Improving the HEAT IDE's Accessibility

COMP8860: Software Engineering

Josh Bassett (jsbb2), Amy Evemy (aee23), Adam Ryan (ajr79), Mos Samkumpim (ss2876)

This project involved adding accessibility features to the HEAT IDE software to make it more accessible for people with severe visual impairments. To achieve this goal we successfully implemented new features to the GUI, including GUI font resizing, new icons, a visual disturbance filter, syntax highlighting options for colorblindness, audio responses for compile status, and an extended setting screen.

I. Introduction

This project presented one of the largest programming tasks of our degree programme thus far. Requiring multiple members of different experience levels and backgrounds to collaborate on a brownfield development project on the HEAT Integrated Development Environment. To complete this task we utilised many software development practices and methodologies that we have learned throughout the module. This included organising our group based upon the Agile methodology and dividing up tasks for concurrent development, regular meetings and constant communication, utilising user stories to guide feature design, integrating testing into the development process, continual use of version control, and liaising with the customer to get continual feedback and facilitate continuous changes. As a result, it will be shown, that we scoped our project to the set timeframe, integrated the features that we had set out to, and delivered an artifact that met customer requirements.

II. Group Organisation

A. Division of Tasks and Concurrent Working

Our group was organised according to the features we had selected to implement in initial meetings. Initially work was divided into pairs or assigned to individuals according to the size or difficulty of the feature. For instance, replacing the icons was seen as a relatively straightforward procedure based upon our understanding of the code base, so this was assigned to an individual. In contrast, building the splash screen was a large task, so this was assigned to a pair. In practice, there was much discussion and crossover between all members of the team, helping each other on their assigned tasks as needed. This allowed us to collaborate while also working on separate features concurrently.

B. Teamwork and Group Member Utilisation

Overall, every team member felt well utilised throughout the process of the project, and we worked well as a team. The creation and tracking of feature implementation and related issues through GitLab's milestone feature allowed us to stay appraised of each members progress. Our stand-up meetings allowed blocks to progress to be brought up, and members frequently looked at each other's code to increase overall group knowledge of the code base and tackle said blocks. Decisions were made as a group, without one person taking an overall lead or manager role, leading to a highly collaborative development process, which was bolstered by our use of the Agile methodology.

III. Development Process

A. Selecting a Development Methodology

Our development process was agreed in a meeting the week prior to project week. In order to facilitate an efficient development process, we decided to organise our approach to the project around an Agile methodology, specifically Scrum. This meant that we laid out our goals in terms of features or tasks that we wanted to complete, within limited time frames. Given the short nature of the project as a whole, these timeframes were given in the form of days, rather than what might typically be weeks in a larger development cycle. Members were assigned tasks, either individually or in pairs, and mutually agreed timelines for the features were specified. We agreed to have stand-up meetings twice a day to discuss what was going well, if we were hitting any roadblocks, and what we might move onto next. Each of us would also perform manual testing of the feature we working on during development, with unit tests developed towards the end. This methodology was chosen out of our desire to implement multiple accessibility features into the GUI in the short amount of time allotted. The constant communication and division of work that Scrum and Agile methodologies enshrine would facilitate a more collaborative development process that made passing any roadblocks much easier. Agile methodologies also place a focus on meeting customer needs, which was an important focus for the project.

B. Communicating with the Customer

Whenever the opportunity presented itself, we discussed in detail with the customer our plans, features, and most importantly, their needs. Our customer provided feedback on work-in-progress features, and we incorporated that feedback into our software. For example, while working on the splash screen, our customer indicated the need for large GUI elements across the screen, with simple interactions. We kept this principle in mind, and as we approached the complete version of the screen, we took our customer's feedback

around increasing the font size of the splash screen into account as well, subsequently making the appropriate changes. This kind of customer-developer interaction ensured a clear focus for our iterative development process.

C. Development Process in Practice

Largely we stuck to our chosen methodology, although the realities of the project required flexibility. The division of work was fair, and no one person was overburdened or felt overstretched. Our timelines for some features were ambitious, and this meant somewhat deviating from the soft sprint deadlines we had agreed. However, we consistently held our twice daily stand-up meetings, and our constant communication meant that solving bugs could be done more efficiently together. Overall, we feel that our chosen methodology was the right one given the compact time frame, multitude of desired features, customer requirements, and small team composition.

IV. Requirements (Solutions)

A. Examining the Problem Task

The project task stated that we should "extend the HEAT system so that it can be used by students who have visual disabilities, such as partial or complete blindness". We individually researched common recommendations for making GUI's more accessible. Common vision impairments included blindness, limited vision, colour blindness, near and far sightedness. These can lead to a decreased ability to discern details on common GUI elements, as many of these are designed without these users in mind. Given that many users who have complete blindness will already utilise in-built operating system screen readers, or external screen reading devices, the necessity of implementing a text-to-speech function within HEAT itself seemed a lower priority compared to GUI improvements. The bulk of icons and text within the software was small, making it less legible for those with vision impairments. Additionally, the icons emphasised colour over shape, and were a low resolution. However the icons and menu items in HEAT already had keyboard shortcuts and hover text, providing some level of accessibility. With these considerations in mind we went forward with a primary focus on improving GUI elements to meet the needs of more visually impaired users. In doing so we referenced the W3C Web Accessibility Initiative reference guide [1].

B. Exploring Possible Features

In our first team meeting the week prior to project week (20th February) we discussed the requirements of the project, including scope and possible features. Two main strands of possible features were identified, namely improving the GUI and implementing in-built text-to-speech functionality. We identified the following features that could be used to improve the GUI:

- Syntax highlighting options for colour-blindness
- Allowing the GUI font size to be adjusted for legibility
- Creating a colour overlay option for the GUI for users who suffer from visual disturbance
- Replacing the icons in the GUI and increasing their size for legibility
- Integrating a settings screen to allow users to select and change these options

This initial collection of features represented a bulk of work for the team. Although we aspired to also include text-to-speech functionality the requirements of learning and implementing external libraries in the short time frame presented risks of overstretching. We agreed that if we finished the GUI features quickly we would then attempt to do so, however this did not become feasible.

C. Deciding on Features

Our feature decisions were decided with reference to the existing software. We knew that font sizing was already present on some elements of the software, namely the Editor and Console windows. Therefore implementing GUI font sizing seemed practical and achievable. Syntax highlighting was similarly present, and thus also made a goal. Adding a visual disturbance overlay colour filter seemed like a good way to increase accessibility for both the visually impaired, as well as those who would otherwise use overlays when reading in the physical world. Replacing the icons appeared simple, and effective for updating the GUI's overall legibility. We understood that implementation of these features also meant making extensions to the options and settings elements of the software, and took that into account when deciding the scope. Subsequently, the need to add a splash screen that presented the options for these features to the user on launch was decided shortly after starting the project.

D. Documenting Feature Progress

To document what features we were implementing as well as their issues, we used GitLab's built in milestones feature. This way we could create issues, assign them, and mark them as complete as the project progressed. GitLab then automatically generated burndown and burnup charts based upon this milestone tracking. This creates a clear timeline alongside our many commits giving greater clarity to our version control history.

V. Design

A. User Stories

Design decisions were driven through our usage of user stories and their associated acceptance criteria. For each feature we implemented we created a user story in the following format:

- As a <type of> user I want <specific feature> to <achieve objective>.

Following this we derived acceptance criteria from the user story. All of these were documented on the wiki on our GitLab repository. For example, to implement new settings

options for our features and ensure usability we derived some acceptance criteria as follows:

1. The settings manager must configure the settings according to user choices
2. The settings manager must present the settings in a way that is accessible to users who require the most accessible option of the software
3. The settings manager must appear first when the product is loaded
4. The settings manager can apply the settings or be dismissed

These user stories guided us in building the new features. We also took into account evolving customer feedback during the development process, and integrated these into our user stories to ensure we met customer expectations. This specific example itself changed based upon customer feedback late into the development process.

B. Design Considerations

Given that HEAT's initial purpose is to be a simple teaching / learning tool, we opted to keep the same overall layout. Not only did this mean we were not cluttering the interface, but also that current users would have continuity. This is also because HEAT follows a standard interface layout that visually impaired users would expect, going from top-to-bottom, and left-to-right. As such, our features retained the overall structure of HEAT's GUI, while increasing legibility for the end user.

The new settings screen that pops up when opening the program is where main design considerations were placed. We created a low fidelity wireframe layout using Figma design software. This gave us a simple blueprint, from which we could layout the relevant components in Java. The customer liked our initial layout of the settings, however they requested that our fonts should be the largest possible in our settings by default. In doing so, a user who opened the program for the first time and needed such settings, would be able to make selections more easily. We made these changes to the settings menu to meet these customer requirements. However, later issues meant that our initial design had to be changed in order to ensure setting functionality within the deadline. We did however keep the feedback of the customer in mind, and integrated large font sizes into this late redesign.

VI. Implementation and Code Quality

A. Implementation Choices

Some features had a clear implementation method given the existing HEAT architecture, and other were more obfuscated upon first inspection. Implementing the icons was functionally a relatively simple change. The icon folder was clearly labelled and we replaced the icon images as needed. The way icons were called in the ActionManager class required relatively little code work. Font resizing existed for the Editor and Console windows respectively, therefore to implement this into the existing settings functions we examined where fonts were set throughout the code. It was not initially clear how this would be integrated into different GUI Java components such as JMenuItems, JButtons etc. This was especially challenging as different windows had differing amounts of components to change. However, a function was constructed that could take an indeterminate amount of objects, of different types. This was then called for each window as appropriate. It was necessary to call this function in each window upon creation, in order to match HEAT's current architecture. A similar approach was used to implement the visual disturbance filter. Each page created required a function to be called. The initial approach to this problem was to attempt to change the background colours of each window to a new user setting. This was marred however by how elements above the background in certain windows overlayed their own white backgrounds. A more elegant solution was derived that utilised Swing's built-in glasspane and JLayered pane methods on JFrame and JDialogs respectively. Upon the creation of a window these would be applied by reading the current users settings file, similarly to how the font resizing feature was implemented. The syntax colour feature was an extension based upon the default syntax style method in the code. A new function maps the chosen colour to the syntax as selected by the user, and this function is called in the WindowManager class. Integrating the audio response feature was fairly straightforward, and only required half a day for one group member. The splash screen seemed straightforward on initial creation, however implementation with our feature settings created many problems on the final day of the project. Our entire approach had to be rethought, and we essentially recreated the entire settings page within HEAT's existing OptionsWindow class on the last day, extending the class significantly.

B. Understanding and Exploring HEAT

As a group we all examined the code of HEAT before the outset of the project, so that we would have some base knowledge to work from. Throughout the project this became an active and collaborative process, as our features crossed similar parts of the software and touched the same existing classes. Our choice of IntelliJ as our IDE at the beginning of the development process was integral in aiding our understanding of HEAT's current architecture. IntelliJ allowed us to track a functions usage across multiple files, perform global searches across the code base, and quickly jump around different classes to understand how they are interlinked. This allowed us to gain a greater understanding of how settings are controlled within HEAT, and therefore extend these options to our new features. This also aided in hunting down what was causing errors in our code, as we developed the new features. HEAT's existing file structure was relatively clear, although the code was confusing in many areas. Variable naming conventions were inconsistent throughout the code base, and some names had spelling errors which made locating certain elements difficult. However, the code was well commented, which provided the necessary context when combined with IntelliJ's features to understand the code correctly. Many of the existing classes

also functioned in a similar manner to one another, which provided a clear roadmap for integrating new features into the software.

C. Successfully Implemented Features

Over the course of the project we successfully implemented all the features we had aimed to do in our initial meeting and discussions. In the process of doing so we tried to match HEAT's existing internal structure and code quality. This included making Manager classes for our new features which could be called from existing classes, commenting our code as appropriate, and extending HEAT's existing implementation of settings in-line with its current functionality.

VII. Quality Assurance

A. Unit Testing

Given the short amount of time allocated to complete the project we implemented some simple unit tests late into development. These tested basic functions of HEAT ensuring correct output from our custom methods and classes. This included the overlay, font sizing, and syntax highlighting class methods, and returned pass/fail messages as appropriate. However, last minute changes to the software meant that we could not fully implement the unit tests into the final version of our code on the "main" branch. Instead the code for these unit tests is available on the branch called "testing". Despite this, manual testing of our features confirmed functionality of the software's final version. This was also bolstered by our pair programming paradigm and peer code inspections.

B. Code Inspections

As we were working in a small team utilizing an Agile development process, informal code inspection of each other's work was frequent. As we helped each other on features we looked intently at others code and helped correct and identify errors. Our experience with Java is similar across the team, so it did not make sense to assign a single person this role, but instead utilise our collective experience to increase efficiency. This process was most evident when we performed merges of our individual branches, solving conflicts in code while maintaining the functionality of independently developed features. For example, understanding the functionality of how HEAT applied user settings in the background required multiple members looking through the code and having detailed discussions. This alongside manual testing by each group member throughout development ensured functionality across the code base, even without wider automated testing.

C. Meeting Customer Requirements

Discussions with the customer were frequent throughout the development process. We made changes to the software iteratively depending on the customer's feedback. Given the initial requirements set out we have implemented multiple features that should aid users with visual impairments. The ability to change the settings also gives the end user the ability to customise according to their liking, which is preferable to a static GUI change. Overall, we feel confident that we have met the customer expectations as they stated, as well as implemented features according to the specification of the project.

VIII. Tools

A. GitLab

We decided from the outset to push the original version of HEAT that we were supplied to the "main" branch of our repository. We then branched off of main with a branch called "dev", to which we would only merge fully working, tested features. This ensured that later merges would be simpler, and that we were not all editing and pushing to the same development branch. Subsequently, more branches were made for other features as needed, and merged as appropriate throughout the project. This detailed version control process allowed us to work on features independently, and merge continuously throughout the project. This also had the added benefit of making the final merges to "dev", and back to "main" seamless. Throughout the course of the development we all became very familiar with using Git and GitLab through the CLI interface provided by Git Bash. We are in agreement that this experience will be very useful in the future, both for personal and professional projects.

B. Other Tools & Resources

In terms of tools we diverged slightly from the recommendations by using a different IDE. Instead of Eclipse we used IntelliJ as our main IDE, this choice was driven by Eclipse's poor implementation of version control with GitLab. IntelliJ gave us better control and had some useful features for understanding the code such as linking to usages of methods. IntelliJ also prevented any errors around using a "workspace" as required in Eclipse, which did not seem to work seamlessly with the university's remote user file systems. The use of IntelliJ also made creating the final JAR artifact much simpler. It has an in-built feature for this that made deployment of the final software easier. This was especially useful as we hit some last minute roadblocks on the implementation of our settings screen.

Figma was also briefly used as mentioned before to create a low-fidelity design for the splash screen. The new GUI icons were sourced from Google Icons, and some custom icons were derived from these to match the design. These icons were chosen for their legibility and distinct shapes, which provided clarity without using colour. Some feedback sounds were also sourced from freesound.org for the audio feedback feature. Care was taken to implement sounds that were recognisable in their intention and meaning, without using language, as that may have presented a language barrier.

IX. Understanding & Insight

Overall we were pleased to integrate all of the GUI features we had initially planned to implement. These were as follows:

- Changed GUI icons
- Implemented a visual disturbance filter with multiple options
- Implemented a syntax highlighting feature with multiple options
- Added font resizing feature for the entire GUI
- Extended the options functionality of the software, with additional clear accessibility options to control the new features
- Implemented an audio response feature into the compile status notification

Although we delivered all the features we had intended to, there was some room for improvement. Notably, the syntax highlighting feature did not save its setting between loads of the software. While we were able to implement setting continuity for all the other new features, the underlying structure of HEAT complicated this for the syntax setting. Additionally, although we implemented custom icons on our accessibility options menu, and we programmed the Main class to show a splash image on load, unfortunately the image did not display when compiled to a .jar file. In the deliverable the features work as planned, however it appears that the .jar file was unable to pull the new image files into the final artifact. Unfortunately, due to time constraints, we were unable to fix this in the final artefact delivered. In the implementation of the font resizing and visual disturbance filter we also hit a wall adding these features to the JFileChooser and JOptionPane components. This is likely due to how these components pull their styling from a system *lookandfeel* setting. Therefore, had we changed these we may have also affected a user's underlying operating system accessibility settings. Nonetheless, we were unable to modify these elements in the given time frame. If we had more time we would have liked to clean up these issues in the final artefact.

A last minute roadblock significantly halted our progress on the final day. We had constructed our custom splash screen to hold our accessibility settings and display it the user on launch. We spent a large portion of this final day attempting to solve the incongruence between the settings functionality on HEAT's backend, with our new frontend setting screen's structure. It is unclear despite our best efforts what the underlying problem is. Two main theories for the problem were derived. Either something went wrong during our attempt to merge that escaped our notice, or the way HEAT stores settings in its Properties file did not allow us to create a separate save setting function in the ActionManager class. A failure to integrate a settings screen would have made our new features unusable. However, we were able to rebuild this entire screen by integrating it into HEAT's existing options screen. We then modified the code so that this options window was displayed to the user on launch, and opened to the accessibility settings tab. This allowed the user to modify the settings as appropriate, and functionality was restored to our features. We also kept our legibility requirements by setting the GUI default font size to the maximum on first launch, ensuring customer requirements were met. Unfortunately, some of our cleaner design elements were lost in the implementation of this new screen. If we had more time we would have liked to fully implement our custom design into the options screen.

Our overall plan for the project seemed well scoped and thought out, even in retrospect of these difficulties. Our choice to focus on the GUI elements at the expense of implementing a text-to-speech feature seems reasonable, especially given that we were able to implement our target features. Choosing an Agile methodology for the development process worked well, facilitating communication and collaboration between members. With a longer timeframe, we likely would have reduced our stand-up meetings to once per day. We also would have liked to implement a text-to-speech feature if given more time, although that may have complicated the creation of our JAR artifact through the inclusion of more external libraries.

X. Conclusions

In conclusion, the project went well overall, although was very challenging with such a short time frame. It felt as though this assessment was perhaps the closest to actual software development in the professional world that we have experienced at university. There is no doubt that this experience has been formative and will be valuable moving forward into our careers. Although we were able to deliver all the features we planned, it is fair to say that it was not without some significant blocks along the way. However, our use of development methodologies, version control tools, and iterative development allowed us to deliver a version of the software that we are happy with.

References

[1] W3C Web Accessibility Initiative, How to Meet WCAG, Available at: https://www.w3.org/WAI/WCAG22/quickref/ (Accessed 20 February 2025)