

# 鍛冶師と不思議な魔塔

解説ドキュメント

MAI ZHICONG  
(バク チソウ)

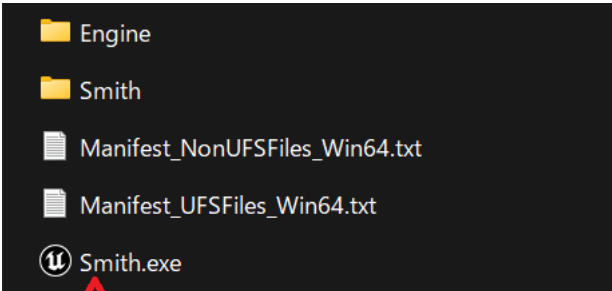
日本電子専門学校 ゲーム制作研究科

住所：東京都新宿区東五軒町 4-14 フレスクーラ神楽坂 601  
TEL：080-9370-9449  
Mail：[23cu0226@jec.ac.jp](mailto:23cu0226@jec.ac.jp)

## 作品概要

ジャンル	3D ロードライク風 RPG
作成環境	Unreal Engine5.3.2・Visual Studio Code
作成期間	3 ヶ月（2024 年 12 月～2025 年 3 月）
作成メンバー数	6 人（プランナー1 人、デザイナー2 人、プログラマー3 人）

## 実行概要

実行環境	PC (Windows11)
実行手順	<div><p>実行ファイル/Smith.exeを起動</p></div>

## フォルダ構成

成果物リスト	フォルダパス
操作説明書	実行ファイル/操作説明
実行ファイル	実行ファイル/Smith.exe
プロジェクトデータ	Project

モジュール	フォルダパス
ダンジョン生成、管理	Project/Source/MapManagement
インゲーム（プレイヤー、UI、ギミック）	Project/Source/Smith
ゲームシステム	Project/Source/SmithGameplay
データ定義、管理	Project/Source/SmithModel
敵パラメーター初期化、アイテム生成	Project/Source/SmithModelInitializer
ターン制バトルシステム	Project/Source/TurnBattleSystem

## 担当箇所

<u>ランダムダンジョン生成、管理</u>
<u>ターン制バトルシステム</u>
<u>ゲームシステム</u>
ダンジョンイベント
アイテム
敵 AI ビヘイビアー制御
<u>プレイヤー制御</u>
<u>データ管理</u>
強化画面、ミニマップ UI 制御

（アンダーバーがある項目を解説します。）

## こだわりポイント

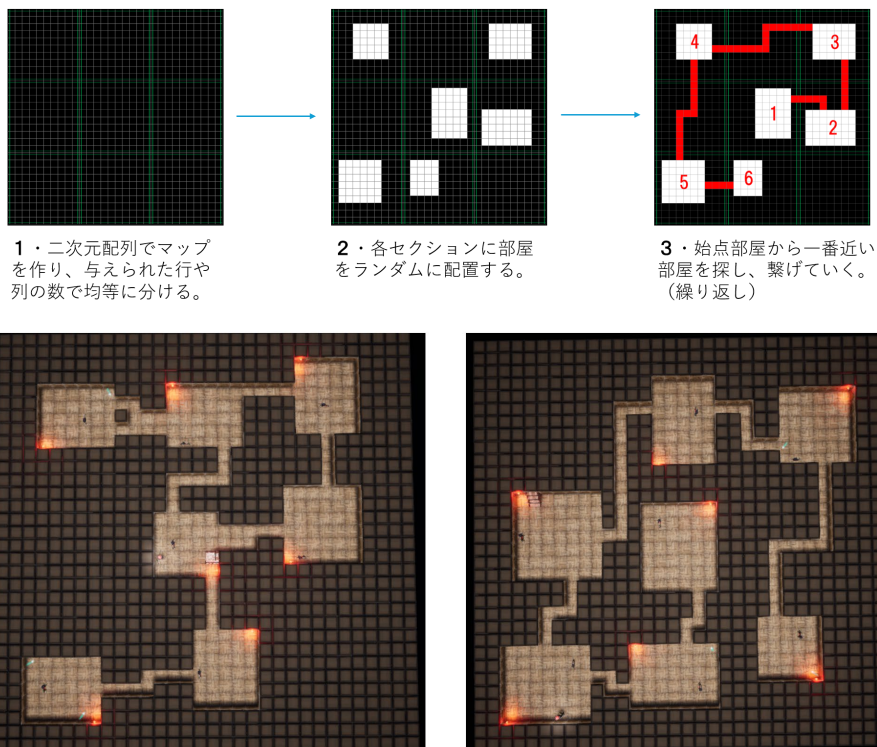
機能追加コスト削減のため柔軟性の高いゲームシステムを構築

仕様変更が簡単に行えるため機能毎にモジュールを作成し、依存関係を減らす

## 担当箇所解説

### ・ ランダムダンジョン生成、管理

#### └ ランダムダンジョン生成



工夫した点：

基本的なロジックを実装し、パラメーターを設定するだけでランダムなダンジョンを生成することができます。

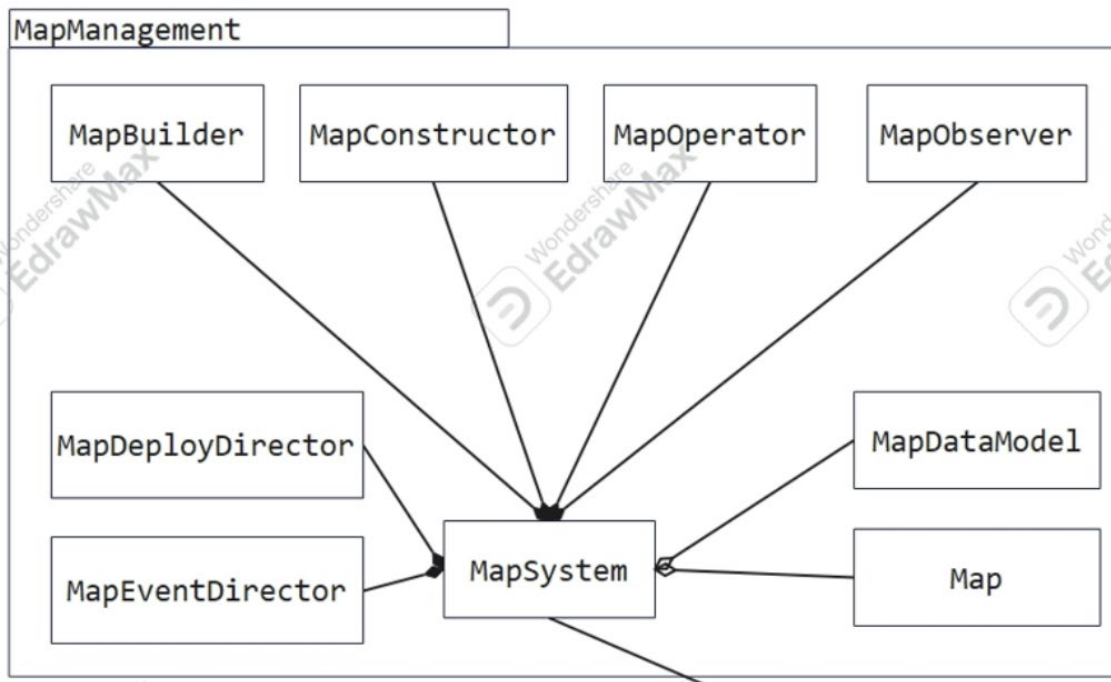
該当ソースコード：

・ 生成ロジック：

Source/MapManagement/MapCore/SmithMap.h

Source/MapManagement/MapCore/SmithMap.cpp

## └ ダンジョン管理



工夫した点：

外部から MapSystem 経由でダンジョンに対して操作命令を出します。

MapSystem の肥大化を防ぐためそれぞれ役割をもつクラスインスタンスを作り、ダンジョンを操作します。

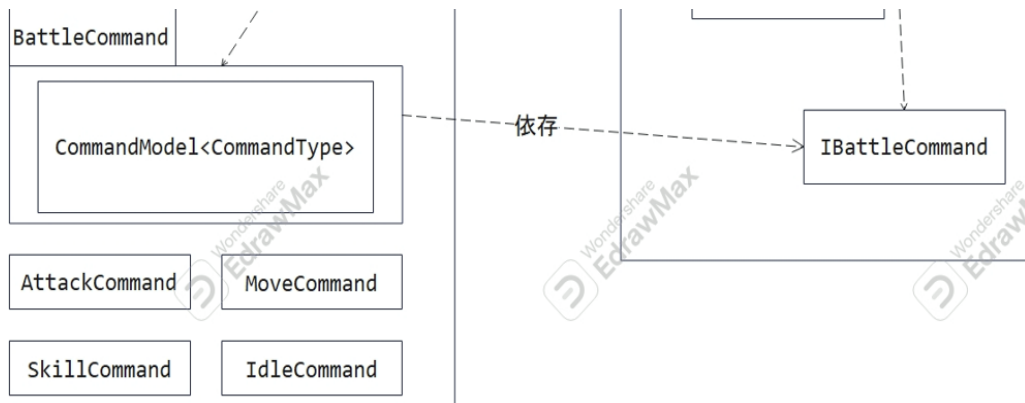
該当ソースコード：

Source/MapManagement/Public/SmithMapSystem.h

Source/MapManagement/Private/SmithMapSystem.cpp

## ・ ターン制バトルシステム

### └ バトルをコマンド化



工夫した点：

コマンドパターンを利用し、プレイヤーや敵の行動をコマンドとして処理します。

該当ソースコード：

#### ・ コマンド：

Source/Smith/BattleCommand/SmithBattleCommand.h

Source/Smith/BattleCommand/AttackCommand.h

Source/Smith/BattleCommand/IdleCommand.h

Source/Smith/BattleCommand/MoveCommand.h

Source/Smith/BattleCommand/SkillCommand.h

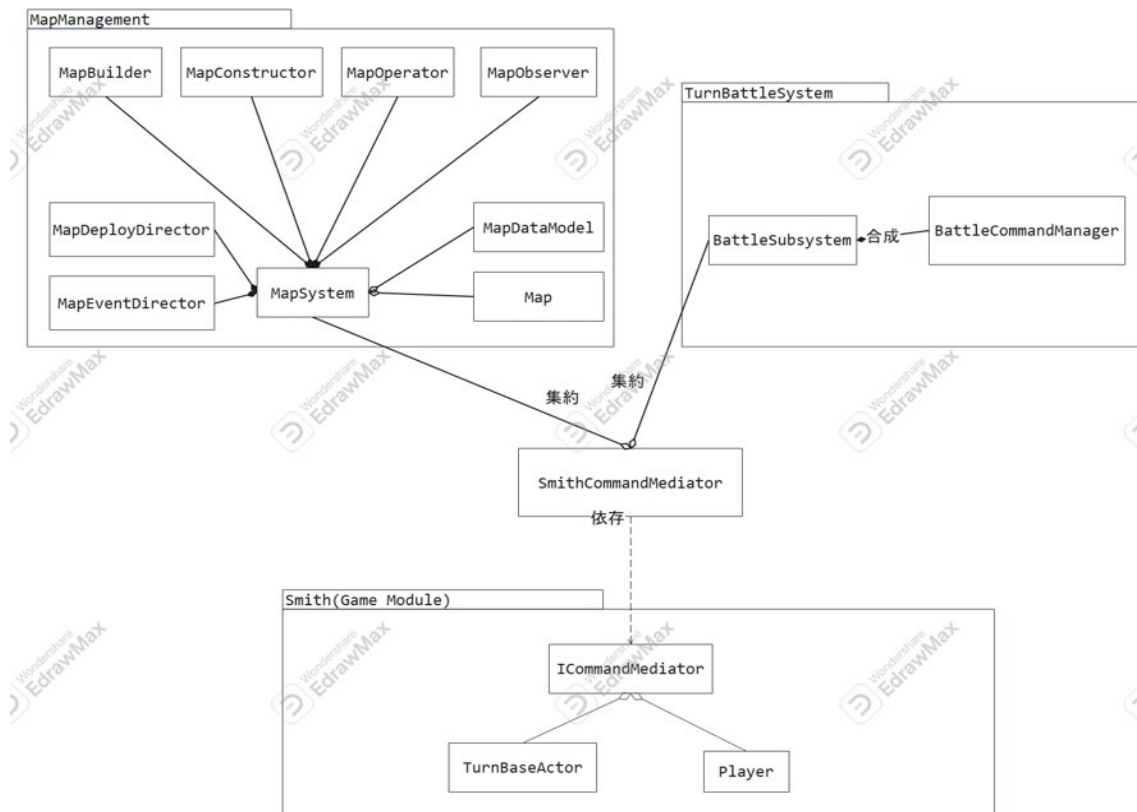
#### ・ コマンド管理：

Source/TurnBattleSystem/Public/BattleCommandManager.h

Source/TurnBattleSystem/Private/BattleCommandManager.cpp

## ・ ゲームシステム

### └ システムへのアクセスを簡易化



工夫した点：

複数のモジュールのアクセスを簡易化するため、一つの仲介オブジェクトを作り、そこで処理をまとめます。

上位モジュールにあるプレイヤーや敵が直接下位モジュールを依存しないように、インターフェースで依存関係を逆転させます。

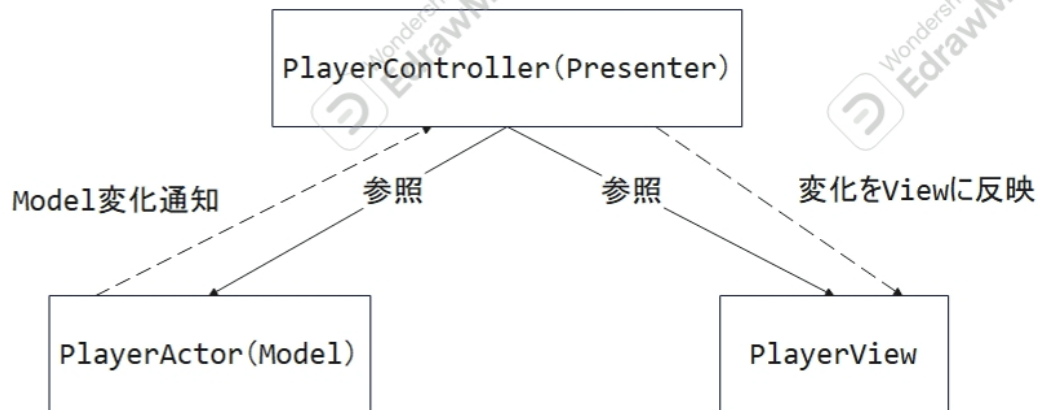
該当ソースコード：

Source/SmithGameplay/Public/SmithCommandMediator.h

Source/SmithGameplay/Private/SmithCommandMediator.cpp

## ・プレイヤー制御

└ 関心の分離により保守性向上



工夫した点：

プレイヤーGUI 周りの設計の保守性を高めるために、ビジネスロジックとユーザーインターフェースを分離させました。Controller が入力イベントを受け取り、Actor は入力に対して適切な計算をし、その結果を View に通知を送る仕組みを作りました。

該当ソースコード：

Source/Smith/Player/SmithPlayerActor.h/cpp

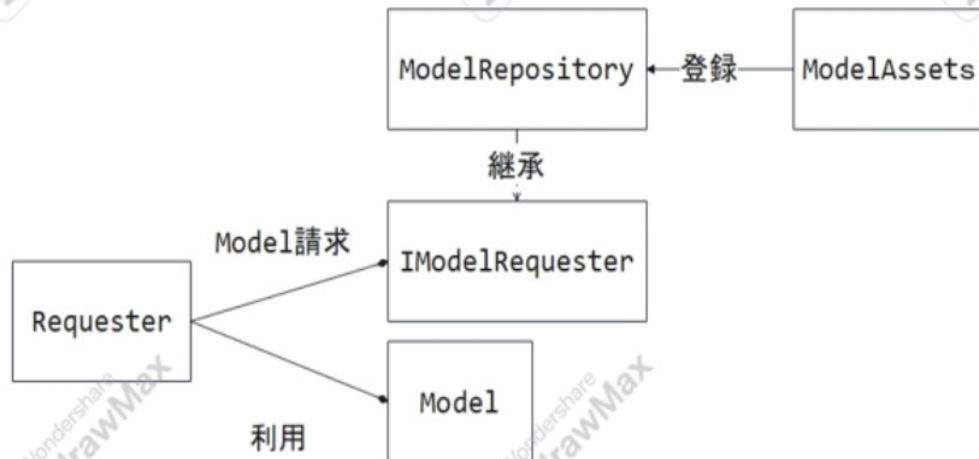
Source/Smith/Player/SmithPlayerController.h/cpp

Source/Smith/Player/SmithPlayerView.h/cpp



## ・データ管理

└データを一括で管理、取得



工夫した点：

ダンジョンパラメーター、ログ出力などのデータ全てをビジネスロジッククラスの中に入れるとクラスが複雑になり、肥大化に繋がります。データとビジネスロジックを分離して、利用する時リポジトリから取得するようにしました。ビジネスロジッククラスが軽量化でき、データ管理がしやすくなりました。

該当ソースコード：

モデル `Source/SmithModel/Models/SmithXXXModel.h/cpp`

アセット `Source/SmithModel/Models/SmithXXXModelDefinition.h`

リポジトリ `Source/SmithModel/Models/SmithXXXModelRepository.h`

## 参考書籍・サイト一覧

- ・ダンジョン生成：

[【風来のシオン風】 いい感じにランダムで、いい感じに恣意的なランダムダンジョンを生成する \[Unity\]](#) （ Qiita ）

- ・ダンジョン管理：

[Facade パターン](#) （ Refactoring Guru ）

[Mediator パターン](#) （ Refactoring Guru ）

- ・バトルシステム

[Command パターン](#) （ Refactoring Guru ）

[【C++】静的ポリモーフィズムによる環境依存コードの抽象化](#) （トイロジック技術開発ブログ）

[Breaking Dependencies - Type Erasure - The Implementation Details](#) by Klaus Iglberger （ GitHub ）

- ・ゲームシステム

Clean Architecture 達人に学ぶソフトウェアの構造と設計 （ 書籍 ）

[クリーンアーキテクチャを少し説明できるようになれる記事](#) （ Qiita ）

- ・プレイヤー制御

[起源から整理する GUI アーキテクチャパターン](#) （ Zenn ）

- ・データ管理

[DDD を実践するための手引き（リポジトリパターン編）](#) （ Zenn ）

- ・Unreal Engine

[Unreal Engine C++ API Reference](#) （ 公式 ）

[Unreal Engine UI Tutorials](#) （ Benui ）

## 自己評価

ゲームプレイ部分	<ul style="list-style-type: none"><li>・ <b>良かった点</b> :<ul style="list-style-type: none"><li>※ランダム生成で毎回違うダンジョンが探索できる。</li><li>※生成したダンジョンが不備なく部屋は全部繋がっている。</li></ul></li><li>・ <b>悪かった点</b> :<ul style="list-style-type: none"><li>※敵やアイテムの種類が少ない。</li><li>※武器強化へのヒントが少ないため、探索して手に入れた素材の使い道がわかりにくい。</li><li>※敵を倒さないとクリアしにくいいため、プレイヤーを考えさせる駆け引きの部分がない。</li></ul></li><li>・ <b>これからの改善点</b> :<ul style="list-style-type: none"><li>※敵、ギミック、アイテムの種類を増やす。</li><li>※強化操作をわかりやすく説明する。</li><li>※武器の種類を増やしてプレイヤーに強化の選択肢を与えることによって、駆け引きを作る。</li></ul></li></ul>
プログラム部分	<ul style="list-style-type: none"><li>・ <b>良かった点</b> :<ul style="list-style-type: none"><li>※コーディング規約があり、全体的にコードの一貫性がある。</li><li>※マジックナンバーが少なく、コードが理解しやすい。</li><li>※一部のデータはロジックと分離し、バランス調整がしやすい。</li><li>※柔軟性の高いシステムを作成でき、変更と修正がしやすい。</li></ul></li><li>・ <b>悪かった点</b> :<ul style="list-style-type: none"><li>※一部のインターフェースはテンプレートを使っていて、抽象度が高く、他のプログラマーが理解しにくい内容になっている。</li><li>※プレイヤークラスは多数のインターフェースを利用して、依存関係が複雑になり、クラスが肥大化になりやすい。</li><li>※同じデータのパラメーターが別々で保存されるため、修正漏れの恐れがある。</li><li>※コメントが少ない。</li></ul></li><li>・ <b>これからの改善点</b> :<ul style="list-style-type: none"><li>※インターフェースを使いやすいように設計し、抽象度の高い部分を隠蔽する。</li><li>※必要最低限のインターフェースを用意して、無駄な抽象度を減らす。</li><li>※コメントを書き、モジュール毎にドキュメントを作成することに心掛ける。</li></ul></li></ul>

## 周りの評価

プランナーから	<ul style="list-style-type: none"><li>・ <b>良かった点:</b><ul style="list-style-type: none"><li>※パラメーター調整がやりやすかった。</li><li>※ダンジョンの作りでランダム生成を作ってくれた。</li><li>※ゲームの要素について相談に乗ってくれた。</li></ul></li><li>・ <b>悪かった点:</b><ul style="list-style-type: none"><li>※プログラマー間のコミュニケーションをもっととってほしかった。</li><li>※要素をゲームとして実装するのが遅れていた。</li></ul></li><li>・ <b>これからの改善点:</b><ul style="list-style-type: none"><li>各プログラマーとの進捗の確認と他のプログラマーが作った要素を早く実装する。</li></ul></li></ul>
プログラマーから	<ul style="list-style-type: none"><li>・ <b>良かった点:</b><ul style="list-style-type: none"><li>※コメントは少ないが、設計面での工夫が随所に感じられる</li><li>※コメントは少ないが、コードのボリュームもあり、作品制作への熱量を感じる</li></ul></li><li>・ <b>悪かった点:</b><ul style="list-style-type: none"><li>※関数ヘッダーコメントくらいは欲しい。</li><li>※クラス分けすぎ。特に1クラス1関数など。</li><li>※ファイルも分けすぎだと思う。1クラス1ファイルでないといけないわけじゃなかったはず。</li><li>※“SmithGod”フォルダ等、命名への配慮はもう少し必要</li></ul></li><li>・ <b>これからの改善点:</b><ul style="list-style-type: none"><li>※分からない訳ではないが多少無茶でも一つのクラスに統合したほうが管理しやすいと思う。</li><li>※関連する小規模なクラスは全部1つのファイルにまとめたほうが効率的だと思う。</li></ul></li></ul>