Technical Report

User stories:

- As a customer, I want to see a list of the games available in the catalogue, so what I can choose the ones I want to include in my arcade machine.
- As a customer, I want to add games to my arcade machine using a game code, so what I can select and customize my machine with the games I prefer.
- As a customer, I want to complete the purchase of my arcade machine and provide my delivery information, so what I can complete the transaction and receive my machine at the desired address.
- As a customer, I want to search for games by category (such as action, adventure, sports, etc.), so what I can easily find games that fit my tastes and preferences.
- As a store manager, I want to have the ability to add new games to the catalog, so what the game catalog stays up to date and customers can choose from the latest available options.
- As a client of the system, I want to select a predefined machine and choose the type of material, so what create a custom machine that is registered with a name of my choice, and I also want the system to adjust the weight, price and energy consumption according to the selected material (wood, aluminum, carbon fiber).
- As a customer of the system, I want to add video games to the registered machines, so what customize it.
- As a system customer, I want to add video games in standard definition or high definition, so what customize my machine to my preferences

Justification of Technical Decisions in the Catalog Arcade Machine

In the development of this program for an arcade machine catalogue, various technical decisions have been made in line with good software design practices, including the use of *design patterns*, component separation, and the implementation of *SOLID principles*. The justification for each of these decisions is detailed below:

1. Factory Design Pattern

The Factory pattern has been implemented for the creation of the different predefined arcade machines (DanceRevolutionMachine, ClasicArcadeMachine, etc.). This pattern is essential in this project for the following reasons:

Abstraction of object creation: The Factory pattern allows the creation of the arcade machines to be abstracted into different specific factories (DanceRevolutionFactory, ClasicArcadeFactory, etc.). This allows the client not to need to know the instantiation details of each machine, keeping the code cleaner and easier to maintain.

Extensibility: If new machines need to be added in the future (for example, an augmented reality machine), this can be done without modifying the existing factories. Simply add a new factory and the corresponding machine class, respecting the OCP (Open/Closed Principle), which states that the code must be open to extension but closed to modification.

2. Separation by Components

The decision to divide the system into components has been key to managing the complexity of the program. The main components identified are:

Arcade machines (Machine and its subclasses)

Video games (VideoGame, VideoGamesCatalog)

Factories (MachineFactory and subclasses)

Customer and Purchase (Client, PurchaseManager)

The justification for this separation is cohesion and modularity:

High cohesion: Each component groups classes that have a high internal cohesion, that is, they are strongly related to each other. For example, Machine subclasses share common behaviors such as adding or removing video games, and price calculation.

Low dependency between components: Components have low coupling with each other, meaning they can evolve independently. This separation facilitates code maintainability and makes it easier to add new features or refactor without affecting the entire system.

3. SOLID principles

SOLID principles have been applied to ensure a flexible and robust design. Here are some examples of their implementation:

S (Single Responsibility Principle): Each class has a single responsibility. For example, the Machine classes and their subclasses are exclusively responsible for the logic associated with the arcade machines, while VideoGame and VideoGamesCatalog handle exclusively the logic related to the video games. This separation of responsibilities facilitates maintainability and testing of each component separately.

O (Open/Closed Principle): As mentioned before, the system is designed in such a way that it can be extended without having to modify the existing code. For example, if you want to add new arcade machine classes or new types of video games, you can do so without altering the way your current classes work.

L (Liskov Substitution Principle): Subclasses of Machine (such as DanceRevolutionMachine or ClasicArcadeMachine) can replace the base class without altering the proper functioning of the system. They all respect the interface defined by the Machine abstract class, ensuring that they can be used interchangeably.

I (Interface Segregation Principle): Although Python doesn't have interfaces in the strict sense, we've applied this principle by designing classes with well-defined responsibilities. For example, Machine defines abstract methods that must be implemented in its subclasses, and it only exposes the methods needed to manipulate video games and calculate the price.

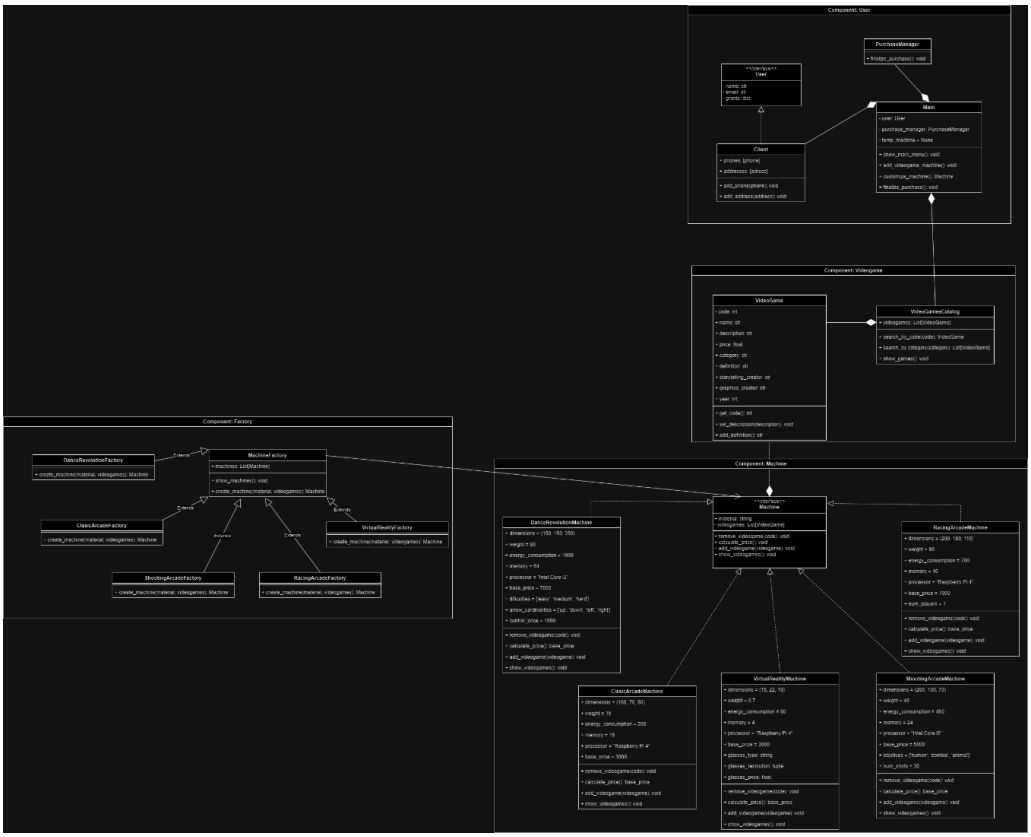
D (Dependency Inversion Principle): Instead of classes directly depending on concrete classes, they depend on abstractions. For example, factories depend on the MachineFactory abstraction, which is then implemented in concrete factories. This makes it easy to change implementations without affecting the overall system.

4. Modularity and Reusability

By using an object-oriented structure and design patterns, the code is highly modularized, which encourages reuse. For example:

The calculate_price and add_videogame methods of the different Machine subclasses use general logic that is inherited and then adapted to the specific needs of each machine. This allows for a high degree of code reuse without duplication.

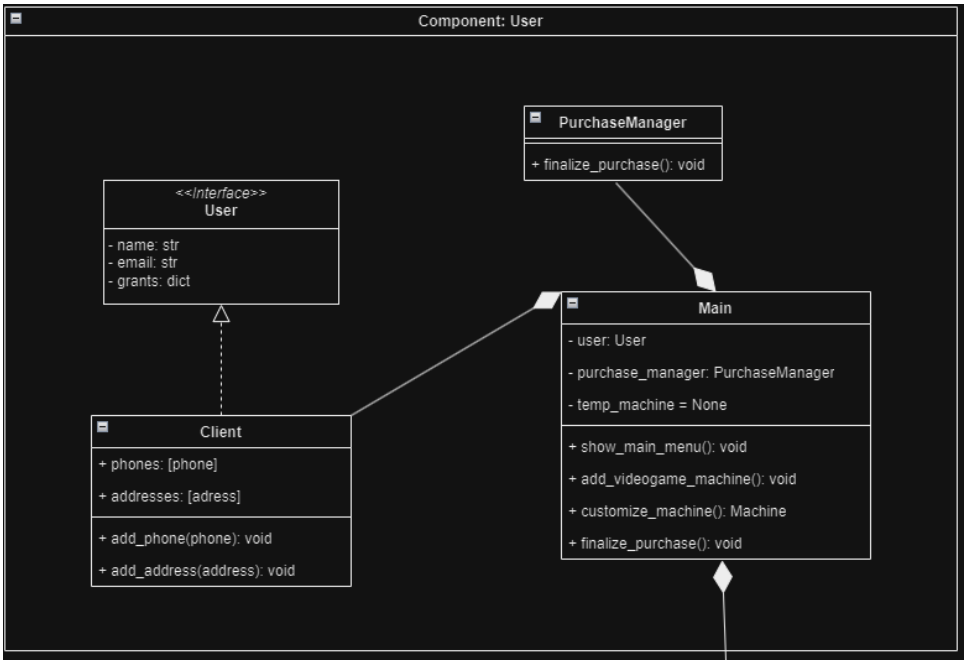
Class Diagram:



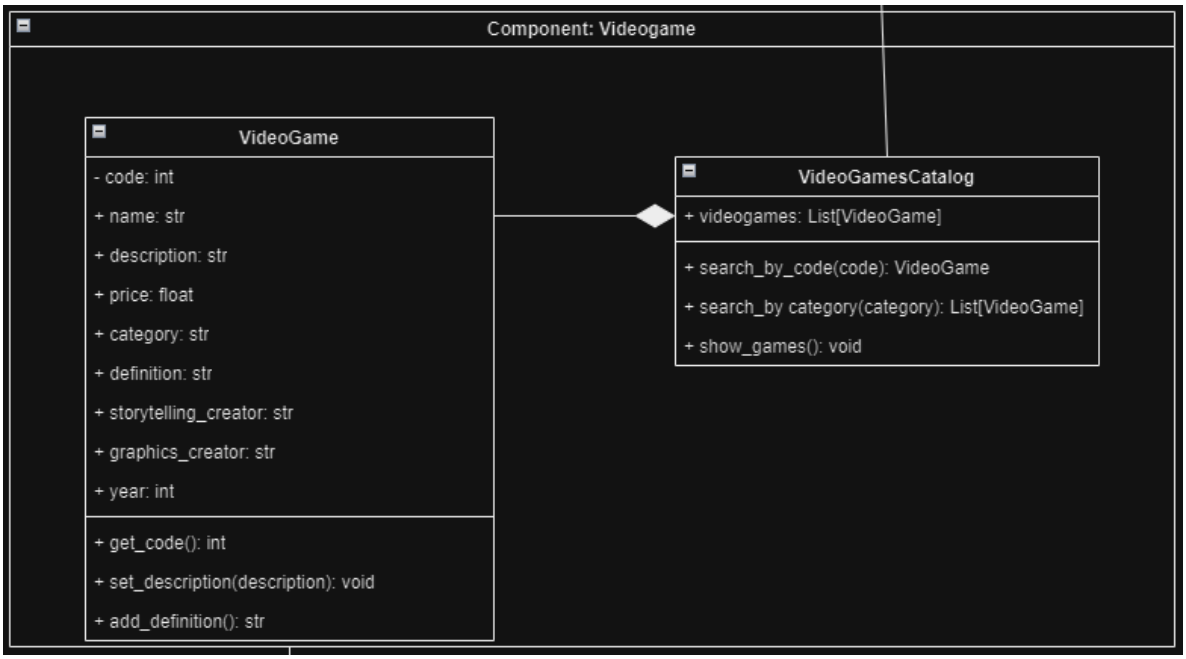
As the image is very large, it is recommended to view it from GitHub:

https://github.com/Cgamez28/Workshops_Software_modeling/blob/201879f527ce87ad911bdae8da62e19952522582/Workshop%202/ClassDiagramWS2.drawio.png

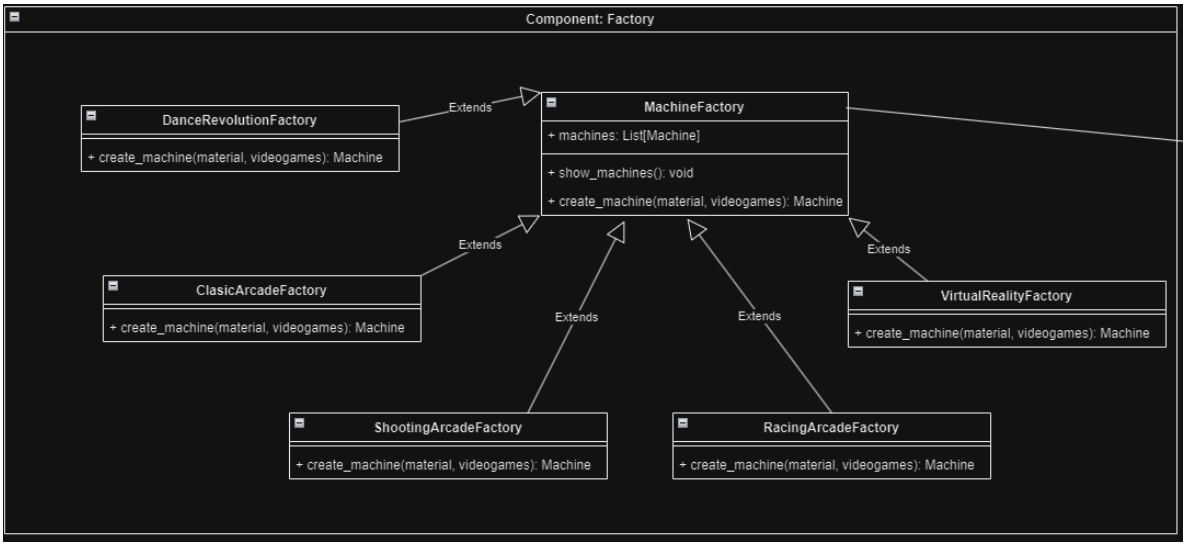
Component: User



Component: Videogame



Component: Factory



Component: Machine