# Whales and Whalers: A Multi-agent System

Gabriel Almeida
89446
gabriel.almeida@tecnico.ulisboa.pt

João David
84971
joaodavid@tecnico.ulisboa.pt

Marcelo Mouta
98812
marcelomouta@tecnico.ulisboa.pt

**Figure 1: Sperm whaling in the North Pacific.**

## ABSTRACT

Recent studies suggest that nineteenth-century sperm whales learned defensive behaviours against whalers through social learning.

In this work, we proposed to model a similar problem using a multi-agent system, featuring both whales and whalers as agents. The objective of the whalers is to capture all the whales as efficiently as possible, while the whales try to avoid being captured, for as long as possible. In order to accomplish this, the whales can communicate between themselves, cooperating to evade the ships, while the whalers can either cooperate or compete in order to successfully capture the biggest number of whales.

We started with simpler baseline implementations for the agents, such as a random walk, and then progressively introduced complexity such as deliberative models combined with the necessary reactive behaviour in order to avoid the present obstacles, ultimately leading to the use of hybrid architectures. By adjusting the possible available parameters such as the number of whales in the environment or the rate of fire of the whalers' harpoons, we expect to measure the differences in performance of each architecture, and finding out the consequences of disabling certain features.

## 1 INTRODUCTION

Recently, we've learned that sperm whales were able to adapt to the presence of open-boat whalers [6], through social learning and teaching one another what to do in the presence of human threats.

Inspired by this article, we decided to model a similar problem with a Multi-agent system. The problem consists of two main agents coexisting in a bounded ocean: the whales and the whalers.

### 1.1 Whales

The whales' objective is to survive for as long as possible. They can communicate with each other in the presence of danger, however, due to the nature of the species, their communication capabilities are limited.

### 1.2 Whalers

The whalers move in boats and can periodically fire a harpoon forward. Their objective is to capture all the whales and avoid colliding with obstacles or one another.

While whalers can communicate better with one another and also have a bigger vision range, they suffer from moving slower. In this sense, it is expected that the whalers should achieve better performance by cooperating to be able to trap the whales.

Beyond cooperation, it is also possible to evaluate the whaler agents when their communication is disabled. This model should have some great behaviour differences in the agents' behaviours, as without cooperation the whalers lose the possibility to coordinate to trap the whales, which should lead to poorer results.

## 2 APPROACH

The environment was implemented using the Unity Engine [5]. As mentioned before, it consists of a bounded pool with two islands, and two types of agents, as shown in Figure 2. The number of whales, as well as the number of ships, is variable but at least one of each is necessary for the environment to work. For this project, we expect the most interesting results with at least two ships, enabling their cooperation or competition.

The agents don't have access to the whole environment, and so must rely on their sensory input. The environment is stochastic, dynamic and continuous. The world can be divided into a series of intervals, each corresponding to a single run, independent of each other.

All agents have a proximity sensor, which indicates whether there is another agent near them. The sensor's polling rate can be

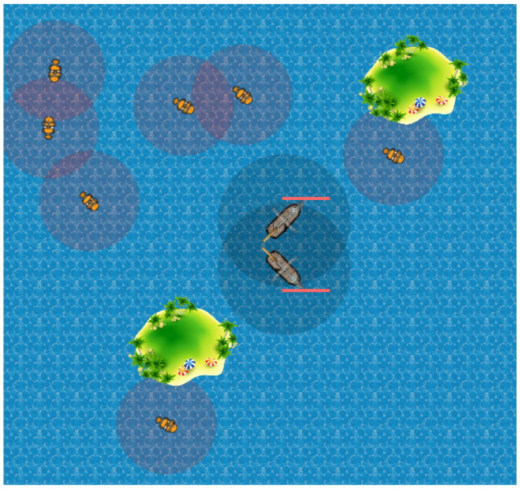


**Figure 2: Simple setting of the multi-agent environment**

changed as well as the sensory input noise, that is, the positions returned are not completely accurate. The sensors can also be blocked by obstacles, such as islands or other agents. The baseline approach assumes a smaller vision range for the whales and a bigger range for the whalers, this might make it possible for the whalers to work together to catch the whale before it notices them.

It is also possible to change the velocity of both agents, however, their movement architectures differ. The fact that the world is bounded has some implications, for example, it is interesting to get the boats to try and trap the whales in the corners, and so the whales should avoid getting stuck on the corners, however, this task proved to be difficult. We'll now see each agent in more detail.

## 2.1 Whales

Whales have limited communication capabilities, and can only communicate their own position through sonars, which should be used to indicate danger.

The whales can rotate and move freely. This, together with the whales' higher speed, provides them with the agility for escaping the whalers as well as their harpoons. The whales need to dodge the harpoons effectively, for example, if they run parallel to their direction, the harpoon will eventually catch the whale, and so it should try and dodge sideways.

With this in mind, the only purpose of the whales is to escape the whalers. They don't have any other objectives besides that, and just need to react to the environment. In this sense, they were modelled as purely reactive agents, although we also decided to try modelling them as learning agents.

## 2.2 Whalers

The whalers possess advanced communication capabilities, enabling them to send to one another anything that might be useful, this will enable them to coordinate with each other when cooperating. It might be interesting to disable the communications and analyse the consequences.

Boats are not as agile as whales, they cannot perform sharp turns or rotate while stationary. This constraint will have some consequences on the agent's decision process, which is also amplified by the fact that the boats have to avoid collisions. Colliding with obstacles or other boats will have a negative impact on the agent's performance.

For capturing the whales, the whalers can use an harpoon. The harpoon has a configurable range and rate of fire. If the rate of fire is lower, the model can take that into account and be more reserved, as by missing the shot the agent will lose more time, which might be important for its performance. If the rate of fire is higher, the agent can shoot at will without any big consequences.

The whalers need a reactive behaviour, for example, for dodging obstacles, but also have to be able to build plan and intentions. They must coordinate with each other and need to constantly reassess their plan, making sure it is still valid. We expect a hybrid agent to perform well for this task. The agent can do multiple tasks at once, such as shooting and avoiding obstacles, but should prioritize the reactive behavior for risk aversion. For an initial approach, we expect a random walk, obstacle-avoiding reactive agent to be a good baseline agent for measuring the performance of more complicated approaches.

# 3 IMPLEMENTATION

We developed the agents iteratively, creating at least a baseline version of each agent and a more complex version with some variations in order to compare their performance. We tested multiple approaches for both agents, either by changing their architecture or by trying different approaches for the same occasions.

## 3.1 Whales

For the whales, we developed three agents:

- Purely reactive agent with simple ship evasion;
- Purely reactive agent with communication and more sophisticated ship evasion;
- Reinforcement learning agent.

We'll now go over the details of each one.

The simplest agent, used as the baseline, was implemented as a very simple purely reactive agent. The agent swims randomly and, when encountering a ship, it would try to flee in the opposite direction of the ship, that is, it would follow the direction given by the vector in Equation 1.

$$position_{whale} - position_{ship} \tag{1}$$

If the whale encountered any obstacle, like an island or the borders of the map, it would give priority to dodging them. By creating this agent, we obtained a good baseline to improve on and to compare results.

This approach had its own issues, for example, giving priority to dodging obstacles instead of whalers proved to be problematic in the case where whalers were also in the vision range. While these problems are known and could be fixed, we believe this agent does a good job for a baseline and decided to only improve those aspects in the advanced version of the whale agent. Besides this, the whales did not make use of any of the advantages of communication.

For the second version of the whale, we still followed a reactive architecture, but this one, even though in a limited amount, tries to solve the problem of having multiple desired outputs coming from the different sensors. We realized that the first version would often get confused when trapped between a wall and a ship, which would cause the agent to ping-pong back and forth between both obstacles.

Inspired by the study [6], we implemented simple communication for the whales. While maybe our communication is not as advanced as the species is able to according to the mentioned article, we still found it to be a worthwhile experience. To implement this behaviour, upon sighting a ship, the whale notifies all other whales of its own position, which we decided was the most a whale could do in this situation. Upon receiving a warning from another whale, a whale could decide whether to react to the notification or not. We settled on a distance-based rule, where a certain whale would decide to swim away from the position, in the same way the first agents did, if the distance is small enough for it to be worth running from.

After communication, most of the time on this agent was spent developing a better way to avoid ships. On the first iteration, we tried having the whale choose the "best" decision for both getting away from the ships and try to avoid the conflict between dodging the wall and distancing from the ship. The first approach was based on the intuiting behind Figure 3. The best direction to follow would be to follow the green vector, equivalent to the distance vector. If that direction was blocked - like shown in the figure - the agent would try the alternatives given by slightly rotating the vector around. This process would be done iteratively until a direction did not find an obstacle obstructing the path.

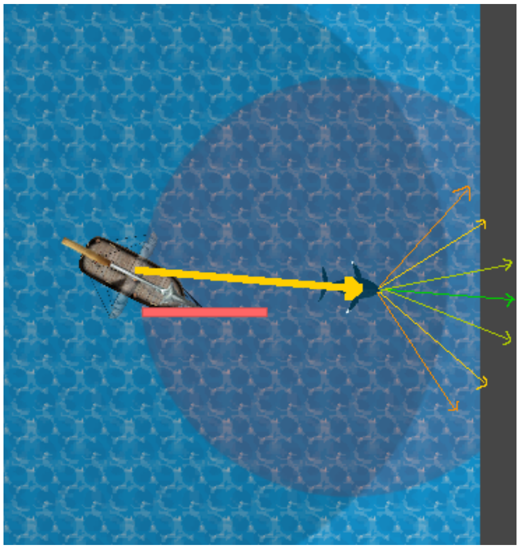


**Figure 3: Best direction for fleeing**

We found this technique to improve on the basic version but still failed because it would often have conflicting reactions happening and, while it could initially choose a good path to follow, it would quickly get confused once the ship approached. To avoid this issue, we resorted to having the agent store the closest ship when deciding to dodge a wall. Even though it might seem like this would create an agent with an internal state, in fact, the current closest ship is given by the sensors at each time-step, and so the only difference is that the agent also passes the information about the closest ship to the layer responsible for avoiding walls. Our solution makes it so that when close to a wall with a ship nearby, the whale will try to follow the wall direction instead of wandering around.

We also added to the agents the ability to dodge the harpoons. When a harpoon enters the whale's vision range, the agent turns to the perpendicular of the harpoon's direction. Since there are two perpendiculars, we choose the one that does not cross paths with the harpoon's trajectory, as illustrated by Figure 4.

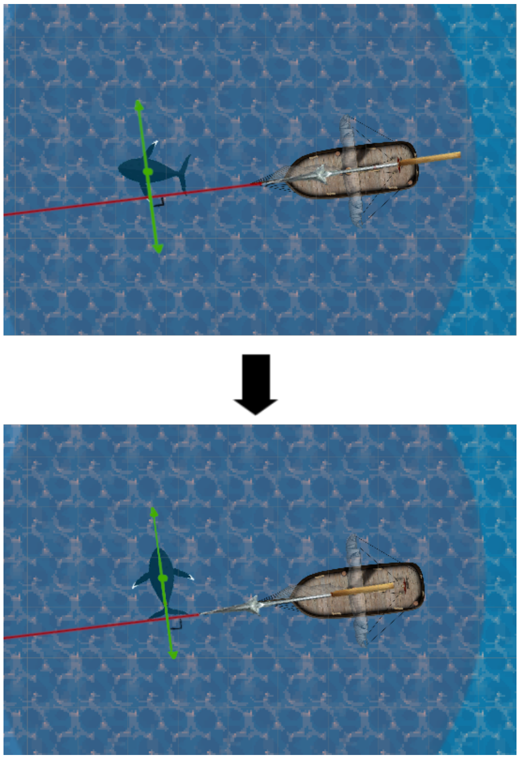


**Figure 4: Choosing the direction for dodging a harpoon**

This proved to be hard due to the speed of the harpoons, which is greater than the speed of the whales and requires fast-acting which, despite our best efforts, does not guarantee the survival of the whale.

Having implemented all of the described behaviour, the agent's architecture priorities are as follows:

(1) If there is an harpoon in sight, dodge it.
(2) If there is a whaler in range, run away from it.
(3) If a wall is close, avoid it.
(4) If there are no observable obstacles, explore.

For the final agent, we resorted to reinforcement learning, because, in fact, whales are capable of learning. For implementing this agent, we made use of Unity's Machine Learning Agents Toolkit (ML Agents) [2]. ML Agents provides a somewhat simple way to setup reinforcement learning for Unity, connecting a learning algorithm implemented over TensorFlow [1] to the game engine, where TensorFlow is responsible for generating the model and Unity handles collecting observations, generating actions and rewards.

We first set up a configuration file with our neural network's hyperparameters. Our training model is set to run with Proximal Policy Optimization [4], a reinforcement learning algorithm. Unfortunately, the version of ML Agents that we used did not support more advanced techniques, such as a variable number of observations, which are available on newer versions, which makes modelling hard problems a harder challenge, which would require a technique such as zero-padding. Instead, we decided to follow an approach more similar to our other agents, and just take into account the closest ship instead of every ship in the vision range. Since the observations have to be numeric, an additional field was added to detect whether there was a ship present or not, in the hopes for the agent to learn that when that flag was set to false, the given position vector was irrelevant. The same technique was applied for detecting the closest harpoon. The agent is aware of its own orientation, as well as the position of the closest ship in relation to its own axes. The agent is also aware of the closest walls in a hot-encoded fashion, with a bit for each direction, left, right and centre. We also normalized the input features, which usually results in faster convergence for neural networks. Our agent also makes use of ML Agents "stacked vectors" for the input features, which work similarly to the technique applied in [3], stacking multiple frames of observations into the most current one, in order to use more context to make a decision, such as inferring the direction of the ship from its movement.

For the actuators, the agent has two continuous output variables, one for its forward velocity and another one for rotating. We tried different rewards, but the important idea is that:

- The whale was rewarded slightly each decision epoch for surviving;
- The whale was penalized slightly for touching walls;
- The whale was highly penalized for being hit with a harpoon, which would end the episode.

We also tried variations by giving the whale a slightly negative reward if a ship was in its vision range, hopefully incentivizing the whale to stay away from ships. We tried with both normalized rewards and more arbitrary rewards to see if it helped in the agent's convergence, but actually seemed to not do much difference. Our learning agent that performed the best used a reward of 1 for each time step survived, a negative reward of -5 for touching walls and -500 for getting hit with a harpoon, the cummulative reward is in Figure 5. This agent, with the final decided parameters once we tested multiple alternatives, was trained over 3.5 hours at 15 times the normal execution speed.

The way that the unity ML Agents toolkit is set up also made it hard to implement communication, which was not possible with the current architectural choices.

## 3.2 Whalers

For the whalers, we developed two agents:

- Purely reactive agent;

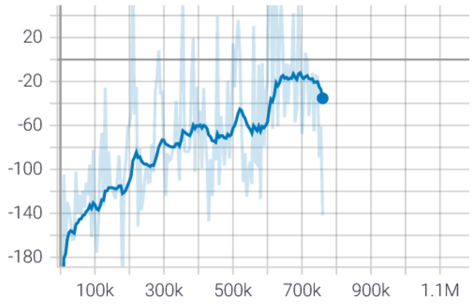


**Figure 5: Cummulative reward for the chosen agent**

- Hybrid agent with reactive component, BDI-based component, communication and coordination.

The first agent consists of a simple reactive agent. When whales are in the vision range, it follows the closest whale and, if the angle between the direction of the ship and the distance vector is smaller than a given threshold, it shoots the harpoon.

This first implementation is also capable of avoiding walls and other ships, even though the latter are treated as if they also were walls, which we worked on on the second version of the agent, to see whether it made a significant difference.

For the second version, the first noticeable difference is in the agent's architecture, consisting of a vertically layered hybrid agent. The reactive layer, naturally, is given the most priority and the rest of the interactions are handled by a deliberative layer. This version is also able to fire harpoons while dodging obstacles, which was not supported in the first version, which only outputted one action.

The deliberative approach is based on the BDI architecture but somewhat adapted to our own use case. The ship has two possible intentions, $explore$ and $follow$, and goals are generated in coordinates, which decide where the ship should move to. Besides that, whales are also identified by a string, which is used to identify the whale being followed and to aid with coordination. This allows the whalers to keep following the whales even if they're not in the vision range anymore because the intention is kept. The base algorithm is as stated in Algorithm 1.

Besides that main structure, we also added some other options, such as allowing the whaler to change his $follow$ intention if another whale was in a favourable position, which should be closer than the current one and at a narrow enough angle to avoid big turns. This agent also shoots depending on the distance, instead of just checking if the range and distance are small enough. Instead, it uses a given threshold (Equation 2) and shoots if the current angle is smaller than the given threshold.

$$angleTreshold \propto \frac{1}{distance} \tag{2}$$

This means that the $angleTreshold$ increases as the distance decreases, allow the whaler to shoot when it's closer to the whales, which would sometimes not happen because when both are close together any movement would greatly impact the angle between both agents.

**Algorithm 1:** Base Deliberative Algorithm

```
Data: goal
if intention is follow then
    if whale still in sight then
        goal = whale.position;
    else if reached goal then
        // set to explore but keep moving forward
        intention = explore;
        goal = forward;
    end
else
    if reached goal then
        goal = random();
    end
end
follow(goal);
```

Another main component of our hybrid agent is communication. For communication, the agents broadcast to other agents their current position, identifier and current intention. Sharing the current intention is important for coordination, as we'll talk about later. Besides those attributes, whalers also communicate when they successfully hunt a whale. This is useful because it allows the whalers that had the intention to follow a certain whale to give up once they know they've been hunted, instead of having to find out by themselves.

With the communication set in place, the agents can coordinate with the knowledge of the other agents and a set of coordination rules. We approached coordination based on roles, deciding on which ships should follow which whales. The set of rules are the following:

- Two whalers can coordinate to hunt a whale;
- These two whalers should be chosen according to the highest capturing potential.

The decision to limit two whalers to each whale was rather arbitrary, but we found it to be a good balance for the environments that we tested in. It proved as a good proof of concept, and it could also be easily changed to more flexible configurations, by allowing to change the number of whalers assigned to each whale.

As mentioned, the process for choosing which agents to target a whale was based on roles. Each agent, through communication, knows the intention of other agents and their current positions. When a whale is sighted, that information is communicated through other whalers which then, individually, compute the score of each whaler to the sighted whale. The agent that has the higher score will change its intention to $follow$ with the given whale identifier and broadcasted position. We opted for the version that does not require communication due to its simplicity and the fact that it does not require any other message sharing besides the normal belief communication, and also because we only had a small number of agents calculating the scores.

The score is calculated according to Equation 3.

$$score \propto \frac{angle(vec_j, vec_i)}{distance_{whaler_j, whale}} \tag{3}$$

Where:

- $whaler_i$: is the whaler that sighted the whale.
- $whaler_j$: is the whaler for which we're computing the score.
- $vec_x, x \in \{i, j\}$: the distance vector between the whale and the corresponding $whaler_x$.

This formula gives a higher score the bigger the angle is between the two vectors, where the angle always lies on the interval $[0, 180]$. Intuitively, it means that whalers that are in opposite positions from the whale are given higher priority because it increases the probability of trapping the whale between the two ships.

While training the whale agent, since it started by executing a random behaviour, we realized that the ship would often go in circles around the whale. This is not a problem with the other agents that we designed since they swim away from the ship, but we realized we had to adapt the ship's throttle based on the distance to the target, coming up with Formula 4, where $a$ and $b$ are constants adjusted to our problem. This formula gets closer to 0 as the ship gets closer to the whale and approximates 1 as it moves away, which corresponds to full throttle.

$$throttle = max\left(0, 1 - \frac{1}{e^{a \cdot distance - b}}\right) \tag{4}$$

Finally, the last part that we developed on this version of the whaler was the component to avoid other ships. Any collision between two ships will result in both losing health points, so they both must work towards dodging one another. Since there are many ways for both ships to dodge one another, we resorted to social conventions to decide which action to choose.

The original implementation was for the ship to turn right if they faced one another and to turn in opposite directions if they are side to side. The first convention worked well as expected, as for the second one, the size and turn radius of the ships were not taken into account, so when they were side by side and turned in opposite directions, they ended up hitting one another. This solution also struggled when a ship was trapped between a wall and another ship. To address those issues, we chose a set of parameters that acted the best on most tested occasions that replicated those situations. The set of final social conventions, also illustrated in Figure 6, are the following:

- When both ships face one another, they both turn right and slow down to avoid colliding.
- If a ship has another one at its right and no other ship or wall around him, it turns left.
- If a ship has another one at its left and no other ship or wall around him, it turns right slowly.
- If an agent is between a ship and a wall, it slows down a goes forward.

After implementing the ship avoidance mechanism, the final agent's reactive behaviour has the following priorities:

(1) If there is any whale in shooting range, shoot it (can be executed simultaneously with others);
(2) If there is any whaler nearby, avoid it;
(3) If a wall is near, avoid it.

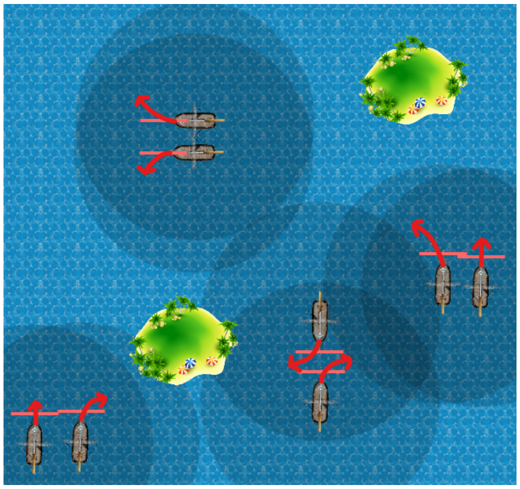


**Figure 6: Ship avoidance social conventions**

## 4 EMPIRICAL EVALUATION

The main objective is for the whalers to capture all the whales as effectively as possible.

In terms of the whalers' performance, we want to minimize the time taken to capture all the whales, which we considered as one of our most important metrics. Each whaler should try to maximize the number of whales captured, which is also an interesting metric to compare.

Each episode can end in one of the two ways:

- The whales are all captured;
- All whalers are disabled.

It is interesting to keep track of how each episode ended because it gives us an insight into the performance of the agents' ability, we expect most episodes to finish by catching all the whales. For this effect, we also measure the whalers' performance on collision avoidance by giving them a health bar, which loses points on collision.

The whales' performance can be measured by the same aforementioned metrics. The whales need to maximize the time they last, so, the elapsed time can be equally used.

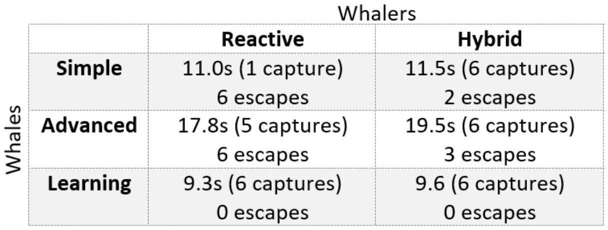
For testing the agents in different scenarios, we built 5 distinct scenes for measuring performance:

- Default scene: scene to try random runs;
- Side/Corner Scene: two scenes where a whale is trapped next to a wall/corner, respectively;
- Ship Avoidance Scene: a scene for testing the different ship avoidance methods.

Besides those, we also have a scene for showing the effect of coordination and a scene for training the whales and for testing the harpoon avoidance.

### 4.1 Side Scene

We used the side scene for evaluating the performance of individual entities. We run every combination of agents until the whale had been captured 6 times or had escaped 6 times (maximum of 11 runs). We use as a metric the average time taken for the capturing runs and the number of runs that ended by having the whale escape. A whale was considered to have successfully escaped if it survived for more than 45 seconds in the specified environment. The results are in Figure 7, where the seconds correspond to the average of the runs.



| | | Whalers | |
|---|---|---|---|
| | | **Reactive** | **Hybrid** |
| Whales | **Simple** | 11.0s (1 capture)<br>6 escapes | 11.5s (6 captures)<br>2 escapes |
| | **Advanced** | 17.8s (5 captures)<br>6 escapes | 19.5s (6 captures)<br>3 escapes |
| | **Learning** | 9.3s (6 captures)<br>0 escapes | 9.6 (6 captures)<br>0 escapes |

**Figure 7: Results for Side Scene**

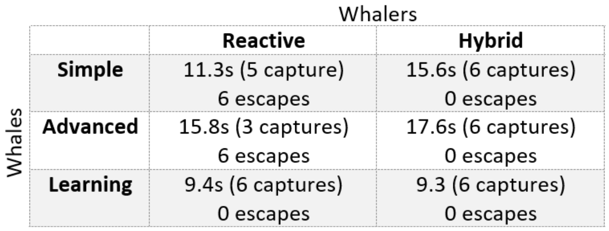
We noticed that in the executions, a lot of the times that the whales managed to escape were achieved by having the whale go into a corner. The ship's reactive behaviour would be triggered and, if unlucky, it would not allow the whaler to shoot as it turned away from the corner. This was quite unexpected, as we initially believed that the whales would possibly struggle once trapped in a corned, but it actually proved to be more of a weakness for the ships instead. This problem is diminished in the hybrid ships, which are also capable of shooting while avoiding walls, but it is still present nonetheless.

We see great improvements from the reactive ship to the hybrid ship. The reactive ship did not manage to achieve the 6 captures, while the hybrid always succeeded. For the hybrid ship, the advanced whales achieved double the average surviving time as the simple whales, which is a positive indicator of the advanced agent's success. While the same happens with the reactive whaler, it's hard to make conclusions for the average time, but the fact that the simple agent was only captured one time while the advanced whale was captured 5 times was surprising and contrasts with the performance achieved when using the hybrid ship.

The hybrid architecture mostly succeeded by having intentions and internals state, which allowed the ship to keep pursuing the whales even after losing sight of them, which proved crucial in capturing the whales.

We also realized that often, the ship would slow down when following a whale because it was running parallel to a wall, which would trigger behaviour for slowing down. After removing that constraint, we obtained the results in Figure 8. We can see that the average capture time reduced for most runs, except for the run with the hybrid ship, but it seemed more like a coincidence than actual deterioration in the agent's performance. Furthermore, in no runs of the hybrid agent did the whale manage to escape.

Finally, we can see that these changes did not affect the learning agent, because it overall performed really poorly, even after trying different hyperparameters and observation sets. This does not mean that the learning agents are a poor fit for this problem, because we believe that with a good set up they would work just as well, if not better, than the other architectures.



| Whales / Whalers | Reactive | Hybrid |
|---|---|---|
| **Simple** | 11.3s (5 capture)<br>6 escapes | 15.6s (6 captures)<br>0 escapes |
| **Advanced** | 15.8s (3 captures)<br>6 escapes | 17.6s (6 captures)<br>0 escapes |
| **Learning** | 9.4s (6 captures)<br>0 escapes | 9.3 (6 captures)<br>0 escapes |

**Figure 8: Results for Side Scene after fix**

## 4.2 Ship Avoidance

We test the ship avoidance mechanics individually, by running the corresponding scene for 45 seconds and taking the average of boat collisions for 10 runs for each agent, take into account, that each boat-on-boat collision counts as 2 collisions because both ships lose health points in that case. The scene consists of 8 ships, having some ships starting facing others and different setups which were used to test the social conventions mechanisms. The results are in Figure 9.

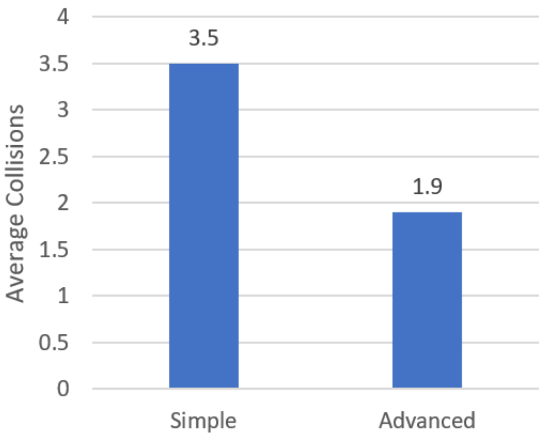
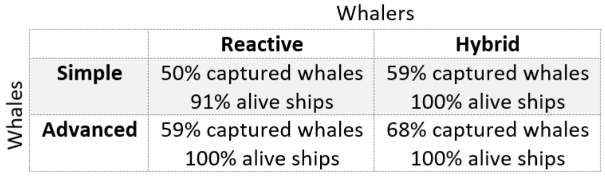


**Figure 9: Average number of collisions in 10 runs**

On average, the advanced ship with social conventions managed to do better than the ship that treated walls the same as other ships. We observed that, in the given scenario, the simple version would always collide in the situation that led the ships to an island, which greatly increased the average. We also observed that the social conventions would result better in more cluttered situations, but would still struggle in situations where there are too many sensory inputs, especially when encountering walls and other ships at the same time.

## 4.3 Default Scene

Finally, we measure the overall performance in the default scene. The default scene differs from the others in the fact that it is used for normal runs instead of pre-baked scenarios. While this allows us to see the agent in action in more realistic scenarios, it also makes it more difficult to extract relevant metrics due to the stochastic nature of the environment. Taking this into account, we decided to instead run every combination of agents for 8 runs, each run lasting 90 seconds. The tests were run with the default parameters, consisting of 7 whales and 3 whalers. Once a run is over, we took into account the percentage of captured whales and surviving ships, which are shown in Figure 10.



| Whales / Whalers | Reactive | Hybrid |
|---|---|---|
| **Simple** | 50% captured whales<br>91% alive ships | 59% captured whales<br>100% alive ships |
| **Advanced** | 59% captured whales<br>100% alive ships | 68% captured whales<br>100% alive ships |

**Figure 10: Average capture rate in default scene**

From these results, we observe that, on average, the hybrid agents are able to capture around 9% more whales than their reactive counterpart. In only two runs was a ship destroyed, and it only happened with the simple version of the whale against the reactive whaler. The advanced whale also showed poorer results in relation to the simple whale, also with an increase of 9 in the percentage of captured whales. We noticed that this was often because the reactive whale would get stuck on corners while the advanced ship would usually swim along with them, which actually made it easier for the whaler to capture it, because, as we've mentioned, our agents struggle with corners.

We were not able to test the default scene for the learning agent, since Unity makes it hard to deploy trained agents in normal scenarios, like we were able to do with our other architectures. However, we do not expect it to perform well, as we could observe from the overall performance during the end of its training process, when the agent's curiosity was close to 0, which means that the agent was performing mostly exploitation actions.

# 5 FUTURE WORK

We believe our work provides a solid base for this problem, but we acknowledge that this it can be taken even further and agents can be improved. In this section, we'll give brief ideas on what could be explored that we considered outside of the expected scope for this project.

For the whales, it could be interesting to try creating a more "agressive" agent that would try to get ships to collide with one another. Due to the nature of the agent, this task is not trivial. Also, even though the learning agents were added to the project after the initial proposal, we would like to furtherly explore this architecture in hopes to achieve better results with it.

For the whalers, first, we believe that trying to optimize the agent to try to prefer capturing multiple whales at once with a single harpoon instead of just one could be an interesting experience, even though it is probably not worthwhile for simulating a more realistic scenario. It could also be interesting to get the whalers to stay at the perimeter of the whales' vision range until enough agents were ready to hunt it down, diminishing the possibility for whales running away.

Lastly, we discussed a more ambitious idea for coordination, which would go beyond simply choosing the best fit for coordinating and would instead combine the previously mentioned perimeter idea. Each whale could be assigned $n$ “pseudo-goals", where $n$ is the number of coordinating agents. Those goals would be the positions to where the ships should move to have a higher possibility of capturing the whale. Figure 11 illustrates those positions for $n = 3$. A more advanced coordination technique could instead use those positions as goals, assigning the closest ship to each ship to a single position based on distances and, once all ships were ready, they would then hunt down the whale, leaving little to no chance for escaping.

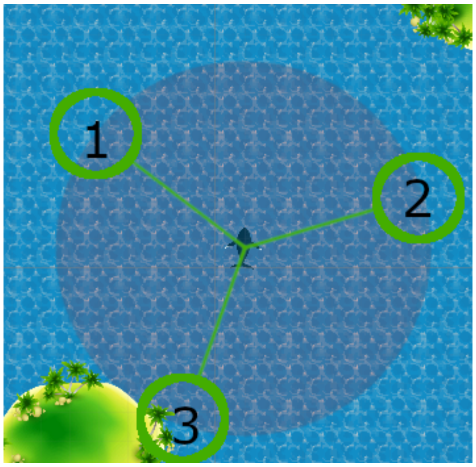


**Figure 11: Pseudo-goals for advanced coordination**

## REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, et al. 2020. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627* (2020). https://github.com/Unity-Technologies/ml-agents/

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[5] Unity Technologies. 2020. *Unity*. https://unity3d.com/unity/whats-new/2020.3.0

[6] Hal Whitehead, Tim D Smith, and Luke Rendell. 2021. Adaptation of sperm whales to open-boat whalers: rapid social learning on a large scale? *Biology Letters* 17 (March 2021). Issue 3. https://doi.org/10.1098/rsbl.2021.0030