System Requirements Specification Index

For

Game Development Utility System

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

TABLE OF CONTENTS

Game Development Utility System

System Requirements Specification

1 Project Abstract

Pixel Perfect Studios is developing a new game engine and needs a utility system for data processing and game mechanics. This assignment focuses on implementing lambda functions to handle game-related calculations, player data transformations, and entity filtering for efficient game operations. Lambda functions will provide concise, reusable operations for common game development tasks.

2 Business Requirements:

Screen Name	Console input screen
Problem Statement	1. System must utilize lambda functions for game data processing 2. Code must demonstrate proper lambda function syntax and usage 3. Functions must be called with correct arguments 4. System must perform operations on Player statistics and inventories, Game entity properties and filtering, Score calculations and leaderboards

3 Constraints

3.1 Lambda Function Requirements

1. Lambda Syntax:
    - Must use the keyword `lambda` followed by parameters and expression
    - Must be single-expression functions
    - Example: `lambda player: player["health"] * 2`

2. Usage Contexts:
    - Must use lambda functions with built-in functions (map, filter, sorted)
    - Must use lambda functions as function arguments
    - Must assign at least one lambda function to a variable
    - Must use at least one lambda function in a list comprehension
3. Parameter Types:
    - Must demonstrate lambda functions with single parameter
    - Must demonstrate lambda functions with multiple parameters
    - Must demonstrate lambda functions that operate on different data types
4. Return Values:
    - Lambda functions must return appropriate data types
    - Some lambda functions must return boolean values for entity filtering
    - Some lambda functions must return numeric values for game calculations

3.2 INPUT REQUIREMENTS

1. Player Data:
    - Input list must contain at least 5 player dictionaries
    - Each player must have attributes: name, level, health, mana, score
    - Example: `{"name": "Wizard1", "level": 5, "health": 80, "mana": 100, "score": 2500}`
2. Game Entities:
    - Input list must contain at least 8 game entity dictionaries
    - Each entity must have: id, type, position_x, position_y, active
    - Example: `{"id": "E001", "type": "enemy", "position_x": 100, "position_y": 200, "active": True}`
3. Item Inventory:
    - Input list must contain at least 6 item dictionaries
    - Each item must have: name, type, value, rarity, equipped
    - Example: `{"name": "Magic Sword", "type": "weapon", "value": 500, "rarity": "rare", "equipped": False}`

4. Game Coordinates:
    - Input list must contain at least 5 coordinate tuples

- Example: `[(10, 20), (50, 60), (30, 40), (70, 80), (90, 10)]`

3.3 Operations Constraints

1. Player Transformations:
    - Must use lambda with map() to transform player statistics
    - Must convert map result to list to display values
    - Example: `list(map(lambda p: p["health"] * 1.5, players))`
2. Entity Filtering:
    - Must use lambda with filter() to select specific entities
    - Must demonstrate filtering with different criteria
    - Example: `list(filter(lambda e: e["type"] == "enemy" and e["active"], entities))`
3. Item Sorting:
    - Must use lambda with sorted() to order items
    - Must demonstrate custom sorting keys
    - Example: `sorted(inventory, key=lambda item: item["value"])`
4. Game Calculations:
    - Must demonstrate using lambda functions for game mechanics
    - Must show calculating distances, damage, or other game values
    - Example: `lambda pos1, pos2: ((pos1[0] - pos2[0])**2 + (pos1[1] - pos2[1])**2)**0.5`

3.4 Output Constraints

1. Display Format:
    - Each operation result must have a descriptive game-related label
    - Results must be formatted for readability
    - Must include before and after data states

3. Output Sections:
    - Player transformation results
    - Game entity filtering results
    - Item sorting results
    - Game mechanic calculation results

4. TEMPLATE CODE STRUCTURE:

1. Game Data Preparation:

- `prepare_player_data()` - creates list of players for processing
- `prepare_entity_data()` - creates list of game entities
- `prepare_inventory_data()` - creates list of game items
- `prepare_coordinate_data()` - creates list of game world coordinates

2. Lambda Function Operations:

- `demonstrate_player_transformations()` - shows lambda functions for player data
- `demonstrate_entity_filtering()` - shows lambda functions for filtering entities
- `demonstrate_item_sorting()` - shows lambda functions for inventory management
- `demonstrate_game_calculations()` - shows lambda functions for game mechanics

3. Advanced Operations:

- `demonstrate_ability_system()` - shows lambda functions for game abilities
- `demonstrate_combat_system()` - shows lambda functions for damage calculations
- `demonstrate_level_system()` - shows lambda functions for experience and leveling

4. Main Program Function:

- `main()` - main program function executing all demonstrations
- Must execute each demonstration function
- Must present results in structured format

5. EXECUTION STEPS TO FOLLOW:

1. Run the program
2. Observe player data transformations with lambda functions
3. See game entity filtering with lambda functions
4. View inventory sorting with lambda functions
5. Observe game mechanic calculations with lambda functions
6. See ability system demonstrations
7. View combat system calculations
8. Observe level system progression calculations