2-Agent Cooperative Snake Game

David Antunes
MEIC
Instituto Superior Técnico
Lisbon Portugal
david.f.l.antunes@tecnico.ulisboa.pt

Hugo Ribeiro
MEIC
Instituto Superior Técnico
Lisbon Portugal
hugofribeiro@tecnico.ulisboa.pt

Mariana Serrão
MEIC
Instituto Superior Técnico
Lisbon Portugal
mariana.serrao@tecnico.ulisboa.pt

ABSTRACT

The following paper follows the development and testing of a multi-agent system based on the popular action game Snake. The game was adapted to have two intelligent agents control two separate snakes, each with its own respective cherry to pick up on the map, while avoiding collision with either snake's body. The goal of the agents is to collectively pick up as many points (cherries) as possible while also "surviving" for as long as they can. To study this situation, several different agent types and strategies were employed and empirical results, in the form of total points obtained, number of steps survived as well as cause of loss, were widely collected for each of them. The analysis and comparison of these results, allowed us to draw conclusions about the adequacy and efficiency of these architectures in the face of this problem/environment. Our goal was to develop a strategy where the agents communicate the path they intend to take, are able to replan if their trajectories intersect and achieve a stable cooperative relation where both reach their individual goals and avoid collisions. We expect collision avoidance to be a mechanism that allows agents to survive more steps than those who don't and coordination to be essential to achieving higher scores and point gathering efficiency.

KEYWORDS

Multi-Agent Systems, Cooperation Games, Snake Game, Advanced Decision Making

INTRODUCTION

We all remember Snake, the game that captivated us during our childhood and left a mark on us. For this project we wanted to elevate this game and take it to the next level. We made the decision to transform the single-player Snake into a multiplayer game. This change not only amplifies the game's difficulty but also adds a twist, as now the agents must communicate and coordinate with each other to avoid collisions and achieve the highest possible score. While this may seem like a futile game, coordination between two agents is a wide and very important field of study that provides insights on rational behavior in societies. This can, for example, be reflected on autonomous car navigation systems, where we have a group of agents (cars) traversing a grid like environment (roads) with the goal of achieving their destination in the most efficient way possible, but of course making sure to prioritize the safety of passengers by avoiding collisions with other agents as well as obstacles in the environment.

During our research for this project we came across 2 articles that caught our attention.The first article explored the use of deep reinforcement learning in playing Snake [1], where the agent continually learns through its interactions with the environment. The author defined the actions and state space, as well as the rewards, for the agent to learn from. After testing, the author concluded that the performance of the agent is influenced by how the state space and rewards are defined. Furthermore, the results demonstrated that the agent always achieved a higher score than a random agent. Although there are similarities, there are also notable differences in our project. In our case, we worked with 2 agents that should not only avoid the walls and themselves but also avoid each other. Another huge difference is that our agents have full observability of the entire environment. These distinctions introduce unique challenges and opportunities for our project, setting it apart from this article.

The second article is about Multi-agent cooperative pathfinding [2] where they propose a new algorithm to avoid collisions.Their algorithm consist in "building a tree that is rooted at the start state and spans towards randomly sampled states from some given state space", after reaching the goal region, the algorithm can follow its edges backwards to obtain the first feasible path. Overcoming this particular challenge was one of our main objectives, as we aimed to enable our agents to achieve the highest score possible. We explored various approaches to tackle this obstacle in order to find the most effective one.

Now to introduce you to our idea and the problems that arise with it. In our design, we have 2 snakes, each with their own unique "cherry" that will be randomly placed on the map. The ultimate objective remains the same as any

snake game: to achieve the highest possible score (total number of cherries captured by both agents). In order to achieve the best performance possible, we believe that 3 things need to be ensured: that the agents can communicate with each other so they can find a path that doesn't cause a collision, that both agents are aware of their size so they don't collide with themselves and of course that our snakes avoid headbutting into the map's borders.

Note that, not all of our agents' architectures have embedded a strategy with mechanisms to meet all of these criteria, which allows us to draw conclusions for behaviors in isolation as well as combination with others..
Thus, our final goal was to test and compare the different agent types, alongside decision making and coordination strategies, to determine which architecture is more capable of facing the problem at hand and achieving the best results.

This project's codebase can be found on a public github repository [3] with instructions on how to install and run.

APPROACH

Environment

An environment is everything in the world that is external to the agent and changes with the agent's actions. It is the agent's source of information both in terms of sensory input and rewards gained.

Since the focus of this project was to develop several different agent architectures and test them against each other, we started off by picking an already made environment that emulates the problem at hand. This consists of a python based 2-player Snake game which can be found in a public repository [4] and was developed by Divya Kustagi [5]. All credits for making this system go to its author(s) as we only intend to change the game logic very slightly:

- Points from both snakes will be added up, in order to make the game cooperative instead of adversarial
- Two cherries will be put on the map instead of only one
- The size of the map and length of the snakes was reduced to avoid unnecessarily complex situations
- The speed of the snakes was increased since the agents input new actions faster than human players did
- A step limit (_) was added to ensure runs did not last infinitely as agents could simply move in circles around themselves and not grab any points

Thus the environment consists of an episodic game (when any of the agents collides with a wall or a snake the environment resets) with a 30x30 grid shaped map with two cherries randomly appearing in any of the cells and the two snakes starting always in the same places. The two agents' task was to explore the map and pass over the cherries' locations to collect them, which makes them instantly reappear in a new cell. We can therefore define the environment as static (since it only changes through the actions of the agents), discrete - with a limited number of actions and possible states that are fully determined by the previous state and new agent actions (deterministic). At each time step, the agent "sees" the environment, obtaining information about both snakes' positions (of the whole body) as well as the cherries' positions and rewards for their actions (only used in Reinforcement Learning) . Alongside having full knowledge of the map's boundaries, we can say this is an accessible environment. All positions in the environment are represented as a list of two coordinates [x, y] which range from 0 to 290 (since each cell of the grid occupies 10 pixels of the canvas).

Agents

The agents that will traverse this environment will be taking actions in the direction of their goal making them proactive.They are all reactive, with the ability to quickly observe the environment and make decisions based on what is known. The agent's possible actions are to move one cell to the Left, Up, Right or Down. The different decision making strategies are autonomous processes that secure the agent's mobility in the map, usually towards the "cherry" and away from obstacles.

We will now present the several types of agents that were deployed in the environment, always in teams of two with equal behavior/architecture.

Random Agents

These agents randomly choose any of the four actions that they have available to them.

We expected to see absolutely terrible results from these agents since given the problem at hand, there is a decent chance they will move in opposite directions in quick succession and thus collide with themselves in the first steps of the run.

Fully Greedy Agents

These agents go straight to the position of their respective fruit without regard to any obstacles. Their decision making process is as follows: They randomly decide to move horizontally or vertically in the map. They then move in the direction that brings them close to their goal. When they decide to move horizontally (or vertically) but are already on the same column (or row, respectively) they move in the other axis.

Their performance was expected to be completely dependent on the positions fruits appear at, since if they have two cross paths to reach the cherries they will likely collide, and might even collide with themselves if the fruit is in the opposite direction they were going previously.

Partially Greedy Agents

Partially greedy agents are, as the name implies, a more rational version of the previous agents. They have been equipped with some (flawed) collision avoidance mechanisms with themselves, the other snake and the walls. The idea behind their behavior is to avoid moving horizontally if any snake's body is adjacent in this axis (same for vertically)

We thought these agents would perform better than the previous type, since they should be able to avoid each other and themselves when moving towards the fruit. The improvement should be patent in both number of steps survived and score obtained.

Agents that follow Social Conventions

This type of agent orders all possible actions at each time step, prioritizing the actions that both increase the distance between the 2 snakes and lead to the fruit. Once they evaluate the available actions, the agents choose the first feasible action. To determine the order of actions, the agent initially assesses the direction in which the snake needs to move to capture the food. It then selects the action that maximizes the distance between the two snakes.

We anticipated that these agents would outperform their predecessors due to their behavior, which generates favorable scenarios that minimize the risk of collisions.

Intention Communication Agents

These agents plan the path they intend to take and share it so others can avoid collisions. The first snake starts by calculating the shortest path from its head's location to the fruit while considering its body and the other snake as obstacles. This is done with an adaptation of the BFS algorithm [6]. The intended path is then communicated to the other snake which calculates its route considering the same obstacles plus the intended path of the first snake. Everytime a snake picks up a fruit they calculate a new intended path. If no route is deemed possible the snake performs a random movement in a direction other than where it came from. It's always the first snake that sends the other its intended path and plays a higher priority role, when this happens the second snake always recalculates its route. Information is exchanged in the form of a list of successive cells of the grid.

We believed these agents could achieve incredible results in terms of steps and score achieved. Score efficiency might not be the best since the second snake takes unnecessarily long paths sometimes to avoid crossing the path of the other snake even if it wouldn't cause a collision.

Reinforcement Learning

These agents learn to make decisions and take actions in an environment to maximize a reward signal. The agent learns by interacting with the environment over multiple episodes, it improves its decision making capabilities by adjusting its Q-Values based on the rewards gained during training.

We started by reducing the size of the grid and the snake to minimize the state space. Secondly, we designed a reward system to incentivize desirable behaviors. When the snake successfully captures its food, we assign a positive reward of 5. However, any collision results in a penalty of -10. Additionally, we introduced a small penalty of -0.5 for each step taken by the snake.

After testing this implementation, we observed that the agent failed to learn and consistently took only an average of 1 step. We hypothesized that this behavior occurred due to the relatively low reward assigned for obtaining the fruit, so the agent seemed to choose an instant loss to maximize its reward instead. To address this problem, we made changes to our reward system. We increased the positive reward for capturing the fruit to 10, assigned a negative reward of -50 from collisions, and maintained the reward of -0.5 for each step.

After implementing the previously mentioned changes, we encountered persistent issues, leading us to conclude that the problem likely stemmed from the extensive state space. This resulted in numerous states that remained unvisited, leading to the Q-values remaining 0 for those states. As a consequence of that, the agent struggled to learn effectively and make informed decisions.

EMPIRICAL EVALUATION

To compare the several agent architectures that we implemented we first had to create a fixed set of metrics to measure performance in terms of the desired behavior and objective we have for the system. The most effective way to see which architecture is better at any game is to compare the total amount of points obtained, or in this case, the total amount of cherries collected.

It's also important to see how agents react when they are unable to come up with a "winning" situation. For this purpose we wanted to look at how many time steps the agents were able to survive no matter how many cherries they grabbed. Of course, we also cared about movement efficiency so it was very relevant to evaluate how many steps on average it takes for the agents to pick up a cherry. This derived measure was computed from the two last metrics mentioned.

On a different note, we were able to analyze the cause of death/loss for each of the agents' runs. Since we expected rational agents (for example) to die very little from collisions with walls and social agents to be able to avoid collision with the snake more than other architectures. The end states were therefore recorded and classified as "collision with wall", "collision with the other snake", "collision with self" and "step limit reached"

All of these 4 metrics were captured for a sufficiently large sample of runs in the environment, 40 episodes, and then averaged out to obtain the expected result for each agent type. The results are shown in the charts below (Figures 1-4).

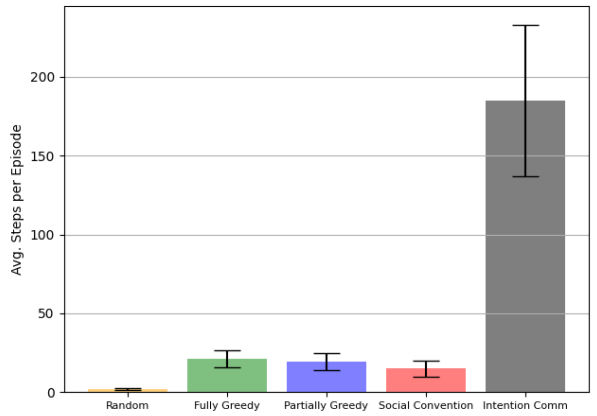

Figure 1: Average steps by agent type

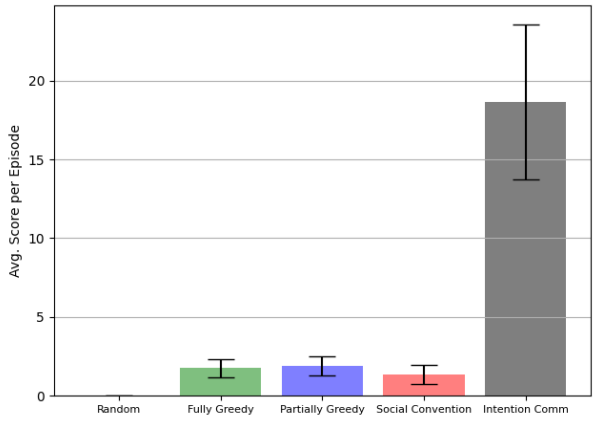

Figure 2: Average score by agent type

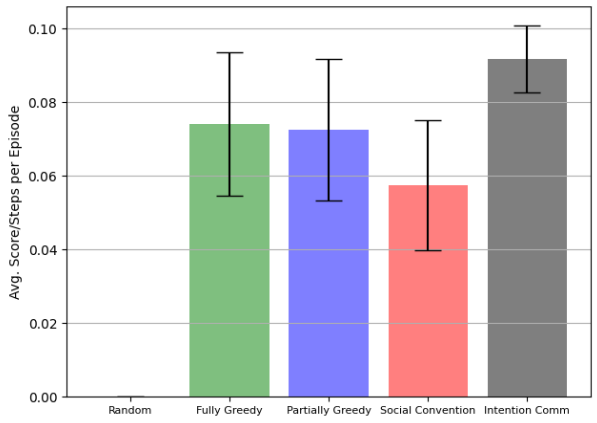

Figure 3: Average score efficiency by agent type

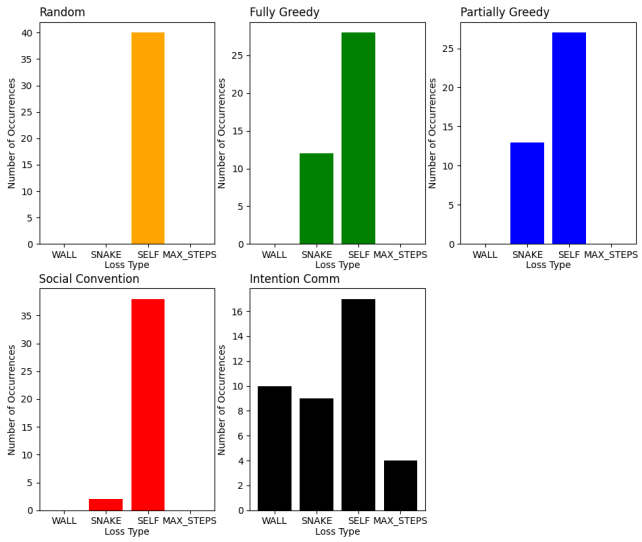

Figure 4: Causes of loss by agent type

Result Analysis

The results gathered for each agent type show the effects of the proposed strategies. Each metric can be analyzed to provide insight into their strengths, weaknesses, efficiency and if they are suited to the proposed scenario (cooperative snake game).

Random Agents

As expected, the Random Agents exhibited extremely low scores and efficiency averages, both of which were at zero. The main reason behind this was their early loss, as evident from their average number of steps, which was just one.

Their random choice of direction meant that at each step, they had at least a 25% probability of colliding with themselves, by making a 180º turn. Consequently, all of the agent's deaths occurred as a result of self-collisions, early in the game.

Fully Greedy Agents

These agents demonstrated a significant improvement in performance compared to the Random Agents. Their movement decisions were entirely dependent on the position of the cherries, which meant their deaths were solely attributed to obstacles in the path between their head and the fruit. This of course meant that none of the losses were a result of colliding with walls.

Since their step average was relatively low, around 22, the step limit was never reached. Consequently, the deaths of the agents were caused by collisions with the other snake or by self-collision, with the latter being the predominant cause.

The agents' early losses prevented high scores, achieving an average of approximately 2. Despite that, their sole focus on collecting the fruit, resulted in the second highest score efficiency.

Partially Greedy Agents

Despite our initial belief that Partially Greedy Agents would have better performance than Fully Greedy ones, all metrics were practically the same.

While they had mechanisms to avoid collision, their greedy component still led them to early losses due to collisions in pursuit of the cherry. Since their options of movements were restricted to those in the direction of the fruit, they were usually faced, early in the game, with scenarios where no decision was possible without collision.

Still, it's interesting to notice that the two last different strategies led to a small change in what the agent was good at. While the Fully Greedy agents showcased slightly higher score efficiency, since they pick the fruit from a direct path whereas the others sometimes will dodge obstacles and take some extra steps, Partially Greedy agents got a slightly higher score average thanks to their ability to reach fruit positions that required collision avoidance which the other agent type can't do. The remaining metrics are practically indistinguishable in average and variance throughout all numeric metrics is the same which shows how both architectures are based on the same foundation.

Agents that follow Social Conventions

Since Social Convention Agents take actions that go towards the fruit, all the collisions were with obstacles between the cherry and their head, eliminating wall collisions. In addition, the agents prioritize actions that move them away from other snakes, minimizing those collisions.

Although death by collision with the other snake was greatly reduced, the agents maintained early losses due to self-collision, the main cause of death. That is evidenced by their extremely low step, score and efficiency averages.

We expected this agent architecture to be an improvement with respect to both Greedy agents so the fact that their results were worse in all possible numerical aspects came as a shock.

Intention Communication Agents

As expected, these agents had the best results. The average score and number of steps were the highest, around 18 and 180 respectively. Even though the agents' strategy led them to take unnecessarily long paths most of the time, the score efficiency was still the best, because they died less in the middle of their path to the fruit, in a sense making all steps count .

The Intention Communication Agents were the first to reach the step limit, as was anticipated by having the most elaborate and sensible strategy so far. However, the deaths were still mostly caused by collisions, in cases where the agent couldn't find any possible route. While 10% of the episodes reached the step limit, most of them were finished by agent's self-collision (42,5% of the 40 analyzed episodes). The increase in wall collisions comes as no surprise and it clearly is a tradeoff that is advantageous.

CONCLUSION

With the results obtained, there is no denying that cooperation is the key factor in this problem domain. While individual strategies that try to work for the benefit of the group (collision avoidance) were better than the baseline, communication is what provided a big leap in terms of successful runs in the environment.

Greedy behaviors were clearly too focused on the individual and in an environment where a mistake leads to the whole group dying/losing this becomes an issue. Therefore, this architecture is not suited for the real world where there isn't the luxury to risk collisions that may injure humans.

While we were not expecting Agents that follow Social Conventions to perform so poorly we believe this implementation was a step in the right direction as it addresses well the problem of collisions with others. In real contexts that are similar to our problem like self-driving cars, there is no possibility for self-collisions which means a strategy as ours would possibly yield great results in the real world (avoid car accidents).

Intention Communication was the way to go: its results were clearly the best even though they were limited by the maximum number of steps allowed. It is a strategy that makes sense in the real world and works similarly to technology that is being researched recently [7]. It prioritizes safety (which is a major concern in self-driving cars) which means the agents can reach their objective most of the time. If our strategy was improved with better priority assignments and preventive behavior for when there is no path available it could possibly become flawless.

We'd like to mention how it is a shame that we were unable to make Reinforcement Learning work as it is a method that usually achieves great results in tasks where it is applied. We are unsure whether it would be possible to apply RL to the real context of cars, since either training would have to be done virtually, which is not a perfect simulation of the real environment, or it would have to be done physically which would be very expensive in terms of resources and time.

In conclusion, our project achieved its main purposes and provided insights both for the problem at hand and similar real world contexts.

REFERENCES

[1] https://towardsdatascience.com/snake-played-by-a-deep-reinforcement-learning-agent-53f2c4331d36

[2] https://dl.acm.org/doi/10.5555/2484920.2485174

[3] https://github.com/marianaserrao/aasma-project

[4] https://github.com/divya-kustagi/python-stanford-cip/tree/master/final_project

[5] https://github.com/divya-kustagi

[6] https://leetcode.com/problems/shortest-path-in-a-grid-with-obstacles-elimination/solutions/2759192/simple-solution-using-bfs-traversal-58-faster-99-44-memory-efficient/

[7] https://en.wikipedia.org/wiki/Vehicle-to-everything