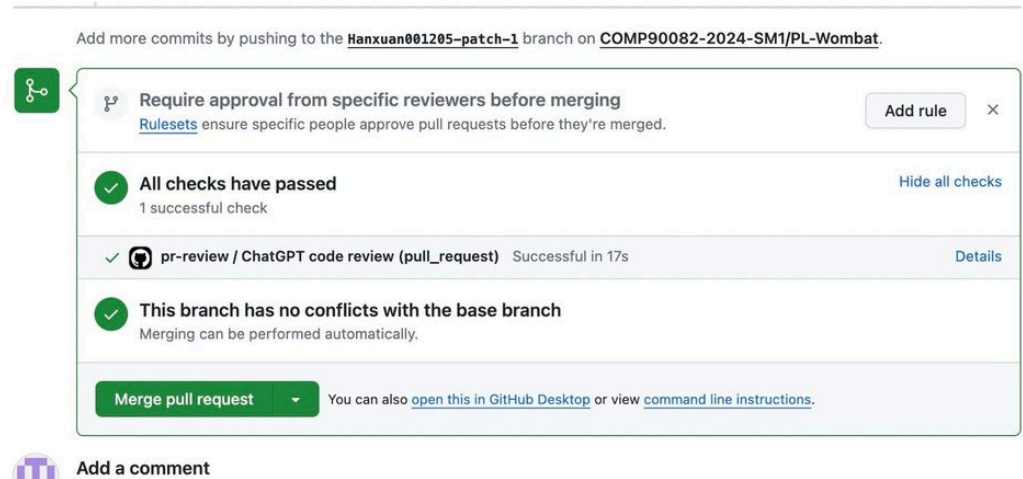


# Code Review Manually

Notice: This is an empty template. Once the pull request is submitted, the AI Code reviewer works, but nothing is automatic replied. This part will be complete after the problem is fixed.

*Now that we have resolved the issue with the auto code review, we will no longer use manual code review.*



As the figure shows above, our code review is without automatic reply.

We have made some attempts, but have not been helpful. In the Ed discussion, some other groups have encountered the same issue as us. Their approach "issue has been solved without specifying any model or files" didn't work for us either. We tried restricting the files and models, but the result was the same. Even when we specified individual files for review, there was still no reply. Therefore, we have decided to proceed with manual ChatGPT code reviews until we resolve the issues with the automatic code review.

Our GitHub action code is shown below:

```
Code Blame 16 lines (16 loc) · 427 Bytes Your organization can pay for GitHub Copilot

1  name: pr-review
2  on:
3    pull_request:
4      types: [opened, synchronize]
5  jobs:
6    review:
7      name: ChatGPT code review
8      runs-on: ubuntu-latest
9      steps:
10       - name: PR ChatGPT Review
11         uses: agogear/chatgpt-pr-review@0.0.2
12         with:
13           openai_api_key: ${ secrets.OPENAI_API_KEY }
14           github_token: ${ secrets.GITHUB_TOKEN }
15           github_pr_id: ${ github.event.number }
16           files: "*.py"
```

We conduct a manual ChatGPT review using the following instructions:

User

I will upload some files one by one, please follow the instruction to review them: Documentation Defects: a. Naming: Assess the quality of

software element names. b. Comment: Analyze the quality and accuracy of code comments.

Visual Representation Defects: a. Bracket Usage: Identify any issues with incorrect or missing brackets. b. Indentation: Check for incorrect indentation that affects readability. c. Long Line: Point out any long code statements that hinder readability.

Structure Defects: a. Dead Code: Find any code statements that serve no meaningful purpose. b. Duplication: Identify duplicate code statements that can be refactored.

New Functionality: a. Use Standard Method: Determine if a standardized approach should be used for single-purpose code statements.

Resource Defects: a. Variable Initialization: Identify variables that are uninitialized or incorrectly initialized. b. Memory Management: Evaluate the program's memory usage and management.

Check Defects: a. Check User Input: Analyze the validity of user input and its handling.

Interface Defects: a. Parameter: Detect incorrect or missing parameters when calling functions or libraries.

Logic Defects: a. Compute: Identify incorrect logic during system execution. b. Performance: Evaluate the efficiency of the algorithm used.

Provide your feedback in a numbered list for each category. At the end of your answer, summarize the recommended changes to improve the quality of the code provided.

Following the instructions (based on ED discussion) above, we conduct a manual code review of the PDDL files in the AnimationProfiles folder and document the review results below:

<div>driverlogDomain.pddl</div> <div><div>1. Documentation Defects</div><div>a. Naming: The naming of predicates and parameters seems to be generally clear and related to the domain of a driver log system (e.g., TRUCK, DRIVER, LOCATION). However, more descriptive names could be used for variables like ?obj1, which is not self-explanatory.</div><div>b. Comment: There are no comments present in the provided snippet. Adding comments to explain the purpose of actions and predicates would enhance maintainability and readability.</div></div> <div><div>2. Visual Representation Defects</div><div>a. Bracket Usage: Brackets seem to be used correctly to define scopes of actions and predicates.</div><div>b. Indentation: Indentation is inconsistent, particularly within action definitions and predicate lists. Proper and consistent indentation would improve readability.</div><div>c. Long Line: Some action parameter lines are long and wrap awkwardly. Breaking these into multiple lines could aid readability.</div></div> <div><div>3. Structure Defects</div><div>a. Dead Code: No dead code is apparent in the snippet provided.</div><div>b. Duplication: No obvious duplication within this limited scope, but broader review is necessary to confirm.</div></div> <div><div>4. New Functionality</div><div>a. Use Standard Method: The actions use a standard structured method appropriate for PDDL. The effectiveness depends on the specific goals and environment setup, which are not detailed here.</div></div> <div><div>5. Resource Defects</div><div>a. Variable Initialization: Variables within actions are appropriately initialized through parameters and preconditions.</div><div>b. Memory Management: As PDDL does not manage memory in the conventional sense (being a planning domain description language), this isn't applicable.</div></div> <div><div>6. Check Defects</div><div>a. Check User Input: Not applicable as PDDL doesn't interact directly with user inputs in this form.</div></div> <div><div>7. Interface Defects</div><div>a. Parameter: Parameters are used correctly within actions. However, clarity could be improved by ensuring that each parameter's purpose and role are understandable from its naming.</div></div> <div><div>8. Logic Defects</div><div>a. Compute: From the actions presented, logical structure follows typical planning requirements. Each action's preconditions and effects logically correspond with its intended operation.</div><div>b. Performance: Efficiency in a PDDL context would relate more to the optimality and complexity of the planning problem. Without running these definitions in a planner with specific goals, performance can't be effectively evaluated.</div></div>	<div><div>Summary of Recommendations:</div><div>Improve Documentation: Add comments to explain each predicate and action clearly.</div><div>Enhance Readability: Standardize indentation and avoid long lines to make the script easier to read and maintain.</div><div>Refine Naming: Use more descriptive variable names, especially where the purpose isn't immediately clear (e.g., ?obj1).</div></div>
--	---

driverlog\_pfile1.pddl

1. Documentation Defects

a. Naming: The names used for objects such as driver1, truck1, package1, etc., are clear and indicate their roles within the logistics domain. Locations labeled as s0, s1, etc., are somewhat ambiguous and could benefit from more descriptive names.

b. Comment: The file lacks comments. Including comments explaining the purpose of specific objects, initial conditions, and goal configurations would enhance understanding and maintainability.

2. Visual Representation Defects

a. Bracket Usage: Brackets are correctly used to delineate different sections such as objects, initial state, and goals.

b. Indentation: Indentation is somewhat consistent, which helps in distinguishing between different sections. However, some lines could benefit from better alignment for improved readability.

c. Long Line: The lines are generally well-formatted and do not extend excessively, aiding readability.

3. Structure Defects

a. Dead Code: There is no apparent dead code in the initial state or goal definitions provided.

b. Duplication: No obvious duplication is visible in the problem definition.

4. New Functionality

a. Use Standard Method: The problem file adheres to standard PDDL methodologies for defining objects, initial states, and goals, which is appropriate for the planning system it is intended for.

5. Resource Defects

a. Variable Initialization: All objects and their states are correctly initialized in the :init section.

b. Memory Management: Not applicable for PDDL.

6. Check Defects

a. Check User Input: Not applicable for PDDL as user input is not directly handled in the problem files.

7. Interface Defects

a. Parameter: Parameters are not directly applicable in this context, but the object declarations and their usage in goals and initial states are correct and consistent with the domain definition.

8. Logic Defects

a. Compute: The logical setup of the initial conditions and goals appears correct based on standard planning problem formulations.

b. Performance: Performance in terms of planning efficiency cannot be determined from the problem file alone without executing the plan with a PDDL solver.

Summary of Recommendations:

Enhance Documentation: Introduce comments to explain the setup of the problem file, especially the purpose behind specific configurations of objects and goals.

Refine Naming: Use more descriptive names for locations to improve clarity (e.g., instead of s0, names could reflect geographic or functional attributes).

Ensure Consistency in Formatting: Align elements more uniformly within lists for better visual inspection and readability.

snakeDomain.pddl

1. Documentation Defects

a. Naming: The naming convention in this file, such as tailsnake, headsake, nextsnake, is clear and specifically tailored to the domain of a snake game. However, names like dummyspoint might need more context or a more descriptive identifier.

b. Comment: The use of semicolons for comments is present, explaining the purpose of each predicate succinctly. This is good practice, but more comprehensive comments explaining complex actions could further enhance clarity.

2. Visual Representation Defects

a. Bracket Usage: The usage of brackets to define actions and predicates is correct and clearly demarcates different sections of the logic.

b. Indentation: Indentation appears consistent throughout, aiding readability significantly.

c. Long Line: Some lines, especially within action definitions, are quite long and contain complex conditions. Breaking these into multiple lines could improve readability.

3. Structure Defects

a. Dead Code: No apparent dead code in the provided snippet.

b. Duplication: The structure seems efficient with no obvious duplication of logic or definitions.

4. New Functionality

a. Use Standard Method: The actions defined appear to follow a standard methodological approach suitable for PDDL, focusing on movement and interaction within a grid-based snake game.

5. Resource Defects

a. Variable Initialization: All variables used in actions are properly initialized through parameters and conditions.

b. Memory Management: As with other PDDL files, memory management is not applicable here.

6. Check Defects

a. Check User Input: Not applicable in PDDL as it does not handle user input directly.

7. Interface Defects

a. Parameter: Parameters are well-defined and correctly used in actions, aligning with their respective predicates for movement and game state manipulation.

8. Logic Defects

a. Compute: The logic for moving the snake and handling game events (like eating or moving without eating) is coherent and should function correctly within a planning environment.

b. Performance: Efficient from a logical standpoint; however, performance metrics would depend on the planner's ability to execute these actions efficiently in the context of a game simulation.

Summary of Recommendations:

Expand Documentation: While basic comments are present, more detailed descriptions of complex actions and their intended effects would be beneficial.

Refine Visual Representation: Consider breaking longer logical expressions into multiple

lines to avoid horizontal scrolling and enhance code comprehensibility.

Clarify Naming: Some names like dummyspoint might benefit from re-evaluation to convey their purpose more clearly or accurately.

snakeProblem01.pddl

1. Documentation Defects

a. Naming: The naming of positions like pos0-0, pos0-1, etc., clearly indicates a grid-based coordinate system, which is appropriate for a snake game problem definition. These names are logical and help in understanding the grid layout.

b. Comment: The provided snippet lacks comments. Adding comments that describe the purpose of the setup and the relationships established, like adjacency, could greatly enhance readability and comprehension for new readers.

2. Visual Representation Defects

a. Bracket Usage: Brackets are used correctly to separate different sections, such as domain, objects, and initial conditions.

b. Indentation: Indentation is consistent, helping to separate predicates clearly, which aids in readability.

c. Long Line: The adjacency relationships are listed in a lengthy, somewhat repetitive manner. While necessary for the domain, breaking these into more visually digestible segments could improve ease of reading.

3. Structure Defects

a. Dead Code: There is no apparent dead code within the initial conditions shown.

b. Duplication: The structure seems appropriate without unnecessary duplication, though the repetitive nature of the adjacency declarations is inherent to the problem definition.

4. New Functionality

a. Use Standard Method: The file adheres to standard PDDL conventions for defining problem specifics, particularly focusing on the grid layout and possible moves.

5. Resource Defects

a. Variable Initialization: All variables (positions) are properly declared and utilized within the initial conditions.

b. Memory Management: Not applicable for PDDL files.

6. Check Defects

a. Check User Input: Not applicable as PDDL does not handle user input directly.

7. Interface Defects

a. Parameter: Parameters are not directly used in this section, but object declarations are correct and consistent with the domain definitions.

8. Logic Defects

a. Compute: The logical relations for adjacency are well-defined and necessary for the functioning of a snake game within a planner. Each adjacency defines potential moves, essential for planning paths.

b. Performance: The performance of the logic will depend on the planner's efficiency in handling these definitions, particularly the number of possible states generated by these adjacency relations.

Summary of Recommendations:

Expand Documentation: Include more comprehensive comments to describe the structure and purpose of the problem setup, especially explaining complex sections like adjacency relations.

Enhance Readability: Consider formatting improvements for lists of relations to avoid visual repetition and enhance clarity.

Refine Data Presentation: While the naming and structure are logical, presenting adjacency in a more compact or grouped form could simplify understanding the grid layout.

snakeAP.pddl

1. Documentation Defects

a. Naming: The naming of predicates and visual elements like snake-head and snake-tail clearly reflect their roles within the game's animation context. These names are intuitive for understanding their purpose in visual effects.

b. Comment: The file includes comments explaining the overall purpose and the specific role of each section. This is an excellent practice that enhances readability and understanding.

2. Visual Representation Defects

a. Bracket Usage: Brackets are properly used to define and segregate different sections and definitions.

b. Indentation: The file shows consistent indentation, which helps in navigating through various definitions and enhances the readability of the file.

c. Long Line: The formatting does not present any overly long lines, and the use of line breaks is appropriate, contributing positively to readability.

3. Structure Defects

a. Dead Code: There is no apparent dead code in the snippet provided.

b. Duplication: The structure avoids unnecessary duplication, with each predicate and visual element uniquely defined.

4. New Functionality

a. Use Standard Method: The file employs a structured approach suitable for defining visual animations in a PDDL-like syntax. This includes custom properties and effects linked to the game's visual representation.

5. Resource Defects

a. Variable Initialization: Variables are well initialized, with parameters and properties clearly defined within visual and predicate elements.

b. Memory Management: Not applicable for PDDL files.

6. Check Defects

a. Check User Input: Not directly relevant, as PDDL files don't handle user input directly.

7. Interface Defects

a. Parameter: Parameters for predicates and visuals are correctly specified and consistently used, providing clear information on their roles.

8. Logic Defects

a. Compute: The logical structure supports the visual and interaction dynamics of the Snake game effectively. It integrates visual cues with game mechanics.

b. Performance: The definitions are logically coherent; however, performance in terms of visualization would depend on the implementation details in the animation engine.

Summary of Recommendations:

Maintain Documentation Quality: The current use of comments and explanations is very beneficial. Continuing this practice will ensure the file remains accessible and understandable.

Monitor Visual Properties: Keep an eye on how visual properties (like color, width, height) are assigned to ensure they align with the intended aesthetic and functional aspects of

ferryDomain.pddl

1. Documentation Defects

- a. Naming: The names used in the file such as car, location, at-ferry, sail, board, and debark are clear and relevant to the domain of ferry transportation, effectively describing the entities and actions within the system.
- b. Comment: The file lacks comments. Including comments explaining the purpose and functionality of actions and predicates would improve understanding and maintainability, especially for those unfamiliar with the domain.

2. Visual Representation Defects

- a. Bracket Usage: Brackets are correctly used to define and separate predicates and actions, which helps maintain structural clarity.
- b. Indentation: Indentation is generally consistent, aiding in the readability of action definitions and conditions.
- c. Long Line: The lines are not excessively long and are broken appropriately to enhance readability.

3. Structure Defects

- a. Dead Code: There is no apparent dead code in the snippet provided.
- b. Duplication: The structure seems efficient with each predicate and action serving a distinct purpose without unnecessary duplication.

4. New Functionality

- a. Use Standard Method: The actions defined follow standard PDDL methodologies for describing planning tasks, suitable for the context of ferry transport, including actions for sailing, boarding, and debarking.

5. Resource Defects

- a. Variable Initialization: Variables within actions are appropriately initialized through parameters and preconditions.
- b. Memory Management: Not applicable for PDDL.

6. Check Defects

- a. Check User Input: Not applicable as PDDL doesn't directly handle user input.

7. Interface Defects

- a. Parameter: Parameters are used effectively within actions, and their roles are clearly defined through preconditions and effects, ensuring that the ferry, locations, and cars interact correctly.

8. Logic Defects

- a. Compute: The logical constraints and action effects are appropriately defined to reflect the real-world operations of a ferry system.
- b. Performance: The performance efficiency of the domain would depend on the specific scenarios and planner used but is designed to be straightforward and effective.

Summary of Recommendations:

Enhance Documentation: Include comments to explain each predicate and action, providing context and clarifying their purposes.

Ensure Consistency in Formatting: While generally good, ensuring that indentation and bracket usage are consistently applied throughout the file will improve maintainability.

Review Logic and Conditions: While the logic seems sound, always ensure that conditions like (not-eq ?from ?to) are robust and consider edge cases or unusual scenarios that may occur in real-world ferry operations.

ferryProblem.pddl

1. Documentation Defects

- a. Naming: The naming of locations (l0, l1, etc.) and cars (c0, c1, etc.) is systematic and functional, which is suitable for a problem file where many similar entities need to be managed. However, more descriptive names could help to give context (e.g., indicating different regions or types of locations).
- b. Comment: The file lacks comments. Including comments to describe the setup, purpose of the problem, and expected interactions between entities (ferry, cars, locations) could greatly enhance understandability.

2. Visual Representation Defects

- a. Bracket Usage: Brackets are used correctly to define and separate the sections and elements, such as objects, initial state, and goals.
- b. Indentation: Indentation is mostly consistent, helping to differentiate between different sections and making the file easier to follow.
- c. Long Line: The representation of distinct not-eq predicates for location pairs, while necessary, results in a repetitive structure that might be visually cumbersome. Grouping related predicates or summarizing them could improve readability.

3. Structure Defects

- a. Dead Code: There is no apparent dead code in the provided snippet.
- b. Duplication: While the not-eq predicates are repetitive, they are essential for defining the problem constraints and ensuring correct problem setup.

4. New Functionality

- a. Use Standard Method: The problem file uses a standard methodological approach for defining ferry logistics scenarios in PDDL, including initialization of entities and goal states.

5. Resource Defects

- a. Variable Initialization: All variables are properly initialized in the :init section, specifying the locations of cars and the ferry as well as the states of these entities.
- b. Memory Management: Not applicable for PDDL.

6. Check Defects

- a. Check User Input: Not applicable as PDDL does not handle user input.

7. Interface Defects

- a. Parameter: Parameters are not used in this section, but the object declarations and their roles within the initial conditions and goals are clearly defined and appropriate.

8. Logic Defects

- a. Compute: The logical setup of initial conditions and goals is sound, with clear relationships established between different locations and entities to facilitate ferry operations.
- b. Performance: The problem is defined efficiently with respect to the ferry's operational logic, although the effectiveness of solving this problem will depend on the planner's capability to handle the constraints and objectives efficiently.

Summary of Recommendations:

Improve Documentation: It would be beneficial to include comments explaining the logic

behind initial states and goal configurations.

Refine Visual Representation: Consider optimizing how predicates are presented to reduce visual repetition and improve ease of understanding.

Clarify Goals: Ensure that the goals are aligned with realistic operational constraints and scenarios, providing clear and achievable objectives for the planning system.

ferryAP.pddl

1. Documentation Defects

a. Naming: The naming within the file, such as location, at-ferry, and on, clearly corresponds to the logical constructs of the ferry domain and helps in understanding their roles within the visual context.

b. Comment: The file includes minimal comments. It would be beneficial to include more detailed explanations, especially describing the effects and the purpose of the custom visuals to enhance clarity for someone unfamiliar with the file.

2. Visual Representation Defects

a. Bracket Usage: Brackets are properly utilized to clearly delineate predicates, actions, and visual elements, which aids in maintaining the structure of the document.

b. Indentation: Indentation is consistent and effectively separates different elements and sections, improving the file's readability.

c. Long Line: The visual properties are defined in a way that avoids overly long lines, making it easier to read and understand the setup for each visual element.

3. Structure Defects

a. Dead Code: There is no apparent dead code; each section contributes to defining how the ferry and related entities are visualized.

b. Duplication: There is no unnecessary duplication; each visual and predicate definition serves a unique purpose.

4. New Functionality

a. Use Standard Method: The file uses a structured approach to define the visual representation of a domain in PDDL, focusing on how elements like locations, ferries, and cars appear and interact visually.

5. Resource Defects

a. Variable Initialization: Visual properties and effects are initialized with precise details, like positions and images, ensuring that every element's visual behavior is well-defined.

b. Memory Management: Not applicable for PDDL.

6. Check Defects

a. Check User Input: Not applicable as PDDL files do not handle user input directly.

7. Interface Defects

a. Parameter: Parameters in predicates and visuals are properly used to define interactions and relationships, ensuring that the visual effects correspond accurately to the underlying logical states.

8. Logic Defects

a. Compute: The logical definitions related to visual effects appear coherent, ensuring that the visual representation aligns with the domain's operational logic.

b. Performance: The design is effective; however, the actual performance will depend on the visual system's ability to render these configurations efficiently.

Summary of Recommendations:

Enhance Documentation: More comprehensive comments detailing the functionality of each visual and predicate definition would improve the file's usability.

Review Visual Definitions: Ensure that all visual properties, such as positions and

dimensions, align with the intended user experience and the practical requirements of the visualization system