# Introduction

This is a report of analyzing a given project of Tetris game in the context of adding several buttons to execute functions of keystrokes. To make the project more extensible, maintainable with higher order of encapsulation, I utilize the design pattern of command. By encapsulating the event handlers of user inputs, the modularity of the program is higher with more ease to extent features into it.

# Code Design

The current code works fine in completing the game logic. The game model is elegantly and briefly implemented. The user uses minimum code to hold several kinds of combinations of Tetris game. However, the handling of events is poorly encapsulated and in the context of adding another kind of event which does the same things with the prior ones, the user must repeat the same logics, which means he will have to copy the whole if else or switch snippet

into the new event handler. When he wants to add something into one kind of event, such as making some changes to the panel that displays the next Tetris block, he will have to add the same logic in both event handlers, which causes code repeating and is prone to errors and inconsistency.

# Feature Extension

My method to add new feature is first encapsulate the keystrokes' responses into commands. Command is an interface which has a single method execute. It has four implementations which is moving left, moving right, turning, and dropping respectively. The Tetris component instance will have a hash map which has key of string and value of Command instance. The handling of moving left, etc. is encapsulated into Command instances, and the event handlers

are changed to execute the command in the map. In such way, the modularity is intensified and the extensibility because if the user wants to make more effects in the moving left event, he will only have made changes to MoveLeftCommand, and the difference is reflected in every event handler. If he wants to add new event handlers, he will have to write new instance.

After encapsulating the commands, the user will simply add four buttons in the Tetris component and set its event handler to execute its respective command, and do not forget to change the original event handler for keystrokes.

图示

描述已自动生成

# Impact

The features of adding control buttons will be inserted into the code elegantly. There is no need of worrying about when adding a new feature in response to an event without adding for another event, so the risks of such kinds of errors are lowered. Whenever the user wants to add another new feature, he will just have to implement the Command Interface and add it to the table and execute the command in response to target events. The execution of user event is encapsulated into separate objects; its modularity is also intensified.

# Conclusion

By encapsulating the original event handling into command objects, the modularity of the program is better so the addition of new features which control the game with buttons is easier and less error prone. Additionally, adding new features into the game is also easier so the impact is not limited in this task.

# Reference

The report does not use any references.