Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2025)

Learning Finite State Machines from Gameplay Video

Dave Goel, Matthew Guzdial

Computing Science Department, Alberta Machine Intelligence Institute
University of Alberta
dgoel1@ualberta.ca, guzdial@ualberta.ca

Abstract

There has been an ongoing trend of pitching neural networks as playable game engines. However, from a game developer's perspective, world models are not very practical since they cannot make changes like in a traditional game engine. Our project learns a model of pacman from gameplay video that can be played in pygame. The learnt model captures the original behaviour of the entities in the majority of the instances.

Overview

With the advent of world models like Genie 2 and GameNGen, there has been an ongoing trend of pitching neural networks as playable game engines (Parker-Holder et al. 2024; Valevski et al. 2025). However, there are a few drawbacks to this approach. From a game developer's perspective, world models are not very practical since they cannot make changes like in a traditional game engine. Additionally, the models do not have a world state and memory, thus they have a tendency to hallucinate. We believe that learning a model that allows developers to easily change the underlying code would be more useful to them.

Our project learns a model of Pac-Man from gameplay video that can be played in pygame. This is done with the help of the Marionette architecture (Smirnov et al. 2021). Marionette is a self supervised model that learns a sprite dictionary from frames of a video and then decomposes each frame into a grid representation where each grid cell contains a sprite from the learned sprite dictionary.

We use a simplified version of Pac-Man for our purposes. The dimensions of the map were reduced to 8x8 and we removed all the ghosts except Blinky from the game. We train the Marionette model to learn the sprite dictionary and process the grid representations for each frame of Pac-Man. Once we have the grids, we learn code for each entity (or sprite) to replicate their behavior in the gameplay video. We do this by scanning through the sequence of grids and learn programs for each of the entities such that their local behaviour matches the observed behaviour. For each frame we try to predict the next frame in the sequence according to the learned programs, using the actual next frame as the ground truth.

Each learned program is a series of if statements. In order to make the space of programs searchable, we created our own Domain Specific Language. This language has the following conditions: exists_in_map() to check if some other entity exists in that frame and is_neighboring() to check if an entity is next to some other entity. Additionally it has the following actions: follow_entity(), change_to_entity(), follow_target_location(). We search over the program space for each entity to minimize the difference between the predicted frame and the ground truth. The full details behind our approach are explained in our prior work (Goel, Guzdial, and Sarkar 2025).

In order to make the demo playable, we convert the grids back into frames inside pygame. While the player can control the actions of the Pac-Man, the actions of the entities (Blinky, power pellets, pellets) are dictated by the learned programs.

The learnt code captured the original behaviour of the entities in the majority of the instances. For example, Blinky learned to follow Pac-Man and transform into a frightened state when Pac-Man ate a power pellet. However, there were some behaviours that our DSL could not represent. For example, in the frightened state, original code makes the entity move around to random points. However this is not possible with the current DSL, thus when entered in this state, the ghost just becomes stationary. Additionally, the DSL treats neighbouring entities as collisions, so a Pac-Man can eat a pellet by just being next to it.

We believe that learning models of a game where developers can read and change the code is more valuable than world models to game production.

Acknowledgements

This work was funded by the Canada CIFAR AI Chairs Program. We acknowledge the support of the Alberta Machine Intelligence Institute (Amii).

References

Goel, D.; Guzdial, M.; and Sarkar, A. 2025. Finite Automata Extraction: Low-data World Model Learning as Programs from Gameplay Video. arXiv:2508.11836.

Parker-Holder, J.; Ball, P.; Bruce, J.; Dasagi, V.; Holsheimer, K.; Kaplanis, C.; Moufarek, A.; Scully, G.; Shar, J.; Shi, J.;

Spencer, S.; Yung, J.; Dennis, M.; Kenjeyev, S.; Long, S.; Mnih, V.; Chan, H.; Gazeau, M.; Li, B.; Pardo, F.; Wang, L.; Zhang, L.; Besse, F.; Harley, T.; Mitenkova, A.; Wang, J.; Clune, J.; Hassabis, D.; Hadsell, R.; Bolton, A.; Singh, S.; and Rocktäschel, T. 2024. Genie 2: A Large-Scale Foundation World Model.

Smirnov, D.; Gharbi, M.; Fisher, M.; Guizilini, V.; Efros, A. A.; and Solomon, J. 2021. MarioNette: self-supervised sprite learning. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781713845393.

Valevski, D.; Leviathan, Y.; Arar, M.; and Fruchter, S. 2025. Diffusion Models Are Real-Time Game Engines. In *The Thirteenth International Conference on Learning Representations*.