Understanding games

Tom Rochette <tom.rochette@coreteks.org>

December 21, 2025 — 77e1b28a

0.1 Context

Games are complex, self-contained environment from which an AGI can learn. Games most often have a user interface and a set of rules, which are generally easily understood by their players. However, is it possible for simple vision algorithms (and maybe more) to understand and extract the mechanics of a game?

I'd like to explore 7 types of games:

- Real Time Strategy (Starcraft)
- First Person Shooter (Counter-Strike, Halo, Battlefield, CoD)
- Simulation/Sandbox (Minecraft, Sims)
- (MMO)RPG (WoW, Diablo, Dark Souls)
- MOBA (LoL, Dota 2)
- Platformers (Super Mario, Sonic, Mega Man)
- Action/Adventure (Zelda)

0.2 Learned in this study

0.3 Things to explore

1 Overview

2 In General

Games most likeliest neural network architectures would be CNN for vision of the game pixels image with RNN for audio cues (if any is available). One of the core goals of the network is to simplify what is perceived so that later on stages deal with more abstract concepts and not pixels. We would also like for the neural network to have the ability to understand locality, that is, things that move together (or do not move) are generally part of the same thing.

In most games the human player will have to develop a strategy. In order to do so, he will have to understand the mechanics of the game (formulate a model).

3 Real Time Strategy

Real time strategy (RTS) games requires the player to make numerous decisions per minutes. In most cases, the player is in control of numerous units which they may control directly. In most cases, once a command has been given to a unit, the game takes over and accomplish a number of things, such as path finding, fog of war updating, assigning which units to attack or which patch of resources to mine and where to return the mined resources, etc.

RTS games are generally discussed at two levels: macro and micro. Macromanagement is generally about high-level decisions such as strategy and tactic, while micromanagement is about minute control of each unit on the field. A good player will excel in both types of management.

In a 1v1 scenario, each player's goal is to acquire as much information as possible about their adversary in order to make the most informed decisions: when it's safe to gather resources, when an attack is coming, what kind of units the enemy player is acquiring, etc. In most games, units types have strengths and weaknesses, so it is crucial to understand and harness those when making decisions. Selecting the wrong type of units to counter an incoming attack could result in the loss of a game for the player making this mistake.

Information acquisition is critical if a player wants to be proactive in his approach to the game. Being reactive will generally lead to poor results compared to a proactive approach.

3.1 What's important

- Acquiring resources
- Know what the enemy is doing
- Counter what the enemy is doing
- Using the enemy weaknesses against him
- Attacking/Targeting the right enemy units (priority, weakness)

3.2 Things an AGI needs to do

- Select units
- Decide what order to give
  - Gathering
  - Moving
  - Attacking
  - Stopping
  - Building
- Issue orders
  - Define location
- Move the camera
- Compute and evaluate its current state
- Determine available actions (given the UI)

3.3 Data structures

- List of units and buildings
- Image/Matrix representation of the location of buildings
- Regions of interest
  - Resources
  - Base location
  - Buildings locations
  - Enemy location
  - Armies locations

4 First Person Shooter

5 Simulation/Sandbox

6 (MMO)RPG

7 MOBA

In this type of game, the player controls a single hero and plays with multiple teammates as well as multiple enemy players. The game is about strategic control of map objectives (as a team) as well as general mechanics

(as a player). The game completes when the structures of one of the two teams are destroyed (or a specific structure is destroyed).

Decision is an important ability in this type of game as the enemy will punish you when you make mistakes. Punishment takes the form of lost gold, lost health or the loss of life (and gameplay time). The players are rewarded for killing NPC as well as enemy heroes.

Most games of this genre separate a game in three phases: laning, midgame, and endgame. Laning is about warming up your mechanics. Midgame is about attempting to destabilize the enemy team. Endgame is about team fights and pushing for objectives in order to win the game.

8 Platformers

Platformers are an interesting genre because they can easily be seen as a path through a level that needs to be optimized. If we take a game such as Super Mario Bros., we only have one real constraint: we must finish the level before the given 300 seconds (5 minutes). Using this single constraint though makes it hard for our AGI to learn as there is no clear gradient from the initial state to the final state (you would have to wait 5 minutes for an episode to complete and have the agent negatively rewarded). Thus, we need to add a couple more constraints/goals.

In order to make the levels more challenging, other constraints such as not dying to enemies or falling into holes are part of the game.

If we look at the game from a reward/punishment point of view, we would reward the AGI for going as far as possible in the level within the least amount of time. Every time the player the AGI controls ends up falling in a hole and dying or touching an enemy and dying or losing its tall Mario aspect, punishment would be applied as we want to avoid these circumstances.

By providing the AGI with a couple of do/do not, it is possible for us to let it learn on its own through reinforcement learning.

In my article Mari/o, I write on the topic of Neuroevolution of augmenting topologies[1]. NEAT is a genetic algorithm which goal is to generate and improve artificial neural networks in order to best respond to some selection criteria (generally a single evaluation metric).

The algorithm works similarly to how a reinforcement algorithm would work, in the sense that it will promote genes/neural networks which better suit the goal given to it. In this particular exercise, the only reward that was given was based on how far Mario had moved to the right (toward the end of the level goal).

The game was basically replaced by a 13x13 grid where there exist 2 types of blocks: white to represent the ground and black to represent enemies or sprites. Thus, the NEAT algorithm was learning to play a dumbed down version of the game. And yet, it took it many hours to get to the point it would be able to complete the level without dying and its performance was still average at best.

On the surface, the NEAT algorithm appears to learn, however I'd suggest that it is far from that. Instead, what it is doing is trying numerous cases that are part of the search space, supposedly attempting to find a balance between the fitness of evolved solutions and their diversity[2] which should lead to a better action model for the same number of attempts compared to a random action agent.

One of the major issues with this approach is that it does not "learn" to infer that something is bad. For instance, if a neural network is configured such that if it sees a hole in front of him and that the action should be to walk directly into that hole, and that the agent is punished for that event, it should attempt to propagate this information backward, for instance by putting more emphasis on recently occurred decisions such as "walk right if hole in front" being a negative/bad decision.

Platformer games likeliest neural network architectures would be CNN for vision of the game pixels image with RNN for audio cues (if any is available). For games like Mario Bros, it seems logical to want to abstract

[1] https://en.wikipedia.org/wiki/Neuroevolution_of_augmenting_topologies
[2] https://en.wikipedia.org/wiki/Neuroevolution_of_augmenting_topologies

the moving sprites to their rectangle equivalent in order to reduce computational complexity. In the same way, it is likely that most of the texture in the game is irrelevant, which would prompt the development of a system in which elements can either be present or absent (a wall vs clouds).

9 Action/Adventure

10 See also