Using Self-Play in Pursuit of Super-Human Performance in a First-Person Game Setting

Jernej Puc, *IŠRM2*

***Abstract*—A simple environment for an adversarial first-person shooter game was defined and simulated in the Unity game engine. The single agent was intended to be trained through reinforcement learning by playing against itself using the self-play mechanism. After encountering insurmountable instabilities, the problem was reformulated, such that the agent was trained in a non-adversarial scenario, but was nonetheless able to play against itself during inference time.**

I. Introduction

SELF-PLAY is a mechanism in adversarial games, which uses the agent's current and past "selves" as opponents during training. This provides a naturally improving adversary against which an agent can gradually improve using traditional reinforcement learning (RL) algorithms.

It remains the subject of ongoing interest, particularly due to its ability to lead to "surprising", i.e. "innovative", behaviour in well-established domains. As such, it was fundamental to a number of the most high-profile results in RL, e.g. AlphaZero (DeepMind – Go, chess, shogi), AlphaStar (DeepMind – StarCraft 2), and OpenAI Five (OpenAI – Dota 2).

The question that emerges is whether the benefits of self-play can be reproduced without industry leading-level resources at one's disposal, albeit in a simpler setting Due to no particular reason, a small and simple arena for a first-person shooter (FPS) game was envisioned, in which agents could face each other one-on-one.

II. Methodology

A. Environment

The Unity game engine was used for environment simulation (involving code in C#). The model and training was first intended to be explicitly defined in Tensorflow (Python), but some restrictions arose from the use of Unity's own ML-Agents toolkit, handling synchronisation between Tensorflow and Unity processes, which is not trivial due to interactions with the experience buffer, simulation parameters etc.

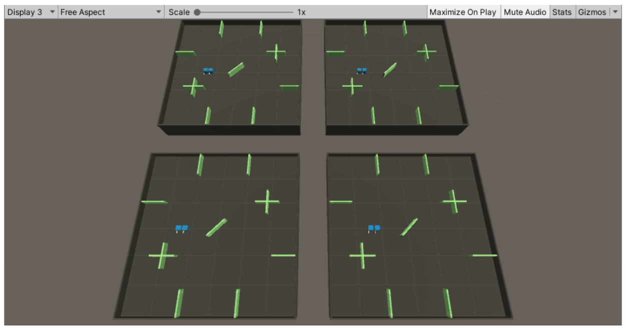
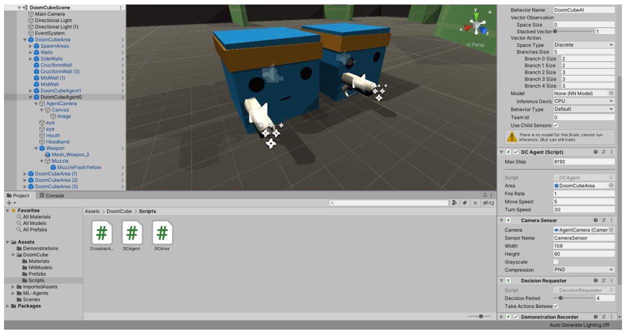


Fig. 1: Top-down view of the training environment. Multiple copies of the area allow parallel accumulation of experiences.



Behavior Name: DoomCubeAI
Vector Observation
Space Size: 0
Stacked Vector: 1
Vector Action
Space Type: Discrete
Branches Size: 5
Branch 0 Size: 2
Branch 1 Size: 2
Branch 2 Size: 3
Branch 3 Size: 3
Branch 4 Size: 3
Model: None (NN Model)
Inference Devic: CPU
Behavior Type: Default
Team Id: 0
Use Child Sensors: ✓
There is no model for this Brain; cannot run inference. (But can still train)

DC Agent (Script)
Max Step: 8192
Script: DCAgent
Area: DoomCubeArea
Fire Rate: 1
Move Speed: 5
Turn Speed: 30

Camera Sensor
Camera: AgentCamera (Camer
Sensor Name: CameraSensor
Width: 108
Height: 60
Grayscale: ☐
Compression: PNG

Decision Requester
Script: DecisionRequester
Decision Period: 4
Take Actions Betwee: ✓

Demonstration Recorder

Fig. 2: A single area instance is inhabited by two agents, who share the same behaviour.

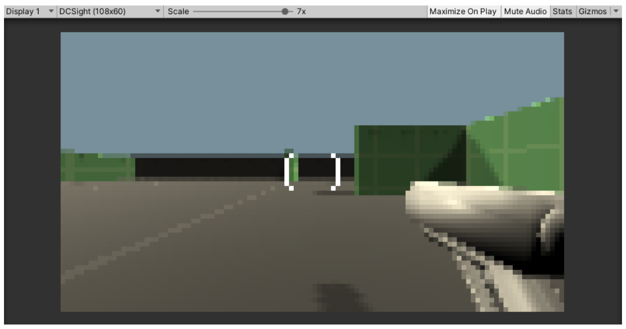


Fig. 3: The agents use only RGB images with 108x60px resolution as input. Note that the displayed cross-hair changes its colour to red if the agent points towards the other.

B. Action space

Restricting the action space to be discrete led to both faster training and inference. The discrete action space was comprised of the following actions (with human input analogues):

- move forward/backward – W/S,
- move left/right – A/D,
- rotate left/right – horizontal mouse movement,
- pull the trigger – left mouse button,
- reduce movement and rot. speed – right mouse button.

C. Architecture

The model architecture consisted of an embedding of visual information, high-level reasoning, and memory. The individual layers, in order of information flow, are the following:

1) 8x8 (stride 4) convolutional layer
2) 4x4 (stride 2) convolutional layer
3) 3x3 (stride 1) convolutional layer
4) Fully-connected layer with 256 nodes
5) Fully-connected layer with 256 nodes
6) Long short-term memory (LSTM) layer with a memory vector of size 128 and a sequence length of 64 frames

D. *Reward function*

The agent receives both per step (dense) and per episode (sparse) rewards. Such rewards assume that e.g. encouraging seeking and staying on the target will speed-up the training, but also impose a bias on the agent:

$$R(t) = \begin{cases} +\frac{1}{\text{MaxSteps}} & \text{if other agent in sight} \\ -\frac{1}{\text{MaxSteps}} & \text{if pulling the trigger} \\ +2-\frac{\text{Steps}}{\text{MaxSteps}} & \text{if victorious} \\ -2+\frac{\text{Steps}}{\text{MaxSteps}} & \text{if defeated} \end{cases}$$

E. *Training configuration*

Training was conducted on a CPU for a maximum of 2 days per run. Of the algorithms and supplemental modules supported by the toolkit, the following were used:

- *Behavioural cloning:* Pre-trains the network to mimic the actions shown in a set of demonstrations.
- *Proximal policy optimisation (PPO):* The policy gradient method proposed by OpenAI (and used in OpenAI Five itself), which serves as the main training algorithm.
- *Generative adversarial imitation learning:* Adds a small reward for behaving similar to a set of demonstrations.
- *Curiosity module:* Adds a small reward for "surprising" experiences (encourages thorough space exploration).
- *Self-play module:* Regularly swaps between the agent's opponents (and teams) and adjusts the rewards based on the agents' ELO ratings.

III. Self-play results

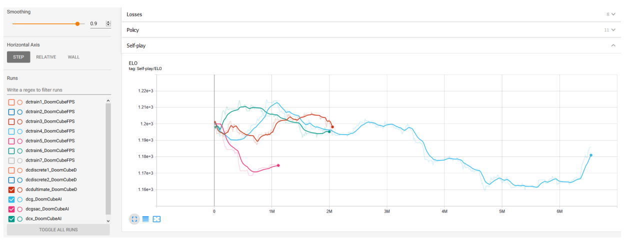


Fig. 4: ELO graph as displayed by the tensorboard.

In adversarial games, an ELO-like rating system may be better in tracking learning progress than the environment reward, as the latter depends on the skill of the agents. In a given training run, the agent's ELO rating should steadily increase, indicating that it is beating its past variations.

As evident from the figure above, self-play experiments proved unstable and failed to lead to expected behaviour. A reply from one of the creators of ML-Agents on Unity's official forum suggested that, instead of doubting the configuration, reward function, or structure of the environment, the problem itself may be too complex.

Then, depending on the problem complexity, training would either need more time before being able to be properly assessed or more careful formulation as well. Elaborating on the latter, the win condition (see the agent and shoot) is very close to the loss condition (be seen by the agent and shot). The agent would thus be rewarded and penalised for the same behaviour, rendering learning impossibly unstable.

IV. Problem reformulation

Consulting the literature, a few related examples (e.g. Arnold or ViZDoom) were successfully trained in single player scenarios, which implied that reformulating the problem might lead to at least some success. Per the reformulation, instead of facing a similarly capable opponent, a single agent was tasked with eliminating 3 stationary (dummy) agents (per episode).

Most of the training configuration and model architecture was left the same, only the self-play module was removed and the reward function adjusted as such:

$$R(t) = \frac{-1}{\text{MaxSteps}} + \begin{cases} \frac{-1}{\text{MaxSteps}} & \text{if other agent in sight} \\ \frac{+1}{\text{MaxSteps}} & \text{if pulling the trigger} \\ 1/2/3 & \text{per consecutive target hit} \end{cases}$$

A. *Results*

After successful convergence, the agent is able to consistently attain a high reward. By comparing the data based on the final 75 AI simulations with 75 recorded human demonstrations, the agent can also be roughly evaluated. The ratios of approx. 3 times more steps per episode, 97% of cumulative reward, and 4 times higher standard deviation indicate that the human "expert" still sets a higher baseline.

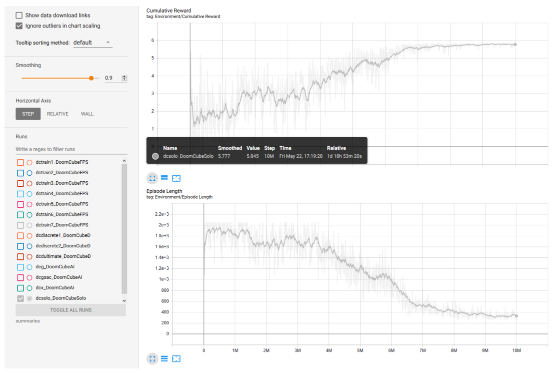


Fig. 5: The cumulative reward and episode length graphs.

TABLE I: Evaluation of AI vs human performance.

	Avg. steps per episode	Mean reward	Std. of reward
AI	329	5.78	$7.2 \cdot 10^{-2}$
Human	113	5.97	$1.8 \cdot 10^{-2}$

B. *Adversarial scenario revisited*

It is not evident that putting an agent, trained on stationary targets, against a dynamic opponent, i.e. itself, would work as intended. Nonetheless, the agent displays seeking behaviour and stays on its target until resolution. The fact that this works is a testament to the complexity of the adversarial scenario: by simply restating the problem (without changing the model architecture etc.), basic behaviour became learnable.

V. Conclusion

In the end, the "simple" FPS setting turned out to be more nuanced than first suspected. Adversarial problems and games with imperfect knowledge, in general, are not to be underestimated and such experiments indeed demand greater processing capabilities. It would be interesting to consider the state, achieved through single-player training, as a starting point for self-play and continue training towards more complex behaviours, but the adversarial scenario might need to be reconsidered in other ways as well.