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

**Public document**

|  |  |
| --- | --- |
| **N1011** | 2022/12/21 |
| **Source** | Requirements (SPG) |
| **Title** | MPAI-SPG Status report  |
| **Target** | MPAI Members |

# Project status

The month was devoted to continue to prepare the demo of MPAI-SPG.



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
	+ Now we have a set of 3 predictive Long-Short Term Neural Networks trained with about 28 million of records, corresponding to 16500 Pong scores
	+ These networks can forecast the next game state at time N+1 given a predefined number of states up to time N
	+ In case of packet latency or loss, SPG starts uses the past game states received from the online game server as input to forecast the next game state. This means that it can forecast, in an approximate way, the next game state, given what happened before.
	+ Network lag simulation: from an external tool to internal synchronous packet loss
		- After some synchronization issues using an external tool we have decided to modify the client source code.
	+ Agent behaviour
		- We noticed some strange behaviours of the paddles: we investigated the reasons of this issue analyzing the training rules.
	+ Physics issues
		- We discovered that strange behaviors of the game were caused by some Physics bugs usually located in the corners of the field.
	+ Physics refactoring
		- Collider changes: from circle to square
		- We blocked the Z axis rotation
	+ Agent’s training
		- We tried to reduce the number of observed parameters, in particular we removed the ball raycast: the resulted agents had worse performance.
		- We kept the ball raycast along with the game state parameters.
	+ Project revision management
		- We transferred the PONG project on GitHub.
	+ Project revision management
		- We moved to MPAI GitLab Server and we fixed some issues related to the project structure.
	+ The Offline data generation can run multiple instances on the same server
	+ Bug fix in the network evaluation scripts: we fixed a bug in the engines scripts that caused in some cases an infinite loop in the evaluation process
	+ The processing of the Physics Engine log files has finished correctly
	+ A new member joined MPAI-SPG: Polytechnic University of Turin is now in the MPAI-SPG team; this group will work on:
		- A new prototype scenario: racing games
		- Anti-cheat feature for MPAI-SPG
		- Improvement of the present architecture, in collaboration with University of Turin
	+ We worked on the physics and behaviour engine scripts to produce training data to feed the training process (using Keras library on TensorFlow framework) of the corresponding SPG neural networks.
	+ The training data for the rule engine need more investigation since the size doesn't match the other engines training data size. Moreover, also the associated script still requires some bug fixes.
	+ Polytechnic University of Turin has started the prototype for the racing game.
	+ PONG: we are syncing the physics of the online prototype with the new changes of the offline prototype.
	+ Polytechnic University of Turin has completed the prototype for the racing game and they are working on the neural networks.
	+ RACING GAME: with ML-Agents implementation now the cars are able to run on every generated track
	+ RACING GAME: we defined three ML-Agents profiles to try different driving styles.
	+ RACING GAME: setup of options to check various input configurations (player VS AI, AI VS AI…)
	+ RACING GAME: implementation of a track generator to train agents
	+ PONG: we have ported the physics object properties from the PONG Offline version to online PONG Server and PONG Client.

PLANS:

* + PONG: updating the trained Physics and Behaviour MPAI-SPG neural networks to the online PONG Server.
	+ RACING GAME: full development of the neural network section of the MPAI-SPG architecture
	+ Setup of the demo with the following schedule:
	1. Reference game without network issues and SPG reference; it will be called «the reference game»
	2. Reference game with a network loss (loss of Controller Data), introduced at second #n without SPG inference
	3. Reference game with a network loss and SPG reference
	4. Recording of the reference game with conditions in A., B., C.
	5. Objective parameters comparison for performance evaluation