Developing Java Based Extensions to the HEAT Program to Accept Visual Disabilities

James Brunkow[1]*, Luke Burston[2]*, Sophie Rogers[3]* and Lukasz Tomaszewski[4]*
(jwb50@kent.ac.uk[1], lb800@kent.ac.uk[2], sr652@kent.ac.uk[3] and lrgt2@kent.ac.uk[4])

*Division of Computing, Engineering and Mathematical Sciences, School of Computing
The University of Kent, Canterbury, Kent, CT2 7NZ, United Kingdom*

***Abstract*—The Haskell language learning tool program HEAT lacks certain features that make it more accessible to people with visual impairments. We aimed to rectify this by implementing certain features based off research into various types of visual disabilities and develop an updated version of the program which is more user-friendly. In our development we implemented features that allow the user change the size of all the fonts within the program, the icon sizes of the all the toolbar icons, various colour profiles of the application to help users with different colour-blindness, and have all these features easily available on a new toolbar which can be hidden. We were unable to implement further features, such as text to speech, and discovered some minor, but non-program breaking bugs, which in the future can be fixed and improved.**

***Keywords*-formatting; style; design; usability; inclusively; accessibility; functionality;**

I. Introduction

The Java based program HEAT (Haskell Educational Advancement Tool) [**?**] is designed as a teaching tool for the Haskell programming language. The program serves this purpose via a basic minimal GUI that has many limitations when considering accessibility features for users with visual disabilities, such as partial or complete blindness. Such features consist of; text scaling, dark mode, other colour modes, invert contrast, GUI scaling, button/ icons scaling, text to speech, such features are addressed in the User Stories **??** as issues needed addressing during this first sprint. In 5 days, we expect to address over half of the user stories and create a fully functioning features within the program with minimal bugs/ issues without interrupting the original functionality of the HEAT program.

II. User Stories

The user stories were formulated based on research, having identified the most common visual disabilities, and some general requirements that come with these. Having conducted background research and identified some key visual disabilities as being full blindness, partial blindness and colour blindness, a range of user stories were formulated to suit these. A couple of these focused on colour blindness, and involved requirements for being able to customise colour options, or colour contrasts, or apply colour filters, to make the application more readable for users with specific colour deficiencies. Additionally, user stories involved situations where users had low vision, or partial blindness, and required the ability to resize text, and make it larger to suit their needs. Background research revealed that this being fully customisable was important, rather than having a limited range of text sizes to choose from. As a result of this, a user story was also added which requested the full window to resize, as opposed to just the text, in order to enhance readability for a wider range of users.

Another element considered through the user stories was the need to hide accessibility options, in order to not overload a user who is not visually impaired, with features which are not necessary or beneficial for them to use. A user story was also produced to request an easier means of turning on accessibility features, for example, by displaying this on the startup screen, which also gives the users who do not require such features the chance to opt out of them initially. The user stories which remain unaddressed required implementation of speech synthesis, which given more time, would be the next feature to be added, but was not possible on the timescale given. Given the other features implemented appeared more significant, given that external screen readers generally can be used, this was not considered a priority, given the development time it would have required.

III. Background Research

The majority of background research conducted centred around the most common visual disabilities, and known technological/design solutions used to make software more accessible to users with these conditions. The most common visual disabilities, which the user stories were then built around appeared to be full blindness, partial blindness (or low vision [**?**]), and colour blindness. Another aspect which also seemed prominent was readability for users with dyslexia, who may benefit from some kind of green/yellow overlay or background, against black text [**?**]. In terms of font type, it is clear that universally, Sans Serif fonts tend to be the most readable, for users with or without visual disabilities. The default font in HEAT was set to Monospaced, which potentially may be difficult for some users to view. A large portion of the research also focused on text to speech, or screen reader use. [**?**] It was clear that this would be a complex feature to implement, and so the decision was made to focus predominantly on visual aspects. A common way of implementing this is through use of the java.accessibility package, which provides the

ability to specify and load assistive technologies, or allow the use of an externally sourced screen reader or magnifier.

The background research gave a good indicator of colour schemes and combinations to avoid, mainly being focused around reds and greens, which are the most commonly difficult to see in users with colour blindness, [**?**] with a blue or total deficiency being more rare). In terms of users with dyslexia, however, certain shades of green may be beneficial. As a result of this, it was decided that as a starting point, 4 colour modes would be appropriate, with potential to develop further with any remaining time. Additionally, the need for the ability to customise features is also apparent, as users with visual impairments have a wide range of needs, which can not be met through a set and limited number of default options.

From looking at the accessibility options within other IDE's suitable for beginners, for example, BlueJ, it became clear that visual disabilities are often not taken into account. A better example, in terms of accessibility considerations is Microsoft Visual Studio [**?**], which has toolbar and text enlargement, text size options, colour customisation and keyboard shortcut customisation. BlueJ however has just the ability to change font size, and only in the code editor, which is not applied to the GUI overall. Both of these existing softwares are however compatible with screen readers, but BlueJ takes a decent amount of steps to actually get this compatibility set up to a working level. This highlighted the importance in not only implementing new accessibility options within HEAT, but also ensuring that these features could be used with accessibility tools.

IV. Development

Before we began our implementation of our feature set, we made sure to examine the code we would be working with, before project week began. We did this so we could roughly identify where in the code we would be looking for variables to use and methods to call, as well as to identify if there was pre-existing code that we could build our functionality in to, to streamline the development process. We then began the process of identifying how much we could implement given the time frame. We concluded that all of our changes should be optional for the user to view, hidden in an accessibility toolbar which we can toggle the visibility of. This became our first goal set in the SCRUM working methodology we chose to adopt for this project week. We chose to treat this week as a short 'sprint', incorporating stand-up meetings every morning, and debriefs every evening to keep a track of what was being worked on, who was working on what, what was successful and what still required attention. We chose to use SCRUM as it provides a strong framework for small development cycles, as well as helping us develop collaborative skills and keep the entire team on track and aware of what is occurring on each day.

We had generated user stories before the week began, collated these to get a list of all possible requirements users could have for HEAT. We then split the list into themes, namely changing the color of the GUI (low blue light, dark mode, specific colorblind modes), changing the size of font and icons, the ability to set up the GUI before even loading the application, the ability to have the code read back to you through text to speech, and the ability for the interpreter to identify like fields (variables, data etc.) and differentiate between them visually. We chose to focus on the user's ability to manipulate the GUI to suit their accessibility needs, as we had identified some usable variables in our initial code review, which would cut down on the development time and with the time constraints of this week, we concluded that these features would provide the most flexibility to address as many use cases as possible.

The distribution of work was relatively even throughout the entire development process, however as the group came to the project with a variety of skillsets, we were able to differentiate the types of work allotted between individuals. One prerequisite we had from the onset was that each member of the group should make at least one commit to GitLab, and one commitment to the Java code in order to ensure that they are familiar with the process.We initially split the group into 2 on Monday, with Luke and James focusing on the first code commit of an accessibility toolbar, with Sophie and Lukasz focusing on researching the visual impairments we would be looking to address with said toolbar. At the end of the day, we were able to determine 3 areas where that would need to be developed for HEAT to cater to visually impaired users: text & GUI size, window color and accessibility options.

Luke, Sophie and James were assigned one of these three, with Lukasz developing GitLab as he was less familiar with Java, and more familiar with using Overleaf and Latex to generate high quality documentation. Sophie worked on the recoloring of the GUI, as she took the lead on the research of the topic so was best informed to do so. Luke took on the development of the accessibility options windows, and James took on the development of font and GUI size manipulation. We made 3 branches representing everyone's development tasks, merging them after each team member had fully implemented the functionality specified by their relevant user stories. Final merging to main occurred on Friday, where Luke focused on fixing any bugs where the rest of the team focused on documentation.

A. *Tools*

We used the issue board to store our user stories and the research surrounding each of the issues reported in them. We also used it to keep track of what was in active development, and what was functional. Aside

from GitLab, we used a java packacge called Flatlaf. Which allows the recolouring of the GUI. It allows us to generate themes which were used to address many issues raised in user stories surrounding changing HEAT's colourscheme.

B. *Implementation*

Our implementation of the HEAT modifications was based on the features identified through our user stories. They were split into 3 categories: accessibility menu options, text and GUI resizing and GUI recoloring. Before beginning the development process, we identified some of the methods and variables we would need to interface with. This directed the development of some features (font size, GUI scale) where there was already functionality similar to what we wanted to implement that just required modification. Features focused around the generating of additional jPanels and windows was duplicated from HEAT's pre-existing panels and windows and repurposed for the implementation we set out to achieve.

Other features, such as the GUI and font recoloring, had no explicit implementation already in place, which led to the exploration of multiple options surrounding its development. The option to include a pre-made solution to recoloring jFrames was made due to the time constraints of the project, as a similar handwritten implementation would have taken far longer to write than we had time available. As soon as the first color scheme was functional, the rest of the schemes were far simpler to implement, as we had a reference to pull from. We located many of the methods we needed our code to interface with through identifying which settings.java setting they pulled from, and back-tracing the path the code takes, either manually or through the use of the eclipse debugger. In all, we successfully implemented:

An accessibility toolbar which is present separate from the original toolbar. It starts hidden, unless specified to open in the first time start wizard. After settings have been changed, it can be hidden again.An accessibility option on the setup wizard which allows the user to boot in for the first time into the options menu to address their accessibility requirements before loading into the GUI. Buttons to increase and decrease the sizes of all text elements in the GUI, as well as granular controls for different field font size in the options menu. This saves in the settings.java file so the GUI should remain the same between sessions using HEAT.Buttons to resize icons in the GUI, including the accessibility toolbar and the original toolbar. Changes to the default running layout of HEAT, to allow it to spawn in Fullscreen mode with all relevant windows shown. Buttons to change the colour palette of the entire GUI, both frame colours and text colours to suit multiple common usability requirements.

V. Quality Assurance

In order to reduce the chance of unforeseen bugs or errors, we engaged in some basic quality assurance practices during and after the development of each feature. These can be boiled down to:

Pair Programming - At various times during the development of our modifications for HEAT, the group has engaged in Pair Programming. This provides multiple benefits to the development, including the re-affirmation that the code being implemented is of sufficient quality. This also has the effect of familiarizing other members of the team with the code that is being implemented, in case they are at some point required to work on it. The opposite is also true, as with the additional knowledge of what code fragments are being updated, other team members are aware what to avoid modifying, in order to avoid merge conflicts. Pair Programming was the primary methodology used for specific functionalities, including the initial development of the accessibility toolbar. As this was performed early during the development cycle, James and Luke were able to quickly build an understanding of the layout of the HEAT Jframe systems.

Pre-push code reviews - Before each major push to branch / main, we arranged to do code reviews before committing and pushing. These allowed the creator to explain what was happening at the stages in a method (to get it clear in their own heads as well as to explain it to their team), and to address what other variables a method interacts with. We were able to do this for almost all pushes, in retrospect this would be something discussed at the beginning of a sprint as part of the procedure for pushing updates, instead of doing it when 2 members of the team were present and available.

Interactions with customers - We have involved customers at multiple points during the development, as a means of making sure the development is on track with what they are expecting as a product. The first instance of this was on Tuesday, in which the customer indicated that on smaller screens from further away, they were unable to read any of the text or see any of the button icons. This refocused our efforts into increasing the size of not only all the elements of the GUI (which was completed by Thursday evening), but also the GUI itself, a task which was complete by Thursday morning. An interaction with the customer on Friday morning indicated the ambiguity of the icon resize functions, which we promptly fixed on the final update of the modifications.

A. *Testing*

We ran multiple unit testing on all the branches before making a final push onto the features dedicated branch and tested as we merged. We explored and ran examples to the customer changing and adapting the

software accordingly to their feedback. The Software was test on each feature as to their intended function and then the the base Haskell program was tested as well to ensure that the program still ran as a learning tool.

VI. Discussion

Our final version of HEAT was successfully created within the time frame that we originally planned on and we were able to make the program run with all of the features that we began implementing. We were not able to start all of our planned features, but those that we were able to start, function how we intended them to and in accordance to our research and User Stories. Unfortunately, there are a couple of bugs within our final version that will be discussed, however they do not render the program broken, even though it may seem so and if we had more time to fix them, we would have. We believe we successfully implemented the SCRUM approach to our project. Each morning we had a stand-up meeting in which we discussed any additional work we had worked on and updated each other on any issues we had and solutions to problems and features. We would then discuss our plans for the day, splitting tasks up individually or into small groups. We found this process throughout the week to be useful as it allowed us to work in parallel on separate parts of the project, whilst still having regular meetings to discuss where we were at and what we needed from each other.

We decided we should address User Stories that would provide a basis for implementing others. For example, from our user story on having a new toolbar with these accessibility features, we wanted to first have the toolbar up and running before placing the features of editing font/icon sizes or colour options. This ended up being relatively simple, as we just needed to copy the pre-existing toolbar, however we also were able to identify how to add buttons to this by first adding a button to the original toolbar that toggles the accessibility toolbar. This was also in line with our user story to maintain regular function for users who do not need these setting.

Resizing the text and icons was a harder feature to implement. HEAT already had settings to do this, but it was only set up to work for the Console and Editor windows. We were able to update this and assign specific text resizing buttons on the toolbar. We also added to the Font Size options window these additions, so that users can specifically and individually set the different parts of the program to have different sized texts. We did not include a limit to the font sizes on the buttons, if we had more time this would have been possible, however we also did not want to limit the maximum font size, as it is tied to the resolution size of the monitor, as font size may be smaller on a person's monitor that has a higher resolution. The icons however were extremely difficult to change as it was difficult to access the action's 'ImageIcons', however it did end up working, and has three size settings that can be chosen. One slight problem we have found is that in the Editor Window, when text is increased, sometimes the line numbers overlap with the text.

By far the hardest user story to implement was the Colour Options, due to how the program had been created. After a day of investigation into the code we were able to utilise a function in the code that set the 'LookAndFeel' of the application, a Java setting that provides a template for JComponents design and interactions. Whilst computers come with a limited set of 'LookAndFeels', we made use of an external library called FlatLaf [**?**]. FlatLaf provides a "Theme Editor" that allowed us to create our own custom themes based on our research, which completely sets the colours of all the JComponents in the application once it is passed through the setLNF() method already provided within HEAT. We made a default light, a dark, a blue filter, and a green light mode which can be easily switched between on both the toolbar and in the options window, as well as being able to set this straight away on the Initial Setup Wizard Window. We believe we successfully answered the user stories involving colour and light doing this, however given more time, we would have liked to implement more.

One bug we found right at the end of our sprint and so were unable to fix, is within the Wizard Window during the user's initial setup of the program. Currently it is unable to save the entered interpreter path if a user selects a colour profile, but this can still be set in the options afterwards and would be fixed in a future update. We believe this is something to do with how changing the colour mode creates the settings file earlier than the program expects. Although, our program still runs well and we have even ensured that any changes to the GUI are saved and will be displayed upon reopening the program.

VII. Conclusion

We were able to implement several features into HEAT, as well as updating fonts and sizes, that would make the program more accessible to users with visual impairments. We were able to complete 8 of our 9 user stories, the only one of which we did not being Text to Speech, however if we had some more time this would have been something we would have tried to implement. The 8 we did complete run successfully and our approach went smoothly. We were able to work well as a team, and our methodology of using SCRUM and the various Quality Assurance techniques utilised helped us to complete these features in time and deliver a working product. Given more time we would like to address a few bugs that were created, however as they do not break the program, this is something we would fix in the future, as well as implementing more features to further assist users.