STATE DESIGN PATTERNS

Microsoft Office User

Adil Salamat
100879914

Table of Contents

Adil Salamat
100879914

Introduction

In this report, we will be discussing the state design pattern. I will give a detailed explanation as to what the state design pattern is, why you would use it for your project, and finally, how I plan to implement the design pattern into my final project.

What is the State Design Pattern?

The state design pattern is a design pattern that allows objects to change their behaviour based on what state that object is. Their internal state can change based on user input. For example, if a user enters the correct password, the state of the program can change which then triggers a different method. States can also change based on how the programs work, for example, if you are playing a game like Pacman, the ghosts changing blue and running away is an example of them changing state. The code written would tell the ghosts that when the special pill is eaten, your state changes from regular state to blue state which results in a change of behaviour.

The state design pattern works by listing the possible states, then initialising the object to its regular state. Based on input or feedback, the state of the object would be updated and returned to the method. This would then trigger a different method which would change the behaviour.

Below is an example from codeproject.com of how the state design pattern would be used

```
class NoStateATM
{
    public enum MACHINE_STATE
    {
        NO_CARD,
        CARD_VALIDATED,
        CASH_WITHDRAWN,
    }

    private MACHINE_STATE currentState = MACHINE_STATE.NO_CARD;
    private int dummyCashPresent = 1000;

    public string GetNextScreen()
    {
        switch (currentState)
        {
            case MACHINE_STATE.NO_CARD:
                // Here we will get the pin validated
                return GetPinValidated();
                break;
            case MACHINE_STATE.CARD_VALIDATED:
                // Lets try to withdraw the money
                return WithdrawMoney();
                break;
            case MACHINE_STATE.CASH_WITHDRAWN:
                // Lets let the user go now
                return SayGoodBye();
                break;
        }
        return string.Empty;
    }
```

Enum machine state lists out the possible states for the ATM machine. We can see that the ATM has been initialised to the first state, no card. There is then a switch case, which dictates what method is executed.

```
private string GetPinValidated()
{
    Console.WriteLine("Please Enter your Pin");
    string userInput = Console.ReadLine();

    // lets check with the dummy pin
    if (userInput.Trim() == "1234")
    {
        currentState = MACHINE_STATE.CARD_VALIDATED;
        return "Enter the Amount to Withdraw";
    }

    // Show only message and no change in state
    return "Invalid PIN";
}
```

This is the first method that executes due to the current state of the ATM, we can see that that program asks for user input, based on that input the ATM will can change state leading to the next method being executed.

Why use State Design Pattern?

The state design pattern may be ideal for your given situation. If you are trying to create an object with polymorphic behaviour, it may be easier to use the state design pattern, especially if there are a number of objects with the same behaviour. It also can remove the dependency of using if else statements. State design patterns can be used to change the behaviour of an object rather than checking its status with if else or switch statements. Finally, new states and behaviours can be added without effecting the other components, giving room for modularity and expansion.

How will I use State Design Patterns?

During my plan, I decide to use state design patterns for my Pacman game. I came to this conclusion early on and also mentioned it above. I will primarily be using the state design pattern for the Ghost objects in my game.

All 4 Ghosts are similar in design and behaviour, their overall goal is to reach Pacman. Their behaviour changes however when Pacman eats a special pill. The Ghosts lose their unique colour, all becoming blue and vulnerable to being eaten by Pacman, they also run in the opposite direction away from Pacman. They change back after being eaten or after a certain amount of time has passed.

I can utilise the state design pattern here as all 4 Ghosts would share the same 3 states. They either are in their normal state where they chase Pacman, their blue state where they run from Pacman, or their dead state where they return home. Using the state pattern would mean that I can create the methods for all 3 of these states, then the ghosts will change based on the user input and what happens in the game. This would save me time from writing the behaviour of each individual Ghost as well as making the code more concise and improving readability.

Conclusion

In summary, we have discussed what the state design pattern is as well as the ideal scenario to use it and when to avoid it. Finally, we acknowledged how I plan to utilize the state design pattern for my final project.

Bibliography

The first source used was to help my understanding of state design patterns, the example and screenshots from this site were also used to further explain my point.

Singh, R. (2019). *Understanding and Implementing State Pattern in C#*. [online] Codeproject.com. Available at: https://www.codeproject.com/Articles/489136/UnderstandingplusandplusImplementingplusStateplusP [Accessed 13 Nov. 2019].

The second source used was a book by Robert Nystrom, this book has an in-depth description of patterns including the state design pattern. I used this book to further understand my knowledge of this pattern and for help on implementing this into my game.

Nystrom, R. (2014). *Game programming patterns*. Place of publication unknown: Genever Benning, pp.87-105.