Name	Shashank Kumar
Email ID	shashank.kumar.phe22@itbhu.ac.in
Phone Number	8340244118
GitHub ID	shashankiitbhu
Discord ID	shashankkumar6398
Current Occupation	Student
Education Details	**Indian Institute of Technology (BHU) , Varanasi** *2nd Year in B.Tech*
Technical Skills	**Java** (Expert) **C++** (Intermediate) **C** (Intermediate) **Kotlin** (Intermediate) **Javascript** (Intermediate) **MongoDB** (Intermediate) **SQL** **REST APIs** **Go** **Postman**

Prepare official releases of TuxType and TuxMath with accessibility extensions for public access

Summary

The project aims to release new official versions of TuxType and TuxMath with enhanced accessibility features for visually impaired users. This initiative is essential for inclusivity, ensuring that educational software remains accessible to all. Key features include screen readers, high contrast modes, and keyboard shortcuts. Compatibility across all platforms, collaboration with accessibility experts, thorough testing, community engagement, and strategic promotion are integral components of the project's roadmap. Sources include accessibility standards, user feedback, expert insights, and community contributions.

Project Details

Understanding the Project

- Expose accessibility features in Tuxmath and Tuxtype
- Fix existing bugs
- Make an official release for GNU/Linux
- Make official release for Microsoft Windows
- Make official release for MacOS
- Prepare and release official packages or installers for all platforms.

1.Expose Accessibility Features

1. Implementing Screen Reader Compatibility Description: Enhance TuxMath's accessibility for visually impaired users by incorporating screen reader compatibility.

Utilize SDL's accessibility support features to provide alternative text descriptions for UI elements. Implementation Steps: Identify UI elements (e.g., equation displays, buttons).

Implement text descriptions using SDL functions (SDL_SetAccessibilityText).

2. Developing High Contrast Modes Description: Improve visibility and usability for users with visual impairments by introducing high contrast modes.
Leverage SDL's rendering capabilities to dynamically adjust color schemes and enhance UI elements. Implementation Steps: Implement a setting to toggle high contrast mode. Use SDL to modify rendering parameters (e.g., color adjustments).

3. Creating Keyboard Shortcuts Description: Facilitate efficient navigation and interaction through keyboard shortcuts for visually impaired users.

Utilize SDL's event handling mechanisms to listen for keyboard events. Implementation Steps: Map common actions (e.g., submitting answers) to keyboard shortcuts. Handle keyboard events using SDL (*SDL_KeyboardEvent*).

2.Fix Existing Bugs

Utilize debugging tools such as gdb (GNU Debugger) and Valgrind to identify memory leaks and runtime errors in the codebase. Implement appropriate fixes, including code refactoring and error handling improvements, to address identified issues. This may involve:

- Analyzing crash logs and stack traces to pinpoint the root cause of bugs.
- Refactoring code sections to improve readability, maintainability, and performance.
- Enhancing error handling mechanisms to gracefully handle unexpected situations and prevent crashes.

3.Official Releases for GNU/Linux, Windows, and MacOS:

Follow platform-specific development guidelines and packaging standards to create official releases for GNU/Linux, Microsoft Windows, and MacOS. This involves:

- Compiling the codebase using appropriate compilers and toolchains for each platform.
- Generating platform-specific installation packages or binaries, ensuring compatibility and optimal performance.
- Adhering to platform-specific distribution channels and requirements for software releases, such as package repositories for GNU/Linux distributions and digital signing for Windows installers.

4.Prepare and Release Official Packages or Installers:

Package the finalized releases into official packages or installers for distribution to end-users. This involves:

- Creating platform-specific installation packages using packaging tools and utilities, such as dpkg and RPM for GNU/Linux distributions, and Inno Setup or WiX Toolset for Windows.
- Including necessary dependencies and resources in the packages to ensure smooth installation and usage.
- Providing clear and user-friendly installation instructions and documentation to guide users through the installation process.

5.Integrating Liblouis with TuxTyping:

1. **Integration Points Identification**
   Identify specific features within Tux Typing where braille translation is needed, such as menu navigation, instructions, and gameplay elements. Determine the scope of braille translation integration to prioritize essential elements for accessibility enhancement.

2. **Liblouis Library Integration:**
   Configure the build environment to include Liblouis as a dependency for the Tux Typing project. Link the Liblouis library with Tux Typing codebase to enable access to braille translation functions.

3. **Braille Translation Logic Implementation:**
   I will Develop logic within Tux Typing to utilize the Liblouis library for translating text into braille format. Implement integration points identified earlier, ensuring seamless translation of relevant text elements. Handle error conditions and exceptions gracefully to maintain application stability.

```
C/C++
#include <stdio.h>
#include <liblouis.h>

int main() {
```

```
    lou_init();

    const char *inputText = "Hello, world!";
    char *brailleText = (char *)malloc(strlen(inputText) * 2 + 1);

    int result = lou_translateString("en-gb-g2.ctb", "unicode", inputText,
brailleText, strlen(inputText) * 2 + 1);

    if (result >= 0) {
        printf("Braille translation: %s\n", brailleText);
    } else {
        printf("Translation failed!\n");
    }

    free(brailleText);
    lou_free();

    return 0;
}
```

4. **User Interface Adaptation:**
   Modify the user interface of Tux Typing to accommodate braille output for visually impaired users. Introduce options for users to enable braille mode and customize braille settings as per their preferences. Enhance feedback mechanisms to provide informative messages and notifications for braille users.

Timeline

Week 1: Project Setup and Planning

- Review project requirements and objectives.
- Familiarize with the codebase of TuxMath and TuxType.
- Set up development environments for GNU/Linux, Windows, and MacOS.
- Define the scope of accessibility features to be implemented.
- Create a detailed project plan with tasks and milestones for the next 12 weeks.

Week 2: Expose Accessibility Features (Part 1)

- Begin implementation of accessibility features, focusing on screen reader compatibility.
- Research and integrate necessary libraries and APIs for screen reader support.
- Develop alternative text descriptions for UI elements to enhance accessibility.
- Conduct initial testing to validate screen reader functionality.

Week 3: Expose Accessibility Features (Part 2)

- Continue implementation of accessibility features, focusing on high contrast modes.
- Dynamically adjust color schemes and enhance UI elements for better visibility.
- Implement toggling mechanisms for enabling/disabling high contrast modes.
- Conduct testing with visually impaired users to gather feedback on high contrast modes.

Week 4: Expose Accessibility Features (Part 3)

- Complete implementation of accessibility features, focusing on keyboard shortcuts.
- Map commonly used actions to keyboard combinations for efficient navigation.
- Develop user-friendly documentation outlining available keyboard shortcuts.
- Conduct usability testing with visually impaired users to ensure effectiveness of keyboard shortcuts.

Week 5: Bug Fixes and Optimization (Part 1)

- Begin identifying and addressing existing bugs and performance issues in the codebase.
- Use debugging tools such as gdb and Valgrind to isolate bugs and memory leaks.
- Implement code optimizations to improve stability and performance.
- Conduct regular code reviews and discussions with mentors to address challenges.

Week 6: Bug Fixes and Optimization (Part 2)

- Continue bug fixing and optimization efforts, focusing on critical issues.
- Address feedback and suggestions from code reviews and testing.
- Perform regression testing to ensure bug fixes do not introduce new issues.
- Document resolved bugs and optimizations for future reference.

Week 7: Official Release for GNU/Linux (Part 1)

- Prepare for official release of TuxMath and TuxType for GNU/Linux platforms.
- Compile the codebase using appropriate compilers and toolchains.
- Generate installation packages compatible with major GNU/Linux distributions.
- Conduct initial compatibility testing and address any platform-specific issues.

Week 8: Official Release for GNU/Linux (Part 2)

- Finalize official release for GNU/Linux platforms.
- Perform thorough testing to ensure compatibility and stability.
- Create release notes outlining new features and improvements.
- Publish official release on relevant platforms and announce availability to the user community.

Week 9: Official Releases for Windows and MacOS (Part 1)

- Begin preparing official releases for Windows and MacOS platforms.
- Compile the codebase for Windows using cross-compilation or native Windows development environment.
- Create Windows installer packages using Inno Setup or WiX Toolset.
- Package the software as macOS application bundles (.app) and ensure compatibility with recent macOS versions.

Week 10: Official Releases for Windows and MacOS (Part 2)

- Continue finalizing official releases for Windows and MacOS platforms.
- Conduct thorough testing to ensure compatibility and stability on each platform.
- Address any platform-specific issues or dependencies hindering the release process.
- Create release notes and documentation specific to Windows and MacOS releases.

Week 11: Prepare and Release Official Packages

- Package the finalized releases into official packages or installers for distribution.
- Ensure all necessary dependencies and resources are included in the packages.
- Create clear and user-friendly installation instructions and documentation.
- Conduct final testing and verification of installation packages on all platforms.

Week 12: Final Testing and Release

- Perform final testing and validation of official packages on all platforms.
- Address any last-minute issues or concerns identified during testing.
- Release official packages for all platforms and announce availability to the user community.
- Monitor feedback and address any post-release issues promptly.

Availability

Number of hours to dedicate to this project per week?	**35 Hours/Week**
Do you have any other engagements during this period?	**NO**

Personal Information

About Me

I am a seasoned Android app developer with a strong track record of creating successful applications.
My experience spans a range of technologies, including the Java programming language, C/C++, Android SDK, Android development, Kotlin, Jetpack Compose, and databases. This diverse skill set positions me well to lead the development of the Maths-Tutor app, leveraging my expertise to create an engaging and educational experience for users.

What is the Motivation to apply to this project?

I Have been previously involved in Zendalona's other projects and I am overall really interested in accessibility apps/tools that Zendalona is building.

Issues solved in other C4GT Projects

https://github.com/dhiway/cord/issues/319	Add tests for PresentationNotFound	https://github.com/dhiway/cord/pull/359

https://github.com/dhiway/cord/issues/318	Add tests for PresentationDigestAlreadyAnchored	https://github.com/dhiway/cord/pull/360

Previous Open Source Contributions

Project Name	Description	Links
Wikimedia Commons App		Link to all merged PRs

Contribution to C4GT's open community:

Have you contributed to tickets in C4GT's open community?	NO
Have you successfully completed C4GT's GitHub Classroom Assignment?	NO
DPG Points	0
Leaderboard Screenshot	0