Text to Game Generator App

Christopher Figueroa
SUNY Polytechnic Institute
Master's Independent Study
Study Advisor: Dr. Michael J. Reale

***Abstract*— Motivated by the limitations of traditional AI prompt approaches, the purpose of this project is to foster a novel set of innovative interactive game development techniques. The convergence of natural language processing and artificial intelligence in the game development workflow presents groundbreaking possibilities. The project's primary goals include real-time video game editing, direct compilation of AI-generated code, and dynamic modification of existing code structures. With a focus on 2D game generation, the Chat to Game Generator Interface provides developers the choice of three innovative game generation and editing techniques.**

I. Introduction

The introduction of the OpenAI's ChatGPT platform has been a game-changer for programmers within the software development industry, offering a potent tool for creating and refining code with efficiency and effectiveness. Game developers, in particular, have reaped significant benefits with the use of this newfound tool. ChatGPT enables them to generate and modify game code through natural language interactions, fostering a more intuitive and creative approach to game development.

However, one notable limitation in this process is the challenge game developers face in immediately knowing whether the generated code functions as intended. The Chat to Game Generator Interface project introduces a set of inventive game generation methods to the realm of real-time video game editing. This project represents the convergence of cutting-edge technologies, focusing on the seamless integration of natural language processing and artificial intelligence into the game development workflow.

Goals of Game Generator:

- Real-time video game editing
- Direct compilation of AI code
- Dynamic editing of existing code

The primary goals of this project include real-time video game editing, direct compilation of AI-generated code, and dynamic modification of existing code structures, all accomplished within the Python programming language. Designed with an emphasis on 2D game generation, the Chat to Game Generator Interface opens up new possibilities for game developers to interactively shape and refine their creations through conversation with AI-powered agents.

The integration of ChatGPT not only enables the generation of game content through natural language input but also facilitates direct communication with the underlying game code, providing an unprecedented level of control and creativity for developers. This project aims to redefine the traditional boundaries of game development by introducing an intuitive and collaborative approach, where the game designer can engage in a fluid dialogue with the system, driving real-time modifications and enhancements to the game code [4].

II. Motivation

The motivation behind the Chat to Game Generator project stems from a desire to revolutionize the conventional methods of 2D game generation by introducing a more intuitive and interactive approach. Traditional game development often relies on hard-coded instructions, a method that can be restrictive and time-consuming, requiring meticulous manual input for even the smallest modifications. Recognizing the limitations of this approach, the project aims to explore and compare alternative methods to 2D game generation.

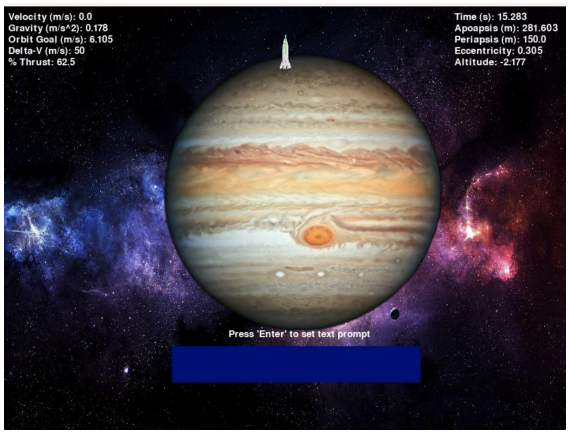

Figure 1: A screenshot of the Planet Orbiter game contained in the 'TextGame.py' file found in the 'Hardcoded Games' folder. This game utilizes the hard-coded instructions generation method.

Three Game Generation Methods:

- Hard-coded instructions
- OpenAI interface
- Dynamic code editing

The desire is for a cohesive system that integrates the power of ChatGPT seamlessly into the game development environment, providing developers with an interactive platform where they can not only generate code through conversation but also witness the immediate results of their creations without the need for additional manual steps. Such a program would bridge the existing gap, further enhancing the efficiency and convenience of game development powered by natural language processing tools like ChatGPT.

The OpenAI interface represents a pivotal component of this initiative, tapping into the power of natural language processing to facilitate a dynamic conversation between developers and the game engine. By integrating OpenAI's advanced language model, developers gain the ability to generate game content through prompt-based interactions, streamlining the creative process and reducing the need for explicit coding.

Furthermore, the project introduces dynamic code editing, a method that allows for the modifications to the game's underlying code structure through a user-text interface. This approach not only allows for rapid iteration but also empowers developers to experiment with different game mechanics and elements without the need to exit the interface. Through these methods, the project aims to demonstrate the feasibility of a prompt based 2D game engine. This project seeks to emphasize efficiency, creativity, and a more accessible development experience while keeping the reliance on external libraries to a minimum.

III. Related Research

In the realm of natural language processing, my research delves into the capabilities of ChatGPT, a language model developed by OpenAI. Founded in 2015 by Elon Musk and Sam Altman, OpenAI has been at the forefront of advancing artificial intelligence, and ChatGPT represents a significant milestone in the field. This AI operates by pre-training on extensive text data in an unsupervised manner, honing its ability to

predict the next word in a sequence. This pre-training process allows ChatGPT to acquire a deep understanding of language patterns, syntax, grammar, and semantics, forming the foundation for its subsequent applications.

The transition from GPT-1 to GPT-4 underscores the evolution of these language models. GPT-1, with its 117 million parameters, demonstrated impressive proficiency across various natural language processing tasks. However, GPT-4, as a large multimodal language model, represents a leap forward by accepting both image and text inputs, showcasing the model's enhanced capacity to process and generate outputs in diverse formats.

My project is intricately connected to this progression, leveraging the language generation capabilities of ChatGPT for the interactive and prompt-based development of 2D games. By incorporating the insights gained from ChatGPT's pre-training and fine-tuning processes, my project aims to offer a novel and accessible avenue for game developers to shape their creations through natural language interactions, aligning with the cutting-edge advancements in language modeling.

Another lesser known AI model, Alpaca, an open-source language model developed by researchers at Stanford University, seeks to replicate the capabilities of ChatGPT but in a more efficient manner. Rooted in Meta's Llama architecture, Alpaca stands out as a tool designed for ethical and accessible language model development. While not available for commercial use, Alpaca opens its doors to small businesses, offering a valuable resource for chatbot development. Alpaca presents a unique opportunity for researchers and developers to harness its capabilities without the constraints of expensive hardware concerns [6].

In comparison tests against ChatGPT, Alpaca demonstrated its prowess by emerging victorious 90 times out of 179 trials, particularly excelling in tasks related to email creation, social media, and productivity tools. This competitive performance underscores Alpaca's potential as a formidable player in the language modeling landscape. In essence, Alpaca's applicability in real-world scenarios, such as aiding researchers in ethical AI and cybersecurity, adds a layer of practicality to its capabilities.

For instance, its utility in scam detection showcases its potential to contribute meaningfully to critical areas of societal concern. As my project is deeply rooted in language model integration, insights from Alpaca's strengths and applications contribute to the broader understanding of how language models, beyond ChatGPT, can play a pivotal role in shaping innovative and ethical solutions for interactive game development [5].

IV. Technical Code Overview

The Chat to Game Generator Interface project is a Python-based 2D game engine that leverages the power of natural language processing through ChatGPT for real-time video game editing. The base game libraries include Pygame for graphics rendering, Math for mathematical computations, and Numpy for efficient array operations. Additionally, the project utilizes the OpenAI library for interfacing with ChatGPT, and the system is compiled in the command prompt using Python 3.11.5.

Base Game Libraries:

- Pygame
- Math
- Numpy
- Random (already included)

ChatGPT Libraries:

- Sys (already included)
- OpenAI

The game generation process involves three distinct methods: hard-coded instructions, the OpenAI interface, and dynamic code editing. The folder titled ‘Hardcoded Games’ contains the hard-coded generator’s main program and its accompanying code. The ‘Dynamically Coded Games’ folder houses the OpenAI and dynamic code generator programs.

Hard-Coded Instructions:

In the hard-coded instructions method, the default base game is the Planet Orbiter, which offers text-based user input to edit existing planets and introduces procedurally generated planets and background elements. This method allows developers to interactively shape the game environment using natural language input.

OpenAI Interface:

The OpenAI interface method utilizes player input to send prompts to ChatGPT, leveraging the OpenAI library to process and incorporate the model's outputs directly into the Python game. This dynamic interaction enables the generation of game content through a conversational interface, offering a unique and flexible approach to game development.

Dynamic Code Editing:

The dynamic code editing method employs the game Pong as the default base game. The main program stores the existing game code as a string. Functions and if-then statements within the code string search for variables, allowing for real-time modification of the game's behavior and mechanics. This method provides a more hands-on and code-centric approach for developers, offering direct control over the underlying structure of the game through dynamic editing.

V. Hard-Coded Instructions

The hard-coded instructions game generation method is exemplified through the execution of the 2D physics-based Planet Orbiter game using Pygame. The game is organized into modular programs, each with a specific role in handling different aspects of the game. The 'Images.py' module manages the display of planet and background images, 'Orbits.py' traces the orbit of the spacecraft, 'Planet.py' defines the properties of each generated planet object, and the primary logic is housed in the 'TextGame.py' main program file.

Hard-Coded Generator Programs:

- Images.py: handles planet and background images
- Orbits.py: traces the orbit of the spacecraft
- Planet.py: represents each generated planet class object
- TextGame.py: the main program file

Within the main program, distinct sections are dedicated to managing planet and spacecraft variables, executing a while running loop, handling spacecraft running variables, and performing physics calculations. The hard-coded instructions come into play when altering the properties of the generated planets. Each planet object, instantiated from the 'Planet.py' file, is stored in a list of planets, which is then accessed in sequential order. The spacecraft's orbital parameters are initially based on the first planet in the list [1].

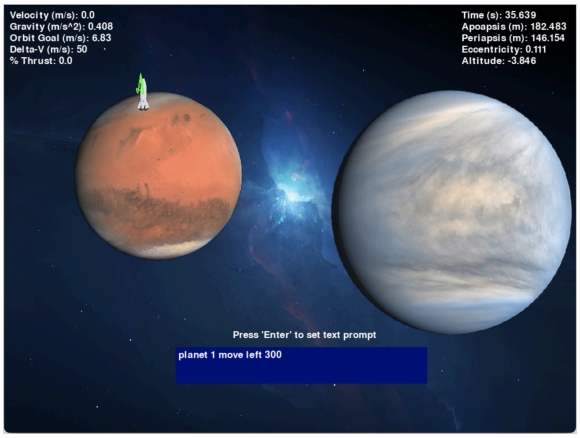

Figure 2: Another screenshot of the 'TextGame.py' file. This runthrough of the hard-coded instructions orbiter game includes a command to move the first planet 300 units to the left.

Main Program sections:

- Planet and spacecraft variables
- While running loop
- Spacecraft running variables
- Physics calculations
- Hard-coded planet alteration instructions

List of hard-coded instructions:

- Move planet up, down, right, or left
- Alter radius of planet
- Alter mass of planet

The hard-coded alteration instructions, defined in the main program, are then systematically applied to each planet in the list. These instructions include commands to move a planet in different directions (up, down, right, or left), alter the radius of a planet, and adjust its mass. This modular and systematic approach enables developers to reference planets sequentially as planet 1, 2, 3, and so forth, streamlining the process of implementing changes to the game environment. By encapsulating these hard-coded instructions, the project ensures a structured and accessible means for developers to interactively modify the attributes of the planets within the game [2].

VI. OpenAI Interface

The OpenAI Interface generation method in the Chat to Game Generator Interface project relies on the OpenAI library to facilitate a dynamic conversation with ChatGPT. The interaction is orchestrated through functions within the program, including the 'return_code' variable and the 'TextChatCode()' function. This design ensures a seamless transition between the OpenAI interface and the generated game, and also allows the user to view and navigate the code generated by the API.

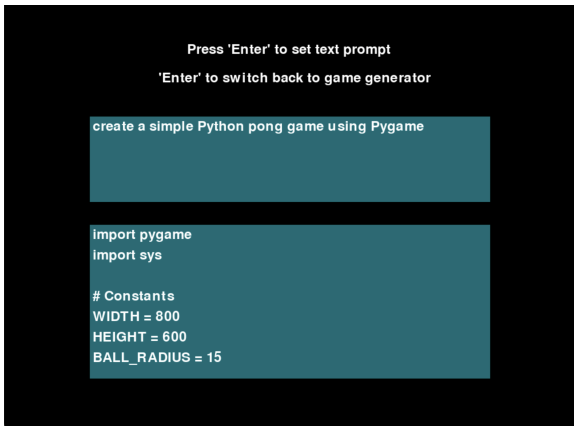

Figure 3: When one compiles the 'TextInput.py' file, one will encounter the text-input window above. The text-box above contains a command to prompt the ChatGPT API interface to create a Pong game.

Functions in Program:

- Return_code variable
- TextChatCode()
- Is_valid_code()
- Text box user input

Accompanying Programs:

- Invalid.py: invalid code program
- BaseGame.py: default test game

TextChatCode Function:

As shown in Figure 3, the TextChatCode() function uses an API key to obtain an AI generated game from ChatGPT. This function takes in the user text input string as an argument, and returns an AI generated game as its 'response' output. The response output's mutable string format allows for direct modification of the game code [4].

```
def TextChatCode(prompt, model="gpt-3.5-turbo"):
    try:
        response = openai.Completion.create(
            engine=model,
            prompt=prompt,
            max_tokens=100,
            n=1,
            stop=None,
            temperature=0.5,
        )

        message = response.choices[0].text.strip()
        return message

    except openai.error.RateLimitError:
        # Handle rate limit error
        invalid.set_error("OpenAI API rate limit reached.")
        invalid.run("TextInput.py") # current game file as
    except Exception as e:
        # Handle other exceptions
        invalid.set_error("An error occurred: {str(e)}")
        invalid.run("TextInput.py") # current game file as
```

Figure 4: The TextChatCode() function above is separated into two main segments. The first part of the function uses the OpenAI API to generate new code. The second part uses a series of exception blocks to generate a black screen with an error message like the one in Figure 5.

Return Code Variable:

The 'return_code' is a string variable that contains a line of code that allows the generated game to switch back to the 'TextInput.py' file upon pressing the 'exit' button on the upper left. The 'return_code' string is appended to the end of the 'response' string code obtained through ChatGPT. The newly modified response string is then passed into the exec() command, which executes the code contained in the string.

IsValidCode Function:

The purpose of the 'Is_valid_code()' function lies in its essential role in code validation. This function uses a try-catch exception to check the syntax and structure of the generated code to ensure it is error-free and ready for execution. The function returns an 'error_str' string variable that represents the output of the try-catch exception. The string is then used as an argument for the 'Invalid.py' program.

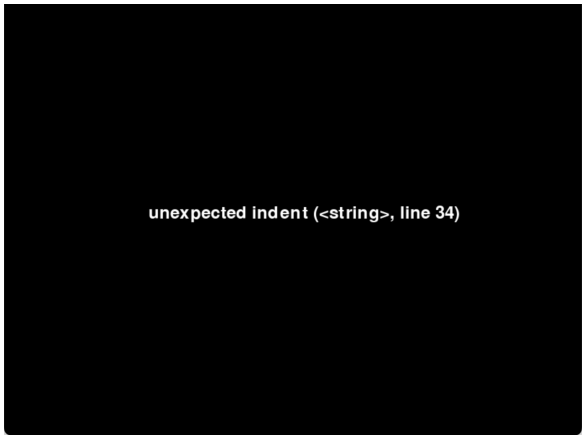

Figure 5: This runthrough of the 'Invalid.py' program mentions an unexpected indent error. This is one example of an invalid window that the Invalid class displays.

Invalid Code Program:

To prevent the possibility of error messages crashing the program, an accompanying program named 'Invalid.py' is introduced. This program contains a class, 'Invalid,' with functions such as 'Set_error()' to define the error message string and 'Run()' to execute a Pygame instance displaying the error message string. In the event that the generated game contains invalid code, the 'Invalid' class is invoked, preventing the program from crashing and instead providing a user-friendly display of the error message on the screen. This proactive approach to error handling enhances the robustness of the OpenAI Interface method,

ensuring a more stable and user-responsive game development environment.

```
# invalid game screen
from Invalid import InvalidCode
invalid = InvalidCode()

# saves invalid error message
error_str = ''
```

Figure 6: The import statement above is needed to import and utilize the 'Invalid.py' program class.

VII. Dynamic Code Editing

The Dynamic Code Editing game generation method in the Chat to Game Generator Interface project introduces a mutable string representation of the existing Pong game. The program comprises essential functions, including the 'Return_code' variable, the 'Old_code' variable representing the Pong game, and the 'Is_valid_code()' function for syntax validation. The 'Text box user input' function captures user instructions through a string variable called 'user_text,' enabling developers to interactively edit the game.

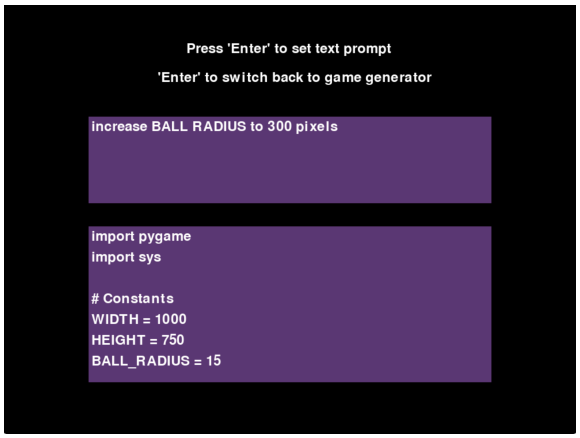

Figure 7: When one runs the 'DynamicGame.py' program, one finds a text-interface similar to the one found in 'TextInput.py'. A command to increase the radius of the ball to 300 pixels is being typed into the user-interface.

Functions in Program:

- Return_code variable
- Old_code (Pong) variable
- Is_valid_code()
- Text box user input
- Game editing instructions

Accompanying Programs:

- Invalid.py: invalid code program
- BaseGame.py: default test game

```
for i, line in enumerate(lines):
    # checks for upper case characters
    match = re.search(r"([A-Z]+)", line)
    print(match)
    if match and user_int != '':
        char_combination = match.group(1)
        # check if the same character combination exists in user_text
        if char_combination in user_text and (user_int is not None) and len(line) < 21:
            lines[i] = re.sub(r'\b\d+\b', user_int, line, 1)
```

Figure 8: The for-loop above searches for instances of words that are in all caps in the Pong.Game.py program. This loop also searches for an exact capitalized match in the user input box.

Game Editing Instructions Workflow:

The dynamic code editing process involves appending the 'Return_code' variable to the base Pong game before any modifications, establishing a baseline for the code execution. Instructions in the dynamic code encompass a variety of modifications, such as moving rackets, changing the color of elements, altering velocities, adding or modifying variables, and more. The user's input, encapsulated in the 'user_text' variable, guides the modifications. The for-loop features in Figure 8 searches for the words that are in all caps. That same loop searches for a matching set of capitalized words in the user input box. The output text box below the input box allows users to view the code before and after the modifications are implemented [3].

Invalid Code Program:

To bolster the robustness of the dynamic code editing method, an accompanying program named 'Invalid.py' is employed.

Like in the OpenAI program, in the event that an invalid code change is produced, the 'Invalid.py' program is invoked to display an error message on the screen, preventing potential crashes and enhancing the user experience by providing clear feedback on the nature of the error.

Pong Default Game:

The default game for the dynamic code generation engine is the Pong game, which is loaded as a string and subsequently modified as needed. This design choice provides a starting point for developers to experiment with dynamic code editing, showcasing the versatility of this method in real-time game development scenarios.

```
# reads contents of default Pong game
file_path = "BaseGame.py"
old_code = ''
with open(file_path, "r") as file:
    old_code = file.read()
```

Figure 9: When one selects the 'Enter' key in the 'DynamicGame.py' file, the Pong game appears in a new window. The Dynamic Game program imports the 'PongGame.py' file to use as its default.

IIX. Technical Challenges

As of the current stage of the project, the hard-coded game generator demonstrates reliability, providing a stable and effective method for interactive 2D game development. The OpenAI game generator has proven functional for the majority of tasks, effectively harnessing the power of natural language processing to facilitate dynamic conversations with ChatGPT. However, there are areas where improvements could enhance its performance and usability.

Current Code Progress:

- Hard-coded game works reliably.
- Dynamic code editor as of now is confined to small scale edits.
- OpenAI is limited by number of API paid uses. Has a few errors that emerge under certain conditions.

The Dynamic code editor program has made notable progress in editing code based on keywords, offering a more flexible approach to game development. However, challenges have been encountered when attempting large-scale edits, revealing an opportunity for refinement and optimization. One area in which the program requires improvement is the ability to insert new functions. As of now, the Dynamic code editor can only alter variables with names listed in all-caps.

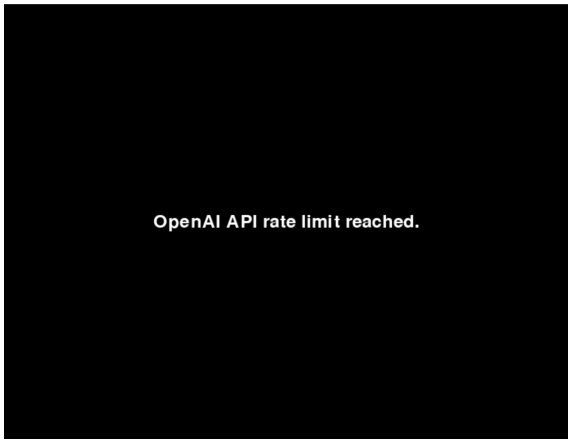

Figure 10: The error message above appears when one reaches the limit on the amount of calls to ChatGPT's API.

The OpenAI game generation program requires more refinement when compared to the other two programs. While the ChatGPT API does provide a versatile and simple to use method to generate game code, this system's usability is limited by one's monetary budget. Aside from the program's reliance on a paid API plan, the OpenAI game generation method also suffers from a few errors which emerge under certain conditions.

OpenAI Glitches:

- 'is_valid_code()' function catches non-existent 'no-reference' errors.
- The window occasionally displays main program instead of game.

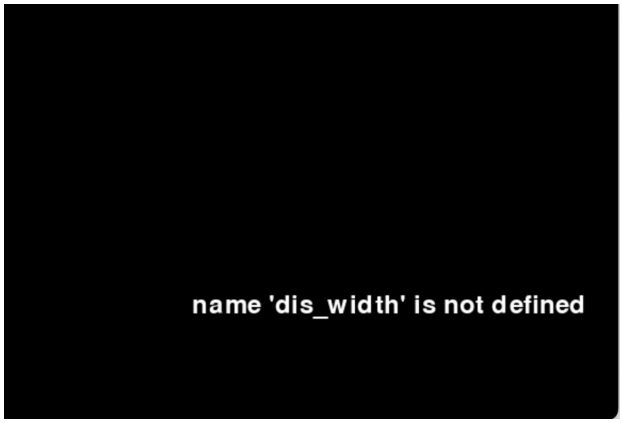

Figure 11: The try-catch exception in the 'is_valid_code()' function mistakenly labels 'dis_width' as a non-existent variable.

When one prompts the TextInput.py file's OpenAI program to generate a game that likely has the majority of its gameplay code within multiple 'for event' loops, the try-catch exception within the 'is_valid_code()' will mistakenly labels global variables as non-existent variables. Figure 11 illustrates the error screen that emerges when one prompts the OpenAI program to generate the game Snake. The only known remedy for this error is to first prompt the program to create the game Pong, and then generate the desired game.

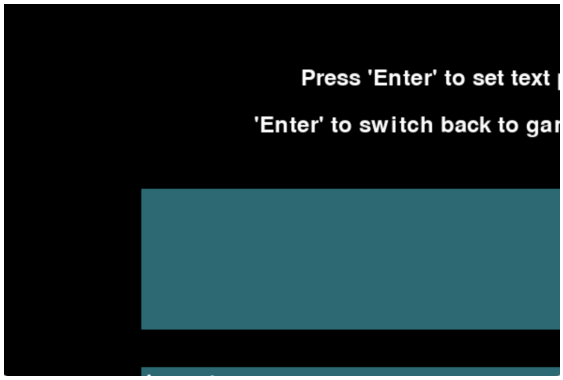

Figure 12: The OpenAI program attempts to generate the desired game at the appropriate resolution. However, the program instead displays the main program at the game's updated resolution.

Another less common error occurs when the OpenAI API is able to generate the desired game, but fails to properly display the game. The cause of this glitch may pertain to the structure of the gameplay for loops in the main program's 'ModifiedTextGame.py' save file. However, the pattern that points to the cause of this issue is less than obvious. The only thing that remains certain is that the main program's process is somehow interfering with the generated game's process as illustrated in Figure 12.

The aforementioned errors in the OpenAI Interface program have causes that will take a considerable time to identify. But once those causes have been rooted out, the OpenAI game generator will achieve its potential as the most versatile of the three natural language processing game generation methods.

IX. Future Work

Looking ahead, the future work prospects for the Chat to Game Generator Interface project involve an expansion into 3D game generation, adding a new dimension to the interactive and dynamic development environment. This expansion will be

facilitated through the integration of a combination of API interfaces and dynamic code editing. Leveraging the existing capabilities of ChatGPT, the project aims to seamlessly transition from 2D to 3D game generation, providing developers with an additional dimension of versatility.

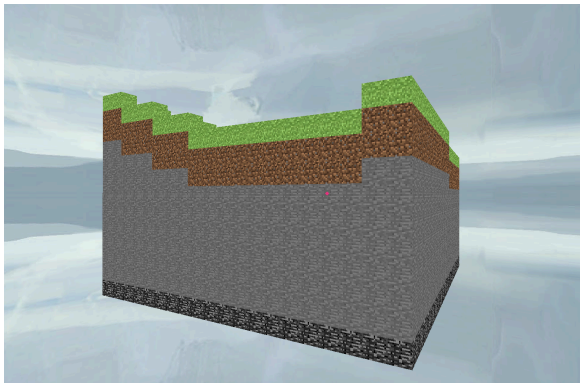

Figure 13: The screenshot above represents a scaled down 3D Minecraft clone created using Python's Ursina game library.

The incorporation of 3D Python game libraries such as Ursina, and the PerlinNoise library, will introduce powerful tools for graphics rendering and procedural content generation, enhancing the visual and creative aspects of the generated games. Additionally, the OpenAI library will continue to play a pivotal role, facilitating natural language interactions and prompt-based game development in the expanded 3D context.

The incorporation of parallel computing, particularly multithreading will allow for real-time code editing, as well as for the simultaneous display of the edited game. The incorporation of multithreading will require identifying the cause of lower level runtime errors. But once these errors have been rectified, the incorporation of parallel computing and 3D game design will ensure that the Chat to Game Generator Interface remains at the frontier of the ever-evolving world of interactive game design [7].

```
2024-01-22 10:57:59.571 python3[1547:50362] *** Terminating app due to uncaught exception 'NSInternalI
nconsistencyException', reason: 'NSWindow drag regions should only be invalidated on the Main Thread!'
*** First throw call stack:
(
        0   CoreFoundation                      0x00007ff803cf018a __exceptionPreprocess + 242
        1   libobjc.A.dylib                     0x00007ff80381642b objc_exception_throw + 48
        2   CoreFoundation                      0x00007ff803d182f6 -[NSException raise] + 9
        3   AppKit                              0x00007ff806d39e7e -[NSWindow(NSWindow_Theme) _postWin
dowNeedsToResetDragMarginsUnlessPostingDisabled] + 307
        4   AppKit                              0x00007ff806d441af -[NSView setFrameSize:] + 2109
        5   AppKit                              0x00007ff806d826fe -[NSTitlebarView setFrameSize:] + 8
6
        6   AppKit                              0x00007ff806d54091 -[NSView setFrame:] + 347
        7   AppKit                              0x00007ff806d826a1 -[NSTitlebarView resizeWithOldSuper
viewSize:] + 95
        8   AppKit                              0x00007ff806d647fb -[NSView resizeSubviewsWithOldSize:
] + 499
        9   AppKit                              0x00007ff806d43f26 -[NSView setFrameSize:] + 1460
        10  AppKit                              0x00007ff806d654f0 -[NSTitlebarContainerView setFrameS
ize:] + 146
        11  AppKit                              0x00007ff806d54091 -[NSView setFrame:] + 347
        12  AppKit                              0x00007ff806d64ec8 -[NSView resizeWithOldSuperviewSize
:] + 694
        13  AppKit                              0x00007ff806d647fb -[NSView resizeSubviewsWithOldSize:
] + 499
        14  AppKit                              0x00007ff806d43f26 -[NSView setFrameSize:] + 1460
        15  AppKit                              0x00007ff806d62c83 -[NSThemeFrame setFrameSize:] + 464
        16  AppKit                              0x00007ff806d6230f -[NSWindow _oldPlaceWindow:fromServ
er:] + 831
        17  AppKit                              0x00007ff806d60da2 -[NSWindow _setFrameCommon:display:
fromServer:] + 3089
        18  libSDL2-2.0.0.dylib                 0x0000000110808ff4 Cocoa_SetWindowSize + 324
        19  libSDL2-2.0.0.dylib                 0x0000000107d1dd5 SDL_SetWindowSize_REAL + 197
        20  display.cpython-311-darwin.so       0x0000000110647412 pg_set_mode + 1842
        21  python3.11                          0x000000010fcb6dd4 cfunction_call + 52
        22  python3.11                          0x000000010fdac095 _PyEval_EvalFrameDefault + 238821
        23  python3.11                          0x000000010fc5289f _PyFunction_Vectorcall + 511
        24  python3.11                          0x000000010fdb2f9f _PyEval_EvalFrameDefault + 267247
        25  python3.11                          0x000000010fc5289f _PyFunction_Vectorcall + 511
        26  python3.11                          0x000000010fc587d4 method_vectorcall + 452
        27  python3.11                          0x000000010fea67fb thread_run + 235
        28  python3.11                          0x000000010fe321d4 pythread_wrapper + 36
        29  libsystem_pthread.dylib             0x00007ff803b9d1d3 _pthread_start + 125
        30  libsystem_pthread.dylib             0x00007ff803b98bd3 thread_start + 15
)
libc++abi: terminating due to uncaught exception of type NSException
Abort trap: 6
(base) MacBook-Pro-3:Dynamically Coded Games Chris$
```

Figure 14: The error above serves as an example of a complex runtime error caused by an attempt to multithread the TextInput's main OpenAI program, and the program's generated game.

X. Conclusion

In conclusion, the Chat to Game Generator Interface project has made significant strides in offering developers an innovative and interactive platform for 2D game generation. The three methods employed—hard-coded instructions, the OpenAI interface, and dynamic code editing—each contribute unique strengths and considerations. While the hard-coded game generation method is the most reliable, it is deemed the least versatile among the three. The OpenAI interface stands out as the most versatile option, leveraging natural language processing for dynamic and expansive game development. However, its dependency on a paid API plan poses a limitation.

Dynamic code editing emerges as a compelling approach, allowing developers to edit existing code and providing greater versatility than hard-coded instructions. Importantly, this method offers a flexible solution that can be utilized without the need

for a paid API plan, addressing cost concerns and making it more accessible to a broader audience. However, it's worth noting that dynamic code editing does not leverage OpenAI for game generation, indicating a trade-off between versatility and the use of advanced language models.

Looking forward, the project's future planned expansions include a transition into 3D game generation, which will utilize a combination of API interfaces and dynamic code editing. This inventive approach reflects the commitment to staying at the forefront of technological advancements in game development, ensuring that the Chat to Game Generator Interface continues to evolve and meet the dynamic needs of developers in the ever-evolving landscape of artificial intelligence and interactive game design.

References

[1] L. George, “Chapter 10 – Orbital Perturbations, Introduction to Orbital mechanics”, *Oer.pressbooks.pub*, p.17-29, 2015.

[2] B. Weber, “Orbital Nomenclature”, *Orbital-mechanics.space*, p.1-4, 2021.

[3] A. Chetanjha, “How to create a text input box with Pygame?”, *Geeksforgeeks.com*, p.2-5, 2023.

[4] S. McKay, “How to Use ChatGPT for Python: The Ultimate Guide”, *Blog.enterpriseDna.co*, p.1-7, 2023.

[5] R. Taori. et al, “Alpaca: A Strong, Replicable Instruction-Following Model”, *CRFM Stanford University*, p.3-13, 2021.

[6] P. Pratim Ray, “ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope”, *Sikkim University*, p.5-16, 2023.

[7] P. Amland, “Ursina 6.1.2”, *pypi.org*, p.1-3, 2023.