32222145

SDD Assessment Task 3

Programming Project

College Connector

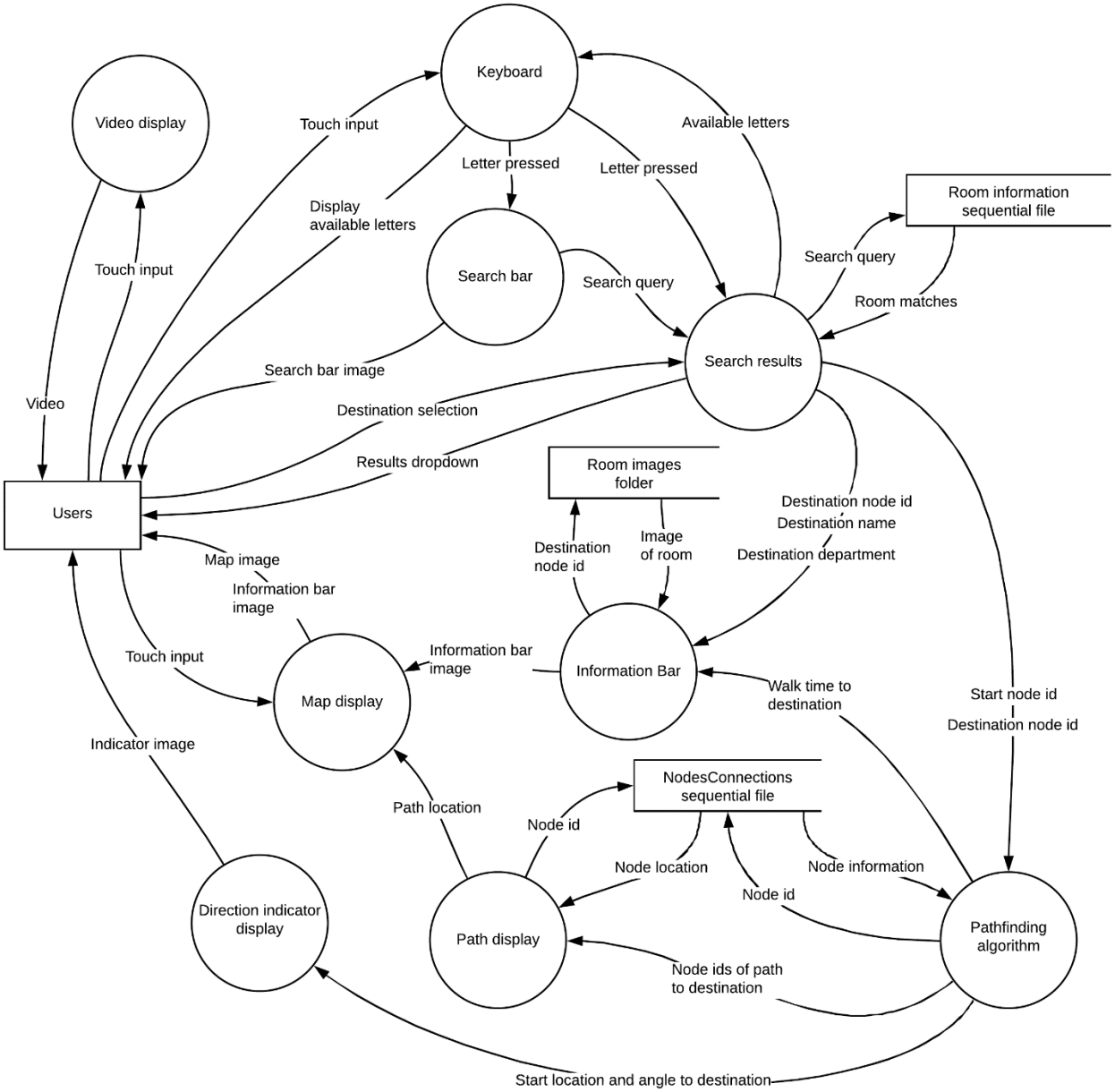
An intelligent touchscreen guidance system for the Barker College Hornsby campus.

Table of Contents

9.2.3 Implementation of Software Solution

The software solution implemented for this project, *College Connector*, has been sent alongside this document and can be run by executing the 'CollegeConnector.java' file. The complete and packaged source code and this executable program can be run successfully by following the Installation Guide for Java 8 and the program itself found below. A video is also presented demonstrating the complete operation of all program features.

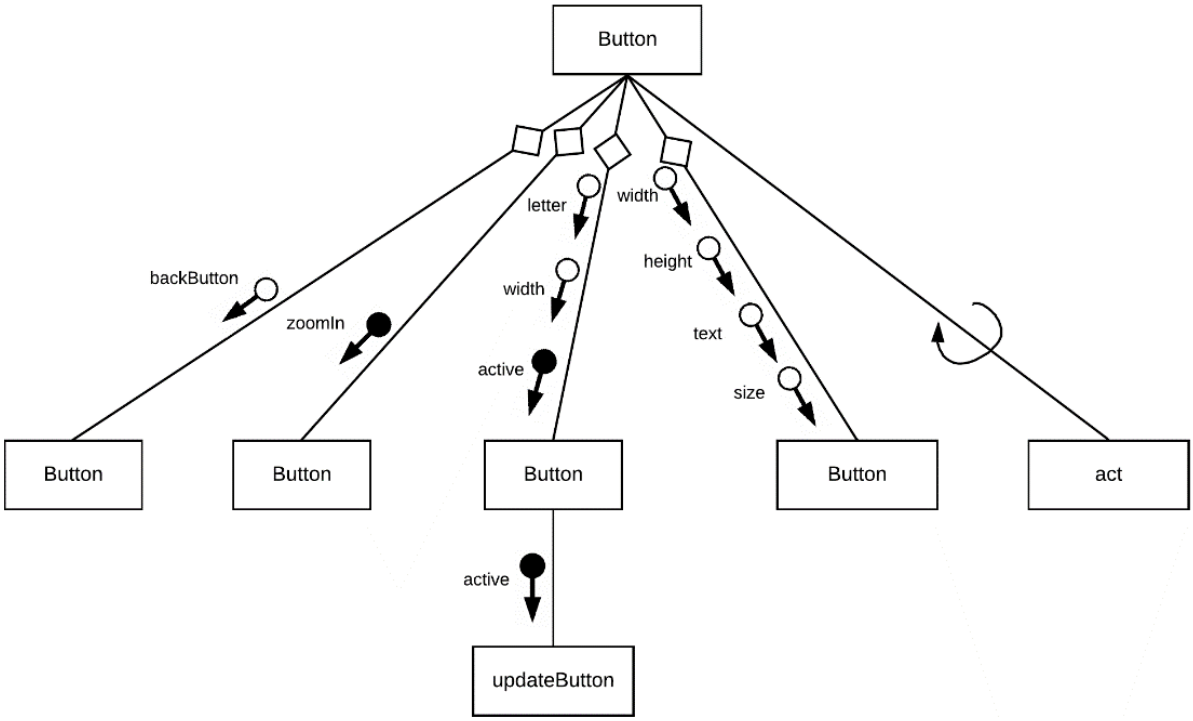
Data Flow Diagram (Level 1)



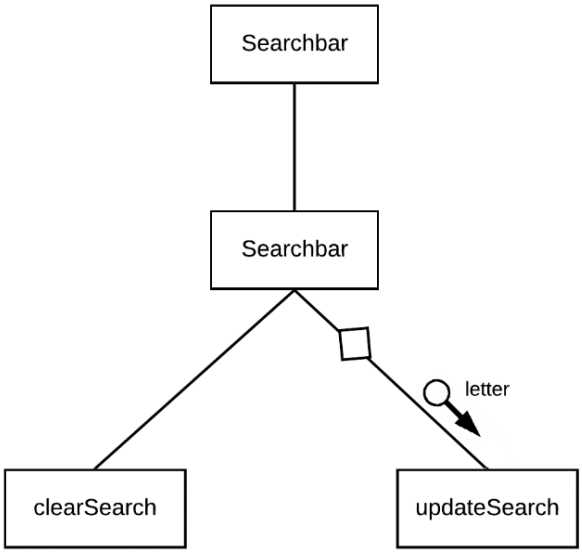
Structure Charts

***Note**: Sub-procedures with the same name as the class itself represent the constructors for creation of the class. These methods execute when the class is instantiated as an object. This is a sub-product of adherence to Object Oriented Programming throughout the program's design. Repeated constructors are due to the implementation of overloading.*

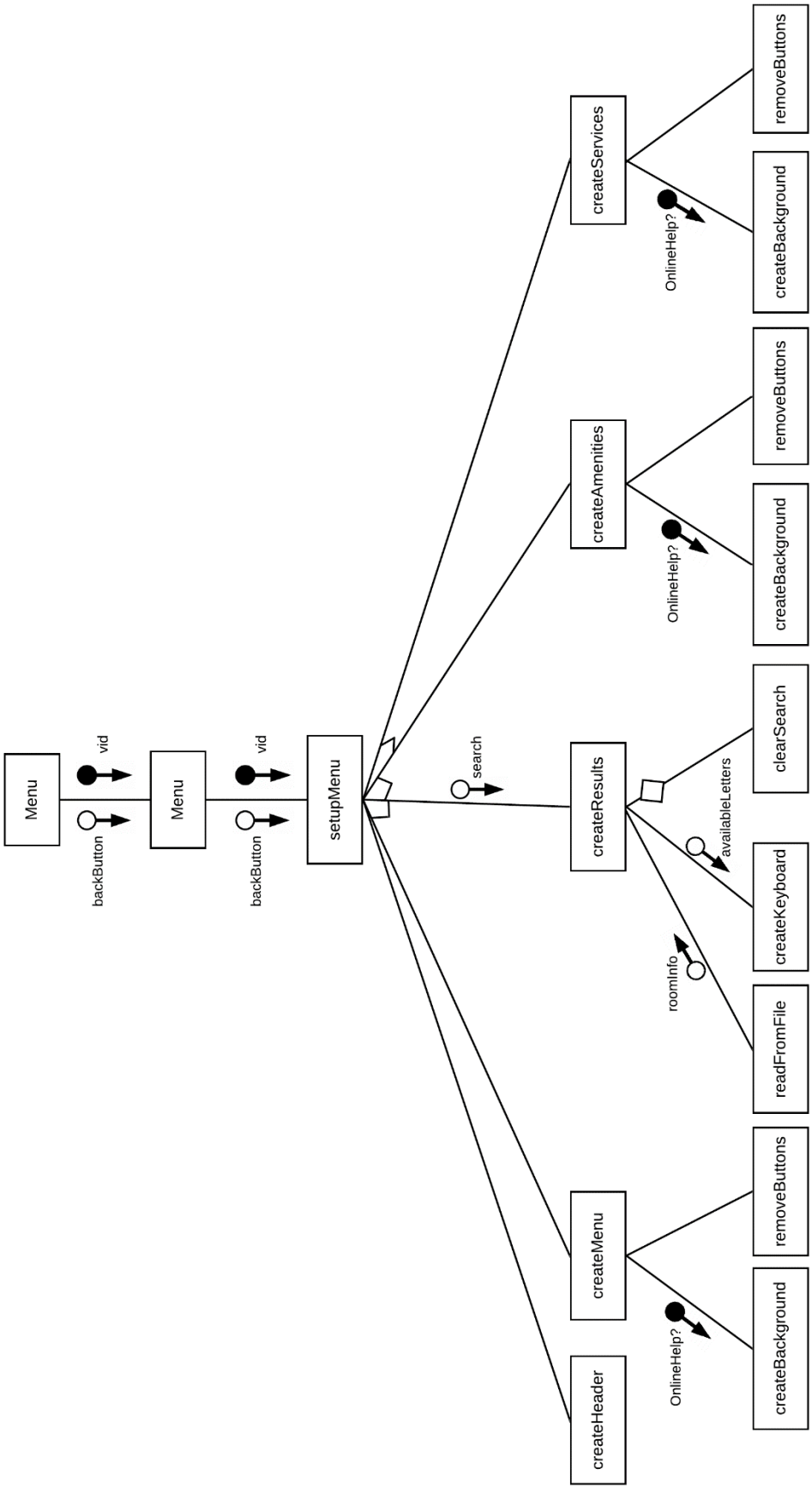
Button class



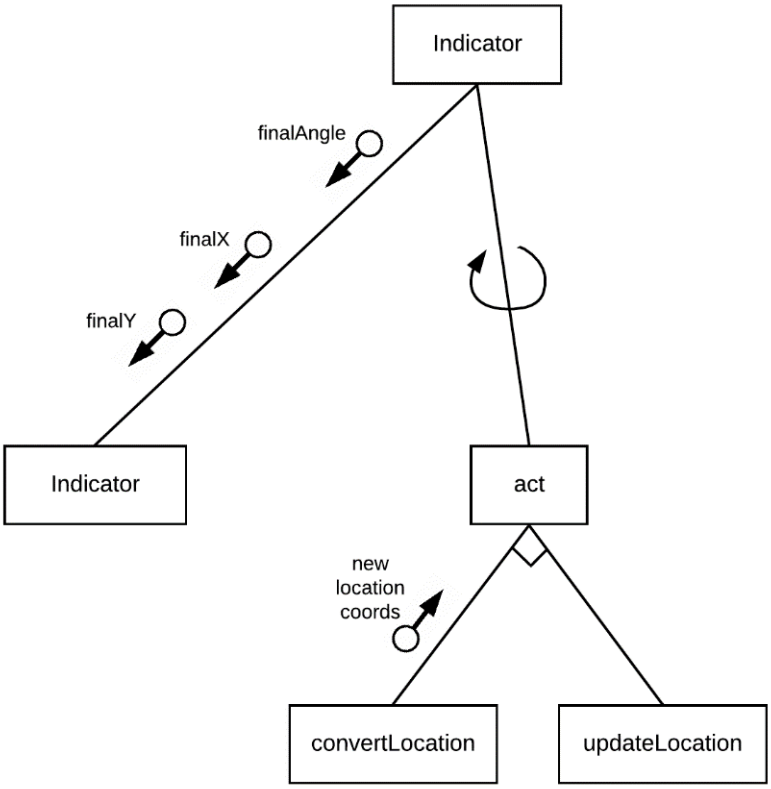
Search bar class



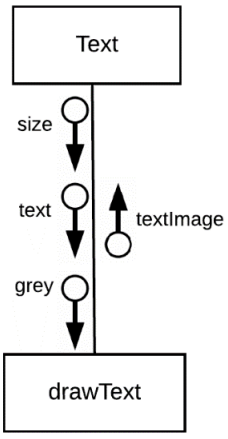
Menu class


Menu
vid
backButton
Menu
vid
backButton
setupMenu
search
createResults
clearSearch
availableLetters
createKeyboard
roomInfo
readFromFile
createServices
removeButtons
OnlineHelp?
createBackground
createAmenities
removeButtons
OnlineHelp?
createBackground
createMenu
removeButtons
OnlineHelp?
createBackground
createHeader

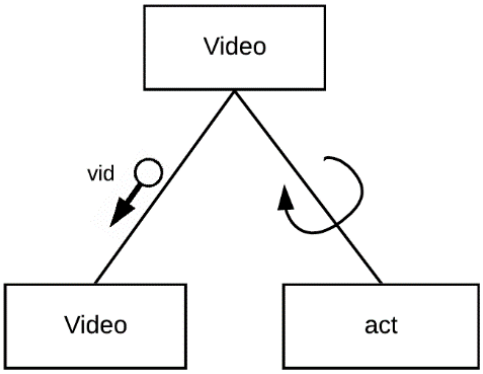
Indicator class



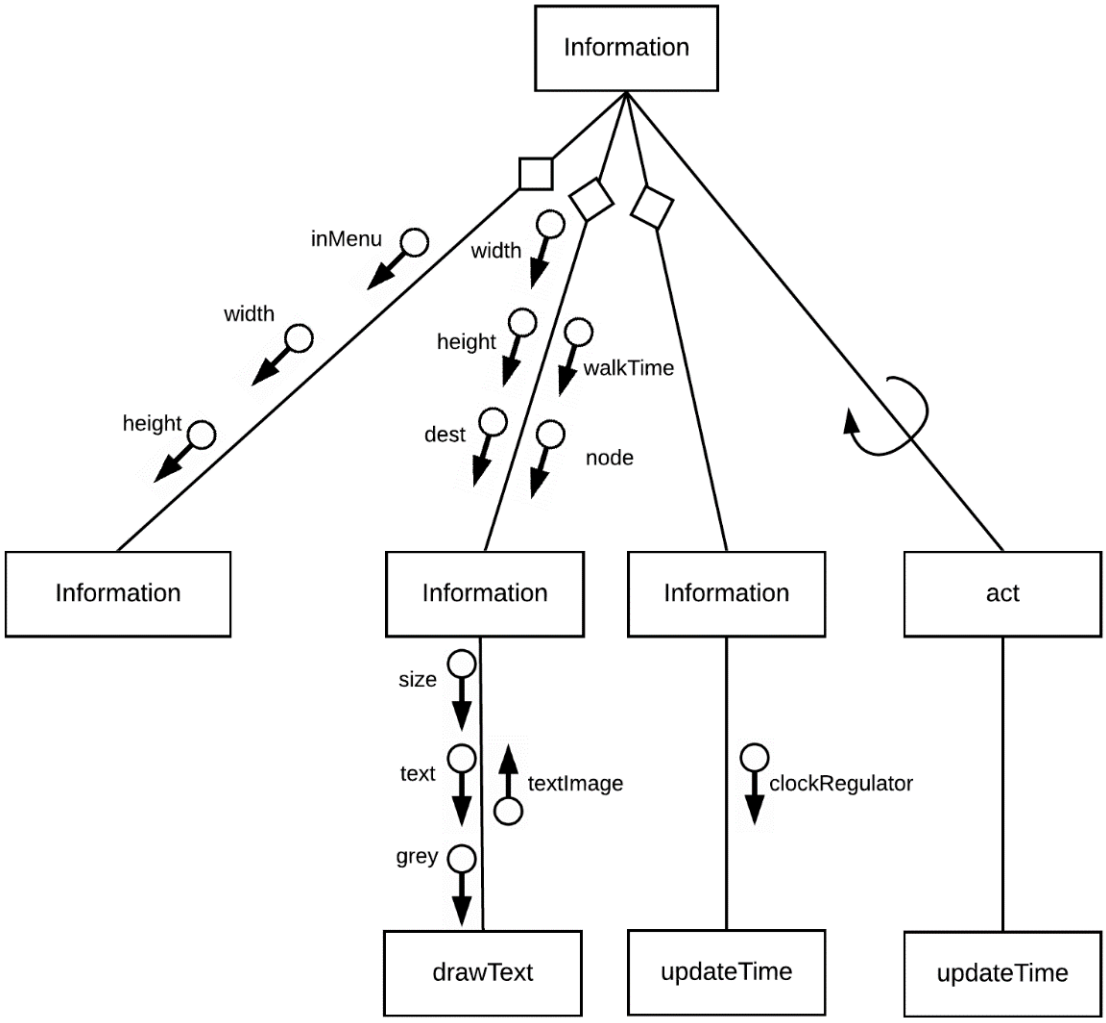
Text class



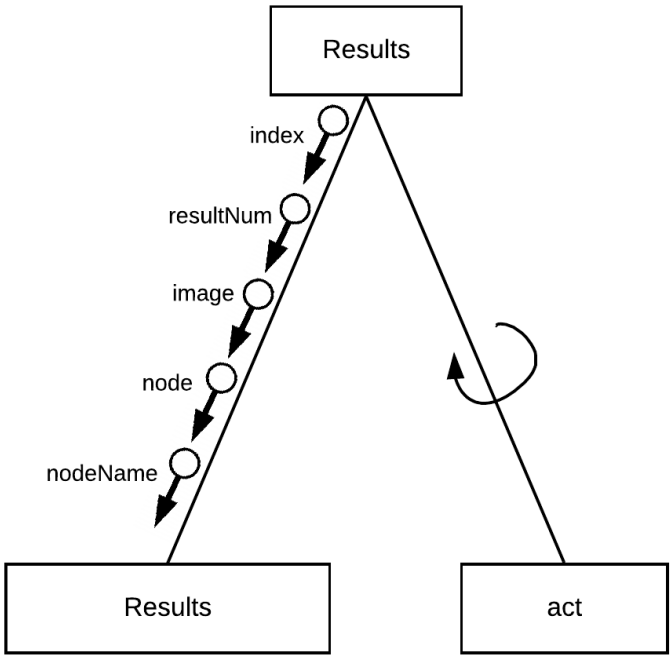
Video class



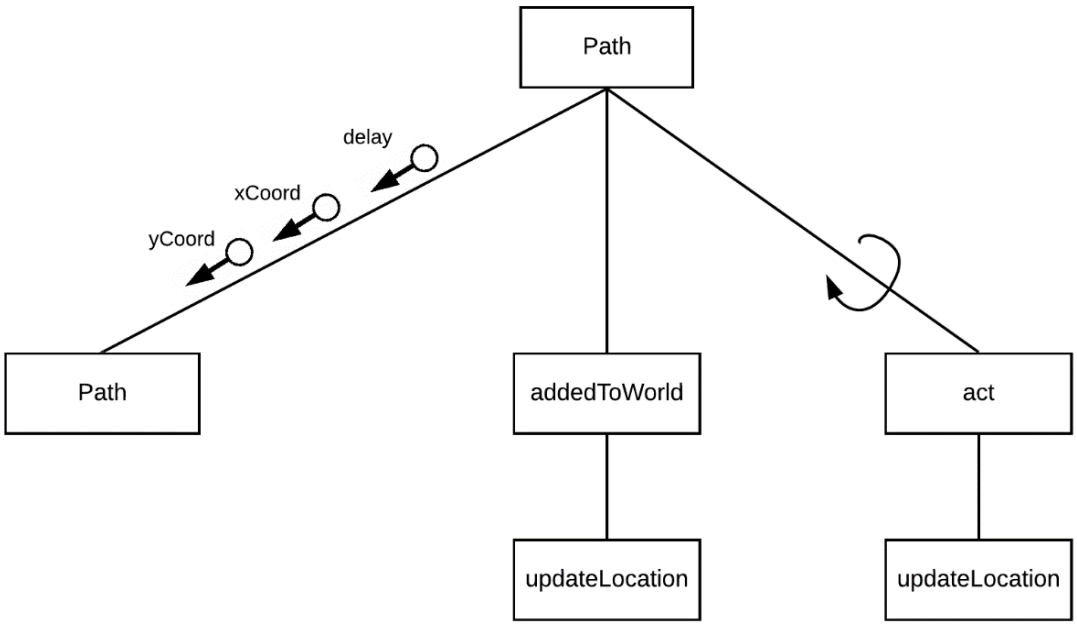
Information class



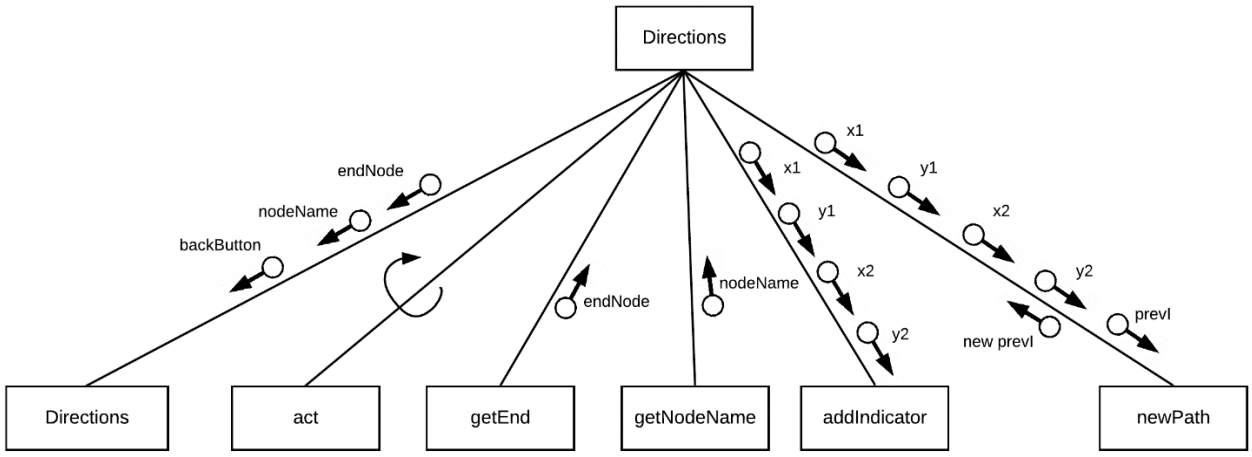
Results class



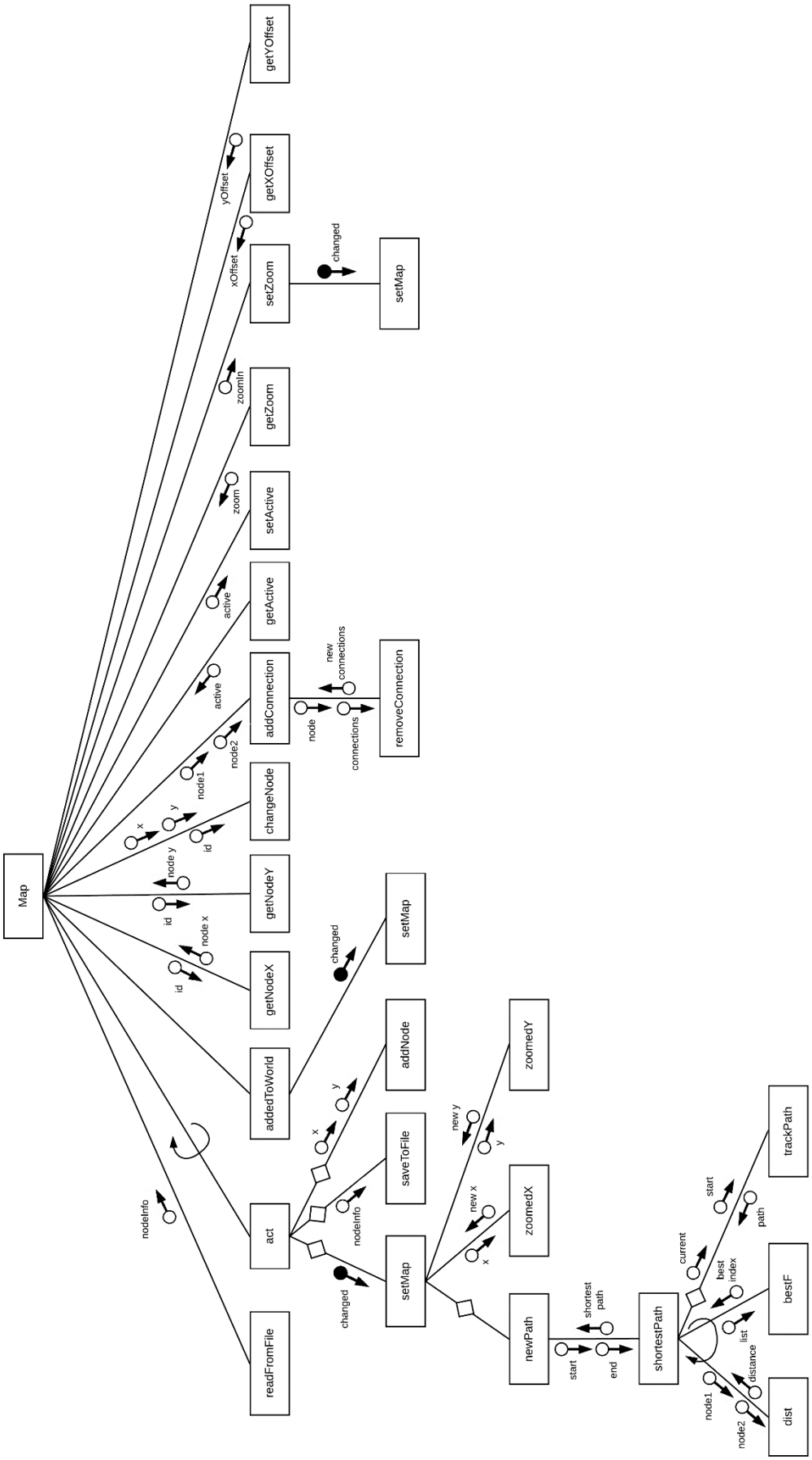
Path class



Directions class



Map class


Map
getYOffset
getXOffset
setZoom
changed
setMap
yOffset
xOffset
getZoom
zoomIn
setActive
zoom
getActive
active
addConnection
new connections
removeConnection
node
connections
changeNode
node1
node2
getNodeY
node y
getNodeX
node x
addedToWorld
changed
setMap
id
x
y
act
addNode
saveToFile
nodeInfo
changed
readFromFile
nodeInfo
zoomedY
zoomedX
new y
new x
newPath
start
end
shortest path
shortestPath
current
best
index
list
distance
node1
node2
dist
bestF
start
path
trackPath

Installation Guide

Hardware Requirements

To run the program, the following **input** devices are required:

- A touchscreen is needed to navigate through different screens in the program and to operate the map interface. Tapping buttons executes different functions in the program including navigating through menus, searching, and zooming. Dragging across the room search results allows the user to scroll through them and dragging on the map will also allow for its repositioning.
- Optionally, a mouse/touchpad may be used to replace the touchscreen inputs if it is preferred. The same functions within the program can be used with mouse inputs instead.

The following **output** devices are also required:

- A computer monitor or other type of display (e.g. a projector screen) is required to present the graphical user interface to the user and allow for clear directions to be shown.

Furthermore, in order to run Oracle's Java 8 (installation guide provided below) and thus run the software solution, the user must meet the following hardware requirements:

- RAM: 128 MB
- Disk space: 124 MB for JRE; 2 MB for Java Update
- Processor: Minimum Pentium 2 266 MHz processor

Further information about the hardware requirements of Java 8 can be found at: https://docs.oracle.com/javase/8/docs/technotes/guides/install/windows_system_requirements.html.

Benchmarking

A benchmark test was conducted to justify such minimum hardware requirements as shown above. Running the program on weaker hardware will introduce excessive load times and thus would be detrimental to usability.

Process used to benchmark

In order to benchmark the program, specialised code was written to time the execution of various modules. After timing many of the main modules, I concluded that the programs execution time hinges on the processing of the main menu screen and the map. Thus, the code shown below was used to time how long both the menu and map took to load. This code uses Java's nanoTime() function to compare the time before and after executing the function and outputs this value to the developer for analysis. These results were collected and rounded to 6 decimal places then tabulated as shown below.

The purpose of these tests is to compare the optimal performance of the program with its performance on a lower spec device and thus justify these minimum hardware requirements set out above.

```
public Menu() {
    super(width, height, 1, false);
    long startTime = System.nanoTime(); // This is used for conducting benchmarking tests

    Greenfoot.setSpeed(60);
    setupMenu(0, false); // Load main menu with idle video hidden
    setPaintOrder(Video.class, Information.class, Searchbar.class, Results.class, Button.class);

    long endTime = System.nanoTime();
    System.out.println((endTime - startTime)/1e+9);
}
```

Map creation processes…

Additional code used to test map load time

```
protected void addedToWorld(World world) {
    long startTime = System.nanoTime(); // This is used for conducting benchmarking tests

    end = ((Directions)getWorld()).getEnd();
    setMap(true); // Update map when added to world
    if (developer) { // Add Node object for each node if in developer mode
        for(int i=0; i < nodeInfo.size(); i++){
            getWorld().addObject(new Node(i), zoomedX(nodeInfo.get(i).getX()),zoomedY(nodeInfo.get(
        }
    }

    long endTime = System.nanoTime();
    System.out.println((endTime - startTime)/1e+9);
}
```

Menu creation processes…

Additional code used to test menu load time

Greater PC Specs		
Hardware	**Device**	Huawei MateBook X Pro Signature Edition
	Processor	Intel Core i7-8550U CPU @ 1.80GHz
	Installed ram	16.0 GB (15.8 GB usable)
Software	**System type**	64-bit operating system, x64-based processor
	Touch screen	Touch support with 10 touch points
	Operating system	Windows 10 Home Version 1903
	Java version	Version 8 Update 251

Lower PC Specs		
Hardware	**Device**	Apple MacBook Pro (13-inch, Mid 2012)
	Processor	2.5 GHz Dual-Core Intel Core i5
	Installed ram	8 GB 1600 MHz DDR3
Software	**System type**	64-bit operating system, x64-based processor
	Touch screen	No
	Operating system	macOS Catalina Version 10.15.1
	Java version	Version 8 Update 211

Results of benchmark testing

Test	PC with greater specs (s)		PC with minimal specs (s)	
	Time to load menu	Time to load map	Time to load menu	Time to load map
1	0.044945	0.063683	0.090359	0.119088
2	0.045605	0.057522	0.092082	0.112276
3	0.047024	0.063329	0.099117	0.116517
4	0.044027	0.055643	0.097946	0.110119
5	0.046399	0.059487	0.099274	0.113164
Avg.	0.046	0.060	0.096	0.114

In conclusion, these results were used to specify the minimum hardware and software requirements as shown above. Any less powerful hardware used to run the program are thus not recommended as they may lead to a hindered user experience.

Software Requirements

As per the installation guide below, the program requires that the user install the **Java 8** runtime environment and that the user's current operating system meets the software requirements for Java 8 as shown below:

Windows

- Windows 10 (8u51 and above)
- Windows 8.x (Desktop)
- Windows 7 SP1
- Windows Vista SP2
- Windows Server 2008 R2 SP1 (64-bit)
- Windows Server 2012 and 2012 R2 (64-bit)
- Browsers: Internet Explorer 9 and above, Firefox

Mac OS X

- Intel-based Mac running Mac OS X 10.8.3+, 10.9+
- Administrator privileges for installation
- A 64-bit browser (Safari, for example) is required to run Oracle Java on Mac.

Linux

- Oracle Linux 5.5+1
- Oracle Linux 6.x (32-bit), 6.x (64-bit)2
- Oracle Linux 7.x (64-bit)2 (8u20 and above)
- Red Hat Enterprise Linux 5.5+1, 6.x (32-bit), 6.x (64-bit)2
- Red Hat Enterprise Linux 7.x (64-bit)2 (8u20 and above)
- Suse Linux Enterprise Server 10 SP2+, 11.x
- Suse Linux Enterprise Server 12.x (64-bit)2 (8u31 and above)
- Ubuntu Linux 12.04 LTS, 13.x
- Ubuntu Linux 14.x (8u25 and above)
- Ubuntu Linux 15.04 (8u45 and above)
- Ubuntu Linux 15.10 (8u65 and above)
- Browsers: Firefox

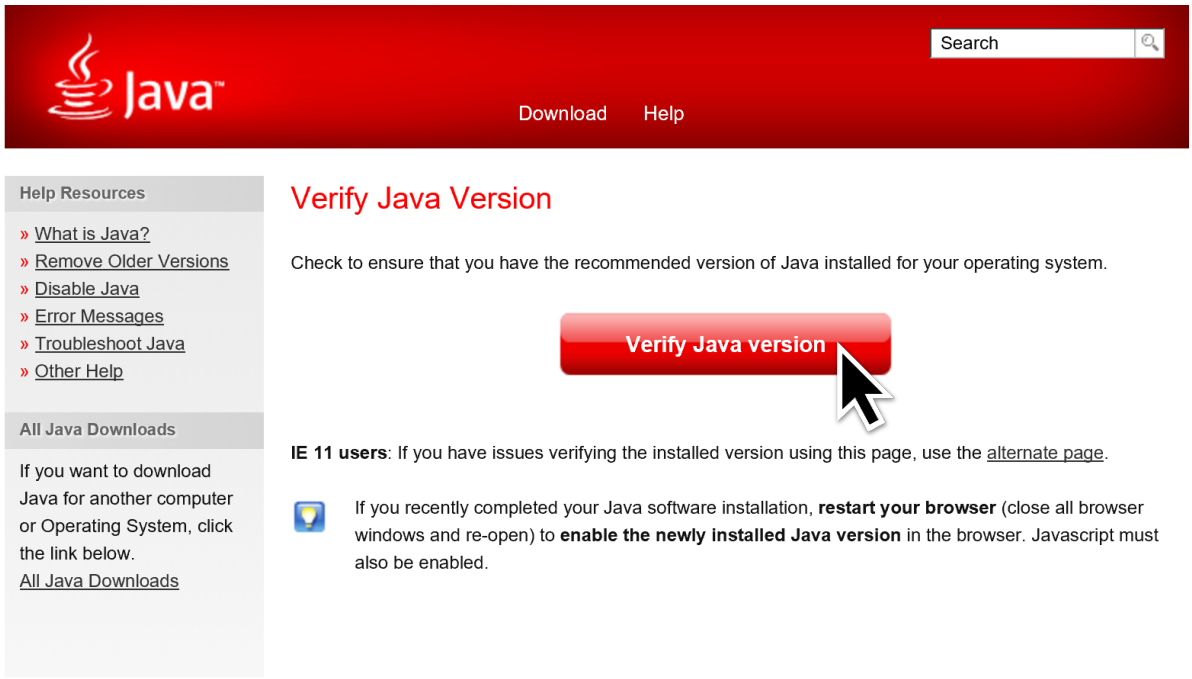
The user must also have a valid executable jar file "CollegeConnector.jar" for the program to run properly.

Further information about Java 8 software requirements can be found at: https://docs.oracle.com/javase/8/docs/technotes/guides/install/windows_system_requirements.html.

Java Installation Process

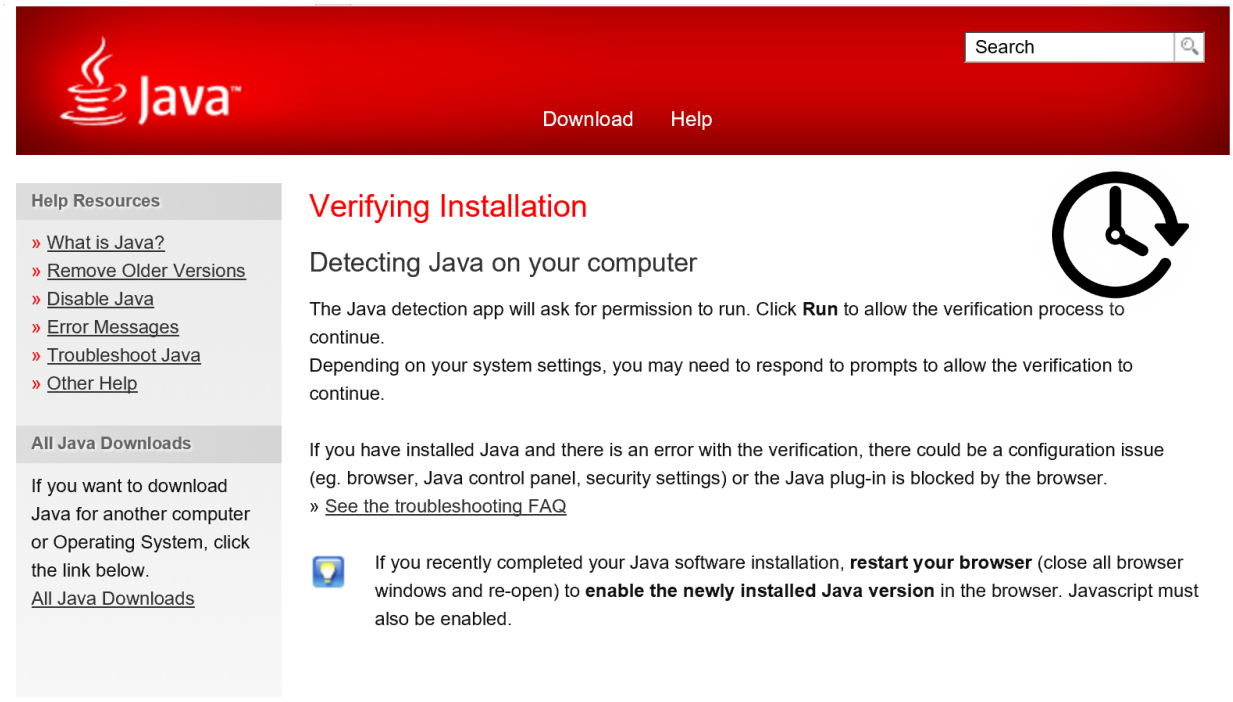
Step 1

To check if the latest version of Java 8 is currently installed on the system visit https://java.com/en/download/installed8.jsp on Internet Explorer, Safari or Firefox (Chrome and Edge do not support Java applications) and click the “Verify Java version” button.



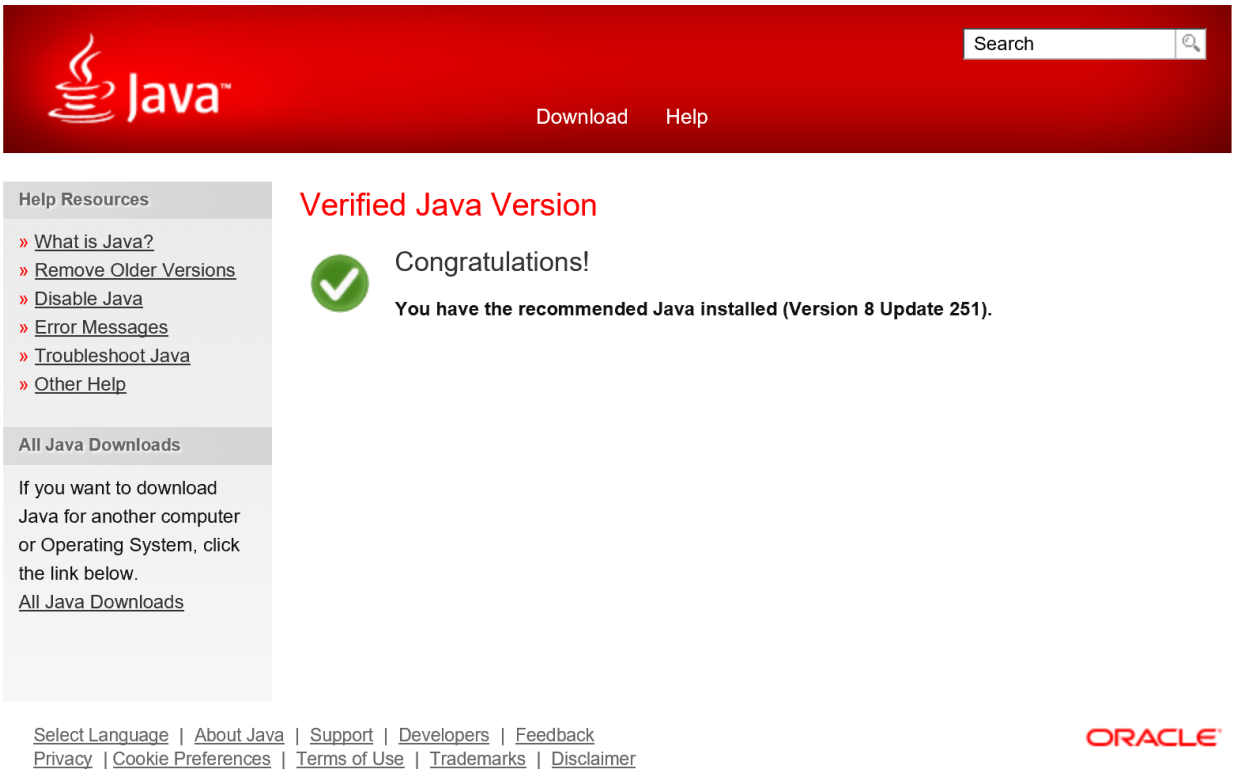
Step 2

If the website indicates that Java is not currently installed, or the current version of Java is out of date, follow the instructions provided to download and install the latest version to ensure the application can run properly.



Step 3

After Java has been successfully installed, repeat Step 1 to verify successful installation. The following screen should be displayed if the installation was successful.



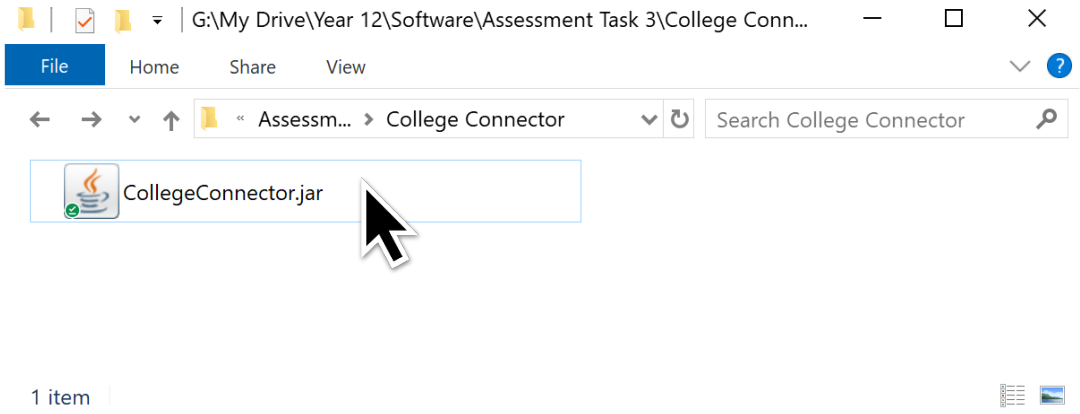
Step 4

Java 8 has now been installed onto the system and College Connector can now be run as per the steps below. Information about your install can be found using Java's About Java application – this is not necessary for the College Connector installation.

College Connector Installation Process

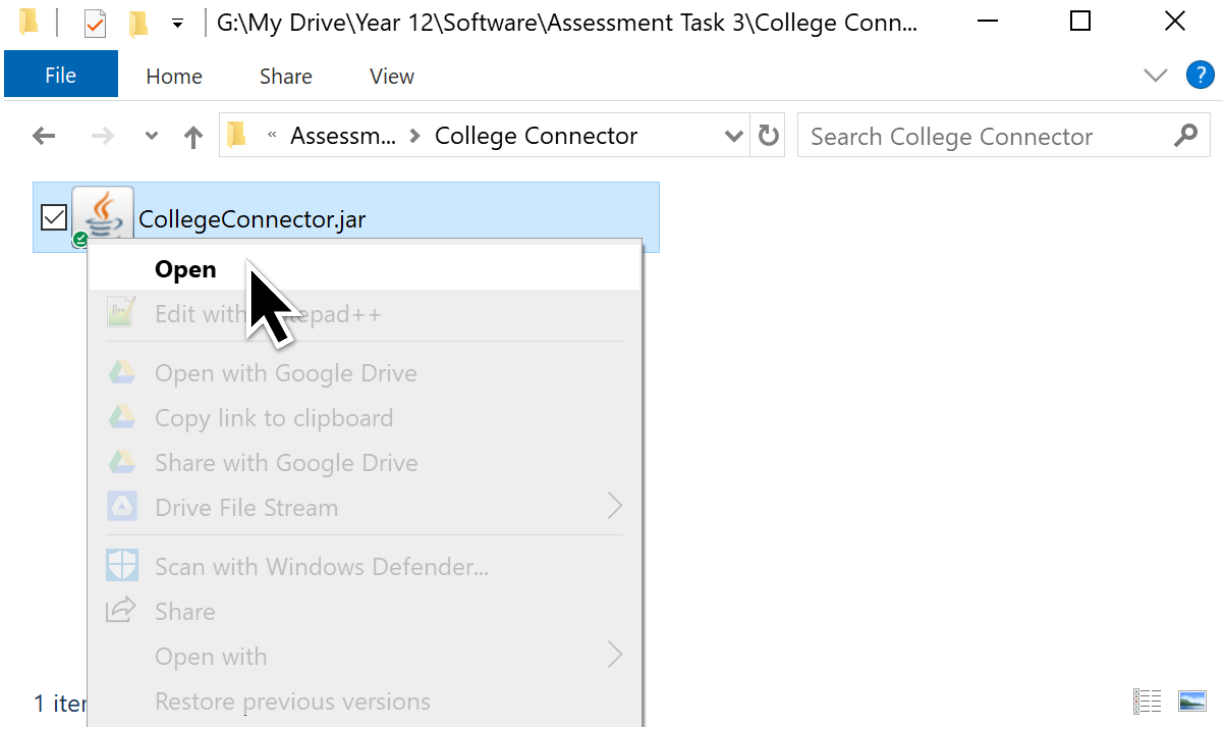
Step 1

Ensure that the correct executable file has been downloaded onto the computer and its location is known. The file is named 'CollegeConnector.jar' and should have a JRE logo as its icon as it is an Executable Jar File.



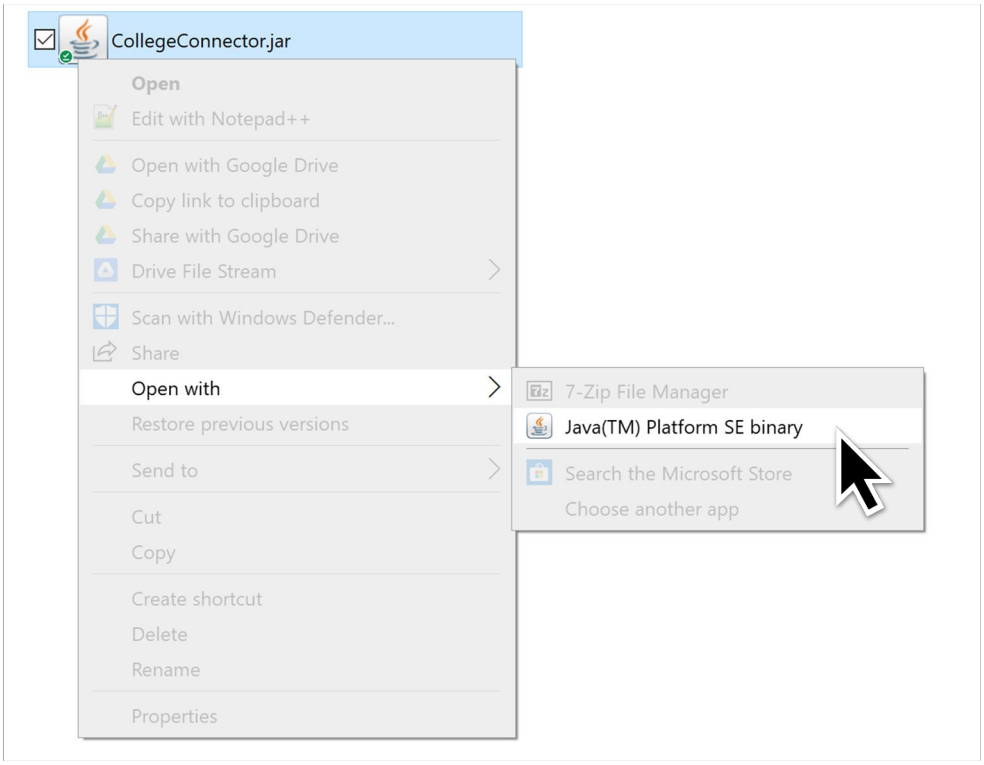
Step 2

If Java 8 is the default program to run '.jar' files, double click the executable file to run the software. Please wait a few seconds for the program to launch and a window to appear. The window shown in Step 4 should now be running. If you are unable to double click the program, instead, right click then select 'Open' in the context menu. If this option is not available, please ensure you have the latest version of Java 8 running on your system by following the Java Installation Guide above.



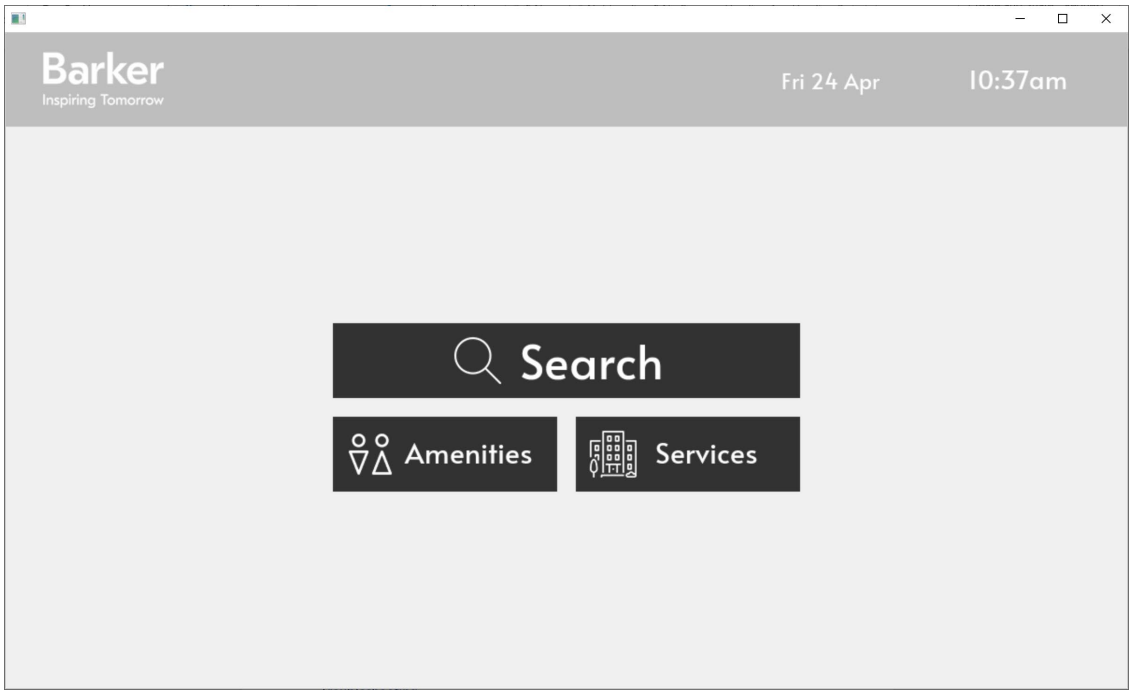
Step 3

Otherwise, right click to open the context menu and hover over the “Open with” submenu and select *Java(TM) Platform SE binary*. If this option is not available, ensure that you have installed the latest version of Java 8 (instructions found above under ‘Java Installation Process’).



Step 4

The program should now successfully run in a new window and can be used for guidance around the school as per the tutorial below.



Troubleshooting

If the program is not functioning properly, it is recommended that the following steps are performed to fix any potential issues or conflicts.

Verify that your computer system meets the requirements for Java 8

As per the hardware and software requirements listed above, your computer must have at least: 128 MB of RAM, 126 MB disk space and minimum Pentium 2 266 MHz processor for the Java Runtime Environment to properly function.

Ensure that the current executable file is being run: CollegeConnector.jar

The following executable JAR file must be run, CollegeConnector.jar. The program does not need to be installed onto the computer – it can simply be run from the file provided using the latest version of Java 8.

Ensure that the latest version of Java 8 is installed on your computer

Please follow the Java Installation Process instruction above to install and verify that the latest version of Java is present on your computer system.

Ensure that your monitor is set to the correct colour settings and screen size

If the program is not displaying correctly on your display, check the colour settings to ensure that they are set appropriately for the program. Furthermore, you must ensure that your screen size is set to the recommended size for your monitor. It is recommended that the software is used on a screen size greater than 1200x700 pixels to maximise user experience.

Ensure that the touchscreen input is working correctly with your display

If you are unable to select buttons on the program or scroll, you must make sure that the touchscreen input used is properly interfacing with your operating system. To further troubleshoot this issue, connect a mouse to the computer and test that it functions properly.

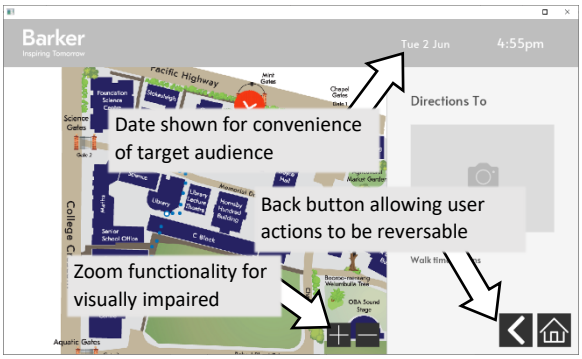
Ensure that sufficient memory is available on the system

If the program does not run smoothly or closes unexpectedly, ensure that enough RAM is available to the program. This issue can be checked by opening Task Manager on Windows or Activity Monitor on macOS while the program is running and ensuring that the memory is not being used excessively (consistently above 90% usage). It is recommended that the computer has over 8GB of RAM to ensure proper operation of the program and enough graphics memory to help the rendering of the user interface.

If issues persist, please contact the developer at CollegeConnector@barker.college.

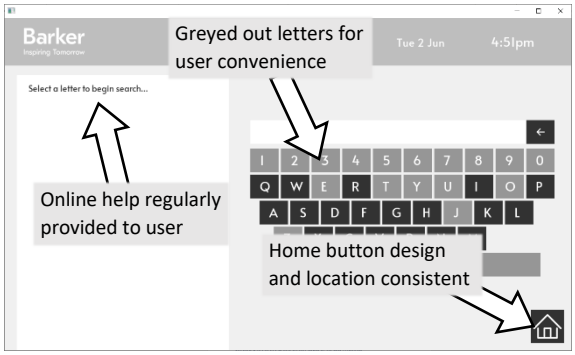
Human-Computer Interface

The human computer interface involves the various ways users can interact and communicate with a system. In my project, a well-designed touch screen user interface is utilised to attain maximum **ease of use** and **accessibility** for users of various cultural, economic and social backgrounds. As such, a multitude of social and ethical issues have been considered and catered for.



Well-designed user interface displaying helpful information and directions to "Bowman Field".

The **target audience** is quite broad for my software, including people from ages 12 to 19 and 40 to 60 of diverse backgrounds. It accommodates the needs of new students and parents that may require to use it, and thus, it must conform to the standards of similar software that these users are familiar with. For example, in the screenshot to the left, a header showing the Barker College logo in the top left and the date and time in the top right is easily recognisable. These elements were designed with easy to read fonts and colours and placed in the expected positions, consistent with other programs of this nature. These measures ensure that users with colour blindness, impaired vision or other language processing disorders like dyslexia can use the program without difficulty. This is aided by the zoom buttons which have been included on the map, much like the design of these buttons in similar programs. The 'back' and 'home' icons are clearly displayed and are consistent with most other programs. These elements help my target audience to be able to easily recognise the actions required to execute desired functions.

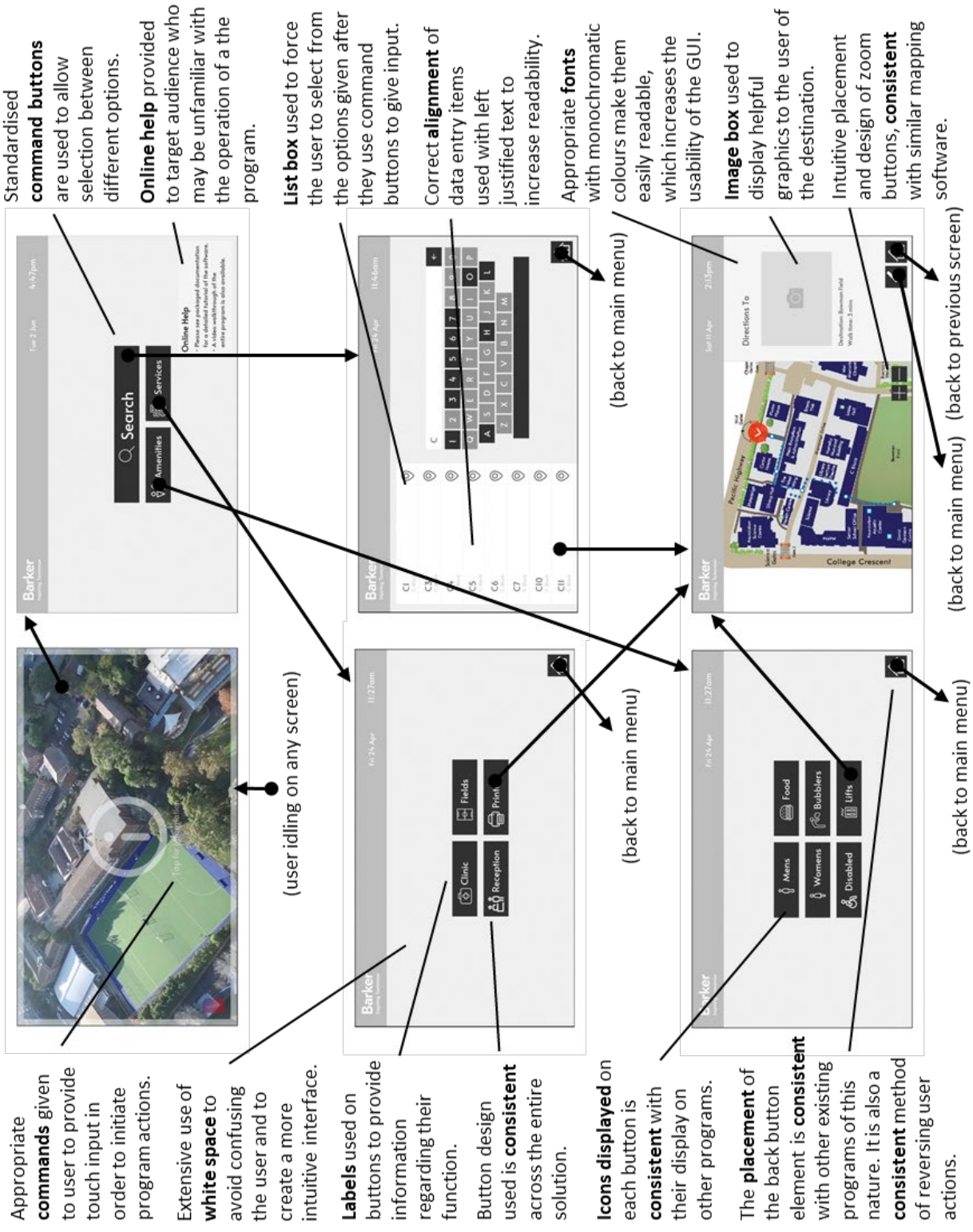


Online help clearly displayed in search menu.

Online help is also available throughout the program to assist my target audience to use it properly. The prominent "Touch for Information" message on the idle screen serving to invite users on to interact with the screen and begin their navigation. Online help is also present in the form of an installation guide and tutorial as packaged with the software which allows for less technologically inclined users to install and use the software efficiently and effectively. Users are provided a notification as to the location of this documentation on the main menu. Furthermore, users are prompted by a "Select a letter to begin search…" message when the search screen is opened. This tells the user the actions expected of them to correctly function the keyboard and search process. Additionally, the greying out of letters which do not yield search results also helps users to find their destination. Thus, the program is highly suited to young and older audiences of any socioeconomic background due to the careful considerations of the graphical user interface design as well as the easy way to interact with such screen elements.

Consideration of social and ethical issues in the user interface

This diagram displays my close adherence to my Human-Computer design carefully designed in Assessment Task 2 in the implementation of my project. The annotations on screenshots of the final software demonstrate my consistent consideration of social and ethical issues regarding my target audience and implementation of online help.



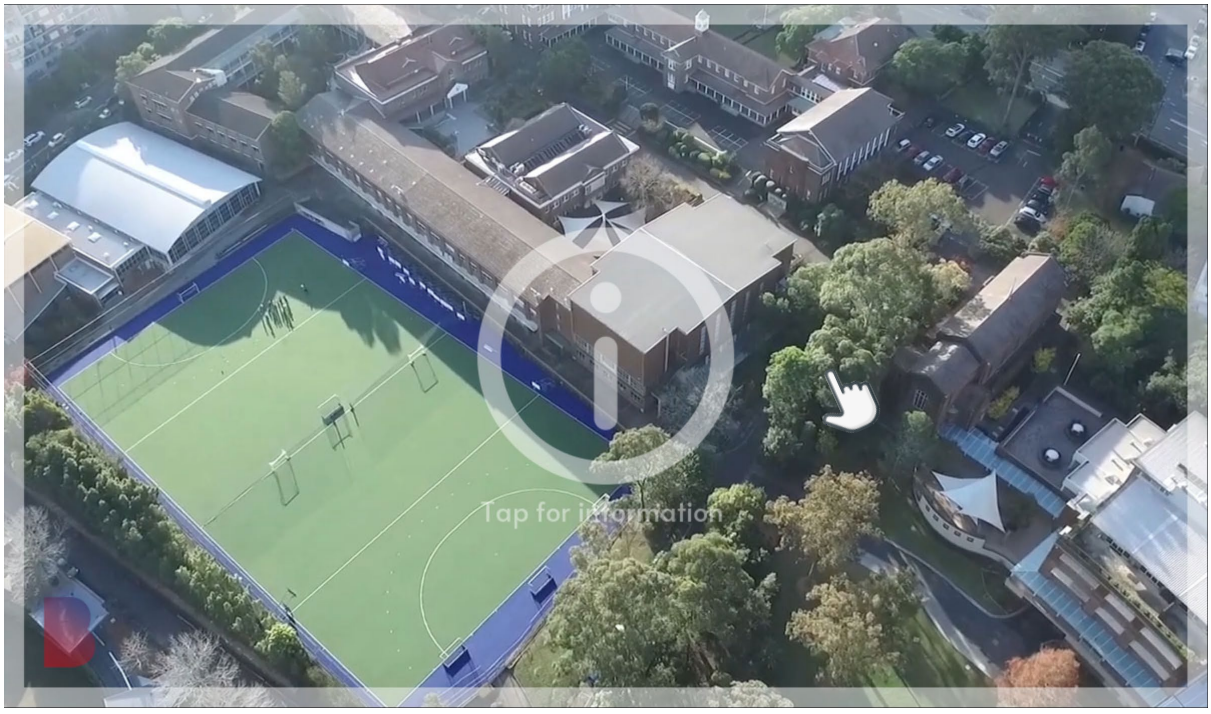
Software Tutorial

A software tutorial includes the flow of the program, a detailed guide to the different commands in the main menu and a step by step outline of how the system is properly operated. It serves as a guide to new users of the program covering the various features of it and utilising screen shots where appropriate.

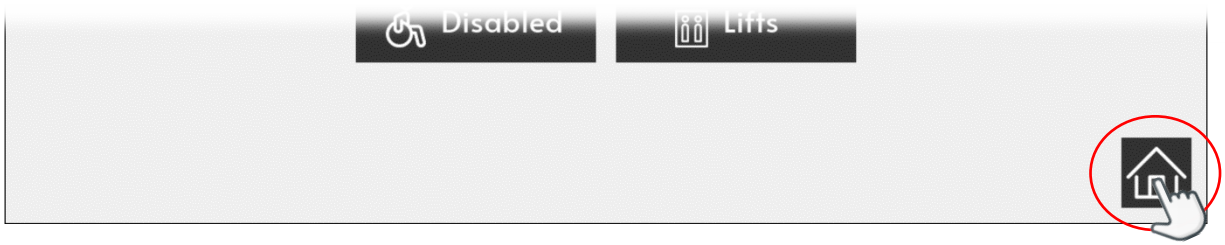
This tutorial guides the user on how to access and use each function in the program to maximise their experience. There are three main screens within the program that must be understood such as to effectively operate the College Connector. Firstly, the idle screen is the starting point, then the menu screens allow for selection of a desired location within the Barker College campus, and finally, the directions screen displays the required information to allow for easy navigation to the selected location. How to properly use and navigate between each of the screens is explained below.

1. Idle Screen Tutorial

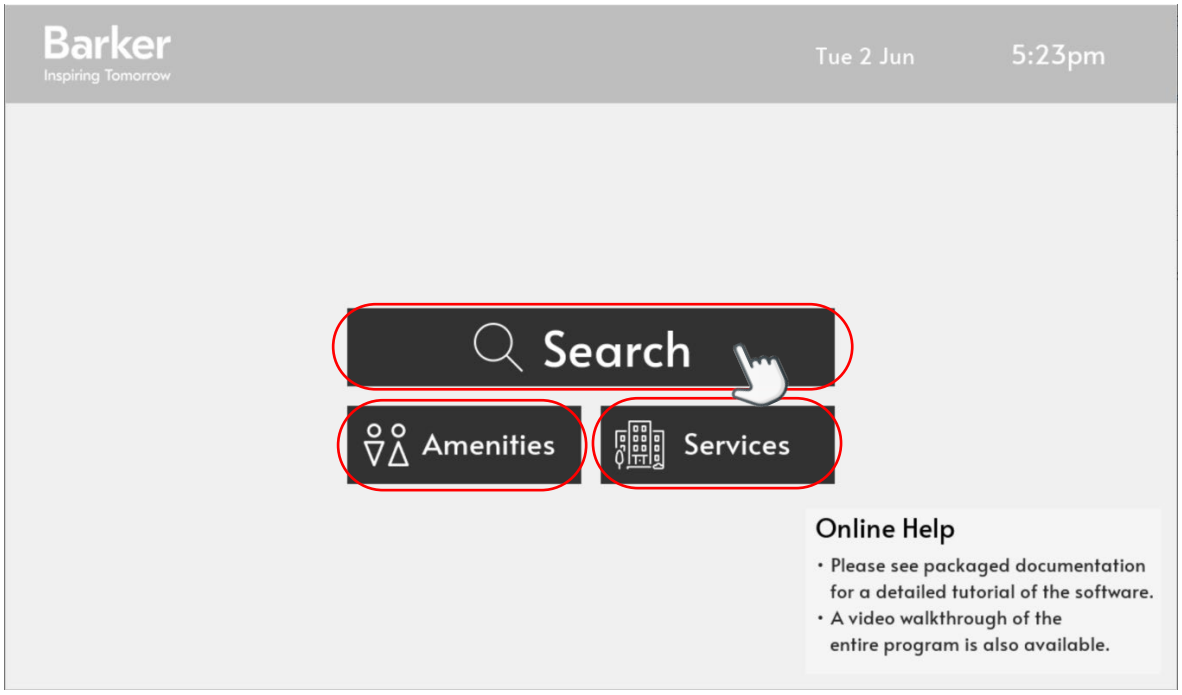
When no one is using the program for 20 seconds, the idle screen will show and play a promotional video of different activities and places around school on repeat. Tapping anywhere on the screen will then display the main menu screen.



To navigate back to the main menu from any other screen within the program, tap the home button in the bottom right corner as seen below.



2. Menu Screens Tutorial



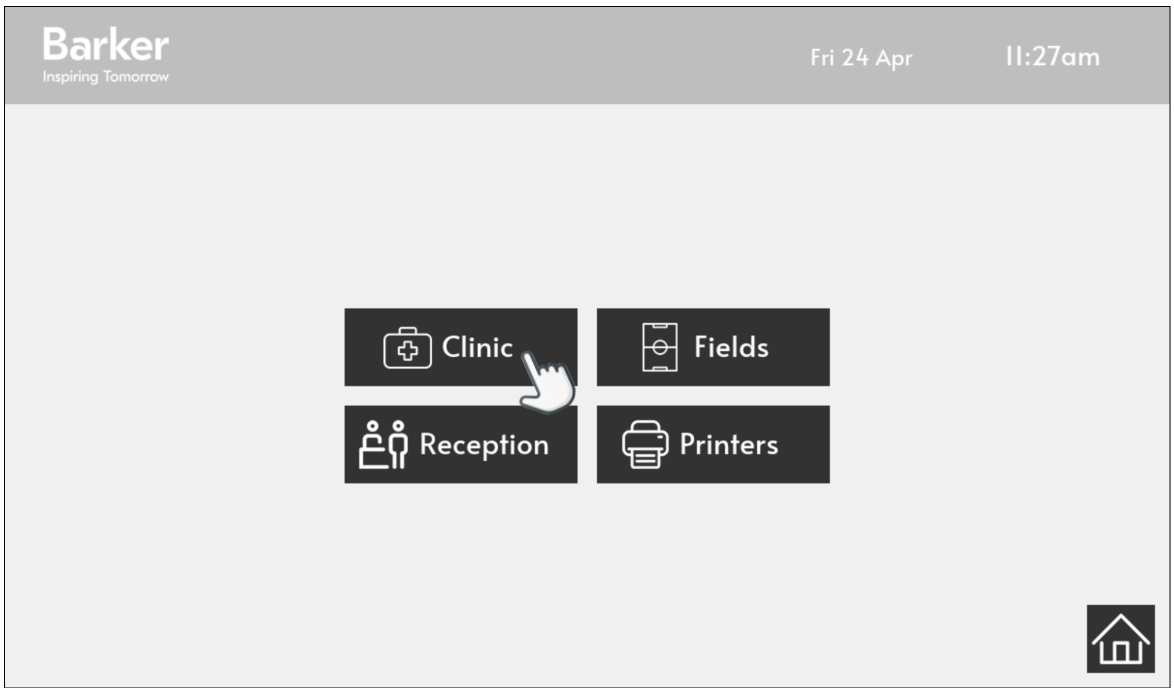
When on the main menu screen, three options are available to make finding directions to different types of places around the campus easier. These options, marked in red above, can be accessed by simply tapping on the desired button using the touchscreen.

Note that the date and time is displayed in the top right at all times.

Option 1: *Amenities*. This option allows for you to navigate to the nearest amenity of the various options listed as shown below.

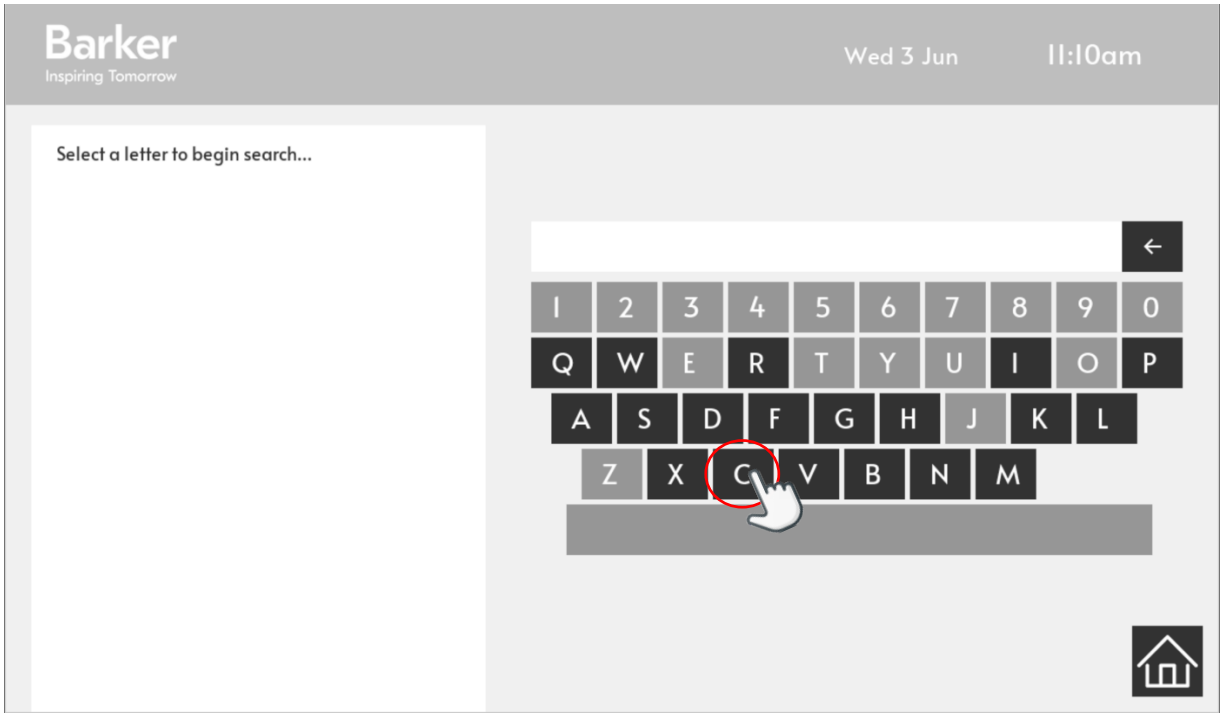
Each button is a different amenity that can be navigated to. Simply tap on any of the buttons to get directions to the closest amenity of that type.

Option 2: *Services*. Similar to the amenities screen, this option allows for you to quickly navigate to the closest of each of the services provided, shown below.

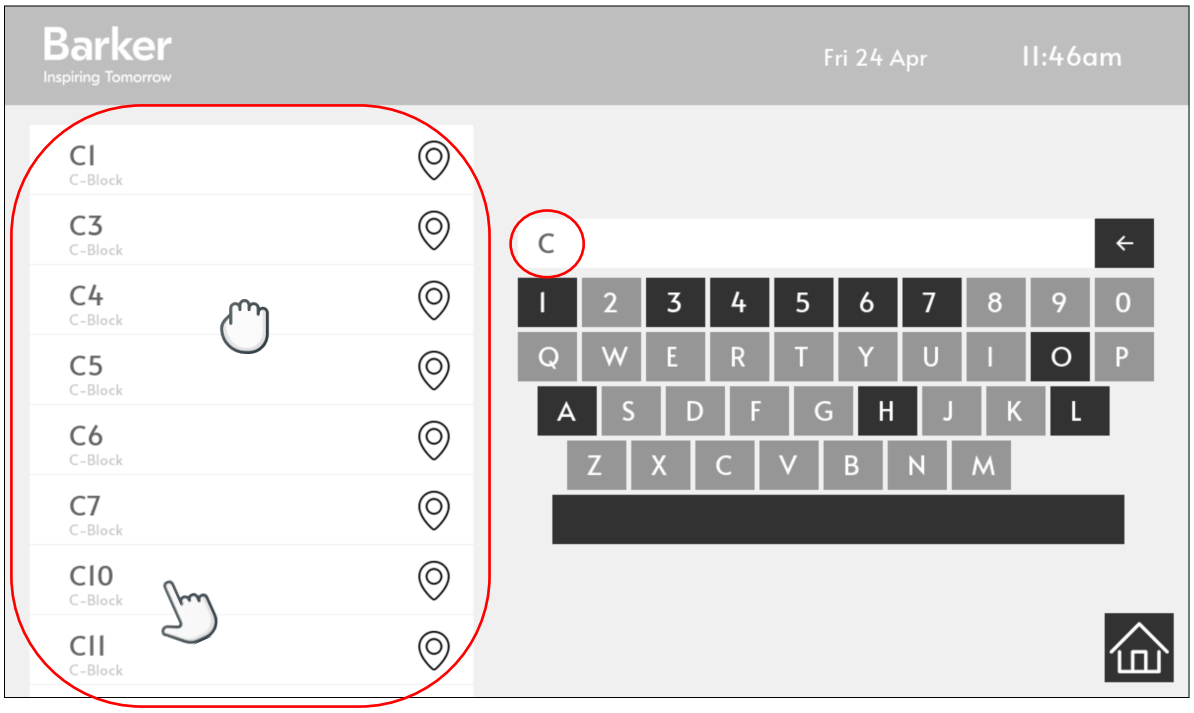


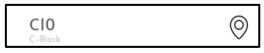
As per the previous option, you can simply tap on any of the buttons to get directions to the closest service desired.

Option 3: *Search*. The search screen allows for a more specific location or room search for anywhere in the middle and senior schools. Letters that can be pressed are displayed as darker grey on the keyboard to the right of the screen. Lighter, greyed out letters cannot be pressed as they do not produce a searchable term. Simply tap any of the available letters to search for the desired room.



For example, searching for the classroom “C10”, you can simply press the letter “C” in the keyboard and all results matching this query will be displayed as shown below.



Displayed to the left of the screen is the results for the search query in the white textbox shown above the keyboard. If this list is too long to be displayed on the screen, it can be scrolled through by simply dragging it vertically upwards across the screen, revealing the results below. Furthermore, clicking on a specific search result (e.g. the “C10” result will 

show the directions screen to allow for navigation. To undo your search, click the left arrow symbol (←) to delete the last letter in your search. Once you have selected your desired destination, you will be presented with the directions on the directions screen as shown next.

3. Directions Screen Tutorial

The directions screen is used to gather necessary information about the destination and the quickest path you can take to get there. A sample directions screen is displayed below:

To the left of the screen is the interactive map which initially displays the direction you must turn to begin navigation (indicated by the flashing red arrow) and the path to get there. This path is indicated with blue dots that flash in the direction of the path.

Zoom in (+) and zoom out (–) buttons are provided towards the bottom of the map to allow for you to scale the map.

Dragging anywhere on the map will allow you to adjust the map view for a greater understanding of the campus.

To the right of the screen is more information on most destinations around the school; including an image, its name, and the estimate time to walk there. If no current image exists, then the placeholder image will appear (as shown on the left).

The two buttons in the bottom right hand corner allow for you to go back to the page you were previously on. The left pointing arrow (<) is the “back” button which brings you to the previous page you were on (search, amenities or services screen) and the home button brings you to the main menu as it does from any other screens.

Once you have observed the directions to your destination, simply walk away from the kiosk in the direction indicated by the red arrow. If the screen is not pressed for 20 seconds, the idle screen will appear again and reset the program for use by the next user.

9.2.4 Testing and Evaluating of Software Solutions

System Testing Test Data Documentation

In order to ensure the correct operation of all functions and methods within my software solution, each has been thoroughly tested with an array of test input data. The expected output to be returned by each method and the actual results are compared below.

System Testing

Different computer systems have been tested to ensure that the program runs efficiently and effectively on various forms of devices and by different users in my target audience. Java 8 supports a wide array of devices as per the installation guide above, so it is expected that such systems perform consistently. The different systems tested are shown below:

Test 1		
Hardware	**Device**	Huawei MateBook X Pro Signature Edition
	Processor	Intel Core i7-8550U CPU @ 1.80GHz
	Installed ram	16.0 GB (15.8 GB usable)
Software	**System type**	64-bit operating system, x64-based processor
	Touch screen	Touch support with 10 touch points
	Operating system	Windows 10 Home Version 1903
	Java version	Version 8 Update 251
Results		
Comment	This test was conducted to check the functioning of the executable file on a high-powered laptop device with fully updated software. The program ran flawlessly, with unnoticeable loading times and full functioning. The touchscreen interface performed as expected with almost instant feedback within the software.	
Recommendation	The program performed as expected so there are no new recommendations for the software.	

Test 2		
Hardware	**Device**	Microsoft Corporation Surface Laptop
	Processor	Intel Core i5-7200U @ 2.69 GHz
	Installed ram	8 GB (7.39 GB available) DDR3 SDRAM 931MHz
Software	**System type**	64-bit operating system, x64-based processor
	Touch screen	Yes
	Operating system	Microsoft Windows 10 Pro (64-bit)
	Java version	Version 8 Update 251
Results		
Comment	This device had Windows 10 Pro edition installed and had less ram than the first device tested. Loading times for the main functions were barely noticeable and the touch screen performed consistently.	
Recommendation	The program performed as expected so there are no new recommendations.	

Test 3		
Hardware	**Device**	Apple MacBook Pro (13-inch, Mid 2012)
	Processor	2.5 GHz Dual-Core Intel Core i5
	Installed ram	8 GB 1600 MHz DDR3
Software	**System type**	64-bit operating system, x64-based processor
	Touch screen	No
	Operating system	macOS Catalina Version 10.15.1
	Java version	Version 8 Update 211
Results		
Comment	A completely different device and operating system which had inferior computing power (older generation processor and less ram) was utilised in this test to ensure that the program would function properly. The program functioned as expected with a slight delay when clicking buttons.	
Recommendation	The delays experienced in the testing indicate the program's resource intensive functioning due to its dependence on the Greenfoot architecture. It is recommended that efforts are put in to lower the ram and processing power used to execute operations in the program. This issue does not jeopardise the user experience as the delay was essentially unnoticeable thus, any changes should not sacrifice functioning of the program.	

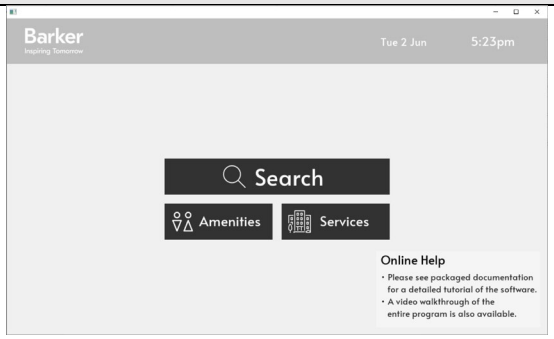
Test 4		
Hardware	**Device**	Huawei MateBook X Pro Signature Edition
	Processor	Intel Core i7-8550U CPU @ 1.80GHz
	Installed ram	16.0 GB (15.8 GB usable)
Software	**System type**	64-bit operating system, x64-based processor
	Touch screen	Touch support was disabled for this test
	Operating system	Windows 10 Home Version 1903
	Java version	Version 8 Update 201
Results		
Comment	An older version of Java 8, which precedes a larger update by Oracle, was used to test the functioning of the software on older systems. The program ran stably as expected with no errors and without any noticeable differences compared with the latest software installed.	
Recommendation	The program performed as expected so there are no new recommendations.	

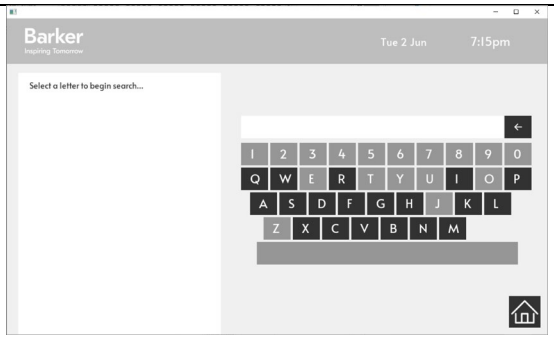
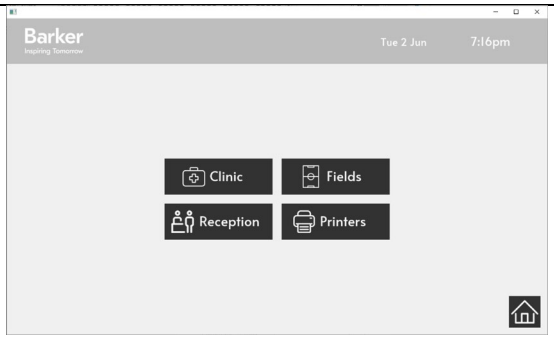
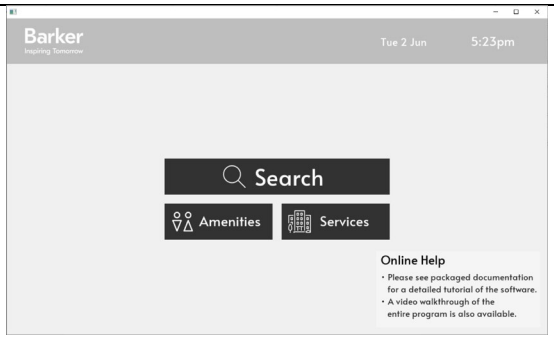
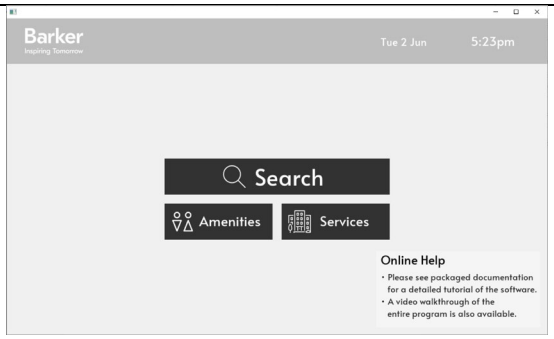
***Note**: When conducting the systems level testing, an identical set of actions were performed to ensure the correct operation of every function in the software. These actions are congruous to the tutorial found above. During the tests, three locations were inputted into the search function: "Bowman Field", "IT11" and "K402". This allowed for every possible pathway in the program to be checked for errors on different systems. Different users in my target audience were told to complete these steps and each different person was able to conduct the tests appropriately and with ease.*

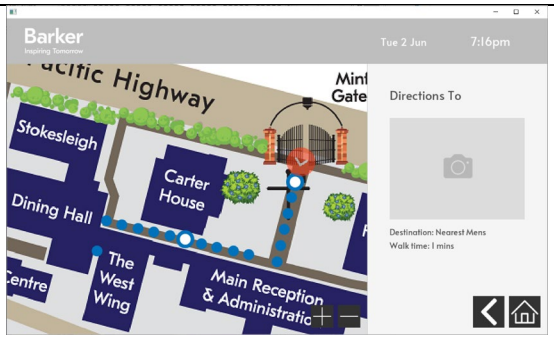
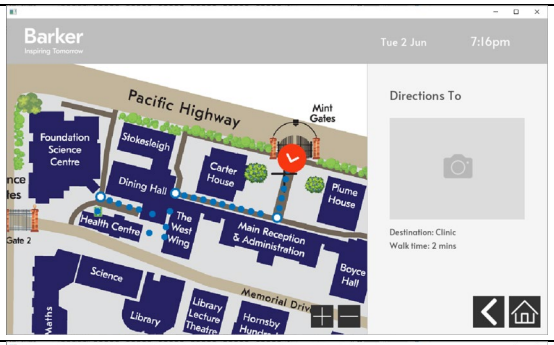
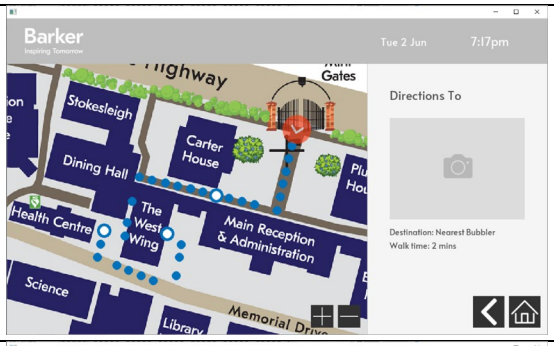
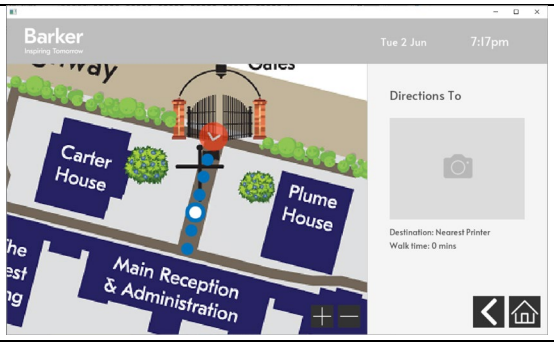
Program Testing

In program testing, actions performed represents the test data into the program. This type of testing is used to assess the interactions between different classes and modules as the program functions as while. Thus, a variety of action were performed to ensure the program performs as expected and appropriate action was pursued in test failure.

Main menu testing

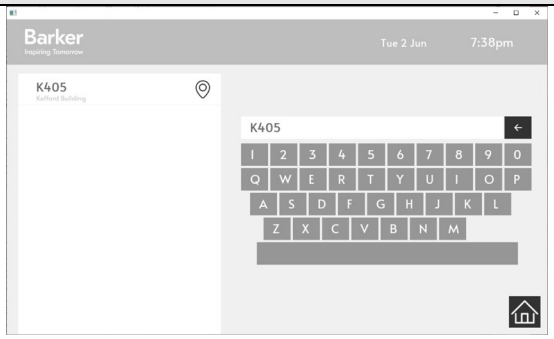
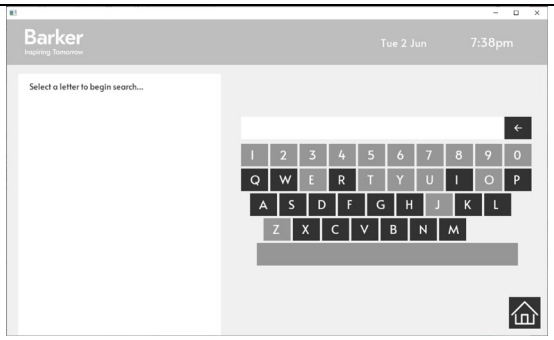
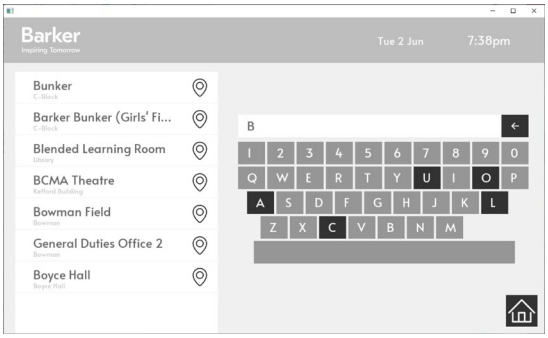
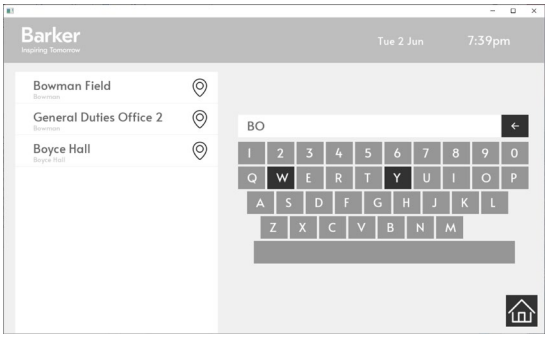
Actions performed	Expected output	Output	Result
Program opened	Main menu shows		Pass
Program opened Waited 25 seconds	Main menu shows then idle screen shows		Pass
Program opened Waited a few minutes	Time in top right should change to correct time		Pass
Program opened Waited 25 seconds Touched on screen	Main menu shows Idle screen shows Main menu appears again		Pass

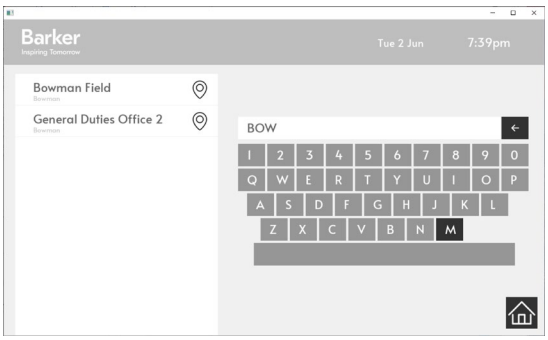
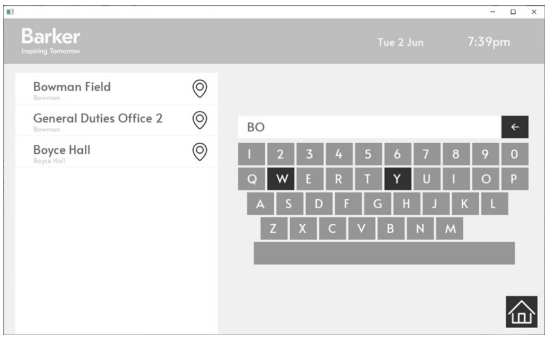
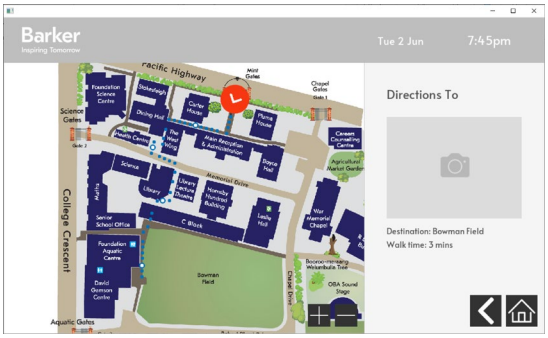
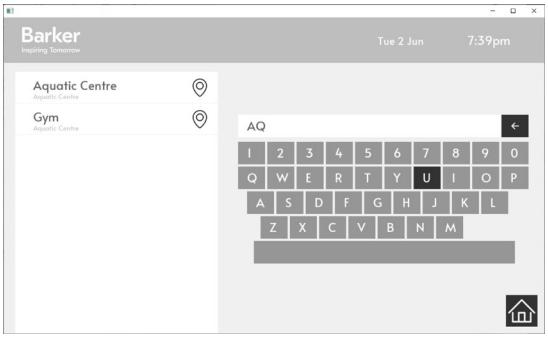
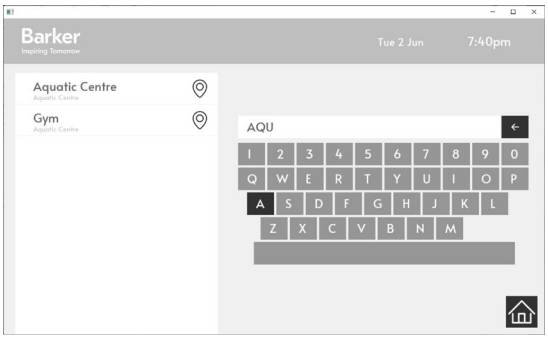
Program opened “Search” button pressed	Search menu appears		Pass
Program opened “Amenities” button pressed	Amenities menu appears		Pass
Program opened “Services” button pressed	Services menu appears		Pass
Program opened “Amenities” button pressed Home button pressed	Main menu appears		Pass
Program opened “Services” button pressed Home button pressed	Main menu appears		Pass

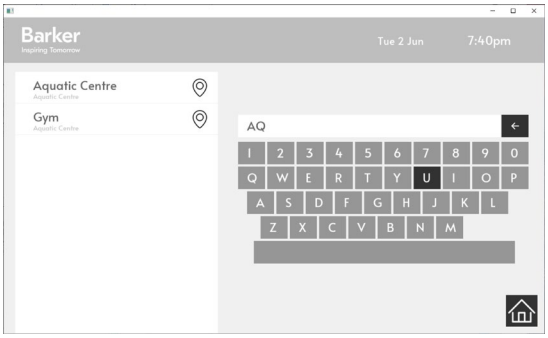
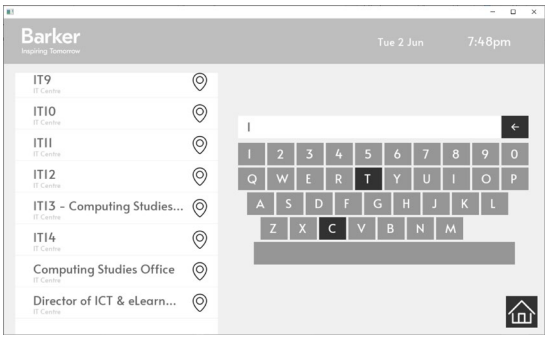
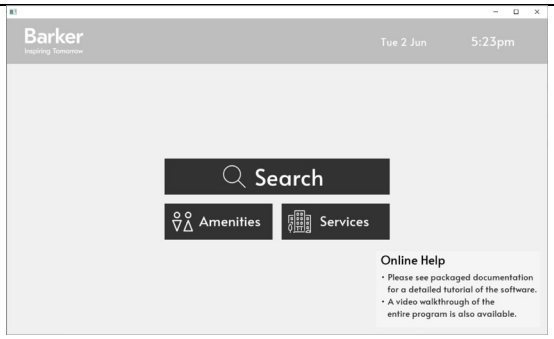
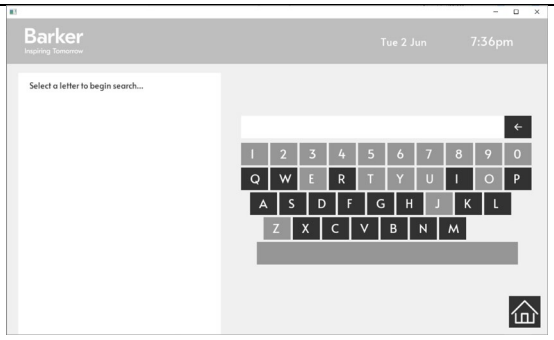
Program opened Amenities button pressed “Men’s” icon pressed	Map showing directions to closest Men’s toilets showing relevant info		Pass
Program opened Services button pressed Clinic icon is pressed	Map showing directions to the clinic showing relevant info		Pass
Program opened Amenities button pressed “Bubblers” icon pressed	Map showing directions to closest bubbler showing relevant info		Pass
Program opened Services button pressed “Printers” icon is pressed	Map showing directions to the closest printer showing relevant info		Pass

Keyboard

To summon the keyboard, the program was opened, and the “Search” button was pressed to show the menu as per the test above.

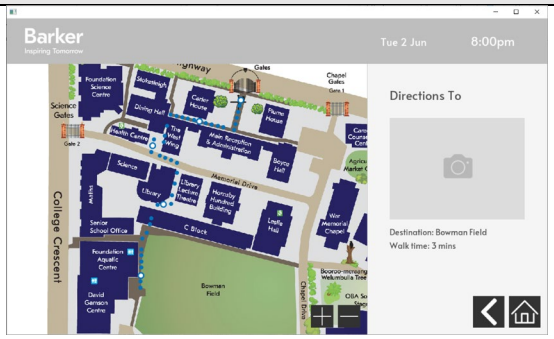
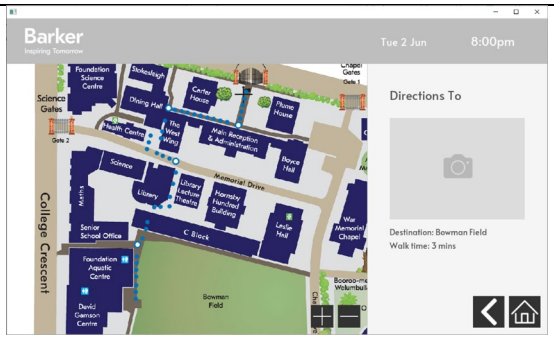
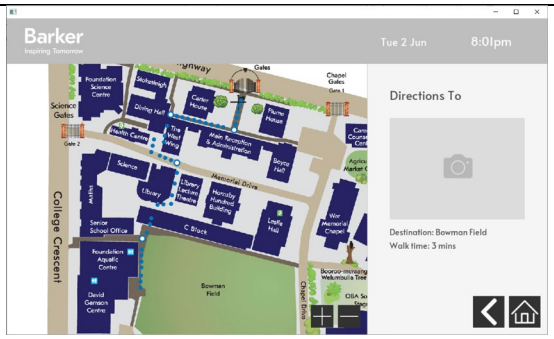
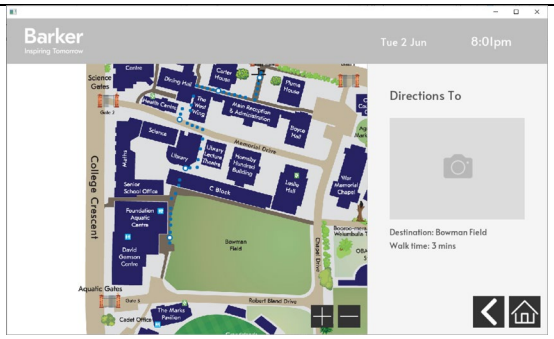
Actions performed	Expected output	Output	Result
Letters “K405” pressed	Results should display this room Entire search should appear in search bar		Pass
Letter “Z” pressed	Nothing should happen as this letter begins disabled Non-grey letters should reflect available letters		Pass
Letter “B” pressed	Relevant results should be displayed for “B” Letter should appear in search bar Non-grey letters should reflect available letters		Pass
Letter “B” pressed Letter “O” pressed	Relevant results should be displayed for “BO” Entire search should appear in search bar Non-grey letters should reflect available letters		Pass

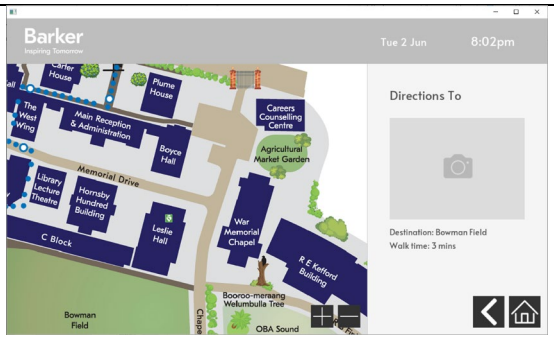
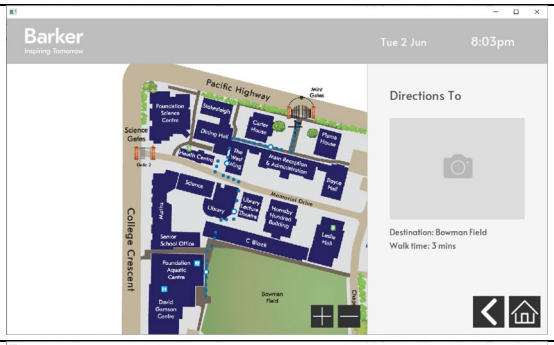
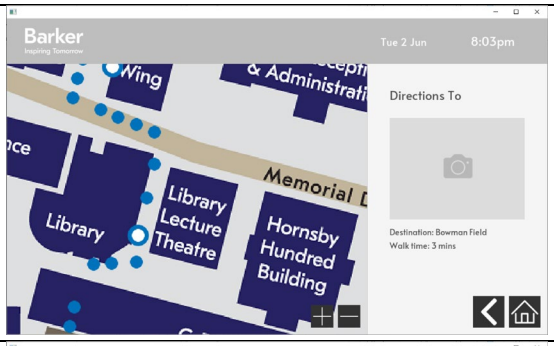
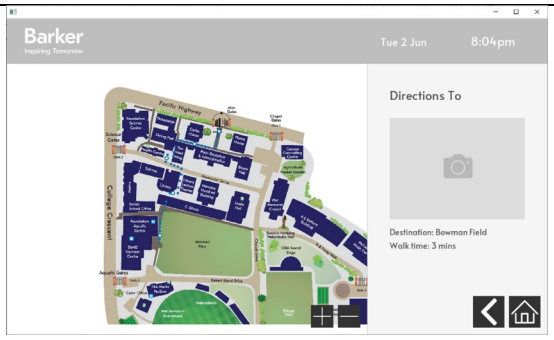
Letter “B” pressed Letter “O” pressed Letter “W” pressed	Relevant results should be displayed for “BOW” Entire search should appear in search bar Non-grey letters should reflect available letters		Pass
Letter “B” pressed Letter “O” pressed Letter “W” pressed Backspace key pressed	Relevant results should be displayed for search “AQ” Entire search should appear in search bar Non-grey letters should reflect available letters		Pass
Letter “B” pressed Letter “O” pressed Letter “W” pressed Bowman field selected	Directions to bowman field should be displayed correctly		Pass
Letter “A” pressed Letter “Q” pressed	Relevant results should be displayed for search “AQ” Entire search should appear in search bar Non-grey letter should reflect available letters		Pass
Letter “A” pressed Letter “Q” pressed Letter “U” pressed	Relevant results should be displayed for search “AQU” Entire search should appear in search bar Non-grey letter should reflect available letters		Pass

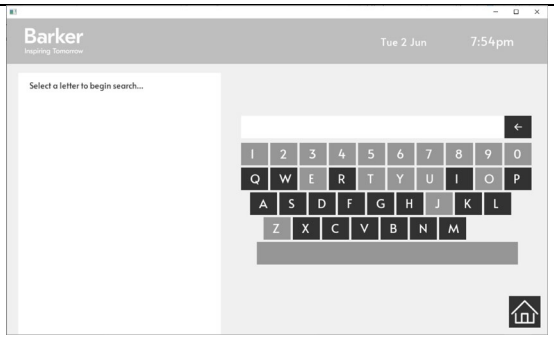
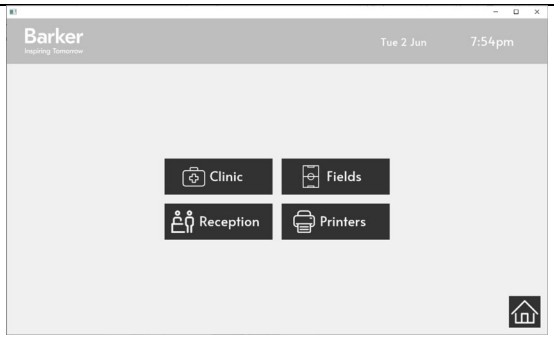
Letter “A” pressed Letter “Q” pressed Letter “U” pressed Backspace key pressed	Relevant results should be displayed for search “AQ” New search query should appear in search bar Non-grey letter should reflect available letters		Pass
Letter “I” pressed Results scrolled to bottom by dragging finger	Relevant results should be displayed for search “I” New search query should appear in search bar Results items should scroll smoothly with finger movement Non-grey letters should reflect available letters		Pass
Home button pressed	Menu should appear		Pass
Waited 3 minutes	Time in top right should change to correct time Non-grey letters should reflect available letters		Pass

Map

To test the map, the search menu was used to select the location, “Bowman field”. Different actions were then performed to assess the functioning of the map directions page.

Actions performed	Expected output	Output	Result
Zoom in button pressed	Map should resize by a small factor		Pass
Zoom in button pressed twice	Map should resize by a small factor twice		Pass
Zoom in button pressed thrice Zoom out button pressed twice	Map should zoom in then out according to the button presses		Pass
Map dragged up and to the right	Map should pan down and to the left		Pass

Zoom in button pressed 4 times Map dragged leftward	Map should zoom in 4 times and pan to the right		Pass
Map dragged to the right as much as possible	Map should pan to the left but should stop at the boundary		Pass
Zoom in button pressed repeatedly	Map should continue to zoom in until a certain boundary		Pass
Zoom out button pressed repeatedly	Map should continue to zoom out until a certain boundary		Pass
Home button pressed	Program should return to the main menu		Pass

Back button pressed from search screen	Program should return to the search page Non-grey letters should reflect available letters		Pass
* Back button pressed from Amenities menu	Program should return to the Amenities menu		Pass
* Back button pressed from Services menu	Program should return to the Services menu		Pass

* *Unlike the other tests, these were conducted by navigating through the different menus to the map screen as opposed to the standard test routine.*

These tests were conducted to assess the interconnections and functioning at the program level. The specific actions selected for testing have been carefully chosen in order to assess every different path in the code possible and ensure their correct operation.

For example, when 'Letter "I" pressed / Results scrolled to bottom by dragging finger', I am testing a variety of module inter-functioning, including the working of the keyboard in selecting a letter, the addition of a letter to the search bar, the correct appearance of the results as well as the code modules which allow for me to fluently scroll through them. Doing many more of such black box tests, in effect, will ensure that user input will produce consistent and correct on-screen outputs as expected.

Also, opening the program and testing each branch of menu options available when I test the Main menu is another way in which I can assess the functioning of my program. Doing so will ensure that new menu creation, the menu buttons, the back button, and home button works flawlessly, and thus, demonstrates the reliable and error-free nature of the software.

Module Testing

The planned test data and expected results are presented in the table below. The pass result is achieved if the actual results of the program using this planned test data matches the expected output. The Greenfoot debugging tools were used to execute each tested method using the input parameters and also to ascertain their final outputs.

shortestPath Method

Test data 1

Parameters	Data type	Value
start	int	52 (Mint Gates)
end	int	116 (Bowman Field)

Pre-set variables	Data type	Value
nodeInfo	ArrayList	342,179 439,198 411,351 464,370 369,367 349,407 … Array has been truncated due to its immense size. Please see 354 row sequential file “nodeinfo.txt” in the “roomData” folder which displays the entire contents of this array. … 1&100, 0&100,150&100, 150&100,294&100,195&100,4&100,3&100, 2&100,61&100, 195&100,2&100,5&100, 4&100,7&100,6&100,189&100, 5&100, … The “nodeconnections.txt” file has also been used to create this array with the corresponding nodes.

Expected output	
ArrayList: [116, 63, 182, 181, 67, 198, 196, 13, 332, 14, 15, 46, 22, 55, 27, 317, 50, 51, 52] This represents the shortest path found from the Mint Gates to Bowman Field.	
Actual output	**Result**
[As expected]	Pass

Test data 2

Parameters	Data type	Value
start	int	75 (counsellors office)
end	int	94 (English building entrance)

Pre-set variables	Data type	Value
nodeInfo	ArrayList	342,179 439,198 411,351 464,370 369,367 349,407 … Array has been truncated due to its immense size. Please see 354 row sequential file “nodeinfo.txt” in the “roomData” folder which displays the entire contents of this array. … 1&100, 0&100,150&100, 150&100,294&100,195&100,4&100,3&100, 2&100,61&100, 195&100,2&100,5&100, 4&100,7&100,6&100,189&100, 5&100, … The “nodeconnections.txt” file has also been used to create this array with the corresponding nodes.

Expected output	
ArrayList: [94, 255, 254, 253, 161, 243, 136, 349, 81, 135, 71, 72, 310, 329, 73, 131, 77, 137, 75] This represents the shortest path found from the counsellors office to the front of the English building.	
Actual output	**Result**
[As expected]	Pass

Test data 3

Parameters	Data type	Value
start	int	59 (Senior school courtyard)
end	int	107 (the Soundstage)

Pre-set variables	Data type	Value
nodeInfo	ArrayList	342,179 439,198 411,351 464,370 369,367 349,407 … Array has been truncated due to its immense size. Please see 354 row sequential file “nodeinfo.txt” in the “roomData” folder which displays the entire contents of this array. … 1&100, 0&100,150&100, 150&100,294&100,195&100,4&100,3&100, 2&100,61&100, 195&100,2&100,5&100, 4&100,7&100,6&100,189&100, 5&100, … The “nodeconnections.txt” file has also been used to create this array with the corresponding nodes.

Expected output	
ArrayList: [107, 277, 112, 80, 171, 66, 65, 62, 67, 181, 182, 68, 58, 59] This represents the shortest path found from the Senior school courtyard to the Soundstage.	
Actual output	**Result**
[As expected]	Pass

createResults Method

Test data 1

Parameters	Data type	Value
search	String	“BO”

Pre-set variables	Data type	Value
roomInfo	ArrayList	312,Stokesleigh,Stokesleigh 65,Bunker,C-Block 342,C1,C-Block 154,C3,C-Block 341,C4,C-Block 338,C5,C-Block 337,C6,C-Block … Array has been truncated due to its immense size. Please see 231 row sequential file “rooms.txt” in the “roomData” folder which displays the entire contents of this array.

Expected output	
Array name	**Value**
results	[Bunker, Barker Bunker (Girls' Fitness), Blended Learning Room, BCMA Theatre, Bowman Field, General Duties Office 2, Boyce Hall]
resultsEnd	[65, 65, 17, 253, 116, 66, 191]
resultsDept	[C-Block, C-Block, Library, Kefford Building, Bowman, Bowman, Boyce Hall]
availableLetters	[U, A, L, C, O]

Actual output	Result
[As expected]	Pass

Test data 2

Parameters	Data type	Value
search	String	“AQUATIC C”

Pre-set variables	Data type	Value
roomInfo	ArrayList	312,Stokesleigh,Stokesleigh 65,Bunker,C-Block 342,C1,C-Block 154,C3,C-Block 341,C4,C-Block 338,C5,C-Block 337,C6,C-Block … Array has been truncated due to its immense size. Please see 231 row sequential file “rooms.txt” in the “roomData” folder which displays the entire contents of this array.

Expected output	
Array name	**Value**
results	[Aquatic Centre, Gym]
resultsEnd	[118, 280]
resultsDept	[Aquatic Centre, Aquatic Centre]
availableLetters	[E]
Actual output	**Result**
[As expected]	Pass

Test data 3

Parameters	Data type	Value
search	String	"C"

Pre-set variables	Data type	Value
roomInfo	ArrayList	312,Stokesleigh,Stokesleigh 65,Bunker,C-Block 342,C1,C-Block 154,C3,C-Block 341,C4,C-Block 338,C5,C-Block 337,C6,C-Block … Array has been truncated due to its immense size. Please see 231 row sequential file "rooms.txt" in the "roomData" folder which displays the entire contents of this array.

Expected output	
Array name	**Value**
results	[Bunker, C1, C3, C4, C5, C6, C7, C10, C11, C12, C13, C14, Sports Office, ICT Manager's Office, Lecture Theatre, Barker Bunker (Girls' Fitness), History Office, Geography Office, Deans Office, Head of Middle School Office, HoBC Office, Chapel, K23, Cadets Office, Commerce Meeting Room, Christian Studies Office, Computing Studies Office, Careers Office, Counsellors Office, Council Room, Clinic, Common Room]
resultsEnd	[65, 342, 154, 341, 338, 337, 334, 335, 339, 340, 194, 328, 65, 209, 215, 65, 200, 200, 62, 213, 213, 309, 84, 278, 180, 274, 220, 75, 75, 303, 195, 218]
resultsDept	[C-Block, Chapel, Chapel, Marks Pavillion, Gamson Centre, Kefford Building, IT Centre, Careers & Councelling, Careers & Councelling, Carter House, Health Centre, Other]
availableLetters	[, 1, 3, 4, 5, 6, 7, H, A, O, L] (space character is a valid entry)
Actual output	**Result**
[As expected]	Pass

newPath Method

Test data 1

Pre-set variables	Data type	Value
start	int	52 (Mint Gates)
end	int	116 (Bowman Field)
path (test data shown above for shortestPath())	ArrayList	[116, 63, 182, 181, 67, 198, 196, 13, 332, 14, 15, 46, 22, 55, 27, 317, 50, 51, 52]
nodeInfo	ArrayList	342,179 439,198 411,351 464,370 369,367 349,407 … Array has been truncated due to its immense size. Please see 354 row sequential file “nodeinfo.txt” in the “roomData” folder which displays the entire contents of this array. … 1&100, 0&100,150&100, 150&100,294&100,195&100,4&100,3&100, 2&100,61&100, 195&100,2&100,5&100, 4&100,7&100,6&100,189&100, 5&100, … The “nodeconnections.txt” file has also been used to create this array with the corresponding nodes.

Expected output	
Output	**Value**
walkTime	3
end	116

Actual output	Result
[As expected]	Pass

Test data 2

Pre-set variables	Data type	Value
start	int	75 (counsellors office)
end	int	94 (English building entrance)
path (test data shown above for shortestPath())	ArrayList	[94, 255, 254, 253, 161, 243, 136, 349, 81, 135, 71, 72, 310, 329, 73, 131, 77, 137, 75]
nodeInfo	ArrayList	342,179 439,198 411,351 464,370 369,367 349,407 … Array has been truncated due to its immense size. Please see 354 row sequential file “nodeinfo.txt” in the “roomData” folder which displays the entire contents of this array. … 1&100, 0&100,150&100, 150&100,294&100,195&100,4&100,3&100, 2&100,61&100, 195&100,2&100,5&100, 4&100,7&100,6&100,189&100, 5&100, … The “nodeconnections.txt” file has also been used to create this array with the corresponding nodes.

Expected output	
Output	**Value**
walkTime	2
end	94
Actual output	**Result**
[As expected]	Pass

Test data 3

Pre-set variables	Data type	Value
start	int	59 (Senior school courtyard)
end	int	107 (the Soundstage)
path (test data shown above for shortestPath())	ArrayList	[107, 277, 112, 80, 171, 66, 65, 62, 67, 181, 182, 68, 58, 59]
nodeInfo	ArrayList	342,179 439,198 411,351 464,370 369,367 349,407 … Array has been truncated due to its immense size. Please see 354 row sequential file “nodeinfo.txt” in the “roomData” folder which displays the entire contents of this array. … 1&100, 0&100,150&100, 150&100,294&100,195&100,4&100,3&100, 2&100,61&100, 195&100,2&100,5&100, 4&100,7&100,6&100,189&100, 5&100, … The “nodeconnections.txt” file has also been used to create this array with the corresponding nodes.

Expected output	
Output	**Value**
walkTime	4
end	107
Actual output	**Result**
[As expected]	Pass

Button() constructor modules

To conduct these tests, the Greenfoot project environment was utilised to create a new Button class with the parameters as defined. This instantiates the class and the resulting object was displayed on the screen and screenshotted into the output section. The values of the resulting object was then viewed using the Greenfoot "Inspect" tool. These tests also verify the correct use of method overloading within this particular class.

Test data 1

Parameters	Data type	Value
backButton	int	2

Pre-set variables	Data type	Value
backButton	int	-1
text	String	null
letter	String	null
zoomIn	boolean	true
active	boolean	null
width	int	null

Expected output		
Output	**Data type**	**Value**
backButton	int	2
text	String	null
letter	String	null
zoomIn	boolean	true
active	boolean	null
width	int	null
Image	GreenfootImage	Dark grey rectangle with an arrow pointing arrow pointing to the left in the centre
Actual output		**Result**
[Values are as expected] Image: 		Pass

Test data 2

Parameters	Data type	Value
zoomIn	boolean	true

Pre-set variables	Data type	Value
backButton	int	-1
text	String	null
letter	String	null
zoomIn	boolean	true
active	boolean	null
width	int	null

Expected output		
Output	**Data type**	**Value**
backButton	int	-1
text	String	null
letter	String	null
zoomIn	boolean	true
active	boolean	null
width	int	null
Image	GreenfootImage	Dark grey rectangle with plus icon in centre
Actual output		**Result**
[Values are as expected] Image: 		Pass

Test data 3

Parameters	Data type	Value
letter	String	“P”
width	int	60
available	boolean	false

Pre-set variables	Data type	Value
backButton	int	-1
text	String	null
letter	String	null
zoomIn	boolean	true
active	boolean	null
width	int	null

Expected output		
Output	**Data type**	**Value**
backButton	int	-1
text	String	null
letter	String	“P”
zoomIn	boolean	true
active	boolean	null
width	int	null
Image	GreenfootImage	Small light grey rectangle with capital “P” in the centre
Actual output		**Result**
[Values are as expected] Image: 		Pass

Test data 4

Parameters	Data type	Value
width	int	240
height	int	80
text	String	“Amenities”
size	int	30

Pre-set variables	Data type	Value
backButton	int	-1
text	String	null
letter	String	null
zoomIn	boolean	true
active	boolean	null
width	int	null

Expected output		
Output	**Data type**	**Value**
backButton	int	-1
text	String	“Amenities”
letter	String	null
zoomIn	boolean	true
active	boolean	null
width	int	null
Image	GreenfootImage	Large dark grey rectangle with “Amenities” clearly displayed to the right of correct icon
Actual output		**Result**
[Values are as expected] Image: 		Pass

The testing of individual modules will ensure the accuracy of the solution. It helps validate the correctness of the final product for the client and simplifies the acceptance process for the client as solution is prepared to be integrated into the overarching system. Testing is one of the steps in quality assurance.

Product Evaluation

Project success

Overall, the project is highly successful in its purpose to provide a working navigation system for Barker College as evident in its high degree of conformance to the project success criteria outlined previously. Firstly, I have met all the requirements defined in the first stage of development criteria. It is clear that the designed product not only meets all these conditions but goes over and above the minimum needs of the client, like the inclusion of a very large searchable database (with almost 400 data points) to allow users to quickly and easily access a path to unfamiliar or unknown areas of the vast Barker College campus. Further, it is evident that my project is a working solution to the client's needs as it allows for easy navigation around the school with little lag and can be run on any computer able to install Java 8. The success is also demonstrated through the project's high conformity to the varied quality assurance practices outlined previously. This has been confirmed by user feedback during the testing phase of development. Also, there are no known bugs or glitches present in the software at this time due to thorough testing at the system, program and module levels, using a wide array of test data. The program has been tested on 64-bit processors and being allocated under 2Mb of ram with the use of a touchscreen and is clearly able to perform effectively, with low loading times under various test conditions. Hence, the program is designed to enable optimal user experience on any kiosk placed around the school. It is also intended to be a desirable and welcoming promotion for all current and future members of the Barker community. Thus, it fully conforms to the success criteria as set out prior to its development in Assessment Task 2.

Adherence to Criteria

As discussed, the high degree of success of College Connector program is evident in its high degree of conformance to the aforementioned criteria. The first criterion of creating an easy to use guidance system for Barker College is met and exceeded. This is evident in the completion of all requirements as set out in defining and understanding the problem phase of development. For example, the program includes an option to search for almost every room within the senior school, showing the walk time and exact path and direction on a map the desired destination.

I have also met the criterion of having a bug free program due to the lack of evidence of any errors or bugs during the compilation and running of the program. A strong indicator for this high success can be found above in the white box and black box testing performed. It can be appreciated that every test conducted – system, program and module tests – have consistently passed with a range of test data that covers a wide variety of possible input data. The high rate of passing indicates that the program is free from any major bugs which may hinder the success of the program.

In addition, the quality assurance practices outlined in the project design documentation have all been implemented into the operation of the solution. The integrity of the data within the system has been thoroughly vetted and user input has been restricted to an onscreen keyboard which does not allow for inappropriate input. Also, all functions which deal with user input have measures in place to deal with malformed input data. Further, the usability of my software solution was a central aspect to the design of the project, a vast array of user and client feedback being utilised to incrementally improve the GUI design to the most optimal layout and visual design as possible. This includes my comprehensive research into similar programs of this type and ensuring that the buttons and dialogues shown on my program are consistent with these. Furthermore, the accuracy of the solution has been carefully considered with a continual testing process in which I use computer generated test data to verify the correctness of the output of every major module as well as the program as a whole. This ensures that the program properly outputs the correct data to the user based on their specific inputs. Moreover, the strict use of comments and meaningful identifier names for future developers has meant that a high standard of maintainability is achieved in my solution. The modular design and top-down approach have allowed me to properly implement Object Oriented Programming which conforms to the necessary standards like encapsulation and appropriate abstraction. Thus, the implementation of these quality management practices have greatly contributed to the high degree of success of my project.

Finally, the efficient and fast execution of the program is another success criterion which has been met by conforming to good programming, management, and error detection and correction practices. This can be observed in the benchmarking tests conducted for the main modules which executes in under 1 second, allowing for a smooth and more streamlined user experience. User feedback has also indicated that the program runs at an adequate speed for effective and efficient navigations with no complaints.

Process Evaluation

The following practices were evaluated throughout the program in order to verify the application of good software development.

***Note**: A project logbook was kept throughout the duration of the project. Only relevant logbook entries have been included in this part of the task as required.*

Programming Practices

Throughout the development of my software solution, I have ensured adherence to the various programming practices, verified in the robustness and high quality of the code. All these practices can be observed in the Software Design and Development Stage 6 Syllabus as provided by NESA.

Firstly, this is evident in the **clear and uncluttered mainline** in each of the classes in the project. A top-down design was used to plan the workings of the program during the defining and understanding phase of development which has ensured the structure and purpose of each section of code is easy to discern. As discussed in my logbook entry (Decluttering of Code), structure charts served as the starting point to the development of algorithms in Java, the mainline being the act sequences of each class. This practice also helps the maintenance process as each class is broken up into main method calls which makes it easy to follow the logic of the entire program. This practice is also very useful in testing and error detection which is heavily simplified for the developer.

Decluttering of my Code	**April 10th, 2020**
Since the last entry, I have now completed the main functioning of the program with the map showing the shortest path to the destination. The act sequence for moving and manipulating the map is currently very cluttered as it has the functions to execute the different map functions. This made it hard to determine what part of the act cycle is creating bad data and thus leading to logic errors within the program. Thus, I have referenced the structure diagrams I have made in the previous assessment task and am in the process of abstracting the functions so that I can unclutter and clear the mainline of this main class and each of the other classes currently in the program. I hope to continue referencing the previous documentation to help create the program in a more ordered manner and to help fix the current code. In terms of progress, I am on schedule with the Gantt chart for the stages within implementation of the program. I hope to finish the first prototype of the project in a few weeks as development is slower due to the pressures of other external tasks.	

Another good programming practice used is the implementation of only **one logical task per subroutine** in my solution. This is evident in every method created serving a single role in the functioning of an algorithm, allowing for an efficient solution of high quality. I have undertaken a rigorous abstraction process to ensure that each method has a clear and specific purpose. The practice allows for a simplified error detection and correction process as code modules not functioning properly can easily be isolated and fixed accordingly. Stubs were also used if the error was not immediately discerned. As seen in my logbook entry later in the modification process, it was much easier and quicker to update and modify the code to fix bugs and create functionality by means of this practice.

Abstraction of Methods **April 18th, 2020**
The functioning of the map is now operational in terms of showing all the nodes possible and the best path. The shortestPath(int start, int end) subroutine now takes the beginning and finishing node and updates the map to display the path identified. The function spans many lines and is quite confusing currently as this function does many tasks in itself. So, I have decided to create different functions for each task, for example, the making of a thick line, the overlay of the paths onto the map and the calculation of the shortest path are now completely separate functions. The new shortestPath() method should only return an array of the nodes for the shortest path, a single logical task. This now needs to be completed on every other function to help with the testing and maintaining of my code. Hence, this process of abstracting each function into doing one logical task will improve the readability and testing of my code in the future. I am also hoping to redesign the user interface as it currently doesn't conform to other programs of that type.

As mentioned previously, the **use of stubs** allows for unwritten functions to output fixed data while the mainline logic is tested. They are small modules that can be used to replace parts of a program which have not yet been completed or may contain errors. Stubs may set values of variables which interact with the calling module or create entire arrays of test data. Stubs reduce the time for development by allowing for modules of code to be written and tested without needing to have all of its dependent methods fully functioning. Shown in my logbook entry during the coding process, stubs were thoroughly utilised to simplify the testing process and ensure that the mainline and different functions can properly interact with each other.

Using a Stub to Replace Function **April 2nd, 2020**

I have now implemented a way to add new nodes to the array of nodes and allow for them to be saved to the program. The file reading function is not fully completed as I am having issues with Greenfoot not recognising the text file in the folder. Hence, instead, I have made a temporary stub method which converts a comma separated string into the array of nodes to return as required to process the shortest path between them. This stub has allowed me to make tremendous progress with optimising the A* pathfinding algorithms and the associated display of its result. After doing this, I must find a way to show the shortest path so that these algorithms could be tested. I have created a temporary thickLine(int node1, int node2) stub which simply creates a 1px thick line from one node to another. This has allowed me to visually see the functioning of the A* algorithm without wasting time making the full method which I plan to finish at a later stage. This function serves as a placeholder for the subsequent function which should create a much thicker and hence more visible line between certain nodes and hopefully a precursor to a more aesthetic dotted path in the future.

All these good programming practices have assisted me in writing my **code for subsequent maintenance**. Adherence to all these practices have enabled me and future maintenance developers, to understand and update the source code as required to improve efficiency. In addition to the good structure of the code, thorough use of internal documentation like comments and intrinsic documentation like the use of meaningful identifier names successfully enriched the long-term maintainability of my code. Coding for subsequent maintenance will inherently improve the manganocene process by ensuring more readable and understandable code. As shown in my logbook entry, this has been achieved in part by referencing the documentation created in the defining and understanding and planning and developing stages of development such as the variable naming conventions outlined in the previous documentation.

Fixing Line Creation Error **April 30th, 2020**

From the previous logbook entry, I have aimed to fix the error which caused all the path lines created on the map to appear incorrectly as I was undertaking the process of abstraction. I realised that there is a high chance I have mistyped something when transferring different code blocks into their own functions. Luckily, all the codes created thus far have been properly commented with its function and every variable has been named meaningfully, allowing for me to easily find the core functions suspected of causing the issues. After many hours of testing the different functions associated with this error, I narrowed the issue down to the thickLine() function in which I had accidentally set both the x and y locations to the x value. This caused the lines to only generate in the line y = x, and thus creating the wrong line segment. When fixing this logic error, the many comments and meaningful identifiers I had used proved to be extremely valuable. I will continue to implement this coding practice as I continue to develop this system.

Error Detection and Correction

Throughout the development of College Connector, various methods of error detection and correction were utilised to mitigate bugs in the software. Errors may cause the program to crash (runtime errors), fail compilation (syntax errors) or result in unwanted outputs (logic errors) and correcting errors involves correcting code which does not function as intended.

Syntax error: An error resulting from an illegal statement in a programs code.
Logic error: An error in the logic of a program that causes incorrect actions to be taken.
Runtime error: An error that occurs while the object code is being executed.

Firstly, **flags** were used to indicate whether a particular function has been executed, a process has occurred, or to indicate the status of a condition during the running of the program. Flags are commonly boolean values that are initially false and are set to true if a particular segment of code has been processed. These can be used for correcting logic or runtime errors when the interpreter or compiler does not provide useful information. The specific location of a bug can be pinpointed with strategic use of flags. Clearly shown in my logbook entry, flags were used to check the flow of the program and narrow down where particular errors were present in the code.

Fixing Node File Reading Error **May 22nd, 2020**

Thus far, most of the program has been implemented and I have fixed the user interface according to feedback received from users about the previous one. Buttons are now larger and more identifiable as to their function. I had also run into a logic error when reading from the nodes file after making major changes to the way nodes are stored. To fix this issue, I created a new boolean variable set to false and made a process which was set it to true if more than 4 node coordinates have been processed by the function. After viewing the status of this variable using the Greenfoot debugging tool, I was able to figure out the root of this issue and promptly fix it. This flag helped me to easily identify the source of the error and correct it. After fixing the error, I tested the function with many different sets of data to ensure that no other bugs were present in this module of code. In addition to this, when testing empty files, a runtime error would cause me program to halt execution. A flag was used to isolate where this error was taking place and I determined that this error was due to my 'for loop' dividing by zero when calculating how long the subsequent array should be. This was fixed by checking first if the variable is equal to 0 and acting accordingly.

Also, I made good use of **debugging output statements** to indicate the order of execution of different functions and the value of variables at particular points in the program's execution. These additional statements inform the developer of the status of particular variables to mitigate any logic error or runtime errors from occurring. It has helped me to determine the nature of variables within subprograms and whether they are being set in the correct order and output the correct values, thus giving greater insight into the flow of execution and the source of errors. As seen in my logbook entry, debugging output statements were used to display the contents of specific variables at crucial stages of execution to narrow down the location and nature of program errors.

Issues with Centring the Map **May 29th, 2020**

Today I have finally fixed the error causing the map to incorrectly centre on the left side of the screen and scale to the window size. I identified the cause of this issue was in the way the new location and size of the map was calculated and set. By checking the values set for zoom, centreX and centreY, I realised that the coordinates were swapped and this appeared to have fixed the issue. But after painstaking testing, I realised that this was not the issue, but discovered a more deep-rooted logic error – my methodology in finding the centre was wrong. The way I found the centre was by taking the average position of all the points in the shortest path. I printed all the values inputted to the console and realised that when there was a cluster of nodes in one side, the map would tend to centre closer to those, meaning the calculations were not what I had intended. To fix this, I wrote code to find only the outermost nodes and take their average instead. This would mean the map focuses on the shortest path as a whole instead of favouring clusters of nodes. The debugging output statements used proved instrumental in detecting and fixing the issue with this part of the code. I have also fixed the issue with the search box and results displaying items which overflow by truncating the display of strings which are too long for their bounding box. In addition, a runtime error involving the calling of an undefined element of the connections array was also fixed by simply adding 1 to the for-loop end parameter as this was not correctly set.

Another method of error detection utilised in the implementation of my project was **peer checking** to gain a new perspective on the program's execution. Peer checking is where team members or colleagues of the developers assess and analyse each other's work to ensure its correct and flawless operation. The original developer may not detect different errors due to their inherent bias due to their familiarity with the system, hence allowing feedback from a new point of view may elucidate further problematic areas. In my project, informal peer checking was mostly used, as shown in my logbook entry. These peers commented on my code and shared their thoughts and ideas which I incorporated.

Peers Checking my Project **May 11th, 2020**

Since the last logbook entry where I had completed the third prototype of my user interface design for the program, I have sought feedback from my peers about this GUI and the functioning of my solution as a whole. I gave the program to my father to "find how to get to Bowman field" without further instructions on how to use it. He was able to navigate throughout the program with ease and ended up on the directions screen to Bowman Field but found that the map did not centre properly and the text on the search screen and the results items were overflowing out of their boxes. Both these errors were previously overlooked due to my familiarity with the program and were quickly recorded and plans were made to amend them. After this, I consulted with one of my classmates as to the functioning of the buttons within the program. After reviewing my code, he commented that the zoom button class should be implemented within the button class as it was of the same thing. He also commented on the way in which I was switching between the different functions to execute when clicking a button as I am using if/else-if statements to pick the correct option. I was informed that Java's "switch" statement would probably be a better choice in terms of processing times but also in the readability of that particular class, as there is a lot going on. I have taken this advice into account and plan to fix and implement such things in the near future.

Project Management Practices

A range of good project management practices were used to ensure that each stage of development of my project were highly efficient and effective and could be achieved in the timeframe required by the client.

Firstly, to develop my project, I strictly adhered to the **software development cycle** to ensure the successful management of each stage of development. Defining and Understanding, and Planning and Evaluating the software solution have enabled me to fully comprehend and analyse the needs of the client and plan out how these needs could be met to a high degree of quality. After this, implementing the program was much more straightforward, the algorithms and tools established earlier enabling an easier process as seen in the logbook entry below. Then the testing and evaluation of the project has helped me to identify and isolate bugs that were present in order to correct the operation of the software, also seen in the previous logbook entries.

Finalising Implementation **May 8th, 2020**

Since writing the last logbook entry, I have now fully completed a working prototype of my project. It includes functioning of the user interface as well as the map seeming to work properly. On the Gantt chart I've been trying to follow, this now marks the point where I should begin the testing and evaluation process and then the maintenance of the product. As I have been adhering to the software development cycle, I have been able to use the understanding and planning completed in the previous stages to build my solution effectively and efficiently. All the previous documentation has been extremely useful in the implementation of the project and now that I am creating test plans for the modules, I can quickly reference my structure charts and past IPO diagrams to easily create the test data. This is because the inputs and outputs of each function have been clearly defined and documented correctly. Adherence to the software development cycle has really helped in the implementation of the code and is continuing to be of use in the final stages of the cycle.

Also, the thorough use of **developer documentation** proved to be a tremendous device in managing the project well. At the beginning of the project, time was invested in the documentation of the overall software solution in the form of algorithms, IPO diagrams, data dictionaries and various **system modelling tools**. This enabled the management of the entire project by indicating the parts of the project that needed completion first, if a part was dependent on previous modules, and allowed for a better estimation of time required for such tasks to be completed to a high degree of quality. This use of time management was quantified, aided by use of a **Gantt chart** to set deadlines for the completion of different parts of the software solution and help to better manage the project development. This chart has helped me to keep on track with the project and complete it before the deadline and to a high degree of quality. The creation of these tools is evident in my logbook entries before the implementation of my code.

Starting Implementation **March 23rd, 2020**

Now that the previous Assessment Task has been completed, I need to start coding the product, but I wasn't sure where to begin. To overcome this, I started with the previous completed documentation to serve as a great starting point in the real implementation of my code. I have now created all the classes in accordance with the class diagrams which were made in the previous task. I decided to start the code for the A* algorithm first because it is the hardest algorithm to be completed in the program and would serve as a marker of the time required to complete subsequent functions. I am coding as closely to the pseudocode algorithms created previously as they serve as a great guide as to how everything fits together. The data dictionary is also helping me to properly set the types and names of variables so that it closely resembles this Map class I am coding currently. I hope to stay on track with the Gantt chart I have made throughout the project to keep me on task and at an efficient pace so that I can meet the deadline for the project. The developer documentation has really helped me to begin creating the bare bones of the task and the thorough consideration put into these developer tools is proving to be very useful.

In addition, the variety of **continuous quality assurance practices** as set out in Assessment Task 2 were implemented throughout the development process to ensure a quality solution would be achieved. Data integrity is closely upheld in my solution as user input has been restricted to only touchscreen presses which allows for more flexibility in the validation of input. This practice has evidently been implemented in the keyboard menu where invalid letters are greyed out and will not execute a search through the stored entries. No other input is considered and thus cannot cause issues with data integrity. I have also spent much time ensuring that all the room names and departments are valid and thus ensuring data integrity. I have also utilised these quality management practices in increasing the usability of the software throughout its design which is also evident in the plentiful inclusion of online help. Through managing the accuracy of my software, I have been able to complete many successful testing processes allowing for a more streamlined development process with minimal delays to expected. These practices have thus proven instrumental in the maintainability of the code as all data, whether provided internally or externally, the GUI and all methods as well as the thorough documentation exhibit a high level of validity and accuracy. Thus, these quality management practices have enabled the successful management of the project as a whole.

Data validation **April 21st, 2020**

Now that the first prototype of my program is up and running, I have the issue of inputting correct data into the list of rooms available to be searched. After spending hours creating a comprehensive list of classes and destinations able to be searched, I have found a list of classrooms in the school on the portal and have attempted to merge these with my current list. I am now in the process of making a python script which can firstly reformat these into a code friendly form (I am considering using comma separated values for each field) and secondly, verify and confirm the accuracy of these external entries. There are two fields that must be checked, the room name and its associated department. For the field name, my python code checks whether it (or permutation of it e.g. K404, K405…) already exists and whether it is capitalised correctly. It then checks whether its associated department name conforms to the previously defined ones and notifies me as to any errors and inconsistencies found. These errors were then manually corrected, and these tests were run until no further errors were detected. This demonstrates the high degree of the integrity of data within the program and has allowed for me to assure the accuracy of my solution. I overcame a great challenge in the implementation process as I attempt to adhere to the quality assurances I had set for this project.

Finally, another project management practice used was the adherence to a **communication plan** between the developer, the clients and intended users of the program. A thorough communication mix was outlined before the software was developed to detail the combination of formal and informal communication skills to be used to accomplish specific tasks (e.g. progress report, algorithms, funding requests) or elicit specific feedback about the proposed system. The communication plan established the frequency and method used by the developer to exchange vital information between the client and the users. This plan helps the developer better interact with their client to understand exactly what their requirements are and how they are to be achieved. Development plans and acceptance testing is to be made in close communication with the client so as to reduce confusion and help fully meet the needs of the client. The development and use of this are evident in the logbook entry below.

Communicating with the Client **April 24th, 2020**

In close accordance with the client, I have now decided to name my project ‘College Connector’ as it bears resemblance to the Barker slogan in 2020 of ‘Connect the College’ and describes the functioning of the program well. The ongoing communication with the client has been modified in recent times due to the COVID-19 crisis and now takes the form of mostly online communication. To ensure the correct completion of this project I have been in close contact with my client, Mr Grant as to the exact specifications he requires of the project. Due to the health risks associated with the pandemic, one on one interviews as outlined in the communication plan could not be formally conducted. Instead, I made use of the Teams program to keep in close contact with Mr Grant and have ensured regular messaging with him about aspects of the project that I was unsure about. He assisted me by explaining the way in which he wanted the structure charts to be done to maximise satisfaction. So, this new means of communication has helped me complete the tasks despite the lack of face to face contact with the client.

References

Anon, 2012. Software and Course Specifications. 4th ed. NSW: Board of Studies. [Accessed 1 Mar. 2020].

Anon, 2019. School Map May19. [ebook] Barker College. Available at: <https://www.barker.college/media/2433/school-map-may19.pdf> [Accessed 13 Mar. 2020].

Arora, S., 2018. Learn Java: Tutorials for Beginners, Intermediate, and Advanced Programmers. [online] Stackify. Available at: <https://stackify.com/java-tutorials/> [Accessed 10 Mar. 2020].

Blanes, A., Blanes, P. and Cuenca, J., 2010. Freepik | Graphic Resources for everyone. [online] Freepik. Available at: <https://www.freepik.com/> [Accessed 3 Mar. 2020].

Davis, S. (2011). Software Design and Development - The Preliminary Course. 2nd ed. Parramatta Education Centre [Accessed 17 Feb. 2020].

Imms, D., (2012). A* pathfinding algorithm. [online] Growing with the Web. Available at: <https://www.growingwiththeweb.com/2012/06/a-pathfinding-algorithm.html> [Accessed 29 Dec. 2019].

Kölling, M. (2016). Introduction to programming with Greenfoot Object-Oriented programming in Java with Games and Simulations. 2nd ed. Pearson Education [Accessed 20 Feb. 2020].

Patel, A., (2016). Introduction to A*. [online] Stanford Theory. Available at: <http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html> [Accessed 13 Jan. 2020].

Gosling, J., (2020). Windows System Requirements for JDK and JRE. [online] Oracle Java Documentation. Available at: <https://docs.oracle.com/en/> [Accessed 5 Mar. 2020].

Sun, K., (2008). [online] Lucidchart.com. Available at: <https://www.lucidchart.com/> [Accessed 26 Jan. 2020].

Swift, N., (2017). Easy A* (star) Pathfinding Python. [online] Medium. Available at: <https://medium.com/@nicholas.w.swift/easy-a-star-pathfinding-7e6689c7f7b2> [Accessed 16 Feb. 2020].

***Note**: All images in the program and on this document are being used under the education licence as allowed by the Australian Copyright Act provisions for educational institutions.*