After reviewing the code for Dynotate and trying to understand how all the different “switchers” were implemented, I modified the code to implement the AI rule comprehension part into our system and it seemed like the AI could somewhat identify a rule from the input at one point (this was weeks ago when we discussed about rules) until the code was further modified and we had index out of range errors and indentation errors, which sounds simple to solve but it became hard to debug due to extensive changes made in the code with multiple files.

After hours of trying to debug the errors, we decided to remove every line of code we added for the rule recognition and start again. I am reporting now the things we did after restarting, excluding details of some debugging we did in-between.

Our code in the app.py file : https://github.com/ianarawjo/dynotate/blob/pira_1/server/app.py

Function def prompt_injection(user_prompt):

Starting all over again, the first step we took, is add a “RULE” logic in this function to help the AI identify if a rule has been given or mentioned by the user, just like how its done for the other switchers. We added this line : "RULE) Define a rule: The user is trying to define a rule or constraint. Keywords like 'need', 'has to', 'always' are good indicators."

We also added a print to extract a rule given by the user and show it on the terminal with the explanation of why it thinks it’s a rule. And then we tested with Dynotate App if it recognises a rule or anything regarding a rule when we give him: “a dog should always be next to a mouse” as an input. The result is not exactly what we hoped for but we knew we were in the right path this time because we got this as output on the terminal:

```
Choice Reasoning: The user is defining a constraint or guideline regarding the s
patial relationship between a mouse and a dog on the canvas.
Map  {}
File reset
File reset
```

And then we realized the mistake we made before, we forgot to “tell” the AI the JSON format it should use to respond to us with the rule so we also added this injection (the highlighted part) in the same function : "The JSON format should be as follows: {\"choice\": \"your_choice\", \"reasoning\": \"your_reasoning\", \"rule\": \"your_rule\"}.”

Function def get_text_from_GPT(user_prompt: str, user_notation: str, …)

In this function, we added the “rule” logic for the AI to have it as an option when interpreting the users input and determining the appropriate an action to take based on its verdict. When we first added the logic, we had little errors which required us to simply initialize a rule list in the beginning that we forgot to put before and here’s the main bit of code we added following a similar structure to the other switchers:

```
elif injection_verdict == "RULE":
    if rule:
        rules["constraints"].append(rule)
        print(f"Rule defined: {rule}")
        print(f"All Rules: {rules['constraints']}")
    print("\nRULE DONE ------------------")
    return f"Rule defined: {rule}", None, None, False, False, False, False, "", mapShape, mapCount
```

And then we tested it on Dynotate, giving a similar input like “a mouse should always be next to a dog” and got this output on the terminal:

```
RULE ----------------------

Choice Reasoning: The user is specifying a constraint or condition where a mouse
 must always be positioned next to a dog. The keyword 'always' indicates a rule.
Extracted Rule: A mouse should always be next to a dog.
Map  {}
Extracted Rule: A mouse should always be next to a dog.

RULE DONE ------------------
```

We see now that the AI recognizes and extracts a simple rule given my the user!

Next, we tried to see if we can give two rules and see if it outputs all the rules. Next, we tested giving two rules to see if it would output both. Initially, it only output the last rule given we were able to update the rules list and keep all the rules provided by the user. This gave us the desired result which is good enough for a beginners step:

First rule:

```
RULE ----------------------

Choice Reasoning: The phrase 'a dog should always be next to a mouse' indicates a rule
or constraint that specifies the positioning relationship between a dog and a mouse. Th
e use of the word 'always' is a clear indicator of a rule being defined.
Extracted Rule: A dog should always be next to a mouse.
Map  {}
Extracted Rule: A dog should always be next to a mouse.
All Rules: ['A dog should always be next to a mouse.']

RULE DONE ------------------
```

Second rule (including first):

```
RULE ----------------------

Choice Reasoning: The user is setting a constraint for the placement of objects with th
e phrase 'should always be'. This indicates a rule or condition that must be applied wh
en arranging these objects on the canvas.
Extracted Rule: a cat should always be next to a dog
Map  {}
Extracted Rule: a cat should always be next to a dog
All Rules: ['A dog should always be next to a mouse.', 'a cat should always be next to
a dog']
```

For the next part, to improve the AI's understanding of rules, we will evaluate whether it can recognize more complex rules defined by user will test to see if it can distinguish multiples rules in one input. Additionally, we plan to test its ability to draw while following the given rules, starting with simple rules at first, as if it works, this would be contribute well for our final product probably. Since we also created games ourselves with our own rules, if the time permits, we'll maybe try explaining the game and its rules to the AI to see if it can store and understand them effectively.