**Introduction of our project:**

Our project prototype is to design an AI control Player in shapez game which can automatically operate to complete tasks in the game. It analyzes the game state in real-time to automate tasks such as resource management and conveyor belt optimization along with some tools(cutter, trash, rotator) to find efficient paths to pass the levels of the game.

This project is valuable and meaningful as it enhances the player's experience by reducing the need for repetitive actions, making the game more enjoyable. It’s also innovative because the AI not only optimizes in-game operations but also learns and adapts to various strategies through machine learning, creating a more intelligent solution. The automation provides a smarter way to engage with the game, freeing players from tedious manual tasks.

**Justification for Prototype Implementation Choices:**

For this project, we generally divided it into three parts: Game Development, Automated Game Playing Management, and AI Agent Design, with all three parts serving the ultimate goal of enabling the AI to play the game.

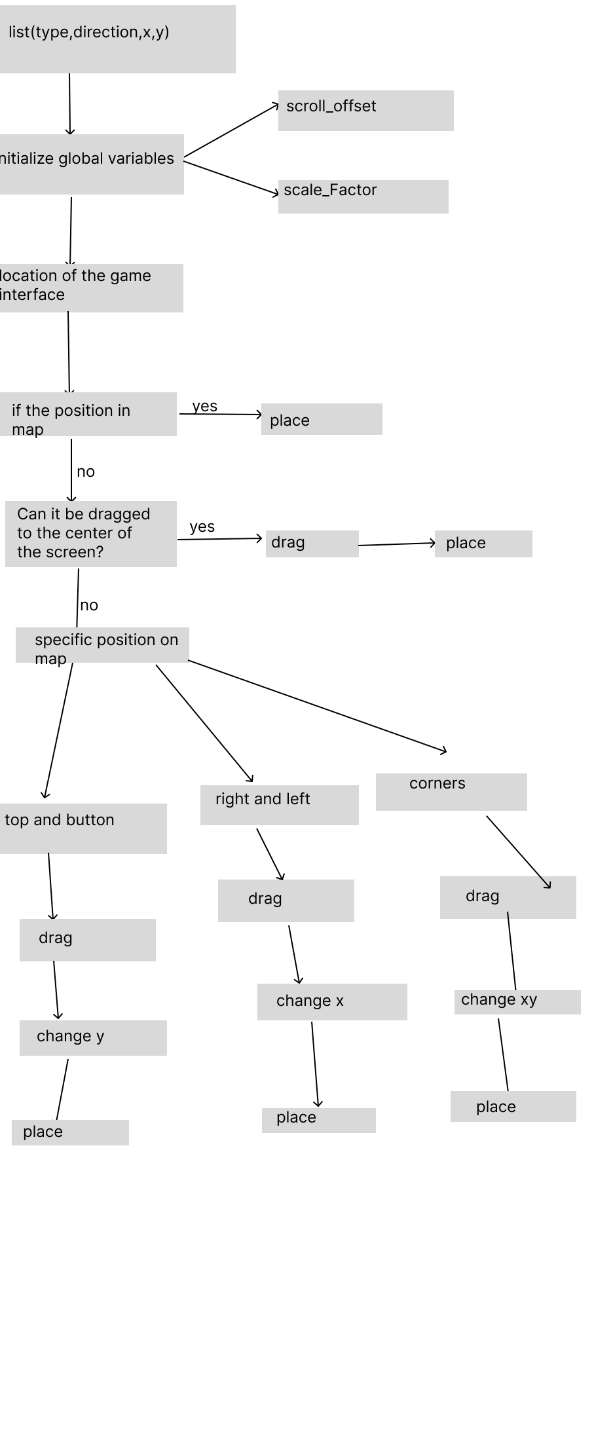
**Game**

To design the game, we utilized a resource code from git hub ,the URL is <https://github.com/jiangqianyu/MyShapez>, Github which is based on Qt Creator. Based on that, we have revised several .cpp files and .h files to fix the original bugs and add new features and functionalities. We also tried to do memory mapping in the game environment so that we can connect the game to the AI agent. To describe the game implementation carefully, we first modified the Playscene.cpp because it has many bugs like game crash and belt placing problem. For example, the array overreach in the function WhichBeltImg() can cause game crash. After dealing with bugs and ensure the game running properly, we then added the zoom function and drag functions so that player can zoom out/in or move to everywhere in the gamemap. To achieve this functions, we introduced several global variables like scaleFactor and Scroll\_offset along with built-in qpainter methods like translate() and scale() and we apply these methods to almost all the functions in the Playscene.cpp. After we completed that, we decided to completely refactor this file because it looks disordered since the code is over 1500 lines and over 35 functions in a single file. Now the file looks better with clear implementation logics. After that, we did further implementation and added more features. For example, we created new tools like Rotator to rotate the shapes in the game. We also tried to added more features like adding tool Tunnels and making belt animation in the game, but we do met too much obstacles on that. The gamemap is also developed, we created two maps and the large map is 240 \* 150, by making changes in config.h and gamemap.cpp, with plenty of resources and the zoom and dragging features looks very good. However, to fit the AI agent well, we finally choose the small one , which is 32 \* 20, because the AI agent in the large map will cost too much time to training. We also added game levels like adding game pass condition such as checking transporting speed of all the shapes by adding new functions in Playscene.cpp , hub.cpp and WindowControl.cpp. Finally, we dealt with memory mapping so that the game memory can directly read by AI agent. To achieve that, we used c++ built-in methods such as CreateFileMapping() , MapViewOfFile() and others to help AI know the memory like buildings, directions, speed and other parameters.

**Game Automation part:**

1 We developed the desktop application using PyQt5 for its rich GUI components and cross-platform compatibility, ensuring a responsive UI on Windows, macOS, and Linux. To handle background tasks without blocking the UI, we used QThread, leveraging its signal-slot mechanism for real-time updates. importlib allowed us to dynamically load and execute Python scripts, offering flexibility for users to run any selected file. To manage thread synchronization, we implemented QWaitCondition and QMutex, ensuring safe pausing and resuming of background threads without concurrency issues. We also used the re module to sanitize variable names generated from file paths, and traceback provided detailed error handling with full stack traces for easy debugging.

2 In an AI-controlled game running project, Hanxi Shen and Haotian Yu were responsible for the development of visual recognition scripts. Our task is to enable AI to recognize objects and scenes in games in real time and make decisions accordingly through image processing technology. This work is crucial to the project, because visual recognition is the basis for AI to understand the game environment, and only by accurately recognizing the game footage can the AI correctly judge the current situation and take effective action. The script shows every step of the AI, so people can intuitively understand the decision that the AI made. The accuracy of visual recognition directly affects the decision-making efficiency of AI, so it is an indispensable key link in AI-controlled games.



**AI PART:**

For AI training algorithms, reinforcement learning methods such as DQN, TRPO, and PPO are commonly used to play games. Initially, we planned to design both DQN and PPO models to compare their performance in the game environment. However, due to time constraints, we ultimately only implemented the PPO model, which successfully played the game.

Here’s a brief explanation of how the PPO model works to solve the task. First, we obtain the game map data (i.e. get the current building layout, resource map, and goal) to inform the AI of the starting conditions. Then, we provide the AI with a list of possible actions (e.g., placing a conveyor belt at position (3,3) with the direction set to "right," as implemented in the create\_action\_space function in shapezenv.py). The agent selects an action, calculates its reward, and subsequently adjusts the action probability distribution for that state. After extensive training, the optimal action A for state S will have a higher probability of being selected by the AI. Eventually, the AI converges on an optimal strategy.

In PPO, three key elements are necessary to complete the task:

1.Designing the observation space and action space.

2.Defining the terminal state (i.e., determining when the AI has completed the game).

3.Designing the reward function and other optimizations.

**Observation and Action Space:**

The observation space generally consists of the entire building map, resource map, and the task details. For example, in Task 1, the observation space should include two m\*n arrays representing the building and resource maps, as well as two integers representing the target shape (a circle, for instance) and the total number required to be delivered to the destination.

However, if we set the target as both the shape and the total quantity needed, it complicates not only data transfer but also model training. For instance, when the correct buildings are placed to complete the task, the game requires time to deliver items to the hub, making it difficult to design an appropriate reward function. Moreover, real-time game data transfer is challenging due to the use of memory mapping. As a result, we opted to check whether the building setup meets the task requirements rather than tracking the delivery process.

For example, if the task is to deliver 20 circles to the hub, we only need to verify that there is a valid route from an extractor (which extracts the required circles) to the hub. Similarly, if the task is to deliver a certain number of shapes within 10 seconds, we can calculate the speed of each route and determine how many routes are needed to meet the requirement.

Meanwhile, to accelerate the training of each step, we need to decrease the dimensions of the observation space. Initially we store 3 maps to store the machines’ type, the machines’ direction and the resource map respectively, After we researched a lot, we used a method called “State compression storage” to store the machines information into one map.

We represented the machine state as machine\_type \*100+direction into a 2d map,

For instance,

文本

中度可信度描述已自动生成

Here the buildings’ map means there is a hub on position (3,3), a extractor on position(1,0) with direction Right, a belt with direction Right on position (1,1)

You can find the machines’ type numbers and direction numbers in Machine.py

**Action Space:**

The total action space is vast. For example, with an n\*m size map and 3 types of buildings, each grid will have 3\*4\*n\*m possible actions. In our game environment, there are 5 buildings, and the conveyor belt has 12 directions. (For instance, the "UP-RIGHT" direction moves an item to the grid to its right, but it differs from the "RIGHT" direction because the UP-RIGHT conveyor belt requires the previous machine to be positioned below it, while the RIGHT direction requires the previous machine to be on the left.)

To help the model converge and achieve tasks more efficiently, I applied a technique called "invalid action masking" (see [here](https://sb3-contrib.readthedocs.io/en/master/modules/ppo_mask.html)). This technique reduces the probability of selecting an invalid action A in state S to a very small value, making it nearly impossible for that action to be chosen. Since we need to reduce this probability during both training and prediction, I refactored the relevant function \_get\_action\_dist\_from\_latent from the MultiInputPolicy class.

**Terminal State:**

In this project, determining the terminal state is crucial for assessing whether the task has been successfully completed (i.e., the shape that meets the requirements is sent to the hub). In the game, the terminal state can be categorized into two conditions:

“done”: The task is successfully completed, meaning the AI has achieved the goal.

“truncated”: The tasked failed due to AI reached the maximum number of steps.

To verify whether the task has reached a terminal state, several functions were designed and written to validate if the task has been completed by checking the paths, resource transporting, and final goal state.

1. check\_goal Function:

The check\_goal function is the core function to determine whether the task is finished. It is responsible for checking if the resources from the miner (extractor) have successfully travelled through conveyors, rotators, cutters, and other machines to finally reach the target place (hub) with the correct shape. The general process is as follows:

starting point: The resources are extracted by a miner and transported via conveyors.

Path tracking: Along the path, each machine is checked to ensure the resource is passing through correctly.

Shape verification: Before reaching the hub, the resource may undergo cutting or rotation, and the final shape need to be verified to see if it matches the target shape.

End state check: One the resource reaches the hub; it checks whether the shape matches the target shape. If the shape matches, this route is marked as “done”, which means successfully completed. Otherwise, the route fails.

1. \_track\_path\_with\_rotator Function:

The \_track\_path\_with\_rotator function is designed to handle complicated paths with multiple machines such as conveyors rotators, cutters, and hub. The function is designed to:

Process rotators: The function tracks how many times a shape passes through a rotator and modifies the shape using the process\_rotate function.

Process Cutters: When passing through a cutter, this function helps tracking both the main and side exit paths to ensure that the resource can reach the hub.

End state check: If the resource on the path finally reaches the Hub and the shape is rotated or cut to match the target shape, the task is successful; if the resource fails to match or enter the trash, the path fails.

1. check\_all\_paths Function:

Under certain circumstances, the system needs to ensure that multiple paths are set up correctly. This function is designed to ensure that required number of paths are successfully established. This is particularly important when the task is extremely complicated. Here’s how it works:

Calculating paths: The function determines the number of required paths based on the ratio of requiredSpeed to currentSpeed.

Validating paths: Each path is checked using the check\_goal function to ensure it works successfully.

**Reward Function:**

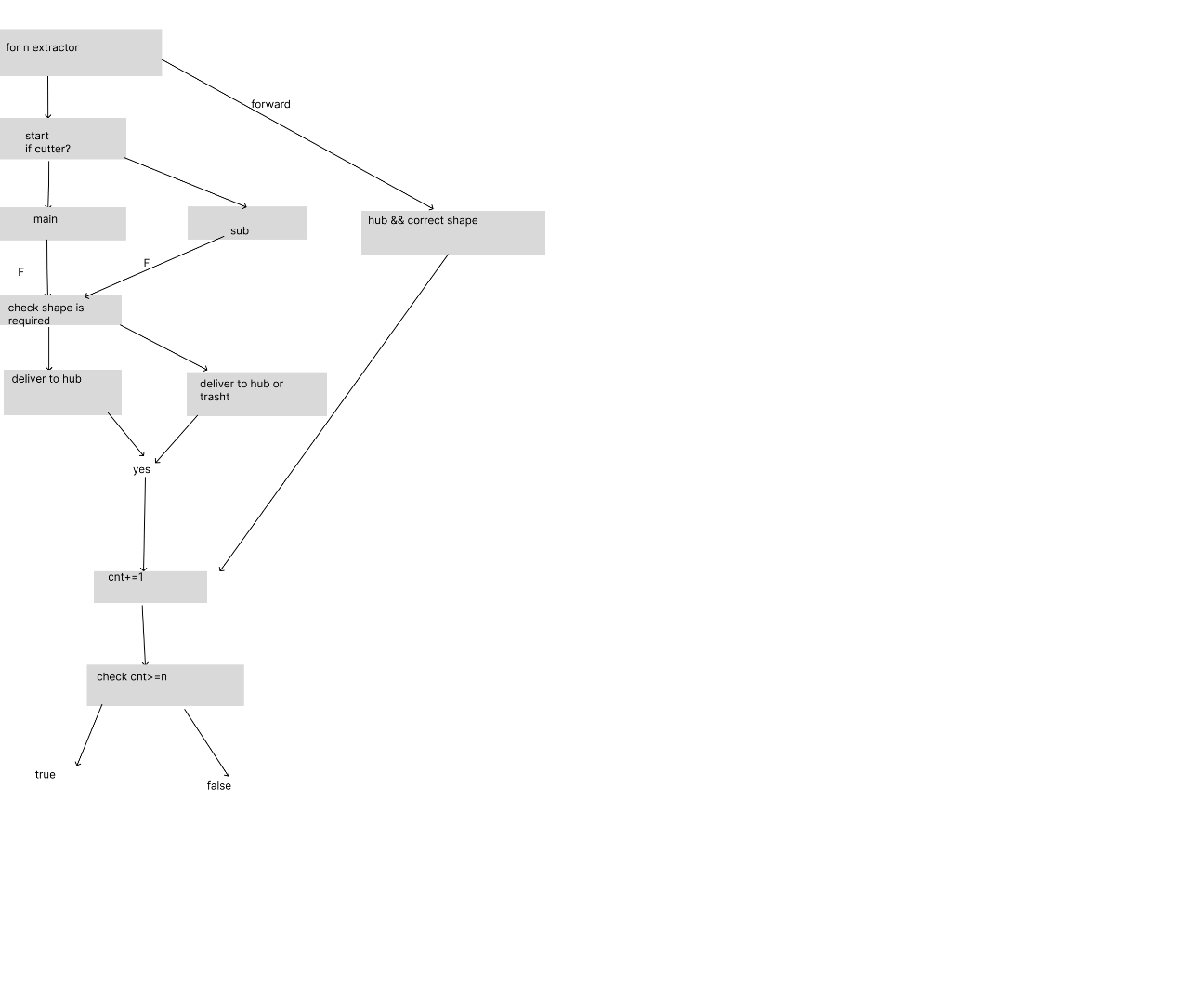
In this project, the reward system is designed to provide positive or negative feedback based on specific actions.

The reward for placing a miner depends on its distance and the direction which is relative to the hub. The closer the extractor is to the hub, the higher the reward, and the distance will be normalized and scaled. In addition, the direction of the miner is considered: if it faces the hub, positive rewards will be given, while a penalty is applied if it faces away. This encourages the strategic placement of miners in the right direction of proximity to the hub and effective resource transporting.

When calculating the reward of placing conveyors, we use the function calculate\_conveyor\_reward. When the AI places a conveyor belt, if it delivering a correct or potentially correct shape, and if it is get closing to the nearest available hub, it will get a positive reward. Otherwise, it will get a negative reward.

The function “calculate\_cutter\_reward” function evaluates whether correct shape will be processed and transported to the hub. If the cutter cut the correct shape (e.g. if the target require a half circle and the cutter cut a full shape into two halves), it will receive a large positive reward, otherwise we give the action a large negative reward compared to the conveying belt.

The calculate\_rotator\_reward function gives a positive reward if a placed Rotator successfully rotates a shape to match the target shape. If the Rotator is placed before the Cutter, where the shape won’t change, a negative reward is applied to discourage inefficient placement.



**Tools and Technologies Used:**

**Game**

In game development, after comparing several platforms or softwares, , we choose to use Qt Creator, because it supports cross-platform application development and easy to connect to our AI agent. Meanwhile, Qt, provides a rich set of libraries and tools that facilitate the creation of high-quality graphics and user interfaces, along with so many methods to implement complex game logics, so it is very suitable for building interactive games. Also, game developers can debug and test games through Qt using built-in debugger without compiling the game. Moreover, we found the basic environment code for shapez in github, although there was a lot of bugs to fixed, it do saved us a lot of time to develop it. Last but not least, we can use MinGW command lines and find dependencies or files to compile and pack the game outside Qt environment after final development.

In order to read the game map data successfully, we used the memory mapping to map the needed variables into a public space, we use CreateFileMapping, MapViewOfFile to share the memory with other process. We also have tried the dynamic library link at first, but as the same virables have different addresses in different process, this method can’t solve the problem.

**Game automation**

For game automation, we utilize the pyautogui library to implement. This Python library is designed for programmatically controlling the mouse and keyboard, making it perfect for automating repetitive tasks within games. After connecting to AI training body, it is easy to perform and simulate human actions. Its straightforward API allows for quick scripting of actions such as clicks, movements, and keystrokes, enabling the creation of automated game scripts.

We use it for three reasons:

Easy to use:

PyAutoGUI is very simple to use, especially for automated operations such as simulating mouse clicks, keyboard input, etc. You can get started quickly without the need for in-depth knowledge of computer vision.

Direct operation screen:

PyAutoGUI is primarily used for automated scripts for image recognition and screen control. It can interact directly with the operating system and automate desktop applications such as clicking, dragging, taking screenshots, keyboard input, and more. This comes in handy for simple game automation.

No need to install additional dependencies:

PyAutoGUI comes with built-in image recognition, which is based on screenshots and does basic pixel matching. For simple image matching needs, there is no need for complex configuration and installation of other vision libraries (e.g., OpenCV).

**AI**

For AI developing, we used the stablebaseline3 library, PPO algorithm and gym environment for this project. The reason why we chose this was stablebaseline3 is one of the most popular library in reinforcement learning, and we don’t need to write the model from the beginning, we only need to refactor a little function to achieve the task and this saved us a lot of time.

Meanwhile gym is also very suitable for the reinforcement learning task, and it is supported by the SB3. Also, it gives us a clear documentation of how to use it, so it is easy to implement and design the agent.

We chose PPO as our AI algorithm to solve the problem.

**Positioning the Prototype in the Competitive Landscape:**

After extensive research, we found that no existing AI model has been developed specifically for playing this game, making our project truly innovative. In addition, we designed a smaller game map compared to the original game, which provides us with more flexibility in AI design. The original game's map is so large that it makes AI training extremely difficult due to the limitations in computing resources. Therefore, the key advantage of our project is that it requires significantly fewer computing resources to train the AI.

Another important advantage is that we have full control over game development. This allows us to ensure that no data is exported to users or third parties, maintaining the confidentiality and security of the game’s internal processes.

**Team Project Management and Collaboration:**

Initially, we divided our team into three groups, each focusing on a specific area: game development, the AI training module, and the game automation module. However, as the project progressed, we realized that communication between different modules was essential. As a result, the team leader needed to be familiar with all parts of the project. This required our team leader to maintain communication with every team member.

Ultimately, Yunpeng Yao took responsibility for game development, while Lianquan Liu designed the front end of the game automation tool and also supported the game development. Haotian Yu and Hanxi Shen primarily focused on developing the back end of the game automation system, and Minjia Wang conducted research and developed the AI agent.

Initially, Yize Xue, our team leader, was meant to focus solely on AI development. However, as the project evolved, we recognized the need for stronger communication between different areas of the project. As a result,the team leader became involved in both game development and AI design.

We also use github as a version control tool to mange different versions of our codes easily, However, since some of us were not quite familiar with this tool, it brought us some difficulty during the collaboration.

**Ethics, Security,and Data Privacy Consideration:**

**Ethical Consideration:** We need to ensure that the AI plays by the same rules as a human player would and that it doesn't engage in behavior that would be considered cheating.

**Data Security:**

Issue: If the AI collects or processes game data, there is a risk that this data could be intercepted, manipulated, or leaked.

Consideration: Implement encryption and secure data handling practices to protect game data and any player-related information the AI might access.

**Potential risks:**

1. **Project management risks:**

If we cannot finish the game environment on time, this will affect project process a lot and as a result, People who are responsible for designing AI for this project may not have enough time to do it.

1. **Resources risks:**

In this project we will use reinforce learning algorithm like DQN to training AI which require a lot of computing resources, if it takes too much time it will lead to a failure.

**Risk matrix:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RISK ID | Risk Description | Likelihood | Severity | Priority | Mitigation Measures |
| 1 | Technical issues may not be resolved in a timely manner | Medium | High | High | Regularly review project progress, ensure timely and effective technical support, and establish fallback technical solutions. |
| 2 | Project progress may be delayed | Low | |  | | --- | | High |  |  | | --- | |  | | Medium | Adopt agile management methods, enhance progress monitoring, and adjust resource allocation as needed. |
| 3 | Insufficient computational resources for AI training | Medium | Medium | Medium | Pre-assess resource needs, prioritize resource allocation for critical tasks. |
| 4 | Major disruptions occur, affecting team members' contributions | Low | Medium | Low | Develop flexible work schedules, enhance team communication and collaboration, and prepare contingency plans. |
| 5 | AI decision-making may lead to unethical outcomes | Low | High | High | Implement fairness checks in AI algorithms, conduct regular ethical reviews |
| 6 | Improper handling of data may lead to privacy violations | Medium | High | High | Comply with data privacy regulations. |
| 7 | Unauthorized access to AI systems may lead to security problems | Low | High | High | Conduct regular security audits and monitor improper attempts |

**Further development:**

**Game**

The basic logics and functionality is complete, however we have made many trials during the implementation process but didn’t make it. The first thing is the belt animation development.

We tried to use \*\*PyQt5\*\* with \*\*QMovie\*\* and \*\*QLabel\*\* to implement the conveyor belt animation. \*\*QMovie\*\* will load the GIF animations, and each conveyor belt's dynamic effect will be displayed via a \*\*QLabel\*\*. Additionally, we will leverage \*\*QMouseEvent\*\* to capture the user's mouse clicks and dragging actions. Upon entering the placement mode, the conveyor belt's animation will update in real-time based on the grid cells the mouse moves over. To ensure smooth animation, we will utilize \*\*QThread\*\* to run the animation update and placement logic in a background thread, preventing the main interface from being blocked. This approach ensures continuous animation in the grid while supporting user interaction and real-time operations.

The second thing is the Tunnel tool development, we tried to develop that by creating two classes, TunnelEntry and TunnelExit. In original game which is published in Steam, when a belt is connecting to a TunnelEntry and another belt connecting to a TunnelExit, the shape on the belt can directly goes from TunnelEntry to TunnelExit,. We do have made the tunnels connected by checking the distance from them, however, we had problem with the shape running display implementation because the previous belt logic and features, if we have more time, we may complete this challenge.

**Game script:**

Front end: The prototype works well but could be improved by enhancing the UI design, making the script execution methods more adaptable, and ensuring better compatibility across different runtime environments.

Back end: The script is extensible in many aspects, mainly reflected in the following key functions:

The first is the adaptive placement feature. As the game interface moves or zooms, the script is able to capture changes in the position of the interface in real time and dynamically adjust the placement coordinates of the building. This feature ensures that buildings are always positioned exactly where they are located on the map, even when the screen moves. This adaptive mechanism solves the problem of misalignment of buildings that can occur when the player moves the game interface, ensuring accurate placement.

The second is dynamic adaptation to environmental changes. Scripts can continuously monitor map information in the game, including the distribution of resources, terrain changes, and updates to free areas. When the game environment changes, the script is able to feed back to the AI in real time, allowing it to adjust the placement strategy of items according to the latest map state. For example, when space in a certain area is occupied, the AI can find a new available area for item placement through a script to avoid building conflicts.

Finally, there is the bulk placement feature. For scenarios that require large-scale placement of the same building, such as a conveyor belt, the script can support batch placement by dragging, which greatly improves the construction efficiency. The AI only needs to delineate the placement area, and the script can automatically lay multiple buildings in the selected area. Not only does this reduce the time required for repetitive operations, but it also makes large-scale construction smoother, especially when placing over long distances, such as laying conveyor belts.

**AI:**

In the Ai environment part, currently we only have 3 buildings to place(extractor, trash, conveying belt, cutter) we also wrote the rotator logic but we didn’t add it into the action space as it will slower the training speed.

For the further development, the developer can add more buildings into the action space to support the whole game. Meanwhile, the reward function can also be improved to accelerate the training.

The observation space can also be one part of the improvement to be considered, when the maps are growing larger, we can simply ignore the most of the map and just focus on the space that close to the hub.

In the AI model side, we can try more hyper parameter (e.g. the learning rate, the clip range) and compare the performance between them.

Reference:

"Proximal Policy Optimization (PPO)." Stable-Baselines3,

https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html#example. Accessed 6th Oct. 2024.

Kauten, J. \*gym-super-mario-bros: Super Mario Bros. for OpenAI Gym\*. GitHub, [https://github.com/Kautenja/gym-super-mario-bros/blob/master/gym\_super\_mario\_bros/smb\_env.py. Accessed 17 Oct. 2024](https://github.com/Kautenja/gym-super-mario-bros/blob/master/gym_super_mario_bros/smb_env.py.%20Accessed%2017%20Oct.%202024).

"Gymnasium API Documentation." Farama Foundation,

https://gymnasium.farama.org/api/env/. Accessed 22nd Sep. 2024.

brn-dev and araffin. \*stable-baselines3: Proximal Policy Optimization (PPO)\*. GitHub, https://github.com/DLR-RM/stable-baselines3/blob/master/stable\_baselines3/ppo/ppo.py. Accessed 6 Oct. 2024.

Microsoft. (n.d.). CreateFileMappingA function (winbase.h) - Win32 apps.

Retrieved September 14th,2024,from <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createfilemappinga>

**PyQt5 Documentation  
PyQt5 is used for creating the GUI components in our project. This official documentation covers widgets, layouts, and threading, which are key elements in our code.  
Available at:** [**https://doc.qt.io/qtforpython-5/**](https://doc.qt.io/qtforpython-5/)**.**

** QThread and Threading in PyQt5  
QThread allows our application to run tasks in the background while keeping the GUI responsive. We can find detailed information about threading utilities like QMutex and QWaitCondition here.  
Available at:** [**https://doc.qt.io/qtforpython-5/api.html**](https://doc.qt.io/qtforpython-5/api.html)**.**

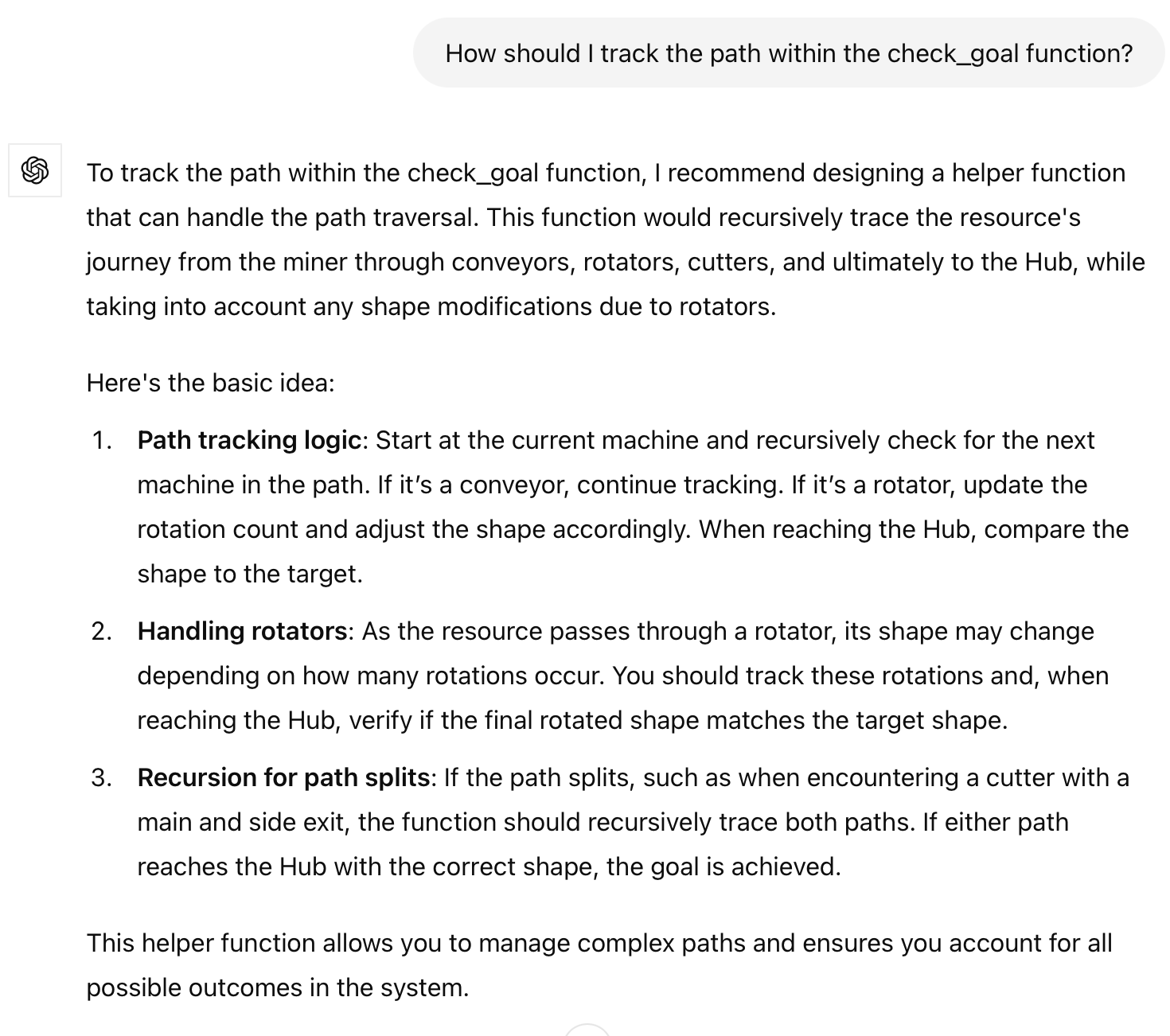
** Signals and Slots in PyQt5  
PyQt5’s signal-slot mechanism is essential for communication between the background thread and the main UI. This document explains how we can declare and use signals and slots.  
Available at:** [**https://doc.qt.io/qtforpython-6/**](https://doc.qt.io/qtforpython-6/)**.**

** importlib for Dynamic Module Loading  
The importlib module allows dynamic loading and running of Python scripts. This is crucial for executing user-selected files in our project.  
Available at:** [**https://docs.python.org/3/library/importlib.html#importing-a-source-file-directly**](https://docs.python.org/3/library/importlib.html#importing-a-source-file-directly)**.**

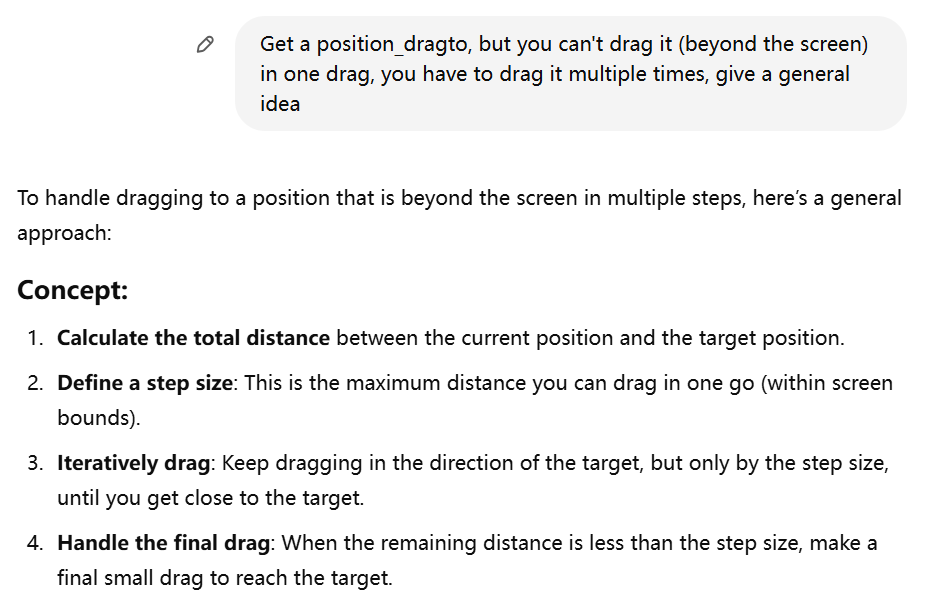
** QFileDialog for File Selection**

Referenced documentation: <https://pyautogui.readthedocs.io/en/latest/>

Appendix:

When designing the check\_goal function, since different complex situations may be encountered on the path, for example, the path may pass through several rotators before cutter, or may pass through several rotors after cutter. At this time, it will become more complicated to design check\_goal, so I asked Chatgpt for some implementation ideas.

When designing the drag function in script, a problem occurs. Since the displayed window is much smaller than the gamemap which is the first version with almost 240 columns and 150 rows, I have to drag multiple times to get the proper position to place tools. I started by figuring out where I wanted to drag and dragged to that position multiple times if I went beyond the screen. But this idea is not conducive to the implementation of the code, because only need to consider the slope of the line from the center point of the game interface to the dragged position, so I asked GPT how to achieve a similar function, and then GPT inspired me to drag the same distance each time when the dragged position is beyond the boundary, and finally drag the remaining distance to completion.



文本

描述已自动生成 文本

描述已自动生成 文本

描述已自动生成 文本

描述已自动生成 文本

描述已自动生成 文本

描述已自动生成 文本

描述已自动生成 文本

描述已自动生成 文本

描述已自动生成 文本

描述已自动生成