***Final Project – Snake***

*Jonathan Anton & Camil Blanchet*

*CS 5100: Foundations of Artificial Intelligence*

***Snake Background***

A picture containing text, clock

Description automatically generated The popular arcade game snake is a single-agent dual-purpose game. Snake takes place in a n*x*n grid where a constantly moving Snake agent seeks food. When the Snake agent’s head eats food the snake body grows by one unit. The objectives of snake are to grow as long of a snake body as possible. If the head of the snake runs into the outside of the grid or into its own body then the snake dies and the game is over. This format leads the snake agent to seek out intelligent paths to efficiently eat the randomly generated crumbs of food. The snake must minimize the area its body takes up as it gets longer as to not close off areas of the grid whereby food might generate once the current food objective has been eaten.

***Online Resources***

There are many examples of simple Snake implementations where-by rule-based agents can achieve a perfect score. PythonSpot’s simple AI agent finds the shortest path the current food location on the grid (1). Amnika Choudhury provided useful background comparing the performances of convolution neural network trained with Q-learning which outperformed both a deep q-learning network based agent and human performance for both game score and survival time. Her writing follows work from researchers at the University of Poland who also implemented a genetic algorithm (2).

Several other useful resources were found but they were either out of scope to be useful for this project or behind a paywall. An evolution-based agent written by Peter Binggeser from Becoming Human would have been a useful resource (3) as would a Toward Data Science literature review titled ‘Training a Snake Game AI: A Literature Review’ written by Thomas Hikaru Clark (4).

***Snake Platforms***

The most popular python implementation of snake can be found on the OpenAI Gym (5). While OpenAI Gym provides a useful resource, we decided to stick with a simpler snake game loop in order to ensure that our modifications to the game logic and agents being used didn’t cause unnecessary bugs in other parts of the codebase. For this reason, we used Wajiha Urooj’s implementation using PyGame (6).

***Performance Measures***

The primary performance metric will be total size of snake when the game is over. The size of the snake is directly proportional to how many ‘food’ bites the snake has eaten. A superior agent is the agent that attains the largest size. If two agents tend to perform similarly in terms of average size when the game is over than a secondary metric to compare is path efficiency. This can be evaluated based on the total number of steps taken before death, with a lower number of total steps for the same length achieved indicating more efficient routing.

***Results***

Running our Manhattan, A\*, Longest Path and Hamilton agents through 1000 iterations of Snake the following statistics were achieved.

|  |  |  |
| --- | --- | --- |
| **Agent** | **Average Length** | **Average Steps** |
| Manhattan | 10 | 80 |
| A\* | 4 | 35 |
| Longest Path | 8 | 72 |
| Hamilton | 25 | 111 |
| Deep Q-Learning | X | X |

The Hamilton Agent performed the best of the agents tested taking an average of 111 steps to achieve an average length of 25. The second performing agent was the Manhattan agent achieving an average length of 10 while taking 80 steps. Next, the Longest Path achieved 8 length while taking 72 steps. The lowest performing agent was A\* agent, achieving 4 length with an average of 35 steps.

***Discussion***

We constructed this project to be a modular plug and play design where any number of ai agents could be created and given from the command line, and their performance metrics measured and compared. Our intention was to build several ai agents, measure their performance, and have them compete for who could obtain the highest score with the lowest cost.

We ran into several issues while implementing and running our ai agents; given more time, we would make several improvements. First, we chose PyGame as the framework to build and run Snake, and this seems to come with some limitations. There are times where certain key events in rapid succession seem to cause problems for the snake and can even lead to a game over. We also noticed flukes with the collision detection with certain key events in rapid succession. We suspect that a different game library might have yielded us different results.

Another point of improvement is with the A\* algorithm. It performs well, however its heuristic is leading it to choose a shortest path between the snakehead and food. In certain games, this is a useful technique, however as the snake grows larger, it often becomes more helpful to pick longer and longer paths to the food to avoid your tail. This is where the idea of a longest path agent came into play; if a snake can choose a longer or longest path to the food, it could effectively maximize the space on the board and avoid its tail. The attempt of longest path agent was to examine paths and neighbors, and if space allowed, successively expand paths between points until a sufficiently long path is returned. The implementation is close, but not quite right; given more time this technique should prove valuable. The last algorithm was an attempt at a Hamilton search agent, which attempts to make a cycle of the board and eat the food along the way, raising its score while also avoiding its tail.

***References***

1. “Snake AI in Pygame.” *Pythonspot*, 21 Apr. 2015, pythonspot.com/snake-ai-in-pygame/.
2. Wei Zhepei, Wang Di, Ming Zhang, Tan Ah-Hwee, 10 Jul. 2018 *Autonomous Agnets in Snake Game via Deep Reinforcement Learning,* Researchgate, 2018 IEEE International Conference on Agents (ICA)
3. Binggeser, Peter. “Designing AI: Solving Snake with Evolution.” *Medium*, Becoming Human: Artificial Intelligence Magazine, 20 Feb. 2020, becominghuman.ai/designing-ai-solving-snake-with-evolution-f3dd6a9da867.
4. Clark, Thomas Hikaru. “Training a Snake Game AI: A Literature Review.” *Medium*, Towards Data Science, 3 Feb. 2021, towardsdatascience.com/training-a-snake-game-ai-aliterature-review-1cdddcd1862f.
5. lusob. “Snake-v0 (Experimental) (by @Lusob).” *OpenAI Gym*,gym.openai.com/envs/Snake-v0/.
6. Says:, Pongsarun. “Snake Game in Python: Snake Game Program Using Pygame.” *Edureka*, 25 Nov. 2020, www.edureka.co/blog/snake-game-with-pygame/.
7. Guo, Xiaoxiao, Singh, Satinder, Lee, Honglak, Lewis, Richard L, and Wang, Xiaoshi. Deep Learn-
8. ing for Real-Time Atari Game Play Using Ofﬂine Monte-Carlo Tree Search Planning. In Ghahra-
9. mani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds.), Advances in
10. Neural Information Processing Systems 27, pp. 3338–3346. Curran Associates, Inc., 2014
11. Guo, Xiaoxiao, Singh, Satinder, Lee, Honglak, Lewis, Richard L, and Wang, Xiaoshi. Deep Learn-
12. ing for Real-Time Atari Game Play Using Ofﬂine Monte-Carlo Tree Search Planning. In Ghahra-
13. mani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds.), Advances in
14. Neural Information Processing Systems 27, pp. 3338–3346. Curran Associates, Inc., 2014
15. Guo, Xiaoxiao, Singh, Satinder, Lee, Honglak, Lewis, Richard L, and Wang, Xiaoshi. Deep Learn-
16. ing for Real-Time Atari Game Play Using Ofﬂine Monte-Carlo Tree Search Planning. In Ghahra-
17. mani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds.), Advances in
18. Neural Information Processing Systems 27, pp. 3338–3346. Curran Associates, Inc., 2014
19. Guo, Xiaoxiao, Singh, Satinder, Lee, Honglak, Lewis, Richard L, and Wang, Xiaoshi. Deep Learn-
20. ing for Real-Time Atari Game Play Using Ofﬂine Monte-Carlo Tree Search Planning. In Ghahra-
21. mani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds.), Advances in
22. Neural Information Processing Systems 27, pp. 3338–3346. Curran Associates, Inc., 2014
23. Guo, Xiaoxiao, Singh, Satinder, Lee, Honglak, Lewis, Richard L, and Wang, Xiaoshi. Deep Learn-
24. ing for Real-Time Atari Game Play Using Ofﬂine Monte-Carlo Tree Search Planning. In Ghahra-
25. mani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds.), Advances in
26. Neural Information Processing Systems 27, pp. 3338–3346. Curran Associates, Inc., 20