Tool-wielding LLMs for Game Progression Assistance

This report documents the independent study conducted by Ivan Martinez-Arias, advised by Adam M. Smith, at UC Santa Cruz during Fall 2024. Through connecting the Gemini LLM with a web-based videogame emulator, we introduce the concept of *live coaches*. We describe how the LLM uses tools to read/write information from the emulator, allowing it to answer users' queries about ongoing play.

Project Motivation

Game Backlog Problem

There are a plethora of games for people to choose from and play. With each year bringing their own influx of new games to play alongside popular games from the past, avid gamers will have trouble attempting to play and finish every game they have noted on their *game backlog*. Game backlogs refer to a list of unplayed games someone personally wishes to play and complete in the future. A related paper on game backlogs is linked here: explaining the purpose of them, the psychology behind it, and more. Game backlogs are a thing that happen for those who wish to play a variety of games in their life accounting for their free time. In design education, we want to expose people to older games, however time is limited to many people and nobody has time to play all these games.

Live-Coach Proposal

Our *live-coach* assistance tool will help with players being able to access older, retro games (e.g. Super Nintendo games) due to their ambiguity on progression. Not every game can be as clear cut as other games and with modern games providing more accessibility and transparency on making progression, it can be difficult to revert back to a retro style with such ambiguity. There will be some players who wish to progress through many games, retro and modern, without feeling as if they are stuck and unable to make progress to beat the game. We wish to experience the many virtues of what makes a game popular, whether it be popular for great or horrendous game design, without frittering away time from being stumped for game progress.

Our approach to this problem is to provide players with a tool that allows for tasteful cheating assistance. Such assistance allows players to conversationally decide at their pace what kind of game assistance they want while allowing them to decide their own level of assistance. Provided guided exposure, we expect players to understand and experience the classics or discussed-about games without investing a lot of time.

Player-Specific Assistance

Assistance is not universally adaptable to everyone's specific problems and requests and can be very finicky for some. Not everyone will want the same pace of assistance like someone else that worked for them (e.g. someone might want hand-holding assistance constantly, others might want it only when they really need it). Not everyone's advice will be applicable for someone's request (e.g. someone might get help with clearing a dungeon with a melee play style but what about someone who prefers ranged playstyles?). And not everyone has the same preferences towards accepting assistance (e.g. some might like visual guides to know what to do, but others can find it overbearing). It can be difficult and frustrating to cater towards a variety of player's preferences of assistance. With the live-coach assistance, it can provide personalized assistance through natural conversations with an instruction-tuned large-language model (LLM) being connected with the game to read the game's memory and register the player's game states.

Research Context

One of the inspirations of this project was based on Mawhorter's Player Assistance using Binary Decision Diagram (BDD) for the game Super Metroid. Mawhorter's contributions provided one fixed form of assistance, with the assistance in the form of path guidance towards what actions to perform from start-to-finish inside Super Metroid. We want to make it more flexible and conversational through connecting it with an LLM that the player user can use at their leisure.

Mawhorter's use of BDDs to provide assistance with a player's path planning in a videogame can be interpreted in the context of Aytemiz' program of repurposing a surplus of game-playing artificial intelligence to make games more inclusive: Your buddy, the grandmaster: repurposing the game-playing AI surplus for inclusivity.

To get started on this project, we need to assemble a few things. We require the **videogame** Super Metroid to be playable on a platform, such as an emulator, and to allow for game states to be read. We also require a **coach system** to generate the advice, such as action path advice from Mawhorter's BDD implementation. Next, we need a **LLM** that can respond to the user's text messages and queries. Finally, we require a **webpage** to connect all of them together to create an easily accessible prototype of the program.

Ideal Outcomes

Our ideal outcome of this project is a program that is able to interpret Super Metroid's current state through reading memory addresses and image analysis and use that information to understand the player's request given the game's context. Players can interact with the coach tool through text or audio speech and communicate with the

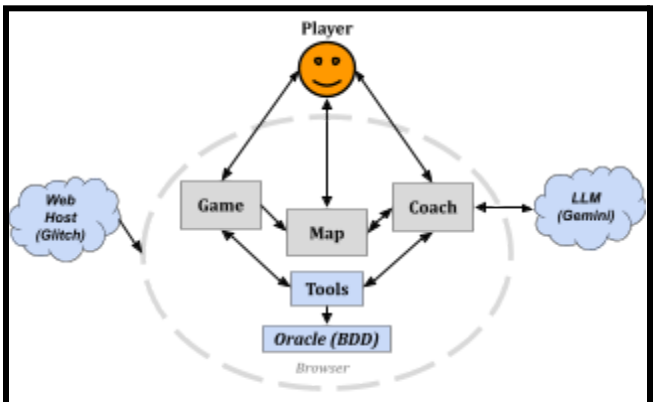

Figure 1: Architecture Design of the Live-Coach Assistance Tool

program as if they were talking with someone knowledgeable about the game. Figure 1 displays the architecture design of the interaction hierarchy for the prototype.

The human **Player** interacts with the Game, Coach, and Map components of the prototype with each aspect interacting vice versa with the **Player. Players** interact with the Game through the game's controls, with the Coach through providing their requests of assistance at any given moment, and with the Map through viewing the navigable embedded map.

The **Game** interacts with the Map and Tools through providing information necessary for their respective counterparts. The **Game** provides information player positional coordinates for the Map to display live where the player is within the game world and provides memory addresses for Tools to read and write accordingly, such as reading the values of the player's health and energy.

The **Map** interacts with the Player and Coach through providing live information. With the Player, it provides a live location of where the player is located (denoted with a Pink Circle for the Player's position). It also allows the Player to know the names and regions of each room through hovering over each room with the mouse cursor. With the Coach, it provides the player's positions in Super Metroid through room names, region locations, and nodes of every interactable object for the Coach to plan out a suggested path for the player to follow.

The **Coach** interacts with the Player, Map, Tools, and the LLM for being an assistant for the player. With the LLM, it is directly connected to Gemini 1.5 Flash due to the program being simple to implement into any web page. Being given a system prompt and instructions, we provide context for the purpose of being a **Coach** for the player in Super Metroid. With the Player, it continuously interacts through responses given in the chat box or shown inside the game world. Answers the **Coach** can provide can range from a variety of example:

- ➢ They can provide statuses of the game state such as the player's health and missile count
- ➢ They can provide clarification for what is being displayed on screen
- ➢ They can provide live action planning for where the player can go to next (Figure 2) with reinforcement assurance if the player is performing the correct actions for progress
- ➢ They can tastefully cheat through adjusting game values such as changing health values, opening doors at a whim, clearing enemies, giving items, creating save points, etc.

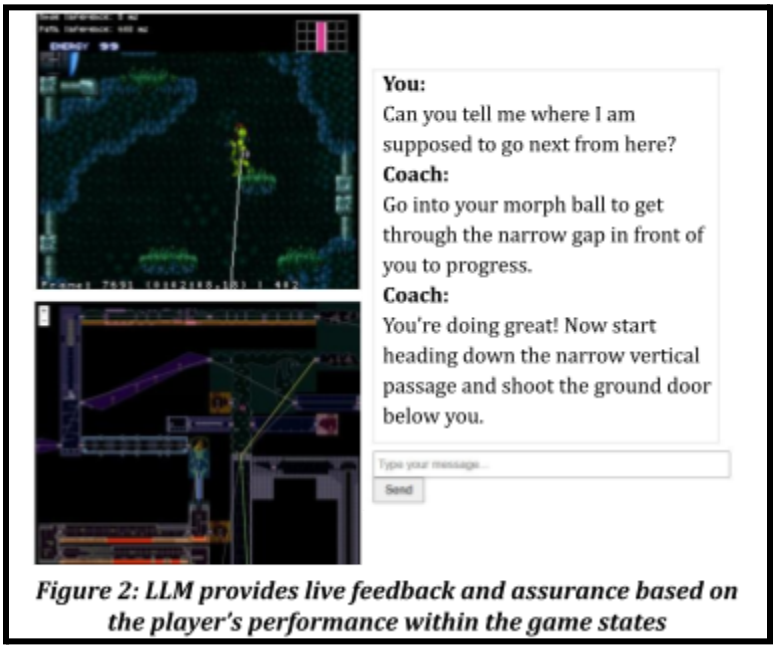

Figure 2: LLM provides live feedback and assurance based on the player's performance within the game states

With the Map, the **Coach** utilizes the player's position in the game world and can transfer that information towards the Tools to create action plans through BDDs (developed by Mawhorter) for a player to follow as suggestions to progress through what they wish to do. Requests for action planning can be as grandiose as asking how to beat the game from start to finish or as small as asking where the item is that they need next. With the Tools, the **Coach** uses developed functions that can decide and call upon what's most appropriate based on the user requests.

The **Tools** interact with the Game, Coach, and Oracle. Originally, **Tools** were a component of the Coach but have been separated for testing each component individually. The Coach has access to a toolbox of functions that they can use and decide what's best appropriate for the player's request at hand. The **Tools** interact with the game for extracting information through the game's memory addresses (using websites such as Jathys and Johnston's RAM pages for knowing what is in each memory address) and adjusting values when necessary. The Coach utilizes these **Tools** for resolving a user query and relays the solution into the game, such as displaying a line for the player to visually see and know what next room they should progress towards. The **Oracle** is an individual component that only interacts with the **Tools** and it provides a resolution towards the player's request such as creating the action planning and generating results utilizing the functions given.

In the overall hierarchy, the Player is interacting with the prototype under the **Web Page hosted on Glitch**. This prototype includes all components that are embedded within the **Web Page**, such as the Super Nintendo emulator with the game ROM of Super Metroid functioning, the Gemini LLM in the chat box portrayed as the Coach, the Tools that are utilized by the Coach, and the embedded Super Metroid Map. The **LLM** is connected through **Google's Gemini 1.5 Flash model** for their embedded implementation inside a web page and functionality of function calling in the Gemini API. In order to use Gemini through this web page, the player/user must generate an API code themselves to provide over to the program. An API code can be generated through a Google account for free with a limit of ~2 million tokens per day.

Preliminary Results

Web Page Implementation

We successfully implemented an LLM chat bot into a static HTML page (code linked here) hosted on Glitch with the assistance of Simon Willison's Gemini Chat App that is hosted on their own HTML page. Knowing that we can use the Gemini API and condense it down to a chat box, we were able to provide our own system prompts so it has context that it is an assistance-based program helping with the game Super Metroid. The prompt is as described below:

When the user asks you questions about the game Super Metroid, they are the player of the game. Use the tools provided to read and write the state of the ongoing game that the user can see in an emulator beside this chat interface.

Super Nintendo Emulator + Utility Map Implementation

Through the assistance of Smith's previous work on embedding a Super Nintendo emulator with a running agent and loading game ROMS, we were able to embed and run Super Metroid on the web page that allows keyboard control inputs and save/load game state features. Creating save states can be done through exporting the current game state and saving it onto the player's local computer. To load that state, the .state file must be placed in the same file system as the code and the current line of code that loads in the game state on runtime inside Toolkit.mjs must be replaced with the new .state file's address. Such functionality allows to skip the tedious progress in the start of the game such as the unskippable cutscene and get right to where you wish to be. With player controls functioning on the web page, Super Metroid can be played without any trouble but unfamiliarity with the controls.

We also incorporated a way to monitor the player's live position within the game world through an embedded Super Metroid Map Utility app created by BinoAl on Reddit. With the assistance of Mawhorter creating an atlas of Super Metroid through noting down information of each room such as their positions and room names, we are able to extract the map information for the Tools to use for their respective purposes.

LLM Coach Implementation with Gemini Documentation

With the Coach, we were able to provide some tools into Gemini that allows communication with the game emulator. The main way of communication for the tools was Gemini's API function calling where given descriptive context of what the created function by the developer does, the LLM can decide themselves what function is most appropriate to call for given the player's query. Despite its versatility, it provided trouble on getting the functionality of function calls to work due to the lack of documentation for Javascript/HTML implementation. Most of the documentation seemed centered around Python implementations, making what we need feel neglected inside the documentation. Further, there were instances where the deployed version of the Gemini service behaved differently than how it was described in the documentation.

Gemini's Safety Features

Another aspect we had trouble with using Gemini were the safety and security features. Due to the nature of Super Metroid and connecting an LLM with the emulator, we were prompted security errors when discussing the topics of "missiles" and "memory addresses". Super Metroid has missiles as an ability to shoot at enemies and doors; however when we ask for our missile count, the LLM might interpret that we are attempting to ask for missile codes in real life and deems the request to be dangerous content. When we want the LLM to read the memory addresses within Super Metroid to know values such as health count, the LLM might interpret that as a hacking attempt, deeming it dangerous as well. With the context of working with Super Metroid and reading the game's memory addresses, we disabled the features (by setting the block level to "BLOCK_NONE").

Coach and Toolkit

Originally, the tools component of the design was embedded within the coach. However due to Google's unclear documentation on new updates for models, our methods of function calling no longer work and Gemini became unreliable. Thus, we had to create the Tools component separately, denoted as the Toolkit.mjs inside the code. Through the toolkit, we separate many components for testing purposes in the case that the tools can function on their own without any reliability on other components not failing. Through the toolkit, we can create new functions that the Coach can call upon when necessary with similar functionality that it had when previously together but now with added security. One useful set of tools we offered were creating JSON specifications of the tools with prompts that describe their purpose and functionality for the LLM to have context on.

Left-Out Features

One feature that had not been fully implemented was Mawhorter's BDD program for Super Metroid. Due to the original project only working on Mawhorter's local computer through Python's Jupyter and Snes9x-RR (a variation of the Snes9x emulator that allows developer tools such as LUA script implementation), it required a vast amount of prerequisites to transfer the program onto another computer. For simplicity reasons and allowance for any future developer to work on the project, we attempted to transfer the prototype onto a web page. We were able to complete roughly half of the transfer onto Glitch such as playing on a Super Nintendo emulator with player controls, loading in game states, showing the embedded map for where the player is live in the game world; however, the action planning BDD had not been transferred yet due to time and complex constraints. The program requires a complete transfer from Python into Javascript, meaning we would need to go into Mawhorter's code files and decipher what files are absolutely necessary to let the program work. We did not want to only transfer the program over from a local computer to a web page, but we also wanted to make the code as concise as it can be so anyone interested in the project can read and understand the program within.

References

- https://dl.acm.org/doi/10.1145/3649921.3650014
- https://tools.simonwillison.net/gemini-chat
- https://bin0al.github.io/Super_Metroid_World_Map/leaflet.html
- https://github.com/aremath/sm_rando
- https://patrickjohnston.org/ASM/Lists/Super%20Metroid/RAM%20map.asm
- https://jathys.zophar.net/supermetroid/kejardon/RAMMap.txt
- https://glitch.com/edit/#!/snes-assistant-agent?path=map-with-chat.html%3A1%3A0
- https://snes-assistant-agent.glitch.me/
- https://dl.acm.org/doi/10.5555/3505464.3505467