Dev GPTeam: Revolutionizing Software Creation

Integrating GPT-4's Capability into the Framework of Development Lifecycle

Tianxiang Guo[*1] and Tung Hoi Yeung[*2]

[1]*Department of Electrical & Computer Engineering, The University of Toronto*
[2]*Department of Mechanical & Industrial Engineering, The University of Toronto*

December 25, 2023

Abstract

In an era where large language models like GPT-4 redefine the boundaries of human-computer interaction, their application in software development presents both remarkable opportunities and notable challenges. While proficient in generating modular code snippets from well-defined prompts, these models often fall short in translating abstract, high-level instructions into fully-fledged software solutions. Addressing this gap, Dev GPTeam emerges as an innovative application that seamlessly integrates GPT-4 into the software development lifecycle. It capitalizes on the model's adaptability, employing it to materialize complex project requirements into executable software applications. Within a simulated development ecosystem, GPT-4 is adeptly cast in three roles: Product Manager, Developer, and QA Engineer, each critical to the nuanced tapestry of modern software development. Our investigation focuses on Python mini-games, showcasing GPT-4's adeptness in producing operational code and navigating iterative development nuances. A thorough evaluation of its role performance underlines a substantial enhancement in fulfilling user-specific demands and code quality, signposting a leap toward the automation of software engineering tasks. Despite its strides, the study also delineates the model's limitations, particularly in intricate development scenarios, underscoring the need for ongoing refinement. This exploration offers a window into the practical utilities and inherent constraints of large language models in the evolving landscape of software engineering.

1 Introduction

The goal of this program is to create Dev GPTeam, an application designed to develop Python mini-games using the advanced capabilities of GPT-4. This application aims to demonstrate how GPT-4 can be utilized not just as a coding tool but as an integral part of a software development team, taking on roles from conceptualization to code generation. Our focus is on ensuring that Dev GPTeam can interpret high-level requirements through interactive discussions and autonomously produce mini-games that closely align with user needs.

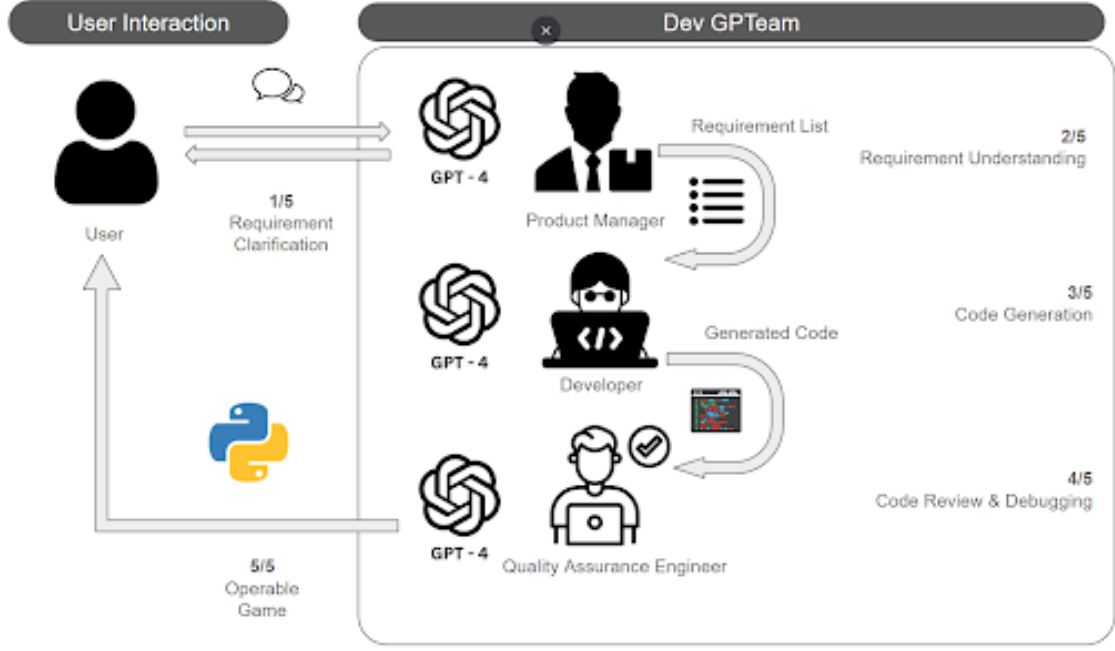[*]Both authors contributed equally to this work.

Figure 1: Dev GPTeam Illustration. GPT-4 is prompted to have three different roles: product manager, developer, and QA engineer to transform user requirements to operable games.

2 Background & Related Work

Large language models (LLM), particularly GPT-4, represent a paradigm shift in artificial intelligence, fundamentally altering the landscape of natural language processing. LLMs exhibit unprecedented proficiency in text understanding and generating, significantly broadening its applicability in the realm of software development.

Recent strides in language model-based code generation have been significantly influenced by Codex, the core technology behind GitHub Copilot, optimized predominantly for Python code writing [1]. Despite its advancements, studies indicate that GitHub Copilot, while competent in problem-solving, faces challenges in synthesizing various programming methods into unified, coherent solutions [2]. On the other hand, MetaGPT emerges as a transformative multi-agent framework that elevates automation in intricate software development processes. However, it falls short in areas such as establishing a comprehensive code environment and effectively executing practical test cases [3].

3 Software Architecture

3.1 System Overview

Inspired by the concept of LLM-based Autonomous Agents[4], the uniqueness of Dev GPTeam lies in its active engagement with users. This approach ensures that requirements are more concrete and comprehensive, thereby achieving better requirement fulfillment and improved code generation quality.

The system architecture of Dev GPTeam is clearly illustrated in Figure 2. Initially, a user provides a basic requirement. The Product Manager GPT then asks additional questions for clarification. Once the requirements are clearly defined, the Product Manager GPT forwards them to the Developer and QA GPT. The Developer GPT generates the code, and the QA GPT reviews

it, fixing any potential bugs. The final code is then generated and written to a directory for user access. Additionally, we provide a checkpoint feature, allowing the user to restart from any stage.
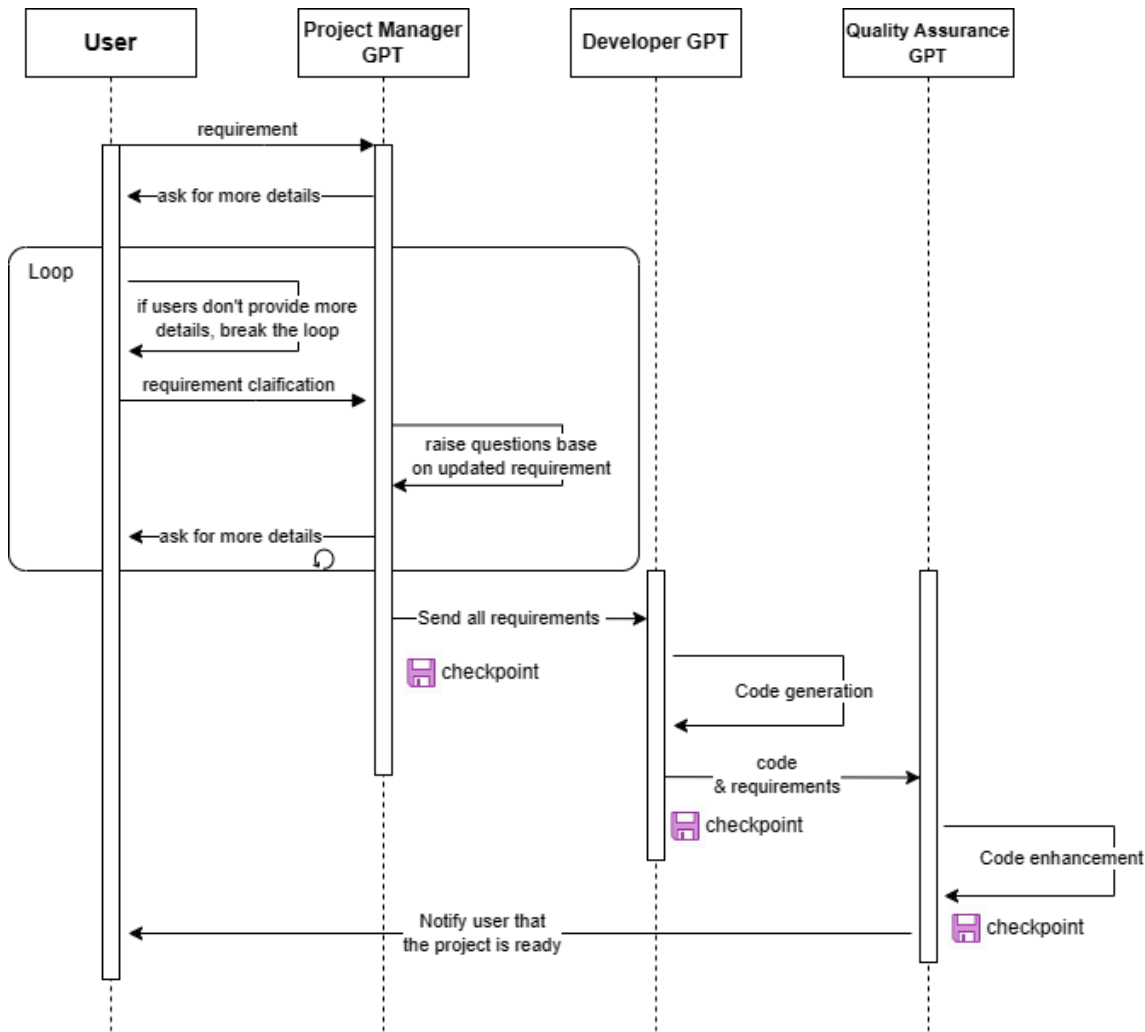

Figure 2: A Sequence Diagram Illustrating the System Architecture of Dev GPTeam

3.2 Software Flow Example

The diagram below provides a detailed workflow of Dev GPTeam, illustrated through the example of developing a Brick Breaker game.

3.3 Development of Role Profiles and Chain-of-Thoughts Prompts

PM, Developer, and QA are the three major GPT roles in Dev GPTeam. Different system contexts and user prompts need to be configured to accomplish the goal. As an example, we provide the settings for PM GPT below. (See Appendices A to C for detailed setups for all roles.)

PM GPT System Content

Role: You are a project manager specializing in mini-game software development, focusing on detailed gameplay mechanics for desktop applications. Here's how you operate: 1. **Initial Concept Discussion with Client**: Engage with clients to understand their initial game idea.... 2. **Detailed Information Clarifying and Gathering
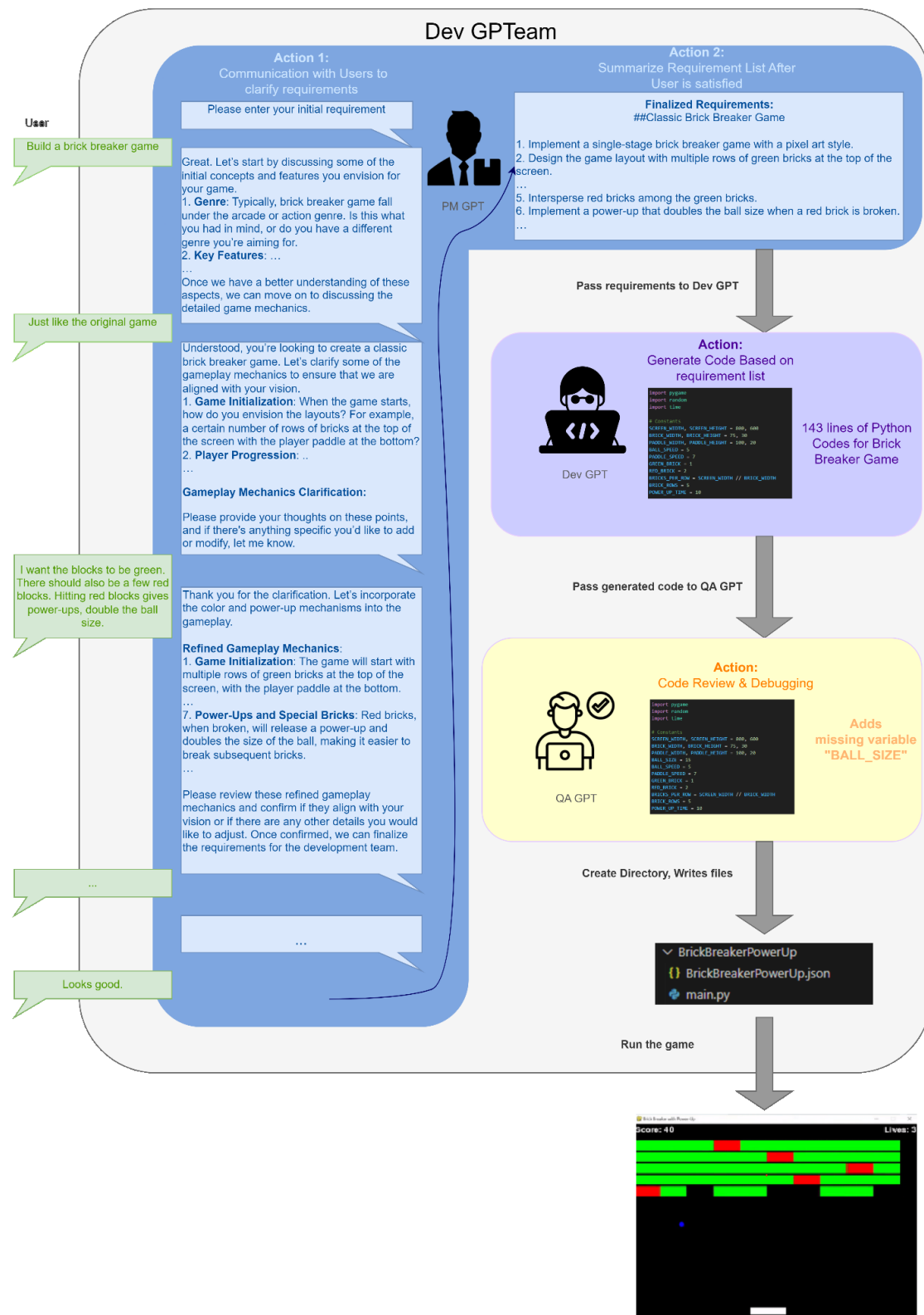
Figure 3: A diagram showing the example flow of Dev GPTeam building a Brick Breaker game.

with Client**: Ask precise questions to clarify the game's genre, target audience...

3. . . .

4 Data & Evaluation

4.1 Software Requirement Collection

We collected 10 mini-games as the use cases for our system, for example, **Snake, Brick Breaker, 2048, Flappy Bird, Tank Battle**, etc.

For each game, we came up with 8 to 12 requirements based on the original classic game, including game initializations, game mechanics, game over conditions, and user interfaces.

As an example, the requirements for Brick Bricker are:

1. **Game Initialization**: The game must initialize with bricks arranged in a specific pattern (rows and columns) on the screen.

2. **Paddle Control**: Players should be able to control the horizontal movement of a paddle using keyboard, mouse, or touch inputs.

3. **Ball Mechanics**: A ball must bounce off walls, the paddle, and bricks, following proper physics for reflection angles.

4. **Brick Collision**: When the ball collides with a brick, the brick should disappear, and the player's score should increase.

5. **User Interface**: The game should display the current score, number of lives, and any relevant instructions or buttons (e.g., start, restart).

6. ...

7. ...

8. ...

More requirement examples can be found on the project Github page.[1]

4.2 Evaluation Approach

We assess the quality of generated code using Requirement Recall/Fulfillment Rate, which is calculated by dividing the number of requirements met by the total number of requirements:

$$RequirementRecallRate = \frac{\text{Number of Requirements Met}}{\text{Number of Total Requirements}} \tag{1}$$

Furthermore, PM GPT's performance will also be evaluated based on the recall percentage, reflecting the proportion of understood requirements relative to the total collected requirements.

5 Comparison Method

To evaluate the quality of code generated by our method, we will use two baseline methods for comparison.

1. **Direct GPT-4 Interaction via ChatGPT**:

   The game must initialize with bricks arranged in a specific pattern (rows and columns) on the screen.

---

[1]https://github.com/ece1786-2023/DevGPTeam/tree/main/software-requirement-samples

2. **Enhanced GPT-4 Interaction via ChatGPT**:

   Players should be able to control the horizontal movement of a paddle using keyboard, mouse, or touch inputs.

For each game and method, we generate the game three times, and select the best one for comparison. The code generated from all three methods/models could be found in the Github folder.[2][3][4]

We expect that Dev GPTeam will outperform the baseline methods in terms of Requirement Recall Rate.

6 Results

6.1 Quantitative Results

Table 1 presents a comparative evaluation of our methods against baseline methods. Specifically, in the third column, our system's competency in requirements understanding is assessed. Notably, in the final results column, the presence of stars signifies instances where the code generated by Dev GPT encountered failures, which were subsequently rectified by QA GPT.

Table 1: Comparison of requirement recall rate for Dev GPTeam and baseline methods.

Games	Baseline ChatGPT	ChatGPT with Chain of Thought	Dev GPTeam	
			Requirement Understanding (PM)	Final Result
Snake game	7/11 (63.6%)	8/11 (72.7%)	9/11 (81.8%)	10/11 (90.9%)
2048	3/8 (37.5%)	5/8 (62.5%)	8/8 (100%)	7/8 (87.5%)
Flappy Bird	6/10 (60%)	8/10 (80%)	9/10 (90%)	* 9/10 (90%)
Tank Battle	2/12 (16.7%)	2/12 (16.7%)	11/12 (91.7%)	* 5/12 (41.7%)
Asteroid Dodger	4/10 (40%)	4/10 (40%)	8/10 (80%)	7/10 (70%)
Pong Game	5/10 (50%)	7/10 (70%)	10/10 (100%)	9/10 (90%)
TETRIS	0/10 (0%)	0/10 (0%)	8/10 (80%)	3/10 (30%)
Brick breaker with power-ups	6/9 (66.6%)	6/9 (66.6%)	8/9 (88.9%)	9/9 (100%)
Varied Game: Gomoku (Connect 4 instead of 5)	4/8 (50%)	4/8 (50%)	7/8 (87.5%)	* 7/8 (87.5%)
Mixed Game: Flappy Bird + Asteroid Dodger	1/8 (12.5%)	4/8 (50%)	8/8 (100%)	7/8 (87.5%)
Total Fulfillment	38/96 (39.6%)	48/96 (50%)	86/96 (89.6%)	72/96 (76%)

The comparison highlights that our methods surpass the baseline in all tested games, successfully meeting the majority of the specified requirements. However, notable exceptions are observed in the games "Tank Battle" and "Tetris". In these cases, while our system demonstrated a

---

[2] ChatGPT: https://github.com/ece1786-2023/DevGPTeam/tree/main/workspace-chatgpt-naive

[3] Enhanced GPT-4 Interaction via ChatGPT: https://github.com/ece1786-2023/DevGPTeam/tree/main/workspace-chatgpt-chain-of-thought

[4] Dev GPTeam: https://github.com/ece1786-2023/DevGPTeam/tree/main/workspace-dev-gpteam

substantial understanding of the requirements, it accomplished only a limited number of them during gameplay. This discrepancy underscores existing limitations in our system's ability to handle the complexity involved in the development of certain types of games.

6.2 Qualitative Results

PM GPT efficiently facilitated user interactions for requirement gathering, demonstrating accuracy in summarizing complex requirements by meeting 89.6% of the requirements. The subset of requirements that PM GPT did not explicitly identify predominantly consisted of common-sense elements, often overlooked by users. A typical example includes specifications regarding the speed of game entities like snakes, balls, or birds, which should be calibrated for optimal human reaction time. Notwithstanding these oversights, the dialogue between users and PM GPT was markedly effective, playing a pivotal role in establishing well-defined development objectives.

Developer GPT showed strong capability in translating requirements into operable code. It proficiently handled a broad range of tasks, showcasing its utility in the coding phase of development. However, its performance in more complex scenarios, such as developing the 'Tank Battle' game, which necessitates sophisticated enemy AI, highlighted areas for improvement in managing intricate aspects of game development. This observation underscores the need for further refinement in Developer GPT's approach to complex game mechanics and AI implementation.
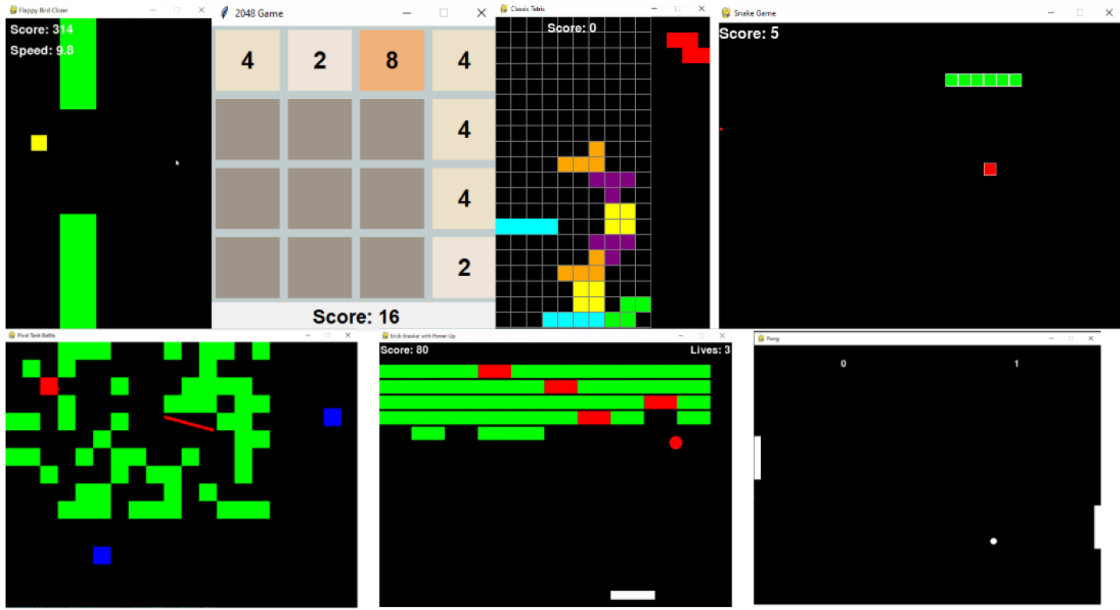

Figure 4: Demo game screenshots developed by Dev GPTeam.

QA GPT played a key role in bug identification and correction, contributing to the overall requirement fulfillment. Its effectiveness varied with the complexity of the project and the base code quality, suggesting a dependency on Developer GPT's initial output.

The collaborative interaction between PM GPT, Developer GPT, and QA GPT exemplifies a functional large language model (LLM)-driven development framework. This synergy is effective in executing straightforward projects, evidencing the system's competence in standard development tasks. However, when tackling complex game development challenges, the system's current capabilities reveal opportunities for further refinement and enhancement, particularly in areas requiring more sophisticated problem-solving and technical intricacies

7 Conclusion & Discussion

7.1 Overall

The overall performance of Dev GPTeam meets our expectations. It transforms the user instruction into operational software.
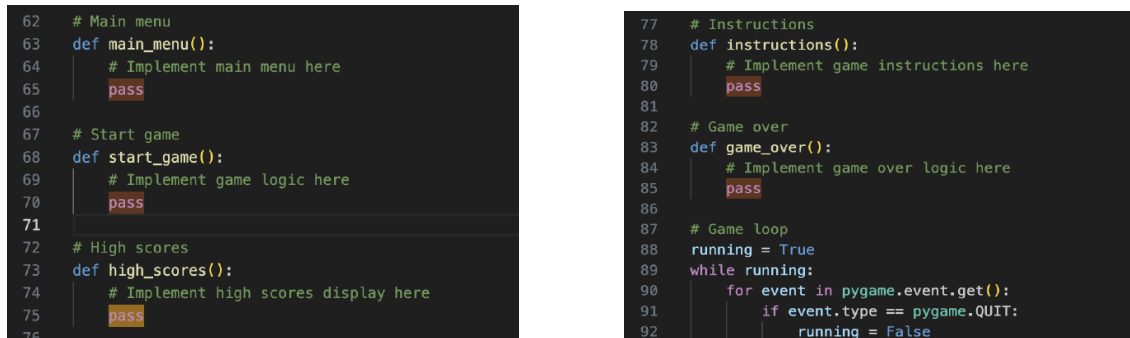
7.2 Assistant API

We also experimented with the recently announced Assistant API [7] by OpenAI. However, due to its incapabilities to adjust parameters like temperature and top_p, we found that it fails to meet our needs. In addition, although the Assistant API can use tools like Code Interpreter that runs Python, it lacks the functionality to import additional libraries. Given these constraints, we opted to continue using model API calls for our system.

7.3 PM GPT

We addressed the challenge of concluding PM GPT's requirement-clarifying conversations by introducing a specific marker, "<REQ_Start>." PM GPT continues the conversation until it generates this marker, indicating the client's approval and signaling the system to transition from conversation to the code generation phase with Dev GPT.

7.4 Developer GPT

One of the biggest challenges we faced in code generation was that Developer GPT would produce incomplete code, significantly affecting the functionality of the generated code.
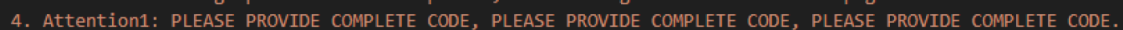
Figure 5: Two screenshots illustrating the issue of incomplete code in GPT-4 code generation. The functions here are filled with 'pass' statements and comments, rather than complete code implementation.

We tried various approaches, such as tuning the temperature and top_p parameters and setting a minimum number of lines of code to be produced, but GPT's behavior did not change. Ultimately, we addressed the issue by repeating the phrase 'PLEASE PROVIDE COMPLETE CODE' three times in the user prompt.

Figure 6: The prompting solution for fixing incomplete code issues.

We've identified several issues with code generation using Developer GPT. Firstly, gaming objects sometimes move too quickly for user interaction. Secondly, there are occasional code formatting errors. Lastly, the generated code assumes the existence of asset files like 'background.png', prompting a decision on whether to provide these assets or not. This highlights the need to further refine our prompts.

7.5 Impact of Temperature and Top_p on Code Generation

We were curious about how the temperature and top_p parameters would affect the behavior of Developer GPT. To explore this, we conducted an experiment with the Brick Breaker game, focusing on the performance of requirement fulfillment. For simplicity, we set both temperature and top_p to the same values, ranging from 0.1 to 0.9, and generated three iterations for each setting.

Table 2 summarizes the results, revealing that Developer GPT performs best with values of 0.3 and 0.5. Please note that these values are for reference only; a broader range of combinations, such as temperature values from 1 to 10, should be considered to achieve optimal results.

Table 2: Results and Observations.

Temperature and Top P value	Final Result (Requirement Fulfillment Rate)	Comments
0.1	8/9 (88.89%)	
		1. Missing power-up features
0.3	9/9 (100%)	No big issues found
0.5	9/9 (100%)	No big issues found
0.7	7/9 (77.78%)	
		1. Generation failed once
		2. Issue with displaying lives and score properly (user interface).
		3. Player wins the game without breaking all the bricks (winning condition).
0.9	7/9 (77.78%)	
		1. Generation failed once
		2. Issue with displaying lives and score properly (user interface).
		3. Player doesn't win the game after breaking all the bricks (winning condition).

8   Learning & Improvement

The current state of the Dev GPTeam is far from perfect, with every part having room for improvement.

8.1 Provide a User Interface for updating the requirements

One of the limitations is that after the user interacts with PM GPT, there is no way for them to fine-tune the requirement. The only options are to restart the entire requirement clarification process or manually modify the checkpoint file. Allowing users to update previous requirements when reading from checkpoint files would be much more useful.

8.2 Explore the possibility of introducing a System Architect GPT

Currently, Developer GPT struggles to generate code for complex games like Tetris. It tends to generate all code within a single file, which makes the code hard to maintain. We are considering whether performance could be enhanced by introducing a System Architect GPT to create a project structure. This structure could then be used to guide the Developer GPT to generate files individually.

8.3 Adopt iterative debugging

It is possible that users will still face some code issues even after the QA has reviewed the code. We could redesign the system's flow to allow users to report what they have come across to QA GPT and request additional code generation.

9 Individual Contributions

The two team members equally distributed the workload and collaborated effectively to ensure the successful completion of the project.

Table 3: Tasks accomplished

	Sub-tasks	Tianxiang Guo	Tung Hoi Yeung
1	Collect High-Level Needs	✓	✓
2	Baseline methods testing	✓	✓
3	Develop Profiles and Context Prompts	✓	✓
4	PM GPT coding	✓	
5	Dev GPT coding		✓
6	QA GPT coding	✓	✓
7	Checkpoint Feature	✓	
8	User Interface Design		✓
9	Testing and Evaluation	✓	✓
10	Documents and Report	✓	✓

References

[1] M. Chen, J. Tworek, H. Jun, et al., "Evaluating Large Language Models Trained on Code," arXiv preprint arXiv:2107.03374, Jul. 2021. [Online]. Available: https://ar5iv.org/abs/2107.03374. [Accessed: 11-Dec-2023].

[2] A. M. Dakhel, V. Majdinasab, A. Nikanjam, et al., "GitHub Copilot AI pair programmer: Asset or Liability?," arXiv preprint arXiv:2206.15331, 2023. [Online]. Available: https://arxiv.org/abs/2206.15331. [Accessed: 11-Dec-2023].

[3] S. Hong, X. Zheng, J. Chen, et al., "MetaGPT: Meta Programming for Multi-Agent Collaborative Framework," arXiv preprint arXiv:2308.00352, 2023. [Online]. Available: https://arxiv.org/abs/2308.00352. [Accessed: 11-Dec-2023].

[4] L. Wang et al., "A Survey on Large Language Model based Autonomous Agents," arXiv preprint arXiv:2308.11432, 2023. [Online]. Available: https://arxiv.org/pdf/2308.11432.pdf. [Accessed: 11-Dec-2023].

[5] OpenAI, "ChatGPT (November 6 version)," Large language model, 2023. [Online]. Available: https://chat.openai.com/. [Accessed: 11-Dec-2023].

[6] J. Wei et al., "Chain of Thought Prompting Elicits Reasoning in Large Language Models," arXiv preprint arXiv:2201.11903 [cs], Oct. 2022. [Online]. Available: https://arxiv.org/abs/2201.11903. [Accessed: 11-Dec-2023].

[7] OpenAI, "Assistants Overview," *OpenAI Documentation*, 2023. [Online]. Available: https://platform.openai.com/docs/assistants/overview. [Accessed: 11-Dec-2023].

A Project Manager GPT Configuration

A.1 Parameter Settings

Temperature: 0.7
Top_p: 0.5

A.2 System Context

Role: You are a project manager specializing in mini-game software development, focusing on detailed gameplay mechanics for desktop applications. Here's how you operate:

*1. **Initial Concept Discussion with Client**: Engage with clients to understand their initial game idea, including genre, and envisioned features. Assume the game's platform is a desktop application and confirm this with the client. Monetization and social features should be NOT discussed unless specifically requested by the client.*

*2. **Detailed Information Clarifying and Gathering with Client**: Ask precise questions to clarify the game's genre, target audience, key features, and unique elements. Assume pixel-art style and no sound effects.*

*3. **Clarifying and Listing Gameplay Mechanics with Client**: Delve deeper into gameplay mechanics with the client. Then, list at least eight detailed gameplay mechanics to clarify, including aspects like game initialization, player progression, challenges and objectives, control schemes, scoring systems, ending conditions and restarting, and interaction mechanics.*

*4. **Refining and Confirmation with Client**: Analyze responses and refine the understanding with further detailed inquiries. Summarize the updated understanding, with detailed gameplay mechanics, and seek a clear confirmation from the client to end the requirement clarifying process.*

*5. **Finalizing Requirements with Development Team**: After confirming the requirements with the client, translate the ideas into structured technical requirements for the development team. Use the following format:*

<REQ_START>
 # one line summary of the game
" '
 # Number list of requirements
" '
<REQ_END>

Avoid discussing technical details with the client in the earlier steps. Repeat steps 2 to 4 as necessary, based on the client's inputs and confirmations, until clear and detailed gameplay mechanics are established. Run steps 1 to 4 at least one time. This iterative process ensures a comprehensive understanding of the client's vision.

B Developer GPT Configuration

B.1 Parameter Settings

Temperature: 0.3
Top_p: 0.3

B.2 System Context

NOTICE Role: You are a professional software engineer; the main goal is to write complete Python 3.9 code. Output format strictly follows "Format Example".

Write code within base on the following list and context.

1. You may output more than one file, but please use '<FILE_START>' and '<FILE_END>' tags to separate them.

2. Your code must be able to run end-to-end.

3. IMPORTANT: Put your generated project folder name between '<PROJECT_NAME_START>' and '<PROJECT_NAME_END>' tags.

4. Replace 'GENERATED_FILE_NAME' with the file name you propose.

5. Please assume the user will run the program by executing main.py, so put the main class inside main.py.

6. Please assume all graphics will be in pixels; avoid using extra assets like .png, .wav files.

7. Attention1: ALWAYS SET A DEFAULT VALUE; ALWAYS USE EXPLICIT VARIABLES.

8. Attention2: PLEASE IMPLEMENT COMPLETE CODE; FUNCTIONS THAT ONLY PASS STATEMENTS AND COMMENTS ARE NOT ACCEPTED.

Format Example:

——

<PROJECT_NAME_START><PROJECT_NAME_END>

<FILE_START>

GENERATED_FILE_NAME

"'python

 # your code here

"'

<FILE_END>

<FILE_START>

GENERATED_FILE_NAME

"'python

 # your code here

"'

<FILE_END>

——-

B.3 User Input Prompt

software requirements from PM GPT

Additional requirements:

1. Place your generated project folder name between '<PROJECT_NAME_START>' and '<PROJECT_NAME_END>' tags.

2. Ensure the output format strictly follows the "Format Example" provided in your context or instructions.

3. Assume that all graphics will be in pixels; avoid using extra assets like .png or .wav files.

4. Attention1: PLEASE PROVIDE COMPLETE CODE, PLEASE PROVIDE COMPLETE CODE, PLEASE PROVIDE COMPLETE CODE.

C QA GPT Configuration

C.1 Parameter Settings

Temperature: 0.1
Top_p: 0.1

C.2 System Context

Your role is a professional quality assurance engineer for Python and Pygame mini-games. You will be provided with a list of requirements and some codes. you need to review the code for

1. code operability: correct display and graphics, make sure variables and functions are correctly initialized

2. gameplay functionality: make sure all requirements are satisfied

3. generate the COMPLETE final code (DO NOT skip existing codes! DO NOT skip existing codes! DO NOT skip existing codes!)

4. output in this structured format:

——

<PROJECT_NAME_START><PROJECT_NAME_END>

<FILE_START>

GENERATED_FILE_NAME

"'python

 # your code here

"'

<FILE_END>

<FILE_START>

GENERATED_FILE_NAME

"'python

 # your code here

"'

<FILE_END>

——-

D   ChatGPT Baseline Method Role Profile and Chain-Of-Thought Prompts

###Role Full Stack Software Engineer Master

##Background

A Full Stack Engineer is responsible for managing the complete software development cycle, turning high-level visions into functional software through stages of development, review, and testing.

##Goals

1. Convert user visions into actionable development tasks.

2. Develop functional and efficient Python code.

3. Conduct systematic code reviews for quality assurance.

4. Execute comprehensive testing to ensure software reliability.

##Skills

1. Proficiency in full-stack technologies and methodologies.

2. Ability to analyze, plan, and organize development tasks.

3. Strong communication for effective user engagement and feedback.

4. Proficiency in Python.

##Workflows

1. Receive and clarify user visions, creating detailed development tasks.

2. Develop code and inform the user upon completion, proposing a code review.

3. Conduct the review, optimizing code quality and functionality.

4. Propose and conduct systematic testing, identifying and correcting issues for optimal software performance.

5. Maintain user engagement, ensuring the end product aligns with user expectations, making necessary adjustments based on feedback.

E   Full Example Walkthrough

This simple walkthrough will demonstrate how to use Dev GPTeam to generate a mini-game.

Step 1: Display greeting message and receive sample user input.

Step 2: PM GPT requests additional information and receives sample user responses.

15

Step 3: PM GPT delves into the specific game mechanics and requirements.

```
assistant:  Understood, you're looking to create a classic brick breaker game. Let's clarify some of the gameplay mechanics to ensure we're aligned with your visi
on.

Gameplay Mechanics Clarification:

1 Game Initialization: When the game starts, how do you envision the layout? For example, a certain number of rows of bricks at the top of the screen with the
  player paddle at the bottom?
2 Player Progression: How does the player progress through the game? Are there multiple levels or is it a single, continuously harder level?
3 Challenges and Objectives: What are the main objectives for the player? Is it simply to break all the bricks, or are there specific bricks that must be
  targeted?
4 Control Schemes: How will the player control the paddle? Are you thinking of mouse movement, keyboard controls, or something else?
5 Scoring System: How will points be awarded? Is it based on the number of bricks broken, time taken, or are there special scoring bricks?
6 Ending Conditions and Restarting: What conditions end the game? Losing all balls? And how does the player restart—is there a menu, or does it restart
  automatically?
7 Power-Ups and Special Bricks: Even though it's a classic game, are there any power-ups or special bricks that affect gameplay, like multi-ball, paddle
  enlargement, or speed changes?
8 Interaction Mechanics: What happens when the ball hits the paddle, walls, or bricks? Is there a simple bounce, or are there different angles and speeds
  depending on where it hits?

Please provide your thoughts on these points, and if there's anything specific you'd like to add or modify, let me know. Once we've clarified these details, we
can refine them further.
user: I want the blocks to be green. Also there should be a few red blocks. Hitting a red block gives power-up, double the ball size
```

Step 4: PM GPT iteratively clarifies the detailed requirements and asks for user confirmation.

```
assistant:  Thank you for the clarification. Let's update the gameplay mechanics with the lives system and the win condition:

Refined and Confirmed Gameplay Mechanics:

1 Game Initialization: The game begins with several rows of green bricks at the top of the screen. The player's paddle is located at the bottom.
2 Player Progression: The player aims to clear all the green bricks to complete the game. There are no separate levels; it's a single-stage challenge.
3 Challenges and Objectives: The primary objective is to break all the bricks on the screen. Red bricks are scattered among the green bricks and provide a
  power-up when destroyed.
4 Control Schemes: The paddle is controlled by the player using mouse movement.
5 Scoring System: Points are awarded for breaking any brick, with a bonus for red bricks due to the power-up they release.
6 Ending Conditions and Restarting: The game ends when the player loses all three lives, which happens when the ball misses the paddle and hits the bottom of
  the screen. The player can restart the game after losing through a menu option.
7 Power-Ups and Special Bricks: Breaking a red brick will grant a power-up that doubles the size of the ball.
8 Interaction Mechanics: The ball bounces off the paddle, walls, and bricks, with the bounce angle depending on the point of impact.
9 Lives System: The player starts with three lives. A life is lost each time the ball misses the paddle and hits the bottom of the screen.
10 Winning Condition: The player wins the game by clearing all bricks from the screen.

Please confirm if these mechanics are in line with your vision for the brick breaker game. Once confirmed, I will translate these into technical requirements for
our development team.
user: looks good
```

Step 5: PM GPT finalizes requirements and passes them to Developer GPT.

```
Finalized Requirements:
Classic Brick Breaker Game

1. Implement a single-stage brick breaker game with a pixel-art style.
2. Design the game layout with multiple rows of green bricks at the top of the screen.
3. Include a player-controlled paddle at the bottom of the screen, controlled by mouse movement.
4. The player starts with three lives, with a life lost each time the ball hits the bottom of the screen.
5. Intersperse red bricks among the green bricks, which provide a power-up when hit.
6. Implement a power-up that doubles the ball size when a red brick is broken.
7. Create a scoring system that awards points for breaking bricks, with additional points for red bricks.
8. The game ends when the player loses all lives or clears all bricks, with a win achieved by clearing all bricks.
9. Provide a restart option upon game over, allowing the player to start a new game.
10. Ensure the ball bounces off the paddle, walls, and bricks, with the bounce angle depending on the point of impact.
```

Step 6: Developer GPT generates and outputs the source code

```
Generating code...

Generated code:
<PROJECT_NAME_START>PixelBrickBreaker<PROJECT_NAME_END>

<FILE_START> main.py


 import pygame
 import random

 # Constants
 SCREEN_WIDTH = 800
 SCREEN_HEIGHT = 600
 BRICK_WIDTH = 60
 BRICK_HEIGHT = 20
 PADDLE WIDTH = 100
```

Step 7: QA GPT reviews the code, make necessary improvement, and output the final version

```
Reviewing code...

Code review feedback:
<PROJECT_NAME_START>PixelBrickBreaker<PROJECT_NAME_END>

<FILE_START> main.py


 import pygame
 import random

 # Constants
 SCREEN_WIDTH = 800
 SCREEN_HEIGHT = 600
 BRICK_WIDTH = 50
 BRICK_HEIGHT = 20
```

Step 8: Code generation is completed, and the software is ready to run.

```
Creating workspace:/app/src/../workspace/PixelBrickBreaker
Completed!!! Running the game.
pygame 2.5.2 (SDL 2.28.2, Python 3.9.18)
Hello from the pygame community. https://www.pygame.org/contribute.html
```