Reinforcement Learning Agent for a Flight Simulation Video Game

Victor Nonea, Radu Iacob, Traian Rebedea
University Politehnica of Bucharest
Splaiul Independenței 313, 060042, Bucharest, Romania
victor.nonea@gmail.com, {radu.iacob, traian.rebedea}@upb.ro

ABSTRACT
This paper is a case study analysis of the viability of using machine learning methods for skilled NPC agents in production level video games and how they compare to their hand-coded counterparts. The implementation and experiments were made in a simple OpenGL game about flying an aircraft through a set of checkpoints without crashing. Our conclusion is that current machine learning methods are not feasible for building the NPC agent, because, while they simplify the agent's design, they exponentially complicate the testing and debugging processes without offering an improved ability for the NPC.

Author Keywords
Reinforcement learning; REINFORCE algorithm; video game; non-playable character; NPC; flight simulation.

ACM Classification Keywords
I.2.6: Artificial Intelligence: Learning
I.3.8: Computer Graphics: Applications

DOI: 10.37789/rochi.2021.1.1.15

INTRODUCTION

A common issue in the design of video games is how to develop the AI of competitive and cooperative non-playable characters (NPCs). These characters, as opposed to most NPCs, have to complete complex tasks and perform said tasks on par with human players. Moreover, some of them will have to exhibit realistically human behaviour, which is to say, act and play similar to how a human would in that situation.

The purpose of this project is to determine the viability of using machine learning methods for production level skilled NPCs, and, more specifically, its advantages and disadvantages to traditional manually programmed agents.

For the purposes of this paper, we will define skilled agents as NPCs that interact directly with the player, competitively or cooperatively, such that their behavior may determine or significantly impact the win/lose state of the game, or of a particular level of the game.

The quality of the skilled agents is very often a make-or-break feature of the game. This is because they are intimately entangled to the core mechanics of the game. In a shooter you need enemies that know how to shoot, take cover, and move organically around the map. In a racing game you need competitors that know how to take turns efficiently, how to get around obstacles and how to recover from crashes. And wherever agents represent humanoid characters, they need to accomplish these tasks in a credibly human way.

Background

For NPC agents, there seems to be a strong preference for statically defined behaviour, represented by finite state machines or behaviour trees [1].

Dynamic behaviour is often still based on these static models, through utility-based AI, which create a point system for a set of possible actions, based on scripted rules, and pick the best action to take [2]. In strategy games, where multiple game steps can be simulated ahead of time, Monte Carlo tree search is preferred by developers [1].

It is worth noting that in video game development, generally the primary concern is not the competence of the NPC agents, but the appearance of competence. This is especially true about enemy NPCs, which are ultimately supposed to be defeated by the human player, but still impress the player with their aptitude along the way. This can be done through dynamic difficulty adjustment, in which you assess how well the player is doing, and if they are doing better than expected, or if other evaluation methods indicate they may be bored, boost the opposing NPC performance, or alter level generating parameters to make the environment more hostile or difficult to traverse, and vice-versa if the player is doing poorly.

The boost in NPC performance does not have to be related to the agent's intelligence, in many situations it could just as well be an artificial advantage. For instance, in a racing game if the opposing NPC racer falls too far behind they may be given more speed and perfect grip until they catch up, or in a strategy game an opposing NPC nation may be given free resources.

The subject of using machine learning to improve NPC behavior is at an awkward intersection between academia, open-source technologies and business interests. Because of that, there is little scientific coverage of this topic specifically. For instance, Unity Technologies created the

Unity ML-Agents Toolkit, an open-source project which integrates PyTorch, the machine learning framework, with the Unity game engine. It is stated to be "beneficial for both game developers and AI researchers" and yet, while it does provide tutorials and documentation for developers, the only featured scientific paper discussing the toolkit presents it exclusively as a *general platform* for AI research, completely disregarding its game development potential [5].

Electronic Arts uses machine learning agents for playtesting as well as NPCs (which they refer to as *game-playing AI*) [6]. They opt to use an engineered lower-dimensional representation of the game state, instead of raw video frames, to allow for smaller and more time efficient models which are reusable across different iterations of the game, and to have more control over what information the agent has access to. They also prefer to use simpler algorithms when the state space allows. They demonstrate in one of their examples how A* with a simple hand-tuned heuristic outperforms a utility-based policy found through an evolution strategy. For more complex tasks in their study, they employed DQN [7], PPO [8] and Rainbow [9]. For their use cases and setup, DQN achieved its asymptotic behaviour faster than the state-of-the-art Rainbow approach. All of this is to say that current academic approaches and assumptions about machine learning may not be appropriate for devising skilled NPCs to meet the needs of the video-game software industry.

ENVIRONMENT

The environment used for these experiments is a video game about a plane flying over the ocean and attempting to reach several targets (called markers) without crashing.

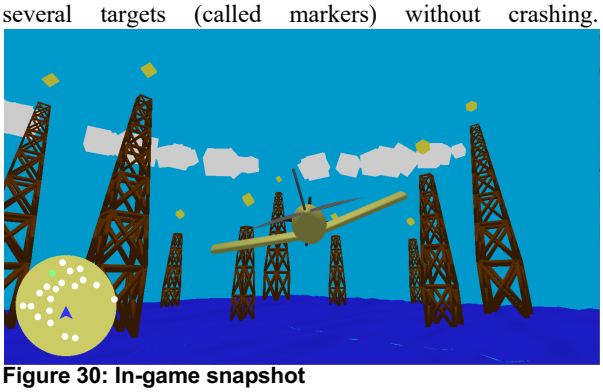

Figure 30: In-game snapshot

As shown in Figure 1, the play area consists of a patch of ocean, traversable by the plane in about 5 seconds from side to side at full speed. It contains 20 towers randomly distributed across the level (except for a safe zone around the plane), each with a random height and a checkpoint, called a marker, on top of it. In the base game, each level has a different random distribution of towers across the scene. For the purposes of training and evaluation, a new level is generated every time the agent reaches a terminal state, i.e., when it finishes the current level or crashes.

The objective is for the plane to reach all markers, in any order, without crashing into the towers or the ocean.

The game has a simple but moderately realistic physics model. The plane's movement is represented by its linear velocity and its angular velocity across its principal axes. There are 3 types of passive physical forces:

- Gravity, which acts as a constant downward acceleration on the plane.
- Drag, which acts as an exponential deceleration relative to the plane's linear and angular velocities.
- Aerodynamic alignment, which pulls the plane's body toward its linear speed direction.

The player controls the plane by acting on it with linear and angular forces. There are 4 control dimensions:

- Thrust, for linear acceleration.
- Pitch, yaw, and roll for angular acceleration.

The angular controls correspond to the aircraft's principal axes, see figure 2.

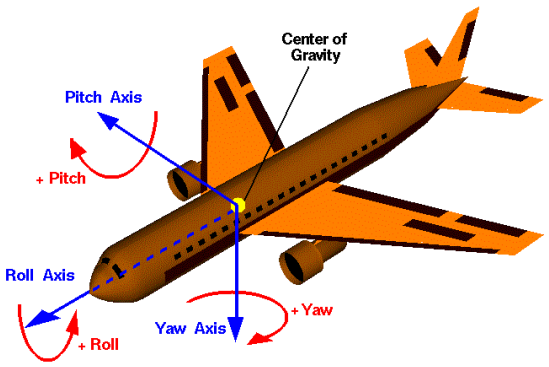

Figure 31: Aircraft principal axes[5]

The plane may only crash into one of the towers, or the ocean. For collisions, the plane and towers use simplified bounding boxes: a sphere for the plane and conical frustums for the towers. The level immediately ends if the bounding box of the plane intersects with the bounding box of one of the towers, or if the plane gets below sea level.

HANDCRAFTED RULE-BASED AGENT

The idea of the handcrafted solution is to individually control the aircraft's altitude, horizontal direction, and steadiness. By steadiness we are referring to the aircraft's alignment to

[5] © https://www.grc.nasa.gov/www/k-12/airplane/rotations.html Last accessed: 25.06.2021

the scene's horizontal plane. The agent relies on the following assumptions:

- The plane's altitude should be equal to that of its target as soon as possible.
- The agent should always yaw toward the direction of the target.
- The plane should be as level as possible, without inhibiting altitude adjustments.
- The agent should maximize thrust.

Based on these assumptions, the agent uses modules for: altitude control, horizontal targeting, and steadiness.

The main advantage of this design is that each module should achieve its function independently, and each can be debugged individually. This pattern of building simpler modules and linking them together is highly regarded in the games industry for AI implementation [1].

REINFORCE AGENT

For the machine learning agent, we used TensorFlow's REINFORCE agent, which is an implementation of the REINFORCE algorithm [3].

The REINFORCE algorithm is the simplest variant of a policy gradient method, which is an algorithm that models the optimal policy directly and updates it through gradient descent. The REINFORCE algorithm approximates the gradient of the objective function through sampling alone, and consists of the following loop [4]:

1. Sample trajectories using the existing policy.
2. Approximate the gradient of the objective function using the rewards of the sampled trajectories
3. Update the policy with a step of gradient descent: $\theta \leftarrow \theta + \alpha \cdot \nabla_\theta J(\theta)$

Observation format

The observations are made relative to the agent's current position and orientation, and capture the following:

- The target's relative position and direction
- The agent's linear and angular speed
- The direction of gravity

Action format

As previously explained, the aircraft has 4 controls: one for overall thrust (longitudinal acceleration), and three for torque (rotational force) on the aircraft's principal axes: pitch, yaw, and roll.

The action format of the agent mimics the input format for human controls: 2 binary values for each of the control dimensions, for a total of 8.

Policy model

The model used to represent the policy is a shallow neural network with 2 hidden layers of 100 and 50 activation cells respectively. The hidden layers use ReLU activation.

Reward model

The simplest way to express progress in this scenario is through the remaining distance to the target at each timestep:

$$reward = \frac{\max(0, best\ distance - current\ distance)}{initial\ distance}$$

During the initial testing phase we noticed on multiple instantiations that the agent would eventually start to roll excessively, and because the observations are relative to the plane's momentary orientation we considered this might make them too difficult for the agent to process. That is why we have added a reward for keeping the plane steady each timestep. The reward is proportional to the angle between the aircraft's up vector and the scene's down vector, and can be written as:

$$\boldsymbol{reward = \frac{\langle aircraft\ up, scene\ down \rangle \cdot dt}{180° \cdot expected\ episode\ duration}}$$

The total reward used was a weighed average between the distance-to-target reward and the steadiness reward.

Training methodology

We defined a training episode as the time interval of the agent traveling between two consecutive markers, the strategy for picking the targets being static. All evaluated models and setup variants were trained 500 episodes.

The best performing agent was one with 20% steadiness reward and fully stochastic training. This means that each training iteration only considered the last training episode. It reached around 80% training gain (see Figure 3).

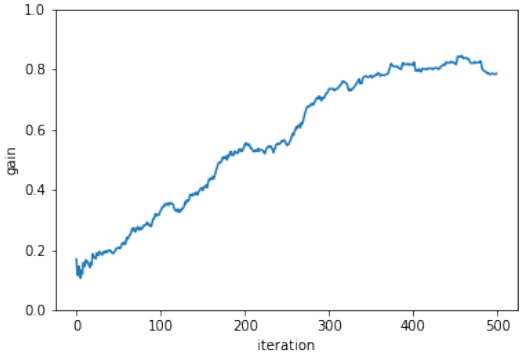

Figure 32: Best performing agent, trained stochastically over 500 episodes, reaches about 80% of maximum expected gain.

The experimental variants had slightly different reward models or train iteration conventions:

- Baseline:
  - 80% distance-to-target, 20% steadiness reward model.
  - 5 episodes train iterations, with rewards normalized across iteration.
- Stochastic training (same reward model).
- Batched unnormalized training.
- 100% distance-to-target reward model.

All variants across most of their instantiations got between 20% and 60% of the maximum expected gain at the end of their training sessions. It is unclear whether the stochastic training variant actually has a technical advantage, or it got the best result from a lucky instantiation.

EVALUATION

Metrics

We will compare the handcrafted and ML solutions based on the following criteria:

- Normal performance: numerical measurements of the bots' effectiveness in the normal game context
  - Average distance travelled: the average proportion of the trip the agent manages to travel to its next target before ending the round (for a usable agent should be near 100%).
  - Catch rate: the proportion of times the agent successfully reaches its next marker.
- Human believability: the extent to which the bot acts similarly to a human controller. This is a subjective observation, but it is relevant from a game development standpoint.
- Robustness / Generality: whether the bot is usable with altered environments. It is also a subjective observation.

Numerical performance

The numerical evaluation is made over 100 training episodes. The best performing agent by a significant margin is the handcrafted bot, with a 70% catch rate and 83.46% average distance traveled. (Table 1)

The best ML solution, produced with a stochastic training setup, gets a similar average distance traveled, of 82.28%, however it does far worse at actually reaching the markers, having a 14% catch rate.

The baseline ML learning solution, 20% steadiness reward model with 5 episode batched training gets a 56.43% average distance traveled, but a 0% catch rate (0 catches over 100 episodes).

We also included a completely untrained ML agent to have as a reference for expected average distance traveled with completely random movement. It is not zero because in most initializations the plane starts angled toward the next target.

Human believability

For the handcrafted solution, a person can observe the individual discrete phases of the altitude module as well as unnatural stabilizing motions from each of the modules, but only if they are paying close enough attention, which may not be the case in practical video game experiences.

All ML solutions exhibited some form of synthetic behaviour. Most models displayed excessive rolling and even the best performing agent flies unreasonably slowly and almost always picks up its target by going too far left and then turning around.

Agent	Avg. distance to target	Catch rate
Handcrafted rule-based agent	**83.46%**	**70%**
REINFORCE agent Stochastic training	82.28%	14%
REINFORCE agent Baseline	56.43%	0%
Untrained agent	14.23%	0%

Table 1: Numerical performance

Robustness / Generality

The handcrafted design is somewhat inappropriate as a general purpose solution. It is heavily reliant on the markers and obstacles being laid out in a big, open space, without requiring any fine maneuvering, such as taking turns with combined roll and pitch rotations.

It is also nonadaptive to changes in the physics engine's parameters. Even during the development of this project we had to add a new adjustable parameter to the bot, after we had made some changes to the environment, because it had become unstable with its altitude adjustment.

That being said, it is far more robust than the ML solutions. The best performing ML agent, after its 500 training episodes would be decent enough at catching markers from below, and yet would be completely stuck when it had to pick up a marker from above.

The one thing that the ML agent did consistently better was catching initially close targets, because it would not get stuck in an orbiting loop. However, this is not that impressive considering that one of the more obvious flaws of the ML agent is that it is constantly moving too slowly, so close proximity targets are inherently easier to catch.

DISCUSSIONS

Modularity and testability of individual modules

The ML algorithm, by its nature, has to treat a multitude of functionalities together, with a single model. This means that if a particular functionality is not satisfied, nothing from the current model is salvageable.

Training nondeterminism

Practical machine learning models are non-linear, which means they can include local optima.

Reinforcement learning algorithms, in practice, often behave non-deterministically, due to their instantiations (Figure 4) or due to the complexities of the environment.

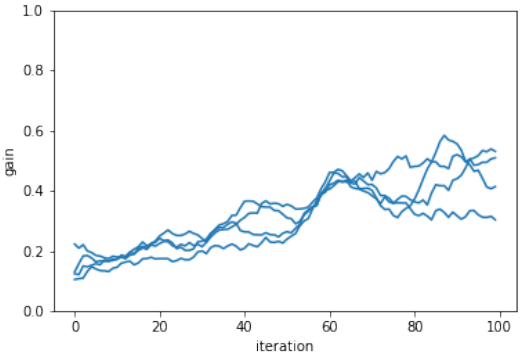

Figure 33: The same reward model and training setup with different initializations

Sometimes the machine learning setup, which is to say the algorithm, model shape, activation functions, batch size and other hyperparameters, are simply inappropriate for the given problem.

All of this put together means that it is never exactly clear when you should continue training the current agent, initialize a new agent, or change the setup altogether. During this project we have been surprised several times by how wrong our predictions over certain setups were, or by how something that we thought was explainable in an early stage of development turned out to be purely due to random behaviour.

It is not just the issue of extra time needed to test different setups or instantiations but the mental burden itself of not knowing whether a passed test is a sign that you are on the correct path.

CONCLUSION

Most issues with the ML approach collectively add up to one thing: unreasonable time investment. The machine learning agents took collectively several orders of magnitude more time to develop and test than the rule-based agent, and even so, the latter proved more reliable.

For our use case environment, and the purpose of building an agent that is skilled, humanly credible and robust for a production level video game, we would say that a handcrafted solution is more appropriate than a machine learning approach.

REFERENCES

1. Coline Molina, Maud Espié, and editor Vincent Manilève. L'art de feindre l'intelligence. https://stories.ubisoft.com/article/ ia-jeux-video-ennemis-intelligence-the-division-2/. Last accessed: 19 June 2021.
2. Dave Mark. AI Architectures: A Culinary Guide (GDMag Article). http://intrinsicalgorithm.com/IAonAI/2012/11/ ai-architectures-a-culinary-guide-gdmag-article/. Last accessed: 28 June 2021.
3. Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8(3):229–256, May 1992.
4. Sergey Levine (instructor). Policy gradients, lecture 5 of CS 285 (Fall 2020) at UC Berkeley. http://rail.eecs.berkeley.edu/deeprlcourse-fa20/static/slides/lec-5.pdf. Last accessed: 1 September 2021.
5. Juliani, Arthur & Berges, Vincent-Pierre & Vckay, Esh & Gao, Yuan & Henry, Hunter & Mattar, Marwan & Lange, Danny. (2018). Unity: A General Platform for Intelligent Agents.
6. Zhao, Yunqi & Borovikov, Igor & De Mesentier Silva, Fernando & Beirami, Ahmad & Rupert, Jason & Somers, Caedmon & Harder, Jesse & Kolen, John & Pinto, Jervis & Pourabolghasem, Reza & Pestrak, James & Chaput, Harold & Sardari, Mohsen & Lin, Long & Narravula, Sundeep & Aghdaie, Navid & Zaman, Kazi. (2020). Winning Is Not Everything: Enhancing Game Development With Intelligent Agents. IEEE Transactions on Games. PP. 1-1. 10.1109/TG.2020.2990865.
7. V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, p. 529, 2015
8. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.
9. M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," arXiv preprint arXiv:1710.02298, 2017