Software-Modeling

Julian David Celis Giraldo – 20222020041

1. I have created a form in Microsoft Forms, with the answers I have recompiled the answers and with it I made the user stories
2. List User Stories:

   2.1 Clear game specifications: As a buyer, I want to see what games are included with the machine, to make sure I am interested in the titles.

   2.2 Joystick and button controls: As a buyer, I want the machine to have joysticks as well as buttons, for a more authentic gaming experience.

   2.3 Multi-console compatibility: As a buyer, I want to know which consoles the machine can emulate, so that I can play titles from different generations.

   2.4 Visual description of the design: As a buyer, I want the program to show me a description of the design of the machine, including color, material and lights when I finish setting it up, so I can imagine how it will look once it is installed in my space.

   2.5 Optimal machine condition:** As a buyer, I want to know that the machine is in good condition and has music and lights, for a more immersive experience.

   2.6 Strong but lightweight material: As a buyer, I want the machine to be made of strong but lightweight materials, so that it is easy to move and durable.

   2.7 Color customization: As a buyer, I want to be able to customize the colors of the controls and the housing of the machine to reflect my personal style.

   2.8 Sound customization options: As a buyer, I want to have the option of choosing different types of sound or music for the machine to suit my listening preferences.

   2.9 Modular design: As a buyer, I want the machine to have a modular design, so that I can easily upgrade or replace components.

   2.10 Support for software upgrades: As a buyer, I want the machine to allow software upgrades for new games or improvements, to keep it up to date over time.

2.11 Customizable lighting options: As a buyer, I want to be able to adjust the lights on the machine, both in color and intensity, to create different environments according to my preference.

2.12 Tactile feedback system: As a buyer, I want the machine to have a tactile feedback system on the controls for a more immersive and realistic gaming experience.

2.13 Add games by code: As a buyer, I want to be able to add games to the machine by code to customize my game selection.

object-oriented principles analysis

The design of the system effectively utilizes object-oriented principles. **Encapsulation** is demonstrated through the use of private attributes in classes like ArcadeMachine, which hides internal data such as _material, _color, _lights, and _sound. These attributes are accessed and modified only through public methods like add_game() and show_info(), ensuring controlled interaction with the internal state.

Abstraction is achieved by defining abstract classes and methods. The ArcadeMachine class serves as an abstract base class with abstract methods such as show_available_games() and is_game_valid(). These methods are not implemented in the base class but must be provided by subclasses like ModernArcadeMachine and RetroArcadeMachine, which offer specific implementations for different types of arcade machines.

Inheritance is used to extend the functionality of the base class. Both ModernArcadeMachine and RetroArcadeMachine inherit from ArcadeMachine, reusing its attributes and methods while adding their own specific features. This allows for code reuse and the extension of base functionality to create specialized machine types.

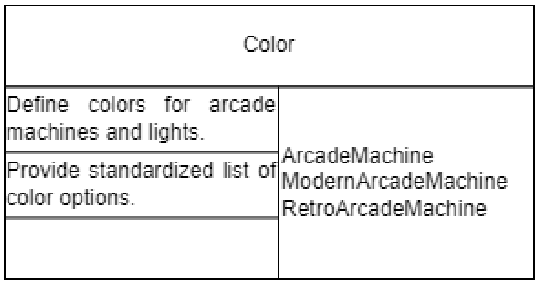
Polymorphism is demonstrated through method overriding. For instance, the show_available_games() method is defined in both ModernArcadeMachine and RetroArcadeMachine, but each class provides a different implementation tailored to its type of machine. This enables the use of the same method name with varying implementations depending on the object's type.

The design also adheres to the **Single Responsibility Principle**, with each class focused on a distinct aspect of the system. The Customer class handles customer information, while the ArcadeCatalog class manages the selection and customization of arcade machines. This separation ensures that each class has a clear and specific purpose.

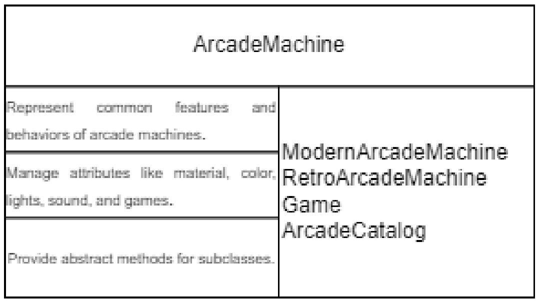
Finally, **low coupling** and **high cohesion** are evident throughout the design. Classes interact through well-defined methods and interfaces, minimizing dependencies. For example, the ArcadeCatalog class handles the overall process of machine customization and purchase, relying on other classes like ArcadeMachine, Game, and Customer for specific tasks. This modular approach reduces inter-class dependencies and enhances the overall organization and maintainability of the system.

Overall, the system's design effectively employs object-oriented principles to create a flexible, reusable, and well-structured architecture.

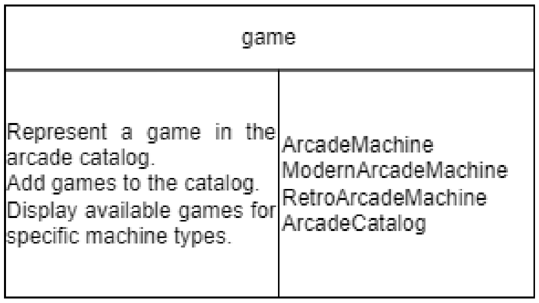
CRC card (Class Responsibility Collaborator): The cars were made in Draw.io



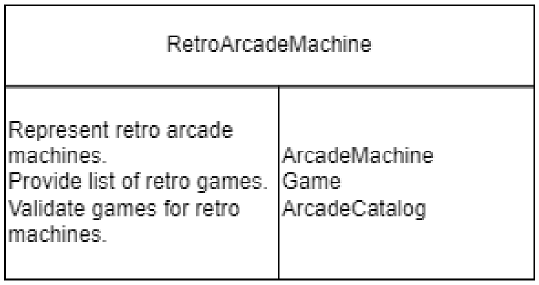
Color	
Define colors for arcade machines and lights. Provide standardized list of color options.	ArcadeMachine ModernArcadeMachine RetroArcadeMachine



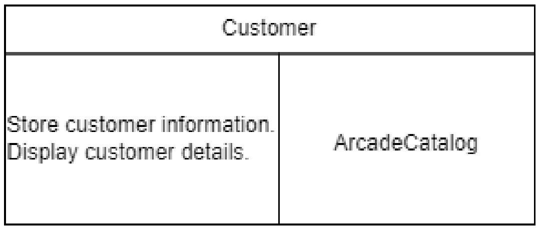
ArcadeMachine	
Represent common features and behaviors of arcade machines. Manage attributes like material, color, lights, sound, and games. Provide abstract methods for subclasses.	ModernArcadeMachine RetroArcadeMachine Game ArcadeCatalog



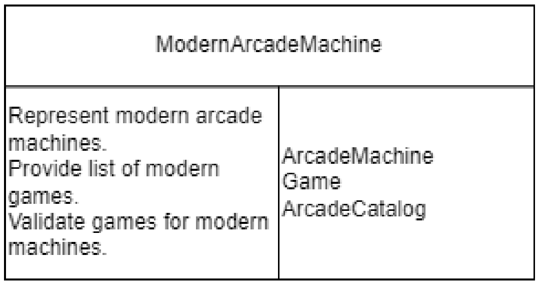
game	
Represent a game in the arcade catalog. Add games to the catalog. Display available games for specific machine types.	ArcadeMachine ModernArcadeMachine RetroArcadeMachine ArcadeCatalog



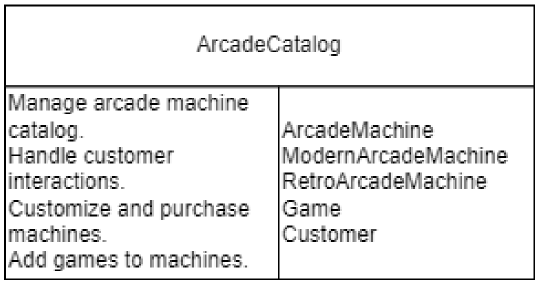
RetroArcadeMachine	
Represent retro arcade machines. Provide list of retro games. Validate games for retro machines.	ArcadeMachine Game ArcadeCatalog



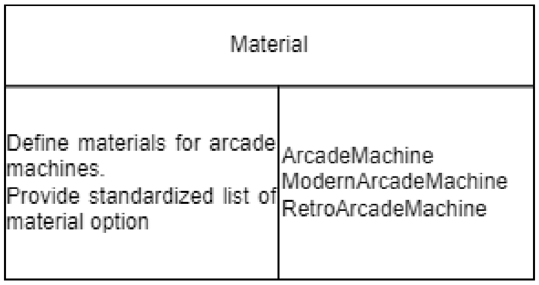
Customer	
Store customer information. Display customer details.	ArcadeCatalog



ModernArcadeMachine	
Represent modern arcade machines. Provide list of modern games. Validate games for modern machines.	ArcadeMachine Game ArcadeCatalog



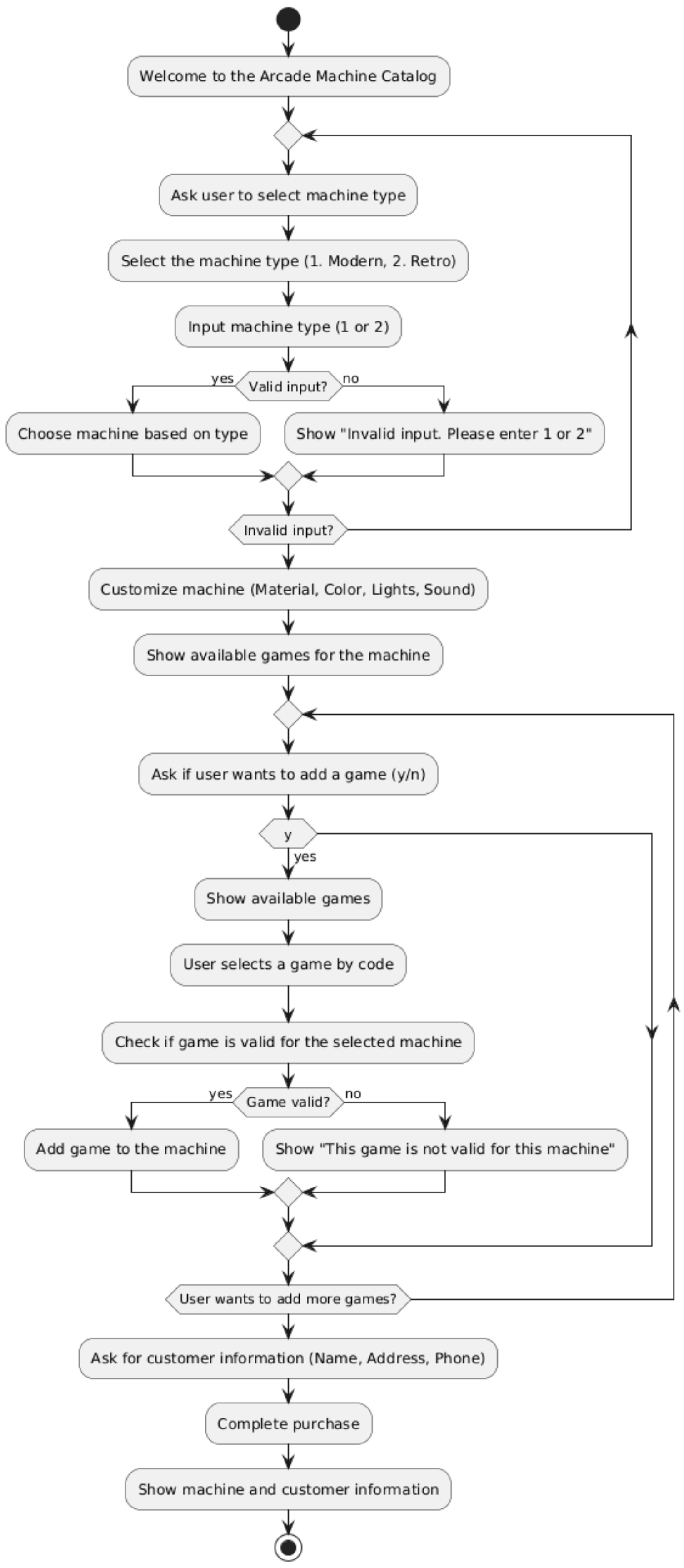
ArcadeCatalog	
Manage arcade machine catalog. Handle customer interactions. Customize and purchase machines. Add games to machines.	ArcadeMachine ModernArcadeMachine RetroArcadeMachine Game Customer



Material	
Define materials for arcade machines. Provide standardized list of material option	ArcadeMachine ModernArcadeMachine RetroArcadeMachine

Diagrams: The diagrams were developed in the tool planttext.com

Activity Diagram:


Welcome to the Arcade Machine Catalog
Ask user to select machine type
Select the machine type (1. Modern, 2. Retro)
Input machine type (1 or 2)
Valid input?
yes
no
Choose machine based on type
Show "Invalid input. Please enter 1 or 2"
Invalid input?
Customize machine (Material, Color, Lights, Sound)
Show available games for the machine
Ask if user wants to add a game (y/n)
y
yes
Show available games
User selects a game by code
Check if game is valid for the selected machine
Game valid?
yes
no
Add game to the machine
Show "This game is not valid for this machine"
User wants to add more games?
Ask for customer information (Name, Address, Phone)
Complete purchase
Show machine and customer information

Secuence Diagram:

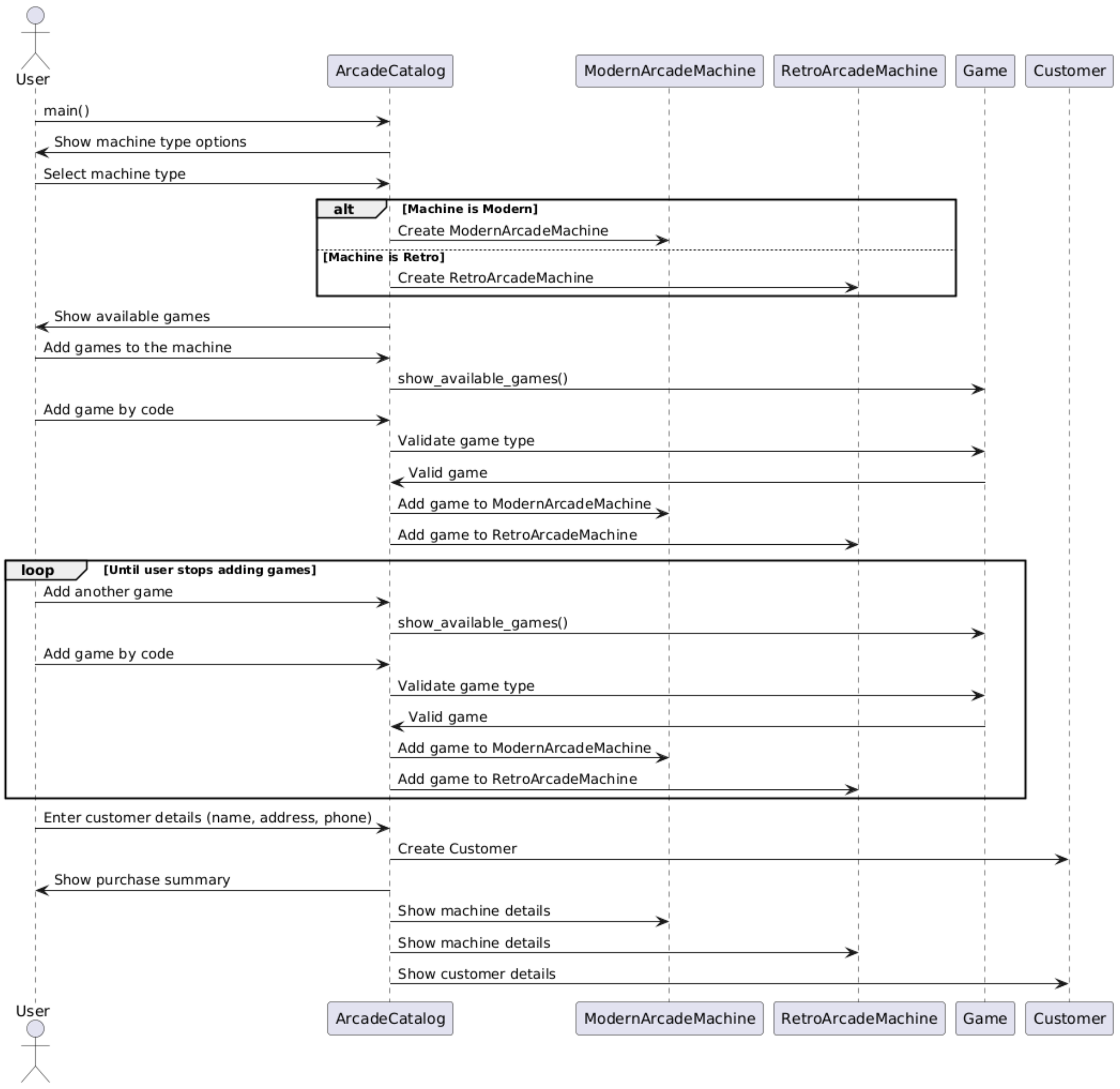
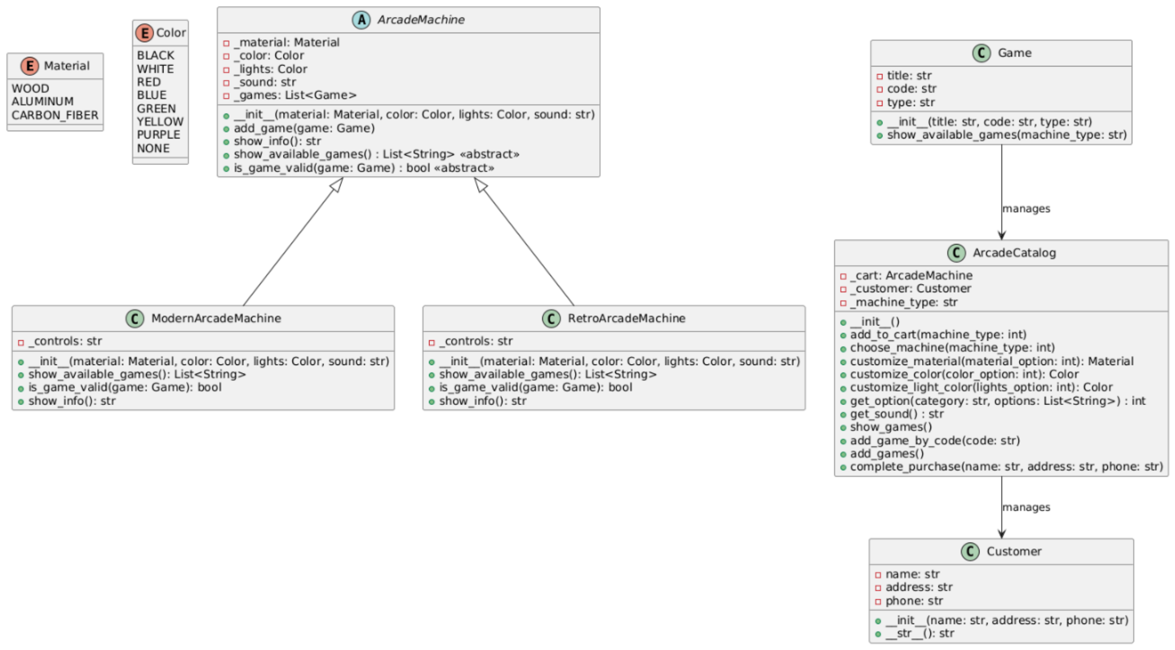


Diagrama Class


Material
WOOD
ALUMINUM
CARBON_FIBER
Color
BLACK
WHITE
RED
BLUE
GREEN
YELLOW
PURPLE
NONE
ArcadeMachine
_material: Material
_color: Color
_lights: Color
_sound: str
_games: List<Game>
__init__(material: Material, color: Color, lights: Color, sound: str)
add_game(game: Game)
show_info(): str
show_available_games() : List<String> «abstract»
is_game_valid(game: Game) : bool «abstract»
ModernArcadeMachine
_controls: str
__init__(material: Material, color: Color, lights: Color, sound: str)
show_available_games(): List<String>
is_game_valid(game: Game): bool
show_info(): str
RetroArcadeMachine
_controls: str
__init__(material: Material, color: Color, lights: Color, sound: str)
show_available_games(): List<String>
is_game_valid(game: Game): bool
show_info(): str
Game
title: str
code: str
type: str
__init__(title: str, code: str, type: str)
show_available_games(machine_type: str)
manages
ArcadeCatalog
_cart: ArcadeMachine
_customer: Customer
_machine_type: str
__init__()
add_to_cart(machine_type: int)
choose_machine(machine_type: int)
customize_material(material_option: int): Material
customize_color(color_option: int): Color
customize_light_color(lights_option: int): Color
get_option(category: str, options: List<String>) : int
get_sound() : str
show_games()
add_game_by_code(code: str)
add_games()
complete_purchase(name: str, address: str, phone: str)
manages
Customer
name: str
address: str
phone: str
__init__(name: str, address: str, phone: str)
__str__(): str

Enumerations (Material, Color): Define the materials and colors that can be used for arcade machines.

Abstract Class (ArcadeMachine): Basic attributes and methods for an arcade machine, including abstract methods that must be implemented by the concrete subclasses. Concrete Classes (ModernArcadeMachine, RetroArcadeMachine): Implement the abstract methods of ArcadeMachine and add attributes and behaviors specific to modern and retro machines, changing the types of controls depending on the version of the machine. Game Class: Represents a game in the catalog, with attributes such as title, code, and type.

Customer Class: Stores customer information to complete the purchase, such as name, address, and phone number.

ArcadeCatalog Class: Manages the arcade machine catalog and customer interaction, including machine customization and game handling.