Patterns

Our game will have a deck of cards set up in an array. Each card will have a math problem on it, like addition, subtraction, multiplication, or division, or a combination of all. The goal of the game is to match the math problems with their correct answers under a time constraint. The faster you are, the more points you obtain. We can implement the following design patterns in our game:

Factory Method:

This pattern will allow us to implement different types of cards using a factory method based on the math operation required. It will encapsulate the creation logic within the factory class. By implementing this pattern, we are able to ensure that our game is able to support various types of math problems without having to rewrite or restructure some of the larger portions of the code.

https://refactoring.guru/design-patterns/factory-method

Strategy Pattern:

This pattern can be employed to implement different scoring methods or difficulty levels in the game. Each strategy encapsulates a different algorithm for calculating scores as well as difficulty levels. By implementing this pattern, we offer a range of challenges for the players, from easy to difficult. Players are able to choose their own levels. New scoring methods can be introduced without changing the core game mechanics. This enhances the game's flexibility and playability.

https://refactoring.guru/design-patterns/strategy

State Pattern:

This is a behavioral design pattern that lets us alter an objects behavior when the internal state changes. There are a finite number of states our game can have, and in each unique state the program will behave a little different. The program is also able to flow from state to state. For example, our game can have a "playing" state or a "game over" state.

https://refactoring.guru/design-patterns/state