

Software Engineering Department  
Braude College

Capstone Project Phase B

24-2-D-14

[Git Repo](#)



**Submitted by:**

**Guy Pariente**

**Almog Elbaz**

**Supervised by:**

**Prof. Miri Weiss-Cohen**

**Dr. Edi Shmueli**

## Contents

<b>General Description .....</b>	<b>3</b>
<b>Project Overview .....</b>	<b>3</b>
<b>Objectives .....</b>	<b>4</b>
<b>System Architecture &amp; Implementation.....</b>	<b>4</b>
<b>Research &amp; Development process .....</b>	<b>5</b>
<b>Literature Review and Initial Research .....</b>	<b>5</b>
<b>Algorithm Design and Planning.....</b>	<b>5</b>
<b>Initial Algorithm Attempts .....</b>	<b>7</b>
<b>Shift to an Iterative Approach.....</b>	<b>7</b>
<b>Simulator Development.....</b>	<b>8</b>
<b>Activity Diagram .....</b>	<b>9</b>
<b>Class Diagram.....</b>	<b>10</b>
<b>Environment Setup and Simulation Framework .....</b>	<b>11</b>
<b>Tools.....</b>	<b>11</b>
<b>Challenges &amp; Solutions .....</b>	<b>12</b>
<b>Results &amp; Conclusions.....</b>	<b>14</b>
<b>Lessons Learned .....</b>	<b>15</b>
<b>References .....</b>	<b>16</b>
<b>User Guide.....</b>	<b>17</b>
<b>?How to run .....</b>	<b>17</b>
<b>?I've ran it – what's now.....</b>	<b>17</b>
<b>Manual Mode.....</b>	<b>18</b>
<b>Automatic Mode .....</b>	<b>20</b>
<b>JSON Format.....</b>	<b>21</b>
<b>The Simulation.....</b>	<b>22</b>
<b>Stats Screen .....</b>	<b>24</b>
<b>Maintenance Guide .....</b>	<b>25</b>
<b>Hardware Requirements.....</b>	<b>25</b>
<b>Software Requirements.....</b>	<b>25</b>
<b>Installation .....</b>	<b>25</b>
<b>Building an executable .....</b>	<b>25</b>
<b>Code Organization and Structure.....</b>	<b>26</b>
<b>Adding/Modifying Features.....</b>	<b>27</b>
<b>Conclusion.....</b>	<b>27</b>



Fig .1. Maritime Transportation

## General Description

### Project Overview

Maritime transportation is a vital component of global trade, accounting for over 80% of international commerce[12]. While it is an efficient and cost-effective means of moving goods, it also introduces significant risks, particularly in busy sea lanes where vessel collisions can occur. Studies indicate that 80-85% of maritime accidents result from human error[13], highlighting the need for autonomous collision avoidance systems that can improve safety and efficiency.

This project focuses on the development of an autonomous collision avoidance algorithm designed to comply with the International Regulations for Preventing Collisions at Sea (COLREGs). While COLREGs define clear rules for ship encounters (such as head-on, crossing, and overtaking scenarios), they are traditionally designed for two-ship interactions. However, real-world maritime environments often involve multiple vessels navigating simultaneously, leading to complex multi-ship collision scenarios. The challenge lies in applying COLREGs dynamically to multi-ship environments while maintaining compliance.

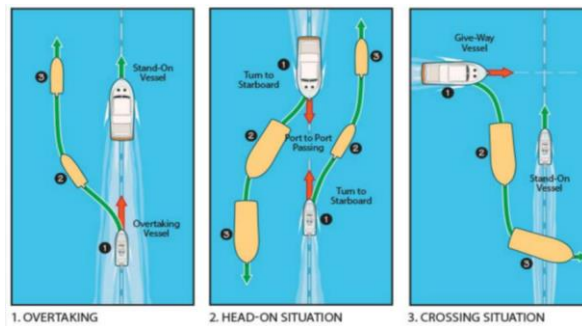


Fig .2. Visual explanation of the 3 COLREG rules we focus on – Overtaking, Head-on, Crossing.

To test our algorithm, we developed a Python-based maritime simulator that models various ship interactions using discrete time steps. The simulator allows us to test different ship configurations, collision scenarios, and avoidance strategies. Ships are assigned roles based on COLREG rules (e.g., give-way or stand-on), and

maneuvering decisions are made based on a CPA/TCPA (Closest Point of Approach / Time to Closest Point of Approach) computation framework.

## **Objectives**

The main objectives of our project are:

- Developing a collision avoidance algorithm that ensures autonomous vessels operate in full compliance with COLREG.
- Implementing a simulation environment to test the algorithm across various conditions.

## **System Architecture & Implementation**

The system is designed as a modular and extensible framework, consisting of three main components:

### **1. Collision Avoidance Algorithm**

- Computes CPA/TCPA for all ship pairs.
- Classifies encounters into head-on, crossing, or overtaking situations.
- Assigns give-way and stand-on roles based on COLREG.
- Conducts incremental maneuvering to optimize safe distance.
- Ensures ships return to their intended courses smoothly after avoiding collisions.

### **2. Python-Based Simulator**

- Simulates multiple autonomous vessels in a defined sea environment.
- Uses discrete time steps to update ship positions and check for collisions.
- Visualizes ship movements, safety zones, and decision-making in real-time.
- Supports scenario testing for two-ship and multi-ship interactions.

### **3. User Interface & Scenario Management**

- Allows users to configure scenarios, including number of ships, speeds, and destinations.
- Provides real-time simulation visualization with ship trajectories.
- Includes a log system to track decisions, maneuvers, and rule compliance.
- Supports scenario replay and debugging tools for algorithm refinement.

# Research & Development process

## Literature Review and Initial Research

The foundation of this project began with an in-depth literature review focused on understanding existing maritime collision avoidance solutions and identifying gaps that our algorithm could address. We reviewed a wide range of research papers covering both traditional optimization methods and advanced machine learning-based approaches. These included solutions like Distributed Coordination Optimization (Negenborn et al.[5]), Ant Colony Optimization (Tsou et al.[2]), Evolutionary Planning (Michalewicz et al.[3]), Dynamic Bayesian Networks (Cho et al.[4]), Multi-Agent Reinforcement Learning (Wei and Kuo[7]), and Artificial Potential Fields (Lyu and Yin[8]).

A key observation from our review was that while many existing solutions provided effective collision avoidance strategies, most were not fully compliant with the International Regulations for Preventing Collisions at Sea (COLREG). This is especially significant because COLREG compliance is crucial for real-world maritime applications. Many solutions optimized trajectories for efficiency rather than adhering to COLREG-defined encounter roles, and most focused on two-vessel scenarios rather than complex multi-vessel situations. Our goal was to address these gaps by developing a collision avoidance algorithm that follows COLREG rules while scaling to multi-vessel interactions.

## Algorithm Design and Planning

After completing the literature review, we began brainstorming possible approaches for solving the collision avoidance problem while ensuring compliance with COLREG. We considered several algorithmic paradigms, including heuristic-based optimization, tree search, and iterative collision checking. One of the initial questions we faced was how to define and detect a "collision" in our simulation environment. This led to the decision to use the concepts of Closest Point of Approach (CPA) and Time to CPA (TCPA).

In maritime navigation, a collision is not simply defined by ships physically overlapping; instead, we must anticipate potential collisions in advance. To achieve this, we use two key metrics:

- Closest Point of Approach (CPA): The shortest distance that will occur between two ships if they continue on their current headings and speeds.
- Time to Closest Point of Approach (TCPA): The time remaining until this closest distance is reached.

By calculating CPA and TCPA for every pair of ships at each time step, we can predict and prioritize potential collision risks. The lower the CPA, the greater the risk of collision. The lower the TCPA, the more urgent the collision risk is. This allows us to sort encounters by urgency and resolve them in order, ensuring that the most immediate threats are addressed first.

In our implementation, the CPA and TCPA are computed using the following formulas:

Given two ships A and B with positions  $p_A, p_B$  and velocity vectors  $v_A, v_B$  the relative position and velocity are:

$$r_0 = p_B - p_A$$

$$v_{rel} = v_B - v_A$$

TCPA is computed as:  $t_{CPA} = \frac{-r_0 \cdot v_{rel}}{|v_{rel}|^2}$ .

If  $t_{CPA} < 0$ , the CPA occurred in the past, so we can clamp it to zero.

The distance at CPA is then:  $CPA = |r_0 + t_{CPA} \cdot v_{rel}|$

If CPA is below a predefined safe distance, a collision is considered imminent and requires resolution.

We also had to decide whether to model ship movements in a continuous manner or to use discrete time steps. After evaluating both options, we chose discrete time steps due to the following reasons:

- Easier to implement and simulate real-time ship movements and decisions.
- Enables iterative collision checking and course correction within each time step.
- Allows for controlled, step-by-step simulation of multi-vessel interactions.

With these design decisions made, we defined three core components for the system:

- Ship Class: Represents individual ships, including attributes such as position, heading, speed, and dimensions.
- Simulator Class: Manages the simulation environment, including updating ship positions, detecting collisions, and resolving conflicts.
- COLREG Utility Functions: Includes functions for classifying encounters (head-on, crossing, or overtaking), assigning give-way and stand-on roles, and determining the appropriate maneuvers based on COLREG rules.

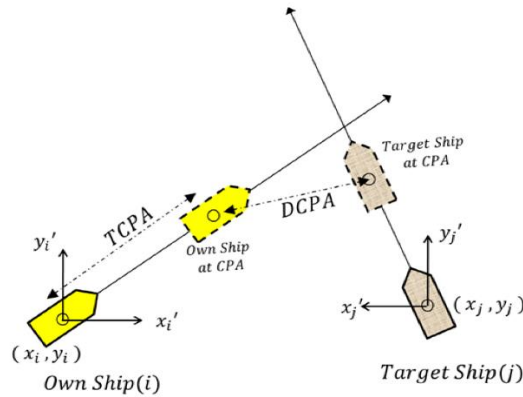


Fig.3. CPA/TCPA visualization.

## Initial Algorithm Attempts

Our first attempt at implementing the collision avoidance algorithm was a tree-search approach with backtracking. The idea was to represent the simulation as a tree of possible states, where each state represents a snapshot of the positions, headings, and roles of all ships at a given time step. The algorithm would search this tree to find a sequence of maneuvers that avoids collisions while complying with COLREG.

However, we encountered several challenges with this approach:

- **State Space Explosion:** The number of possible states grew exponentially as the number of ships and time steps increased, making the search computationally expensive.
- **Difficulty Representing States:** Capturing the entire state of the simulator at each time step and backtracking to previous states was complex and inefficient.
- **Unrealistic Assumptions:** The tree-search approach assumed perfect knowledge of future ship positions and did not account for the dynamic and uncertain nature of real-world maritime scenarios.

## Shift to an Iterative Approach

Due to the challenges with the tree-search approach, we pivoted to a simpler but more dynamic iterative approach based on CPA/TCPA calculations. In this approach, the simulator performs the following steps at each time step:

1. **Collision Detection:** For each pair of ships, calculate the CPA and TCPA. If a potential collision is detected (CPA below a safe distance), prioritize collisions by sorting them based on TCPA and CPA.
2. **Role Assignment:** Classify the encounter as head-on, crossing, or overtaking, and assign give-way and stand-on roles based on COLREG.
3. **Collision Resolution:** The give-way ship attempts incremental starboard maneuvers to increase the CPA. If no improvement is possible, the stand-on ship may take corrective action as a last resort.
4. **Reversion to Course:** Once ships have been collision-free for several time steps, they gradually return to their original headings.

This iterative approach allowed us to dynamically check for and resolve collisions at each time step without the need for complex state representations or backtracking. The use of CPA/TCPA calculations made the collision detection and resolution process more efficient and mathematically grounded.

## **Simulator Development**

To test and validate the algorithm, we developed a Python-based maritime simulator using Pygame for visualization and interaction. The simulator includes the following features:

- **User Interface:** Allows users to configure scenarios, including the number of ships, starting positions, and destinations.
- **Real-Time Visualization:** Displays ship positions, headings, and safety zones in real-time.
- **Scenario Management:** Supports loading and saving scenarios for testing different configurations.

This simulator provides a controlled environment for testing the algorithm under various conditions, from simple two-ship encounters to complex multi-ship scenarios. It allowed us to observe the algorithm's behaviour, identify issues, and make improvements iteratively.



## Activity Diagram

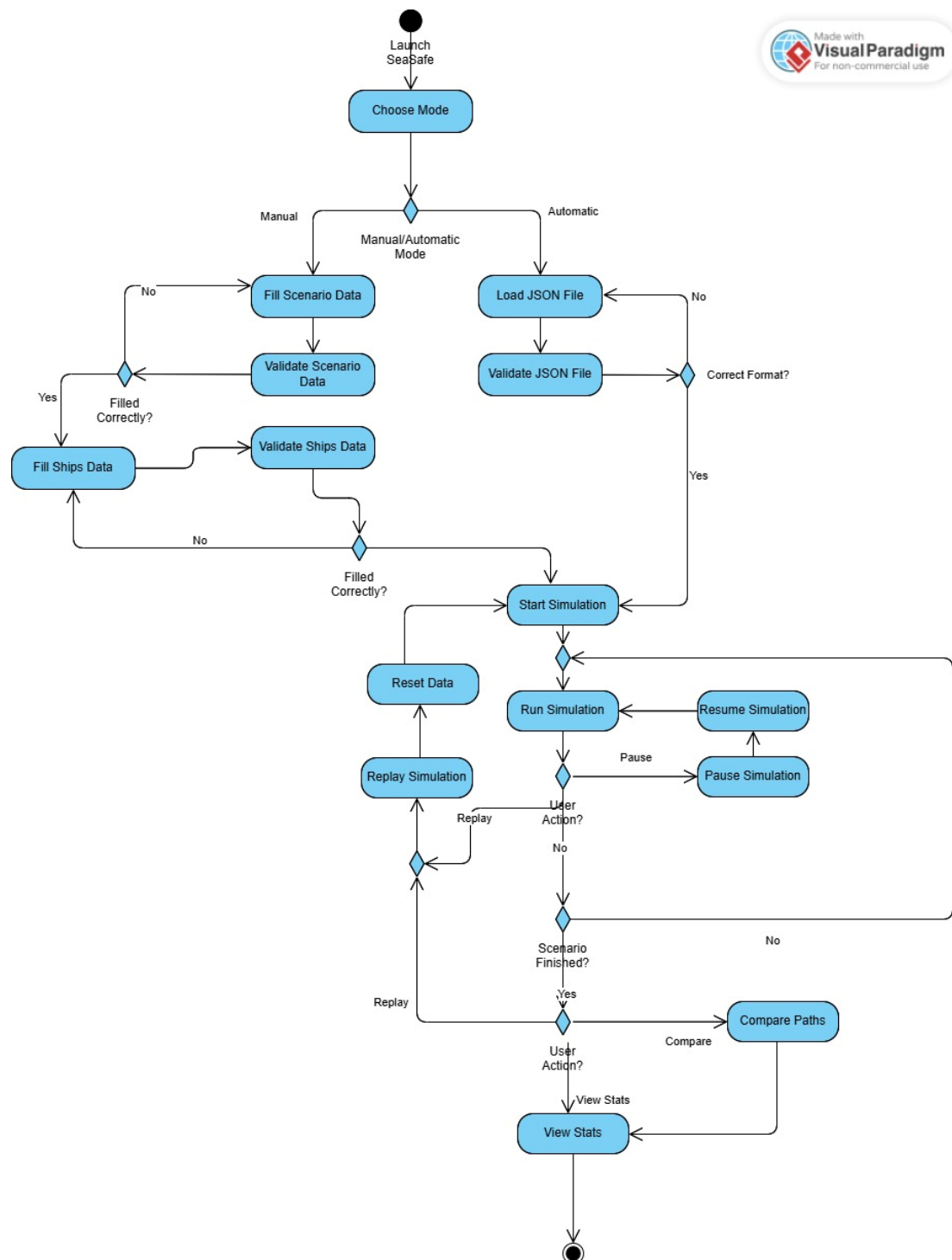


Fig 4. Activity Diagram.

## Class Diagram

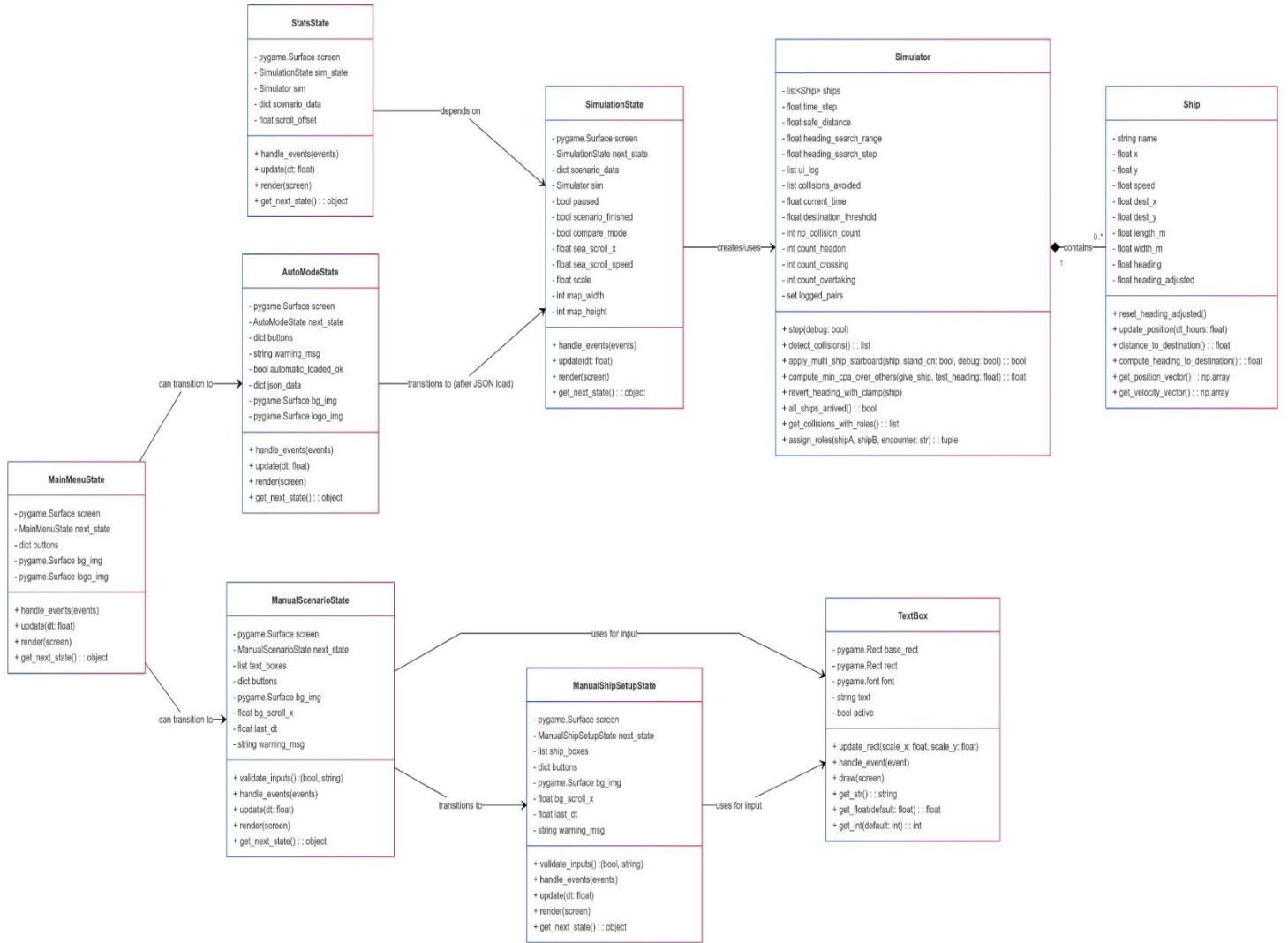


Fig 5. Class Diagram

## Environment Setup and Simulation Framework

Before implementation, we needed to design a simulation framework that accurately models real-world maritime navigation. This required several key design decisions:

1. Time Step vs Continuous simulation:  
We chose a discrete-time simulation approach, where ships update their positions at fixed time intervals (e.g., every 30 seconds)
2. Grid representation:  
To provide a structured maritime environment, we designed an NxN map grid where each unit represents one Nautical Mile, ship positions, speeds and headings are computed using real-world units(knots, degrees, meters) and map size is configurable.  
The grid based approach allows for scalable and clear ship movement tracking.
3. Collision detection and Safety Zones:  
We defined a safety zone – a circular area around ships that must remain clear, if there's another ship in the safety zone – that's a collision. We detect collisions by calculating CPA and checking if it is under our safety distance.
4. Ship Maneuverability and Heading Adjustment  
Each ship can adjust its course incrementally based on the constraints, such as maximum heading change per time step, or heading step (smallest angle increment checked for collision resolution).

## Tools

1. Python: Provided a straightforward syntax for building and iterating on complex collision logic, offering numerous relevant libraries.
2. Pygame: Formed the foundation of our 2D visualization, enabling step-by-step display of ship maneuvers, collision zones, and updated headings.
3. Tkinter: Delivered basic GUIs for scenario loading (JSON files) and interactive parameter selection, making it easier for testers and end-users to run experiments.
4. PyInstaller: Facilitated creating a single executable, simplifying deployment to collaborators without requiring a full Python install.
5. Matplotlib: Supported prototype analysis, e.g. plotting distances over time or comparing baseline vs. starboard-only approaches for specific collision scenarios.

## Challenges & Solutions

During the development of this project, we faced several challenges related to both algorithmic complexity and system implementation. Below, we outline the major challenges we encountered and the solutions we devised to overcome them.

### 1. Difficulty in Implementing Tree-Search for Collision Avoidance

Initially, we attempted to develop a tree-search algorithm to solve the collision avoidance problem by representing the simulation as a state-space search problem. However, we faced two significant obstacles:

- Defining a "state": We needed a way to store the entire maritime environment (including ship positions, velocities, and headings) at a given time step and then search through possible future states. However, accurately capturing and efficiently representing the dynamic simulation proved extremely difficult.
- Exploring the state space: The vast number of possible future states made backtracking computationally infeasible. Expanding every decision tree branch for all ships quickly led to exponential growth in complexity, making it impractical to compute optimal maneuvers in real-time.

Solution:

We shifted to a more mathematical approach using CPA/TCPA calculations, which allowed us to dynamically assess and resolve potential collisions at each time step instead of relying on a predefined search tree. This iterative approach significantly improved performance and allowed for real-time decision-making without being constrained by state-space limitations.

### 2. Implementing a Complex Multi-Ship System in Python Without Prior Experience

Neither of us had prior experience in developing large-scale simulation systems or working extensively with Python for real-time applications. Designing an interactive simulation with a physics-based model required learning both the language and best practices for structuring large projects.

Solution:

Before writing any code, we spent significant time carefully planning the project structure, breaking it into well-defined classes and modules. This included defining:

- A Ship class to encapsulate ship properties (e.g., position, velocity, and size).
- A Simulator class to handle ship interactions, detect collisions, and update movements.
- A COLREG module to handle encounter classification and decision-making.

By modularizing the system and adhering to clear object-oriented principles, we reduced complexity and made debugging significantly easier. As we gained familiarity with Python, we also leveraged libraries such as PyGame for visualization and NumPy for efficient numerical calculations to enhance performance.

### 3. Handling Multi-Ship Scenarios with COLREG Compliance

One of the major challenges was ensuring COLREG compliance in multi-ship interactions. COLREG was originally designed for two-ship scenarios, with clear rules dictating how a give-way ship and a stand-on ship should behave. However, in a real-world maritime environment, ships often encounter multiple vessels simultaneously, leading to nested, conflicting collision-avoidance decisions.

Solution:

To extend COLREG to multi-ship scenarios, we prioritized encounters pairwise:

- Instead of attempting to resolve all ships at once, we sorted collisions based on TCPA and handled the most urgent conflicts first.
- Each step, the simulator re-evaluated CPA/TCPA after every maneuver to ensure ships adapted dynamically to changing situations.
- If a give-way ship couldn't improve CPA within constraints, the stand-on ship was allowed to take limited corrective action as a last resort, maintaining compliance while increasing safety.

This hierarchical decision-making process helped balance COLREG compliance with practical collision resolution in complex multi-ship scenarios.

#### 4. Preventing Zigzagging and Erratic Maneuvers

In early simulations, ships frequently displayed zigzagging behavior, where they made excessive small course corrections, leading to inefficient and unnatural movements. This was caused by the CPA/TCPA-based approach reacting too aggressively to small changes in distances, leading ships to adjust their course too frequently.

Solution:

To reduce erratic maneuvers, we implemented several improvements:

- Smoothing course corrections: Instead of making drastic heading changes at each step, we limited the maximum allowed heading change per iteration.
- Collision-free stability threshold: Ships now require a certain number of consecutive collision-free steps before returning to their original heading, reducing unnecessary back-and-forth corrections.
- Collision prioritization: Instead of reacting to all potential CPA reductions, ships now only maneuver when the CPA falls below a critical threshold, preventing overcorrection.

These refinements resulted in smoother ship movements, preventing unnecessary zigzags while still maintaining effective collision avoidance.

## Results & Conclusions

The primary objective of our project was to develop a collision avoidance algorithm that ensures safe and efficient navigation for autonomous ships while adhering to COLREG regulations. Our final implementation successfully meets these goals in two-ship scenarios, with ships reliably following COLREG-compliant maneuvers for head-on, crossing, and overtaking encounters. For multi-ship interactions, we demonstrated that a systematic pairwise collision-resolution approach allows COLREG-based decision-making to scale, despite the framework originally being designed for two-vessel situations.

Through iterative CPA/TCPA-based evaluations, our method dynamically adjusts heading changes in real-time, ensuring that vessels maneuver safely while minimizing unnecessary course deviations. The algorithm effectively balances compliance with COLREG, efficiency in route navigation, and computational feasibility, making it a viable solution for autonomous ship navigation in complex environments.

The following screenshots (Fig.3 and 4) are screenshots of the simulator of 2-ship scenarios and multi-ship scenarios, they showcase our algorithm in action. Each ship has a trail that represents her path towards her destination, with our algorithm applied.

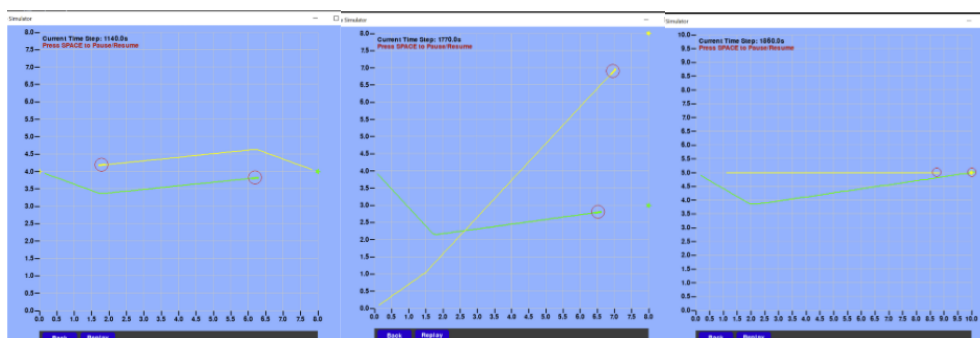


Fig.6. – Head-on, Crossing and Overtaking scenarios

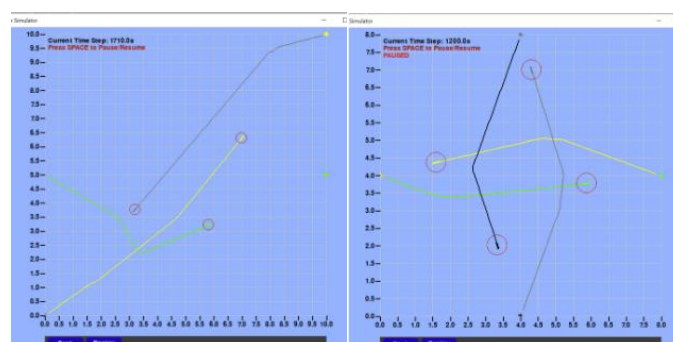


Fig.7. – 3-4 Multi-Ship Scenarios

While our approach provides a functional rule-based solution, certain limitations remain. Handling larger fleets and integrating speed adjustments could further enhance the system's realism and adaptability. Future work may explore hybrid approaches, combining optimization techniques with rule-based logic, to further improve efficiency and decision-making in dense maritime traffic.

In conclusion, we successfully developed and tested a COLREG-compliant collision avoidance system, demonstrating its effectiveness in both simple and complex multi-ship scenarios. This work serves as a foundation for further research in autonomous maritime navigation and real-world applications of AI-driven collision avoidance systems.

## Lessons Learned

Throughout the development of our collision avoidance algorithm, we encountered various challenges that provided valuable insights into both our approach and potential areas for improvement. While our project successfully implemented a rule-based COLREG-compliant navigation system, certain decisions in our development process could have been adjusted to enhance efficiency and scalability.

One major takeaway is that we should have explored Reinforcement Learning (RL) earlier as an alternative to tree search. RL is widely used for complex decision-making in dynamic environments, making it highly relevant for collision avoidance problems. Unlike tree search, which requires manually defining and exploring the state space, RL can learn optimal policies through simulation, adjusting its behavior dynamically based on past experiences. Additionally, various existing RL frameworks could have accelerated our development process by providing pre-built tools for decision-making and environment modeling.

Another key lesson is that our early implementation phase lacked rapid prototyping. Initially, we spent significant time on designing a structured class-based architecture without testing key algorithmic concepts quickly enough. A more agile approach, where we experimented with basic navigation and collision detection first, could have led us to identify flaws and inefficiencies earlier, saving development time.

Lastly, a more user-friendly and interactive simulator interface would have enhanced the project's usability. While our simulation provides useful visualizations, adding real-time scenario editing, parameter tuning, and AI-driven analysis tools would have made testing and validation more efficient.

## References

1. Li, L.-N., Yang, S.-H., Cao, B.-G., and Li, Z.-F., "A summary of studies on the automation of ship collision avoidance intelligence," Journal of Jimei University, China, Vol. 11, No. 2, pp. 188-192 (2006)
2. Ming-Cheng and Hsueh, Chao-Kuang (2010) "THE STUDY OF SHIP COLLISION AVOIDANCE ROUTE PLANNING BY ANT COLONY ALGORITHM," Journal of Marine Science and Technology: Vol. 18: Iss. 5, Article 16.
3. Modeling of Ship Trajectory in Collision Situations by an Evolutionary Algorithm, Roman S´mierzchalski, and Zbigniew Michalewicz, Senior Member, IEEE
4. Intent inference of ship maneuvering for automatic ship collision avoidance, Yonghoon Cho, Jungwook Han, Jinwhan Kim
5. Li, S., Liu, J., & Negenborn, R. R. (2019). Distributed coordination for collision avoidance of multiple ships considering ship maneuverability. Ocean Engineering, 181, 212-226. <https://doi.org/10.1016/j.oceaneng.2019.03.054>
6. Andreas Nygard Madsen, Magne Vollan Aarset, Ole Andreas Alsos, Safe and efficient maneuvering of a Maritime Autonomous Surface Ship (MASS) during encounters at sea: A novel approach, Maritime Transport Research, Volume 3, 2022.
7. Guan Wei and Wang Kuo, COLREGs-Compliant Multi-Ship Collision Avoidance Based on Multi-Agent Reinforcement Learning Technique, 2022
8. Lyu, Hongguang. (2018). COLREGS-Constrained Real-time Path Planning for Autonomous Ships Using Modified Artificial Potential Fields.
9. COLREG-Compliant Optimal Path Planning for Real-Time Guidance and Control of Autonomous Ships, Raphael Zaccane 2021
10. Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4th Edition). Pearson.
11. Jordan Gleeson, Matthew Dunbabin, Jason J. Ford, COLREG Scenario classification and Compliance Evaluation with temporal and multi-vessel awareness for collision avoidance systems, Ocean Engineering, Volume 313, Part 3, 2024, 119552, ISSN 0029-8018,
12. Review of Maritime Transportation, United Nations trade & development
13. Hasanspahić, N.; Vujičić, S.; Frančić, V.; Čampara, L. The Role of the Human Factor in Marine Accidents. *J. Mar. Sci. Eng.* 2021, 9, 261. <https://doi.org/10.3390/jmse9030261>



# User Guide

This section is for the end-users that are going to use SeaSafe.

## How to run?

There's a `seaSafe.exe` file that is an executable file that you can double click and run, before running, make sure you have the following installed in your PC:

- Python
- PyGame
- NumPy

In case you do not have those installed, please visit the [Python's website](#) for instructions for how to install those.

## I've ran it – what's now?

Great! When you open SeaSafe, you'll see the main menu screen:

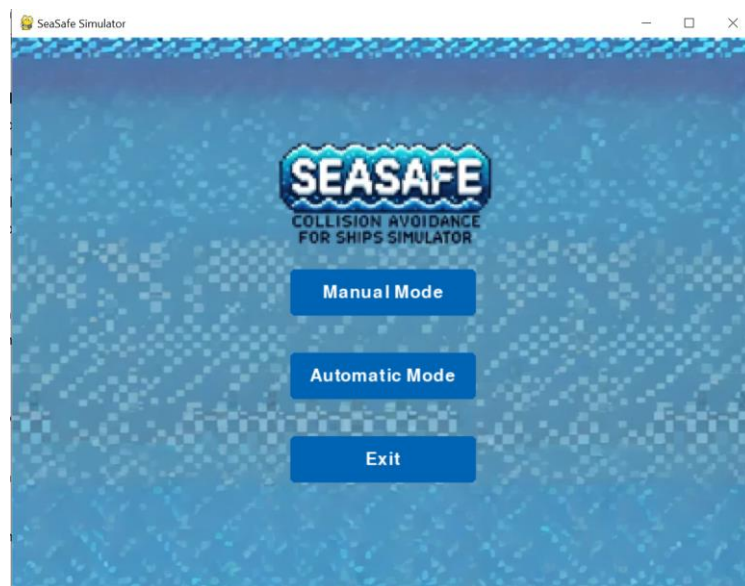


Fig.8. SeaSafe Simulator main screen

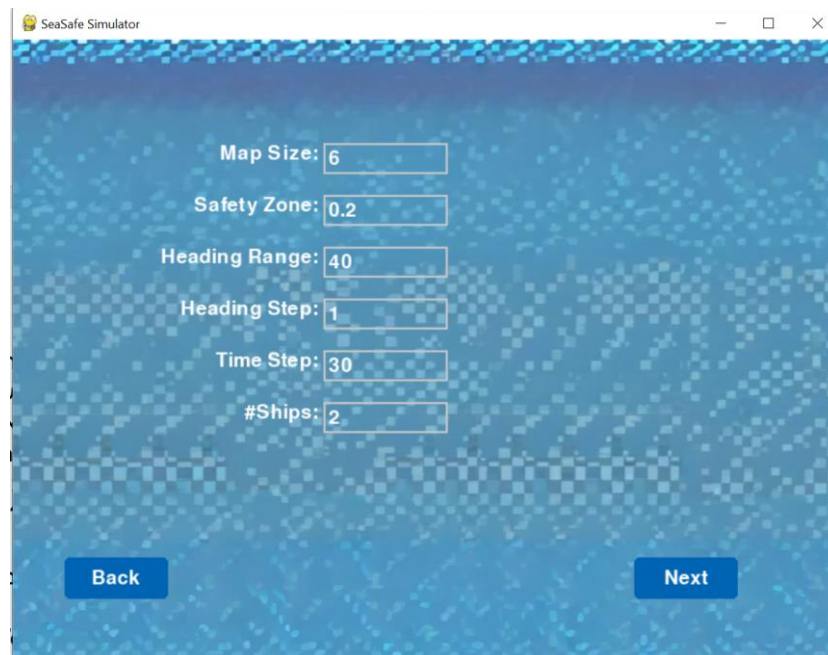
There are 3 buttons:

- Manual Mode – this will open up a manual setup mode – where you can define a scenario by the UI.
- Automatic Mode – this will open up a automatic setup mode, where you can input JSON file with the setup. We provide some setups for example in the “scenarios” folder. Feel free to explore those first before creating one of your own.  
If you are not familiar with JSON syntax – please visit [this introduction](#) and our JSON Format section.
- Exit button.

## Manual Mode

So, you feel like the UI setup is the one for you and that's great, because we provide the tools for that.

The following screen is the first manual mode setup screen:



SeaSafe Simulator

Map Size:

Safety Zone:

Heading Range:

Heading Step:

Time Step:

#Ships:

Fig.9. Manual Scenario setup screen

Here you need to fill the initial simulation environment you want to setup, let's review each box:

- Map Size – This number is in Nautical Miles, you should provide a number between 0 and 20.
- Safety Zone – This is in Nautical Miles, that's a safety circular area around ships that must remain clear of other vessels.
- Heading Range – What is the maximum turn that ship can take at each time step? This value is between 0 and 50.
- Heading Step – The algorithm first check if heading step is enough to avoid collision, for example, If you set up heading range = 30 and heading step = 10, the algorithm will check if 10 degrees maneuver is enough, if not, 20 degrees and so on until 30.
- Time Step – In seconds, discrete time step.
- #Ships – Number of ships in the scenario, this value is between 2 and 8.

After you fill the values and press 'Next', you'll see the following screen:

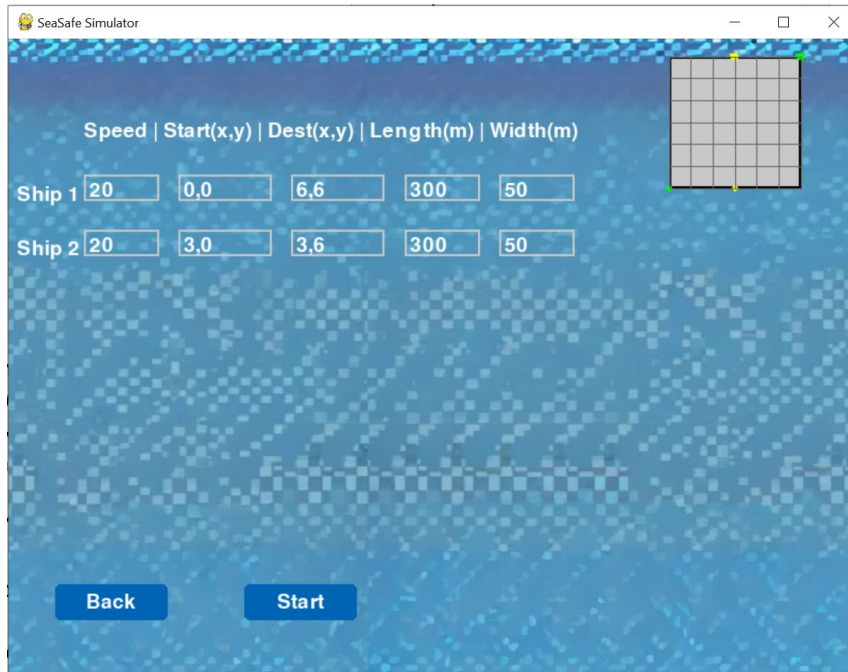


Fig.10. Manual ship setup screen

This screen is the **Manual ship setup** screen, here you will fill the following for each ship:

- Speed: speed of the ship in Knots, value between 0 and 50.
- Start(x,y) – Start position of the ship in the map.
- Dest(x,y) – Destination position of the ship in the map.
- Length – Length of the ship in meters, value between 100 and 800.
- Width – Width of the ship in meters, value between 50 and 200

In addition, you can see the minimap in the top right corner, it represents the map and the ships start and destination points (star – destination, source – circle).

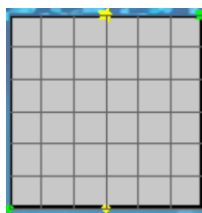


Fig.11. minimap.

After you fill the ship setup data and press “Start”, the simulation will start running, for more information, visit the Simulation section.

## Automatic Mode

If you prefer using the JSON format of the scenario, that's the place for you.

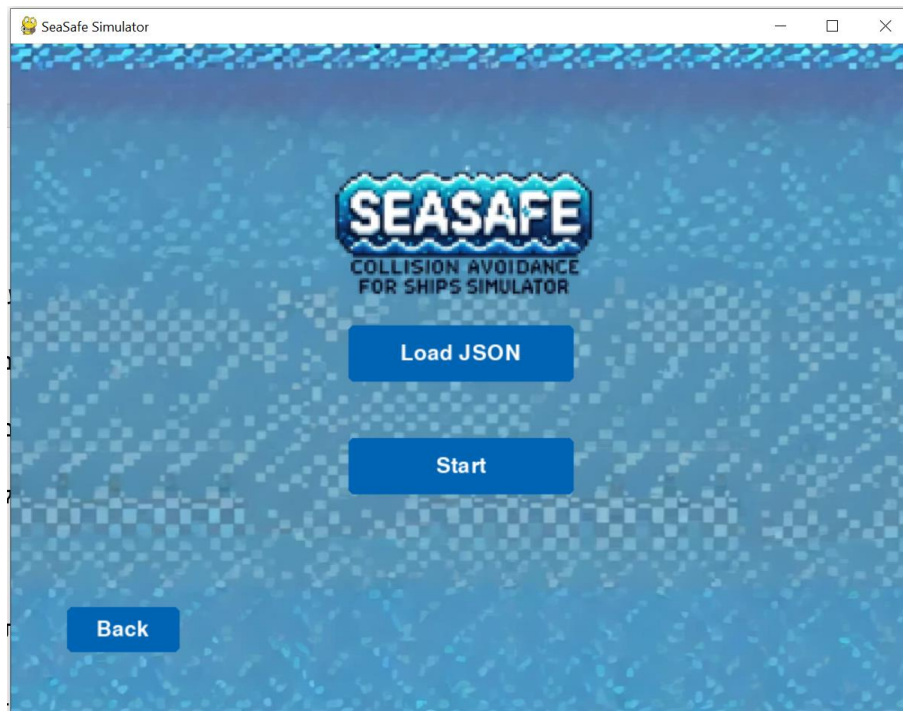


Fig.12. Automatic mode screen

This is the screen you'll see after pressing the "Automatic Mode" button in the main menu, here you have 2 buttons.

- Load JSON – will open your system explorer for you to choose a valid JSON object.
- Start – After you load a valid JSON file, the this will start the simulation.

If you load up a valid JSON file, with a valid format, you'll see "JSON loaded successfully" message. So make sure you get this message so you can start the simulation.

## JSON Format

Here's an example for a JSON format, a 2-ship crossing scenario:

```
{
  "map_size": 8.0,
  "num_ships": 2,
  "safe_distance": 0.2,
  "heading_range": 40,
  "heading_step": 10,
  "time_step": 30,
  "ships": [
    {
      "name": "ShipA",
      "length_m": 400,
      "width_m": 100,
      "speed": 15,
      "start_x": 0.0,
      "start_y": 4.0,
      "dest_x": 8.0,
      "dest_y": 3.0
    },
    {
      "name": "ShipB",
      "length_m": 400,
      "width_m": 100,
      "speed": 20,
      "start_x": 0.0,
      "start_y": 0.0,
      "dest_x": 8.0,
      "dest_y": 8.0
    }
  ]
}
```

## The Simulation

That's the core part of the simulator, the place where you can verify and test the algorithm after you set up the scenario.

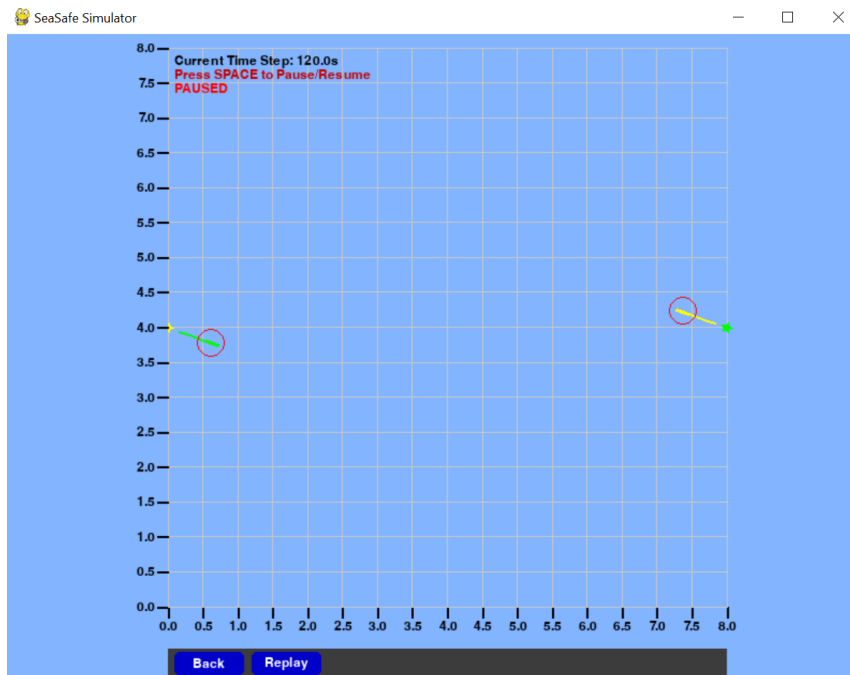


Fig.13. The simulation Running

So here you can see the map of the scenario, which represented by a  $N \times N$  Grid, you can see the coordinates, The yellow and green rectangles are the ships, each ship get a unique color.

In addition, that red circular zone around the ships is the safety zone we discussed earlier, that must be clear of other ships. In the top left corner the current time step is shown and you can see as the simulation forwarding this number increases.

The yellow star represents the destination of the yellow ship, in this scenario both ships have the same destination, so only one ship destination star is presented.

In addition, in the lower part of the screen, you have the panel – you can go back to the previous screen by clicking “Back”, you can even replay the scenario by pressing “Scenario”.

Pressing the space bar in your keyboard will stop the simulator, pressing it again will resume it.

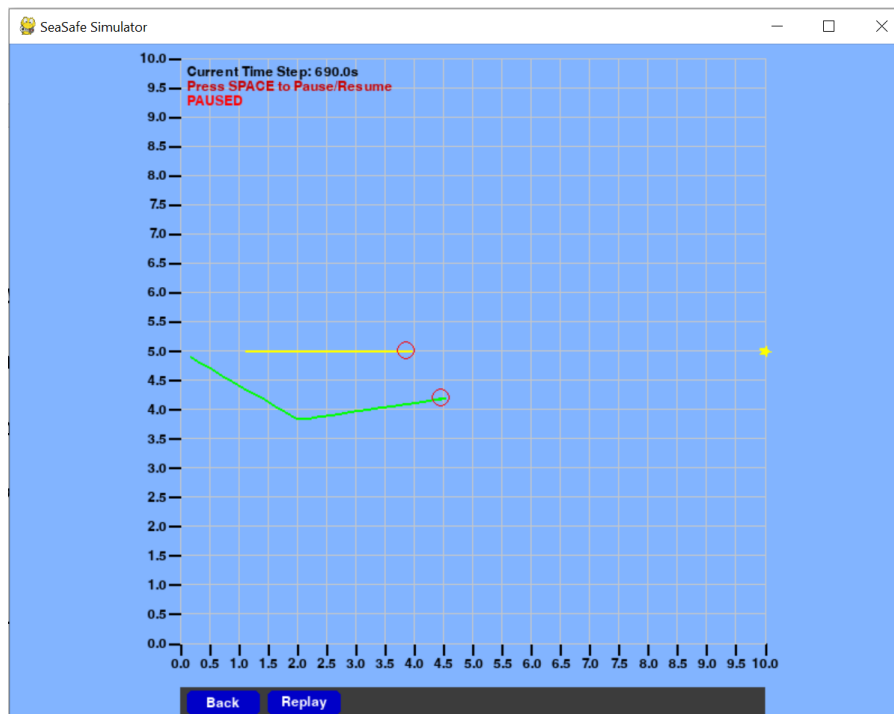


Fig.14. Pausing the simulation

We moved forward the simulator a little bit, you can see that each ship has a unique-colored trail that represents the path the ship has passed.

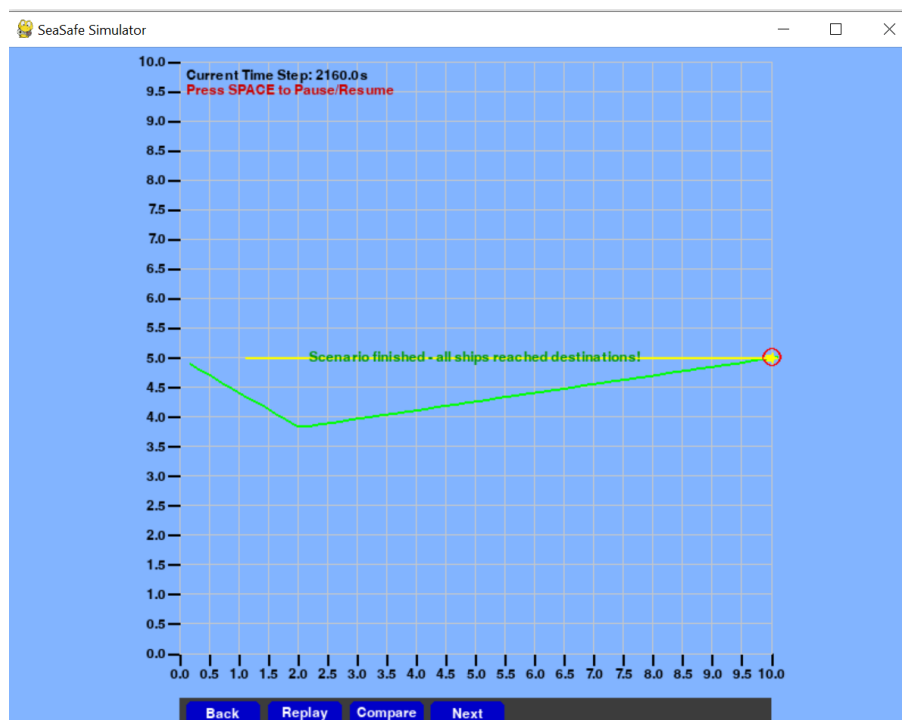


Fig.15. The ships reached their destination.

When all ships reach their destinations, the simulation stops and shows a success message that all ships reached their destinations, In this moment, 2 more buttons appears at the panel:

- Compare button – this draws the original ship paths in dashed lines, so you can see how the algorithm changed the ship original courses to their destinations.
- Next – will move you the the next screen – the Stats screen.

## Stats Screen

After running a scenario, you can view the stat's screen by clicking “next” in the simulation panel.

This screen shows insights about the simulation, how many time did it take each ship to reach the destination? How is it compared to the optimal time (straight course to destination), What encounters were classified? Which collision avoidance event took place?

This is a scrollable screen, so you can use the scroll wheel in your mouse to navigate in it.

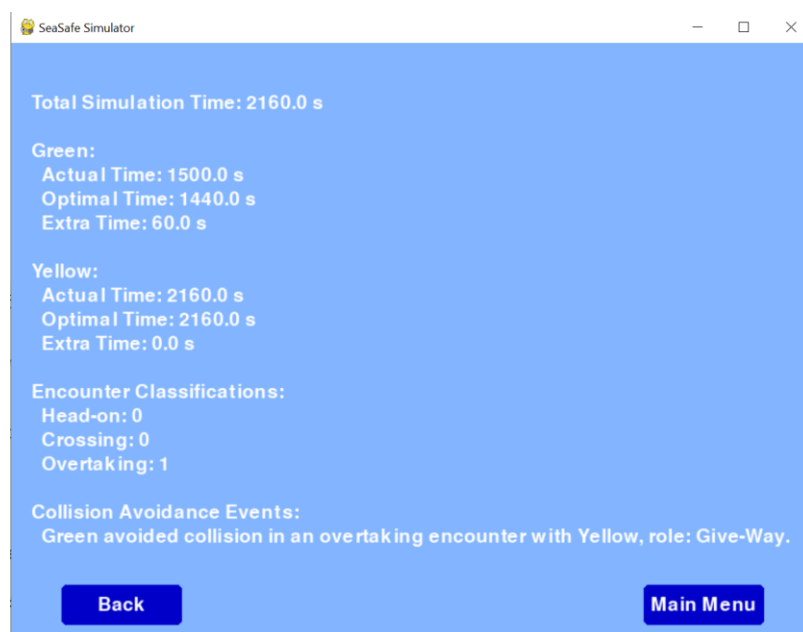


Fig.16. Stats screen

From the stats screen, you can go back to the simulation screen by clicking “Back”, or navigate to the main menu by clicking “Main Menu”.

The User guide showcased a successful scenario of the simulator and showed the flow from the main menu screen to the stats screen, which is the end of the flow.



# Maintenance Guide

The purpose of this Maintenance Guide is to ensure that the deliverables of the SeaSafe project continue to be usable and maintainable after the initial development is complete. This document provides guidelines for applying updates, modifications, and improvements that will help extend the system's lifecycle and support its ongoing use. It is intended for developers and technical maintainers who may be responsible for implementing changes or troubleshooting issues in the future.

## Hardware Requirements

- Processor: Modern multi-core processor (Intel Core i5 or equivalent recommended)
- Memory: 4 GB RAM minimum (8 GB recommended for smoother simulation performance)
- Display: Minimum resolution of 800×600 (higher resolutions are supported)
- Storage: At least 200 MB of free disk space for installation and temporary files

## Software Requirements

The SeaSafe Simulator is developed in Python and uses the following software components:

- Python: Version 3.13.1
- Pygame: Version 2.6.1
- NumPy: Version 1.24.2

Additional software dependencies (such as Tkinter) are included in standard Python distributions. The system is designed to run on Windows, macOS, and most Linux distributions.

## Installation

To run a local development, do the following:

1. git clone [https://github.com/gpariente/seaSafe\\_RuleBased.git](https://github.com/gpariente/seaSafe_RuleBased.git)
2. cd seaSafe\_RuleBased
3. pip install -r requirements.txt
4. python main.py

## Building an executable

1. Run `pip install pyinstaller`
2. Pyinstaller --onefile --windowed main.py

## Code Organization and Structure

The project is modularized into several components:

- main.py: Entry point and state manager.
- simulator.py: Contains the core simulation logic (collision detection, CPA/TCPA calculations, and collision avoidance maneuvers).
- colreg.py: Provides helper functions for calculating relative bearings, CPA/TCPA, and classifying encounters in compliance with COLREG.
- ship.py: Defines the Ship class and its associated behaviors.
- UI and Drawing Modules: Files such as ui\_component.py and draw\_utils.py contain reusable user interface components and drawing routines.
- States Directory: Each state (main menu, automatic mode, manual scenario, manual ship setup, simulation, and statistics) is defined in its own module under the states/ directory.

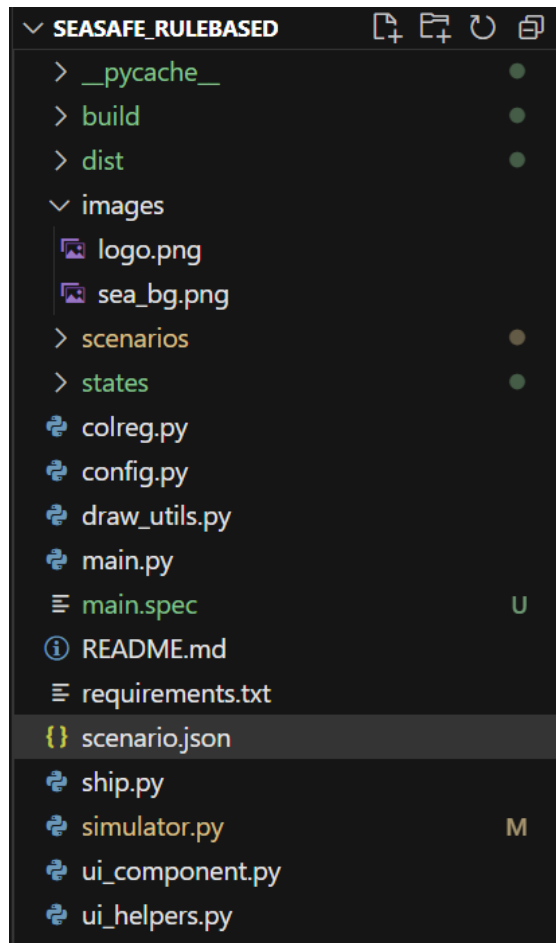


Fig.17. code organization and structure.

## Adding/Modifying Features

- Adding New Features:  
New functionality should be added as separate modules or as extensions of the existing state classes. For example, if you need to implement a new COLREG scenario or a new type of UI interaction, create a new state file under the `states/` directory or add new helper functions to `draw_utils.py`.
- Updating the Collision Avoidance Algorithm:  
If changes are required to the collision avoidance logic (e.g., improving CPA/TCPA calculations or adjusting maneuver strategies), modifications should be made in `simulator.py` and, if needed, in `colreg.py`. Ensure that any changes maintain compliance with COLREG rules and that updated logic is tested in both the simulation and stats screens.
- User Interface Changes:  
Adjustments to the look and feel of the simulator should be done in the UI-specific modules (e.g., `ui_component.py`, `draw_utils.py`, and the files in the `states/` directory). Changes to fonts, colors, or layout should be centralized in `config.py` where possible.

## Conclusion

This Maintenance Guide is intended to provide a clear path for future developers to install, run, and extend the SeaSafe Simulator. By following the guidelines above, maintainers will be able to ensure the longevity and reliability of the system, implement improvements, and keep the simulator compliant with COLREG requirements while adapting to future needs.