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

|  |  |
| --- | --- |
| **N408** | 2021/11/24 |
| **Source** | Requirements (GSA) |
| **Title** | MPAI-GSA Status report  |
| **Target** | MPAI Members |

# Project status

The month was devoted to achieve a full working prototype of Pong with the authoritative server.



Figure 1 - Pong game

It started with an off-line version that was extended and brought online.

To comply with the specifications of MPAI-SPG (*Figure 2*), the type of online architecture chosen was "client/authoritative server".

Two programs were developed, one representing the game server and the other representing the clients. At runtime there will be two different instances (process C – C1 & C2) of clients both connected to the game server (process S). The clients are interfaced to digital controllers: two directions and one fire button.



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 implemented 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
* PLANS:
	+ Design of neural networks following the refined architecture
	+ Collection of the logs, playing a huge number of games autonomously

# MPAI-SPG REFINED ARCHITECTURE

The aim is to train our AI Neural Networks inside the Server without giving them the exact knowledge about Game Engine’s way of working. This way the standard would be usable also with new technologies.

Making references to the classic MPAI-SPG Schema, we can see that each Online Game Server component (Physics, Rules and Behavior component) (blue boxes) get in input a set of data called GM and return their results as a new set of data GM’. This sets of data are different depending on the software used to build the game, so we cannot base our standard on this data format to make an interoperable standard. Instead, we can see that, independently from the development platform used, one thing that will be always visible is the Game State, that can be referred as the set of data composing it (players position, ball position, velocity, score, etc.). So, we must develop our AI training on this set of data, to make our standard interoperable and long term usable.

            

*Fig.1 : MPAI-SPG Architecture*

Defined a new set of data composing the Game State (referring to “Pong”, these are player’s position and velocity, ball’s position and velocity, scores, etc.) called **N**, the blue Game State Engine creates a new Game State using user’s controls (CD) received from clients in order to create the next Game State **GS**. At moment ***t*** the Game State **N** is sent to the AI together with the Controller Data (CD). The way the AI inside the Server should work is to create the next Game State **N+1 at time *t+1*** without using the same technique used by Blue Boxes (Game State Engine, Physics Engine, Behavior Engine, Rules Engine work together), but using Prediction instead.

In this way the AI should work like a human brain, which creates approximations about the world and doesn’t make physics calculation about objects movements. This should look like a person which knows the game and, watching the game from the outside, knows how the things should go to make the Game State consistent.



*Fig 2: MPAI-SPG Refined Architecture*

Referring to the Refined Architecture Schema, Game State **N** arrives to the AI and is divided between the three Sub-NN (Physics, Rules and Behavior) by the “Divider”, which is not a AI anymore but is a script which sends correct data to the three NNs. Each NN takes from the Divider its own data as input and makes a prediction referring to its role (Physics engine takes physics data in input and returns a physics output). The 3 outputs are collected together by the “Composer”, which also is not a AI anymore and returns a new Game State **N+1**, that represents predicted Game State.

Each NN will be trained referring to two subsequent Game States taken from log files generated by Online or Offline game. Each NN will take a subset of the Game State **N** and will have to return a subset of the Game State **N+1**.

This way, we can train our 3 NNs using two following states taken from Only or Offline game played from automatic players. At the end of the training, the set of NN inside the Server, given a Game State **N** and a User’s move CD will be able to predict the next Game State **N+1** without using Game Engine technique, but using prediction instead, to give a different “way of thinking” about game’s flow.