RiftWalker

Software Engineering Documentation

Michael Krumpe, Mitchell Myers, Blake Peterman

(12/15/2025)

Table of Contents

1. Purpose

The video game industry is currently witnessing a renaissance for independent developers. In 2024 alone, indie games captured nearly 48% of all full-game revenue on Steam, generating approximately $4 billion. Titles such as *Balatro*, which sold over 3.5 million units within its first year, and the critically acclaimed *Animal Well* have proven that unique atmosphere and solid mechanics can rival the commercial success of AAA productions. This shift demonstrates that a focused, novel experience created by a small team can resonate deeply with a global audience, provided the core gameplay loop is engaging and polished.

The democratization of powerful development tools fuels this surge in indie success. We have seen this firsthand with the Godot engine, which powered 37% of entries in the 2024 GMTK Game Jam—a massive leap from 19% just the year prior. These accessible, open-source tools act as a force multiplier for groups like ours, allowing us to tackle complex programming challenges and fully flesh out intricate systems that would have previously required a much larger workforce.

As the indie scene has evolved, specific genres have risen to dominance. The roguelike market, valued at $2.1 billion in 2024, accounts for over 20% of all action games released on Steam. This genre's emphasis on high replayability through "permadeath" and procedural generation contrasts sharply with the traditional RPG, which prioritizes long-term narrative investment and permanent character growth. While both genres drive player engagement, they typically do so through opposing mechanics: one through constant resetting, and the other through cumulative permanence.

Our group has decided to embrace the challenge of reconciling these two distinct design philosophies. Drawing inspiration from the thriving $2.1 billion roguelike market and the narrative depth of classic RPGs, we aim to create *Riftwalker*. This unique experience leverages the replayability of procedural runs while maintaining a satisfying sense of long-term progression and community interaction.

2. Scope

We named our project "Riftwalker" to reflect the core narrative of a squad of bounty hunters trapped in a temporal anomaly known as a "Rift." This storyline serves as the foundation for our game's roguelike structure, in which players must navigate procedurally generated realms while managing a party of three distinct allies. Our goal is to create a mechanically rich, turn-based strategy RPG using Godot 4 that seamlessly integrates with a custom ASP.NET Core companion website.

We designed the experience to bridge the gap between local and online play. Players can enjoy the full depth of our tactical combat and character progression—defined by Body, Mind, and Spirit attributes—completely offline. However, when connected, our system transforms into a cloud-integrated ecosystem. Game saves are automatically synchronized to the cloud to ensure data safety, and run statistics are uploaded to a global leaderboard. By connecting the game to our website, we aim to foster a competitive and collaborative community where players can not only safeguard their progress but also share their achievements and analyze the strategies of top-ranking “Riftwalkers”.

3. User characteristics

3.1 Key users

User Role Responsibilities

Our Key Users are active participants in the complete Riftwalker ecosystem. Their responsibilities extend beyond simply playing the game; they are expected to synchronize their local progress with the online platform, actively participate in community discourse, and monitor the leaderboard. Crucially, this group serves as the primary feedback loop for development, reporting bugs, identifying balance issues, and suggesting feature improvements for both the client and the website.

Subject Matter Experience

Target users have intermediate to advanced proficiency in the Roguelike and RPG genres. They are familiar with core mechanics such as turn-based combat, attribute optimization (min-maxing), and procedural risk management. We anticipate these users will have played similar titles (e.g., Slay the Spire, Darkest Dungeon) and can quickly grasp Riftwalker's specific systems without extensive tutorials. Furthermore, they are experienced in online community etiquette and can drive constructive discussions.

Technological Experience

Key users exhibit high digital literacy. They are comfortable installing non-standard desktop applications, managing local file structures (for saves/logs), and navigating web interfaces. They possess the technical competence to distinguish between intended game mechanics and software anomalies (bugs), and can effectively communicate these findings to the development team.

Other Characteristics

We are targeting enthusiastic early adopters aged 16–35 who are deeply invested in strategy games and interactive storytelling. This demographic values community engagement and has a positive, constructive attitude toward technology and online forums. They are not just consumers, but active contributors who enjoy dissecting game mechanics and sharing their discoveries with a like-minded community.

3.2 Secondary users

User Role Responsibilities

Secondary Users represent the casual player base. Their engagement is primarily focused on the core gameplay loop—combat and exploration—rather than the broader online ecosystem. Unlike Key Users, they typically do not utilize the website features, participate in forum innovations, or actively monitor leaderboards. Their data contribution is passive (automated run uploads) rather than active community participation.

Subject Matter Experience

This group possesses a foundational-to-intermediate understanding of the Roguelike and RPG genres. While they understand the basic turn-based format and progression systems, they may lack deep knowledge of optimal build strategies ("min-maxing"). They are generally consumers of the experience provided, rather than contributors to the "meta" discussion or community strategy guides.

Technological Experience

Secondary Users demonstrate standard consumer-level digital literacy. They are proficient in launching applications and basic internet browsing. Still, they are not expected to troubleshoot technical issues, modify local files, or produce community content (e.g., video guides, detailed bug reports). Their interaction with the technology is functional and utility-driven rather than exploratory.

Other Characteristics

This demographic spans a wide age range (10–50) and is characterized by a neutral or strictly pragmatic attitude toward technology. Their engagement is often constrained by time availability or a preference for low-friction entertainment. They play Riftwalker for immediate enjoyment and are unlikely to engage with systems that require deeper investment, such as competitive ranking or community forum management.

3.3 Unimportant users

User Role Responsibilities

Unimportant Users represent the non-target audience or incidental user base. Their interaction with Riftwalker is expected to be minimal, often resulting from accidental acquisition or bulk purchasing (e.g., bundles/gifts) rather than genuine interest. This group typically churns rapidly, does not engage with online features, and provides no meaningful data or community value beyond the initial transaction.

Subject Matter Experience

This group lacks relevant gaming literacy, particularly within the Roguelike and Strategy RPG genres. They are likely to find the core systems frustrating or opaque due to a fundamental misalignment with the game's intended complexity. They possess no prior experience with similar titles and have no motivation to participate in the learning curve or online discourse.

Technological Experience

Unimportant Users typically exhibit limited technological proficiency restricted to specific contexts (e.g., mobile-only usage) and are unfamiliar with PC gaming ecosystems. They are unlikely to navigate forums, troubleshoot fundamental installation issues, or understand the standard conventions of desktop software interactions.

Other Characteristics

This demographic is characterized by a general indifference or aversion to the specific technology required to play Riftwalker. It includes users outside the target age range (e.g., very young children or older adults with incompatible interests) and those who simply prefer entirely different forms of entertainment. Their engagement is often negligible, as the product fundamentally does not meet their needs or expectations.

4. Assumptions and Dependencies

4.1 Operating System

Development and Quality Assurance were conducted exclusively on Windows 10 and 11 environments. While the Godot engine supports cross-platform exports, all file system interactions (specifically regarding user:// save paths) and input mappings have been verified primarily for the Windows ecosystem. Extending support to Linux, macOS, or Mobile platforms would require a dedicated validation phase to ensure that directory handling and input device compatibility perform as expected.

4.2 Godot Engine Version

The game client was architected using Godot 4.x, leveraging the updated features of GDScript 2.0. Contrary to initial plans, the logic was implemented entirely in GDScript without requiring C++ extensions, simplifying the build process. The project is strictly bound to the Godot 4 runtime. Backporting to Godot 3.x is not possible due to fundamental changes in the engine's API and scripting language.

4.3 Hardware Performance

The application was optimized for standard desktop configurations featuring modern integrated or mid-range dedicated GPUs. Performance profiling confirmed that the game consistently maintains the target 60 FPS on these systems. While lightweight, the use of Godot 4's newer rendering backend means that extremely low-spec hardware or unoptimized mobile devices may struggle to maintain the target frame rate without further optimization.

4.4 Web Server & Hosting Environment

The "Online Mode" relies on a custom ASP.NET Core Web API backend backed by a SQLite Server database. The client’s networking stack is designed to communicate with specific REST endpoints for authentication and data synchronization. The game's online features are entirely dependent on the availability of this particular web infrastructure. The system gracefully falls back to "Offline Mode" if the server is unreachable.

4.5 Data Privacy & Security

Security best practices were integrated into the core architecture. This includes implementing HTTPS for transport security, device-based authentication to minimize user data collection, and SHA-256 integrity hashing to validate the submission of run data. Any future shift towards a complete username/password account system would require a significant overhaul of the current lightweight AuthManager to handle credential storage and recovery securely.

4.6 Third-Party Libraries/Assets

The project successfully maintained compliance with open-source and educational licenses for all utilized assets, including fonts, sound effects, and plugins. Usage was tracked to ensure no proprietary violations occurred. The current asset pipeline is clear for educational or non-commercial distribution. Commercialization would require a re-audit of specific "non-commercial only" assets.

4.7 Project Schedule & Team Skillset

The team successfully surmounted the learning curve for the required technologies. Competency in GDScript for client-side logic and C#/SQL for backend management was achieved within the development timeframe, enabling delivery of the core feature set. Features that required skills outside this core competency (e.g., advanced peer-to-peer multiplayer or complex 3D shader work) were descoped in favor of a polished single-player experience with asynchronous online elements.

5. Requirements Specification

R.1 Game Core Mechanics

R.1.1 Ally Statistics & Attributes

- **R.1.1** Player-controlled characters (Allies) shall have the following statistics: Body, Mind, Spirit, Health (Max HP), Magic Points (Max MP), Strength, Defense, Magic Strength, and Speed.
    - **R.1.1.1** The Body, Mind, and Spirit statistics shall be assigned ally-specific base values at the start of each run.
    - **R.1.1.2** A player shall have the ability to increase these statistics at designated upgrade points (campfires).
    - **R.1.1.3** All statistics must be derived dynamically every time they are accessed.
        - **R.1.1.3.1** The System shall calculate *Health* (Max HP) using the formula *(Body + Spirit) * 20*.
        - **R.1.1.3.2** The System shall calculate *Speed* using the formula *(Body + Mind) * 5*.
        - **R.1.1.3.3** The System shall calculate *Magic Points* (Max MP) using the formula *(Mind + Spirit) * 5*.
        - **R.1.1.3.4** The System shall calculate *Strength* using the formula *(Body + Mind) * 2*.
        - **R.1.1.3.5** The System shall calculate *Magic Strength* using the formula *(Mind + Spirit) * 2*.
        - **R.1.1.3.6** The System shall calculate *Defense* using the formula *(Body + Spirit) * 2*.
    - **R.1.1.4** The *AllyBattler* shall persist *damage_taken* between battles to maintain state continuity.
    - **R.1.1.5** The *AllyBattler* shall persist *mana_used* between battles to maintain state continuity.
    - **R.1.1.6** The *AllyBattler* shall clamp *health* values between 0 and *Max Health*.
    - **R.1.1.7** The *AllyBattler* shall clamp *magicPoints* values between 0 and *Max MP*.

R.1.2 Turn Management

- **R.1.2** The Turn System shall manage the flow of combat.
    - **R.1.2.1** The *TurnManager* shall strictly enforce a sequential flow of action where only one Battler acts at a given time.
    - **R.1.2.2** The *TurnManager* shall determine execution priority by sorting all active Battlers.
        - **R.1.2.2.1** Sorting shall be based on the *Speed* stat in descending order.
        - **R.1.2.2.2** This sort shall occur at the beginning of every turn cycle.
        - **R.1.2.2.3** If the *Speed* stat is identical, the stable sort order shall remain consistent.
    - **R.1.2.3** The *TurnManager* shall manage input blocking states.
        - **R.1.2.3.1** Input shall be allowed during the *Decision Phase* for player characters.
        - **R.1.2.3.2** Aside from skipping, input shall be blocked during the *Execution Phase* while animations or AI logic are resolving.
    - **R.1.2.4** The *TurnManager* shall skip the turn of any battler flagged as *isDefeated*.
    - **R.1.2.5** The *BattleScene* shall monitor global victory/defeat conditions.
        - **R.1.2.5.1** A "Victory" condition shall be detected immediately if the count of active *EnemyBattlers* reaches zero.
        - **R.1.2.5.2** A "Defeat" condition shall be detected immediately if the count of active *AllyBattlers* reaches zero.

R.1.3 Damage & Status Mechanics

- **R.1.3** The System shall calculate and apply damage and effects.
    - **R.1.3.1** Physical Damage shall be calculated using the formula *(BaseDamage + Strength) - TargetDefense*.
    - **R.1.3.2** Magic Damage shall be calculated using the formula *(BaseDamage + MagicStrength) - TargetDefense*.
    - **R.1.3.3** The System shall ensure final damage is never less than 0 (clamped).

- **R.1.3.4** Critical Hit Logic:
  - **R.1.3.4.1** The System shall roll for a Critical Hit on every offensive action with a 10% probability.
  - **R.1.3.4.2** If a Critical Hit occurs, the final damage value shall be multiplied by 2.
- **R.1.3.5** Status Effect Logic:
  - **R.1.3.5.1** The *EnemyBattler* shall automatically skip its turn if it is under a *DisablingStatusEffect* (e.g., Sleep/Stun).
  - **R.1.3.5.2** The duration of any active *DisablingStatusEffect* shall be decremented by one at the start of the affected battler's turn.
  - **R.1.3.5.3** The *EnemyBattler* shall become immune to a specific *DisablingStatusEffect* for two turns immediately after that effect wears off.

R.1.4 Enemy AI Behavior

- **R.1.4** The AI shall determine enemy actions without human intervention.
  - **R.1.4.1** The *EnemyBattler* shall select an action from its *actions* list.
    - **R.1.4.1.1** Selection shall use a weighted random algorithm based on the *enemyActionChance* property.
    - **R.1.4.1.2** If weighted selection fails, the AI shall default to the first available action.
  - **R.1.4.2** The Targeting System shall select valid targets based on *ActionTargetType*.
    - **R.1.4.2.1** If the type is *SINGLE_ALLY*, the AI shall randomly select one living target from the *allies* group.
    - **R.1.4.2.2** If the type is *ALL_ALLIES*, the AI shall select every living member of the *allies* group.

R.2 World & Meta-Progression

R.2.1 Map Generation

- **R.2.1** The *MapGenerator* shall create a procedural level layout.
    - **R.2.1.1** Structure Constraints:
        - **R.2.1.1.1** Every map shall consist of exactly 10 horizontal layers (Layer 0 to Layer 9).
        - **R.2.1.1.2** Layer 0 shall consist exclusively of *BATTLE* nodes.
        - **R.2.1.1.3** Layer 9 shall consist of exactly one *BOSS* node.
        - **R.2.1.1.4** Layers 1 through 8 shall contain a randomized count of nodes (between 3 and 4 per layer).
    - **R.2.1.2** Node Type Distribution:
        - **R.2.1.2.1** There shall be a 20% probability for any eligible node to be an *ELITE* encounter.
        - **R.2.1.2.2** Exactly one *SHOP* node shall be injected at a random position between Layers 1 and 8.
        - **R.2.1.2.3** Exactly one *REST* node shall be injected at a random position between Layers 1 and 8.
    - **R.2.1.3** Connectivity Rules:
        - **R.2.1.3.1** Every node in Layer *N* shall connect to at least one node in Layer *N+1*.
        - **R.2.1.3.2** Connections shall only be formed if the vertical grid index difference is <= 1 (Lateral Distance).

R.2.2 Navigation Logic

- **R.2.2** The *MapScreen* shall manage player movement through the generated graph.
    - **R.2.2.1** Interaction Locks:
        - **R.2.2.1.1** The System shall lock interaction with any node that is not present in the *next_nodes* list of the player's current occupied node.
        - **R.2.2.1.2** The System shall visually distinguish locked nodes (e.g., Greyed out) vs Available nodes.
    - **R.2.2.2** New Game+ (Rift) Logic:

- **R.2.2.2.1** A "RIFT" button shall appear only after the Layer 9 Boss has been marked as *is_visited*.
- **R.2.2.2.2** Clicking the RIFT button shall increase *Global.map_base_difficulty* by 10.
- **R.2.2.2.3** Clicking the RIFT button shall trigger the generation of a completely new map.

R.2.3 Economy & Loot

- **R.2.3** The *LootManager* shall calculate rewards.
  - **R.2.3.1** Coin Calculation Formula: *(Health*0.1 + Strength*0.2 + Magic*0.2) * DifficultyMultiplier*.
  - **R.2.3.2** Minimum Reward: The System shall enforce a minimum reward of 10 coins per battle.
  - **R.2.3.3** Difficulty Scaling:
    - **R.2.3.3.1** The *DifficultyMultiplier* shall be calculated using *1.0 + ((RemoteRound - 1)^2 * 0.05)*.
    - **R.2.3.3.2** *RemoteRound* is defined as *Global.current_round + Global.battle_round_offset*.

R.2.4 Shop & Inventory

- **R.2.4** The *ShopMenu* shall facilitate item transactions.
  - **R.2.4.1** Purchase Validation:
    - **R.2.4.1.1** The System shall prevent item purchase if *Global.coins* is less than *Item.price*.
    - **R.2.4.1.2** The System shall play an error sound (*Audio.denied*) upon failed validation.
  - **R.2.4.2** Transaction Execution:
    - **R.2.4.2.1** *Item.price* shall be deducted from *Global.coins* immediately.
    - **R.2.4.2.2** The *Item* resource shall be appended to the selected Ally's inventory array.

- **R.2.4.2.3** The Game State shall be saved immediately.

R.3 User Interface & Feedback

R.3.1 Menus & Navigation

- **R.3.1** The UI shall provide navigation and configuration options.
  - **R.3.1.1** Title Screen:
    - **R.3.1.1.1** The "Start Game" button shall display "Continue" if *Global.run_in_progress* is true.
    - **R.3.1.1.2** The "Start Game" button shall display "Start Game" if *Global.run_in_progress* is false.
  - **R.3.1.2** Settings Menu:
    - **R.3.1.2.1** A CheckBox shall allow the user to toggle "Fullscreen" mode.
    - **R.3.1.2.2** A Slider shall allow the user to adjust Master Volume (Linear to Decibel mapping).
    - **R.3.1.2.3** A Slider shall allow the user to adjust *Game Speed*, updating *Global.game_speed*.
  - **R.3.1.3** Leaderboard Menu:
    - **R.3.1.3.1** The Menu shall display a "Loading..." indicator while fetching data.

R.3.2 Visual & Audio Feedback

- **R.3.2** The Application shall provide immediate feedback for actions.
  - **R.3.2.1** Damage Numbers:
    - **R.3.2.1.1** The System shall instantiate a *DamageNumber* scene at the unit's position when damage is taken.
    - **R.3.2.1.2** The Damage Number text shall match the integer amount of damage dealt.
  - **R.3.2.2** Audio Cues:

        - **R.3.2.2.1** *Audio.btn_mov* shall play when the mouse hovers over any interactive button.
        - **R.3.2.2.2** *Audio.purchase* shall play when an item is bought.
        - **R.3.2.2.3** *Audio.store_bell* shall play when entering the Shop scene.

R.4 Data Persistence

R.4.1 Serialization Schema

- **R.4.1** The *Global* system shall serialize game state into a JSON-compatible Dictionary.
    - **R.4.1.1** Root Level Data:
        - **R.4.1.1.1** The Schema shall include *coins* (int).
        - **R.4.1.1.2** The Schema shall include *current_round* and *highest_round* (int).
        - **R.4.1.1.3** The Schema shall include *current_username* (string).
        - **R.4.1.1.4** The Schema shall include *run_in_progress* (bool).
    - **R.4.1.2** Deep Ally Data (*allies_data*):
        - **R.4.1.2.1** The Schema shall store *body*, *mind*, and *spirit* per ally.
        - **R.4.1.2.2** The Schema shall store *damage_taken* and *mana_used* per ally.
        - **R.4.1.2.3** The Schema shall store *items* as a list of Resource Paths (strings).

R.4.2 Local Storage Triggers

- **R.4.2** The Application shall save to *user://savegame.save* at critical moments to prevent data loss.
    - **R.4.2.1** Save shall occur immediately after any Battle Victory.
    - **R.4.2.2** Save shall occur immediately after any Shop Transaction.
    - **R.4.2.3** Save shall occur immediately after any Upgrade Confirmation.
    - **R.4.2.4** Save shall occur immediately upon generating a new Map.

R.5 Networking & Server

R.5.1 Authentication Logic

- **R.5.1** The *AuthManager* shall handle user identification.
    - **R.5.1.1** Device ID Generation:
        - **R.5.1.1.1** The Manager shall generate a persistent unique *DeviceId* using *OS.get_unique_id()* if one is missing.
        - **R.5.1.1.2** This ID shall be stored locally in the save file.
    - **R.5.1.2** Auto-Login:
        - **R.5.1.2.1** The Client shall attempt to log in via *POST /api/login* automatically on app launch.
        - **R.5.1.2.2** The Client shall send *DeviceId* and *current_username* in the body.
        - **R.5.1.2.3** Upon 200 OK, the Client shall store the returned *access_token* and *user_id*.

R.5.2 Cloud Synchronization

- **R.5.2** The Client shall sync Save Data with the Server.
    - **R.5.2.1** Download:
        - **R.5.2.1.1** The Client shall trigger *GET /api/save/{userId}* immediately upon successful login.
        - **R.5.2.1.2** The Client shall parse the returned binary blob as the new Game State.
    - **R.5.2.2** Upload:
        - **R.5.2.2.1** The Client shall trigger *POST /api/save/upload* immediately after any local save operation.
        - **R.5.2.2.2** The Request shall use the *multipart/form-data* content type.

R.5.3 Server API Contracts

- **R.5.3** The Server shall provide RESTful endpoints.

- **R.5.3.1** Login Endpoint (*POST /api/login*):
  - **R.5.3.1.1** The Server shall create a new *AccountModel* record if the *DeviceId* is unknown.
  - **R.5.3.1.2** The Server shall update the *Username* if *DeviceId* matches, but the name differs.
  - **R.5.3.1.3** The Server shall return JSON containing *user_id*, *username*, and *access_token*.
- **R.5.3.2** Run Upload Endpoint (*POST /api/upload-run*):
  - **R.5.3.2.1** The Server shall validate the *X-Integrity-Hash* header.
  - **R.5.3.2.2** The Integrity Hash shall be a SHA256 signature of *SECRET_SALT + Round + Coins*.
  - **R.5.3.2.3** The Server shall return *400 Bad Request* if validation fails.
  - **R.5.3.2.4** The Server shall map *total_coins* to *RunModel.Score*.
  - **R.5.3.2.5** The Server shall map *highest_round* to *RunModel.Status*.
- **R.5.3.3** Leaderboard Endpoint (*GET /api/leaderboard*):
  - **R.5.3.3.1** The Server shall return the top 10 runs.
  - **R.5.3.3.2** Runs shall be grouped by User (showing only the user's best run).
  - **R.5.3.3.3** Sorting shall prioritize *Highest Round* (Status) Descending.
  - **R.5.3.3.4** Sorting shall use *Total Coins* (Score) as a secondary tie-breaker.
- **R.5.3.4** Cloud Storage Endpoint (*POST /api/save/upload*):
  - **R.5.3.4.1** The Server shall accept *multipart/form-data* input containing the binary save file.
  - **R.5.3.4.2** The Server shall upsert the binary data into the *GameSaves* table associated with the user.

R.6 Non-Functional Requirements

R.6.1 Performance

- **R.6.1** The System shall meet performance standards.
    - **R.6.1.1** The Game Client should maintain a target frame rate of 60 FPS on standard desktop hardware.
    - **R.6.1.2** The Game Client should load the *BattleScene* from the *MapScreen* in under 1 second.
    - **R.6.1.3** The Server should process Login requests in under 500ms under normal load.

R.6.2 Reliability & Fault Tolerance

- **R.6.2** The System shall remain robust during failure states.
    - **R.6.2.1** The Game Client shall not crash if the Server API is unreachable.
    - **R.6.2.2** The Game Client shall fall back to "Offline Mode" upon connection failure.
    - **R.6.2.3** The Game Client shall retry failed cloud uploads silently (Best Effort).
    - **R.6.2.4** The Server shall handle database connection failures gracefully by returning *500 Internal Server Error*.

R.6.3 Security

- **R.6.3** The System shall protect data integrity and privacy.
    - **R.6.3.1** The System shall not expose raw Database Connection Strings in the client-side code.
    - **R.6.3.2** The Server shall sanitize all string inputs to prevent SQL Injection.
    - **R.6.3.3** The Server shall limit the number of API requests allowed from a single DeviceId to 60 requests per minute to prevent Denial of Service (DoS) attacks.
    - **R.6.3.4** The Server shall validate the "Magic Numbers" (file signature) of the uploaded save file to ensure it is a valid Godot resource file before storing it, rejecting any executables or scripts.
    - **R.6.3.5** The Server shall log all failed X-Integrity-Hash validation attempts, including the DeviceId, Timestamp, and IP Address, to a separate SecurityAudit table for administrative review.

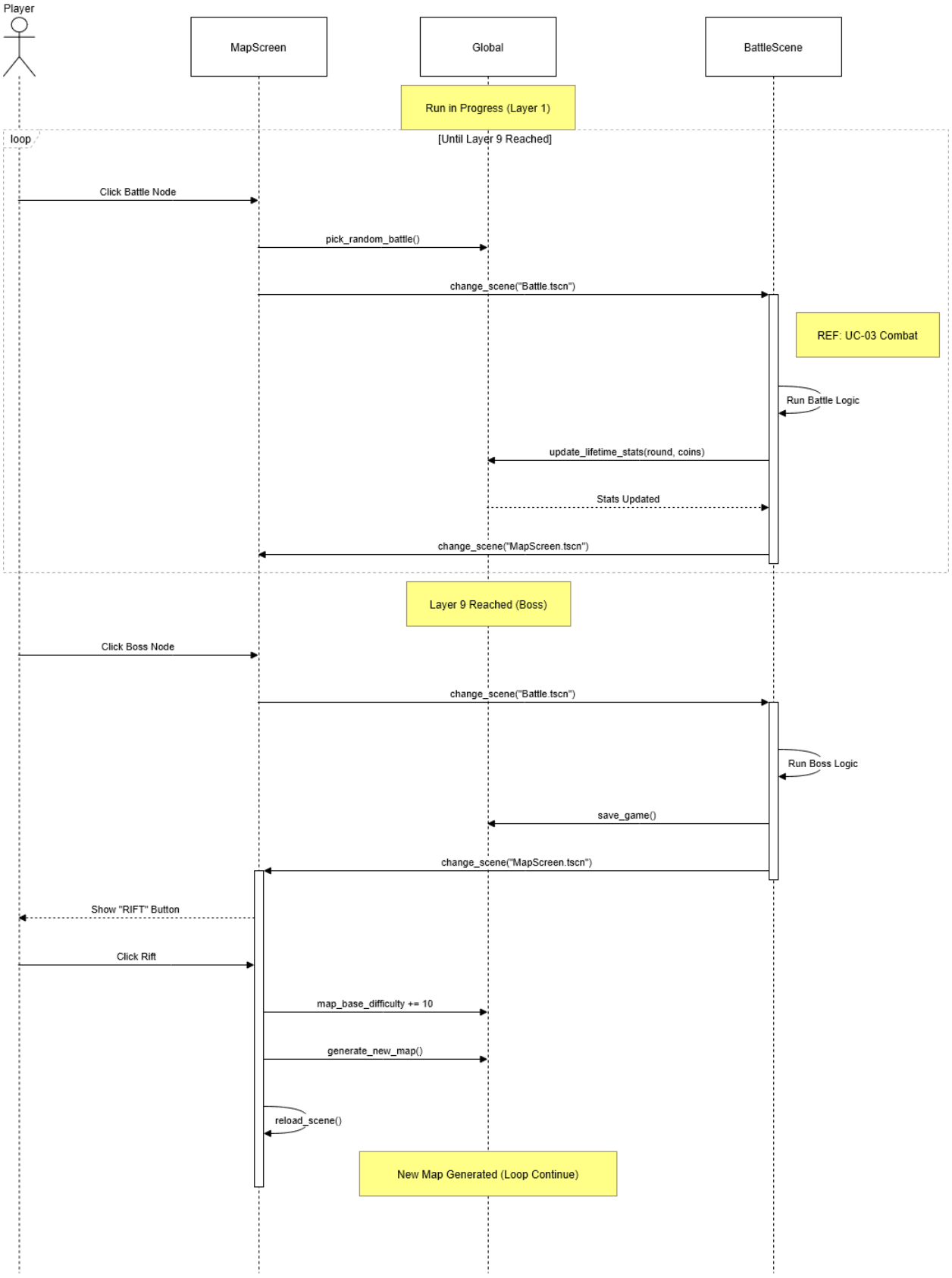
- **R.6.3.6** The Server shall reject any RunModel payload where Coins or Round values exceed the theoretical maximum integers possible within a standard gameplay session (e.g., Coins < 0 or Coins > 1,000,000).
- **R.6.3.7** Replay Prevention: The System shall verify that the DeviceId has not already submitted a specific RoundID or RunTimestamp to prevent "Replay Attacks" of valid scores.

6. Use Cases

6.1 Core Gameplay Loop (Offline/Client-Side)

Use Case ID	UC-01
Use Case Name	Run Gameplay Loop
User Goal	To successfully navigate through a generated dungeon map, defeat enemies in turn-based combat, and upgrade characters to delay defeat for as long as possible.
Scope	Riftwalker Game Client (Local)
Level	Primary Task
Primary Actor	Player
Preconditions	1. The Game Client is launched. 2. The Player is at the Main Menu. 3. Global.run_in_progress is initialized.
Minimal Guarantee	The game state is saved locally after every battle or transaction, preventing loss of progress in case of a crash.
Success Guarantee	The player completes all 10 layers (including the Boss) and is allowed to “rift” to the following map.
Trigger	Player clicks the "Start Game" or "Continue" button on the Main Menu.
Success Scenario	1. Player enters the Map Screen. 2. Player clicks on a BATTLE node. 3. The system loads the BattleScene. 4. Player selects "Attack" for their Ally. 5. Enemy is defeated (isDefeated = true). 6. System awards Coins and XP; 7. Player returns to the Map Screen 8. Steps 2-7 repeat until Layer 9 is reached. 9. Player defeats the Boss. 10. System displays "Rift” button. 11. Player clicks the button. 12. Player is transported to the following map.
Extensions	3a. Player Defeat: 1. All Allies drop to 0 HP. 2. System triggers "Defeat" state.

	3. System deletes the active run (run_in_progress = false). 4. The system returns the player to the Main Menu. 5a. Game Crash: 1. App terminates unexpectedly. 2. (On Restart) Player clicks "Continue". 3. The system loads the state from the last completed node (Minimal Guarantee).

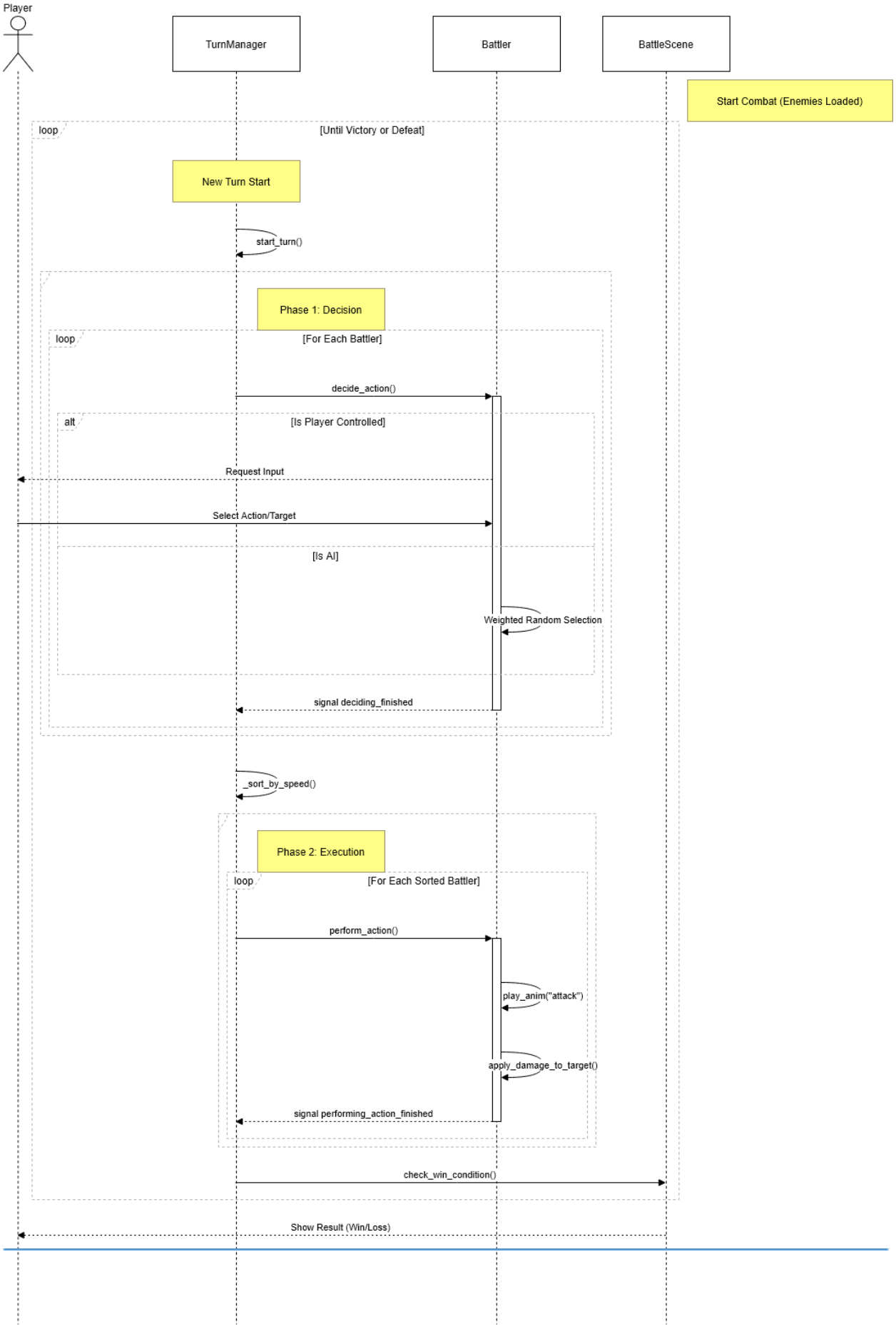

Player
MapScreen
Global
BattleScene
Run in Progress (Layer 1)
loop
[Until Layer 9 Reached]
Click Battle Node
pick_random_battle()
change_scene("Battle.tscn")
REF: UC-03 Combat
Run Battle Logic
update_lifetime_stats(round, coins)
Stats Updated
change_scene("MapScreen.tscn")
Layer 9 Reached (Boss)
Click Boss Node
change_scene("Battle.tscn")
Run Boss Logic
save_game()
change_scene("MapScreen.tscn")
Show "RIFT" Button
Click Rift
map_base_difficulty += 10
generate_new_map()
reload_scene()
New Map Generated (Loop Continue)

6.2 Progression & Persistence (Online/Server-Side)

Use Case ID	UC-02
Use Case Name	Online Persistence & Sync
User Goal	To automatically back up game progress to the cloud and publish high scores to the global leaderboard.
Scope	Game Client & Web Server (System-Wide)
Level	Sub-function Task
Primary Actor	Registered User (Authenticated Player)
Preconditions	1. The Game Client has active Internet connectivity. 2. The Web Server is online. 3. The Client has a valid DeviceId.
Minimal Guarantee	If the network fails, the game continues in "Offline Mode" and queues the data for later sync (or saves it locally only).
Success Guarantee	The local save file is bit-for-bit matched to the server copy, and the player's run score is visible on the public leaderboard.
Trigger	Game Application Launch (Auto-Login) OR User completes a Run (Score Upload).
Success Scenario	1. Player opens the game. 2. AuthManager sends POST /api/login with DeviceId. 3. Server returns access_token and user_id. 4. Client requests GET /api/save/{id}. 5. Client overwrites local savegame.save with cloud data. 6. Player completes a run (See UC-01). 7. Client sends POST /api/upload-run with X-Integrity-Hash. 8. Server validates hash and updates RunModel. 9. Browser displays the new score on the Leaderboard.
Extensions	2a. New User: 1. Server sees an unknown DeviceId. 2. Server creates a new Account. 3. Server returns new user_id. 4a. Connection Timeout: 1. Client receives 408 Request Timeout or Connection Error. 2. System logs error "Offline Mode Active". 3. Game continues using local save data. 7a. Anti-Cheat Failure: 1. Client sends an invalid Hash. 2. Server returns 400 Bad Request. 3. Client logs "Score Rejected" (Score is not posted).

6.3 Turn-Based Combat

Use Case ID	UC-03
Use Case Name	Execute Combat Encounter
User Goal	To strategically select actions for party members to defeat all enemy units without suffering a total party wipe.
Scope	BattleScene (Local)
Level	Primary Task
Primary Actor	Player
Preconditions	1. BattleScene is loaded. 2. BattleData (Enemies/Allies) is initialized. 3. TurnManager is active.
Minimal Guarantee	The state of all units (HP/MP) is preserved for the next turn; the game does not hang if an animation fails.
Success Guarantee	All EnemyBattler units are marked isDefeated, and the "Victory" sequence triggers.
Trigger	Entering a BATTLE or BOSS node triggers the Scene load.
Success Scenario	1. TurnManager begins a new turn. 2. System prompts Player for input for each active Ally. 3. Player selects "Skill" -> "Target" for Ally 1, 2, and 3. 4. System sorts all battlers (Ally + Enemy) by Speed. 5. System resolves actions sequentially (highest speed first). 6. Player watches animations and damage numbers. 7. Steps 1-6 repeat until all Enemies are 0 HP. 8. System emits battle_won signal.
Extensions	5a. Stun Effect: 1. Active battler has Status.STUN. 2. TurnManager skips their turn (perform_action is blocked). 3. Status duration decrements. 7a. Player Defeat: 1. All Allies reach 0 HP during the Execution phase. 2. System emits battle_lost signal. 3. Run terminates (See UC-01).


Player
TurnManager
Battler
BattleScene
Start Combat (Enemies Loaded)
loop
[Until Victory or Defeat]
New Turn Start
start_turn()
Phase 1: Decision
loop
[For Each Battler]
decide_action()
alt
[Is Player Controlled]
Request Input
Select Action/Target
[Is AI]
Weighted Random Selection
signal deciding_finished
_sort_by_speed()
Phase 2: Execution
loop
[For Each Sorted Battler]
perform_action()
play_anim("attack")
apply_damage_to_target()
signal performing_action_finished
check_win_condition()
Show Result (Win/Loss)

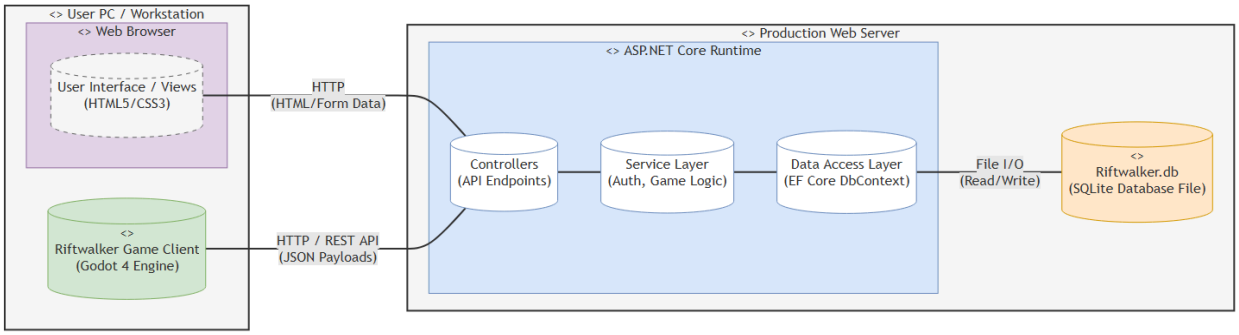
6.4 Purchase Item (Shop)

Use Case ID	UC-04
Use Case Name	Purchase Item from Shop
User Goal	To exchange earned currency (Coins) for items that improve Combat capability.
Scope	ShopMenu / LootManager (Local)
Level	Sub-Function Task
Primary Actor	Player
Preconditions	1. Player is at a SHOP node. 2. Global.coins > 0.
Minimal Guarantee	Coins are only deducted if the item is successfully added; Inventory integrity is maintained.
Success Guarantee	Global.coins decreases by Item.price, and Ally.items increases by 1.
Trigger	Player clicks on an Item Card in the Shop UI.
Success Scenario	1. Player hovers over item to see stats/price. 2. Player clicks the Item to purchase. 3. System checks if Global.coins >= Item.price. 4. System deducts Item.price from Global.coins. 5. System adds Item resource to the active Ally's inventory. 6. System plays Audio.purchase and updates the Coin UI. 7. System triggers automatic save.
Extensions	3a. Insufficient Funds: 1. Global.coins < Item.price. 2. System plays Audio.denied sound. 3. Transaction aborts; No coins deducted.

7. High-Level Architecture

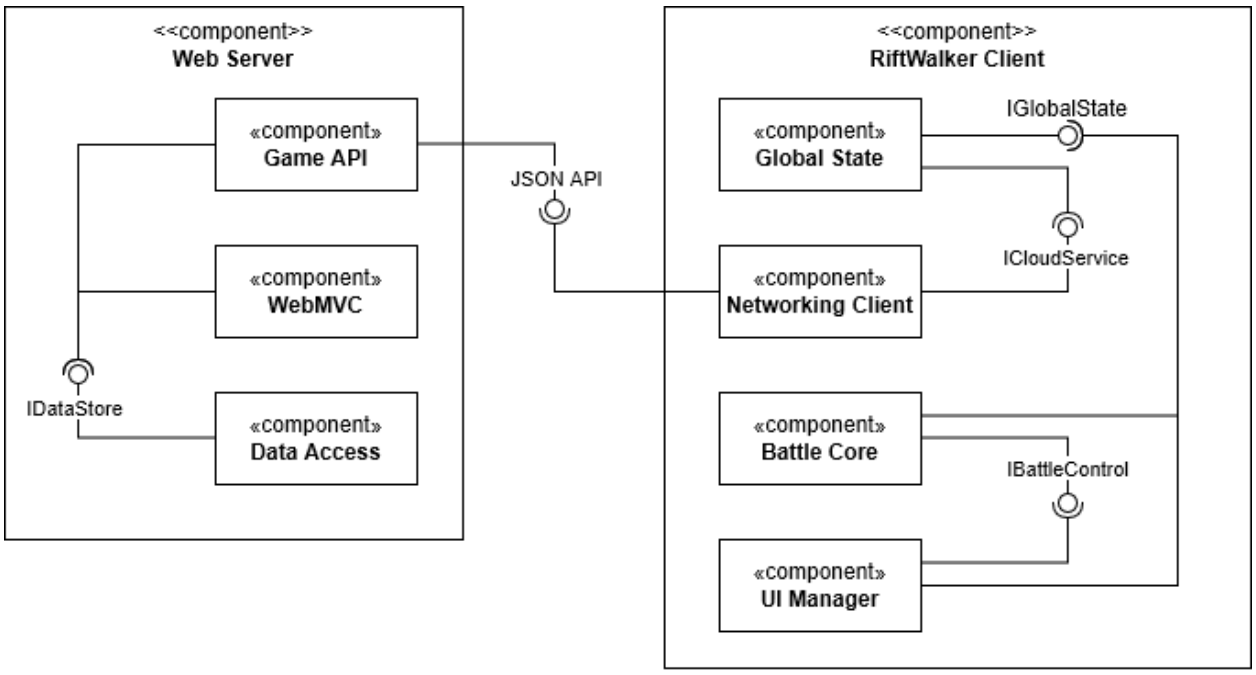
7.1 Architectural Views

1) Deployment Diagram



The deployment diagram demonstrates the use of physical hardware, execution environments (Godot/ASP.NET), and HTTP/SQLite in the RiftWalker system.

2) Component Diagram



The component diagram illustrates the logical breakdown (UI, Battle, Net, Data) of the RiftWalker system and how the system components use Interfaces to communicate with one another.

7.2 Design Patterns

1) Singleton Pattern

- **Implementation:** AuthManager.gd, Global.gd
- **Context:** The game requires a persistent user session and global settings (volume, game speed) that must survive across scene transitions (e.g., moving from Main Menu to Battle Scene).
- **Justification:** We chose the Singleton pattern (implemented via Godot Autoloads) to create a single source of truth for the User's authentication state. Since the Game Client loads entirely new scenes for battles, we needed a persistent node to hold the access_token and user_id. Passing this data manually between every scene would have created spaghetti code and brittle dependencies. The Singleton guarantees that any component, anywhere in the game, can reliably access the current user session without setup.

2) Observer Pattern

- **Implementation:** SignalBus.gd, Godot Signals (login_success, turn_phase_finished)
- **Context:** The User Interface (HUD) needs to update when game events happen (e.g., Player takes damage, Turn ends), but the Game Logic shouldn't know about Label nodes or UI implementations.
- **Justification:** To prevent tight coupling between our Game Logic and the UI, we adopted the Observer pattern using Godot's Signal system. This allows the Battle Manager to simply emit a turn_ended signal without caring who is listening. The UI Manager 'observes' these signals and updates the screen accordingly. This decision was critical for allowing our UI designers to iterate on the HUD layout without breaking the underlying battle code.

3) Strategy / Command Pattern (Data-Driven Design)

- **Implementation:** AllyAttack Resources (slash.tres, fireball.tres)
- **Context:** The game features multiple characters with distinct move sets. Hard-coding every attack method in the Battler script would lead to a massive, unmaintainable file.
- **Justification:** We utilized the Strategy pattern by defining Actions as interchangeable Resource objects. Instead of writing a function for every single attack, we defined a generic perform_action() contract. This allows our designers to create new skills just by creating a new Resource file in the editor and tweaking numbers (Damage, Target Type) without writing a single line of code. It effectively encapsulates the 'algorithm' of an attack into a portable object.

4) State Pattern (Finite State Machine)

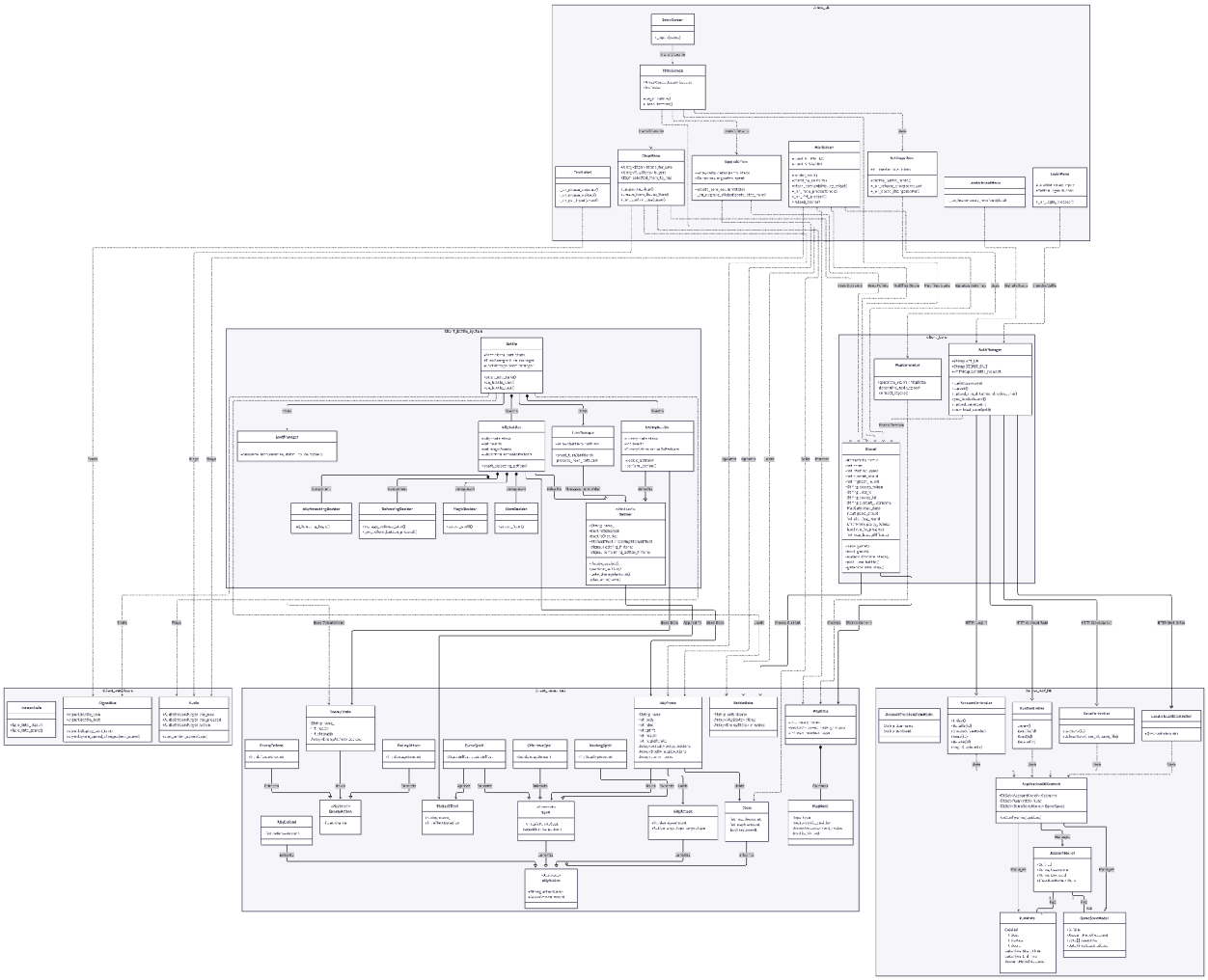
- **Implementation:** TurnManager.gd (Phases: Decision -> Sorting -> Execution)
- **Context:** A turn-based battle system has distinct phases that must be executed in a strict order. Mixing this logic into a simple _process loop would be error-prone.
- **Justification:** We implemented a lightweight State Machine within the Turn Manager to strictly enforce the flow of battle. By separating the 'Decision Phase' (where AI thinks) from the 'Execution Phase' (where animations play), we prevent race conditions where an enemy might try to attack while the player is still selecting a move. This structure ensures a bug-free, deterministic gameplay loop.

5) Model-View-Controller (MVC)

- **Implementation:** ASP.NET Core Web Server (AccountController, AccountModel, Views)
- **Context:** The backend needs to provide both the Game Client (API) functionality and the web interface for Users (Browser).
- **Justification:** On the server side, we favored the standard MVC architecture to separate our Data Access logic from different presentation layers. By isolating our Database logic into a shared layer, both the GameRunAPI (for the game client) and the WebMVC (for the browser leaderboards) use the same data rules. This ensures that a score uploaded by the game appears instantly and correctly on the website, while keeping the codebase organized and scalable.

8. Detailed Design

8.1 Class Diagram



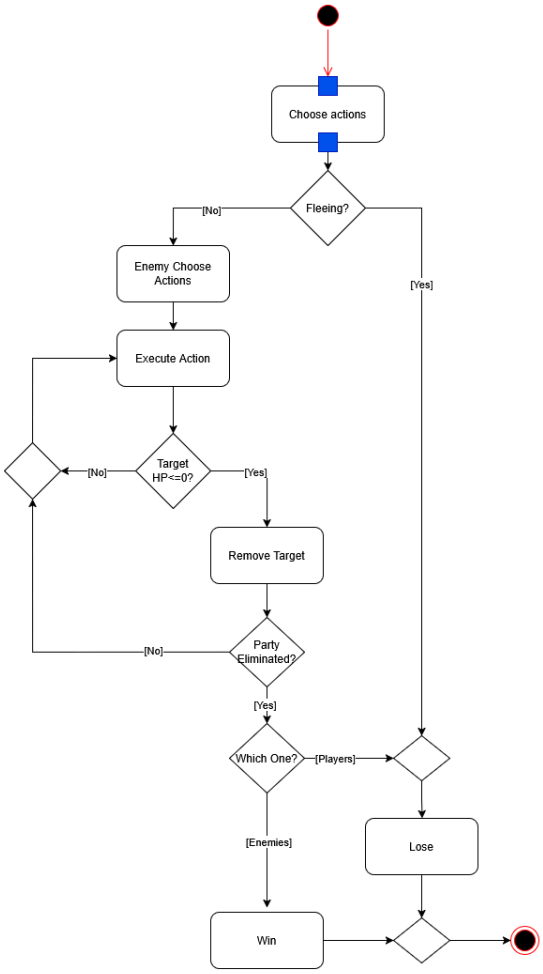
Since this diagram is too large to be read in this document, here is a link to the dynamic view.

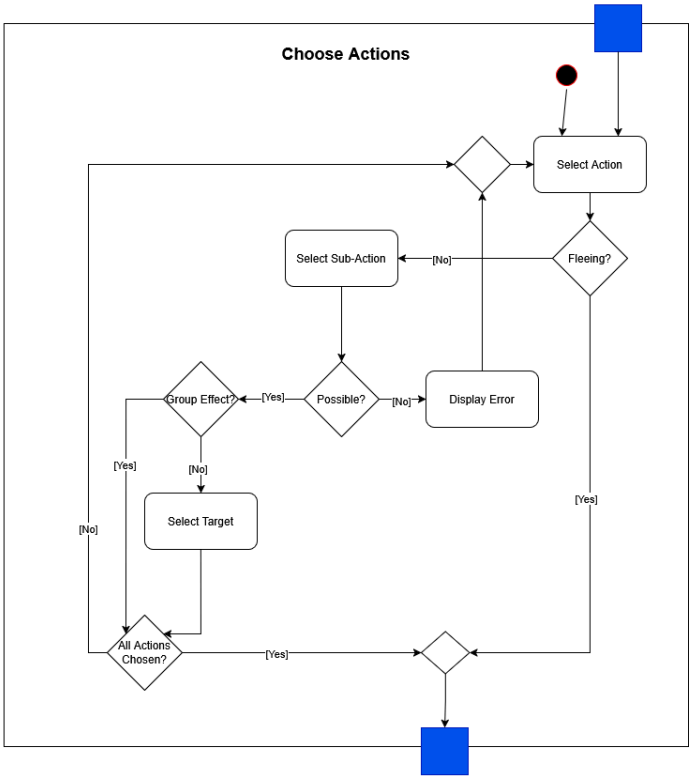
You may notice a variety of unique objects, such as “AudioStreamPlayer,” which aren’t defined in the diagram. This is because Godot has **hundreds** of built-in reference types, known as *node types*. For more information, check out Godot’s Nodes and Scenes documentation.

You may also notice that most of the defined methods are public. This is because in GDScript (Godot’s language), strict encapsulation is handled differently than in, say, Java. Since the language lacks private or protected keywords, Godot follows the “Convention over Configuration” philosophy, similar to Python. It enforces encapsulation at the developer level through conventions and trusts developers to respect boundaries.

We strictly follow the official GDScript Style Guide. Any method or variable prefixed with an underscore (e.g., _process_next_battler()) is effectively private. Even though the code technically allows external access, our team treats these as “internal-only” and does not call them from outside the class.

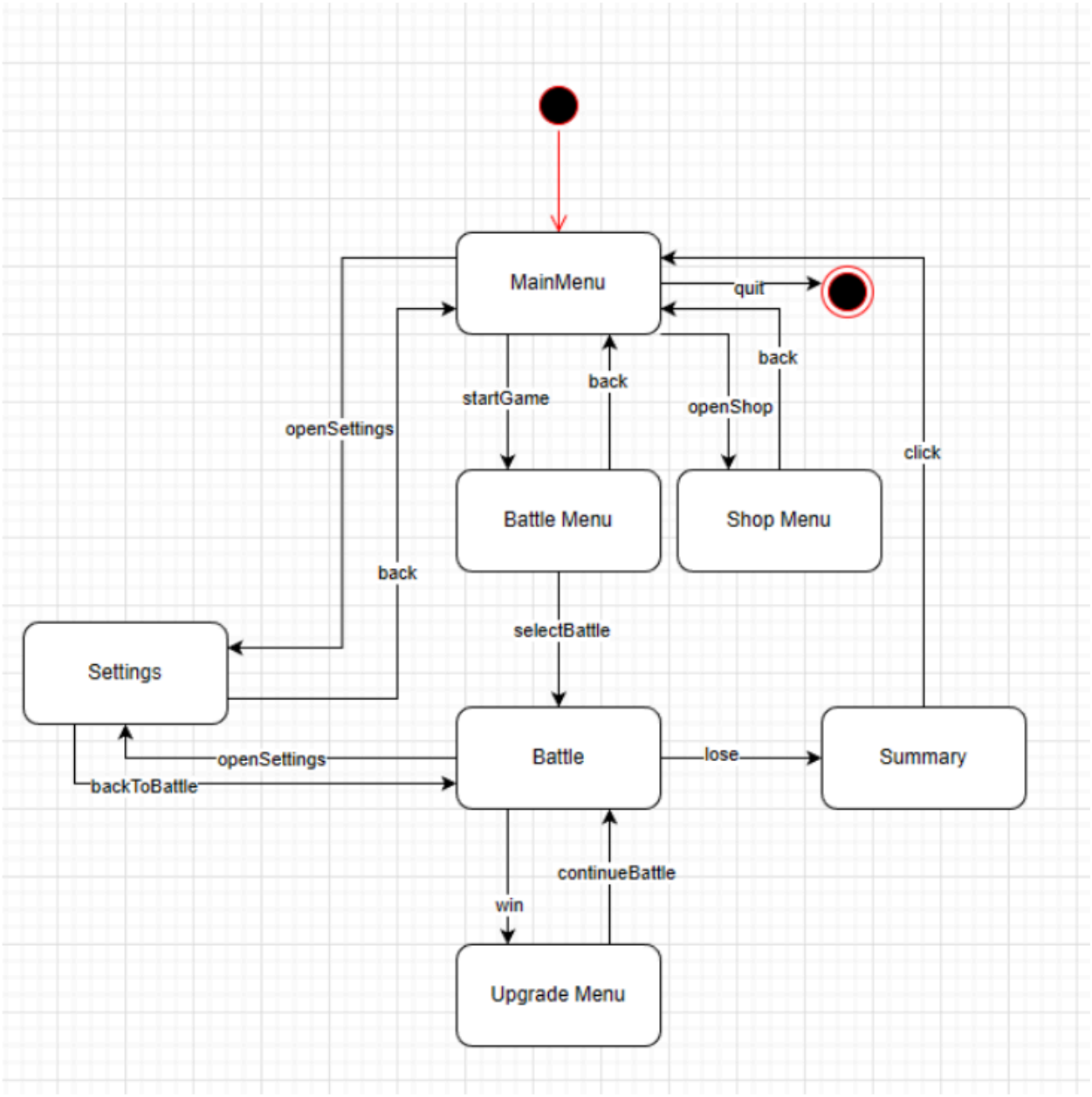
8.3 Activity Diagram





This activity diagram showcases the logical flow of the battle scene.

8.4 State Diagram



This state diagram demonstrates how the player can navigate between scenes.

9. Project Impact

9.1 Individual Impacts

Effect on Actions, Behavior, and Presence

- **Cognitive Engagement & Strategic Thinking:**
    - *Impact:* Riftwalker's turn-based combat and roguelike progression force players to engage in critical decision-making. Unlike passive entertainment, the system requires the user to analyze risk/reward ratios (e.g., "Do I spend gold now or save for the next layer?").
    - *Result:* This fosters an adaptable, long-term-planning mindset in the individual player, potentially sharpening their problem-solving skills in high-pressure micro-environments.
- **Resilience to Failure:**
    - *Impact:* As a roguelike, "Defeat" is a core mechanic, not a system failure. Players must accept loss (perma-death of a run) and restart.
    - *Result:* This cultivates emotional resilience. The individual learns to dissociate failure from "wasted time" and instead views it as a learning opportunity, interacting with digital media in a healthier, more growth-oriented way.
- **Accessibility & Convenience:**
    - *Impact:* The implementation of AuthManager's device-based auto-login allows seamless entry without friction.
    - *Result:* This reduces the barrier to entry, allowing individuals with limited time (e.g., casual sessions) to maintain a persistent digital presence without the administrative burden of account management.

9.2 Organizational Impacts

Effect on Performance, Culture, and People

- **Technical Upskilling & Versatility:**
    - *Impact:* The decision to fuse a Game Engine (Godot 4) with an Enterprise Web Stack (ASP.NET Core) required the team to bridge two distinct software cultures.
    - *Result:* The organization now possesses a rare hybrid competency. Team members are no longer just "Game Devs" or "Web Devs" but Full-Stack Engineers capable of architecting complex, distributed systems. This directly increases the organization's technical velocity and market value.
- **Culture of Documentation & Structure:**
    - *Impact:* The strict adherence to UML standards (Lollipop interface notation, sequence diagrams) and Design Patterns (Singleton, Command) forced a shift from "Cowboy Coding" to "Engineered Solutions."
    - *Result:* This created a disciplined engineering culture. Communication overhead was reduced because artifacts (diagrams) became the source of truth, creating a more scalable and resilient team structure where knowledge is less tribal and more institutional.
- **Live-Ops Mindset:**
    - *Impact:* Moving from a local-only game to a server-backed ecosystem (Leaderboards, Cloud Saves) introduced "always-online" responsibility.
    - *Result:* The organization had to adopt a service-oriented mindset, prioritizing uptime, data integrity (X-Integrity-Hash), and backward compatibility. This matures the team's operational capabilities, preparing them for larger-scale SaaS products.

9.3 Societal Impacts

Effect on Policy, Law, Communities, and Social Structures

- **Data Privacy & Digital Identity:**
    - *Impact:* By using DeviceId rather than personal emails/passwords, the project navigates the complex landscape of digital privacy (GDPR/CCPA) by minimizing PII (Personally Identifiable Information) collection.
    - *Result:* This promotes a "Privacy by Design" philosophy in the broader development community, demonstrating that compelling user experiences can exist without invasive data harvesting. It respects the user's digital sovereignty.
- **Community & Meritocracy:**
    - *Impact:* The Global Leaderboard creates a structured meritocracy where recognition is based solely on skill (Highest Round reached).
    - *Result:* This fosters a competitive micro-community. While small-scale, it mirrors broader social structures of competition. By enforcing anti-cheat measures (Checksums), the system strictly upholds "Fair Play," reinforcing social contracts that denounce cheating and value earned success.
- **Digital Preservation:**
    - *Impact:* The Cloud Save system ensures that a user's digital labor (their progress) is not tied to fragile local hardware.
    - *Result:* This contributes to the societal shift towards "Access over Ownership." Users increasingly expect their digital lives to be fluid and device-agnostic. Riftwalker supports this modern social expectation, reinforcing the norm that digital data should be persistent and ubiquitous.