigation in a game environment. The game is centered around a single-sailed boat whose only propulsion is the wind, and whose only user-controls are sail angle adjustment (sheeting) and boat heading adjustment (rudder action). This work defines the environment configuration, describes the implementation details of each algorithm, and analyzes the results as obtained from simulation of the agents' performance in the game.**

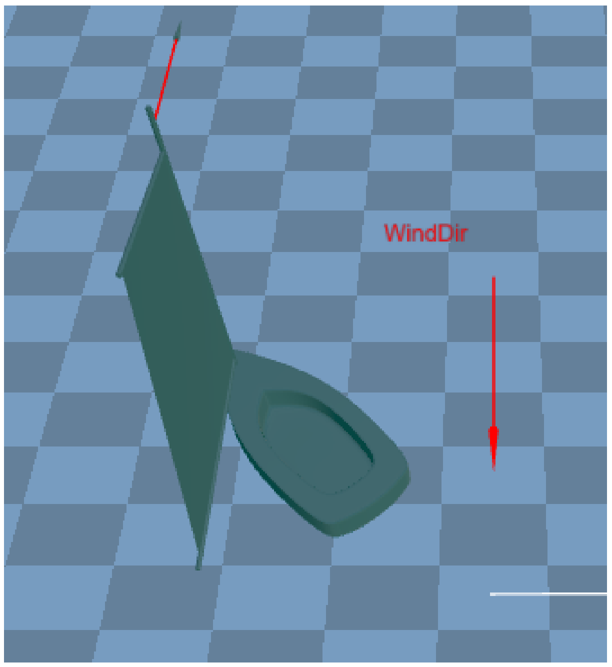

Fig. 1. Screen capture of sailing game [view code]

I. Introduction

This project focuses on developing a comprehensive game framework that leverages AI to enhance navigation, control, and decision-making processes for unmanned sailing vessels. The developed auto sailor agents operate by accessing the same controls that a user can adjust, namely boat heading and sail angle. The proposed model was developed in JavaScript, and it is feasible to be integrated with classical AI techniques, which is the approach we ultimately chose.

The game world consists of a plane of points, a boat object, and a wind object. During agent development up to the present time, the wind object is a static direction for all points in the game world. Boat speed is dependent on wind-sail angle and boat heading relative to the wind. Given these two values, the boat's target speed is a percentage of the wind vector's magnitude, as determined by a static lookup table dependent on wind-sail angle and boat heading relative to the wind. Once a target speed is obtained, the boat will accelerate or decelerate to match that speed. This behavior is meant to simulate momentum.

To implement the agents, we pass the boat object to the agent along with the wind vector. The agent makes multiple minimal copies of the boat object (without animating them) to predict future locations and speeds in order to make its decisions.[1] As a baseline comparison for all our agents, we created a Reflex agent. We then created three search agents: Greedy Depth-Limited Search, Depth-Limited Search, and A* Search.

Experimental results are carried out to verify the effectiveness of the proposed Auto Sailor agents; agents will be evaluated by how optimally they choose paths to the target, as determined by sailing experts on the game development team (Jasmine Owens and Josh Nougaret). They will additionally be judged by whether they cause animation lag, as this behavior is not acceptable in a game environment.

II. Literature Review

The field of autonomous sailing has gained considerable attention in recent years as advancements in artificial intelligence and robotics have paved the way for autonomous vehicles. Nowadays, the development of autonomous vehicles

[1]This will not be a scalable strategy; in a future implementation, we will likely wish to shift to a numbers-only calculation to avoid copying and manipulating physical geometries. This will likely require significant refactoring of the code base in order to avoid duplicating code.

is one of the most popular areas of investigation, especially for scientific explorer applications. The ability of these drones to stabilize and orientate themselves depends on a variety of external parameters like wind, water, obstacles, etc. Most of the previous projects developed in this area consider the wind force, velocity, and direction as the main factors that destabilize the system.

Self-steering systems for sailboats have been around since at least the early 1900s [4]. Early gear was entirely mechanical, designed simply to keep the boat travelling on a straight course. More modern systems incorporate various sensors and may include an AI agent that helps avoid obstacles or maintain a course to a fixed location [2] [3].

Several studies have explored the development of navigation and control systems for autonomous vessels. The integration of AI techniques, such as reinforcement learning and deep neural networks, has been investigated to enhance the decision-making capabilities of auto sailors. Research in this area has focused on optimizing trajectory planning, collision avoidance, and adaptive control strategies. Route optimization is a critical aspect of autonomous sailing, and machine learning algorithms have been explored to predict optimal routes based on historical data, weather patterns, and vessel performance. The literature reveals efforts to develop intelligent systems that can adaptively adjust sailing routes in real-time, considering factors such as wind speed, currents, and sea state.

A number of case studies and prototypes have been documented, showcasing the real-world implementation of AI-driven auto sailors. This game scenario is a simplification of real-world conditions; the boat in our sailing model can be assumed to maintain its current heading without user or agent input. Additionally, obstacle avoidance is beyond the scope of this project. However, some of the techniques used in obstacle avoidance navigation [1] [5] [6] [7] can likely be repurposed for the pathing problem we are attempting to solve.

Although we looked at the state-of-the-art, it was more complex than we would need for a game simulation; and because of the way the boat controls are set up, it was possible to model it like the Pacman search algorithms we implemented, which was a more accessible entrance point to developing an AI in this case.

III. Methodology

The agents are implemented in Javascript code files along with all the game-engine and graphics code that is required to run the game.

The user has access to two controls for the boat: boat heading and sail angle. We simplified the problem from a two-parameter search optimization to a single-parameter search optimization for our agents by reflexively adjusting the sail angle to a prescribed optimal angle. These optimal angles are based on the game's boat speed physics control matrix, and are calculated statically at run-time. This optimization is what makes the game environment most suitable for classical search techniques, as the available actions for the agent can be reduced to a simple set { `LEFT`, `RIGHT`, `AHEAD` }.

The Reflex agent generates a set of three future states, based on the action list. The agent then chooses the action that produces the maximum radial velocity towards the target. The choice pool is reduced in certain cases (namely, upwind near the wind) in order to prevent the boat from stalling; the agent does not know that if it turns towards the wind, in the future it will slow down and eventually stop, so we prescriptively prevent this from happening. The Reflex agent is also encouraged to turn away from the wind when its speed is below a certain threshold, as an additional incentive towards correct behavior.

Greedy DLS focuses on exploring paths that appear to be the most promising based on a heuristic evaluation. The Greedy DLS agent recursively creates a search tree, and at each step, chooses the locally best result based on Euclidean distance. An additional penalty is added to the heuristic at low speeds to actions that result in low speeds. The depth of the search tree is limited dynamically. When the vessel is farther away from the target, it generates a longer tree. The idea is that nearer the target, fewer actions are needed to successfully navigate.

The DLS agent explores a tree or graph up to a certain depth, limiting the depth of the search to control computational complexity. The DLS agent uses a priority queue to manage states and choose the best actions, which does result in better pathing. The implementation is otherwise similar to greedy DLS. Trying a few modifications to this in an effort to speed it up along with some sort of pruning idea could improve the algorithm, but we did not implement these strategies at this time.

The A* algorithm is a popular pathfinding algorithm that balances completeness and optimality, making it well-suited and extensible for navigating through different wind conditions, avoiding obstacles, and reaching a specified destination. Our A* agent uses a time estimation function based on distance to the target and radial speed for the heuristic and manages node expansion through a priority queue in heap memory. The A* algorithm combined with better resource management for system memory makes this the technically most impressive algorithm implementation.

In a comparative analysis of search algorithms for autonomous sailboat navigation, the evaluation criteria are essential to objectively assess the performance of different algorithms. We considered several.

- Travel Time - Estimate the amount of time it takes for the boat to reach the target point.
- Computational Time - Estimate the time taken by each algorithm to compute a solution. Faster algorithms are desirable, especially for real-time decision-making in dynamic environments.
- Path Optimality - Assess how close the generated paths are to the optimal solution. Optimal paths are those with the minimum cost.
- Ease of Implementation - Consider the ease with which each algorithm can be implemented and integrated into the autonomous sailboat system. Evaluate factors such as code complexity and resource requirements.

Travel Time is used as a heuristic for Reflex and A* agents, and so is perhaps not a true candidate for evaluation. Computational time is additionally difficult, as real-time performance is browser, machine, and instance dependent. [2]. We do print out the length of the action sequence list, but lower scores here do not mean better performance, and neither do higher. Node expansion is something we may look into for future evaluation. For now, when evaluating our agents, we visually assess the path optimality while the simulation is running, looking for large sail angles that take the boat far away from the target or for start-up lag when the algorithm begins its calculations.

IV. Results

In this section, we present the results obtained from the simulations performed on the four different agents created for this project: Reflex Agent, Greedy Depth-Limited Search agent, Depth-Limited Search agent, and A* agent.

- The Reflex agent utilizes a set of three future states and chooses the action that maximizes the radial velocity towards the target. Although it successfully navigates to the target, the code implementation is complex and somewhat fragile, and the generated route is sub-optimal.
- The Greedy Depth-Limited search agent recursively creates a search tree and selects the locally best result based on Euclidean distance. The depth of the search tree is dynamically limited. However, the agent tends to turn far away from the target due to its greedy decision-making approach and does not consider long-term payoff, and sometimes also 'misses' the target and has to backtrack. The agent is also considerably slower than the reflex agent and therefore unsuitable for a game environment.
- The Depth-Limited search agent, an improvement over the greedy Depth-Limited Search agent, uses a priority queue to manage states and choose the best actions. Its results are slightly better than the greedy Depth-Limited Search agent, but it is noticeably slower to run. Despite attempts to speed it up, the slow startup time makes it unsuitable for a game environment.
- The most effective agent implemented in this project is the A* agent. It performs the search efficiently using a priority queue in heap memory. The A* agent finds the most optimal control sequence out of all the search agents. The A* agent does not have a notable start-up lag.

V. Discussions

The success of the reflex agent in reaching the target is dependent on special casing to prescribe an action. This reduces the amount of calculating power required, meaning the agent performs very well time-wise with no start-up lag. However, its sub optimal route and complex code indicate room for improvement.

The inferior performance of the greedy DLS agent can be attributed to its greedy decision-making approach, which ignores long-term payoff considerations. This leads to turning far away from the target point. The slow start-up runtime is probably due to recursion costs in expanding nodes.

While the depth-limited search (DLS) agent shows better routing compared to the greedy DLS agent, it tends to display an even slower runtime than that of the greedy approach. Pruning criteria and further tweaking may be necessary to make the DLS agent more efficient.

The A* agent stands out as the most effective algorithm implemented in this project. By utilizing a priority queue and heap memory, it achieves efficient search performance and shows no start-up lag. The A* agent's speed, nearly equivalent to the reflex agent, makes it a promising choice for the game environment, and it is better at finding a route that truly optimizes the time spent travelling towards the target.

In conclusion, the results obtained from the simulations demonstrate the advantages and limitations of each agent in this search environment. While the reflex agent and A* agent deliver favorable performance, the DLS and greedy DLS agents fall short in terms of efficiency and optimality. Further improvements can be made by exploring alternative heuristics, transition models, and algorithms to enhance the behavior of the agents.

None of the agents seem to adjust the heading in a way that truly mimics real-world sailing, but this will likely change as further improvements are added to the sailing physics. Additional adjustments to the penalties in the algorithms' heuristics could also encourage more realistic behavior.

VI. Conclusion

Through the creation and evaluation of multiple agents, we have showcased the diverse strategies that can be employed to solve the same problem.

The last agent we create and evaluated, the A* search agent, has successfully met the initial requirements of the project, demonstrating its ability to navigate the boat to a set point without negatively impacting game performance. As such, further AI development for this environment will likely begin with enhancements of the A* search agent.

We may also continue to experiment with the Reflex agent; although the A* agent has clearly superior navigating performance, the Reflex agent has proved to be a valuable benchmark for evaluating the performance of new algorithms. In other words, if a newly developed agent cannot better the performance of the Reflex agent, it should be considered a failure.

Overall, we are highly satisfied with the project's outcomes. It has provided us with valuable learning experiences and has effectively showcased the potential of AI agents in addressing complex problems.

References

[1] Helmi Abrougui, H. Dallagi, and S. Nejim, "Autopilot Design for an Autonomous Sailboat Based on Sliding Mode Control," Automatic Control and Computer Sciences, vol. 53, pp. 393–407, 2019. doi: 10.3103/S0146411619050031

[2] in testing, all of these were found to produce different results

[2] Helmi Abrougui, Samir Nejim, Saber Hachicha, Chiheb Zaoui, and Habib Dallagi, ”Modelling of an Intelligent Control Strategy for an Autonomous Sailboat - SenSailor.” Ocean Engineering, vol. 241, December 2021. doi: 10.1016/j.oceaneng.2021.110038

[3] H.C. Burmeister and M. Constapel, ”Autonomous Collision Avoidance at Sea: A Survey,” Frontiers in Robotics and AI, vol. 8, September 2021. doi: 10.3389/frobt.2021.739013.

[4] W.J. Daniels and H.B. Tucker, *Model Sailing Craft, 3rd Edition*. Chapman & Hall, 1952, p. 239. Excerpt accessed through Vintage Model Yacht Group.

[5] B. Erfianto, A. Adrianne, and R. R. Arif, “On the Experiment of Path Planning Using Multi-way Points with A* Algorithm for Autonomous Surface Vehicle,” KINETIK, vol. 6, no. 2, May 2021. doi: 10.22219/kinetik.v6i2.1244

[6] A. Moreno-Ortiz, D. Sánchez-Orozco, L. López-Estrada, C. Tutivén, Y. Vidal and M. Fajardo-Pruna, ”Modelling of an Intelligent Control Strategy for an Autonomous Sailboat - SenSailor,” 2022 5th International Conference on Advanced Systems and Emergent Technologies (IC_ASET), Hammamet, Tunisia, pp. 34-38, March 2022. doi: 10.1109/I_ASET53395.2022.9765928.

[7] J. Ren, J. Zhang, and Y. Cui, “Autonomous Obstacle Avoidance Algorithm for Unmanned Surface Vehicles Based on an Improved Velocity Obstacle Method,” ISPRS International Journal of Geo-Information, vol. 10, no. 9, p. 618, Sep. 2021. doi: 10.3390/ijgi10090618.