Vision Evolution: Natural Selection shaped by environment

Yassine Abdennadher (299273), Ambroise Borbely (295747), Matthieu André (300245)
CS-503 Final Project Report

***Abstract*—In this project we investigate the adaptation of an agent's perception system to its environment in the task of gathering resources under the constraint of energy. We simulate the agent, its perception and the environment using a game engine, then we explore different visual configurations using a 2 step process alternating between changing the agents vision and training its behavior to evaluate the performance of its vision. With this we are able to train an agent to gather resources as well as to explore different types of vision. Our code is available at: https://github.com/ambor1011/CS-503-project/tree/master**

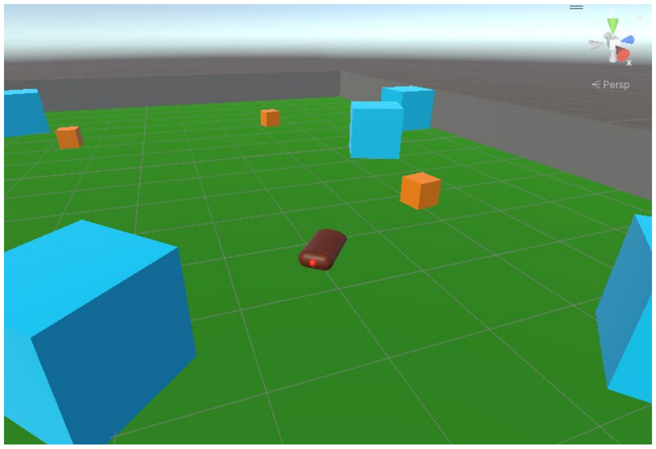

Figure 1. Our simulation environment with the agent, obstacles and resources

I. Introduction

In nature there exists a great deal of variety. There are different environments, different plants and different animals. Among all living things we observe a great deal of diversity in shape, color, size and more... On of the feature that changes among particularly animals is their perception system. Multiple theories exist as to explain the emergence of this diversity and why particular animals have a particular vision. There are multiple theories that attempt to explain vision such as structuralism that argues that vision is the of its parts or Gestalt-ism that is a proponent of the theory that vision is more than the sum of its parts. We we're however most convinced by the ecological theory which argues for the importance and influence of the environment on the agents perception system. To investigate this theory further we want to run experiments in simulation where the environment will influence the development of the vision of agents.

Investigation of this problem will give profound insights in multiple fields. By investigating the trade-off between different visual capabilities under the constraint of energy we will be able to better identify the important features of a perception system which will allows us to develop more adapted systems to the environments where they are deployed, while minimizing the number and type of perception sensors that are employed. Moreover by placing just the perception sensors the system will also need less processing power than if too many sensors are placed without optimization.

II. Related Work

While a great deal of research exist on the adaptive configuration of an agent to its task or its environment, not much research has been put to into the computational design of a perception system. We theorize that this is probably due to the computational difficulty of doing so as you need to evaluate each new configuration and contrary to the design of a body for which a physical simulation could be used to evaluate. For a perception system you usually need to use learning based methods as there are not physically based methods. So we mainly got inspired by those works such as Evolving Virtual Creatures [1] and Flexible Muscle-Based Locomotion for Bipedal Creatures [2].

III. Method

A. Simulation

To conduct our study we needed designed a simulation environment in a popular game engine called Unity. We chose Unity for its flexibility and an already well established simulation framework called MLagents [3] which takes care of the communication between Pytorch and Unity as well as providing implementation of multiple reinforcement learning algorithms such as PPO [4].

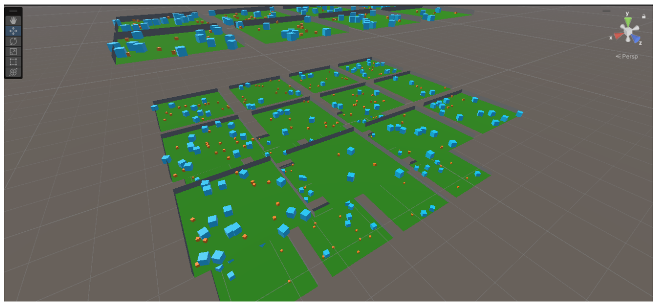

Figure 2. Multiple environments to speed up training

B. *Environment*

Our simulation environment as seen on the picture consists of a green square plane where the agent can move, bordered by walls so that the agent does not need to learn to not fall off as this was not one of our goals. On the plane we then place 3 more entities: small orange cubes representing food resources, big blue cubes representing obstacles blocking the agents movement and vision and finally we place the agent itself. Both the resources and the obstacles are placed by sampling x and y positions from a uniform distribution while taking care of not placing resources or the agent into the obstacles as the agent would be stuck or the resource would be invisible and unreachable (as the agent is also placed randomly). At the end of each episode we regenerate a new environment by repeating the procedure.

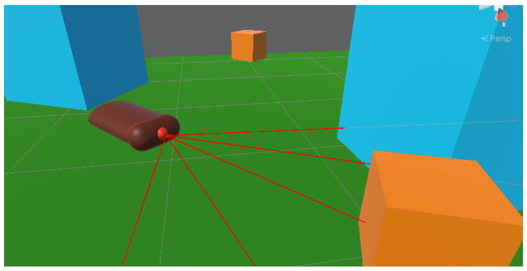

Figure 3. Visualization of the agent's vision with 5 eyes

C. *Agent*

Our agent is modeled by the brown shape. Inspired by point-nav we decided to make the agent's actions discrete so that it can be trained more effectively without sacrificing the freedom of movement. Hence our agent can take one of 3 actions: turn left, turn right or go forward. On turning the agent turns by 15° this way it can do a full rotation in 24 steps allowing it can change its direction nimbly without turning too much. Each agent has a vision configurations that can change from agent to agent. In this project we explored 2 factors namely the number of eyes and the field of view of the agent. We limited ourselves to 2 factors as with each new factor the dimension of the vision spaces increased which would make our vision algorithm take exponentially longer time to explore the visual parameters with respect to the dimension. On the topic of eye placement we decided to place them evenly on the along the field of view, for example an agent with 5 eyes and a field of view of 100° would have eyes spaced by 25 degrees. Talking about eyes, the vision of our agent is implemented using rays meaning that one eye corresponds to 1 ray which then returns the color (RGB vector) of the object it hit or the 0 vector if there was no hit. Using rays instead of a more realistic sensor such as a camera was a strategic choice along 2 levels. First and foremost there is the computational cost using where using rays speeds up the simulation by multiple times as we can pass these observations directly to the network instead of having to train a vision encoder at the same time as the policy and we can run the simulation without rendering the scene. The second was complexity as it already took us a great deal of effort to connect Unity and Pytorch and in the scope of this project it was more important for us to work on the evolution of vision over realism. Finally each agent has an energy level that allows it to stay alive as long as it stays above 0.

D. *Training*

Each time the vision configuration of our agents changes we need to retrain the resource gathering behavior to fully use its visual capacities. For that we use the Proximal Policy Optimization algorithm as it is stable among other reinforcement learning methods and allows the agent to learn effectively. At the start of each episode a new environment with obstacles and resources is generated and the agent is placed on an unobstructed place. All the agent start with the same energy value and at each step they consume a bit energy depending on the number of eyes as with more eyes the agent needs a larger neural network which needs more computation that translates into consuming more energy by step, and they receive energy if they gather a resource. An episode lasts until one of the following: the agents energy drops below zero and it dies, the agent gathers all resources or the episode reaches a max length. Finally during training we run multiple agents and environments in parallel (usually 20) to speed up the training.

E. *Vision Algorithm*

Once the agent is trained we rank it with respect to agent with a different vision, with an agent score being computed by its cumulative reward. The vision search space is discretised as well to allow for faster exploration. We also limits to the 2 parameters to be somewhat reasonable in terms of the space to explore. Thus we limit the number of eyes between 1 and 19 with a step size of 2 and the field of view between 15 and 180 degrees with a step size of 15. To explore the vision space we use a discrete gradient ascent where for an agent we investigate its neighbors. Given that we vary the 2 parameters : this can be represented as a grid and we look at the neighbors to the left, top, right and bottom. After evaluating the neighbors we pick the neighbor with the highest cumulative reward and we repeat the process. If on the other hand no neighbor has achieved a higher score, we found a local maximum and we end the evolution for this run.

IV. Experiments

To evaluate our method we run multiple experiments. We start the agent with one set of parameters, namely the field of view (fov) and number of eyes. The agent is then trained and evaluated using the cumulative reward. At each step the

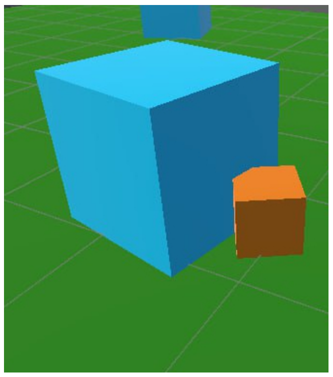

Figure 4. The food and obstacle are represented as small orange and large blue blocks respectively

agent gets a small penalty reward of -0.005 as the agent consumes energy (proportionally to the number of eyes) at each step. We give a big penalty of -1 if the agents energy becomes negative and it ”dies” and finally we give a big reward of +1 for each food resource collected. We then run the vision adaptation multiple times starting from different positions on the 2d grid of parameters. We decided to start in each of the corners as well as from the center to maximize the covered space.

A. *Results*

We successfully implemented a pipeline that allows us to change the perception of our agents using python scripts communicating with the Unity game engine and we are able to train agents, evaluate them, change their perception and repeat. We observe that using our evolution algorithm we visit multiple vision configurations and always reach a local maxima. We also observed that in our parameter space there are multiple local maximas. Next we observe that the exploration of the vision space usually does not go very far before converging and stopping. Finally we observe that agents that only have one 1 eye perform much worse than the other agents.

Start Vision	**End Vision**	**Highest Cumulative Reward**
fov 15, eyes 1	fov 15, eyes 9	-5.32
fov 15, eyes 19	fov 30, eyes 19	-2.61
fov 180, eyes 1	fov 180, eyes 19	-3.07
fov 180, eyes 19	fov 165, eyes 17	-3.06
fov 90, eyes 9	fov 90, eyes 7	-4.43

Table I
EXAMPLE TABLE WITH START VISION, END VISION, AND HIGHEST CUMULATIVE REWARD ATTAINED

B. *Analysis*

After running the experiments and looking at the results we can deduces multiple insights. Namely the fact that most of the evolution of vision stays in the neighborhood along the local maxima indicates that there are multiple configurations that get stuck with the current vision exploratory algorithm, never reaching the global maximum. We also see that having only 1 eye is very disadvantageous and just the fact of having 3 and probably 2 makes a big difference. This is explained by our ray implementation. With a single ray you cannot know the proper direction in which to turn if you see something of interest as if it your ray is not colliding with it anymore you do not know if it is to the right of you or the left, on the other hand if you have at least 2 rays (in our case the next possible number was 3) you can learn to infer the proper rotation. We also see that between 2 neighbor in the vision space the difference in cumulative reward is small which indicates a pretty smooth gradient which is expected as we change our agent’s configuration only a little bit a a time.

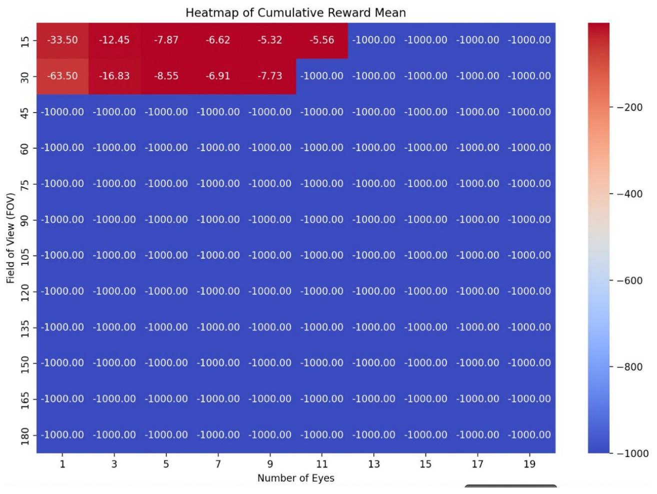

Figure 5. Exploration result starting with 1 eye and 15 fov, that converged on 9 eyes and 15 fov

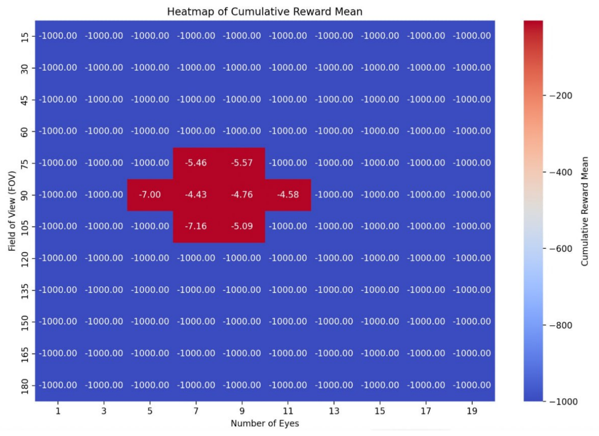

Figure 6. Exploration result starting with 9 eye and 90 fov, that converged on 7 eyes and 90 fov

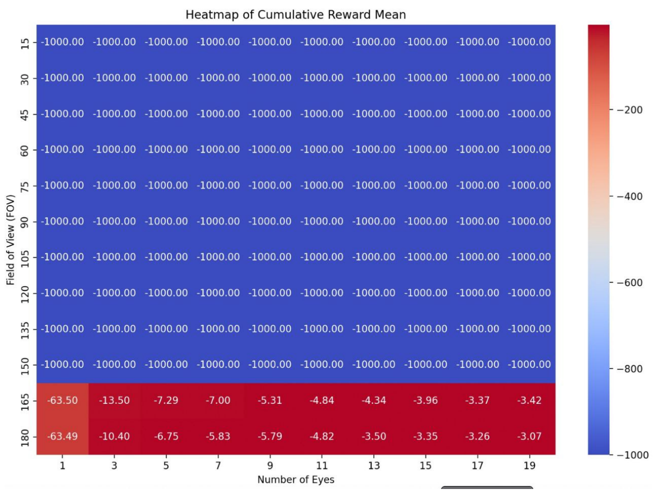

Figure 7. Exploration result starting with 1 eye and 180 fov, that converged on 19 eyes and 180 fov

V. Conclusion and Limitations

A. Conclusion

In summary we have explored the adaptation of a visual agent to a changing environment ensuring that our agent does not overfit to a single environment by memorizing it and using its vision parameters. We are able to train an agent to seek resources while avoiding obstacles using its vision. Moreover we have developed an algorithm that allows us to explore the visual parameters that are similar to the agent and to evaluate their performance using the cumulative reward.

B. Takeaways

Our main takeaways from this project are that you can adapt the perception of an agent to an environment using our method. At the same time we learned about the computational cost of doing so and addressed it by taking key simplifications such as using rays instead of a camera and keeping the simulation environment from getting too complex. We also learned that computational design for a perception system is a promising field of research as we were able to steer to evolution of our agents perception towards a vision with better performance.

C. Limitations and possible Solutions

While our work presents a good initial base to explore the computational design of perception of agent. There are many possible extensions and it could be ameliorated. In the current iteration we use rays as the perception mechanism for our agents which is not very realistic. Instead we could use a camera based perception and add a visual encoder This was however computationally a lot more expensive so we decided against pursuing this route with the short timeline of the project. The second limitation we observed was that our exploration algorithm for vision was quite conservative, quickly reaching a local maxima. We could address this by adding some stochasticity to our algorithm to for example have a small probability of jumping to another vision configuration or instead of taking the greedy approach and taking the cell with highest score, select on of the best cells in a top K style.

VI. Individual contributions

- Ambroise and Yassine worked on fixing the training of the agent as upon closer inspection the learned behavior as the results were not satisfying yet.
- Ambroise implemented the algorithm that changes the vision parameters after an agent has been trained as well as the communication between unity and pytorch.
- Yassine ran experiments to obtain results
- Matthieu analyzed the results and wrote the report.
- Matthieu Made the presentation.

References

[1] K. Sims, "Evolving virtual creatures," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '94. New York, NY, USA: Association for Computing Machinery, 1994, p. 15–22. [Online]. Available: https://doi.org/10.1145/192161.192167

[2] T. Geijtenbeek, M. van de Panne, and A. F. van der Stappen, "Flexible muscle-based locomotion for bipedal creatures," *ACM Trans. Graph.*, vol. 32, no. 6, nov 2013. [Online]. Available: https://doi.org/10.1145/2508363.2508399

[3] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2020. [Online]. Available: https://arxiv.org/pdf/1809.02627.pdf

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.