04

CHAPTER

ゲームの企画と
データ設計

コードを書く前に「何を作るか」を明確にする — LLM を壁打ち相手に

本書のサンプル章

AGENDA

この章で学ぶこと

所要時間： 約 1 時間

01 **LLM とのブレストセッション**
ゼロからアイデアを広げて固める

02 **MVP（最小機能）の決定**
完璧を目指さず、まず動くものを作る

03 **データモデル設計**
GameSession と ChatMessage を定義

04 **Django モデルの実装**
models.py に書く

05 **マイグレーションの実行**
設計をデータベースに反映

06 **管理画面で確認 + SQLite の選定理由**
動くデータベースまで完成

PART 1

LLM とのブレスト

対話して、企画をゼロから形にする

WHY

なぜ LLM とブレストするのか

ANALOGY

プログラミング = 建築

いきなり釘を打ち始めない。
まず設計図を描く。

悪い例

とりあえずコードを書き始める
→ 途中で迷子になる

良い例

LLM と対話して企画を固める
→ 設計図ありで迷わない

ブレストのコツ 4 選

1. **段階的に質問する**
   大枠 → 詳細の順
2. **「初心者向け」を明示**
   シンプルさを強調
3. **選択肢を提案してもらう**
   「いくつか提案して」
4. **具体的な例を求める**
   コード例まで聞く

FLOW

ブレストの 3 段階

01

IDEATION

アイデア発散

Q 「どんなゲームにする？」

A LLM が 4 つの候補を提示
→ 廃墟研究所 / 魔法学校 / 宇宙船 / 時間旅行

02

MECHANICS

メカニクス深掘り

Q 「初心者向けに、何を含める？」

A コア要素 / 追加要素 / 避けるべき複雑さ
を 3 つに分類

03

NARRATIVE

ストーリー骨格

Q 「3 幕構成で考えたい」

A 目覚め → 探索 → 脱出
複数エンディングも提案

STORY

3 幕構成 — 廃墟研究所からの脱出

ACT 1

目覚め

INTRODUCTION

- 薄暗い部屋で目覚める
- 記憶が曖昧、ここがどこか不明
- 部屋を調べて最初の手がかり
- 扉を開けて廊下へ

ゴール：施設の構造を理解し始める

ACT 2

探索

DEVELOPMENT

- 複数の部屋を探索
- 研究日誌から過去を知る
- 謎の事件が判明
- パズルや謎を解く

ゴール：脱出方法の手がかりを集める

ACT 3

脱出

CLIMAX

- 脱出口を発見、しかし鍵が
- 最後の謎を解く
- 緊迫した状況
- 無事に脱出 / 失敗

ゴール：無事に脱出（複数エンディング）

PART 2

MVP の決定

完璧主義を捨て、まず動くものを作る

MVP

Minimum Viable Product

最小限の機能で動く製品 — 「恥ずかしいくらいシンプル」で OK

完璧主義

「全部入り」を目指す

- \+ 美しい UI
- \+ BGM ・効果音
- \+ 複数エンディング
- \+ アイテムシステム
- \+ キャラクター画像
- \+ セーブ機能
- \+ ユーザー認証
- \+ …

→ 3 ヶ月経っても完成しない

MVP

シンプルに始める

- ✓ シンプルな入力欄
- ✓ AI の応答表示
- ✓ 会話履歴

→ 1 週間で動くものができる
→ 遊べる！次の機能を追加したくなる

ROADMAP

機能を 3 つの Phase に分ける

PHASE 1

MVP（本書で実装）

機能	優先度 / 難易度
チャット UI	*高 / 低*
Gemini API 連携	*高 / 中*
会話履歴の表示・保存	*高 / 中*
ゲームリセット	*高 / 低*
基本プロンプト設計	*高 / 中*

PHASE 2

拡張機能（挑戦したい人へ）

機能	優先度 / 難易度
複数の選択肢提示	*中 / 低*
アイテムシステム	*中 / 高*
ゲームオーバー判定	*中 / 中*

PHASE 3

さらなる発展

機能	優先度 / 難易度
複数エンディング	*低 / 高*
BGM/ 効果音	*低 / 中*
ユーザー認証	*低 / 中*
マルチプレイヤー	*低 / 超高*

USER STORIES

MVP で実現したい 3 つの体験

01 初回プレイ

1. サイトにアクセス
2. ゲーム開始のメッセージ
3. AI が最初の状況を生成
4. プレイヤーが行動を入力
5. AI が応答を生成・表示

02 継続プレイ

1. 以前プレイしたユーザーが再訪
2. 前回の会話履歴が表示される
3. 続きから再開できる

03 リセット

1. 「最初からやり直したい」
2. リセットボタンをクリック
3. 会話履歴がクリア
4. 新しいゲームが開始

本書で使う LLM

Gemini API

API がシンプル / 日本語に強い / Google 提供で安定。次章で API キー取得から実装まで。

PART 3

データモデル設計

アプリが扱うデータの構造を、最初に固める

WHY DESIGN FIRST

設計を最初に行う 3 つの利点

01

開発がスムーズ

何を実装すべきかが明確。
迷わずコードが書ける。

02

後から変更しやすい

全体像を把握しているので、
変更の影響範囲が分かる。

03

バグが減る

データ関係性が明確なので、
整合性のないデータが作られない。

今回ゲームに必要なデータは？

✓ ゲームセッション (いつ始まったか / 進行中か終了か)

✓ 会話メッセージ (誰が・何を・いつ発言したか)

✗ *不要 (今は) : ユーザーアカウント / アイテム / スコア*

ENTITY 1

GameSession — 1 つのゲームプレイ

FIELD	TYPE	DESCRIPTION	EXAMPLE
id	*Integer*	主キー（自動採番）	1
session_id	*String*	セッション識別子（ランダム文字列）	abc123…
created_at	*DateTime*	作成日時 — いつプレイしたか	2025-01-30 10:00:00
updated_at	*DateTime*	更新日時 — 放置判定に使う	2025-01-30 10:15:30
is_active	*Boolean*	ゲーム進行中か（ True / False ）	True

ENTITY 2

ChatMessage — プレイヤーと AI の会話履歴

FIELD	TYPE	DESCRIPTION
`id`	*Integer*	主キー
`session`	*ForeignKey*	どの GameSession か（ CASCADE ）
`role`	*String*	発言者 : user / assistant / system
`content`	*Text*	発言内容（長文可）
`timestamp`	*DateTime*	発言日時

RELATIONSHIP

1 対多 （1 : N）

GameSession

1 つのゲーム

has many ▼

ChatMessage

複数の発言

on_delete=CASCADE
→ 親が消えると子も消える

models.py

Django で実装

class GameSession

```
from django.db import models

class GameSession(models.Model):
    """ゲームセッション"""
    session_id = models.CharField(
        max_length=100,
        unique=True,
    )
    created_at = models.DateTimeField(
        auto_now_add=True,
    )
    updated_at = models.DateTimeField(
        auto_now=True,
    )
    is_active = models.BooleanField(
        default=True,
    )

    class Meta:
        ordering = ['-created_at']
```

class ChatMessage

```
class ChatMessage(models.Model):
    """チャットメッセージ"""
    ROLE_CHOICES = [
        ('user',      'ユーザー'),
        ('assistant', 'アシスタント'),
        ('system',    'システム'),
    ]

    session = models.ForeignKey(
        GameSession,
        on_delete=models.CASCADE,
        related_name='messages',
    )
    role    = models.CharField(
        max_length=10, choices=ROLE_CHOICES)
    content = models.TextField()
    timestamp = models.DateTimeField(
        auto_now_add=True)
```

MIGRATION

「設計図 → 手順書 → 建物」の 3 ステップ

01

DESIGN

設計図を描く

ACTION

```
models.py を書く
```

OUTPUT

```
class GameSession(models.Model): ...
```

= 建築の設計図

02

PLAN

手順書を作る

ACTION

```
$ uv run python manage.py
makemigrations
```

OUTPUT

```
→ migrations/0001_initial.py
```

= 工事の手順書

03

BUILD

実際に建てる

ACTION

```
$ uv run python manage.py migrate
```

OUTPUT

```
→ db.sqlite3 にテーブル作成
```

= 実際に建物を建てる

なぜ本書は SQLite で十分なのか

本書で SQLite を選んだ理由

1. **環境構築の簡素化**
   RDS 不要 / EC2 1 台で完結
2. **初心者に優しい**
   接続エラーに悩まない / ファイル 1 つ
3. **コスト**
   AWS 無料枠内で完結（RDS 不要）
4. **学習目的には十分**
   同時アクセス少 + データ量小

SQLite が適している規模

同時ユーザー	~ 100 人
データ量	~ 数 GB
書き込み頻度	低 ~ 中
構成	単一サーバー

PostgreSQL/RDS 移行のサイン

- 100 人以上の同時接続
- 「 database is locked 」が頻発
- 本格サービスとして公開する時

DONE

設計完了。次は LLM を命に変える。

この章で学んだこと

- ✓ LLM を使った企画のブレスト
- ✓ MVP（最小機能）の定義
- ✓ データモデルの設計
- ✓ Django モデルの実装
- ✓ マイグレーションの実行
- ✓ 管理画面でデータ確認
- ✓ SQLite の利点と限界

NEXT

第 5 章

Gemini API との連携

- ▸ Gemini API キーの取得
- ▸ プロンプトエンジニアリング
- ▸ 実際に物語を生成する

→ ゲームに命を吹き込もう

設計段階で LLM を活用すると、ベストプラクティスを学べる。