13

CHAPTER

機能拡張のアイデア

LLMと相談しながら、何を、どの順で作るかを決める

すべてを実装する必要はない — 自分のプロジェクトに合ったものを少しずつ

AGENDA

この章で学ぶこと

01 追加できる機能

マルチエンディング・セーブ・ランキング・画像生成 ほか

02 LLMとのアイデアブレスト

良い質問の型と、深掘りの対話例

03 実装の優先順位付け

ユーザー価値・難易度・ROIで判断する

04 段階的な開発

MVP思考とフィードバックループ

PART 1

追加できる機能

ゲームをさらに面白くする6つのアイデア

FEATURE CATALOG

6つの拡張機能 — 何が作れるか

[分] マルチエンディング

選択で結末が分岐

本章で扱う設計ポイント・落とし穴・テスト観点を併記

[セ] セーブスロット

複数プレイを並行

本章で扱う設計ポイント・落とし穴・テスト観点を併記

[ラ] ランキング

最短クリア・最少手数

本章で扱う設計ポイント・落とし穴・テスト観点を併記

[シ] シナリオ共有

ユーザー独自シナリオ

本章で扱う設計ポイント・落とし穴・テスト観点を併記

[音] 音声読み上げ

Web Speech APIで朗読

本章で扱う設計ポイント・落とし穴・テスト観点を併記

[画] 画像生成

DALL-Eで場面を描く

本章で扱う設計ポイント・落とし穴・テスト観点を併記

FEATURE 01

マルチエンディング — 選択で結末が変わる

ゲーム状態に基づいて分岐させ、UXを「閉じる」

設計ポイント

01 分岐の根拠を「ゲーム状態」に

LLMの気分で揺れないよう、手がかり数や危険行動回数を
サーバー側でカウントして条件に使う

02 終了判定の形式を決める

[GOOD_ENDING] のようなタグ方式 or JSON構造化
（ending_typeを返させる）

03 到達時のUXを作る

クリア画面・再プレイ導線・達成エンディング一覧で
体験を「閉じる」と満足度が上がる

落とし穴

! タグの誤混入

LLMが説明文中でタグを出すと誤判定。
「最終行にだけ出す」など制約を強める

! 二重終了

終了済みセッションに再度終了処理が走ると
統計が壊れる → is_active でガード

テスト観点

*check_ending() は文字列解析の境界条件で落とし
やすい — ユニットテストで先に固める*

IMPLEMENTATION

マルチエンディング ― 4ステップで実装

モデル → プロンプト → 判定 → 一覧表示

STEP 1

モデルに ending_type を追加

```
ending_type =
models.CharField(
    max_length=50,
    blank=True, null=True
)
```

STEP 2

プロンプトに分岐条件

```
【エンディング分岐条件】
良: 全手がかり+危険回避
悪: 危険行動 3回+
秘: 隠しアイテム発見
```

STEP 3

判定 utils.check_ending

```
if '[GOOD_ENDING]'
   in response_text:
  session.ending_type
  = 'good_ending'
  session.is_active=False
```

STEP 4

達成一覧ページ

```
GameSession.objects
 .filter(
    ending_type__
    isnull=False)
 .values('ending_type')
 .distinct()
```

再プレイ導線とコレクション画面まで作ると体験が「閉じる」

FEATURE 02

セーブスロット複数対応 — 並行プレイを支える

1プレイ = 1 GameSession で扱うのがシンプル

設計ポイント

セーブの単位

1プレイ=1 GameSession に統一すると、
一覧・再開・集計がシンプル

識別子の扱い

Cookie保存は手軽だが推測リスクあり。
ログイン紐付けや署名付きCookieを検討

保持期間

max_age=30日 など期限を決め、
不要データのクリーンアップも設計に含める

落とし穴

URL直打ちで他人のセーブが見える

/game/load/12345/ を推測されると漏洩。
認可チェックを最優先で塞ぐ

複数タブの競合

同セーブで複数タブが進行 → どちらが
「勝ち」か決めないとUXが壊れる

実装スニペット

response.set_cookie

current_session_id を 30日で保存

get_object_or_404

load_game(session_id) でロード

save_slots_view

updated_at降順で最近10件を表示

テスト: client.get(...).cookies に current_session_id が入るか確認

FEATURE 03

ランキング機能 — 何を競うかを決める

計測はサーバー側で確定。クライアント申告は信用しない

設計と落とし穴

- **計測はサーバー側で**
  時間・行動回数はサーバーで計算・保存
- **対象の定義**
  good_ending のみ・期間別など条件を決める
- **同点の扱い**
  タイブレーク順を決めて表示の一貫性を担保
- **clear_time が NULL の並び**
  DBによってNULL扱いが違う → クリア済みのみ
- **件数増加時の負荷**
  並べ替え/集計は重い → インデックス・キャッシュ

ranking_view (views.py)

```
def ranking_view(request):
  fastest_clear = (
    GameSession.objects
    .filter(ending_type='good_ending')
    .order_by('clear_time')[:10]
  )
  return render(request,
    'ranking.html',
    {'fastest_clear': fastest_clear})
```

モデル拡張

```
clear_time = models.IntegerField(
    null=True, blank=True)  # 秒
action_count = models.IntegerField(
    default=0)

def calculate_clear_time(self): ...
```

FEATURE 04

シナリオ共有 — ユーザー生成コンテンツ

system_promptを自由入力にする際の安全配慮が必要

入力バリデーション

長さ制限・禁止語・HTMLの扱い（XSS）など
最低限の防御を入れる

プロンプト注入対策

system_prompt 自由入力は規約違反を誘発しやすい。
公開時はモデレーションや審査フロー

所有者と公開範囲

「自分だけ」「限定」「全体」の公開設定を持つと
運用しやすい

落とし穴

- **Scenario.objects.all() のまま公開**

  個人用シナリオも漏れる → 必ず公開範囲で絞り込む

- **人気順の偏り**

  play_count は初期露出が勝ちやすい → 新着枠や期間補正

Scenario model (models.py)

```
class Scenario(models.Model):
  title = models.CharField(max_length=200)
  description = models.TextField()
  system_prompt = models.TextField()
  created_by = models.CharField(max_length=100)
  created_at = models.DateTimeField(auto_now_add=True)
  play_count = models.IntegerField(default=0)
```

FEATURE 05 / 06

音声読み上げ ・ 画像生成 — UI拡張とAPI連携

ブラウザAPIと外部AIサービスの組み合わせ

FEATURE 05

音声読み上げ (Web Speech API)

- **対応状況**
  ブラウザ差分あり → 非対応ならボタン非表示
- **読み上げ対象**
  DOMセレクタがズレると壊れる → 構造で寄せる
- **キュー制御**
  speechSynthesis.cancel() で重なり防止を決める

```
const u = new SpeechSynthesisUtterance(text);
u.lang = 'ja-JP';
u.rate = 1.0;
speechSynthesis.speak(u);
```

FEATURE 06

画像生成 (DALL-E連携)

- **コスト管理**
  高コスト → 回数制限・キャッシュ・ジョブ化
- **保存先**
  S3等に保存して再利用できると運用が楽
- **コンテンツポリシー**
  公開サービスではモデレーション必須

```
response = openai.Image.create(
  prompt=f'Game scene: {scene}',
  n=1, size='512x512')
image_url = response['data'][0]['url']
```

PART 2

LLMとのアイデアブレスト

良い質問の型と、深掘りの対話例

PROMPT FRAMEWORK

良い質問の型 — ブレストの精度を上げる

「アイデア出し → 絞り → 実装案」を段階的に依頼する

目的

誰の体験を、どう良くしたいか

例: 初回ユーザーの離脱を減らす

制約

技術スタック・期限・コスト・形態

Django + Gemini API / 2週間以内

評価軸

価値・難易度・リスク

規約・セキュリティ・運用コストも含む

出力形式

箇条書き・表・タスク分割

「機能名・説明・実装時間・難易度」の表

TIP

「アイデア → 絞り込み → 実装案」を1往復で済ませず、段階を分けて依頼する方が結果が安定する

DIALOG SAMPLE

実際の対話例 — アイデア10個を引き出す

条件を明確にすると、カテゴリ整理された回答が返ってくる

あなた

質問プロンプト

```
テキストアドベンチャーゲームを
もっと面白くするアイデアを
10個提案してください。

条件:
- Django + Gemini APIの構成
- 技術的に実装可能
- ユーザー体験を向上させる
```

Gemini

カテゴリ別に12個の提案

ゲームプレイ

マルチエンディング・隠しアイテム・実績・難易度

ソーシャル

プレイ動画共有・コミュニティシナリオ・ランキング

UI/UX改善

音声読み上げ・タイピングアニメ・ダークモード

技術的拡張

画像生成 (DALL-E)・音楽生成 (Mubert)

PART 3

実装の優先順位付け

ROI で「何から作るか」を決める

PRIORITIZATION

ROI評価 — 価値 ÷ 難易度で並べる

リスク・運用コスト・可逆性も合わせて見る

評価の3軸

ユーザー価値

1〜5

技術的難易度

1〜5（低い方が良）

ROI

価値 ÷ 難易度

+ もう一段だけ現実に

- リスク（規約/法務/悪用）
- 運用コスト（監視・対応）
- 可逆性（戻せるか）

評価例 (ROI 高い順)

機能	価値	難易度	ROI	優先度
音声読み上げ	3	1	3.0	★★★★★
マルチエンディング	5	2	2.5	★★★★☆
セーブスロット複数	4	2	2.0	★★★☆☆
ランキング	3	2	1.5	★★☆☆☆
画像生成	4	3	1.33	★★☆☆☆
シナリオ共有	5	4	1.25	★☆☆☆☆

→ フェーズ1: 音声 + マルチエンディング / フェーズ2: セーブ + ランキング / フェーズ3: 画像 + シナリオ

PART 4

アジャイルとMVP思考

小さく作って、フィードバックで育てる

AGILE / MVP

「縦に切る」段階的開発

完璧を目指さず、80%でリリース → フィードバックで改善

ウォーターフォール (旧)

【6ヶ月後】 全機能を実装

↓

リリース

↓

ユーザー「これじゃない…」

↓

手遅れ

アジャイル (MVP思考)

- **MVP1 (1ヶ月)** 基本ゲーム
- **MVP2 (2ヶ月)** セーブ機能追加
- **MVP3 (3ヶ月)** 音声読み上げ追加
- **継続的な改善** → フィードバック → 改善

重要: 「縦に切る」

UIだけ / DBだけ / バックエンドだけを先に作って
「完成した気になる」のが最大の落とし穴

画面 → 処理 → データ → テスト を一貫させて積む

フィードバックの収集方法

- **Google Forms**
  アンケート作成
- **Discord/Slack**
  コミュニティ意見
- **Google Analytics**
  ユーザー行動分析
- **GitHub Issues**
  バグ報告・要望

CHAPTER 13 — まとめ

機能拡張のアイデア

01

追加できる機能

マルチエンディング・セーブ・ランキング・シナリオ共有・音声・画像生成

02

LLMとのアイデアブレスト

目的・制約・評価軸・出力形式 を明示し、段階を分けて依頼する

03

実装の優先順位付け

ROI = 価値 ÷ 難易度。リスク・運用・可逆性も併せて評価する

04

アジャイル / MVP思考

「縦に切る」小さなMVPを積み、フィードバックで継続的に改善する

完璧を目指さない / 小さく始めて、大きく育てる