堕天勇者と終焉の王都 -魔王に捧ぐ反逆の剣-

～Game Development Summary～

2025.11.22

日下　拓海

目次

1. 基本情報

ゲーム基本情報

ゲームタイトル ： 堕天勇者と終焉の王都 -魔王に捧ぐ反逆の剣-

コアコンセプト ： 3すくみ×3ターン制戦闘システムを採用した爽快感のあるRPG

対応プラットフォーム ： PC（Windows / macOS）

使用言語 ： C++17

制作期間 ： 2025.8 - 2025.11

所要時間数の概数 ： 100～150時間程度

動作環境 ： Windows 10 / 11、macOS

制作人数 ： 1人

ゲーム開発における役割 ： 全工程を担当しました

ゲームアピールポイント（3点）

・3すくみ×3ターン制を採用することで、無傷で一方的に敵を攻撃することもできる非常に爽快感のあるゲーム性になるように設計しました。

・『終焉解放』状態に移行することで、ゲームが苦手な方がすぐにゲームオーバーになってしまい楽しめない、という状況を軽減できます。

・RPGでは禁忌の行動である、街の住民と戦って倒すという非日常的な行動を取ることができます。本作の堕天勇者という立場を存分に利用することで、心を痛めつつも街の住民を倒して街を滅ぼすという貴重な体験ができます。

開発者情報

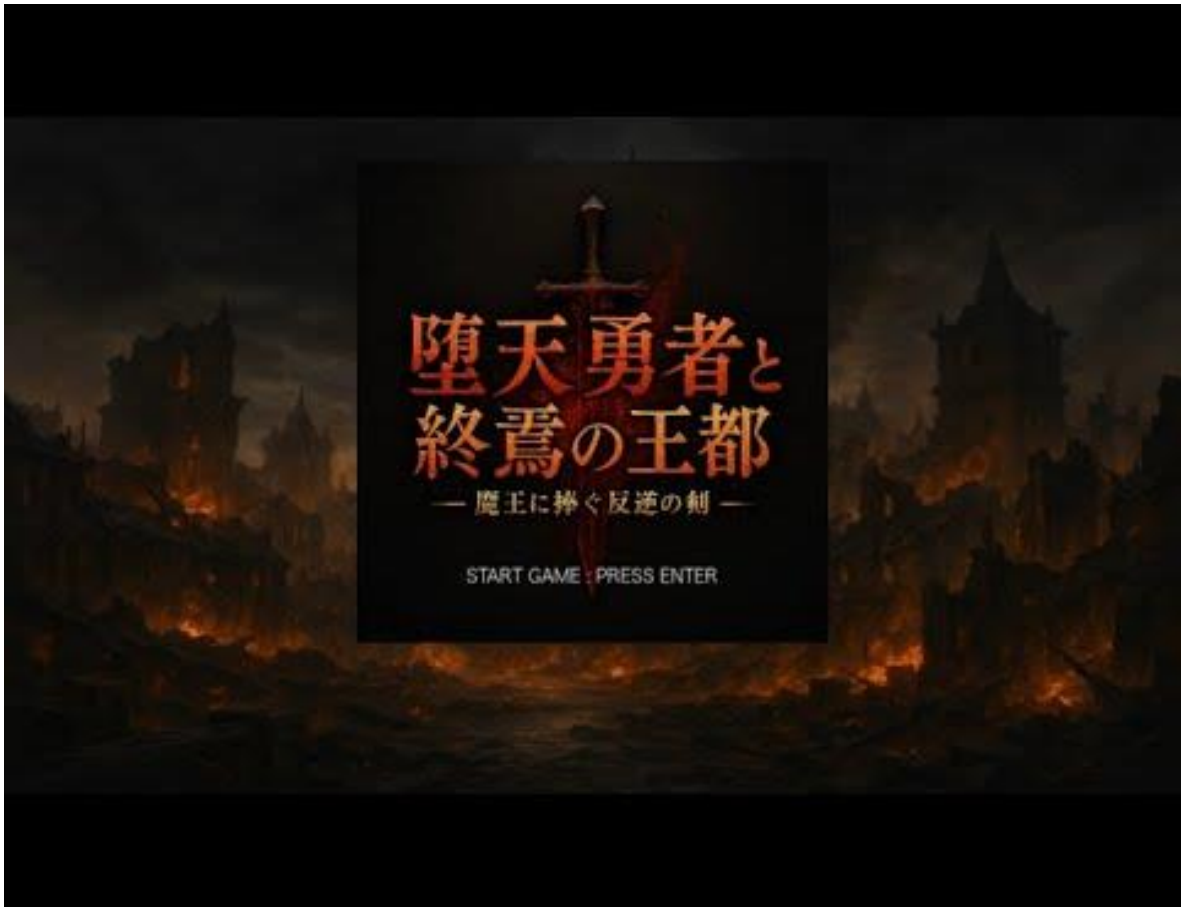
開発者名 ： 日下　拓海

所属 ： 弘前大学理工学部電子情報工学科

連絡先 ： takumi.090414528@gmail.com

プレイ動画／紹介動画

https://www.youtube.com/watch?v=vVOepP3KciU



本作のプレイ映像と、ストーリー、戦闘システム等に関する紹介をまとめた動画です。

2. ゲーム概要

基本ルール

本作の通常戦闘は、「攻撃／防御／呪文」による 3 すくみをベースとしたシステムです。プレイヤーは事前に 3 ターン分の行動を選択し、その行動と敵の行動を照らし合わせて、3 ターンそれぞれの勝敗を判定します。最終的に、3 ターン中の勝利数が多い側が、一方的にダメージを与えることができます。

プレイヤーの HP が敵の攻撃によって 0 になった場合、一度だけ「終焉解放」という特殊状態に移行します。この状態では、通常時よりも多い 5 ターン分の行動をあらかじめ選択することが可能になります。終焉解放中に設定した 5 ターンの行動によって敵を倒すことができれば戦闘勝利となり、倒しきれなかった場合はゲームオ

ーバーとなります。

通常戦闘とは別に、「住民戦」と呼ばれる特殊な戦闘も存在します。住民戦では、プレイヤーは「攻撃／隠密」を、住民は「恐怖／求援」のいずれかを選択します。

- 住民が「恐怖」を選択している状態でプレイヤーが「攻撃」を選択した場合、攻撃は成功します。
- 住民が「求援」を選択している状態でプレイヤーが「攻撃」を選択した場合、衛兵に発見されてゲームオーバーとなります。

そのため、行動予測 UI で住民の行動傾向を確認しつつ、相手が「求援」を選びそうな場面では「隠密」を選択してやり過ごすことが求められます。

フィールド上には一定間隔で敵が出現し、プレイヤーはそれらを倒しながら制限時間内に指定のレベルまで到達することを目指します。目標レベルに達すると「夜の街時間」が開始されます。夜の街時間では、衛兵に見つからないよう注意を払いながら住民との戦闘を行っていきます。最終的に、街に存在する全ての住民を倒し、街そのものを滅ぼすことが本作の最終的なゴールとなります。

コアゲームループ

フィールドを探索→敵との戦闘→レベルアップ→指定レベル達成→夜の街開始→住民を倒す(1 夜につき 3 人まで)→昼の街に戻り再度指定レベルまで戦闘を繰り返す

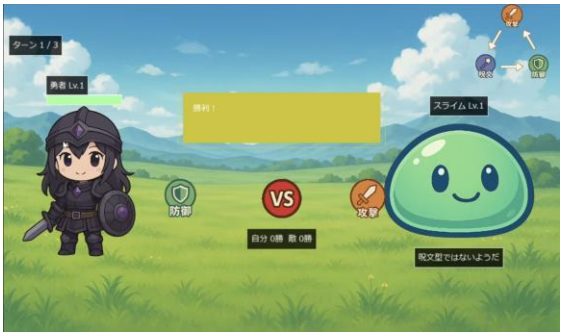
操作方法

W/A/S/D ： 移動

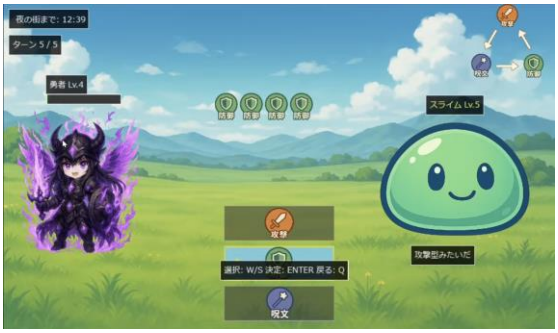
ENTER ： 決定/進行/インタラクション

ESC ： セーブ/ゲーム終了

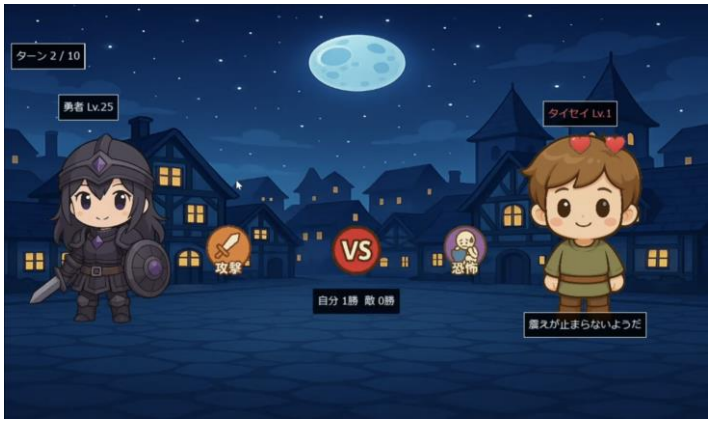
実際のゲーム画像



通常戦



終焉解放



住民戦

プレイ時間の目安(プレイヤーの習熟により異なります)

想定クリア時間 ： 1 時間程度

想定プレイヤー像

・3 すくみを考慮した行動選択のような、思考が必要なゲームが好きな方

・勇者の立場で街を滅ぼすといった、一般的には悪いことをゲームで行うのが好きな方

・アクションゲームのような、操作が複雑なゲームが苦手な方

※詳しいルールや実際のプレイは、紹介動画で解説しています。

3. 技術概要

使用技術

エンジン/フレームワーク：

・自作エンジン（SDL2）
言語：

・C++17
主なライブラリ：

・SDL2（ウィンドウ管理・入力処理）
・SDL2_ttf（フォント描画）
・SDL2_mixer（オーディオ）
・nlohmann/json（JSON 処理）
・SDL2_image（画像処理）
開発環境：

・クロスプラットフォーム対応（Windows / macOS）
ビルド・ツール類：

・CMake（ビルドシステム）

・Git / GitHub

主なシステム・モジュール

・SDL2Game：
　メインゲームループを管理し、毎フレームの更新処理と描画順序を制御します。タイトル画面やフィールドなど、画面遷移も一括して管理します。

・GameStateManager：
　ゲーム状態の読み込み・解放・進行を管理します。状態パターンに基づき、各 GameState のライフサイクル（enter / update / render / exit）と状態遷移を制御します。

・BattleState / BattleLogic / BattlePhaseManager：
　戦闘システムを担当します。3 ターン分のコマンドを事前選択して読み合いで勝敗を決める戦闘ルールを実装し、戦闘ロジック、フェーズ管理、アニメーション、エフェクト、UI 描画を分離して管理します。

・Player / Character：
　プレイヤーおよびキャラクターのステータス、ストーリー進行、信頼度などの情報を管理します。

・Graphics：
　SDL2 を用いたグラフィックス描画を担当します。テクスチャとフォントの管理、テキスト・図形の描画処理を提供します。

・InputManager：
　入力処理を担当します。キーボード入力の取得とキー押下状態の管理を行います。

・MapTerrain / TownLayout：
　マップおよび地形を管理します。地形の種類（草原、森、川、橋など）やマップデータ、地形レンダリングに加え、街（建物、住民、衛兵、城など）のレイアウト情報も扱います。

・AudioManager：
　オーディオを一元管理します。SDL2_mixer を用いて BGM や効果音の読み込み・再生・停止を行い、シングルトンパターンで実装しています。

・UIManager：
　ユーザーインターフェースを管理します。ボタンやラベル、メッセージボックスなどの UI 要素の追加・更新・描画・入力処理を統合的に扱います。

・Save/Load System：
　ゲームのセーブ／ロード機能を担当します。プレイヤーのステータス、進行状況、ゲーム状態などを JSON 形式で保存・読み込みします。

設計の方針・意識したこと

レイヤードアーキテクチャと責務分離：
　アプリケーション層、コア層、ゲームロジック層、エンティティ層、グラフィックス層、入出力層、UI 層などにレイヤー分割し、上位レイヤーが下位レイヤーのみに依存する構造としました。SOLID 原則（特に単一責任の原則）を意識し、プレイヤー情報や戦闘処理なども役割ごとにクラスを分割することで、保守性と拡張性を高めています。

状態パターンによるゲーム進行管理：
　ゲーム全体の画面やフェーズは GameState を基底とする各 State クラスとして実装し、GameStateManager がライフサイクル（enter / update / render / exit）と遷移を管理します。これにより、メインメニュー、フィールド、戦闘、夜パート

などを独立したモジュールとして実装でき、画面追加や仕様変更に対応しやすい構造としています。

継承・ポリモーフィズムとデータ駆動設計：
　キャラクターやゲーム状態などは基底クラスから派生させる設計とし、共通処理を集約しつつ、新しい種類の要素を追加しやすくしています。また、UI 設定やセーブデータは JSON で管理し、コードを変更せずにバランス調整や UI レイアウト変更が行えるようにしました。

リソース管理と開発効率：
　オーディオや UI 設定などの共通リソースはシングルトンで一元管理し、重複読み込みを防いでいます。さらに、UI 設定ファイルのホットリロード機能を実装し、ゲームを再起動せずに UI の調整を行えるようにすることで、開発・調整サイクルの効率を向上させています。

テスト・品質向上のために行った工夫

・コマンドライン引数によるデバッグ機能：
　起動時に「--debug」オプションを指定することで、特定の状態から直接ゲームを開始できるようにしています。room、town、night、castle、demon、field、battle などの状態や、各種敵との戦闘状態（battle_slime、battle_goblin、battle_dragon など）を指定可能です。デバッグモードではプレイヤーのレベルやステータスを自動設定し、テストプレイにかかる手間を削減しています。

・UI 設定のホットリロード機能：
　UIConfigManager が UI 設定ファイル（ui_config.json）の更新時刻を監視し、変更があれば自動で再読み込みします。ゲームを再起動せずに UI レイアウトや色などを調整できるため、UI 周りの検証・微調整を素早く行えます。

・JSON 形式によるデータ管理：
　セーブデータと UI 設定を JSON 形式で管理しています。セーブデータではプレイヤーのステータス、進行状況、ゲーム状態などを保存し、UI 設定では各 State の UI 要素の位置・サイズ・色などを定義します。これにより、コードを変更することなくテスト用データの作成やバランス調整が行いやすくなっています。

拡張性

・状態パターンによる拡張性：
　GameState 基底クラスから MainMenuState、FieldState、BattleState などの各

State を派生させています。新しい State を追加する際は GameState を継承して登録するだけでよく、既存の State に大きな変更を加えずに画面やフェーズを拡張できます。

・敵システムの拡張性：
　Character 基底クラスから Enemy を派生させ、EnemyType で敵の種類を定義しています。新しい敵を追加する際は EnemyType に種類を追加し、ステータスや報酬、行動パターンを定義するだけで対応できます。敵ごとの行動 AI や特殊技も Enemy クラス群を拡張する形で実装できるため、新しい敵キャラクターを増やしやすい構造になっています。

・戦闘システムの拡張性：
　戦闘は BattleState を中心に、BattleLogic（戦闘ロジック）、BattlePhaseManager（フェーズ管理）、BattleAnimationController（アニメーション）、BattleEffectManager（エフェクト）、BattleUI（UI 描画）といったクラスに役割を分割しています。新しい戦闘フェーズや特殊技、演出を追加する場合も、対応するクラスに処理を追加するだけで拡張でき、既存の戦闘フローへの影響を最小限に抑えられます。

4. 開発プロセス

開発プロセスの概要(タイムライン)

2025.08-09：プロトタイプ版の実装（ゲーム全体の基本機能・コアシステム）

2025.11：プレイフィールと設計を見直し、戦闘システムを中心にゲーム全体をほぼ全面的に再設計・再実装

開発開始時は夏季休暇期間だったため、週 50 時間以上の作業時間を目標に、集中的にプロトタイプ開発を行いました。その後、実際のプレイ感やコードの見通しを踏まえて 11 月に構成を大きく見直し、特に戦闘まわりを中心にゲーム全体をリファインしています。

開発体制・担当範囲

開発体制 ： 個人開発

担当範囲 ： 企画、ゲームデザイン、設計、実装、テスト、デバッグまで、ゲーム開

発の全工程を一人で担当しました。
※GitHub 上では開発環境の都合で、Windows 実機検証用に協力者のアカウントをコラボレーターとして追加していますが、実装・作業内容はすべて本人によるものです。

ワークフロー・ツール

・バージョン管理：
Git / GitHub を使用しました。master ブランチとテスト用の stg ブランチを分けて運用し、機能ごとに細かくコミットを行うことを意識して開発しました。

・タスク管理：
Notion を使用しました。タスクを優先度ごとに整理し、重要なものから順に実装を進めました。また、開発中に発生した課題とその解決方法を随時まとめるほか、開発スケジュールの管理にも活用しました。

開発における課題とその解決

開発中に直面した課題とその解決について、課題→解決→効果→学びという 4 つの観点から紹介します。

課題①：戦闘とレベル上げループが単調で「作業」になっていたこと

課題：

当初のプロトタイプでは、RPG でよくある「自分と敵が 1 ターンずつ交互に攻撃する」戦闘システムを採用していました。攻撃コマンドで通常攻撃、呪文コマンドで MP を消費して威力の高い呪文（攻撃力の 1.25 倍ダメージなど）を使う、というオーソドックスな構成で、フィールドに出る → 敵にぶつかって戦闘開始 → 攻撃コマンドを連打 → 敵を倒してレベルアップ、というループになっていました。

しかし、この構成では「とにかく攻撃を選び続ければレベルが上がる」だけのゲームになってしまい、レベル上げも戦闘も作業感が強く、プレイしていてすぐに飽きてしまう問題がありました。

解決：

攻撃／防御／呪文を単なるコマンドとしてではなく、「三すくみの関係を持つ行

動」として再定義しました。敵ごとに「攻撃を選びやすい」「呪文を選びやすい」などの行動傾向を設定し、プレイヤーはその傾向を読みながら自分の行動を選択する、という構造に変更しました。これにより、プレイヤーは毎ターン「どの行動を選べば有利を取れるか」を考える必要が生まれ、単純なコマンド連打から「読み合い」を楽しむ戦闘へとシフトさせました。

効果：

戦闘中に常に相手の行動傾向を意識しながらコマンドを選ぶ必要が出てきたことで、「とりあえず攻撃連打」で済む場面がほとんどなくなりました。レベル上げも、単に数値を上げる作業ではなく、「どう行動を読んで有利を取るか」を試行錯誤する時間になり、戦闘を繰り返すモチベーションが大きく向上しました。結果として、ゲーム全体の面白さと没入感を支えるコア要素として、戦闘システムが機能するようになりました。

学び：

RPG の既存の戦闘システムにそのまま倣うのではなく、「何が面白さを生んでいるのか」を分解して、自分なりの答えを設計することの重要性を実感しました。特に、本作では「読み合い」と「選択の重み」をどう作るかを意識し、自分自身が遊んでいて本当に楽しいと思える戦闘システムを考案し、実装することの大切さを学びました。

課題②：読み合いに失敗したプレイヤーがすぐゲームオーバーになってしまう問題

課題：

当初は、プレイヤーの HP が 0 になった瞬間に即ゲームオーバーになる設計としていました。そのため、運が悪く連続で一方的に攻撃を受けてしまった場合や、三すくみの読み合いに慣れていないプレイヤーがミスを重ねた場合、短時間でゲームオーバーになってしまう状況が発生していました。結果として、「少しミスするとすぐに終わってしまう」「読み合いが得意でないと遊びにくい」と感じられ、モチベーションが続きにくいという問題がありました。

解決：

ゲームオーバーまでの猶予を持たせつつ、最後にもう一度だけ逆転のチャンスを与えるために、「終焉解放」という復活要素を導入しました。HP が 0 になっても一

度だけ特殊状態に移行し、その状態で最大 5 ターン分の行動をあらかじめ選択できるようにすることで、「ここから逆転できるかもしれない」というラストチャンスをプレイヤーに提示する構造に変更しました。

効果：

終焉解放という特殊状態に入ること自体がひとつの見せ場となり、「やられたけれど、ここからまだ戦える」という前向きな気持ちでプレイを続けやすくなりました。終焉解放中は通常よりも多くのターンを使って大きなダメージを狙えるため、一方的に攻撃されて負けてしまうだけの展開が減り、逆転勝利のパターンが生まれました。その結果、読み合いに慣れていないプレイヤーでも、すぐに心が折れずに戦闘を繰り返しやすくなりました。

学び：

難易度や緊張感を重視するあまり、「ミスしたら即ゲームオーバー」という設計に寄りすぎると、一部のプレイヤーにとっては理不尽さや敷居の高さにつながってしまうことを実感しました。一度だけ強力な救済を用意することで、「挑戦しがい」と「遊び続けやすさ」のバランスを取れることを学びました。今後のゲーム制作でも、失敗したときにどこまでプレイヤーにリトライの余地を残すかを意識して設計していきたいと考えています。

課題③：住民戦が運ゲーに感じられ、理不尽さが目立っていたこと

課題：

住民戦では、プレイヤーは「攻撃／隠密」、住民は「恐怖／求援」を選択するという二択の読み合いを採用しています。当初の実装では、住民の行動傾向や次の行動を予測するための情報がほとんどなく、「攻撃を選んだら、たまたま住民が求援を選んでいて即ゲームオーバーになった」という状況が頻繁に発生していました。

その結果、プレイヤーから見ると「自分の判断よりも運に左右されている」「どれだけ考えても結局は当たり外れ」という印象が強くなり、意図していた“緊張感のある読み合い”よりも、理不尽さやストレスの方が目立ってしまう問題がありました。

解決：

プレイヤーが次の行動をまったく読めない状態を避けるために、「行動予測 UI」を導入し、住民が次に選びそうな行動をヒントとして表示する仕組みを追加しました。

行動予測UIのテキストを読むことで、「次のターンは必ず恐怖を選ぶ」といった形で、ある程度住民の行動を予測できるようにしました。一方で、行動予測UIばかり見ていれば常に正解が分かってしまい、逆に単調になるのを防ぐため、一定確率で「様子をうかがえない」といったメッセージを表示し、次の行動が予測できない状況も意図的に混ぜるようにしました。

効果：

多くの場面では、行動予測UIを手がかりに「ここは攻撃で押すべきか」「求援されそうだから隠密でやり過ごすべきか」を考えられるようになり、プレイヤーの選択に根拠が生まれました。一方で、予測できない状況では、一か八かで攻めるか守るかを自分で決断する必要があり、「読める場面」と「読み切れない場面」が混ざることで、住民戦全体の駆け引き感と緊張感が強まりました。

その結果、完全な運任せという印象は薄れ、「読んで当てた／外した」という納得感のある読み合いとして感じてもらいやすくなりました。

学び：

二択のシンプルな読み合いであっても、プレイヤーに何の手がかりも与えないと、簡単に「運だけで決まる」と受け取られてしまうことを実感しました。重要なのはランダム要素そのものではなく、「プレイヤーが自分なりの根拠を持って選択できるかどうか」であり、そのためには情報の出し方や“不確実さの混ぜ方”を丁寧に設計する必要があると学びました。今後も、リスクの高い選択をさせる場面では、理不尽さではなく「納得感のある緊張」を生む情報設計を意識していきたいと考えています。

5. まとめ

本作品で達成できたこと

・「攻撃／防御／呪文」の3すくみと、3ターン先読みのコマンド選択を組み合わせた戦闘システムを実装し、無傷で一方的に攻撃できる爽快感と、読み合いの緊張感を両立したバトル体験を実現しました。

・本作固有の要素である終焉解放や住民戦を通してプレイヤーの感情に強く働きかけるゲーム体験を形にしました。

・C++17 と SDL2 を用いた自作エンジン上で、状態パターンによる画面管理、JSONベースのセーブ／ロードやUI設定、デバッグ起動などを含むゲーム全体を一人で設計・実装し、Windows / macOS 向けのプレイアブルなRPGとして完成させました。

今後の展望

・本作の「読み合い」というゲーム性を生かし、ローカル通信やインターネット通信による対戦機能を実装し、プレイヤー同士で駆け引きを楽しめるようにしたいと考えています。

・「レベル上げ → 夜の街 → レベル上げ」というループがやや単調に感じられる部分があるため、敵の行動パターンのバリエーション追加や、ループの合間に挿入されるストーリーイベント、別フィールドへの移動解禁などを導入し、プレイ中常に新しい体験が得られる構成にしていきたいです。

・現状では敵にスキルや特殊技がなく、モンスターが変わっても違いが分かりにくい課題があるため、敵専用のスキルや特殊技、固有演出を追加し、敵ごとに異なる攻略や見た目のインパクトを楽しめる戦闘へと発展させていきたいと考えています。