Design and Development Of A Generative Agent That Models Human Conversational Behavior

Ibrahim El Kaddouri
Katholieke Universiteit Leuven
Leuven, Belgium
ibrahim.elkaddouri@student.kuleuven.be

***Abstract*—This paper outlines the design and development of a generative agent that can hold a conversation with players in the ETAP game. ETAP is a serious first-person, single-player dialogue game set in a fictitious 3D environment aimed to teach players how to recognize emotions.**

The generative agent assesses the level of trustworthiness and the emotion it should experience while conversing with the player. The following emotions comprise the emotional state the agent can experience: happy, angry, disgust, fear, surprised, sad or neutral.

The first goal would be to make sure the agent converses in a way that closely resembles real-world communication. It should avoid making pointless remarks or asking questions and neglecting user's statements in the process.

The second goal of this paper is for the generative agent to provide the player with the right degree of trust and emotion based on the ongoing conversation. These trust and emotion values are used to visualise the facial expression of the agent when conversing with the player. As such, if the player is talking about the death of a family member, the agent shouldn't have an overly happy facial expression.

In this paper, it has been observed that when a user acts disrespectfully toward the agent, the agent shows the right kind of facial expression back. The generative agent nevertheless keeps displaying artifacts that suggest the dialogue was generated by a large language model. Regretfully, no exact measurements were made to ascertain the agent's effectiveness. The project code[1] is open source and a small part of the game[2] can be played online.

List of Figures

Contents

[1]https://github.com/ChristianPoglitsch/EmpathicAgents
[2]https://etap.gamelabgraz.at/

I. Introduction

The World Health Organization (WHO) estimates that one in every 100 children globally receives an ASD diagnosis [14]. Individuals with autism spectrum disorder (ASD) frequently have co-occurrence of psychiatric disorders [11], such as depression, anxiety, attention deficit hyperactivity disorder (ADHD), psychotic symptoms, and emotional instability syndromes [3].

Autism spectrum disorder (ASD) can negatively affect individuals' ability to communicate. ASD patients find everyday conversations challenging because of problems in recognizing social signals and interpreting emotions [13]. The treatment for the so-called 'social blindness' is usually intensive individual therapy sessions or group therapy sessions. However, this type of therapy is expensive and therefore often occurs only rarely [13].

The Early diagnosis and personalized Therapy in Autism spectrum to Prevent severe disorders (ETAP) project aims to prevent severe disorder by significantly increasing access to emotional recognition training for those with ASD and at low(er) cost [13]. To this end, Poglitsch et al. proposes the development of a game that focuses on social interaction and discusses the requirements for building such a game [10] [9].

The game is intended to virtualize emotional recognition therapy. It would consist of a player (the ASD patient) and a non-playable character (NPC) who would interact with each other via a chatting system with verbal communication. The agent would be able to recognize the emotional state of the player by the frequency at which the individual makes reactions and measuring the stress level of the individual (via skin impedance, heart rate) [13].

An important aspect here is that one should strive for hyper-realistic and responsive avatars in order to show detailed emotions to the individual. Moreover, the agent should show as little to no artifacts of being a Large Language Model (LLM) when conversing. To evaluate whether the agent displays empathy properties, there are certain metrics that could be used, such as the Bot Usability Scale, System Usability Scale, Social Responsiveness Scale [9].

In short, one can use large language models to achieve this goal together with a realistic visualization of the agent to lead to the creation of a conversation partner. The use case is to provide an opportunity to practice and establish social skills for people with ASD.

The long-term goal for this game is to be as as effective as a real 1-on-1 therapy session. It is supposed to be used 15 minutes a day, 5 days a week, for at least a month. To the authors' knowledge, there is no such virtual, interactive health care system on the market. This is an innovative solution to the problem posed above.

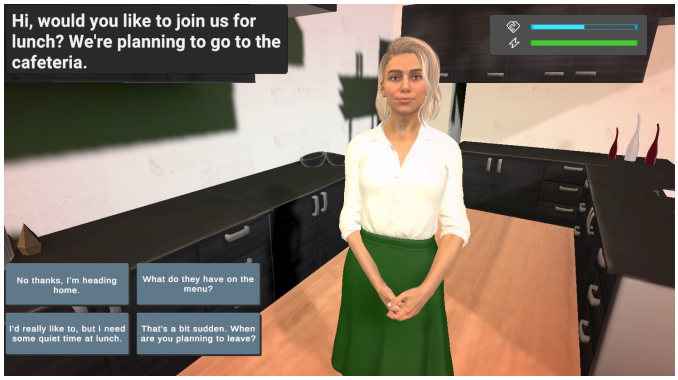

Fig. 1. Part of a dialogue quest with colleague Camila to be completed by the player. This is not a conversation with the agent, but a dialogue quest.

A. Game Overview

ETAP is a single-player first-person dialogue game set in a fictional world, which can be compared to a very limited version of the following games: *Firewatch*, *Life is Strange*, *The Stanley Parable*, *A Plague Tale: Innocence* or *Heavy Rain*.

In ETAP, the player can navigate an office scene while completing quests. When the player encounters a colleague, the player can begin a conversation with the NPC. Each NPC has a distinct personality, plan and memory which allows for a more personal interaction with the player. During a conversation, the agent can express different emotions. If an NPC is angry at a player, it may try to finish the conversation and show an angry face. Each character has different parameters for expressing emotions, some show their emotions clearly, while others are more restrained.

The player will also be presented with quests during the game which can include dialogue quests, see Figure 1. The player must complete the dialogue in order to continue the gameplay. When presented with options during a dialogue, the player can make different choices on how the conversation should continue. Each choice affects two values: 'Energy' and 'Social Acceptance'. Choosing an antisocial option or responding in a way that harms the relationship between the NPC and the player will lower the 'social acceptance' score. The 'Energy' score goes down each time the player interact with anybody. The goal is to avoid dropping these values too low in order to finish the quest. Additionally, each choice affects the NPC's emotional state which is represented by six core emotions. These emotions are happy, angry, disgust, fear, surprised, sad or neutral and are part of Plutchik's wheel of emotions [8].

Additionally, the game includes mini-games such as *Ekman*, *Emoty Crush*, *Jump* and *Memory*. One notable mini-game, 'Ekman' is inspired by Paul Ekman and Wallace V. Friesen's work [2]. In this mini-game, the player has a limited amount of time to recognize the correct facial expression shown on screen. The player is also required to show specific emotions on their face when prompted within a given time limit.

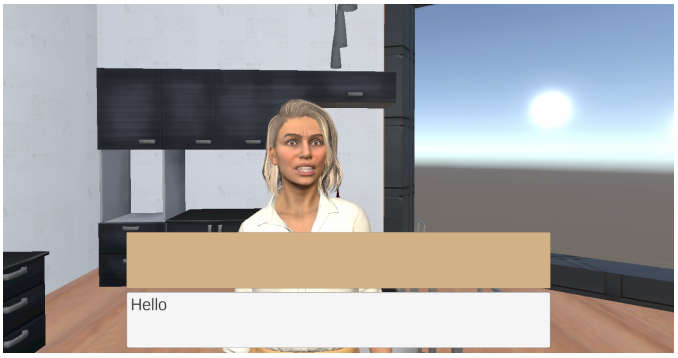

Fig. 2. A scene during development where the player is conversing with colleague Camila.

B. System Design

The system compromises of the following components:

- **The AI Subsystem:**
  - When the player is talking to an NPC, the NPC should act and respond in a way similar to human interaction, see Figure 2.
  - The AI subsystem is responsible for generating responses for the agents and inferring emotions.
- **The Dialogue Subsystem:**
  - Responsible for handling the dialogues in the game.
- **The Mini-games Subsystem:**
  - Mini-games serve as filler for the game.
  - They also serve as a training ground to improve the player recognition of emotions.
- **The Quest Subsystem:**
  - The game consists of quests that the user must complete.
  - Quests include activities such as picking something up, looking for something, calling somebody, and finishing dialogues with NPCs.
- **The Statusbar Subsystem**

II. Related Work

Virtual reality games have been used in the past for assisted learning in order to study the effects and potential of improving emotional experiences of autistic people [1] [5] [4] [6].

This project will be heavily based on the paper by Joon Sung Park et al., in which an agent architecture was proposed that relies on a LLM with memory to query past experiences [7]. It is also enhanced by giving the LLM agent a personality and the ability to plan events. This paper will explain this architecture in more detail and how it will eventually end up in the game system.

III. Generative Agents

The large language model is extended with a database to store the agent's past experiences. It retrieves some memories from the database and makes new observations based on those memories. Eventually, it retrieves the relevant memories in order to decide how to plan its day, see Figure 3 [7].

The architecture consists of three big parts. The first part is the 'memory stream'. This is the database that stores past conversations with players or agents. It also stores events that the agent perceives in its surroundings and events that are reflections of earlier events that were present in the database. In order to retrieve the correct memories out of the database, the architecture optimises for three separate variables: 'relevance', 'recency' and 'importance'. The second part is called 'reflection' where multiple past events are retrieved from the memory stream in order to generate new insights from it. The third part is 'planning' or the ability to plan the day of the agent based on past memories but also on personal information of the agent which is stored in 'scratch' memory [7].

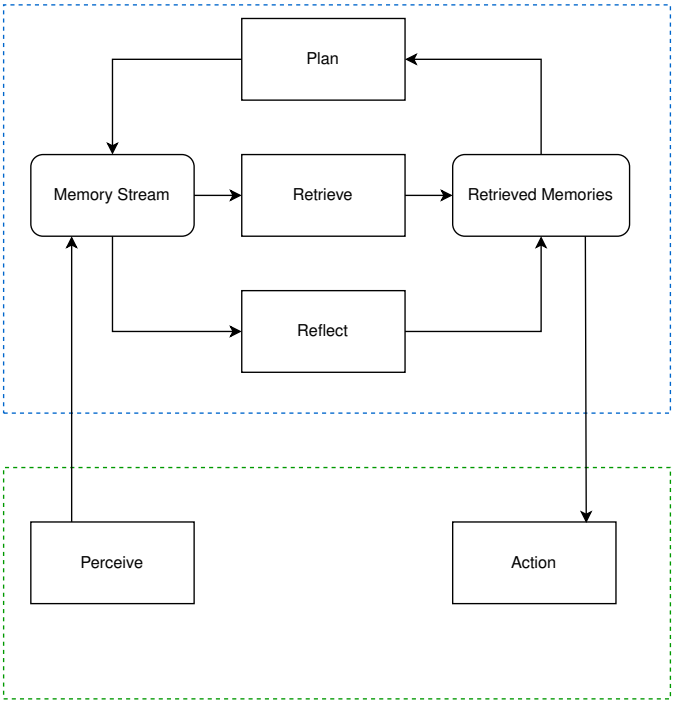

Fig. 3. The Generative Agent Architecture [7].

A. Memory

The need for a memory stems from the fact that a large language model has a finite context window size and not all past experiences are relevant for a certain prompt. At its core, the memory stream is a big JSON file composed of a list of concept objects or nodes. Each node stores a certain number of parameters, such as 'description' and 'poignancy'.

There are three types of events or types of nodes that can be stored in the database. It can be an observation , in which case the event is assigned an 'event' value. If it came from a conversation with the player, it is assigned 'chat' and lastly if it came from a reflection, it is assigned 'thought'. These are the three types of nodes which can be added to the the database [7].

```
"node_<int>": {
 "node_count": <int>,
 "type_count": <int>,
 "type": <"event"| "chat" | "thought">,
 "depth": int,
 "created": string,
 "expiration": null | string,
 "subject":
    <"{world}:{sector}:{arena}:{object}">,
 "predicate": string
 "object": string
 "description": string
 "embedding_key": string,
 "poignancy": 1,
 "keywords": list<string>,
 "filling": list<string>
}
```

The following is an example of how such an object can be filled with some arbitrary data. In the example below, some agent observed that the bed in the main room is in use.

```
"node_9": {
 "node_count": 9,
 "type_count": 6,
 "type": "event",
 "depth": 0,
 "created": "2024-08-13 11:11:20",
 "expiration": null,
 "subject":
    "Graz:apartment:main_room:bed",
 "predicate": "be",
 "object": "used",
 "description": "bed is being used",
 "embedding_key": "bed is being used",
 "poignancy": 1,
 "keywords": ["used", "bed"],
 "filling": []
```

In order to retrieve the correct memory objects, the following three parameters should be taken into account: recency, relevance and importance. Recency is an exponential decay function over time since the memory was last retrieved. Importance weighs how import a single memory object is. This is calculated by querying the LLM and inferring how important that event was. This is also the poignancy score that can be seen in the memory object above. The higher the score, the more important the event is. Relevance is measured by comparing the description of the memory object with some query key. If they are semantically close or equivalent, the relevance score would be high. This type of inference is called 'semantic search' and is done by comparing the embedded vectors of both the query and the description key inside the memory object.

B. Reflection

In order to reflect on past events, the large language model is prompted to give 3 questions that would provide the most information about particular subjects. The prompt is accompanied by 100 data entries from the database, which would provide contextual background for the inquiry. Do remember that each memory object has a subject field which is used in this particular prompt. Once these 3 questions are generated, they would be used as queries for retrieving relevant memory object out of the database. Finally, the language model is prompted once more to extract new insights from the retrieved data.

C. Planning

Planning corresponds to filling in a day's worth of activities. A single planning entry includes information such as location, start time and duration. When planning, it is important to know what the agent does for work, what hobbies they practise, who they have met and where they work. This type of data will be important for creating schedules that make sense. A schedule that makes sense, is one that doesn't conflict with previous days. It is not possible to work as a full-time professor one day and study to become a doctor the next. In this architecture, the planning is made in several steps. In the first step, the large language model is queried with agent specific information, which is stored in 'scratch' memory. This information includes the name, age, personality characteristics and more, see Appendix VIII. In this prompt, the previous day schedule is also included and the task is for the large language model to infer a new day schedule. The new plan is saved in the database and later decomposed into hour long segments and finally into 15 minutes chunks.

IV. Conversational Architecture

A. Prompting For Conversation

The message entered by the player is formatted in JSON in the form of `PromptMessage` and is sent to the python endpoint, see Figure 4. For further information about the JSON format, see Appendix VIII. After receiving this message, the Python endpoint generates the agents' response by adding specific memories about the player that the agent recalls. The server method in turn returns four important variables. The utterance (response), the emotion, the trust level and the end boolean variable. The emotion value is one of the six emotions described by Plutchik's wheel of emotions. The trust level is a numeric value from 0 to 10 where a lower number equates a lower trustiness between the player and the NPC.

Those four values are sent back to the Unity endpoint formatted in a JSON of type `PromptResponse` . The Unity scene displays the utterance back on the screen where it would be visible for the player. Based on the end variable received, the conversation may be ended on part of the agent.

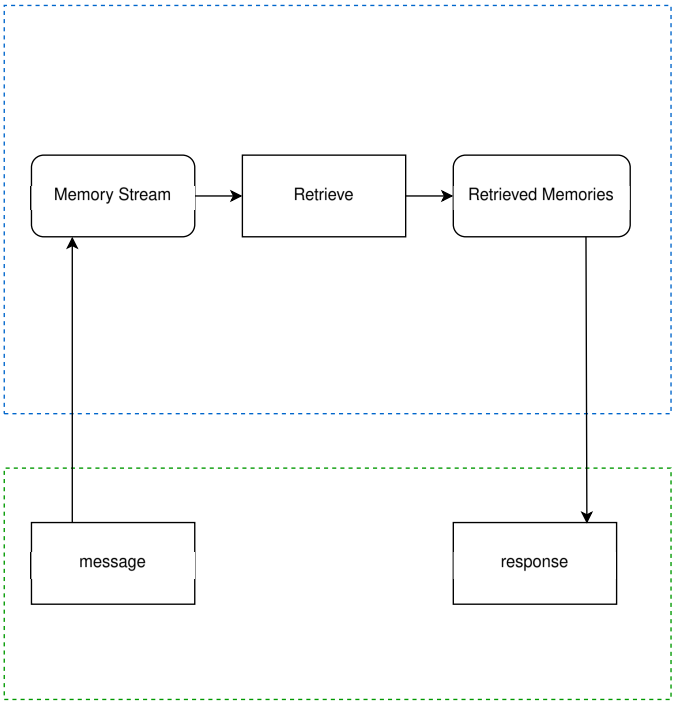

Fig. 4. The generative agent architecture when chatting with a player. The blue lines represent the python space whilst the green lines represent the unity space.

The trust and emotion variables are used in order to make the character express the right kind of facial expression.

```
utt, emotion, trust_level, end =
server.prompt_processor(
    player_name,
    persona_name,
    player_message,
    LLM_model
)

response_data = PromptResponseData(
    utt=utt,
    emotion=emotion,
    trust_level=trust_level,
    end=end
)

response_message = PromptReponse(
    type=ResponseType.PROMPT_RESPONSE,
    status=StatusType.SUCCESS,
    data=response_data,
)
sending_str =
response_message.model_dump_json()

socket.send_data(sending_str)
```

B. *Self-determinism of Agents*

The agents can also take actions in the environment without initiation from the player. These agents regularly perceive their surroundings at set intervals. The environment that's being perceived has a certain spatial hierarchy, similar to how objects in real life have a spatial hierarchy. To make the comparison, in order to specify an address, you start with the country, followed by the city, the street and finally the house number. Even within a house, you can specify the location of a certain object by identifying the room in which the object lies.

The environment can be decomposed into sectors, arenas and then individual objects. For example, an object like a 'Bible' can be found in a 'church' which is located in the city called 'Graz', which is part of the planet 'Earth'. In this case, the object 'Bible' will have an identifier as `"Earth:Graz:church:Bible"` . Each object in the environment has this kind of spatial information attached to it. When combined with other objects, these identifiers can be used to construct a tree data structure.

```
"Kortrijk": {
    "Saint Martin's Park": {
        "cafe": [
            "refrigerator",
            "cafe customer seating",
        ],
        "public restroom" : [
            "shower",
            "bathroom sink",
            "toilet"
        ]
    }
}
```

This asset is constructed during the development of the game and is therefore static. Naturally, not every single object in the environment needs to be included in this tree data structure. Only objects that agents can interact with are included. Agents perceive their surroundings within a certain radius and, for example, may only perceive the subtree 'cafe' in the example above. Assume that the character observes a 'Bible' and it observes a person called Camila using a knife to cut an arm of. All this information is collected at the Unity side and send in the format `MoveMessage` to the python endpoint, see Appendix VIII.

```
EventData(
    action_event_subject=
        "Graz:Church:room:Bible",

    action_event_predicate=None,
    action_event_object=None,
```

```
    action_event_description=None,
),
EventData(
    action_event_subject="Camila",
    action_event_predicate="is using",
    action_event_object="knife",
    action_event_description=
        "using knife to cut an arm of",
),
```

The python endpoint processes these events, adds them to its memory stream, and retrieves other similar events from memory. The tree structure that's present in the subject of one of these events can then be converted into natural language for generative agents. For instance, 'Bible' being a child of 'room' is converted into 'there is a Bible in the room'. Based on these retrieved events, the agent may change its plans. if the event involves a person instead of an object, it may decide to interact with that person. Lastly, it reflects on events in memory. This information is summarised in the following code block.

```
def move(
    perceived_events,
    llm_model,
) -> str:
    self.add_to_memory(perceived_events)
    self.retrieve(perceived_events)
    self.plan(llm_model, self.retrieved)
    self.interact(self.retrieved,
                  llm_model)
    self.reflect(llm_model)
    return self.action
```

The action that is returned is a string, which is in the format `{subject}:{predicate}:{object}`. The action string specifies to the agent in the Unity environment to move to the location in the subject field, if it indeed contains a tree structure as discussed earlier. The agent should in that case interact with the object at that location. If the subject is a person, then it can start a conversation with the person. This action string can once again be stored in the unity system so that the next person observing the scene knows that `Camila:is_using:the_knife`. The movement and animation of the agent when given such an action string can be implemented using Convai's AI[3] capabilities with unity, which is not the focus of this paper.

V. Discussion

Due to time constraints, there is no systematic evaluation of the effectiveness of the system. As a result, the discussion will be based on the experience of the author using the system during development.

One clear problem that came to light was the slow inference time when chatting. The time needed for the agent to generate a response is of critical importance and should be minimized as mentioned in the introduction. The author imagines the underlying cause behind the slow inference time is that there is a lot of prompts that need to be queried before the actual query for the response message. To explain this further, it's important to understand the conversation architecture, as described in section IV-A and in Figure 4.

First, the agent retrieves general information about the player from the memory stream based on the name of player as the query key. This involves searching through the entire database for past interactions with the player. This search process includes duplicating the database, sorting it and measuring the parameters 'relevance', 'importance' and 'recency' by going through the database 3 separate times. After which, the top 30 nodes are selected.

After obtaining some good information about the player, the next step would be to try to find out what the relationship is between the player and the character. This requires another query to the LLM and once the relationship status is found, there is a new query to the database. The database is queried to retrieve memory objects that are relevant to the information about the relationship status and the current chat history. This involves duplicating the database again, sorting it and choosing the top 30 elements.

The reader must remember, this is done every time a response must be generated. It doesn't take a genius to see that significant time gains can be achieved here. the system redundantly retrieves information that's already available if it was cached. Also, using a vector database instead of a JSON file can increase performance. A vector database is made to optimise queries for semantic search.

There needs to be a further investigation into the architecture from Joon Sung Park et al. [7]. Particularly if it is well-suited for real-time communication and how and where performance optimisation can be done. This should be achieved by identifying the hot paths within the codebase, by using any profiling tool available for python.

Another point of discussion is the difference between using a local LLM and openAI's LLM. The local LLM that has been used is made by 'mistralAI' and is called 'mistralai/Mistral-7B-Instruct-v0.2' [4]. From experience, it was observed that the local model took significantly longer to respond compared to querying the OpenAI API [5]. The model was run on a powerful workstation equipped with a Nvidia GeForce RTX 3090, which has 24GB of VRAM. While hardware limitations likely contributed to this delay, it is possible that the system could be further optimized by reviewing the code and batching prompts where possible in order to fully use the advantages of a GPU. However, one key takeaway is that running such a local model on everyday machines may not be feasible. This raises questions about how the final product should be designed. If

[3]convai.com/blog/adding-actions-for-ai-characters-in-unity

[4]By the time of writing this paper, v3 replaced v2.

[5]https://openai.com/index/openai-api/

the goal is for the game to run smoothly on average gaming PCs, it may be more practical to query LLM models via a service rather than run them locally.

A. Prompting For Conversation

In this project, two fictional agents were build with their own personal characteristics, see Appendix VIII. During development, they were prompted in order to see what kind of conversation they would hold. From experience, it was observed that they do remember earlier conversations, they also talk in a way that reflects their personality, but sometimes they show artifacts of being a large language model, for example, pretending knowing the person whom they are talking to even if they had no prior conversation.

As mentioned earlier, while generating a response, there are other variables generated as well, such as the trust level, the emotion value and the end value. From experience, the author noticed that the trust value correctly reflected the situation based on the given prompts. In the scenario where the user would comes off as aggressive and not show any appreciation whatsoever by constantly cursing the agent, resulted in the trust level reducing to lower values. There were more difficulties in getting the end boolean variable right. It was observed that it sometimes may stop unnaturally in a situation where it could actually continue the conversation.

Overall, more extensive testing is needed in order to know the effectiveness and reliability of this method. Once again, no extensive testing was done in order to evaluate the resulting implementation of the agents. The utterances were from experience somewhat believable responses, but it is without a doubt not yet fully human. It is sometimes too eager to help. It does not have awkward situations such as humans would have. Once again, further testing is needed as will be explained in section VII.

B. self-determinism of agents

The author observed a big discrepancy between using a local model compared to openAI GPT-4 when it came to self-determinism. In some of the complex prompts, the GPT-4 version successfully gave the answer to the location where a certain action should take place, whilst the local model would hallucinate and throw new locations or straight up not respect the formatting rules. This could be due to the fact that the 7B model is not expressive enough or due to the fact that the prompt is too complex and should be split up into multiple smaller prompts in order for the local model to understand the query fully.

VI. Conclusion

This paper provided the design and development of a generative agent nearly capable of modelling human conversational behaviour. By implementing the same architecture as proposed by Joon Sung Park et al. that combined LLM's with memory, reflection, and planning functionalities. Additionally, this paper explained how to achieve facial expressions when interacting with the agent. This was achieved using emotion and trust values and giving the agent the ability to stop the conversation if wanted.

One of the problems that came to light during development was the slow inference time, particularly when using a local LLM. This issue stems from redundant attempts to query the memory of an agent. There was high need for performance optimization, possibly by using vector databases or caching mechanisms. Furthermore, comparisons between the local Mistral-7B model and OpenAI's GPT-4 revealed some differences in response accuracy. This raised some questions about the feasibility of using local models in real-time applications on consumer hardware.

While the system demonstrated promising results, such as agents remembering past conversations and responding appropriately based on trust and emotion. There were also limitations experienced in generating fully human-like interactions. There is a need for more research and testing to ensure a more natural, believable conversations.

VII. Future Work

One of the key problems should be addressed as fast as possible, which is the focus on optimising the system. Especially in context of real-time performance. It would also be interesting to see to what degree local LLM's can compete with OpenAI's model.

Furthermore, extensive testing is needed in order to rigorously measure the believability factor of the agents. One of the ways in how to test the humanness of the generative agent is through conducting theory of mind test, the ability to track other people's mental states [12]. In those kind tests, humans and LLM's are compared on set of measurements that try to measure different human abilities, such as understanding false beliefs and interpreting indirect requests and recognizing irony and faux pas [12].

The goal would be to test the agent on the effectiveness of simulating human behaviour in order to use it in the ETAP game. One type of test that could be used is to let the agent be an interviewer, and let a group of test subject (humans) solicit for a fictional job. Each person would tell the interviewer about his set of skills and why he should be hired. At the end of the day, the system would query the agent in order to know which person would be the best fit for the job. Measurements can be taken with metrics such as the bot usability scale, system usability scale and social responsiveness scale [9].

REFERENCES

[1] Haneen Almurashi, Rahma Kammoun, Walaa Alharthi, Mohammed Al-Sarem, Mohammed Hadwan, and Slim Kammoun. Augmented reality, serious games and picture exchange communication system for people with asd: Systematic literature review and future directions. *Sensors*, 22, 02 2022.

[2] Paul Ekman and Wallace Friesen. *Unmasking the Face: A Guide to Recognizing Emotions From Facial Clues*. 01 2003.

[3] Dominique Endres, Ludger Tebartz van Elst, Simon Meyer, Bernd Feige, Kathrin Nickel, Anna Bubl, Andreas Riedel, Dieter Ebert, Thomas Lange, Volkmar Glauche, Monica Biscaldi-Schäfer, Alexandra Philipsen, Simon Maier, and Evgeniy Perlov. Glutathione metabolism in the prefrontal brain of adults with high-functioning autism spectrum disorder: An mrs study. *Molecular Autism*, 8, 03 2017.

[4] Kamran Khowaja, Dena Al-Thani, Bilikis Banire, Siti Salwah Salim, and Asadullah Shah. Use of augmented reality for social communication skills in children and adolescents with autism spectrum disorder (asd): A systematic review. In *2019 IEEE 6th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, pages 1–7, 2019.

[5] Panagiotis Kourtesis, Evangelia-Chrysanthi Kouklari, Petros Roussos, Vasileios Mantas, Katerina Papanikolaou, Christos Skaloumbakas, and Artemios Pehlivanidis. Virtual reality training of social skills in adults with autism spectrum disorder: An examination of acceptability, usability, user experience, social skills, and executive functions. *Behavioral Sciences*, 13(4), 2023.

[6] Yiu-kai Ng and Maria Pera. Recommending social-interactive games for adults with autism spectrum disorders (asd). pages 209–213, 09 2018.

[7] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023.

[8] Robert Plutchik. The nature of emotions human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice. *American scientist*, 89(4):344–350, 2001.

[9] Christian Poglitsch and Johanna Pirker. A qualitative investigation to design empathetic agents as conversation partners for people with autism spectrum disorder. In *2024 IEEE Conference on Games (CoG)*, pages 1–4, 2024.

[10] Christian Poglitsch, Saeed Safikhani, Erin List, and Johanna Pirker. Xr technologies to enhance the emotional skills of people with autism spectrum disorder: A systematic review. *Computers Graphics*, 121:103942, 2024.

[11] Tamara E Rosen, Carla A Mazefsky, Roma A Vasa, and Matthew D Lerner. Co-occurring psychiatric conditions in autism spectrum disorder. *International review of psychiatry*, 30(1):40–61, 2018.

[12] James Strachan, Dalila Albergo, Giulia Borghini, Oriana Pansardi, Eugenio Scaliti, Saurabh Gupta, Krati Saxena, Alessandro Rufo, Stefano Panzeri, Guido Manzi, Michael Graziano, and Cristina Becchio. Testing theory of mind in large language models and humans. *Nature Human Behaviour*, pages 1–11, 05 2024.

[13] Furtwangen University, Technical University Graz, University Hospital Freiburg, Obuda University, and University of Canterbury. Early diagnosis and personalized therapy in autism spectrum to prevent severe disorders (etap). ERA PerMed, 2022. Project ERAPERMED2022-276.

[14] Jinan Zeidan, Eric Fombonne, Julie Scorah, Alaa Ibrahim, Maureen Durkin, Shekhar Saxena, Afiqah Yusuf, Andy Shih, and Mayada Elsabbagh. Global prevalence of autism: A systematic review update. *Autism Research*, 15, 03 2022.

VIII. Appendix

A. UDP Protocol

The start message is sent when the unity client is being set up.

```
{
  "type": "STARTMESSAGE",
  "data": {
    "fork_sim_code": "string",
    "sim_code": "string"
  }
}
```

The prompt message is sent when the player wants to talk with a NPC.

```
{
  "type": "PROMPTMESSAGE",
  "data": {
    "persona_name": "string",
    "user_name": "string",
    "message": "string"
  }
}
```

The move message is periodically sent from the unity client to the python endpoint. After each observation made by the agent, a move message is sent.

```
{
  "type": "MOVEMESSAGE",
  "data": [
    {
      "name": "string",
      "curr_location": {
        "world": "string",
        "sector": "string",
        "arena": "string"
      },
      "events": [
        {
          "action_event_subject": "string",
          "action_event_predicate": "string",
          "action_event_object": "string",
          "action_event_description": "string"
        }
      ]
    }
  ]
}
```

Each persona or agent needs to know the current time, this time can be set from the unity endpoint. The current time is used in order to make plans.

```
{
  "type": "UPDATE_META_MESSAGE",
  "data": {
    "curr_time": "string"
  }
}
```

Each persona or agent has certain information or parameters that can be adjusted. Those parameters are stored in a JSON file at the python server. With this request, those values can be changed from the unity endpoint.

```
{
  "type": "UPDATE_PERSONA_MESSAGE",
  "data": {
    "name": "string",
    "scratch_data": {
      "first_name": "string",
      "last_name": "string",
      "age": "integer",
      "living_area": {
        "world": "string",
        "sector": "string",
        "arena": "string"
      },
      "recency_w": "integer",
      "relevance_w": "integer",
      "importance_ele_n": "integer"
    }
  }
}
```

the name of the character which the user is playing can be changed.

```
{
  "type": "UPDATE_USER_MESSAGE",
  "data": {
    "old_name": "string",
    "name": "string"
  }
}
```

If a persona character will be added to the game, it can also be dynamically added through the following call where certain parameters about the persona needs to be known upfront.

```
{
  "type": "ADD_PERSONA_MESSAGE",
  "data": {
    "name": "string",
    "scratch_data": {
      "curr_location": {
        "world": "string",
        "sector": "string",
        "arena": "string"
      },
      "first_name": "string",
      "last_name": "string",
      "age": "integer",
      "innate": "string",
      "look": "string",
      "learned": "string",
      "currently": "string",
      "lifestyle": "string",
      "living_area": {
        "world": "string",
        "sector": "string",
        "arena": "string"
      },
      "recency_w": "integer",
      "relevance_w": "integer",
      "importance_w": "integer",
      "recency_decay": "integer",
      "importance_trigger_max": "integer",
      "importance_trigger_curr": "integer",
      "importance_ele_n": "integer"
    },
    "spatial_data": {
      "FantasyLand": {
        "Northern Realm": {
          "Dragon's Lair": ["string"],
          "Ice Cavern": ["string"]
        }
      }
    }
  }
}
```

B. *Persona Characteristics*

```
{
  "curr_location" : {
    "world": "Graz",
    "sector" : "Saint Martin's Church",
    "arena" : "cafe"
  },
  "daily_plan_req": "Florian gardens at 9am
    everyday, and studies in the afternoon game
    development until 6pm, after which he goes
    for some drinks in the evening.",
  "name": "Florian",
  "first_name": "Florian",
  "last_name": "Schwarz",
  "age": 34,
  "innate": "sarcastic, dark and vulgar humor,
    helpfull",
  "look": "Florian has blond hair and blue eyes.
    Body size is on the smaller side.",
  "learned": "I'm often sarcastic, like when I
    make an absurd statement such as ransomware
    in Rust not being morally wrong. C# Unity
    GameDev by day, Programmer specializing in
    Java, working on a POS system like Orderman,
    by night. Systems: Acer Nitro 5, Lenovo X1
    Yoga with Arch Linux, both with arch linux.
    Browsers and Editors: Ungoogled-chromium,
    VSCodium. IDEs: IntelliJ, Android Studio.
    Prefer Java for coding tasks.",
  "currently": "Currently you are in Graz at
    the Cafe Mild",
  "lifestyle": "Florian goes to bed around 11pm,
    awakes up around 6am, but sometimes if he
    has a deadline to reach, the sleeptimes can
    become hectic and he would sleep at 4am and
    wake up at 7am. He likes balancing learning,
    coding, and project planning. Seeking
    mentorship or collaboration opportunities.
    Seeking detailed explanations for technical
    concepts.",
  "living_area": {
    "world": "Graz",
    "sector" : "Florian's apartment",
    "arena" : "main room"
  },
  "curr_emotion": "neutral",
  "curr_trust": {},
  "recency_w": 1,
  "relevance_w": 1,
  "importance_w": 1,
  "recency_decay": 0.995,
}
```