Development of a Configuration GUI for MotionInput:

Enhancing Gaming Accessibility Through Visual Profile Management

Yuma Noguchi

A dissertation submitted in partial fulfillment
of the requirements for the degree of
BSc Computer Science

Department of Computer Science
University College London

February 28, 2025

Abstract

This project presents the development of a Configuration Graphical User Interface (GUI) for MotionInput, an accessibility software that enables alternative methods of computer interaction. The GUI aims to enhance the gaming experience for users with physical disabilities by providing an intuitive interface for managing visual profiles and mapping configurations.

The implementation utilizes the WinUI 3 framework to create a modern, native Windows application that integrates seamlessly with the existing MotionInput ecosystem. The GUI allows users to create, modify, and manage visual profiles that define how physical movements are translated into game controls, making games more accessible to players with diverse physical abilities.

Key features include a visual profile editor, real-time configuration preview, profile management system, and integration with various input devices. The project emphasizes user-centered design principles and accessibility guidelines throughout its development process.

This dissertation details the research, design, implementation, and evaluation of the Configuration GUI, demonstrating its effectiveness in improving the accessibility of computer games for users with physical disabilities.

Declaration

I, Yuma Noguchi, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Signed: ________________________

Date: ________________________

Acknowledgements

I would like to express my sincere gratitude to my project supervisor, Dr. Dean Mohamedally, for his invaluable guidance, support, and feedback throughout this project. His expertise and insights have been instrumental in shaping both the direction and outcome of this work.

I am deeply grateful to MotionInput team for their technical support and for providing me with the opportunity to contribute to such a meaningful project that makes a real difference in people's lives.

Special thanks to the UCL Computer Science department for providing the resources and environment that made this project possible.

Finally, I would like to thank my family and friends for their unwavering support and encouragement throughout my academic journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

MotionInput is an innovative accessibility software that enables users with physical disabilities to interact with computers and games through alternative input methods. While the core functionality of MotionInput has proven valuable, the need for an intuitive and user-friendly configuration interface has become increasingly apparent.

1.2 Problem Statement

The existing configuration process for MotionInput requires manual editing of configuration files, which presents a significant barrier for users, especially those with limited technical expertise. This complexity can deter potential users and limit the software's accessibility benefits.

1.3 Project Objectives

The primary objectives of this project are:

- To develop a graphical user interface for managing MotionInput configurations
- To simplify the process of creating and modifying visual profiles
- To provide real-time preview and testing capabilities
- To ensure the interface itself adheres to accessibility guidelines
- To integrate seamlessly with the existing MotionInput ecosystem

1.4 Technical Approach

The project utilizes the WinUI 3 framework to create a modern, native Windows application. This choice ensures optimal performance, system integration, and accessibility features while maintaining consistency with Windows' design language.

1.5 Contributions

The key contributions of this project include:

- A user-friendly interface for managing MotionInput configurations
- An intuitive visual profile editor
- Real-time configuration preview capabilities
- Enhanced profile management features
- Comprehensive documentation and user guides

1.6 Dissertation Structure

The remainder of this dissertation is organized as follows:

- Chapter 2 reviews relevant background literature and existing solutions
- Chapter 3 details the requirements gathering and analysis process
- Chapter 4 describes the implementation of the Configuration GUI
- Chapter 5 presents testing and evaluation results
- Chapter 6 concludes with reflections and future work recommendations

Chapter 2

Background and Literature Review

2.1 Gaming Accessibility

2.1.1 Current State of Gaming Accessibility

Gaming accessibility has become increasingly important as video games become a more integral part of society. However, many games remain inaccessible to players with physical disabilities due to input method limitations.

2.1.2 Existing Accessibility Solutions

Various hardware and software solutions have been developed to address gaming accessibility:

- Adaptive controllers (e.g., Xbox Adaptive Controller)
- Motion control systems
- Voice command interfaces
- Eye-tracking technologies

2.2 Artificial Intelligence in Accessibility

2.2.1 Machine Learning Models

Overview of AI technologies used in accessibility:

- Computer vision models for pose estimation
- Real-time gesture recognition
- Movement prediction algorithms
- Transfer learning applications

2.2.2 ONNX Runtime

Detailed examination of ONNX (Open Neural Network Exchange):

- Cross-platform model interoperability
- Performance optimization capabilities
- Integration with Windows applications
- Hardware acceleration features

2.3 Stable Diffusion and Visual Generation

2.3.1 Image Generation Technology

Analysis of Stable Diffusion and its applications:

- Text-to-image generation
- Image modification and inpainting
- Style transfer capabilities
- Real-time processing considerations

2.3.2 Applications in User Interface Design

How generative AI enhances UI development:

- Dynamic asset generation
- Custom icon creation
- Visual feedback generation
- Accessibility enhancement through visual aids

2.4 Motion Input Technologies

2.4.1 Computer Vision in Human-Computer Interaction

Discussion of computer vision technologies in input methods:

- MediaPipe integration for pose estimation
- Real-time facial tracking systems

- Hand and finger tracking precision
- Movement detection algorithms

2.4.2 MotionInput System Architecture

Technical analysis of the MotionInput system:

- Core system components
- AI model integration
- Input processing pipeline
- Profile configuration system

2.5 Software Architecture and Design Patterns

2.5.1 MVVM Pattern

Detailed analysis of the Model-View-ViewModel pattern:

- Separation of concerns
- Data binding mechanisms
- Command patterns
- State management
- Unit testing advantages

2.5.2 Clean Architecture

Principles of clean architecture in modern applications:

- Dependency inversion
- Use case driven development
- Repository pattern
- Interface segregation
- Domain-driven design

2.5.3 Design Patterns in C#

Implementation patterns specific to C# development:

- Async/await patterns
- Observer pattern with events
- Factory patterns
- Singleton and dependency injection
- Command pattern implementation

2.6 Modern GUI Development

2.6.1 WinUI 3 Framework

Analysis of WinUI 3 as a modern Windows UI framework:

- XAML-based UI development
- Integration with native Windows features
- Performance optimization techniques
- Accessibility implementation
- MVVM toolkit integration

2.6.2 AI-Enhanced User Interfaces

Emerging trends in AI-powered UIs:

- Intelligent interface adaptation
- Predictive user interaction
- Context-aware assistance
- Personalization through machine learning

2.7 Related Work

2.7.1 Similar Projects

Review of relevant accessibility and AI projects:

- AI-powered accessibility tools
- Computer vision-based input systems
- Machine learning in gaming interfaces
- Open-source AI integration examples

2.7.2 Technical Implementation Patterns

Key architectural considerations:

- AI model deployment strategies
- Real-time processing optimization
- Cross-platform compatibility
- Performance-accessibility balance

Chapter 3

Requirements Analysis and Specification

3.1 Research Methodology

3.1.1 User Research

Description of methods used to gather requirements:

- Analysis of existing MotionInput user feedback
- Review of accessibility guidelines and standards
- Study of similar configuration interfaces
- Consultation with project stakeholders

3.2 Functional Requirements

3.2.1 Core Configuration Features

Essential configuration capabilities:

- Create, edit, and delete visual profiles
- Configure input-to-action mappings
- Save and load profile configurations
- Import/export functionality
- Real-time preview of configurations

3.2.2 AI Integration Requirements

AI-related functionality:

- ONNX model integration for pose estimation
- Stable Diffusion integration for visual assets

- Real-time processing of video input
- Model performance optimization

3.2.3 User Interface Requirements

GUI-specific requirements:

- Intuitive profile management interface
- Visual feedback for configurations
- Drag-and-drop functionality
- Configuration validation system
- Profile comparison tools

3.3 Non-Functional Requirements

3.3.1 Performance Requirements

Performance criteria:

- Response time for UI interactions ($<$ 100ms)
- Frame rate for preview functionality ($>$ 30 FPS)
- Memory usage optimization
- Startup time ($<$ 3 seconds)

3.3.2 Usability Requirements

Accessibility and usability standards:

- WCAG 2.1 compliance
- Keyboard navigation support
- Screen reader compatibility
- High contrast mode support
- Customizable UI scaling

3.3.3 Technical Requirements

System specifications:

- Windows 10/11 compatibility
- .NET 6.0 or higher
- WinUI 3 framework integration
- MVVM architecture implementation
- Git version control system

3.4 Data Management Requirements

3.4.1 Configuration Data Structure

Analysis of data structure requirements:

- JSON vs XML format considerations
- Schema versioning strategy
- Backward compatibility needs
- Data validation rules
- Error handling approach

3.4.2 Storage and Persistence

Data storage requirements:

- Local file system integration
- Profile backup mechanisms
- Data migration tools
- Automatic save features
- Data recovery options

3.5 Evolution from Previous Versions

3.5.1 Previous Implementation Analysis

Review of earlier versions:

- Version 1.0: Python-based implementation
- Version 2.0: C++ transition
- Version 3.0: Initial Windows Forms GUI
- Version 4.0: Current WinUI 3 redesign

3.5.2 Key Design Decisions

Critical choices and their rationales:

- Migration from Python to C#
- WinUI 3 over Windows Forms
- MVVM architectural pattern adoption
- JSON configuration format selection
- AI model integration approach

3.5.3 Lessons Learned

Insights from previous versions:

- Performance bottlenecks identified
- User interface pain points
- Configuration management challenges
- Integration complexity issues
- Development workflow improvements

3.6 Design Constraints

3.6.1 Technical Constraints

Limitations and considerations:

- Windows platform specificity
- Hardware resource limitations
- Integration with existing MotionInput codebase
- Third-party library dependencies

3.6.2 Development Constraints

Project boundaries:

- Development timeline
- Resource availability
- Testing environment limitations
- Documentation requirements

3.7 Requirements Prioritization

3.7.1 MoSCoW Analysis

Prioritization of requirements:

- Must Have: Core configuration features
- Should Have: Real-time preview, AI integration
- Could Have: Advanced visualization features
- Won't Have: Cross-platform support

3.8 Validation Criteria

3.8.1 Acceptance Criteria

Success metrics:

- Functional testing pass rate
- Performance benchmarks
- Accessibility compliance
- User satisfaction metrics

Chapter 4

Implementation

4.1 System Architecture

4.1.1 MVVM Implementation

Description of the MVVM pattern implementation:

- Model layer design
- ViewModel implementation
- View bindings and interactions
- Service layer architecture

4.1.2 Core Components

Key system modules:

- Visual Profile Engine
- Configuration Manager
- AI Integration Service
- Input Processing Pipeline

4.2 User Interface Development

4.2.1 WinUI 3 Framework

Details of UI implementation:

- XAML-based interface design
- Custom control development
- Navigation system
- Accessibility features

4.2.2 Profile Editor Interface

Implementation of core UI features:

- Visual profile creation tools
- Real-time configuration preview
- Drag-and-drop functionality
- Interactive feedback system

4.3 AI Integration

4.3.1 ONNX Runtime Integration

Implementation of pose estimation:

- Model optimization techniques
- Real-time inference pipeline
- Performance monitoring
- Error handling strategies

4.3.2 Stable Diffusion Features

Visual asset generation system:

- Asset creation workflow
- Style transfer implementation
- Performance optimizations
- Resource management

4.4 Performance Optimization

4.4.1 Threading Model

Concurrency implementation:

- Async/await patterns
- Background processing

- UI thread management
- Task coordination

4.4.2 Resource Management

Performance considerations:

- Memory optimization
- GPU utilization
- Cache management
- Load balancing

4.5 Integration with MotionInput

4.5.1 Core System Integration

Connection with main application:

- Communication protocols
- Event handling system
- Profile synchronization
- Error recovery mechanisms

4.5.2 Profile Deployment

Configuration deployment process:

- Validation procedures
- Hot-reload implementation
- Rollback capabilities
- Status monitoring

4.6 Development Workflow

4.6.1 Version Control

Development process:

- Git workflow
- Code review procedures
- Documentation approach
- Team collaboration

4.6.2 Testing Implementation

Testing strategy:

- Unit testing approach
- Integration testing
- UI automation
- Performance testing

Chapter 5

Testing and Evaluation

5.1 Testing Strategy

5.1.1 Testing Approach

Overview of testing methodology:

- Unit testing of core components
- Integration testing of system modules
- UI automation testing
- Performance benchmarking
- Accessibility testing

5.2 Unit Testing

5.2.1 Core Components Testing

Testing of individual components:

- Model layer unit tests
- ViewModel behavior verification
- Service layer functionality
- Configuration management testing

5.2.2 AI Component Testing

Validation of AI features:

- ONNX model integration tests
- Stable Diffusion functionality
- Performance profiling
- Error handling verification

5.3 Integration Testing

5.3.1 System Integration

Testing of component interactions:

- Profile management workflow
- Configuration synchronization
- Event handling system
- Error recovery mechanisms

5.3.2 MotionInput Integration

Testing integration with main system:

- Communication protocol testing
- Profile deployment verification
- Real-time data processing
- System state synchronization

5.4 User Interface Testing

5.4.1 Functional Testing

UI functionality verification:

- Navigation testing
- Input validation
- Event handling
- State management

5.4.2 Accessibility Testing

Validation of accessibility features:

- Screen reader compatibility
- Keyboard navigation

- High contrast mode
- UI scaling tests

5.5 Performance Testing

5.5.1 Benchmark Results

Performance measurements:

- UI response times
- Memory usage patterns
- CPU utilization
- GPU performance

5.5.2 Load Testing

System behavior under load:

- Multiple profile handling
- Concurrent operations
- Resource utilization
- System stability

5.6 User Evaluation

5.6.1 User Testing Sessions

Feedback from user testing:

- Usability assessment
- Feature completeness
- User satisfaction
- Improvement suggestions

5.6.2 Feedback Analysis

Analysis of user feedback:

- Common usability issues
- Feature requests
- Performance concerns
- Overall satisfaction

5.7 Testing Results

5.7.1 Test Coverage

Overview of test coverage:

- Code coverage metrics
- Functionality coverage
- Edge case handling
- Known limitations

5.7.2 Issues and Resolutions

Documentation of issues:

- Critical bugs identified
- Resolution approaches
- Performance optimizations
- Future improvements

Chapter 6

Conclusion

6.1 Project Summary

6.1.1 Key Achievements

Major accomplishments of the project:

- Successful implementation of WinUI 3-based configuration GUI
- Integration of AI technologies (ONNX, Stable Diffusion)
- Improved accessibility through intuitive interface design
- Enhanced profile management capabilities
- Real-time preview and testing features

6.1.2 Technical Innovations

Notable technical contributions:

- Modern MVVM architecture implementation
- Efficient AI model integration
- Performance optimizations for real-time processing
- Robust error handling and recovery systems

6.2 Critical Evaluation

6.2.1 Strengths

Project strengths and successes:

- Intuitive user interface design
- Robust system architecture
- Efficient performance optimization
- Strong accessibility features

6.2.2 Limitations

Current limitations and constraints:

- Platform-specific implementation
- Resource intensive AI operations
- Limited cross-device support
- Performance constraints with multiple profiles

6.3 Future Work

6.3.1 Potential Improvements

Areas for future enhancement:

- Cross-platform compatibility
- Advanced AI model integration
- Enhanced profile sharing capabilities
- Extended customization options

6.3.2 Research Opportunities

Future research directions:

- Advanced pose estimation techniques
- Improved real-time processing
- Machine learning for user preferences
- Automated profile optimization

6.4 Personal Reflection

6.4.1 Learning Outcomes

Personal development achievements:

- Deep understanding of WinUI 3 development
- Experience with AI integration in desktop applications

- Practical MVVM implementation skills
- Project management experience

6.4.2 Challenges Overcome

Significant challenges addressed:

- Complex system integration
- Performance optimization
- AI model deployment
- User experience design

6.5 Impact Assessment

6.5.1 Accessibility Impact

Contribution to accessibility:

- Improved gaming accessibility
- Enhanced user configuration capabilities
- Reduced technical barriers
- Increased gaming inclusion

6.5.2 Project Legacy

Long-term project impact:

- Foundation for future development
- Contribution to MotionInput ecosystem
- Documentation and best practices
- Open-source contributions

Appendix A

Appendices

A.1 Development Setup

A.1.1 Environment Configuration

Development environment details:

- Visual Studio 2022 setup
- Required SDK installations
- Development tools and extensions
- Build configuration settings

A.2 Code Examples

A.2.1 MVVM Implementation

Key code implementations:

```
public class ProfileViewModel : ObservableObject
{
    private string _profileName;
    public string ProfileName
    {
        get => _profileName;
        set => SetProperty(ref _profileName, value);
    }

    public ICommand SaveProfileCommand { get; }

    public ProfileViewModel()
    {
        SaveProfileCommand = new RelayCommand(ExecuteSaveProfile);
    }

    private void ExecuteSaveProfile()
    {
        // Profile saving logic
    }
}
```

A.2.2 AI Integration Code

ONNX integration example:

```
public class PoseEstimationService
{
    private InferenceSession _session;

    public async Task InitializeModel(string modelPath)
    {
        var sessionOptions = new SessionOptions();
        sessionOptions.GraphOptimizationLevel =
        ↪  GraphOptimizationLevel.ORT_ENABLE_ALL;
        _session = await Task.Run(() => new InferenceSession(modelPath,
        ↪  sessionOptions));
    }

    public async Task<PoseData> EstimatePose(byte[] imageData)
    {
        // Model inference implementation
    }
}
```

A.3 Technical Documentation

A.3.1 API Documentation

Key interface definitions:

```
public interface IProfileService
{
    Task<Profile> CreateProfile(string name);
    Task<bool> SaveProfile(Profile profile);
    Task<Profile> LoadProfile(string profileId);
    Task<bool> DeleteProfile(string profileId);
    IAsyncEnumerable<Profile> GetAllProfiles();
}
```

A.4 User Guide

A.4.1 Installation Guide

Installation steps:

- System requirements
- Installation process
- Configuration setup
- Troubleshooting steps

A.4.2 User Manual

Usage instructions:

- Creating new profiles
- Configuring input mappings
- Testing configurations
- Managing profiles

A.5 Performance Data

A.5.1 Benchmark Results

Detailed performance metrics:

- Response time measurements
- Memory usage statistics
- CPU utilization data
- GPU performance metrics

A.6 Project Timeline

A.6.1 Development Phases

Project timeline details:

- Research phase: September - October 2023
- Design phase: October - November 2023
- Implementation: November 2023 - February 2024
- Testing: February - March 2024