Competitive and Cooperative Multi-Agent System to Play The Human-Knot Game

Group 09: Alexandre Pires – 92414, Diogo Fouto – 93705, João Fonseca – 92497

ABSTRACT

While there are many multi-agent systems for cooperative games already, we propose and study new ones that play a game that creates action constraints among cooperating agents – the Human-Knot. In this game, agents in the knot have to coordinate their actions and consider their impact on one another. We present the benefits, limitations, and difficulties of each developed agent, which include Social Roles and Reinforcement Learning approaches.

1 INTRODUCTION

Here we introduce the motivation behind our project, present some related past work and where our project meets and diverges from it, and formally present the problem our multi-agent system tries to solve. We also present the objectives of our project. Finally, we introduce the architectures of our intelligent systems.

1.1 Motivation and Related Work

There are many collaborative multi-agent teams that play against each other in competitive games (see [1] and [3]), but while the agents' actions usually influence the outcome of the game, they rarely directly influence the possible actions (or outcomes of the actions) of one another.

For example, in Open AI's Hide and Seek [1], agents collaborate and even coordinate the roles among themselves – which also happens to be an objective of our project –, but an agent's action places no direct constraints on what actions its teammates can immediately take. With this in mind, we aim to create a game where the agents must act not only in ways that seem ideal to them and the team's overall objective, but also to consider how their actions impact their teammates' ability to act.

The aforementioned multi-agent game presents two other problems we aim to avoid: a complex set of rules and many complex interactions with the environment, which can impair the finding of working strategies. To do so, we designed a simpler environment where strategies requiring cooperation and coordination can still emerge.

1.2 Problem Definition

The problem our multi-agent system must solve is a modified Human-Knot game, as illustrated in Figure 1. Here, each of the two teams is composed of N agents. One team (the *fleers*) has to escape the knot for as long as possible, the other (the *catchers*) has to capture every *fleer* by encircling them with the knot. Each player is able to move in any direction in a two-dimensional space. However, the *fleers* cannot penetrate the knot and the *catchers* are chained together, as illustrated.

AASMA Project, IST, 2022, Lisbon, Portugal

https://doi.org/

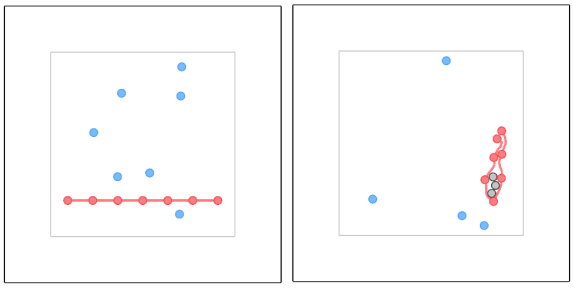

Figure 1: Left: A possible start setup for the game. The red agents are the *catchers*, and the blue agents are the *fleers*. Right: A possible middle-game setup for the game. In this case, three *fleers* (represented in gray) have been captured.

Encirclement happens when a *fleer* is inside a closed loop that the knot creates (the knot ends must meet). Once a *fleer* is encircled, it turns gray and can no longer move on its own – it is captured. However, the *catchers* can still push it around the arena. Furthermore, active *fleers* can revive gray *fleers* by touching them.

In order to prevent the *fleers* from running to the corners of the black wall and prevent their capture, they have a smaller walk-able space – the square composed by the gray lines in Figure 1.

The game ends when the timer runs out – the *fleers* win – or when the *catchers* capture all the *fleers* – the *catchers* win.

1.3 Relevance and Objectives

The relevance of the project is the study of this problem. The objective is to find the best possible solution to it. In practice, this will be a *Catcher* system that picks actions considering their outcome on other *catchers*. As such, we focused our efforts on developing several *Catcher* models. For the *fleers*, though, we created two simple models and leave as future work the development of better ones that strategically exploit the *catchers*' knot-length limitation in a coordinated manner and create opportunities to escape and save caught or endangered teammates.

2 EXPERIMENTAL SETUP

We used Unity Engine to create the game partly because of its simplicity and partly because of its 'Unity ML-Agents Toolkit' [8], which we employed to ease the development of Reinforcement Learning (RL) agents. The source code and the game itself are available on our GitHub page.

For the environment we fixed the following parameters: 7 Agents on each team; 60 seconds for each game; arena size; and movement speed.

The *catchers* spawn in a randomly positioned straight line inside the gray sqare, either horizontally or vertically. The *fleers* spawn in random positions, also inside the gray square.

All the agents can observe the following properties: the position of all *catchers* and *fleers*, the positions of the walls, if a *fleer* is caught or not, and if the *catchers* are making a circle or not.

2.1 *Fleers*

Since we focused our efforts on the *catchers*, we developed only two types of *fleers*: a *random* agent, which moves in random directions, and a *reactive* agent, which moves in the opposite direction of the *catchers* and saves the closest caught *fleer* if there is a clear straight line between them. When starting a round, the type of *fleer* is selected randomly from these two options.

2.2 *Catchers*

For the *catchers* we developed: a *V-Formation Agent*, a *Social Roles-based Agent*, an *Individual RL Agent*, a *Centralized RL agent*, and a *RL Agent with Curriculum Learning*. We also considered implementing a random agent, but ended up discarding it since the chance that they encircle a *fleer* is extremely low.

2.2.1 ***V-Formation Agent.*** For our baseline, we devised a functional model with no learning required. It was based upon studying the flight formation of birds. The basic idea was: when the game begins, *catchers* will travel to a V-formation flight position and the *catcher* in the midpoint will act as captain of the fleet and dictate its actions, namely, choosing the target *fleer* and the direction to take.

In our implementation, this idea became the Flight-Formation mode. The other mode is Circle mode, which we use to shift the *catchers* to a circle position and catch the *fleers* inside it. Here are the two modes in more detail:

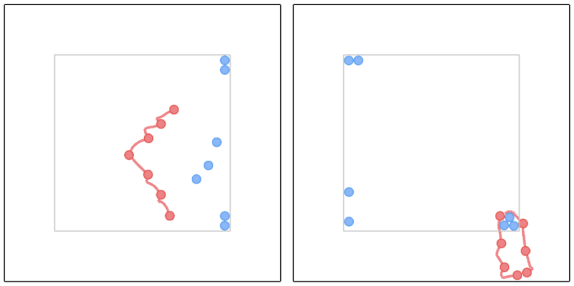

Figure 2: Left - *Catchers* in Flight-Formation Mode. Right - *Catchers* in Circle Mode.

- **Flight-Formation Mode:** the midpoint *catcher* chooses the nearest *fleer* as the target and, similarly to a deliberative agent architecture with a *blind commitment strategy*, it will not deviate from that decision until that *fleer* is caught. For the remaining *catchers*, we calculate their flight position by taking the vector from the midpoint catcher toward the target *fleer*, rotating it by 45°, normalizing it, and multiplying it by the difference of the midpoint index with the *catcher*'s own index (indexes are 0 to N-1 from left to right). Once the fleet is in formation, it moves toward the target.
- **Circle Mode:** once a *fleer* enters the triangle that the two endpoints and the midpoint form, we calculate a meeting point for the endpoint *catchers* to meet and close the circle. This meeting point is simply the midpoint of the two endpoints. However, if the endpoint *catchers* encounter an obstacle in their path, we rotate their direction by 120° and give them three seconds to get around it. While the endpoint *catchers* are moving toward the meeting point, the midpoint *catcher* moves toward the target and the others follow the neighbor that is closest to the endpoint of their side of the V-formation.

As one can see, this is mostly a reactive model. At the start of the game, the *catchers* default to Flight-Formation mode. When nearing a target, they switch to Circle mode. And once the target is caught, they return to Flight-Formation. As we will show in Results and Discussion, action and reaction with nary a dash of intelligence make this baseline our best model. However, since this is a reactive model, we had to severely simplify its algorithm, which could lead to problems down the line. For example, one simplification that is likely to be problematic is the assumption that the Flight-Formation mode should change to Circle mode when a *fleer* is inside the triangle made by the endpoints and the midpoint; if the *fleer* is actually outside the formation, a deadlock will occur.

2.2.2 ***Social Roles-based Agent.*** The attribution of roles is one way of engineering cooperation in a multi-agent system. It seemed appropriate for our problem because the state of the game was completely observable and because we could make the order of the roles, the attribution potential, and the fact that each agent could only have a single role, "common knowledge" to all agents. We just needed to create the roles, assign a set of actions to them, define an order for them, engineer a potential function, and attribute a role to each *catcher* as resolved by the aforementioned function.

With this theoretical framework in mind, we arrived at an agent architecture that is mostly reactive within a Social Roles-based cooperation framework. The target is selected in a similar manner to the previous agent, so we also find a resemblance to a deliberative agent architecture with a *blind commitment strategy* here. This agent, then, is a sort of hybrid agent, and the system is multi-agent, since the agents make decisions independently and cooperate and coordinate their actions.

These are the five roles we created:

- **Overtaking End Left / Right (OEL / R):** this is the role for the *endpoint catcher* that is closest to the left / right-side of the current target. Its objective is to overtake the target on the assigned side (i.e. left or right) so that the rope could then be tied and the *fleer* captured.
- **Tying End Left / Right (TEL / R):** this role would be assigned to an *endpoint catcher* if it has already overtaken the target, and was closest to its left / right-side. Its objective is to tie the rope and capture the *fleer* by moving towards the other *endpoint catcher*.
- **Follower (F):** this is the only role that could be assigned to *non-endpoint catchers*. Their only objective was to maintain a certain distance from the target, so that they would not get in the way of the *endpoints*.

The order that we have chosen for these roles was OEL ∼ TEL ≻ OER ∼ TER ≻ F, and our potential function is described in Algorithm 1.

Defining a correct potential function and a correct action space for each role in our game is difficult for many reasons, and even though we didn't manage to achieve a correct implementation, we think it would be possible to do so with more time. With this in mind, we expect the results of this architecture to be mediocre. Nonetheless, with a better potential function and action spaces, we still consider this a viable solution for our problem.

2.2.3 ***Individual RL Agent.*** With this model, each *catcher* is an individual agent that impacts its mates' actions, so we have a multi-agent system. These agents are rewarded as described in Table 1. We reward touching *fleers* to foster both learning to follow the fleers and increase the chance of connecting both ends during training; we punish touching the black (outer) wall to stimulate the emergence of strategies inside the *fleers'* area; and we penalize the passage of time in order to promote quick wins. The remaining are self-evident.

Unity Engine's ML Agents framework provides two built-in RL algorithms: Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC). PPO is a policy gradient method that OpenAI uses [10], which achieved particularly good results with Atari baselines. However, it is *on-policy*, as opposed to SAC, which is *off-policy* and more sample-efficient [6]. We decided on PPO, though, because, in our environment, rewards are handed out frequently, which is not an ideal scenario for SAC. PPO is also reportedly more stable [10][6]. Our agents will have two continuous actions in $[-1, 1]$: the x and y-axes. We leave SAC and other RL algorithms as future work.

Our environment has a fundamental problem: the bigger rewards are hard to come by, and when they do they are not easily reproducible since the positions of the agents are randomized every round. To stop the agents from repeatedly trying actions that only once gave good results, we tested with a Curiosity module in PPO. Although it is most applied when rewards are sparse [9], we hope that actions like encirclement, which rarely come by, are favored over, for example, just touching *fleers*.

Another expected limitation is that, given the observations, it might not be clear that the *catchers'* actions directly affect the actions of the teammates. A possible solution is adding more observations, which we leave as future work.

Lastly, the *catchers*'s perception of the other agents will be limited since their observations will make them seem like part of the environment, potentially leading to a "moving target" problem, since each catcher will adjust to others' actions by itself, leading to different actions by every one the next around, a subsequent individual policy adjustment to those actions, and so on.

2.2.4 ***Centralized RL Agent.*** Another approach to RL is to consider a "hyper-agent" that controls all the *catchers* at once. This way the *catchers* are neither individual agents nor do they interact with each other. So this is not a multi-agent system but an autonomous agent system.

The general setup is the same as in 2.2.3, but instead of the agents learning a policy within two continuous actions (the X and Y directions), one agent will learn a policy within fourteen continuous actions (2 actions × 7 agents). This may seem harder at first, but the "hyper-agent" will be in control of every *catcher*'s action and will not interpret them as an environment event. Because of this, we expect the results to be better and the training shorter. Nonetheless, the problems mentioned in 2.2.3 remain, which still make a good policy difficult to learn.

Outcome	Reward
1 Second Passes	-0.01
Touch Black Wall	-0.01
Touch Uncaught Fleer	+0.01
Catch Fleer	+0.5
Fleer Saved	-0.5
Win Game	+100
Lose Game	-1

Table 1: The rewards for each outcome.

2.2.5 ***RL Agent with Curriculum Learning.*** While devising the previous RL architectures, we speculated that letting a fully-formed *Catcher* team learn the game by finding rewards could be counterproductive. Catching a *fleer* involves a complex process of coordination and cooperation between all the players, and the chances that those will happen without proper training are extremely slim. Formally: "Deep learning methods attempt to learn feature hierarchies. Features at higher levels are formed by the composition of lower level features. Automatically learning multiple levels of abstraction may allow a system to induce complex functions mapping the input to the output directly from data, without depending heavily on human-crafted features. (…) Training deep architectures is a difficult problem" [2]. To solve this, we studied other methods and discovered that, "unsupervised models such as Restricted Boltzmann Machines (RBMs) can be used to initialize the network in a region of the parameter space that makes it easier to subsequently find a good minimum of the supervised objective" [4]. [2] goes on to prove that a technique called Curriculum Learning acts similarly to the unsupervised pre-training in [4] and makes for a better training regimen than a typical RL strategy.

What, then, is Curriculum Learning? "Since neural network architectures are inspired by the human brain, it seems reasonable to consider that the learning process should also be inspired by how humans learn. One essential difference from how machines are typically trained is that humans learn the basic (easy) concepts sooner and the advanced (hard) concepts later" [11]. This is what Curriculum Learning is: a sequence of training levels, from easy to hard.

For our implementation of Curriculum Learning, we reused the Individual-Based RL model in 2.2.3 and the rewards in Table 1 and added a learning curriculum. The main components of a curriculum (see [5]) are:

- **Environment Parameter:** the parameter that changes every level;
- **Training Levels**;
- **Threshold:** reward needed to pass to the next level;
- **Minimum Number of Lessons per Level**.

In our case, the environment parameter is the team size. We start with the minimum number of players (one *catcher* and one *fleer*) and progress steadily. The training levels are as follows:

(1) ***Catcher* to *Fleer***: The goal is for the one *catcher* to touch the one *fleer*. The threshold is the reward of touching the fleer for a few seconds minus the time it takes to reach it.
(2) **Two *catchers* moving**: Same goal as level 1, but now two *catchers* have to cooperate and coordinate their actions.
(3) **Three *catchers* moving**: Same goal as level 2, but with three *catchers*.
(4) **Forming a circle**: The goal is to form a circle with three *catchers*. The minimum number of lessons for this level is much lower than for the others because we don't want the *catchers* to learn just to form circles.
(5) **Catching a *fleer***: The goal is for four *catchers* to move toward a *fleer*, form a circle, and catch it.
(6) **Winning the game**: The goal is to win the game, with the number of *catchers* ranging from five to ten. There is no limit of lessons for this level; the training continues until stopped.

This model takes the strength of traditional RL and submits it to a proper training regimen, so we expect good results. However, a long training time and inevitable level tweaks are limitations that can delay and affect the model's usefulness. Furthermore, Curriculum Learning has one major downside called "catastrophic forgetting." This means that, as the training progresses, an agent is bound to forget some of what it has learned in the past. To counterbalance this, we implemented *interleaving* – repeating and mixing training levels.

3 EMPIRICAL EVALUATION

To evaluate the performance of our multi-agent system, we present the following metrics:

- **Duration of the game**, *D*. If the game is very short, it's a sign that the *catchers* did very well. But if it was the timer that ended the game, then the *fleers* managed to have at least one of their members survive, beating the *catchers*;
- **Number of *fleers* caught**, *C*. This measures the coordination of the *catchers*; better coordination leads to more catches.
- **Revive / Catch ratio of the *fleers***, *RC*. This measures the cooperation capacity of the *catchers* to not only capture, but also to stop *fleers* from saving those that were caught. The closer it is to zero, the higher the level of cooperation and coordination of the *catchers* regarding this component of the game.
- **Net Force Quotient**, NFQ.

  Definition 3.1. If we have i *catchers*, $\vec{F}^i$ is the force i is exerting to move, and $\vec{E}_i$ is the sum of the forces that other *catchers* are exerting on *catcher i* through the rope, we define the Net Force Quotient (NFQ) for the x axis as

  $$\text{NFQ}_x = \frac{|\sum_i (F_x^i + E_x^i)|}{\sum_i (|F_x^i| + |E_x^i|)}$$

  If $\sum_i (|F_x^i| + |E_x^i|) = 0$, then NFQ = 0. NFQ_y can be defined in a similar way.

  NFQ measures coordination between *catchers*, since it will be 0 if there's no movement (either because none of them is accelerating or because they're moving in a non-constructive way), or be closer to one if they're moving in a coordinated way.

We will use these metrics to compare each system's performance and shortcomings. All these metrics are updated in real time.

An additional metric for the learning *catchers* is the evolution of the reward obtained during training. A stagnated reward coupled with sub-optimal performance in the other metrics suggests problems related to reward distribution and/or training initialization and implies fundamental performance limitations in a given system.

4 RESULTS AND DISCUSSION

Here we present the results of each agent type. First, we compare them over 100 rounds of the game. Then, we analyze each in more detail.

4.1 General Results

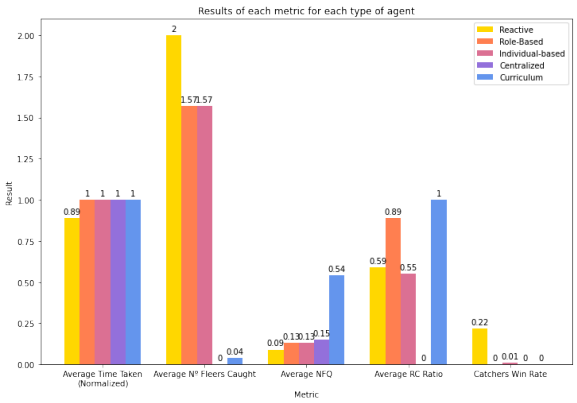

Figure 3: Results of each metric for each type of agent, obtained from an average of 100 games. Time is normalized, so a value of 1 corresponds to 60 seconds (A full round duration). The *fleer* population was mixed.

The general results are in Figure 3. Our baseline – the V-Formation agent – had the best results in every metric except NFQ. The Individual-based RL agent had the second best win rate and a low R/C ratio, which means it was effective at blocking *fleers* from saving one another. The Social Roles-based agent had a worse win rate and R/C ratio when compared to Individual-based but had an equal average number of *fleers* caught, which suggest a similar *fleer*-catching ability but a worse *fleer*-blocking strategy. The other two agents – Centralized and Curriculum – had the poorest results. Centralized seems to have an remarkably low R/C ratio, but this is misleading; it caught no *fleers*, so none could be revived. However, Curriculum scored a high NFQ, which suggests that the *catchers* coordinated their actions well.

4.2 V-Formation Agent

A quick glance over Figure 3 reveals the superior performance of V-Formation Agents: the win rate dwarfs the others by more than

20%, the average time taken is 10% better, and the R/C ratio is the second lowest. We now dissect the reasons for these results (the *pros*) and explore the model's problems and limitations (the *cons*).

The *pros*:

- **Flight Formation Triangle**: to decide whether the *catchers* should enter Circle mode or not, we check if a *fleer* is inside the triangle that the endpoints and the midpoint create. The strength of Flight Formation is that we already construct this triangle ahead of time, and so it is easier to achieve the conditions to catch a *fleer*.
- **Avoiding Obstacles**: if the *catchers* are in Circle mode and encounter an obstacle, we rotate their direction by 120° and give them three seconds to veer around it. This helps in avoiding entanglements and deadlocks.
- **Captain of the Fleet**: to avoid coordination problems, only one *catcher*, the midpoint, makes decisions. The other *catchers*' job is to follow its lead when in Flight Formation mode, and reach the meeting point when in Circle mode.

The *cons*:

- **No Rotation**: in some situations, the *catchers* will entangle themselves and create a deadlock. For instance, if after catching a *fleer* the next target happens to be behind the midpoint, the *catchers* in both sides of the V-formation will try to travel to their flight position through a path that penetrates the other side. Figure 4 illustrates this situation. Another example is when the *catchers* crash into a wall; their direction vectors cross one another and cause another entanglement. In other words, the formation enters a deadlock when it should, but can't, rotate.
- **No *Fleer* Blocking**: to win the game, *catchers* have to catch all *fleers*, but the *fleers* can save one another. In this model, there's no method to prevent this from happening.

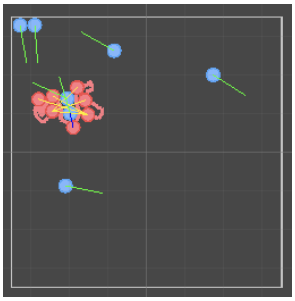

Figure 4: *Catchers* in deadlock. The yellow lines show the trajectory the *catchers* follow. The green lines show the *fleers*'.

Overall, the V-Formation Agent model behaves as intended and delivers intriguing results. With more time, though, we could fix the situations where it falters and achieve an even better outcome.

4.3 Social Roles-based Agent

As Figure 3 shows, this agent had very bad results. It won no games and caught on average between one and two *fleers* per round. The *fleers* revived their peers 89% of the time, which reveals that the agent had a poor *fleer*-blocking strategy (this was expected since it wasn't programmed into the potential function or the action spaces of the roles). As for the NFQ, a close-to-zero value unmasks the little coordination the agents displayed.

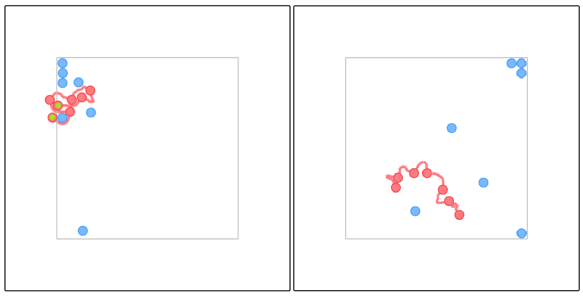

Figure 5: Left: The endpoints have a lime dot on top of them. As one can see, spacing each *catcher* in relation to each of its neighbours wouldn't solve the lack of coordination presented here. Right: the position that was thought of when engineering this agent.

We now attempt to explain the reasons for these results. The action space for each role and the potential function that we defined proved to be incorrect, hard to think about, and hard to rectify. We developed various iterations for both of them, but never accomplished a better performance with any of them. This was due to technical reasons, which are related to the representation of the environment, and to the difficulty of the game itself. There was also an issue related to the *blind commitment strategy*: sometimes the *catchers* would lock onto a *fleer* that would circle the arena *ad infinitum* and prevent the catchers from ever catching it (unless, of course, the *fleer* crashed into another one). This phenomenon is known as *overcommitment*. To solve it, the *catchers* need to re-evaluate their intention more often, and a single-minded commitment strategy would probably suffice.

The major problem, though, was the inevitable deadlock induced by a lack of coordination. Since the actions taken and the roles assigned only use vectors (i.e. straight lines) and angles between them to be computed, the catchers entangle themselves and thwart each other's movement (see left-hand side of Figure 5). One way of fixing this would be by making it so the agents always maintain a certain distance from each other while also acting their role and chasing the target their own way. This sounds straightforward, but it's not clear how one would implement this; the simple sum of seven forces (6 away from the other *catchers* + 1 toward the target) to each agent's action would create a very confused, perhaps even seemingly random, direction for the agent. Even if we only took into consideration the neighbors of each *catcher* and tried to maintain a certain distance from them, the problem would linger.

As for technical difficulties, it proved extremely difficult to define what *behind the target* meant in a Cartesian plane with added physics; *behind* is always relative to the position of the observer and to the angle in which the observer is observing the target, and sometimes even the direction of the velocity of the target. While what we have suggested works for the configuration on the right-hand side of Figure 5, it stops working when the *endpoints* are stacked as illustrated on the left-hand side of Figure 5.

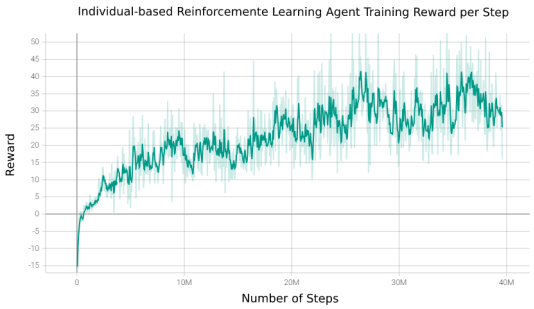

Figure 6: The average reward and standard deviation every 50000 steps of training for Individual-based Reinforcement learning. *fleers*

4.4 Individual RL Agent

This model had a troubled training process (Figure 6 shows the reward obtained per number of steps). It caught a few *fleers* and even won some rounds, but it learned no *fleer*-blocking strategy, like the R/C ratio suggests. Furthermore, it had trouble in coordinating its actions and in making circles. It also suffered from entanglements.

Nonetheless, we witnessed promising strategies. *Catchers* frequently pushed *fleers* against the gray walls, which allowed to catch them and prevent their revival. It also learned a diagonal *sweeping* strategy that exploits the movement of the greedy *fleers* in order to catch them.

The main impediments were the random starting positions and the random *fleer* types. This means that actions that worked in a round (e.g: encircling in the top right corner) very often didn't work in the next. This leads to a worse performance every few good runs, as Figure 6 shows.

It is worth mentioning that the agent trained only for 17 hours. It is expected that, with further training, the model would give more respectable results, but it is unclear how long that would take. We leave a more extensive training process as future work.

Despite the low win rate, the model boasts of a respectable number of average *fleers* caught, which suggests a better performance is possible with future optimizations.

When researching the difficulty of the training, we experimented fixing the *fleer*'s type and initial positions, which yielded the results in Figure 7. Even though this ensues a deterministic game, which is easier to win, the outcome was much better and gave hope for future work. The catchers learned much-hoped-for strategies. For example, after capturing a *fleer*, they often prevented its revival.

We also experimented with the curiosity module, but the result was disappointing: there was no evolution in the training. This may be due to the frequent handing out of negative rewards, which punishes curiosity (see [9]), or due to hyper-parameter tuning.

4.5 Centralized RL Agent

This agent had a very poor performance. It learned to touch as many *fleers* as it could, but didn't learn to catch any. This happens because the reward for touching *fleers* is easy to come by while catching them is not. Figure 8 illustrates this. As the agent learns to touch the *fleers*, the reward increases, but it plateaus before the

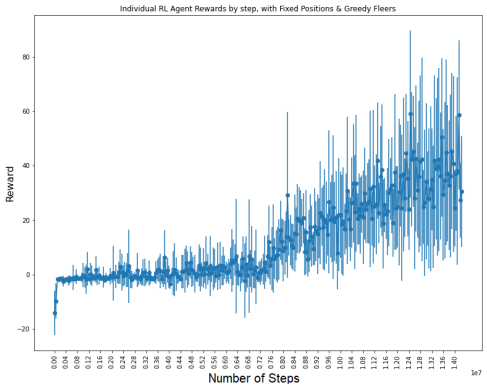

Figure 7: The average reward and standard deviation every 50000 steps of training for Individual-based Reinforcement learning with fixed starting positions and only greedy. *fleers*

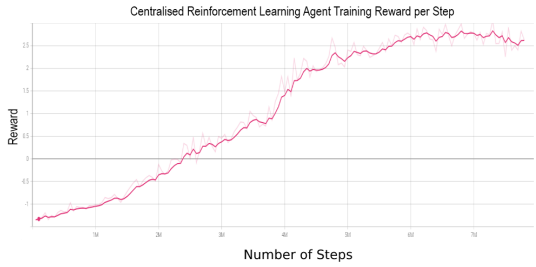

Figure 8: The average reward and standard deviation every 50000 steps of training for the Centralized RL agent. *fleers*

it ever learns how to encircle one. We recognize that our reward distribution promotes the wrong task.

Other limiting factors are the dimensions of the action space, which totals 14 continuous actions. This is particularly clear here as opposed to the Individual RL model because there is only one agent, which leads to fewer overall samples and, consequentially, a slower training. The basic solution is more time to train, which we didn't have.

Furthermore, we note that this agent displayed consistently more synchronized movement than the *Individual RL* agent. The *catchers* frequently surrounded the *fleers* in a single long line, the better to touch as many of them as possible. We believe this is because the *Centralized* agent has full control of all *catchers*, which leads to a faster understanding of the underlying dynamics related to the knot.

We conclude that performance might increase with more training time and a reward system that promotes encircling and not merely touching *fleers*. These we leave as future work.

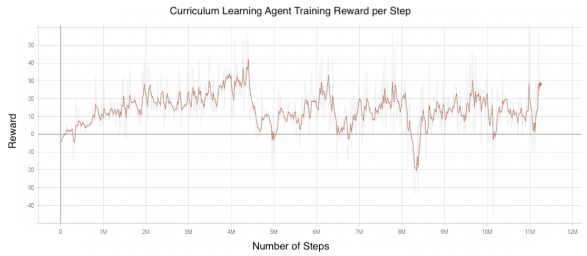

Figure 9: The average reward and standard deviation every 50000 steps of training for Curriculum Learning without *Interleaving*.

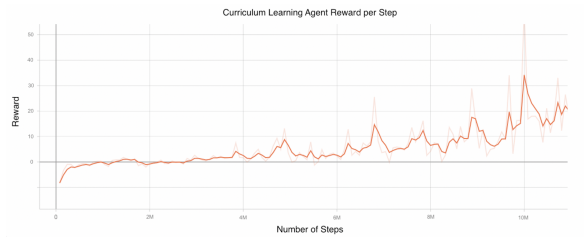

Figure 10: The average reward and standard deviation every 50000 steps of training for Curriculum Learning with *Interleaving*.

4.6 RL Agent with Curriculum Learning

Although promising, the Curriculum Learning (CL) model didn't live up to its full potential. As we said in section 2, CL takes the strength of RL and puts it through a, hopefully better, practice regimen. We hoped that with the levels described in section 2.2.5, the model would retain what it had learned, improve level by level, and, in the end, win the game. Unfortunately, this didn't happen. Curriculum Learning won no games, caught a paltry 4% of *fleers*, and, from a statistical point of view, ended up being one of the worst performing models.

However, statistics don't convey the full picture. When playing the game, CL *catchers* tried to position themselves in a formation that allows touching many *fleers*, and, when trying to catch one, they almost managed to form a circle around it. This suggests that with further training Curriculum Learning might deliver decent, and even good, results. For now, though, we will try and shed some light on what happened and hint at future work.

Our first attempt at implementing CL delivered poor results (see Figure 9). The model quickly learned and glided through the levels, but by the time it reached the last one it had completely forgotten the previous ones — a phenomenon which, if you recall, is called *catastrophic learning*. For instance, in the first two levels it successfully learned to travel toward a *fleer*, touch it, and follow it around the arena, but a few levels afterward it gave up on doing that and instead was hooked on making circles. To amend this we implemented *interleaving* and repeated older levels throughout the curriculum to remind the catchers of what they might have forgotten otherwise. This delivered better results (see Figure 10), but was still a far cry from what we expected.

Another problem we had was that, although the model had learned to catch *fleers* in an earlier lesson, it started valuing touching them far more and completely dismissed the idea of catching one. This suggests an imbalance in the reward system (see Table 1). We tried to tweak it a little, but we didn't have much time and, therefore, the results were subpar. In the future, a more careful and well-thought-out reward system could deliver a much better outcome.

The last problem was the endless need for tweaking the curriculum. There are many variables to take into account when designing the levels: the reward threshold to advance to the next levels, the number of lessons, which levels go in what order, how many times to repeat each level, etc. Their never-ending tweaking prevented the final version of Curriculum Learning from training for as long as we wanted it to.

Because of all this, Curriculum Learning wound up performing worse than the other RL models. Nevertheless, we strongly believe that with an improved curriculum design, a better reward system, and more training time, Curriculum Learning would reach new heights and, potentially, outperform all the other models.

5 CONCLUSION

With this project we sought to study the capacity of cooperation and coordination among agents in a setting where their actions restrict one another. Although we never deemed it a trivial task, we were surprised by how difficult it proved to be. Reinforcement Learning, the family of algorithms we assumed would overwhelmingly outperform our simple rule-based agents, wasn't able to achieve this at all. We still consider it to have great potential, but, to unleash it, different concepts, innovative ideas, and a lot of work remain.

6 FUTURE WORK

Many other frameworks and models remain to be explored. Because of the difficulty of the encirclement task, we recommend the following:

- *Imitation learning*. For example, Generative Adversarial Imitation Learning in a context where the Discriminator tries to distinguish a demonstration from a Generator imitation [7];
- Other *Reinforcement Learning* approaches. For instance, the Soft Actor-Critic algorithm, which is supported in Unity Engine's ML Agents framework;
- *Social Roles*-based agents with better potential function and actions spaces;
- Different strategies for *Reactive* agents.

Furthermore, and as mentioned previously, little training time severely limited the performance of our RL agents. We recommend training the same models for a longer while.

Finally, there is the design of better *Fleer* models. Smarter revival mechanisms, *catcher*-distraction abilities, and better evasion strategies are concepts worth exploring. One could adapt the models studied in this paper or devise entirely new ones.

REFERENCES

[1] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. 2019. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528* (2019).

[2] Y. Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. *Journal of the American Podiatry Association* 60, 6. https://doi.org/10.1145/1553374.1553380
[3] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).
[4] Dumitru Erhan, Y. Bengio, Aaron Courville, and Pascal Vincent. 2009. Visualizing Higher-Layer Features of a Deep Network. *Technical Report, Univeristé de Montréal* (01 2009).
[5] gzrjzcx. 2018. ML-agents. https://github.com/gzrjzcx/ML-agents/blob/master/docs/Training-Curriculum-Learning.md. (2018).
[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. PMLR, 1861–1870.
[7] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. *Advances in neural information processing systems* 29 (2016).
[8] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. 2018. Unity: A General Platform for Intelligent Agents. (2018). https://doi.org/10.48550/ARXIV.1809.02627
[9] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*. PMLR, 2778–2787.
[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
[11] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. 2021. Curriculum Learning: A Survey. (2021). https://doi.org/10.48550/ARXIV.2101.10382

A APPENDIX

Algorithm 1 The potential function algorithm

Require: catcher
Require: $\vec{C}$
Require: $\vec{T}$
  if catcher.isEndpoint = False **then**
    catcher.role ← F
  else
    if $\vec{C}$.up · $\vec{T} \geq 0$ **then**
      behindMagnitude ← $||\vec{T}|| \times 1.15$
    else
      behindMagnitude ← $||\vec{T}|| \times 0.85$
    end if

    if behindMagnitude < $||\text{proj}_{\vec{T}}\vec{C}||$ **then**
      if $\theta_{TC} < 0$ and TEL is not assigned **then**
        catcher.role ← TEL
      else
        catcher.role ← TER
      end if
    else
      if $\theta_{TC} < 0$ and *OEL* is not assigned **then**
        catcher.role ← OEL
      else
        catcher.role ← OER
      end if
    end if
  end if