ABDELRAHMAN GAMAL
8039

MOHAMED BASSEM
8050

MOHAMED KHALED
8045

EIAD ALAA
7993

DESIGN PATTERNS

THE ITERATOR DESIGN PATTERN:

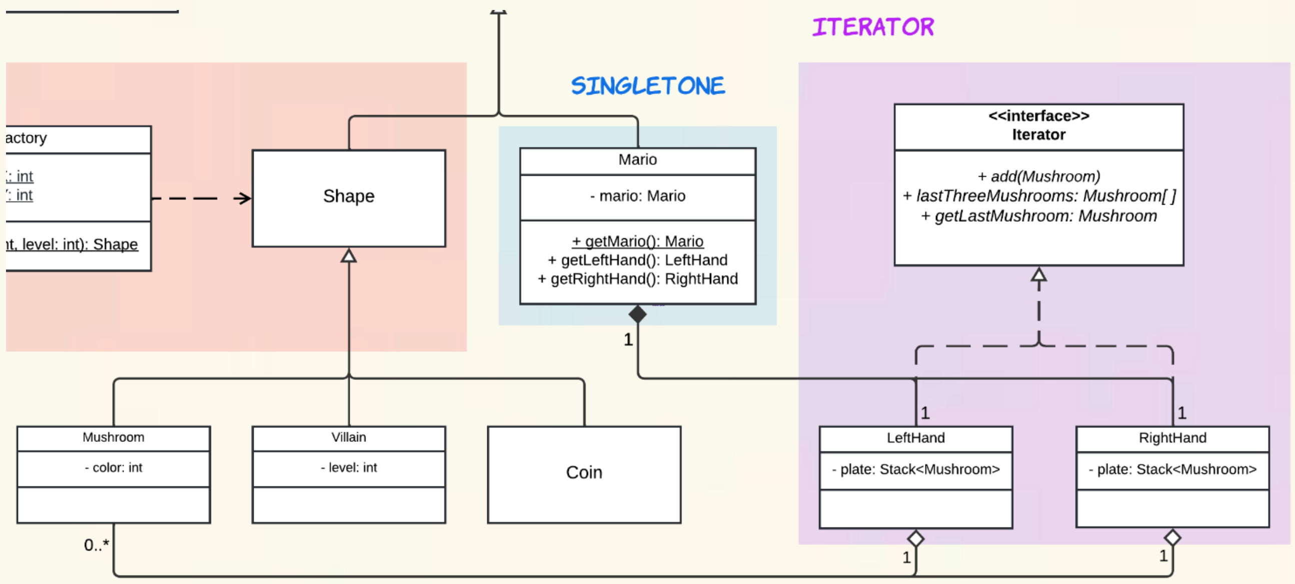
The Iterator design pattern is a behavioral pattern that provides a systematic way to access elements of a collection (such as lists, arrays, or trees) without exposing the underlying details of the collection's implementation.

In our code we used it in accessing the data of lefthand and right hand stacks that contains mushrooms and we implemented three function that are not the typical next () function, each function has a role in accessing data of the stack

1- `void add (Mushroom mushroom)` which adds a certain mushroom to the stack.

2- `Mushroom getLastMushroom()` is a getter of the last mushroom that entered the stack

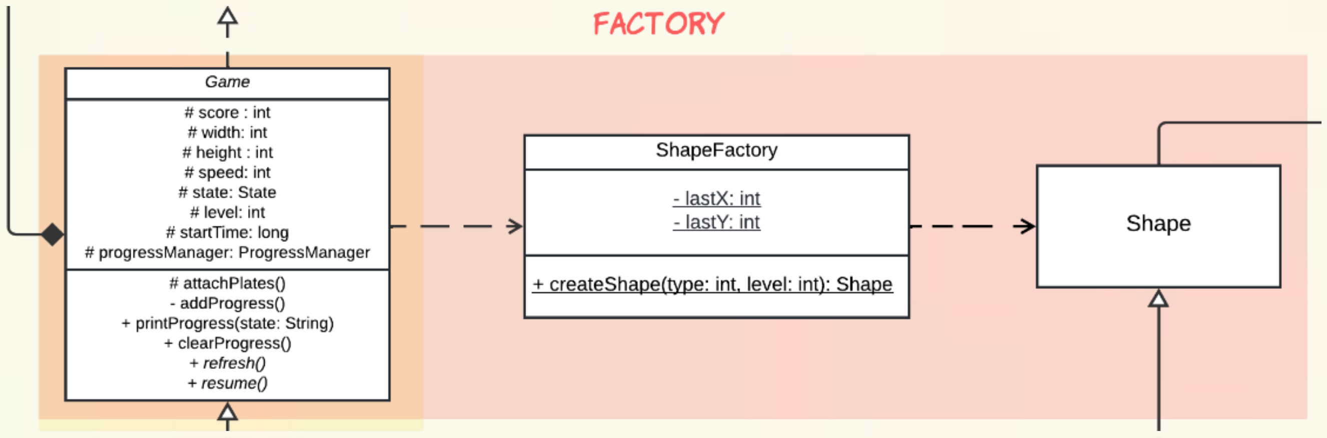
3- `Mushroom[] lastThreeMushrooms()` checks if the last three mushrooms has the same color or not



THE FACTORY DESIGN PATTERN:

The Factory Design Pattern is a creational pattern that provides an interface for creating objects in a super class but allows subclasses to alter the type of objects that will be created.

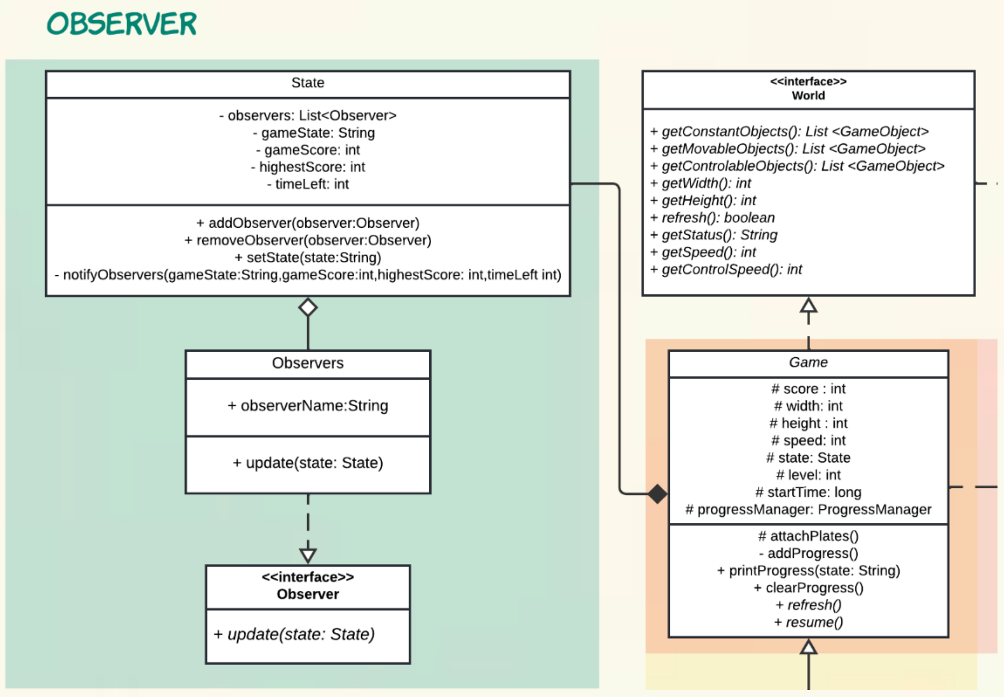
In our code we used Shape Factory that we use whenever we want to create any shape we want an instance of.



THE OBSERVER DESIGN PATTERN:

The observer design pattern is a behavioral pattern that defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

We used the observer design pattern to make the player observe the last points he scored. Using the observer design pattern will make it easier to add more observers to the game and other states being spectated.

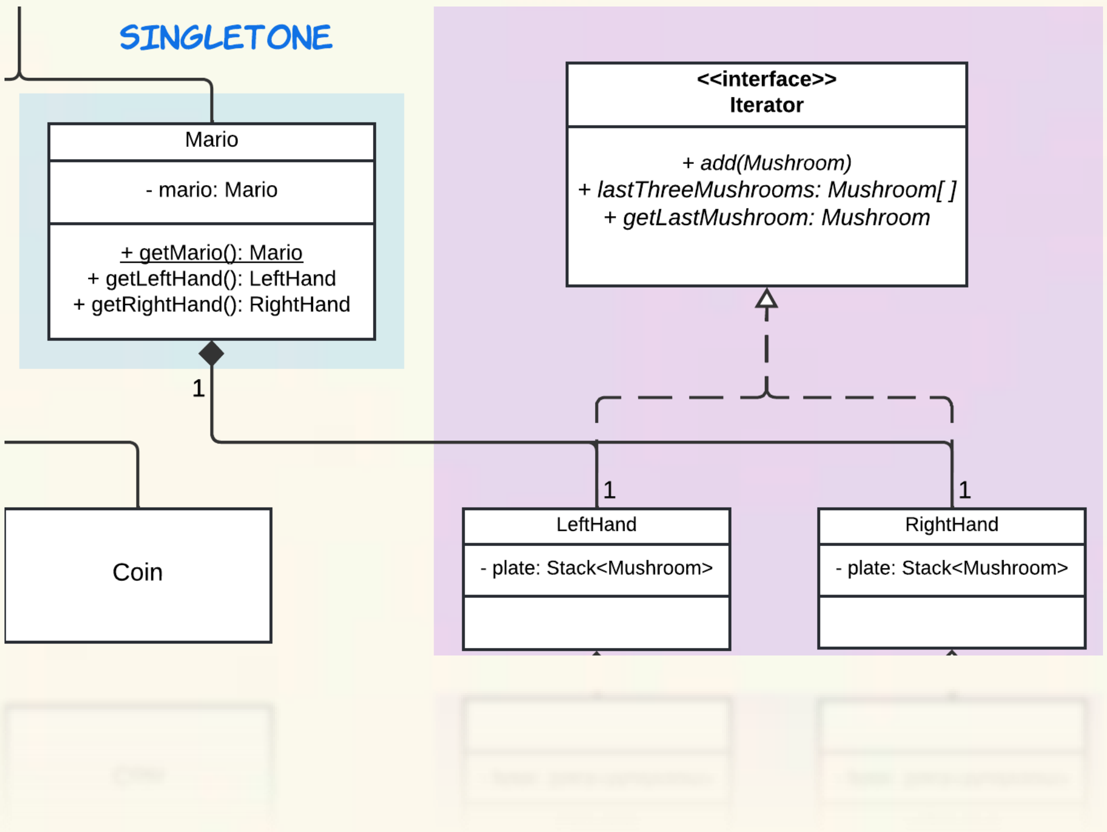


THE SINGLETON DESIGN PATTERN:

The Singleton Design Pattern is a creational design pattern that ensures a class has only one instance and provides a global point of access to that instance.

It is particularly useful when exactly one object is needed to coordinate actions across the system.

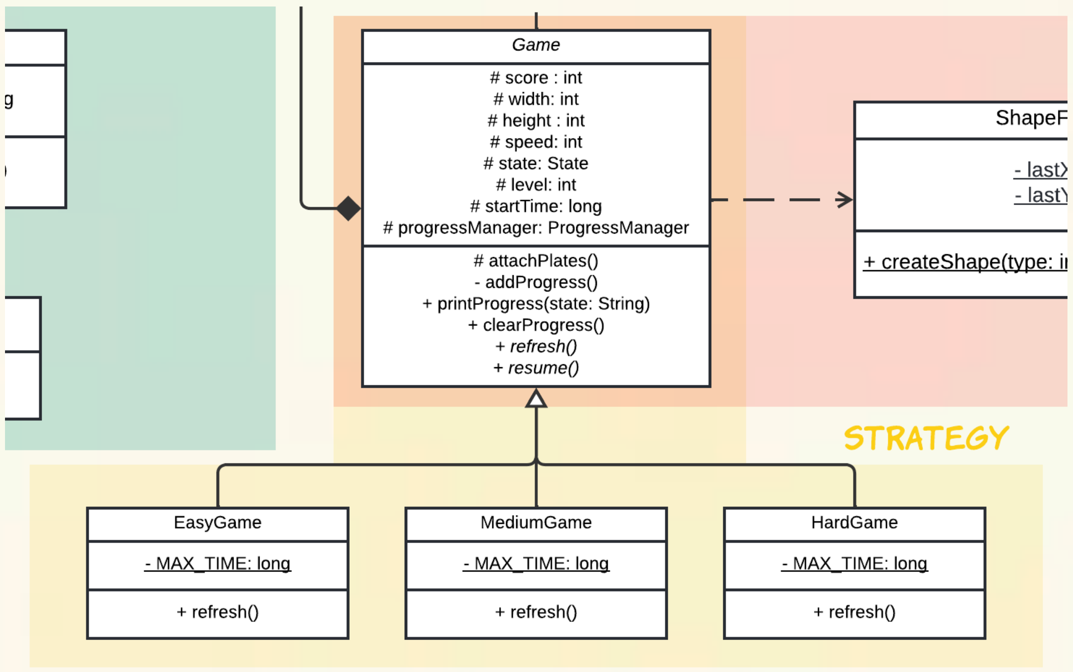
As shown in the below class diagram, **Mario** class generates only one instance of its type and provides a global point of access for the rest of the classes in the game.



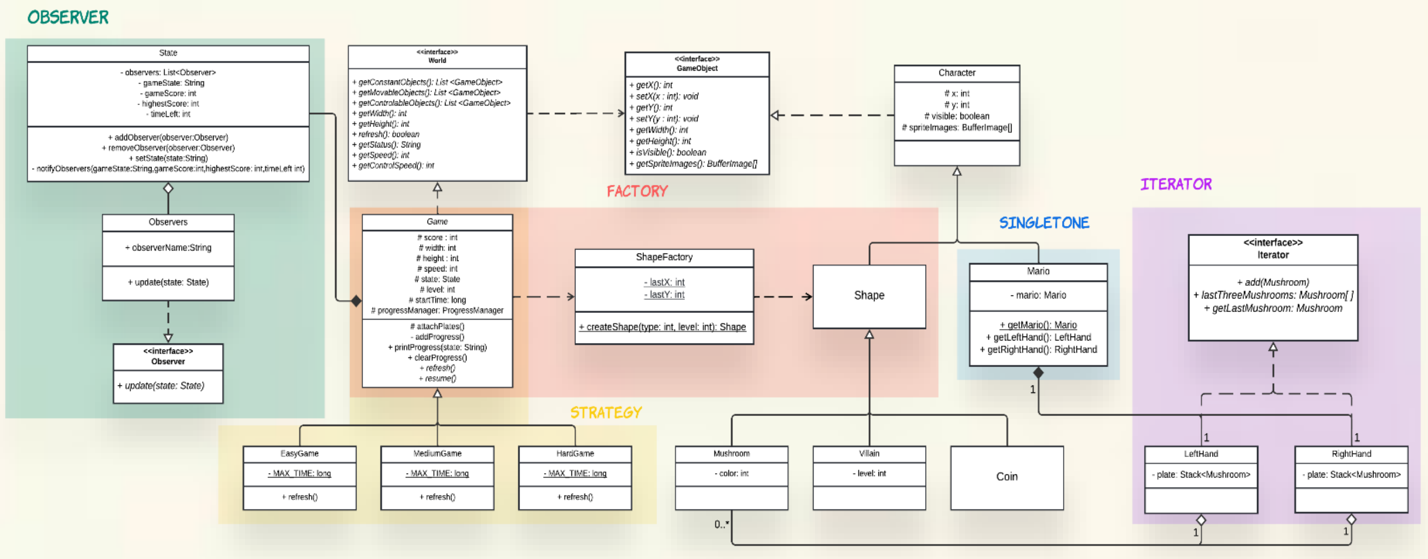
THE STRATEGY DESIGN PATTERN:

The Strategy Design Pattern is a behavioral design pattern that defines a family of algorithms, encapsulates each algorithm, and makes them interchangeable. It lets the client vary the algorithm independently from the clients that use it.

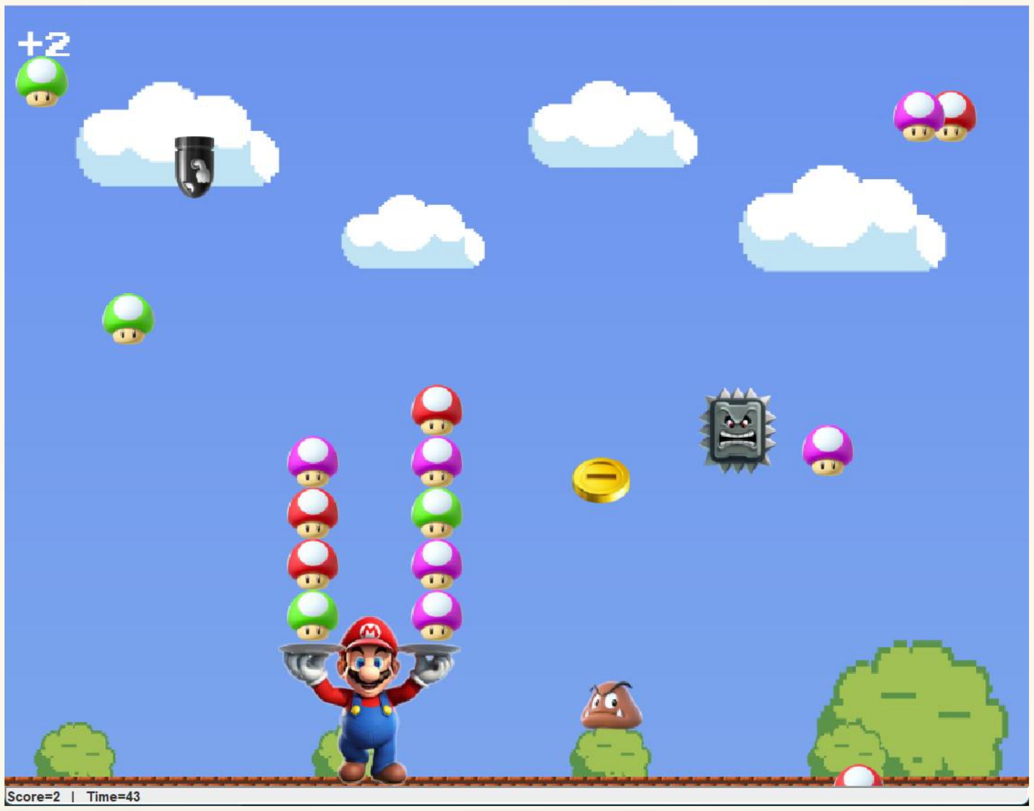
In our game, we have applied this pattern on the game difficulty system, it is used to represent different difficulty levels as separate strategies. Each difficulty level encapsulates its own algorithm for setting the game's difficulty, and the game can switch between these strategies based on the user choice. Below class diagram of this design pattern.



UML CLASS DIAGRAM



GAME SCREENSHOT


+2
Score=2 | Time=43