Design Overview for Idimon

Name: Nguyen Van Huy Quang
Student ID: SWS00658

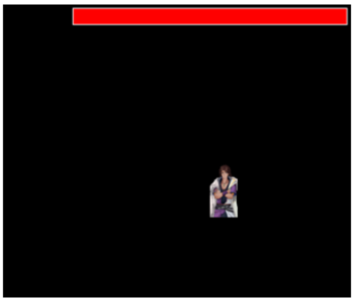
Summary of Program

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.

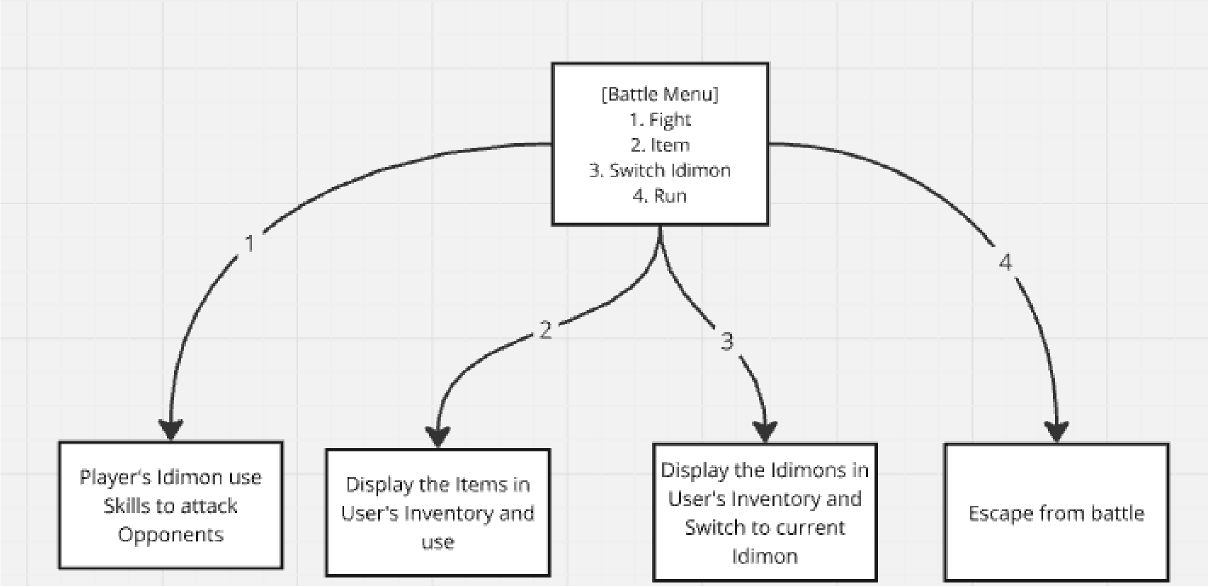
Players move around the map, which includes different objects such as grass (where Idimon encounters occur) and healers (to restore health). By pressing "X," players can open the game menu, which includes options to view Idimon statuses, check inventory, and manage their team. The "Z" key is used to 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]

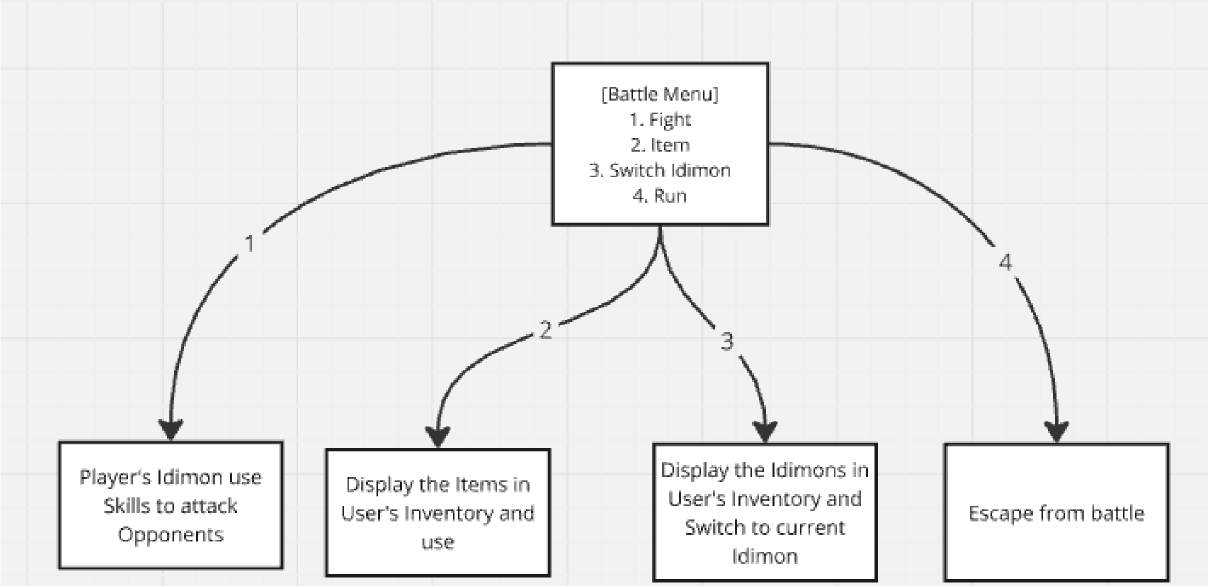
Player encounters an Idimon in the grass!



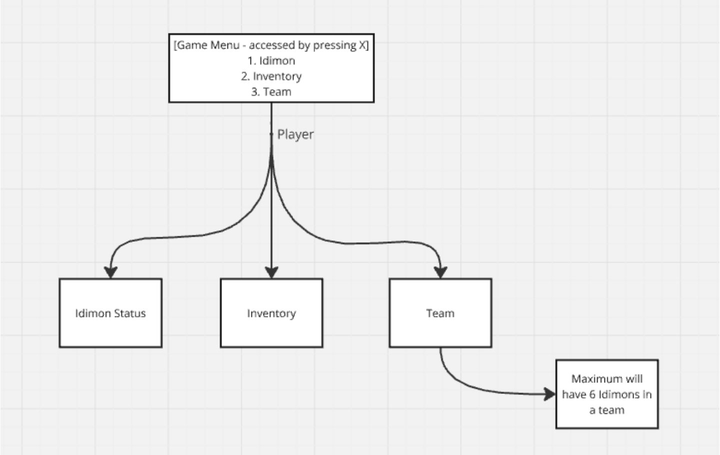
[Battle Menu]


[Battle Menu]
1. Fight
2. Item
3. Switch Idimon
4. Run
1
2
3
4
Player's Idimon use Skills to attack Opponents
Display the Items in User's Inventory and use
Display the Idimons in User's Inventory and Switch to current Idimon
Escape from battle

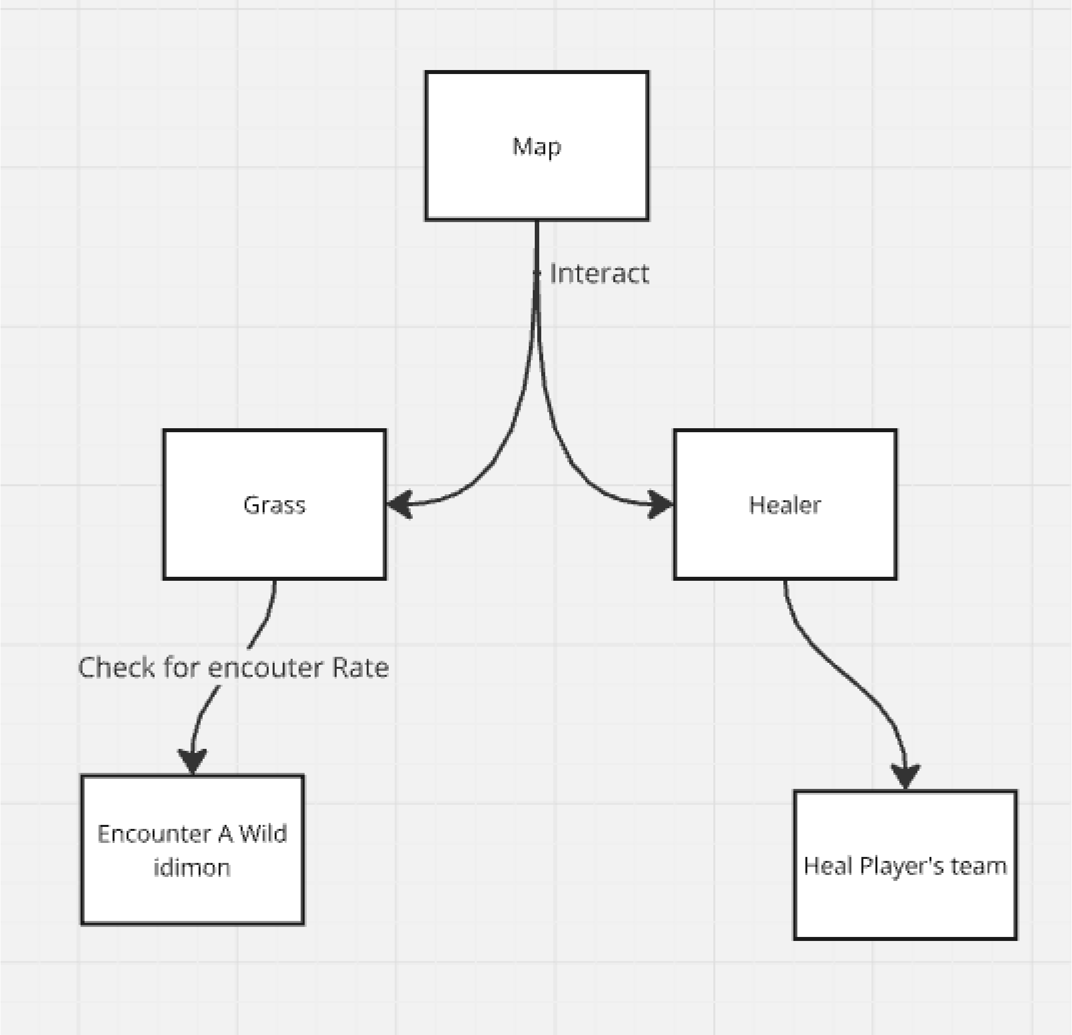
[Catch Idimon]


[Battle Menu]
1. Fight
2. Item
3. Switch Idimon
4. Run
1
2
3
4
Player's Idimon use Skills to attack Opponents
Display the Items in User's Inventory and use
Display the Idimons in User's Inventory and Switch to current Idimon
Escape from battle

[Game Menu]


[Game Menu - accessed by pressing X]
1. Idimon
2. Inventory
3. Team
Player
Idimon Status
Inventory
Team
Maximum will have 6 Idimons in a team

[Map]



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,

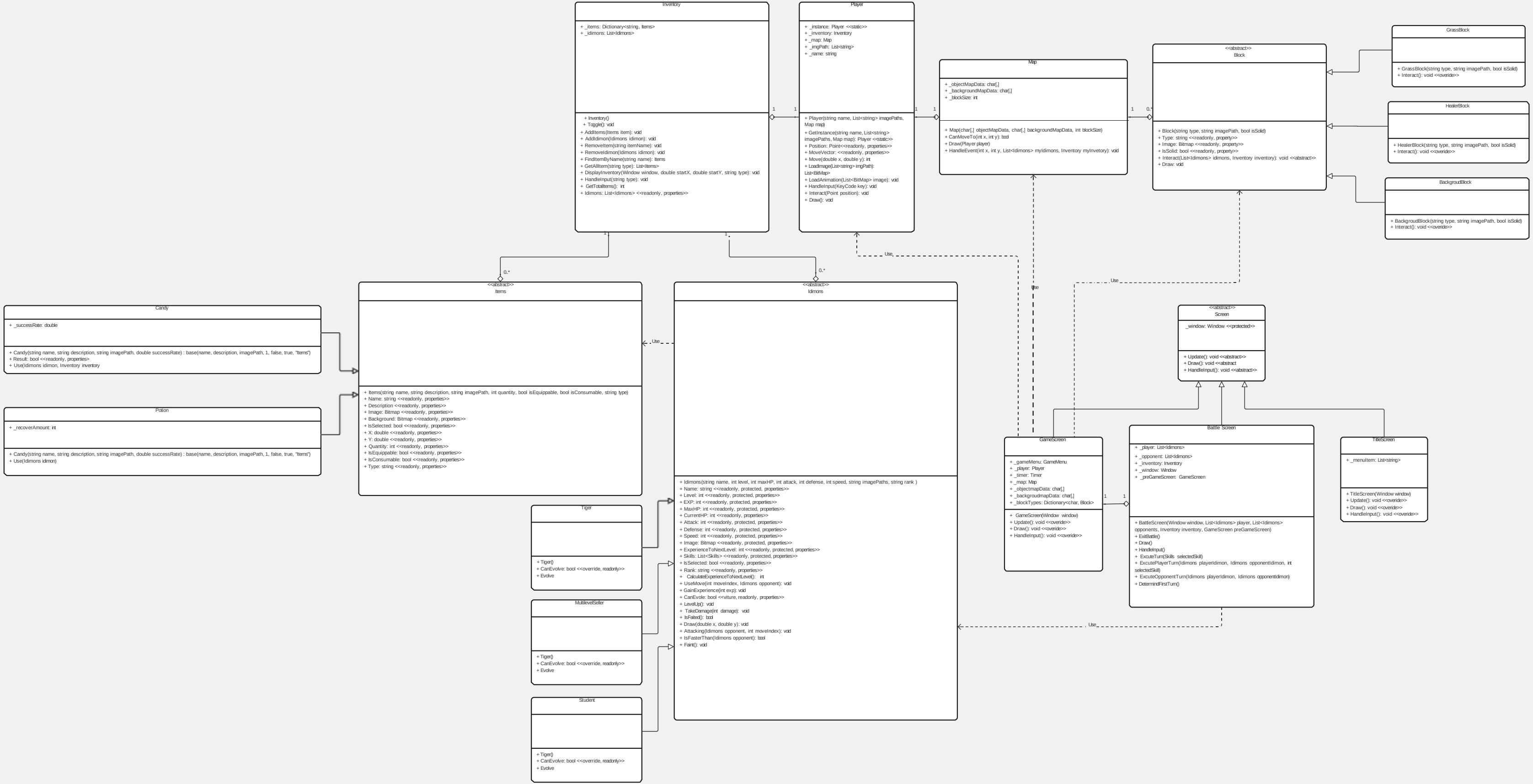
- Abstract Factory Method Pattern:

- 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.

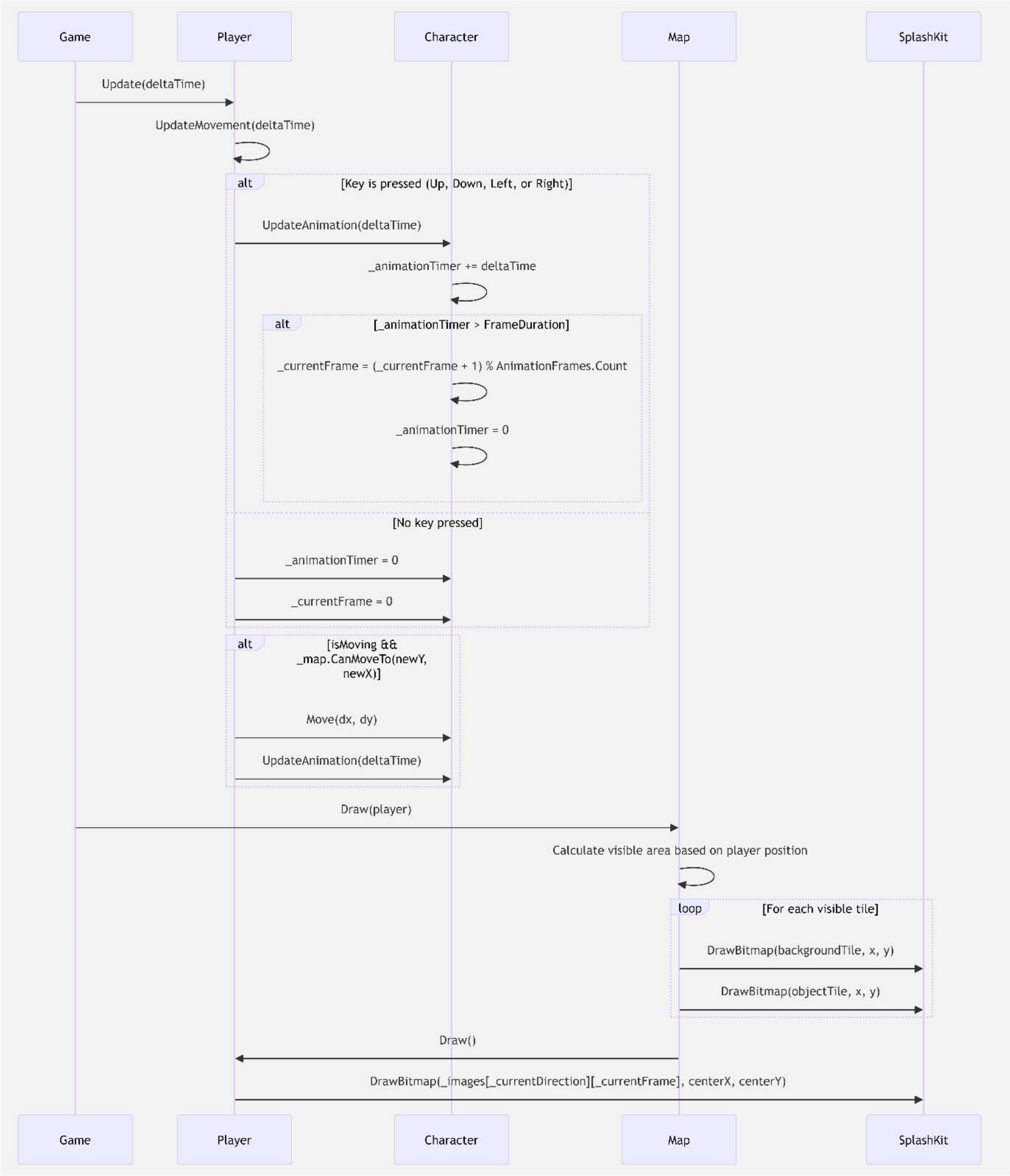
- Singleton Pattern: The Player class uses the Singleton pattern to ensure only one instance of the Player exists. This is implemented through:

- 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


Inventory
+ _items: Dictionary<string, Items>
+ _idimons: List<Idimons>
+ Inventory()
+ Toggle(): void
+ AddItems(Items item): void
+ AddIdimon(Idimons idimon): void
+ RemoveItem(string itemName): void
+ RemoveIdimon(Idimons idimon): void
+ FindItemByName(string name): Items
+ GetAllItem(string type): List<Items>
+ DisplayInventory(Window window, double startX, double startY, string type): void
+ HandleInput(string type): void
+ GetTotalItems(): int
+ Idimons: List<Idimons> <<readonly, properties>>
1
1
Player
+ _instance: Player <<static>>
+ _inventory: Inventory
+ _map: Map
+ _imgPath: List<string>
+ _name: string
+ Player(string name, List<string> imagePaths, Map map)
+ GetInstance(string name, List<string> imagePaths, Map map): Player <<static>>
+ Position: Point<<readonly, properties>>
+ MoveVector: <<readonly, properties>>
+ Move(double x, double y): int
+ LoadImage(List<string> imgPath): List<BitMap>
+ LoadAnimation(List<BitMap> image): void
+ HandleInput(KeyCode key): void
+ Interact(Point position): void
+ Draw(): void
1
1
Map
+ _objectMapData: char[,]
+ _backgroundMapData: char[,]
+ _blockSize: int
+ Map(char[,] objectMapData, char[,] backgroundMapData, int blockSize)
+ CanMoveTo(int x, int y): bool
+ Draw(Player player)
+ HandleEvent(int x, int y, List<Idimons> myIdimons, Inventory myInvetory): void
1
0..*
<<abstract>>
Block
+ Block(string type, string imagePath, bool isSolid)
+ Type: string <<readonly, property>>
+ Image: Bitmap <<readonly, property>>
+ IsSolid: bool <<readonly, property>>
+ Interact(List<Idimons> idimons, Inventory inventory): void <<abstract>>
+ Draw: void
GrassBlock
+ GrassBlock(string type, string imagePath, bool isSolid)
+ Interact(): void <<overide>>
HealerBlock
+ HealerBlock(string type, string imagePath, bool isSolid)
+ Interact(): void <<overide>>
BackgroudBlock
+ BackgroudBlock(string type, string imagePath, bool isSolid)
+ Interact(): void <<overide>>
1
1
0..*
0..*
Use
Use
Use
Candy
+ _successRate: double
+ Candy(string name, string description, string imagePath, double successRate) : base(name, description, imagePath, 1, false, true, "Items")
+ Result: bool <<readonly, properties>
+ Use(Idimons idimon, Inventory inventory
Potion
+ _recoverAmount: int
+ Candy(string name, string description, string imagePath, double successRate) : base(name, description, imagePath, 1, false, true, "Items")
+ Use(Idimons idimon)
<<abstract>>
Items
+ Items(string name, string description, string imagePath, int quantity, bool isEquippable, bool isConsumable, string type)
+ Name: string <<readonly, properties>>
+ Description <<readonly, properties>>
+ Image: Bitmap <<readonly, properties>>
+ Background: Bitmap <<readonly, properties>>
+ IsSelected: bool <<readonly, properties>>
+ X: double <<readonly, properties>>
+ Y: double <<readonly, properties>>
+ Quantity: int <<readonly, properties>>
+ IsEquippable: bool <<readonly, properties>>
+ IsConsumable: bool <<readonly, properties>>
+ Type: string <<readonly, properties>>
Use
<<abstract>>
Idimons
+ Idimons(string name, int level, int maxHP, int attack, int defense, int speed, string imagePaths, string rank)
+ Name: string <<readonly, protected, properties>>
+ Level: int <<readonly, protected, properties>>
+ EXP: int <<readonly, protected, properties>>
+ MaxHP: int <<readonly, protected, properties>>
+ CurrentHP: int <<readonly, properties>>
+ Attack: int <<readonly, protected, properties>>
+ Defense: int <<readonly, protected, properties>>
+ Speed: int <<readonly, protected, properties>>
+ Image: Bitmap <<readonly, protected, properties>>
+ ExperienceToNextLevel: int <<readonly, protected, properties>>
+ Skills: List<Skills> <<readonly, protected, properties>>
+ IsSelected: bool <<readonly, properties>>
+ Rank: string <<readonly, properties>>
+ CalculateExperienceToNextLevel(): int
+ UseMove(int moveIndex, Idimons opponent): void
+ GainExperience(int exp): void
+ CanEvole: bool <<viture, readonly, properties>>
+ LevelUp(): void
+ TakeDamage(int damage): void
+ IsFaited(): bool
+ Draw(double x, double y): void
+ Attacking(Idimons opponent, int moveIndex): void
+ IsFasterThan(Idimons opponent): bool
+ Faint(): void
Tiger
+ Tiger()
+ CanEvolve: bool <<override, readonly>>
+ Evolve
MultilevelSeller
+ Tiger()
+ CanEvolve: bool <<override, readonly>>
+ Evolve
Student
+ Tiger()
+ CanEvolve: bool <<override, readonly>>
+ Evolve
<<abstract>>
Screen
_window: Window <<protected>>
+ Update(): void <<abstract>>
+ Draw(): void <<abstract
+ HandleInput(): void <<abstract>>
GameScreen
+ _gameMenu: GameMenu
+ _player: Player
+ _timer: Timer
+ _map: Map
+ _objectmapData: char[,]
+ _backgroudmapData: char[,]
+ _blockTypes: Dictionary<char, Block>
+ GameScreen(Window window)
+ Update(): void <<overide>>
+ Draw(): void <<overide>>
+ HandleInput(): void <<overide>>
1
1
Battle Screen
+ _player: List<Idimons>
+ _opponent: List<Idimons>
+ _inventory: Inventory
+ _window: Window
+ _preGameScreen: GameScreen
+ BattleScreen(Window window, List<Idimons> player, List<Idimons> opponents, Inventory inventory, GameScreen preGameScreen)
+ ExitBattle()
+ Draw()
+ HandleInput()
+ ExcuteTurn(Skills selectedSkill)
+ ExcutePlayerTurn(Idimons playerIdimon, Idimons opponentIdimon, int selectedSkill)
+ ExcuteOpponentTurn(Idimons playerIdimon, Idimons opponentIdimon)
+ DetermindFirstTurn()
Use
TitleScreen
+ _menuItem: List<string>
+ TitleScreen(Window window)
+ Update(): void <<overide>>
+ Draw(): void <<overide>>
+ HandleInput(): void <<overide>>

Sequence Diagram


Game
Player
Character
Map
SplashKit
Update(deltaTime)
UpdateMovement(deltaTime)
alt
[Key is pressed (Up, Down, Left, or Right)]
UpdateAnimation(deltaTime)
_animationTimer += deltaTime
alt
[_animationTimer > FrameDuration]
_currentFrame = (_currentFrame + 1) % AnimationFrames.Count
_animationTimer = 0
[No key pressed]
_animationTimer = 0
_currentFrame = 0
alt
[isMoving && _map.CanMoveTo(newY, newX)]
Move(dx, dy)
UpdateAnimation(deltaTime)
Draw(player)
Calculate visible area based on player position
loop
[For each visible tile]
DrawBitmap(backgroundTile, x, y)
DrawBitmap(objectTile, x, y)
Draw()
DrawBitmap(_images[_currentDirection][_currentFrame], centerX, centerY)
Game
Player
Character
Map
SplashKit