Abigail

～ AI Desktop Mascot ～

研究者：マエブチ タカオ

１　はじめに

1.1 Abigail とは

Abigail は、静止画像を単一の操作でアニメーション化されたアバターへ変換し、さらにプログラム実行、リマインダー設定、ファイル出力などの機能を統合した AI 搭載デスクトップマスコットである。

1.2 なぜ作ったか/目的

学習や読書の際、画像や PDF などコピー不可のテキストを保存したり、詳細情報を取得したい場面が多々あった。そういったテキストはスクリーンショットを撮って AI/OCR にアップロードするか、重要な部分を自分で再入力する必要がありましたが、時間的負担が大きく、効率性に欠けていた。そこで、デスクトップアシスタントがあれば便利だろうと考えました。しかし、それらのほとんどは無料で使える機能が限られていたり、デザインが味気なかったり、他にも便利だと思った機能がいくつか統合されていなかったりしました。そこで、Abigail のようなプロジェクトを思いつきました。また、 大学に出願するために、クリエイティブなものが欲しかったです。

1.3 使用ソフトウェア

- vim
- Visual Studio 2019
- Python
- animateddrawings + rokoko
- Docker
- Anaconda

２　研究の内容

2.1 クライアント/リクエスト処理

Abigail のクライアントは完全に Python で記述されており、 Tkinter を使用して UI と、 クライアントから送信された JSON リクエストまたはサーバーで受信された JSON リクエストの送信、 処理、 シリアル化を行う requests ライブラリを設計しています。
サーバーは、3 種類の異なるリクエストを処理するために 2 つの異なる Docker コンテナを使用します。 画像が埋め込まれた OCR プロンプトリクエストと標準テキストプロンプトリクエストは、 明示的に Docker コンテナ 1 にルーティングされます。
OCR 処理では、画像が埋め込まれたリクエストは TesseractOCR と補助的なエッジケース処理 Python スクリプトを使用して処理され、 標準テキストプロンプト処理では、 API をプロンプトし、解析して最初のキーシグナリングテキストを含む JSON レスポンスを返す Flask スクリプトが使用されます。 Docker コンテナ 1 へのすべてのリクエストで使用されるキー。 画像埋め込み GIF 生成リクエストは Docker コンテナ 2 にのみルーティングされます。 これらのリクエストは GIF 処理され、 JSON リクエストの最初のキー値と埋め込み画像チェックによって区別された後、 Docker コンテナ 2 に転送されます。 その後、 埋め込み画像は facebookresearch の AnimatedDrawing に転送され、 Rokoko で事前に作成され手動で修正された 5 つのプリセット CSV ファイルと BVH ファイルに基づいて 5 つの異なる GIF が作成されます。 クライアントはこれらの GIF を受信し、自動ダウンロードして新しい Tkinter フレームとして設定します。

2.2　サーバー

2 台の個人用ラップトップを交互にサーバーとして使用しました。1 台目は Arch Linux、 もう 1 台は OpenBSD で、それぞれ Hyprland (Wayland) と i3 (x11) を使用しています。 サーバーは、2 つの Docker コンテナ(詳細は **2.1**)、 仮想 Anaconda 環境、 ngrok インスタンス、 Flask スクリプト、 Tailscale インスタンス、そして VNC および SSH サーバー（どちらもパスワード保護）を同時に実行しています。 animateddrawings をサポートするため、 Mesa OpenGL 3.3 ドライバーをインストールし、 Wayland 使用時には WLR_NO_HARDWARE_CURSORS を 1 にする必要がありました。

図 1　　学校のコンピュータを VNC 経由でサーバーに接続

図 2　　Tkinter で作った UI

3　研究過程と成果

4 月

・ どのようなコア機能を追加するかを計画
・ 各部分がどのように相互作用するかを計画
・ 相互作用をどのように保護するかを計画

6 月

・ 最初の UI プロトタイプを作成
・ LAN サーバーに接続、API 呼び出しを実装
・ LAN サーバーに OCR サポートを追加
・ ngrok トンネリングを介して WAN へ移行
・ テキスト送受信 JSON リクエスト処理を実装

7 月

・ animateddrawings サポートを実装
・ docker コンテナ化を実装
・ 画像埋め込み JSON リクエスト処理を実装
・ 追加機能を実装 （クイズ、タグ処理コマンド）

10 月

・ Abigail 固有の Anaconda 仮想環境の setup
・ docker コンテナ化 + anaconda 仮想環境の相乗
・ 標準的な rokoko の bvh ファイル生成
・ bvh ファイルと animateddrawings の互換性実装
・ キャラクターの作成

(4) 研究成果

4.1 キャラクター作成の成果

Abigail のキャラクターは私が作ったシンプルなキャラクターで、AI に改良を依頼しました。アニメーションもすべて animateddrawings で作成しました。ただし、サーバーの処理能力が低かったため、同じ BVH ファイルの使用は避けました。

図 4

4.2　AI アニメーション化の成果

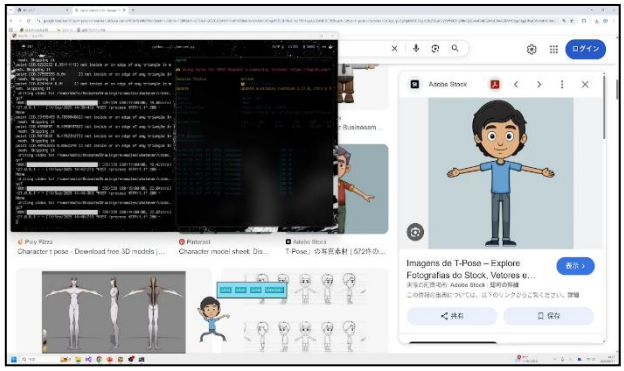

図5

(5) 苦労した点 ・ 問題点

技術的には、複数のリクエスト タイプを処理し、Wayland と X11 の両方ですべてを動作させ、Rokoko の BVH ファイルを他のものと連携するように変更する方法を見つけるのに苦労しました。

しかし、全体的な最大のハードルの一つは、後で拡張できるように作ることだった。最初は大変だったが、animateddrawings と LAN Abigail ミラーリングのサポートを実装しなければならなくなった時に、その苦労が報われたと感じた。

(6) まとめ:

大変な作業でしたが、チームで作業する理由を直接体験し、抽象度の高いプロジェクトが舞台裏でどのように機能するかについて多くのことを学びました。これは私にとって初めての中規模プロジェクトでしたが、着手する前に、Hyprlandの開発者がHyprlandの開発時に直面した問題について書いたブログ記事「It's not Awesome, it's Hypr!」を読んだことを思い出しました。彼は、コードを書く前に何をするかを事前に計画し、読みやすいコードにすることを重視していました。私はその点を念頭に置いてプロジェクトに着手しましたが、プロジェクトの途中で突然生産性が低下したのを覚えています。コード全体をクリーンアップして、できるだけ摩擦をなくすことに決め、ようやく再び前進し始めました。

ようやく、オッカムの剃刀/KISS 原則がコンピューターサイエンスで頻繁に登場する理由が分かりました。