**プロジェクト「シャドウバース対戦支援AI」：技術選定とシステム設計**

このドキュメントは、要件定義で定められた機能を実現するための具体的な技術スタックと、システム全体のアーキテクチャを定義するものです。

**第1部：技術選定**

各コンポーネントの要件を満たし、開発効率とパフォーマンスのバランスを考慮した技術スタックを以下のように選定します。

|  |  |  |
| --- | --- | --- |
| **カテゴリ** | **推奨技術** | **選定理由** |
| **プログラミング言語** | **Python 3.x** | AI・機械学習分野のライブラリ（PyTorch, OpenCV等）が最も充実しており、迅速なプロトタイピングから本格的な開発まで対応可能。コミュニティも活発で情報収集が容易。 |
| **画像認識** | **OpenCV + Tesseract OCR** | **OpenCV:** スクリーンショットからの盤面や手札のカード位置特定（テンプレートマッチング）、カード画像の切り出しなど、高度な画像処理に必須。 **Tesseract OCR:** Google製の高精度なOCRエンジン。カード名、攻撃力、体力、コストといったテキスト情報を画像から読み取るために利用。 |
| **AIフレームワーク** | **PyTorch** | 研究分野で主流であり、柔軟で直感的なモデル構築が可能。動的な計算グラフは、状況が刻々と変化するカードゲームのモデリングに適している。「AlphaZero」のような強化学習アルゴリズムの実装例も豊富。 |
| **AI推論高速化** | **ONNX Runtime** | PyTorchで学習させたモデルをONNX形式に変換し、C++等で最適化されたONNX Runtime上で推論を実行。Pythonの柔軟性を保ちつつ、リアルタイム性が求められる推論部分のパフォーマンスを大幅に向上させることが可能。 |
| **UI/オーバーレイ** | **PyQt6 / PySide6** | クロスプラットフォーム（Windows/Mac）に対応した高機能なGUIフレームワーク。半透明ウィンドウの作成や、ゲーム画面への正確なオーバーレイ表示といった、本プロジェクトのUI要件を満たす上で最も強力な選択肢。 |

**第2部：システム設計**

**2.1. システムアーキテクチャ**

本AIシステムは、以下の5つの主要モジュールから構成されます。各モジュールは連携して動作し、リアルタイムでの対戦支援を実現します。

コード スニペット

graph TD

subgraph "ユーザーPC"

A[シャドウバース ゲーム画面] -->|定期的にキャプチャ| B(1. 画面キャプチャモジュール);

B --> C(2. 画像認識モジュール);

C -->|ゲーム状態データ (JSON)| D(3. AI推論エンジン);

D -->|推奨手データ| E(4. UI/オーバーレイモジュール);

E -->|画面に描画| A;

F(5. メインコントローラー) --> B;

F --> C;

F --> D;

F --> E;

end

subgraph "AIモデル (PyTorch/ONNX)"

D -- 推論実行 --> G[盤面評価/方策ネットワーク];

end

style A fill:#f9f,stroke:#333,stroke-width:2px

style E fill:#ccf,stroke:#333,stroke-width:2px

* **1. 画面キャプチャモジュール (Screen Capture Module):**
  + 役割：PC上で実行されているシャドウバースのゲーム画面を、定期的に（例：0.5秒ごとに）スクリーンショットとして取得します。
* **2. 画像認識モジュール (Image Recognition Module):**
  + 役割：取得したスクリーンショットを解析し、ゲームの状態を構造化データ（JSON形式など）に変換します。
  + 処理内容：
    - OpenCVによるテンプレートマッチングで、手札、場のフォロワー、リーダー、PPなどの位置を特定。
    - 特定した領域の画像を切り出し、Tesseract OCRでカード名や数値をテキストデータ化。
* **3. AI推論エンジン (AI Inference Engine):**
  + 役割：画像認識モジュールから受け取ったゲーム状態データに基づき、「次善手」を計算します。
  + 内部アルゴリズム：モンテカルロ木探索（MCTS）と、事前に学習させたニューラルネットワーク（盤面の価値を評価する「価値ネットワーク」と、有望な手を予測する「方策ネットワーク」）を組み合わせ、高速かつ高精度な探索を行います。
* **4. UI/オーバーレイモジュール (UI/Overlay Module):**
  + 役割：AI推論エンジンが算出した結果（推奨カード、攻撃対象など）を、ゲーム画面上に視覚的に表示します。
  + 実装：PyQtを使い、背景が透明なウィンドウを作成。その上に矢印やハイライトを描画し、ゲーム画面に重ねて表示します。
* **5. メインコントローラー (Main Controller):**
  + 役割：上記すべてのモジュールを統括し、全体の処理フローを制御します。ユーザーによる設定（AIのON/OFFなど）の受付も担当します。

**2.2. 処理フロー**

ユーザーがゲームをプレイしている間の、AIの内部的な処理フローは以下の通りです。

1. **起動:** ユーザーがAI支援ツールを起動。メインコントローラーが各モジュールを初期化し、オーバーレイウィンドウを準備。
2. **ループ開始:** メインコントローラーが一定間隔（例：0.5秒）で処理ループを実行。
3. **画面取得:** 画面キャプチャモジュールがゲーム画面のスクリーンショットを取得。
4. **盤面解析:** 画像認識モジュールがスクリーンショットを解析。
   * 前回の盤面データと比較し、変化がなければ以降の処理をスキップして次のループへ（CPU負荷軽減）。
   * 盤面に変化があれば、現在のゲーム状態（手札、PP、盤面、体力など）をJSONデータとして生成。
5. **AI推論:** 生成されたゲーム状態データをAI推論エンジンに渡す。
   * AIエンジンは、制限時間内（例：3〜5秒）でモンテカルロ木探索を実行し、最も評価値の高い行動プランを決定する。
6. **結果描画:** AIの計算結果（例：{ "action": "play", "card": "楽園の双翼・ガルラ", "target": "enemy\_follower\_3" }）をUI/オーバーレイモジュールに渡す。
7. **画面表示:** UI/オーバーレイモジュールは、受け取った指示に基づき、ゲーム画面上の「楽園の双翼・ガルラ」のカードをハイライトし、攻撃対象のフォロワーへ矢印を描画する。
8. **ループ継続:** 次のループへ戻り、2からの処理を繰り返す。

**第3部：推奨される開発ステップ**

本プロジェクトは大規模になるため、以下のステップで段階的に開発を進めることを推奨します。

* **Step 1: 基盤構築と画像認識の確立**
  + **ゴール:** ゲーム画面から盤面情報を正確にデータ化する。
  + タスク：画面キャプチャ、カード位置の特定、OCRによるカード名・数値の読み取り機能を実装する。まずは特定のクラス・デッキに絞って開発する。
* **Step 2: ルールベースAIとUIのプロトタイピング**
  + **ゴール:** AIの提案を画面に表示するまでの一連の流れを作る。
  + タスク：単純なルールベース（例：「コスト通りにカードを出す」「有利トレードを優先する」）のAIロジックを実装し、UI/オーバーレイモジュールと連携させて画面に表示する。
* **Step 3: 機械学習モデルの学習と統合**
  + **ゴール:** AIの思考能力を飛躍的に向上させる。
  + タスク：トッププレイヤーのリプレイデータ等を収集・整形し、PyTorchで価値/方策ネットワークを学習させる。学習済みモデルをAI推論エンジンに組み込む。
* **Step 4: 機能拡張と最適化**
  + **ゴール:** 総合支援ツールとしての完成度を高める。
  + タスク：リプレイ分析機能やデッキ構築支援機能を追加。ONNX Runtimeを導入し、パフォーマンスの最適化を行う。

以上の技術選定とシステム設計に基づき、開発を進めることで、要件定義に沿った高機能な対戦支援AIの実現が可能となります。