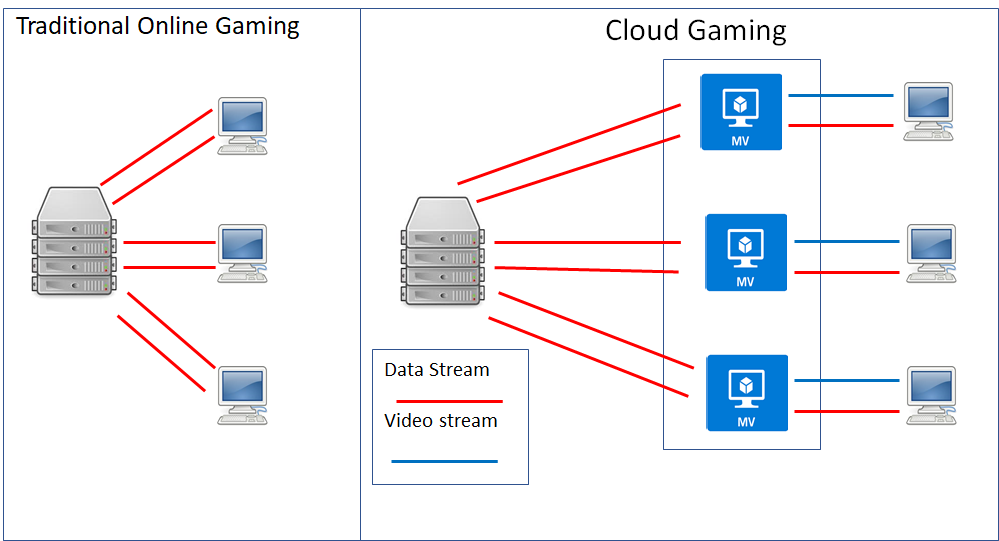
|  |  |
| --- | --- |
|  | Moving Picture, Audio and Data Coding by Artificial Intelligence  www.mpai.community |

|  |  |
| --- | --- |
| **N157** | 2021/02/17 |
| **Source** | Davide Cavagnino, Leonardo Chiariglione, Maurizio Lucenteforte, Marco Mazzaglia |
| **Title** | Proposed Draft MPAI-SPG Use Cases and Functional Requirements |
| **Target** | Audio-Events-Data |

# Introduction

Currently, online gaming can be classified in two categories illustrate by *Figure 1*:



*Figure 1 – Two categories of online gaming*

In traditional online gaming, at steady state:

1. The server
   1. Receives the data sent by the client(s).
   2. Processes the data.
   3. Sends an input-dependent sequence of data to the clients
2. Each client
   1. Uses the data from the server to create and display appropriate video frames.
   2. Collects input data from the player.
   3. Sends the data to the server.

In cloud gaming:

1. The server
   1. Receives a sequence of data from the Virtual client(s).
   2. Computes the Game State.
   3. Sends data to the Virtual clients.
2. Each Virtual client
   1. Receives the data from the server.
   2. Computes the video frame.
   3. Sends video frame and other server data to its client.
3. Each client
   1. Display the video frame received from its Virtual client.
   2. Collects input data from the player.
   3. Sends the data to the server.

In case the connection has temporary high latency or packet loss (in the following called network disruption) two strategies may be used to make up for missing information

1. Client-side prediction when information from the client does not reach the server or from the server does not reach the client
2. Server-side prediction when information from the client does not reach the server

In a game a finite state Game machine calculates the logic of the game from the inputs received from the game controller. The client reacts to such user input before the server has acknowledged the input and updated its Game state. If an updated Game state from the server is missing, the client predicts the Game state locally and produces a video frame that is potentially wrong. When the right information from the server reaches the client, the client Game state is reconciliated with the server Game state.

# Off-line and on-line game models

Local console

Client-server (online gaming)

Client-server (cloud gaming)

A particular implementation may use additional components.

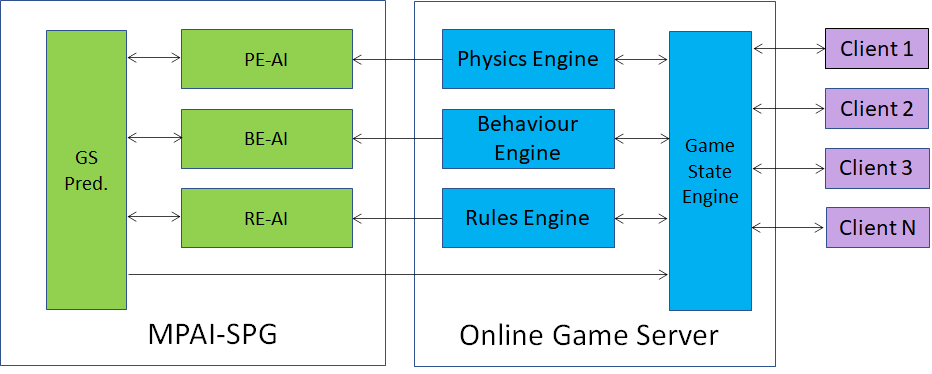
# The MPAI-SPG solution

MPAI Server-based Predictive Multiplayer Gaming (MPAI-SPG) intends to exploit the potential of Artificial Intelligence (AI) and Machine Learning (ML) to address the serious problem of online and cloud gaming in two main cases:

1. The server misses information from one or more clients and does not have sufficient information to compute the new game state and communicate information back to the clients (in the following called “missing data”).
2. One or more clients produce deliberately faulty information to the server in the following called “cheating prevention”).

An architecture that is capable to mitigate the impact of the two cases is depicted in *Figure 2*.

Note that, if particular implementation uses additional components, they may have their Twins.



*Figure 2 – An MPAI-SPG enhanced online gaming setting*

# Operation of MPAI-SPG

The operation of MPAI-SPG is described in 4 cases obtained from the combination of

1. Missing data
   1. Static neural networks
   2. Continuous-training neural networks
2. Anti-cheating
   1. static neural networks
   2. dynamic neural networks

Feasibility of Continuous-training neural networks is for further study.

## Missing data – Static neural networks

### Normal operation

1. Data from clients reach the server (input data).
2. Game State Engine (GSE) dispatches data to the appropriate Game Engines (GE)
3. Each GE processes the data and sends:
   1. Processed data to GSE
   2. Data received from GSE to their Twins (green boxes)
4. GSE sends the game state to the clients.

### Missing data from client 4

1. Data from clients 1-2-3 reach the server (input data).
2. Game State Engine (GSE) dispatches data received to the appropriate Game Engines (GE).
3. Each GE processes the data the best it can and sends input data from clients and processed data to its Twin.
4. Each Twin
   1. Feeds its NN with the data received.
   2. Sends the output of its NN to GS Pred.
5. GS Pred
   1. Feeds its NN with the data received from the Twins.
   2. Sends the output of its NN to GSE.
6. GSE sends the game state to the clients.

## Missing data – Continuous learning neural networks

### Normal operation

1. Data from clients reach the server (input data).
2. GSE dispatches data to the appropriate GEs
3. Each GE processes the data and sends:
   1. Processed data to GSE.
   2. Data received from GSE to their Twins (green boxes).
4. Twins feed input data and data from their Twin GE to their NN (continuous training)
5. GSE sends the game state to the clients.

### Missing data from client 4

The process is the same as in 4.1.2.

1. Data from clients 1-2-3 reach the server (input data).
2. Game State Engine (GSE) dispatches the data received to the appropriate Game Engines (GE).
3. Each GE processes the data the best it can and sends input data from clients and processed data to its Twin.
4. Each Twin
   1. Feeds its NN with the data received.
   2. Sends the output of its NN to GS Pred.
5. GS Pred
   1. Feeds its NN with the data received from the Twins.
   2. Sends the output of its NN to GSE.
6. GSE sends the game state to the clients.

## Cheating prevention – Static neural netwoks

### Normal operation

1. Data from clients reach the server (input data).
2. Game State Engine (GSE) dispatches the data received to the appropriate Game Engines (GE).
3. Each GE processes the data and sends input data and processed data to its Twin.
4. Each Twin
   1. Feeds its NN with the data received.
   2. Sends the output of its NN to GS Pred.
5. GS Pred
   1. Feeds its NN with the data received from the Twins.
   2. Sends the output of its NN to GSE.
6. GSE compares its own data with the data from GS-Pred.

### Missing data from client 4

1. Data from clients reach the server (input data).
2. Game State Engine (GSE) dispatches the data received to the appropriate Game Engines (GE).
3. Each GE processes the data the best it can and sends input data and processed data to its Twin.
4. Each Twin
   1. Feeds its NN with the data received.
   2. Sends the output of its NN to GS Pred.
5. GS Pred
   1. Feeds its NN with the data received from the Twins.
   2. Sends the output of its NN to GSE.
6. GSE sends the game state received from GS Pred to the clients.

## Comments

We should explore he possibility for GS Pred to receive the GS to improve its NN, in case GSE has recognised that there is no cheating.

# Data flows and data types

## Data flows

1. From clients to GSE
2. From GSE to
   1. Physics Engine
   2. Behaviour Engine
   3. Rules Engine
3. From Physics Engine to PE-AI
4. From Behaviour Engine BE-AI
5. From Rules Engine BE-AI
6. From PE-AI to GS Pred
7. From BE-AI to GS Pred
8. From RE-AI to GS Pred
9. From GS Pred to GSE
10. From GSE to GS Pred (? - it can be a valid flow in the anti-cheating use case)

## Data types

It is assumed that there are several neural networks, specialised in a certain type of information.

Behavior engine AI

Rules engine AI

Physics engine AI

Game state engine AI

They generate three types of 'messages' that constitute the types of data that circulate and define neural networks:

controller data

game message

game state

Each neural network has a set of data input that is classified as controller data, game message or game state.

They are classified according to this hierarchy.

**Data definition**

Each data can be “RAW” or “predicted”. If it is RAW, the data are from different clients and they have been computed correctly because no information is missing.

If data is missing, the neural networks create the predicted missing data. After this step, produced data is labeled adding “p” as suffix.

**(p) stands for predicted**

**Controller Data ==> label CD(p)**

They represent the input data from different controllers; they can be found in the behaviour engine that manages inputs and character states.

Digital inputs cd (inputID, 1/0, playerID) ==> cd (inputID, 1/0, playerID, predicted)

Analogue inputs cd (inputID, value, playerID) ==> cd (inputID, value, playerID. predicted)

Motion inputs cd (inputID, valuesArray [], playerID) ==> cd (inputID, valuesArray [], playerID, predicted)

**Game Messages ==> label GM(p)**

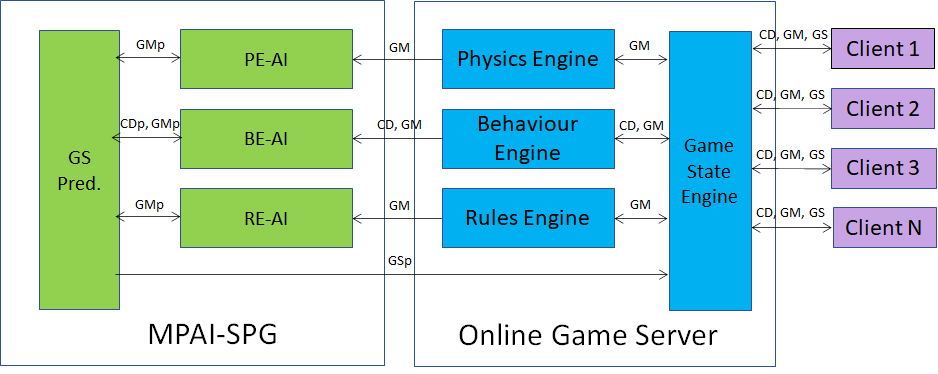
They represent the different game messages developed in the online game server; they includes all game messages defined in Physics Engine, Behaviour Engine and Rules Engine.

gamemessage (timestamp, methodName, SourceID, DestinationID, typeOfCommunication, param1, param2,.....,paramK) ⇒ gamemessage (timestamp, methodName, IDSource, IDDestination, typeOfCommunication, param1, param2,.....,paramK, predicted)

**GameState ==> label GS(p)**

It is game state that represents the status of the game at time “**t**”. It is the recollection of all meaningful information useful to each client in order to continue the game and to render ther graphics for all players in each client.

GameState (timestamp, gameMessage1, gameMessage2,...., gameMessageK) ⇒ GameState (timestamp, gameMessage1, gameMessage2,...., gameMessageK, predicted)



*Figure 2 – Data flow in online game server and MPAI-SPG*