# Design Overview for Idimon

Name: Nguyen Van Huy Quang

Student ID: SWS00658

# Summary of Program

This program is a turn-based RPG game where players explore a dynamic world filled with unique creatures called Idimons. Players can encounter, battle, and catch Idimons to add them to their team. Each Idimon has its own rank and appearance rate, adding an element of rarity and strategy to the game.

Players move around the map, which includes different objects such as grass (where Idimon encounters occur) and healers (to restore health). By pressing "X," players can open the game menu, which includes options to view Idimon statuses, check inventory, and manage their team. The "Z" key is used to interact with objects on the map. In battles, players use their Idimons to fight wild Idimons, gaining experience and potentially evolving them. Idimons can be caught using various candies, each with its success rate.

[Player is moving around the map with animation]



Player encounters an Idimon in the grass!

 A screenshot of a video game

Description automatically generated

[Battle Menu]

A diagram of a system

Description automatically generated

[Catch Idimon]

A diagram of a system

Description automatically generated

[Game Menu]

A diagram of a flowchart

Description automatically generated

[Map]

A diagram of a game

Description automatically generated

# Required Roles

Describe each of the classes, interfaces, and any enumerations you will create. Use a different table to describe each role you will have, using the following table templates.

Table 1: Player Details

|  |  |  |
| --- | --- | --- |
| Responsibility | Type Details | Notes |
| **Position** | Field: Vector2 | Player's position on the map |
| **Move** | Method: void LoadAnimation (Vector2 direction) | Updates player's position |
| **Animate** | Method: void Animate() | Handles player animation |
| **OpenMenu** | |  | | --- | | Method: void OpenMenu() | | Opens the game menu |
| **Interact** | Method: void Interact() | Interacts with objects on the map |

Table 2: Idimon Details

|  |  |  |
| --- | --- | --- |
| **Responsibility** | **Type Details** | **Notes** |
| Name | Field: string | Idimon's name |
| Level | Field: int | Idimon's level |
| Experience | Field: int | Idimon's experience points |
| MaxExperience | Field: int | Experience needed to level up |
| Health | Field: int | Idimon's current health |
| MaxHealth | Field: int | Idimon's maximum health |
| Evolve | Method: void Evolve() | Handles Idimon's evolution |
| GainExperience | Method: void GainExperience(int exp) | Adds experience to Idimon |
| IsCaught | Field: bool | Indicates if Idimon is caught |

Table 3: Inventory Details

|  |  |  |
| --- | --- | --- |
| **Responsibility** | **Type Details** | **Notes** |
| Items | Field: List<Item> | List of items in inventory |
| AddItem | Method: void AddItem(Item item) | Adds item to inventory |
| UseItem | Method: void UseItem(Item item) | Uses selected item |
| DisplayInventory | Method: void DisplayInventory() | Displays inventory to player |

Table 4: Item Details

|  |  |  |
| --- | --- | --- |
| **Responsibility** | **Type Details** | **Notes** |
| Name | Field: string | Item name |
| Quantity | Field: int | Quantity of item |
| Use | Method: void Use() | Uses the item |

Table 5: GameMenu Details

|  |  |  |
| --- | --- | --- |
| **Responsibility** | **Type Details** | **Notes** |
| Open | Method: void Open() | Opens the game menu |
| DisplayIdimonStatus | Method: void DisplayIdimonStatus() | Displays Idimon status |
| DisplayInventory | Method: void DisplayInventory() | Displays inventory |
| DisplayTeam | Method: void DisplayTeam() | Displays team management |

Table 6: MapObject details

|  |  |  |
| --- | --- | --- |
| **Responsibility** | **Type Details** | **Notes** |
| Position | Field: Vector2 | Object's position on the map |
| Interact | Method: void Interact(Player player) | Interaction logic with player |

Table 7: BattleSystem details

|  |  |  |
| --- | --- | --- |
| **Responsibility** | **Type Details** | **Notes** |
| StartBattle | Method: void StartBattle(Idimon playerIdimon, Idimon wildIdimon) | Initializes battle |
| PerformTurn | Method: void PerformTurn(Action action) | Handles player's turn in battle |
| CalculateExperience | Method: int CalculateExperience(Idimon defeatedIdimon) | Calculates experience gained |
| CaptureIdimon | Method: bool CaptureIdimon(Idimon wildIdimon, Item candy) | Attempts to capture Idimon |

Table 8: Blocks details

|  |  |
| --- | --- |
| **Value** | **Notes** |
| GrassBlock | Grass block map object |
| HealerBlock | Healer block map object |

Table 9: ActionType details

|  |  |
| --- | --- |
| **Value** | **Notes** |
| Attack | Perform an attack |
| UseItem | Use an item |
| SwitchIdimon | Switch to another Idimon |
| Run | Attempt to run away from battle |

Table 10: IdimonsData details

|  |  |
| --- | --- |
| **Value** | **Notes** |
| Tiger | Idimon name Tiger |
| Student | Idimon name Student |
| MultilevelSeller | Idimon name MultilevelSeller |
| Idiot | Idimon name Idiot |

Table 11: MenuType details

|  |  |
| --- | --- |
| **Value** | **Notes** |
| IdimonMenu | Menu for Idimon |
| InventoryMenu | Menu for inventory to display items |
| MainMenu | Menu when starting the game |
| GameMenu | Menu to store others Menu |

Design Pattern

- Template Method Pattern: The Idimons abstract class defines a skeleton of algorithms in the form of abstract and concrete methods. Subclasses like Student and tiger then fill out or override these methods. For example:

* LevelUp(), TakeDamage(), and Heal() are defined in the base class.
* Evolve() is an abstract method that subclasses must implement.

- State Pattern:

 The IsFainted() method checks the state of the Idimon, which could be seen as a very basic form of the State pattern.

 The player's direction state is managed using the \_currentDirection variable, which affects how the player is drawn and moves. This resembles a simple State pattern,

- Abstract Factory Method Pattern:

* The Screens class serves as an abstract base class that defines a common interface for all screen types (e.g., BattleScreen, GameScreen, TitleScreen). This approach allows for different implementations of the Update, Draw, and HandleInput methods while maintaining a consistent interface.
* By defining these methods as abstract, subclasses are forced to provide specific implementations, ensuring that each screen adheres to the expected behavior.

- Singleton Pattern: The Player class uses the Singleton pattern to ensure only one instance of the Player exists. This is implemented through:

* A private static \_instance variable
* A private constructor
* A public static GetInstance method that creates the instance if it doesn't exist or returns the existing instance

# Class Diagram

# Sequence Diagram

Provide a sequence diagram showing how your proposed classes will interact to achieve a specific piece of functionality in your program.