07

CHAPTER

ゲームロジックを改善する

「単なるチャットボット」から「体験型ゲーム」へ進化させる

本書のサンプル章

AGENDA

この章で学ぶこと

01 プロンプトの改善
LLM と対話してプロンプトを洗練させる

02 特殊コマンドの実装
/restart /hint /status /help

03 ゲームオーバー判定
キーワード + 行動回数で終了判定

04 ゲームクリア判定
脱出キーワードで成功検出

05 Django Debug Toolbar
SQL クエリ・テンプレ・実行時間を可視化

06 ログ出力 + LLM 相談
logger.info + エラー解決ループ

PROMPT

LLM と相談してプロンプトを改善

YOU

現状の課題を伝える

システムプロンプトを
改善したいです。

現状の問題点：
1. 同じ行動の繰り返しに
同じ応答が返る

2. ヒントが直接的すぎて
すぐに答えがわかる

3. ゲームの緊張感が
足りない

改善案を提案してください。

GEMINI

3 つの改善案

1 **行動回数をカウント**

3 回繰り返したら新しいヒント

2 **ヒントの段階的提供**

曖昧 → 具体的 → 直接的

3 **ゲームのテンション管理**

時間経過で危機感を演出

→ システムプロンプトの修正案を送ります

IMPROVED PROMPT

改善された GAME_SYSTEM_PROMPT

game/prompts.py — GAME_SYSTEM_PROMPT（抜粋）

```
【ヒントの出し方】
- 初回： 曖昧な描写
  「何か気になるものがある」
- 2 回目： やや具体的
  「机の引き出しが少し開いている」
- 3 回目： 直接的
  「机の引き出しには古びた鍵がある」

【テンション管理】
- 行動回数 10 回超え： 危機感を演出
  （足音、物音など）
- ゲームオーバー条件：
  危険な行動を 3 回繰り返す
  時間切れ（30 ターン）

- ゲームクリア条件：
  研究所の出口を見つけて脱出する
```

3 つの改善ポイント

ヒントの段階化

プレイヤーが詰まっても
楽しさを保てる

テンション演出

危機感で没入感 UP

終了条件の明示

ゲームの締まりを作る

PART 2

特殊コマンド & ゲーム判定

「 / コマンド」と終了判定でゲームらしさを足す

COMMANDS

4 つの特殊コマンド

/restart

リセット

ゲームを最初から
やり直す

実装

全 ChatMessage を削除

/hint

ヒント

現在の状況での
ヒントを表示

実装

前回の AI 応答を参考に

/status

状況確認

行動回数と
進行状況

実装

セッション情報も

/help

ヘルプ

使用可能な
コマンド一覧

実装

SPECIAL_COMMANDS から

utils.py

判定 + 処理を分離する

is_special_command

判定

```
SPECIAL_COMMANDS = {
  '/restart': 'ゲームをリセット',
  '/hint':    'ヒントを表示',
  '/status':  '進行状況',
  '/help':    'コマンド一覧',
}

def is_special_command(
        message):
    message = message\
        .strip().lower()
    return message in \
        SPECIAL_COMMANDS
```

handle_special_command

処理

```
def handle_special_command(
        command, game_session):
    cmd = command\
        .strip().lower()

    if cmd == '/restart':
        ChatMessage.objects\
          .filter(
            session=game_session\
          ).delete()
        return "リセット完了"

    elif cmd == '/help':
        return help_text
    # ... 他のコマンド
```

PART 3

ゲーム状態管理

ゲームオーバーとクリアで「終わり」を作る

GAME OVER

ゲームオーバー判定の 3 条件

01

危険な行動を 3 回繰り返す

AI 応答内のキーワードで判定

```
['死んでしまった', 'ゲームオーバー',
 '力尽きた', '命を落とした']
```

02

行動回数 30 回オーバー

ChatMessage を user で count

```
→「時間切れ。研究所に
  閉じ込められました...」
```

03

GAMEOVER キーワード

AI が明示的に終了を宣言

```
is_active = False で
セッション無効化
```

check_game_over & check_game_clear

check_game_over

終了判定

```
def check_game_over(
        response_text, gs):

    #  キーワードチェック
    keywords = [
        '死んでしまった',
        'ゲームオーバー',
        '力尽きた',
        '命を落とした']

    if any(k in response_text
           for k in keywords):
        return (True, "…")

    #  行動回数チェック
    if total_actions >= 30:
        return (True, "時間切れ")

    return (False, "")
```

check_game_clear

クリア判定

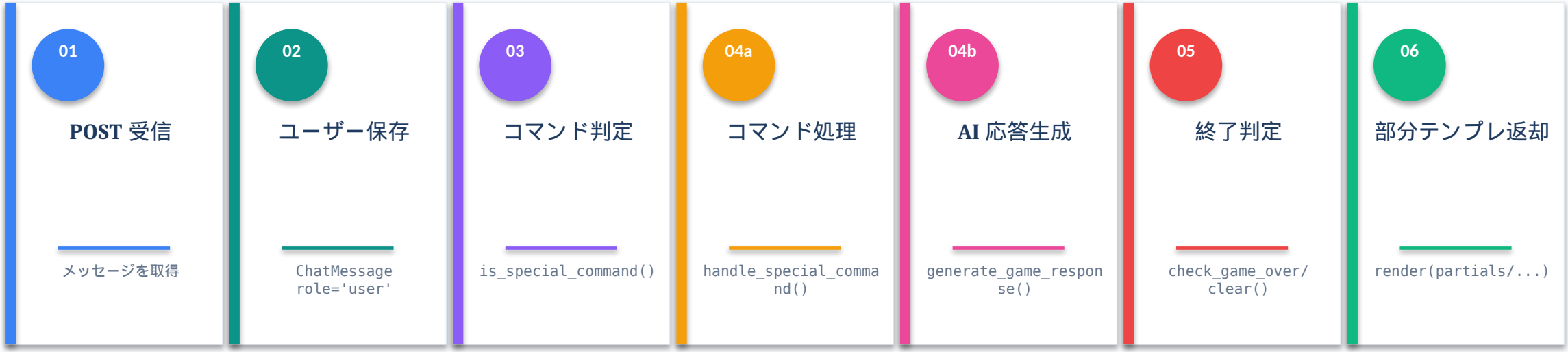
```
def check_game_clear(
        response_text):

    #  クリアキーワード
    keywords = [
        '脱出成功',
        '外の世界',
        'ゲームクリア',
        '研究所を出た',
        '自由になった',
        '無事に脱出',
        '脱出に成功']

    return any(
        k in response_text
        for k in keywords)
```

FLOW

game_action の処理フロー



STEP 03 で分岐

```
if is_special_command(message):
→ 04a: handle_special_command
→ role='system' で保存
```

```
else:
→ 04b: generate_game_response
→ role='assistant' + 終了判定
```

PART 4

デバッグの実践

Debug Toolbar + ログ + LLM 相談で詰まりを解消

django-debug-toolbar を導入する

01 INSTALL

uv でインストール

```
$ uv add django-debug-
toolbar
```

02 INSTALLED_APPS

settings.py に追加

```
'debug_toolbar',
```

03 MIDDLEWARE

ミドルウェア最上部に

```
'debug_toolbar.middlew
are
  .DebugToolbarMiddlew
are',
```

04 INTERNAL_IPS

開発環境のみ許可

```
INTERNAL_IPS =
['127.0.0.1']
```

05 URLs

server/urls.py に追加

```
if settings.DEBUG:
    urlpatterns = [

path('__debug__/',
        include(

debug_toolbar.urls)),
    ] + urlpatterns
```

PANELS

Debug Toolbar の主要パネル

SQL

SQL クエリ一覧

実行時間・実行計画
N+1 問題の発見

Templates

使用テンプレ

渡されたコンテキスト
変数を確認

Time

実行時間

ページ読込の内訳
SQL/render 時間

Settings

Django 設定

settings.py の値
環境変数も

Headers

HTTP ヘッダ

リクエスト / レスポンス
両方確認

Logging

ログメッセージ

アプリケーションの
ログ出力を表示

TIP: SQL パネルで同じクエリが多数 → N+1 問題。 select_related() / prefetch_related() で最適化。

ロギング — 重要な処理を記録する

server/settings.py — LOGGING

```
LOGGING = {
  'version': 1,
  'formatters': {
    'verbose': {
      'format':
        '{levelname} {asctime} {module} {message}',
    },
  },
  'handlers': {
    'console': {'class': 'logging.StreamHandler',
                'formatter': 'verbose'},
    'file': {'class': 'logging.FileHandler',
             'filename': BASE_DIR / 'debug.log',
             'formatter': 'verbose'},
  },
  'loggers': {
    'game': {                # アプリ名
      'handlers': ['console','file'],
      'level': 'INFO',
    },
  },
}
```

views.py で使う

```
import logging

logger = logging\
    .getLogger(__name__)

logger.info(
  f"msg={message=}")
```

ログレベル

DEBUG	詳細
INFO	通常
WARNING	警告
ERROR	エラー

LLM DEBUG

エラーが出たら LLM に貼って聞く

「分かるまで悩む」より「数秒で抜ける」

YOU

エラーを貼り付ける

```
Django でエラーが発生しました。
原因と解決方法を教えてください。

【エラーメッセージ】
AttributeError:
'GameSession' object has
no attribute 'messages'

【関連コード】
messages = session\
    .messages.all()\
    .order_by('timestamp')
```

GEMINI

原因と解決を即提示

```
ChatMessage の session
フィールドに related_name
が定義されていません。

class ChatMessage(...):
  session = models.ForeignKey(
    GameSession,
    on_delete=models.CASCADE,
    related_name='messages'
  )

修正後、マイグレーション：
$ uv run manage.py\
    makemigrations && migrate
```

BEST PRACTICES

プロンプトエンジニアリングの 4 原則

01

明確な役割を与える

「ゲームマスターです」と
具体的に

```
「あなたは GM です。
臨場感ある描写を返して」
```

02

具体的な制約を設ける

数字で指示する

```
「 150 文字以内で応答」
```

03

Few-Shot Learning

良い応答例 + 悪い例を
プロンプトに含める

```
良い例： 鍵が入っている
悪い例： 何もない
```

04

CoT (Chain of Thought)

段階的に考えさせる

```
1. 危険か？
2. 警告すべきか？
3. 代替案は？
```

DONE

ゲームらしさを獲得。次はデプロイへ。

この章で学んだこと

- ✓ プロンプトを LLM と相談して改善
- ✓ ヒント段階化 + テンション管理
- ✓ 特殊コマンド（/restart /hint /status /help）
- ✓ ゲームオーバー判定（3 条件）
- ✓ ゲームクリア判定（キーワード）
- ✓ Django Debug Toolbar 導入
- ✓ ロギング（LOGGING + logger.info）

NEXT

第 8 章

AWS の設定

- ▸ AWS アカウント準備
- ▸ デプロイのための設定
- ▸ クラウドへの第一歩

→ いよいよ世界に公開

プロンプトは「一度書いて終わり」ではない。継続的に改善する。