|  |  |
| --- | --- |
| Immagine | Moving Picture, Audio and Data Coding by Artificial Intelligencewww.mpai.community |

|  |  |
| --- | --- |
| **N571** | 2022/02/23 |
| **Source** | Requirements (GSA) |
| **Title** | MPAI-GSA Status report  |
| **Target** | MPAI Members |

# Project status

The month was devoted to

1. Refine the SPG Digital Twin implementation inside the Online version of Pong, switching it from synchronous to asynchronous definition
2. Create a bigger dataset for Neural Networks training
3. Study trained LSTM Neural Networks behaviour inside the Online Pong to find a suitable configuration.



Figure 1 - Pong game



Figure 2 - MPAI-SPG Architecture

Legenda:

Green: new improvements

Red: TODO

Black: general status of the project

Both clients and server use Unity 3D as core game engine.

Features created so far:

* **Client**:
	+ Process C can send controller data to Process S
	+ Defined a CSV template to make the client log file
	+ Process C can obtain the ownership of one paddle sending an explicit request to the Process S.
	+ Process C can send notifies to Process S through the so-called “RPCs” (Remote Procedure Calls), to specify some actions or some sort of communication
	+ Process C implements Client-side Prediction related to his paddle.
	+ Process C implements interpolation (paddle position fixed with Lerp function)
	+ Process C implements server reconciliation (tick number for each data request)
* **Server**:
	+ Data exchange explanation between Game State Engine and game engines (Physics, Rule and Behavior)
	+ Defined a CSV template to make the server log file (game messages and game states)
	+ Game Server use the Photon architecture as space to instantiate each game
	+ Process S is a service program, running in “Batch/Headless mode” as Unity instance
	+ Process S can receive data from process C and acknowledge receipt of this data
	+ Process S instantiate both paddles and ball, then send (through RPCs) their ID to the clients so that they have a reference to those object
	+ Process S can handle the data (CD) sent by the process C in order to update the GS
	+ Process S is able to manage the physics both of the ball and the client paddles, sending the resulting data (GS) to both clients.
	+ Process S is able to synch both ball and client paddles.
	+ Better management of lag compensation
	+ Process S has
		- Game State Engine (GSE)
		- Physics Engine (PE)
		- Behaviour Engine (BE)
		- Rules Engine (RE)
	+ Game State Engine can send data to other three engines (PE/BE/RE)
	+ The three engines process the data correctly and send data to GSE
* **AI:**
	+ ML Agents developed, to simulate games in order to train the Neural Networks, using two techniques:
		- Imitation Learning
		- Reinforcement Learning
	+ Imitation learning Idea:
		- Build a Demo with human input (Teacher)
		- Define config.yaml with hyperparameters specifics (GAIL = GAN like learning)
		- Start learning. Agent acts as student and learns how to behave as similar as his Teacher. Time required is significantly shorter than other methods (in terms of minutes not hours)
		- NN produced will be the brain of our Client in Pong Game Online
	+ Integration of “ml-agents” framework inside Pong online version. This way will be possible to move paddle in an automatic way using inference
	+ Game simulation tests produce coherent data (log file) usable in any Neural Network
	+ Reinforcement Learning Idea:
		- A set of observed data is chosen (player position, opine position, ball velocity…)
		- Using its NN under development and observed data as input, the player executes an action and gets a reward. The reward is positive if it is correct in order to win the game, negative if is wrong.
	+ After multiple matches executed, paddle will be able to hit the ball
	+ Learning optimization (both Imitation and reinforcement learning) in terms of:
		- Physics accuracy
		- Hyperparameters selection
	+ Introduction of a raycasting system in order to improve Reinforcement Learning:
		- When the ball collides with a paddle, a series of rays is drawn on the game field representing ball trajectory ending on the opponent’s side of the field. This way the opponent will know where the ball will go and will been able to execute the correct action
	+ Introduction of a refined architecture (see the following chapter) that assigns “Divider” and “Composer” roles to Game State Engine AI
	+ Choice of the “Long Short-Term Memory” Neural Network Architecture to develop the 3 Simple Engines inside the MPAI-SPG Architecture (Physics Engine, Rules Engine, Behavior Engine) and we started the first training of them, using a small dataset of real game states taken from a Pong simulation
		- Made a comparison with the MLP (MultiLayer Perceptron) to analyze performances
	+ Creation of the Prediction System and application on the Pong Offline version:
		- Integration using Barracuda Framework
	+ Creation of Collector and Dispatcher for input and prediction data management
	+ Design of hardware requirements and availability at the end of December 2021 of the virtual architecture to create data and to train neural networks
	+ Received the hardware resources of Computer Science Department, University of Torino (CSD-UniTo).
	+ Collection of logs using CSD-UniTo hardware:
		- Using the trained AI Pong paddles, able to automatically play Pong, we collected around 15 millions game states
	+ Neural Networks training:
		- A first set of training experiments was made in order to find a suitable predictive architecture using the 15 millions records dataset generated at the previous step
		- The experiments were executed testing different kinds of architectures and changing network hyper parameters, like learning rate, batch size, optimizers, number of nodes inside LSTM layers
	+ Prediction System integration into Pong Online version:
		- To make inference using trained neural networks, a set of scripts representing the “refined architecture” was implemented using the Unity Barracuda Framework. In this way we were able to use Physics, Behavior and Rules Predicted engine inside the Pong Online game
		- This system is primarily used to test the trained neural networks previously described.
	+ Tests about different kinds of neural network architectures to find a setup able to correctly predict game states
		- Different hyperparameters were studied, like number of nodes, number of layers, optimizer, dropout values to cite some of them
		- Initially working with NNs with 500 or 1000 nodes for layers, we created an architecture able to make predict the game state with 50 nodes per layer
	+ Switch of the SPG prediction system implementation from synchronous to asynchronous:
		- Studying the server application behaviour, we found out some freezes and deadlocks that couldn’t permit the correct operation of the server. We found out that the Barracuda inference method takes some hundredths of a second to be completed. Switching the system into an asynchronous computation enables the inference method to be executed on another thread, avoiding graphic rendering freezes during server application execution. Note that this kind of approach executes the inference operation at a lower rate (about 12 frames/sec).
		- Studying the behaviour of the NNs created before this change, we noted that they were not very precise inside their predictions, forecasting ball’s position far from the correct one. This happened because the inference method was executed at a lower rate than the one used to collect the training data
	+ We computed an average measure of the time needed by the inference method to be executed
	+ Change inside the data collection scripts in Pong Offline to execute a game state sampling at the same rate of the inference execution calculated at the previous step
	+ Generation of a new training set from Pong Offline using this new script implementation
	+ Neural Network training that uses this new dataset
	+ Study of the Neural Networks behaviour inside Pong Online:
		- Neural Networks behave quite good with correct input data, while they are still not quite precise to execute predictions based on its old predictions as inputs
		- Neural Networks with a different number of timesteps in input were studied, noting that, increasing this quantity, the Barracuda inference method will require increasing time to be completed
* PLANS:
	+ We want to improve Neural Networks definition and training to create systems able to predict game states from previous predictions as input
	+ Network Latency simulation inside Online Version to the prediction system.

# MPAI-SPG VIRTUAL ARCHITECTURE

Once the software solution was consolidated, the atomic structure for producing and processing the data that will train the neural networks was also formalized.

Since December 2021 in collaboration with the Department of Computer Science of Turin, the architecture is available and it uses the set of virtual machines shown in the drawing below.



*Fig.1 : MPAI-SPG Virtual Architecture for Pong prototype*

We setup two steps to complete the process:

1. Creation of the log files by the LPA machine, which runs multiple instances of the Unity project. At the end of the operation, we will have a set of log files generated.
2. Using the log files to learn the neural network on the NNTA machine, which runs a python program with TensorFlow.

To better describe this atomic setup, we report the minimum requirements of the architecture, starting with physical servers:

**LPA**: Intel Core i7-11700KF, 16 GB RAM, hard disk space 2TB SSD, Windows 10 Pro, Unity 2020.3.4f1, MLAgents 2.0, Nvidia GTX 1660

**NNTA**: Intel Core i7-11700KF, 32 GB RAM, hard disk space 2TB SSD, Ubuntu Server 20.04.3 LTS, Python 3.7.9, Tensorflow 2 (pip version - Python > 19.0), Keras (Python API, it would be included in the Tensorflow installation)

The requirements will be translated in virtual requirements, comparing benchmarks of PassMark (I.E. https://www.cpubenchmark.net/)