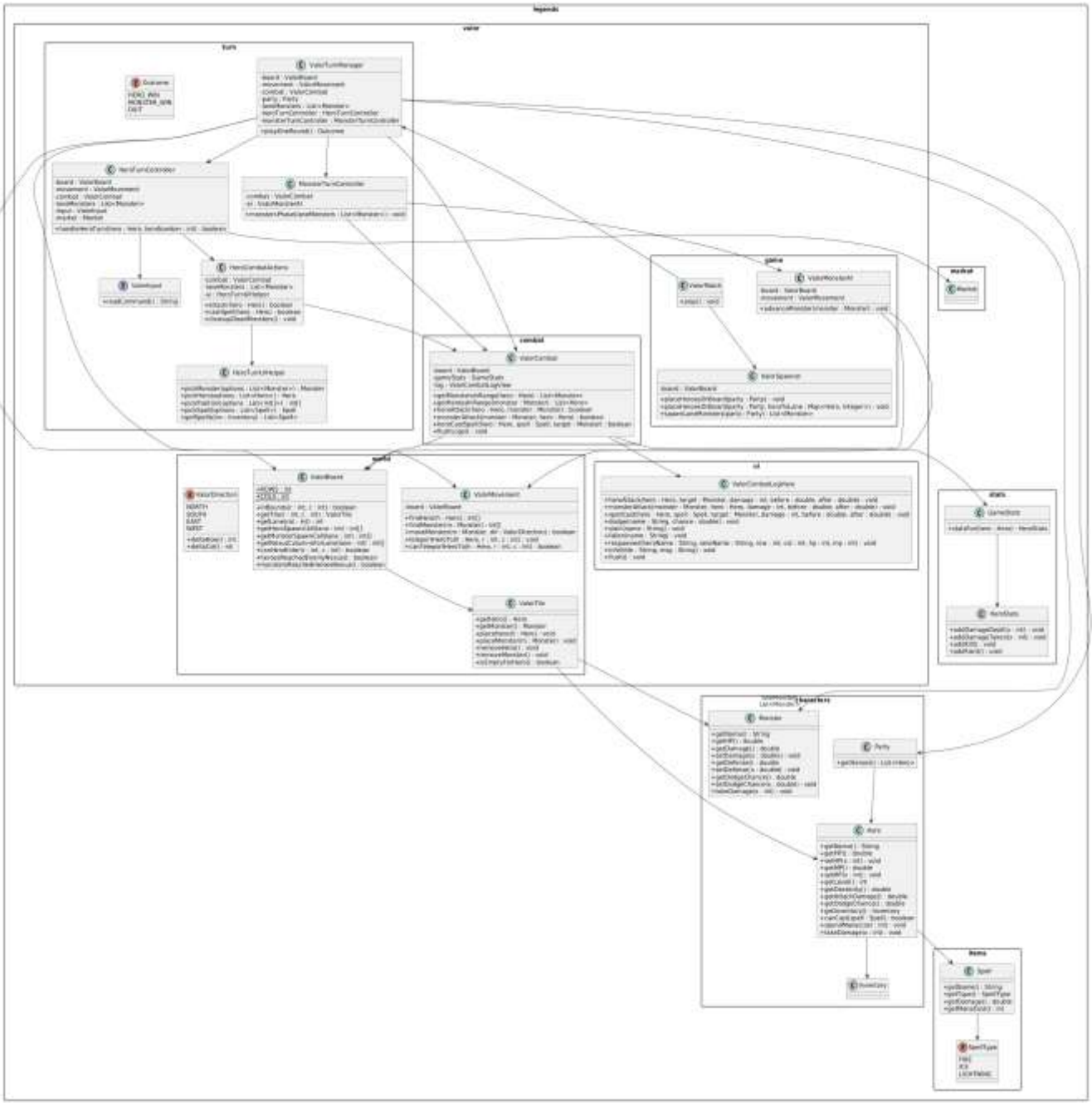
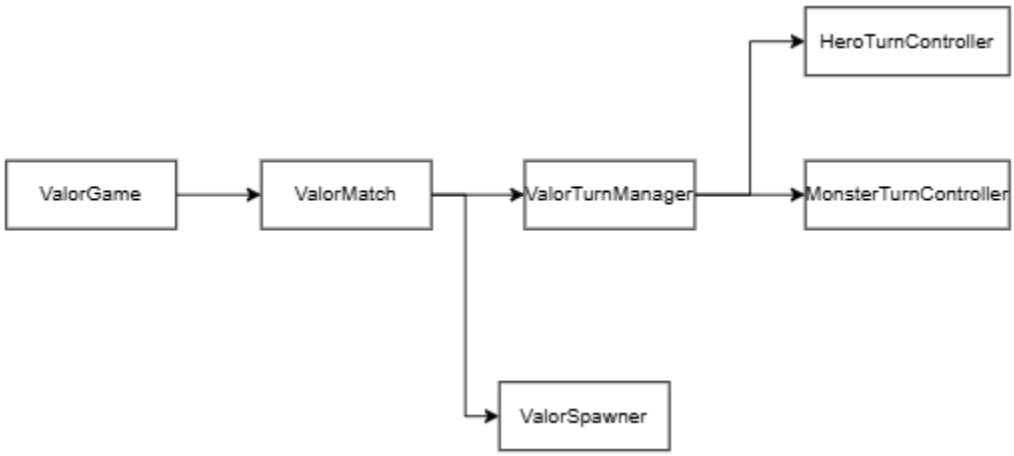
# Legends of Valor – Design Document

## UML Diagram






## Overall Architecture (Scalability)

The system is organized into clearly separated layers. ValorGame and ValorMatch handle high-level game orchestration and the match loop. ValorTurnManager coordinates each round by sequencing hero and monster phases and checking win conditions. Action controllers (HeroTurnController, MonsterTurnController) manage turn flow and input handling. Core game rules are isolated in domain classes such as ValorMovement and ValorCombat. UI and view helpers are kept separate from logic, ensuring the system scales without creating large, monolithic classes.

## Extensibility Decisions

Hero behavior is intentionally split into specialized action components (combat, movement, equipment), accessed through a single facade (HeroActionService). This allows new hero abilities or mechanics to be added without modifying the turn controller. Monster behavior is encapsulated in a separate AI class (ValorMonsterAI), enabling different strategies or difficulty levels to be introduced independently. Overall, new features can be extended by adding new classes rather than rewriting existing ones.

## Project Structure

### legends.game

- **Main**
  Entry point. Creates a LegendsApp instance and calls run().
- **LegendsApp** Serves as the main entry point for the Legends application: holds Game.
  - Displays the top-level game selection menu.
  - Accepts and validates user input for game choice.
  - Launches the selected game mode.
  - Manages application-level control flow and exit behavior.
- **LegendsGame**
  Orchestrates the whole game for Legends: Heroes & Monsters: holds the current GameState, the WorldMap, Party, and Market.
  - Initializes map (MapGenerator), market (Market), and data (DataLoader).
  - Runs the main game loop (gameLoop) which repeatedly calls render, reads input, handleInput, and update on the current state.
  - Shows the intro screen and triggers hero selection before entering exploration.
- **GameState (interface)**
  Defines the contract for all game states:
  - void render()
  - void update(LegendsGame game)
  - void handleInput(String input)
  - boolean isFinished()
- **ExplorationState**
  Handles overworld exploration:
  - Shows the map and party position.
  - Accepts movement inputs (W/A/S/D), I for inventory, Q to quit.
  - Triggers markets when stepping on M tiles.
  - Randomly starts battles based on tile type.

- **BattleState**
  Turn-based combat controller:
  - Manages heroes' turn order and monsters' actions.
  - Offers actions: Attack, Cast Spell, Use Potion, Change Equipment, Show Stats, Flee.
  - Applies damage, dodge chance, spell effects and end-of-round regeneration.
  - On victory: rewards XP and gold, revives fainted heroes, then returns to exploration.
  - On defeat: ends the game.
- **InventoryState**
  Inventory management:
  - First screen: choose which hero's inventory to view.
  - Second screen: per-hero inventory showing items with aligned table formatting.
  - Allows equipping weapons/armor and using potions.
- **MarketState**
  Market interaction:
  - Main market menu: Buy, Sell, Back.
  - Buy menu: choose category (Weapons, Armor, Potions, Spells), then item and buyer hero.
  - Sell menu: choose hero, category, and item to sell at 50% value.
  - Uses Market getters and hero inventory.
- **HeroSelection**
  Hero selection flow:
  - Loads Warriors, Paladins, Sorcerers via DataLoader.
  - Enforces PDF rules: HP = level × 100, MP = level × 50 at game start.
  - Prints grouped hero list with stats and ANSI colors.
  - Allows the player to choose between 1 and 3 heroes.
  - Returns a fully constructed Party.

**legends.valor** ***(only high-level summary here)***

- **valor.combat - ValorCombat** Implements core combat mechanics for Legends of Valor encounters. Initializes ValorBoard and GameStats.
- Determine valid targets within attack/cast range on the Valor board
- Execute hero and monster combat actions (attacks and spell casts)
- Apply damage, dodges, and spell debuffs using game rules
- Update per-hero statistics and emit combat log output
- **valor.game - ValorGame** Acts as the entry point for the Legends of Valor game mode.
- Starts and runs a Valor match session
- Handles quit and replay flow for the game mode via askPlayAgain
- Delegates post-game processing to the post-game controller via ValorPostGameController
- Integrates persistence(SaveManager) and leaderboard services (LeaderboardService) for end-of-game actions
- **valor.game - ValorIntroScreen** Displays the introductory instructions and rules for Legends of Valor.
- **valor.game - ValorLaneSelector** Collects user input to assign each of the three heroes to a Valor lane.
- **valor.game - ValorLaneState** Implements the interactive lane gameplay state for Legends of Valor.
- **valor.game - ValorMarketController** Controls market interactions during Legends of Valor gameplay.
- **valor.game - ValorMatch** Orchestrates a full Legends of Valor match from setup through repeated rounds.
- **valor.game - ValorMatchSetup** Builds and initializes all components required to start a Legends of Valor match.
- **valor.game - ValorMonsterAI** Determines movement decisions for monsters during Legends of Valor gameplay.
- **valor.game - ValorPostGameController** Manages end-of-match flow for Legends of Valor after a match concludes.
- **valor.game - ValorSpawner** Handles initial placement and spawning of heroes and monsters on the Valor board.

- **valor.game - ValorState** Defines a common interface for Legends of Valor gameplay states.
- **valor.turn - ConsoleValorInput** Provides console-based input handling for Legends of Valor turn actions.
- **valor.turn - HeroActionService** Provides a unified facade for hero actions during Legends of Valor turns.
- **valor.turn - HeroCombatActions** Encapsulates all hero combat-related actions in Legends of Valor.
- **valor.turn - HeroEquipmentActions** Encapsulates equipment, inventory usage, and market access actions for Valor heroes.
- **valor.turn - HeroMovementActions** Encapsulates hero movement and board-rule actions for Legends of Valor turns.
- **valor.turn - HeroTurnController** Controls the interactive turn flow for a single hero in Legends of Valor.
- **valor.turn - HeroTurnMenuView** Renders the hero turn menu UI for Legends of Valor in the console.
- **valor.turn - HeroTurnUIHelper** Provides reusable console-selection prompts for Legends of Valor hero turn actions.
- **valor.turn - MonsterTurnController** Controls the monster phase of a Legends of Valor round.
- **valor.turn - ValorInput** Abstraction over player input for Legends of Valor to decouple game logic from direct console I/O.
- **valor.turn - ValorTurnManager** Coordinates turn execution for a Legends of Valor round.
- **valor.ui - ValorCombatLogView** Renders formatted combat log messages for Legends of Valor encounters.
- **valor.ui - ValorEndScreenRendering** Renders the Legends of Valor post-game end screen and loaded-save summaries.
- **valor.ui - ValorRoundStatusView** Displays a per-round status snapshot of the Valor board.
- **valor.world.terrain package** Creates terain instances via TerrainFactory for all the terrain-enabled board cells (terrain.BushTerrain, terrain.CaveTerrain, terrain.KoulouTerrain)

- **valor.world - ValorBoard** Represents the 8x8 game board for Legends of Valor.
- Handles board generation, lane structure, terrain placement,hero/monster positioning, movement validation, rendering,and win-condition checks.
- **valor.world - ValorCellType** Enumerates all possible cell types (nexus, plain, inaccessible etc) on the Legends of Valor board. Each type defines its display symbol and basic accessibility rules.
- **valor.world - ValorDirection** Represents the four cardinal movement directions used on the board. Each direction defines how it changes a unit's row and column position.
- **valor.world - ValorMovement** Applies movement rules for heroes and monsters on a ValorBoard. Handles standard movement, teleport placement checks, terrain enter/exit hooks, and lane-based "no bypass" constraints.
- **valor.world - ValorTile** Represents a single cell on the board. A tile has a fixed cell type, optional terrain behavior, and may contain at most one Hero and one Monster.

**legends.characters** ***(only high-level summary here)***

- **Hero (abstract)**
  Base class for all hero types (Warrior, Paladin, Sorcerer). Holds common stats (level, HP, MP, STR, DEX, AGI, gold, inventory, etc.) and combat helpers (attack damage, dodge chance, usePotion, equipWeapon/Armor, etc.).
- **Warrior, Paladin, Sorcerer**
  Concrete hero types with different stat emphases (melee, balanced, magic).
- **Monster**
  Represents an enemy; has HP, damage, defense, dodge chance, level, etc.
- **Party**
  Contains the list of heroes and current Position.
    - Movement (moveTo)
    - Utility like getAliveHeroes(), getRandomAliveHero(), allDead(), printStats().
- **Inventory**
  Holds a hero's items and provides getItems(), addItem, removeItem, and printFormatted(int width).

**legends.items**

- **Item (interface)**
  Common contract for all items:

    - String getName()
    - int getPrice()
    - int getRequiredLevel()

- **Weapon**
  Adds damage, handsRequired and getHands().
  toString() prints level, damage and hands.
- **Armor**
  Adds reduction (damage reduction).
- **Potion**
  Adds effectAmount and List<PotionAttribute> describing which stats it affects.
- **Spell**
  Adds damage, manaCost, and SpellType (FIRE, ICE, LIGHTNING).
  BattleState applies type-specific debuffs (defense, damage, dodge).
- **PotionAttribute (enum)**
  HEALTH, MANA, STRENGTH, DEXTERITY, AGILITY.
- **SpellType (enum)**
  FIRE, ICE, LIGHTNING.

**legends.market**

- **Market**
  Holds category-separated lists: List<Weapon>, List<Armor>, List<Potion>, List<Spell>.
    - Constructor 1: takes separate lists.
    - Constructor 2 (preferred): takes List<Item> and sorts into type-specific lists.

**legends.world**

- **Tile (abstract)**
  Base tile; defines:
    - boolean isAccessible()
    - String getSymbol()
- **CommonTile**
  Accessible, shown as . on the map.
- **InaccessibleTile**
  Not accessible, shown as X.

- **MarketTile**
  Accessible market tile, shown as M.
- **Position**
  Simple row/col pair for the party's location.
- **WorldMap**
  2D grid of Tiles with helper methods:
  - setTile, getTile, inBounds, canMove.
  - print(Position partyPos) draws a colored grid with borders and party marker P.
- **MapGenerator**
  Randomly generates a WorldMap with configurable size:
  - ~20% InaccessibleTile, ~15% MarketTile, remaining CommonTile.
  - Can be easily tweaked to adjust map density and difficulty.

**legends.ui**

- **BarUtils**
  Builds colored HP/MP bars given current, max, and bar length.
- **Colors**
  Centralized ANSI color constants (foregrounds, some backgrounds, bold, underline).
- **ConsoleUI** Centralized ANSI color codes and text styling with utilities for boxed sections, aligned tables, ANSI-safe padding/centering, and consistent console output formatting.

**legends.data *(high-level)***

- **DataLoader**
  Reads text files for heroes, items, and monsters and constructs Java objects at startup. Also exposes collections like loadWarriors(), loadPaladins(), etc.
- **MonsterFactory**
  Utility for generating a set of monsters suitable for the current party (e.g., based on level and size).

## Brief Evaluation

After completing the project, we believe our design decisions largely achieved the intended goals of scalability and extensibility. Separating turn coordination, action handling, and core game logic helped keep classes focused and readable, and made it easier to debug and extend features during development. While some components could be further generalized with more time, the overall structure met our expectations and allowed the system to evolve without major refactoring.