**PATTERNS IMPLEMENTED IN THE BOAT RACING CODE**

**1.Visitor Pattern and Strategy Pattern for movement and collisions**

In our project we have multiple dynamic objects that implement different types of movement (random, linear, by input) in order to simplify everything and to ensure that the objects access only the movement that they are able to do, we use the pattern Strategy and Visitor.

In order to carry out all types of movements, we have used a strategy pattern called Movement Strategy through which dynamic objects can access the move method to perform their corresponding movement, so that the objects do not depend on this pattern, both this and the visitor pattern are connected so that the movement depends on the visitors and not the objects themselves.

The different strategies implemented are:

- AI Controlled Strategy for rivals

- Straight Movement Strategy for obstacles

- Random Movement Strategy for obstacles

- Player Controlled Strategy for the boat related to the player

Every move type is totally independent from another.

**2. Singleton Pattern for the different screens of the project**

Another characteristic pattern is the Singleton, used to give a global view of the General Controller class but accessing its instance.

The General Controller acts as a central point of control for various screens and controllers in the project. By ensuring there's only one instance of General Controller, we guarantee that all parts of the game access and modify the game state consistently. This avoids issues related to having multiple instances modifying the state independently, which can lead to bugs and inconsistent behaviour.

**3. Observer Pattern for inputs of the user**

The observer pattern is a crucial part of the project, it is visible in the Input Manager class.

The Input Manager class is responsible for handling user input events (such as key presses). When using the Observer pattern, it does not need to know the details of the actions to be performed in response to these events. Instead, it simply notifies subscribed listeners. This decouples the input handling logic from the specific actions to be performed, making the code more modular and easier to maintain.

It can notify multiple listeners about an input event. This is useful if multiple parts of the game need to react to the same input event. It also adheres to the Single Responsibility Principle by focusing solely on handling input and notifying listeners, rather than containing the logic for handling specific input responses.