Madeline's Policy Climb: Climbing Mt. Celeste

Dhrumil Patel
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
d396pate@uwaterloo.ca

***Abstract*—This work analyzes the effectiveness of NEAT and Deep Q-Learning to train agents to play Celeste, a 2D "platformer" with a heavy emphasis on extremely precise movements known to be challenging for humans. Our results demonstrate that trained agents can reasonably complete the starting level of Celeste.**

***Index Terms*—Reinforcement Learning, NeuroEvolution of Augmenting Topologies, Deep Q-Learning, Celeste, Gaming**

I. Introduction

Celeste is a popular 2D platformer game known to be challenging for humans. The game features a player (Madeline) that can move through any combination of mechanics: running left and right, jumping, climbing and dashing. Completing the game's levels require precise actions that combine the given mechanics to form new ones like mid-air dashes, wall jumps and "wave-dashing", a momentum-based traversal technique with precise timing and spatial accuracy. The many obstacles and rapid sequences of inputs make it a suitable challenge for evaluating reinforcement learning agents.

Creating reinforcement learning (RL) agents to play platformers like Celeste represents a compelling application domain at the intersection of precise control, real-time decision-making, and visual perception. These games demand precise actions in dynamic environments filled with obstacles, tight timing windows, and complex traversal mechanics. Platformers like Celeste challenge agents to develop robust policies that can adapt to nuanced terrain layouts and unforgiving failure conditions, often with sparse or delayed rewards. In some scenarios, agents may need to learn how to backtrack away from a linear path to the goal, requiring a more robust policy.

II. Previous Techniques

RL agents have been applied to games that require moving single and multiple agents simultaneously in games like DOTA2 [1], Atari [3], Super Mario [5], and even Celeste [6] (the game at the focus of this study). RL has also been applied for games that require complex planning such as Go [4], Chess [7] and StarCraft [8]. From previous successes, it appears that RL agents are capable of tackling Celeste, which is largely about moving a single agent through an environment according to some determined plan. The previously seen work for Celeste, along with the agent trained for Super Mario, a similar 2d platformer, see success with the NEAT algorithm [2]. This work uses the NEAT algorithm with different inputs. We note that seeing success with the NEAT algorithm using different inputs than the previous work highlights the robustness of the NEAT algorithm.

Deep Q-Learning has been applied to a number of games, namely the Atari games [3] as a method for learning an effective policy directly from raw images. Deep Q-Learning has also shown resilience to sparse rewards and high exploration complexity (some levels of Celeste require backtracking) in platformers like Sonic the Hedgehog [9]. Thus, there is good evidence to suggest that Deep Q-Learning may perform well in Celeste.

A. NEAT vs. Deep Q-Learning

NEAT evolves neural architectures from provided inputs and outputs, which makes it well-suited for finding simple representations that enable the agent to optimize a given fitness function. However, the evolution process evolves single neuron connections at a time, which implies that the algorithm may require many generations to optimize a complex fitness function that requires a complex network. In contrast, Deep Q-Learning updates all of its model parameters for every batch of training episodes, which suggests that it may converge on an effective policy earlier. However, NEAT can evolve complex architectures purely from the environment, but the model architecture in Deep Q-Learning to model the policy is fixed. We train agents using both algorithms and compare their differences.
[1]

III. Methodology

A. Reward Function

In both NEAT and Deep Q-learning, we need to create a reward function to characterize desired agent performance. We let one episode comprise the agent's performance until death, which can be caused by mechanisms in the game, along with 2 mechanisms proposed in this work to hasten the training process.

1) The player dies from an in-game mechanism (colliding with obstacles like spikes, falling through the bottom of the level, etc.)
2) The player's x position doesn't change by at least 0.25 units for 5 seconds

[1]Code available at https://github.com/dhrumilp15/madelines-policy-climb

3) The distance between the player's current and previous position doesn't change by at least 0.25 units for 5 seconds
4) The player has spent 20 seconds in the current level, but has not completed it

We add the second and third conditions to quickly end episodes in which the agent presses inputs that don't result in movement (e.g. pressing the left and right inputs at the same time). Although there may be scenarios in the game where the player shouldn't move (e.g. while waiting for obstacles in the environment to move out of the way), the agent would often simply not move at all, especially at the start of training. The last condition acts as a catch-all condition for scenarios in which the agent repeats the same moves indefinitely (i.e. jumping between the same two obstacles) without making meaningful progress in the level.

The reward function is then the difference between the number of completed levels and the distance to the end of the level before the agent died:

$$\text{fitness} = \text{num}_{\text{levels}} - (\text{distance to end})_{\text{level}} \quad (1)$$

We measure the distance to the end of the level as the distance from the player's last position to the top-right corner of a level. Although the player need not pass the top-right corner of a level to complete it, the true position that the player must pass is generally towards the top-right corner of a level, so this is a reasonable heuristic.

B. Caveats to the Reward Function

The initial reward function also used a time penalty and a penalty for not moving to encourage the agent to complete the levels as fast as possible. However, there were edge-cases in which an agent that dies early in the level while moving could obtain a higher fitness than an agent that completes more of the level but dies from not moving. Due to the difficulty of tuning the balance between penalties, we removed the penalties. Remarkably, we found that the agent still learned to complete levels with the simpler reward function.

C. Future efforts for the reward function

Possible avenues for improving the reward function may include exploring:

1) Giving the agent a reward for completing the level quickly
2) Improving the accuracy of the distance to the "goal" position of a level (coordinates such that, if a player were to pass these coordinates, the player would complete the level)
3) Obtaining a per-level estimate for the amount of time the agent should need to complete it

D. NEAT

We use the NEAT algorithm with 5 genomes and sigmoid activations. The agent receives inputs corresponding to the player's position (2 floats), speed (2 floats), the distances to obstacles in the 8 cardinal and inter-cardinal directions, along with the obstacle types (16 integers), stamina (1 float), distance to the goal (1 float), whether the player is on the ground, swimming, climbing, and still has their dash (4 booleans) for a total of 26 inputs. The agent can press any combination of the game input keys (up, down, left, right, z, x, c) and thus has 7 possible outputs. In this work, we observe that an agent trained using the NEAT algorithm under this configuration is able to complete the starting level of Celeste within an average of 16 generations.

E. Deep Q-Learning

We use a Deep Q-Learning Network with 3 convolutional layers followed by 2 linear layers to predict the best move among the 2^7 possible options (any combination of the 7 input keys can be pressed). The agent receives the reward obtained from 1 at each timestep. We observe that the DQN model can learn to complete the first level of Celeste after 1000 episodes.

IV. Conclusion

This paper evaluates the efficacy of NEAT and Deep Q-Learning to train agents to play Celeste. Using a simple reward function, agents trained using either method can complete levels of Celeste. We also observe that the agent using NEAT learns to complete levels faster than the one trained with Deep Q-Learning. Finally, we suggest that a more accurate reward function may result in faster training.

References

[1] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," arXiv preprint arXiv:1912.06680, 2019. [Online]. Available: https://arxiv.org/abs/1912.06680

[2] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002. [Online]. Available: http://nn.cs.utexas.edu/?stanley:ec02

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[4] Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016). https://doi.org/10.1038/nature16961

[5] S. Bling, "MarI/O," Available: https://gist.github.com/ d12frosted/ 7471e2123f10485d96bb

[6] hdrien0, "Celeste-NEAT," Available: https://github.com/hdrien0/Celeste-NEAT

[7] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint* arXiv:1712.01815, 2017. [Online]. Available: https://arxiv.org/abs/1712.01815

[8] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, and R. Tsing, "StarCraft II: A new challenge for reinforcement learning," *arXiv preprint* arXiv:1708.04782, 2017. [Online]. Available: https://arxiv.org/abs/1708.04782

[9] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman, "Gotta learn fast: A new benchmark for generalization in RL," *arXiv preprint* arXiv:1804.03720, 2018. [Online]. Available: https://arxiv.org/abs/1804.03720