Abstract

This project aims to examine if particular lines of programming code can help readers easily identify its functionality - referred to as “beacons”. In a program containing a sort function, for example, advanced programmers might observe the swap code inside a loop and comprehend that it is a sorting algorithm without much further examination. This project focuses on collecting data from non-novice programmers to determine whether or not these beacons can be detected. This data was collected by way of an eye-tracking problem solving experiment. Participants were presented with basic Java programs and were asked to determine, from a list of possible options, what output would be produced. Eye-tracking allows for the collection of fine grained movement details from our eyes. This data demonstrates where the focus of participants lies while they are solving the problem in real time. After the completion of the experimental phase, all eye-tracking data was analysed and associated heat maps and scan paths were produced. The results show promise that there is some evidence that supports the theory that beacons exist within programming comprehension. From the analysis of participant efficacy data, other interesting information was also discovered. For example, it was found that programming comprehension and maturity may be directly related as older participants performed better. A follow-on experiment with novice programmers will soon be undertaken. In this experiment, the same test cases from this thesis will be reused, but the discovered beacons will be highlighted to some participants to see if they perform better.

Contents

Declaration 1

Acknowledgements 1

Abstract 2

Contents 3

Chapter 1: Introduction 4

Section 1.1: Topic 4
Section 1.2: Motivation 4
Section 1.3: Problem Statement 4
Section 1.4: Approach 5
Section 1.5: Metrics 5
Section 1.6: Achievements 5

Chapter 2: Technical Background / Related Work 6

Section 2.1: Topic Material 6
Section 2.2: Technical Material 7

Chapter 3: The Problem 8

Section 3.1: The Tobii and Clearview 8
Section 3.2: Psychopy 8
Section 3.3: OpenSesame 9

Chapter 4: Solution (Design and Implementation) 10

Section 4.1: Test Cases 10
Section 4.2: Eye-tracking Equipment 11
Section 4.3: Experiment Builder 12
Section 4.4: Experiment Implementation 15

Chapter 5: Evaluation 17

Section 5.1: Experiment Output 17
Section 5.2: Eye-tracking Analysis 17
Section 5.3: Efficacy Analysis 19

Chapter 6: Conclusions 20

Bibliography 21

Chapter 1: Introduction

Section 1.1: Topic

This project is focused on determining if beacons can be detected as subjects attempt to comprehend Java code. According to Susan Wiedenbeck: “beacons are lines of code which serve as typical indicators of a particular structure or operation” [1]. In other words, beacons are particular lines of code that can quickly help us identify the functionality of the code. For example, in a sort program, advanced programmers might notice the swap code inside a loop and realise quickly that it is a sorting algorithm. Wiedenbeck showed this in her aforementioned paper.

The primary goal of this study is to expand upon the work of Wiedenbeck (and other similar studies, discussed in Chapter 2) to determine if beacons exist in programming languages and code blocks, and, if so, to demonstrate that they are an important factor in understanding that programs function. The aim of this project is to prove the hypothesis for much smaller blocks of code than originally used, such as a simple loop or an array manipulation.

This project will be undertaken with the aid of eye-tracking equipment. It has been shown that studying a person’s gaze can record fine grain information in space and time allowing for a deeper understanding of comprehension levels. [2] With the examination and analysis of the collected eye-tracking data, we will attempt to determine if some areas of code are indeed beacons.

Section 1.2: Motivation

The primary motivation of this project relates to Education and Programming Comprehension. According to a recent study by the Irish Times, "about one-third of Computer Science students across all institutes of technology are dropping out after first year in college” [3]. Similarly the report discusses high dropout rates among students progressing from first to second year in Universities. This is a serious issue that needs to be addressed. This project will attempt to examine the process of program comprehension and attempt to determine if beacons exist in small codes blocks. If their presence is verified these beacons could be used to help focus learners attention on particular code blocks if the learner is struggling.

From the results of this body of work, we will begin to understand how the experienced programmer breaks down and comprehends problems. Any beacons that are discovered could, in the future, be harnessed and incorporated into a learning system. Then we might be able to more efficiently teach programming to learners in Computer Science. With this project, the aim is to discover and verify the existence of beacons for use in future research and future implementation within learning systems.

Section 1.3: Problem Statement

In order to fulfil the motivation, an experiment will need to be constructed that can test the programmers programming comprehension on simple tasks. Initial test cases were developed which were simple enough that novice programmers could understand the code (for future experimentation / verification) but difficult enough that experienced programmers would take some time to solve them in order to collect good data. This was the first technical challenge of this project.

Secondly, these test cases needed to be incorporated into an experimental design using a Psychological experiment builder (for example, Psychopy or OpenSesame, detailed in Chapter 3 & 4) and integrated with an eye-tracking device (Tobii or EyeTribe, detailed in Chapter 3 & 4).

Section 1.4: Approach

Once the choices were made about which software and hardware to be used for the experiment (and research was completed), the development phase of the project began. This involved the creation of the experiment presented in Section 1.3. The experiment needed to encompass the test-cases, a short behavioural survey and integration with the chosen eye-tracker to aid analysis.

In order to ensure the validity of the experiment, two eye-trackers and two experiment builders were tested before making the decision of which technologies to use. In the case of each experiment builder, a Stroop test was first built in order to get familiar with the software. The Stroop test is a simple experiment that involves saying the colour of a word on the screen rather than the word. Doing this helped verify the software and aided in the final decision of what to use. This is detailed in Chapter 3 and 4.

This experiment would initially be performed on a group of 24 "experienced" programmers (4th years / PhD students). By experienced we mean that these participants would all have at least two years' experience using Java and were taught their introduction to Computer Science modules using this language. A mixture of subjects with differences in gender, age, glasses wearers were chosen to participate to ensure there would be as little bias in the dataset as possible.

Section 1.5: Metrics

Once the experimentation phase was complete, the analysis phase would commence. For this phase of the project, the collected data from the 24 participants (both eye-tracking data and efficacy data) would be analysed. This analysis would provide answers to the question: "What, if any, beacons exist in the given test cases" as well as other questions based on the survey provided in the experiment.

The output from this analysis will include heat maps and scan path graphs for all eye-tracking data and mean / p-test analysis of all efficacy data. The particular tools used and outputs obtained are detailed in Chapter 5.

Section 1.6: Achievements

Through the implementation of this project:

- The groundwork has been laid for future research in the area through the creation of a functional experiment as well as the creation of a dataset that can be further analysed and built upon in the future.
- The state-of-the-art research in the area of programming comprehension has been advanced upon. Some researchers have already discovered the existence of beacons. This work has been both verified and advanced on, as is detailed in Chapter 5**.**
- A lot has been learned about eye-tracking experimentation and analysis. This will be very advantageous as an eye-tracking based PhD will be embarked upon next year.

Chapter 2: Technical Background / Related Work

Section 2.1: Topic Material

In addition to Wiedenbeck's [1] work in Programming Comprehension, the early theory for beacons was formulated based on an experiment called "Duncker's radiation problem" [4]. The general concept of this experiment was as follows:

A diagram, Figure 2.1, is shown to participants (medical students) which shows an inoperable tumour surrounded by healthy tissue. The participants are asked how they could cure this ailment.

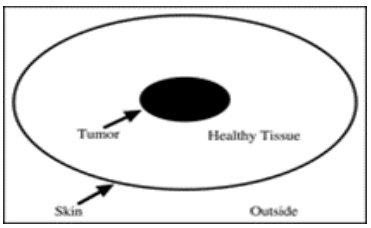

Fig 2.1: Diagram of a tumour underneath healthy skin

If a laser was directly fired at the tumour, healthy tissue would be damaged. According to the paper; "The solution requires firing multiple low-intensity lasers from several angles outside the healthy tissue so that they converge at the tumour, with combined intensity sufficient to destroy it."

The first time the experiment was ran, it was a base test where 14 medical students with no knowledge of the problem were asked to cure the ailment from the diagram with no assistance. While only 34% of participants succeeded in their task, by analysing the successful candidate's eye-tracking data, the experimenter learned that part of the skin was the "beacon" in the diagram.

In the second experiment, three different groups were tested. Group 1 repeated the experiment in the same way as phase 1. Group 2 were shown the diagram with the beacon animated (only animated slightly to be noticed at a cognitive level, imagine something pulsating very slowly with a small pulse radius). Group 3 were shown the diagram with the wrong area of the diagram animated.

The results of experiment 2 were as follows:

- Group 1 and 3 had approximately the same results as in the control group (experiment 1). That is, without animation of the beacon, and interestingly, when the wrong area of the diagram was highlighted, participants were only ~33% likely to give the correct answer.
- Group 2 however were able to identify the solution statistically significantly more often. That is, when the beacon identified from the control experiment was highlighted, 67% of people successfully found the solution.

Based on the findings of this experiment it was hypothesised that beacons may exist when people comprehend code and that perhaps people who successfully comprehend code could aid those that do not comprehend it as readily. This is what Wiedenbeck's [1] work related to and it is the aim of this project to verify and extend upon her work.

Another key researcher in the area is Brooks [5] whose top-down theories are hypothesis-driven. The programmer begins by making general hypotheses about the program's function and then looks for concrete evidence by scanning for beacons. If found they may conclude that the hypothesis is correct, if not found, they must revise/reject the hypothesis. Soloway *et al.* [6, 7] built on this, providing a plan-based theory. Here, understanding begins with the programmer hypothesising a high-level program goal, then decomposing it into sub-goals. The programmer utilises previously acquired schematic knowledge (plans) to satisfy the sub goals and ultimately the top-level goal.

Aschwanden *et al.* [8] and Crosby *et al.* [9] also expanded on the work of Wiedenbeck. They evaluated the use of beacons in programming comprehension. In summary, experienced programmers appear to use beacons, novices do not or if they do they are different to those of experience programmers. Poor naming conventions render novice programmers unable to comprehend code that they could comprehend when informative names are used.

It is clear that by analysing gaze during code reading processes, we will be able to learn a lot about how the experiment participant thinks. A person's gaze gives a record of their visual behaviour on a super fine-grained scale, in both space and time. The observation of eye movements adds an objective source of information about programmer behaviour to the collection of research methods in computing education which can inform the teaching and learning of programming.

Section 2.2: Technical Material

Before undertaking this project, Java programming was my main language of choice. However, in order to successfully complete this project, I had to learn Python. This is mainly because most vision based systems are Python based. To do this, I spent a short amount of time on Codecademy and other similar sites in order to learn the language. This was done in the early stages of the project.

Other technical material viewed / used included:

- "Getting started" information for the Psychopy experiment builder[1]. This was the starting point in all experimental research. An online video tutorial was also followed[2].
- A similar tutorial on the OpenSesame website to build a basic experiment in OpenSesame[3] and information relating to integrating the EyeTribe with the PyGaze toolkit[4].
- Information regarding the eye-tracking toolkit on the Tobii eye-tracker, ClearView[5].
- Information relating to an analysis toolkit "PyGazeAnalyser"[6] and its source code[7].

All of this material was very useful for different reasons. As will be seen in Chapter 3, Psychopy and the Tobii will be set aside in favour of OpenSesame and the EyeTribe. All research done on the other tools was vital however and greatly expanded the knowledge gained in the area.

[1] http://www.psychopy.org/gettingStarted.html
[2] https://www.youtube.com/watch?v=VV6qhuQgsiI
[3] http://osdoc.cogsci.nl/tutorials/step-by-step-tutorial/
[4] http://osdoc.cogsci.nl/devices/pygaze/
[5] www.hum.uu.nl/uilots/lab/resources/User_manual_ClearView_2.7.pdf
[6] http://www.pygaze.org/2015/06/pygaze-analyser/
[7] https://github.com/esdalmaijer/PyGazeAnalyser/

Chapter 3: The Problem

This project is focused on determining if beacons can be detected in Java code snippets. In order to complete this a number of avenues of investigation needed to be assessed. The first issue concerned which eye-tracking hardware to use. The second issue related to the psychology experiment builder that would be most appropriate. This would need to be highly modular and provide easy access to eye-tracking data collection. This chapter will look at these issues in detail.

Section 3.1: The Tobii and Clearview

The Tobii X60 eye-tracker [10] was the first piece of eye-tracking equipment that was examined. This piece of hardware is essentially a large PC monitor with an integrated camera that fires at multiple angles to build up a live image of the user's retina. A piece of software named Clearview is included with the Tobii. This is an all in one solution for eye-tracking. You run a server, open up Clearview and set it to collect your screen gaze data. On the face of it, this seemed like a perfectly simple solution. Unfortunately, multiple issues arose.

Two weeks were spent testing this piece of hardware and trying to get it operational. Primarily, the issued related to software licensing being outdated and hardware limitations of the machine that currently holds the Tobii software. The hardware issue related to the machine with the installed software only having 512MB of RAM. It was unable to run two pieces of software at once. Due to the licensing issues, the software could not be moved to a new lab machine.

The only solution that may have worked for collecting data would have been running any experiment on one machine and running a team viewer software on the Tobii PC to display it. Even with this setup, there may have been a lot of latency between all the running elements on both PC's and the data would not have been easy to collect nor verifiable. Given these challenges, a new solution had to be designed from scratch. The EyeTribe eye-tracker [11] was another piece of eye-tracking equipment available for this project. After examining and verifying the EyeTribe, it was decided to use it as the eye-tracker of choice. The EyeTribe is discussed in more detail in Section 4.2.

Section 3.2: Psychopy

According to their website; "PsychoPy is an open-source application to allow the presentation of stimuli and collection of data for a wide range of neuroscience, psychology and psychophysics experiments." [12]. It is based on the drag and drop build approach, but it is also developed using Python which allows for a lot of fine-tuned changes. In the original specifications for this project, Psychopy was the target toolkit to use due to its previously successful use within the department.

The first step to verify the software was to follow a video tutorial provided by the developers. This focused on building a Stroop test as presented in Chapter 2. This was a simple to build experiment that included the creation of a "conditions" file, two information screens and a main trial sequence that repeated a set number of times. The build took less than half an hour and it was very straight forward. It was also easily translatable into an early copy of the main experiment. This is included with the supporting material of this report.

One of the major issues that led to Psychopy not being chosen as the final experiment system was the fact that it does not link easily with any eye-tracker – Its primary function is not this. An external piece of software would have needed to be run that would collect any eye-tracking data. Since this could not be integrated with the experiment itself, there would have been no way to log the start and end points of each test case. This problem was simply too large of a hurdle to navigate and as such, development with Psychopy stopped in November. The focus then moved to using the OpenSesame application.

Section 3.3: OpenSesame

The original project specifications had to be ruled out at this point. OpenSesame integrated with the EyeTribe was suggested as a new solution. Early research into these products showed that they would integrate seamlessly together. The EyeTribe will be discussed in more detail in Chapter 4.

To gain familiarity with OpenSesame, a tutorial was followed which was similar to the Psychopy tutorial and introduced all of the major developmental elements of the builder and made completion of the target experiment very simple [13]. More research was required to be completed outside this sample experiment in terms of logging any eye-tracker results but this material was readily available on the development website and did not take much extra time [14].

OpenSesame uses “blocks” in a drag-and-drop style format to build up experiments without needing to interact with the underlying Python code regularly. These blocks can be seen within the red box in Fig 3.1. All elements are connected into the task list (highlighted within the green box in Fig 3.1) and then modified in the main section of the screen. These events run linearly. An example of block usage, in the case of a basic experiment a “loop” block could be used where the parameters are set and then a “sequence” of events that need to happen for each trial are defined.

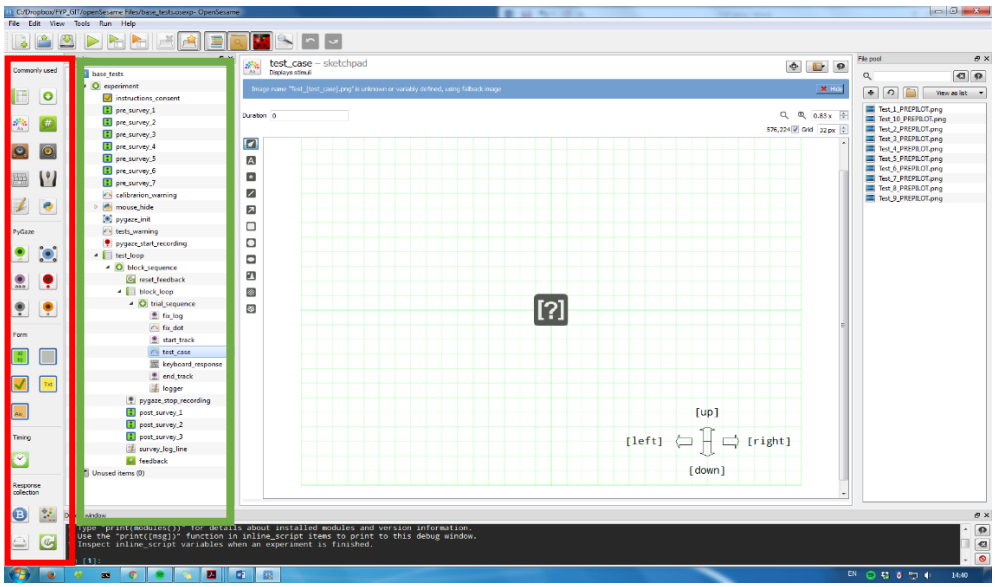

Fig 3.1: General Interface of the OpenSesame platform (showing the final experiment)

Once familiarisation with OpenSesame and the EyeTribe was complete, it was decided that these would be the tools of choice. The development phase of the project could now commence.

Chapter 4: Solution (Design and Implementation)

After completion of the evaluation of both experiment builders and both eye-tracking devices, attention turned to designing and building the final setup of the experiment. This involved three main areas:

1. What would be the best test cases to use?
2. What experiment builder should be used?
3. What eye-tracking equipment / analysis toolkit would be the best to use?

Section 4.1: Test Cases

Since all Computer Science students in Maynooth University are taught Java as their introductory language, Java was chosen as the language to construct the test cases in. In addition, a good balance between question difficulty, estimated time taken to solve and the amount of text that would be displayed on the screen was decided upon. Research of Wiedenbeck's paper [1] and the Eye Movements in Programming (EMIP) Work Shop data sets [15] provided a good starting basis for the creation of two test cases. The remaining test cases were developed entirely by the author.

Fig 4.1 shows a code example from the EMIP Workshop that has been used in previous experiments by members of the Computer Science Education community. This seemed to be a very deceptive code segment as it prints "Hello Sund" when most people might expect it to print "Hello Sun". The colour blocks represent "areas of interest" (AOI's) which are used as an analysis aid to track which block of code the fixations of participants fall in. AOI's were not incorporated into the current work, due to restrictions, but they are something that would be considered useful in future work.

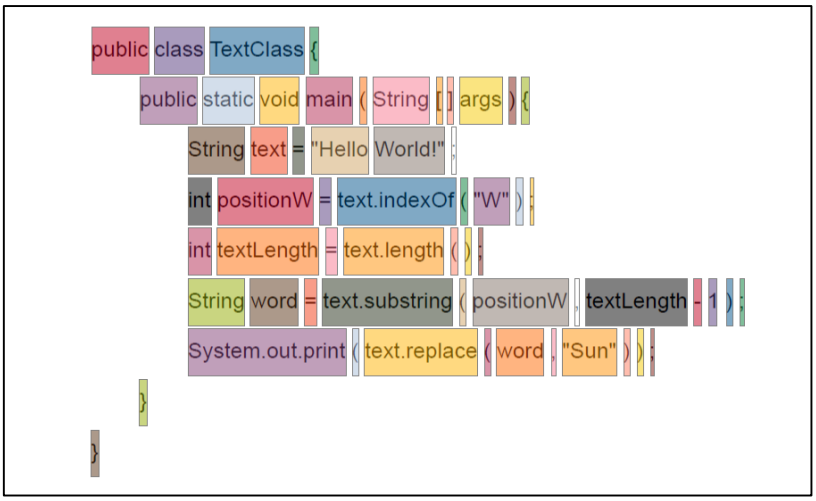

Fig 4.1: Sample AOI's highlighted for a code segment

Fig 4.2 shows the same test case, displayed in an Eclipse IDE format. These images are how the test cases were presented to the experiment subjects within the experiment environment. It was decided to go with this view as it would minimise the amount of clutter (only so much screen space could be used) and it would be more familiar to most students. The balance between these two points was very important.

```
String text = "Hello World";

//Store an index

int positionW = text.indexOf("W");

int wLen = text.length();

String w = new String();

w = text.substring(positionW, wLen-1);

//Replace w substring in text with Sun

System.out.print(text.replace(w, "Sun"));
```

Fig 4.2: Sample test case structure

This test case is one example of the test cases used. It was chosen due to its use in other published experiments. This would provide a benchmark to verify the obtained solutions against. For similar reasons, another test case involving lines of code in Fig 4.3 was chosen.

```
temp = Array[i];
Array[i] = Array[i+1];
Array[i+1] = temp;
```

Fig 4.3: Swap Space Code

Wiedenbeck [1] has shown that the segment of code shown in Fig 4.3 is considered to be a beacon. By including this particular test case, it was hoped to verify her experiment further and determine if the gathered data on students matches her results.

All the other test cases were chosen to span a range of difficulties, time to solve, mathematical and non-mathematical solutions. The aim was to get as much eye-tracking time on some test cases as possible to see where people focus their attention and gaze when calculating, and on the "easier" cases, to see if they hone in on one particular spot immediately upon seeing the code block. All the test cases created can be seen in Appendix A, with each image presented in the same order as they were presented during the experiment.

Section 4.2: Eye-tracking Equipment

Having discussed the issues relating to the Tobii eye-tracker in Chapter 3, it was determined that a different eye-tracker would be used to interface with the experiment builder of choice. This led to the decision to use the EyeTribe [11, 16].

The EyeTribe is an affordable, portable and modular piece of eye-tracking equipment – see Fig 4.4(a). It costs less than $100 and the main aim of its developer is to bring eye-tracking into everyday life. It can accurately determine the gaze of a person to an area the size of a fingertip[8].

[8] https://www.youtube.com/watch?v=2q9DarPET0o

The EyeTribe eye-tracker runs at frequencies of up to 60Hz. It has multiple cameras on both sides that work together to build up an accurate picture of where a person is focusing. These cameras can be seen in Fig 4.4(b). According to the developers, "The technology within relies on infrared illumination and uses advanced mathematical models to determine the point of gaze" [11].

Fig 4.4: (a): EyeTribe equipment and (b): EyeTribe cameras illuminated

The EyeTribe was a very straightforward product to use and it interfaced smoothly with OpenSesame (presented in Section 3.3). It produces very readable data files compared to the Tobii that are easier to analyse.

Section 4.3: Experiment Builder

In Chapter 3, the reasons for using OpenSesame as the experiment builder of choice were presented. OpenSesame has a comprehensive GUI, support for Python scripting and it is open-source [17]. All of this together made it very user friendly and intuitive to use.

For the main experiment, ten test case images were designed, developed and incorporated. OpenSesame makes this incorporation easy as it provides a "file pool". This means there would not be a need to have images linked from an external folder and the experiment could be run from a single executable. Each image formed the basis for each one of the trials and needed to be undertaken only once in a straightforward sequence. This will be discussed further in Section 4.4.

Another advantageous feature of OpenSesame is that it allows you to build any surveys and forms required into the interface of the experiment and collect the results in one file alongside all other behavioural data. This greatly reduced the amount of files needed for analysis. This allowed for the integration of a consent form in to the experiment and would be the first thing participants are presented with. All consent is stored digitally as a yes / no response and commencement of the experiment was cancelled if a participant chose to reject (this did not happen).

After consent for participation and a response was provided, and before implementation of the test cases, the participants were asked the following questions:

1. What age are you?
2. What gender are you?
3. What is your dominant hand?
4. Would you consider yourself a "Morning Bird", "Night Owl" or neither one?
5. How many years have you been programming for?
6. Rate your Java knowledge from 1-10
7. Is it Morning, Lunch, Afternoon, Evening or Night right now?

These questions are generally standard to any neurological experiment. They provide a lot of insight between different groups / classifications of people and often present interesting analytical points. These could be as simple as “Are males or females better at understanding Java?” or as complex as “Are Morning Bird’s working in the evening less likely to perform optimally?”. While the developed experiment does not aim to answer these questions directly, some data was obtained that looks at questions like this in a minor fashion. This will be presented in Chapter 5.

After the completion of the survey, the PyGaze elements of the experiment activated. PyGaze is the data collection toolkit that is in-built into OpenSesame. This toolkit provides the ability to collect eye-tracking data with relative ease. A compatible eye-tracker is connected and initial set-up steps are completed. The output from this is a file (.tsv) that contains 60 data points per second of accurate eye location data. One key feature of the PyGaze blocks in OpenSesame is the ability to add custom log items. This means that the experiment can log the start time of a given test case and a finish time for the same test case with precise accuracy. This is vital when it comes to interfacing with PyGazeAnalyser at a later stage.

The next step involves the main sequence of the experiment. This is simply a loop over all 10 of the created trials. Each iteration contains the following sequence of events:

1. A fixation Dot is presented in the centre of the screen for 3 seconds (The purpose of this is to regain any potentially lost focus and to act as verification when analysis is done).
2. Mark the start of test case in log.
3. Run the test case (until a keyboard response is entered).
4. Mark the end of test case in log.

In theory, this is quite a simple structure with some control required for the test cases. The important element for this is known as the conditions table. The conditions table for this experiment can be seen in Fig 4.5. As can be observed, it allows for the definition of any variables required. In this case, the correct response (column 1) for each posed question was defined, the 4 possible answers named ‘left’, ‘right’, ‘up’ and ‘down’ (columns 3-6) to correspond to the input keys and a counter (column 7) to keep track of what test case is currently running. A “test_case” variable (column 2) was also defined which contains the names of all the image files within the file pool.

	correct_response	test_case	left	right	up	down	count
1	up	1_PREPILOT	1	2	3	4	1
2	down	2_PREPILOT	1	2	3	x	2
3	left	3_PREPILOT	'cT'	'Ct'	'S '	' S'	3
4	up	4_PREPILOT	World	Sun	Hello Sund	Hello Sun	4
5	left	5_PREPILOT	FalseTrue	TrueFalse	TrueTrue	FalseFalse	5
6	down	6_PREPILOT	error	54	45	55	6
7	up	7_PREPILOT	0	1	2	3	7
8	left	8_PREPILOT	40	75	60	25	8
9	right	9_PREPILOT	60	90	75	110	9
10	left	10_PREPILOT	36	25	48	24	10

Fig 4.5: Conditions Table

With these variables in place, the final piece of the puzzle could be assembled, namely, the on-screen display. This is performed using a "sketchpad" block. This allows one to draw, import or otherwise present any stimuli required. The main sketchpad view for this experiment is shown in Fig 4.6. In the bottom right corner of the screen the possible answers are presented (highlighted inside the red box). The square brackets signify that the named element is a variable from the conditions table and the correct set of information should be picked based on the current test case. Similarly the [?] block in the middle of the screen is a variable image item which picks the correct image for the current test case and formats it correctly.

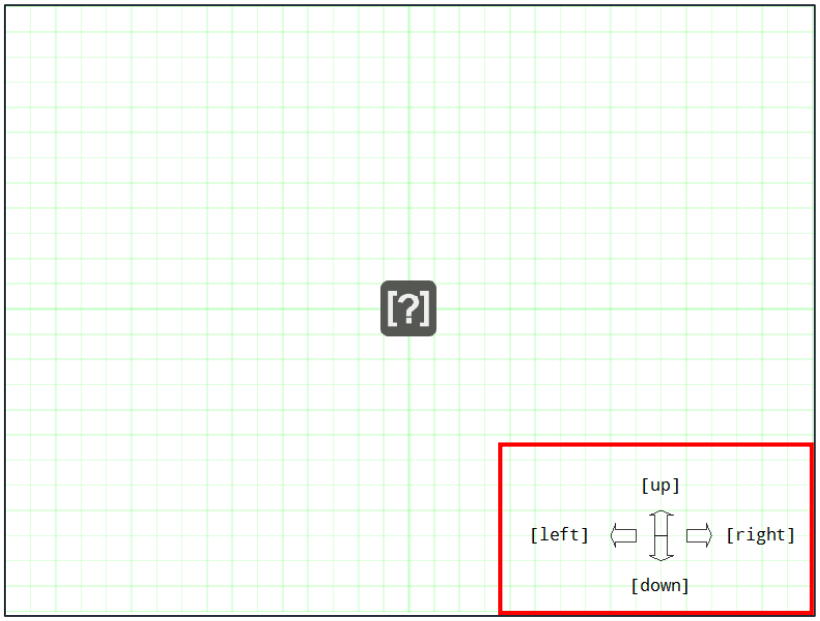

Fig 4.6: Sketchpad Block

After the completion of the trials, a post survey was presented to the participants similar to the pre-survey. This survey posed the following questions:

- How many do you think you got right?
- Did you feel focused?
- Did you feel that the following lines of code gave a significant hint as to what question 7 was doing?

```
temp = Array(i);
Array(i) = Array(i+1);
Array(i+1) = temp;
```

The first two questions would simply add some extra efficacy data that could be analysed in minor detail. The third question is quite important as it will aid in verifying Wiedenbeck's [1] previously determined beacon findings. As soon as this survey was completed, the participants were told how many questions they got right and were told the experiment was finished. If they had any questions about particular code implementations or any goals of the project, this was discussed this with them before they left.

Section 4.4: Experiment Implementation

Once the first draft of the experiment was completed, organisation of the testing phase began. Before the experiment could be ran on the main target audience, a few other elements of importance had to be examined first. These included:

- Deciding on the best location in which to implement the experiment.
- Deciding on external variables of importance and setup.
- Distribute a sign-up sheet to recruit undergraduates.
- Run a pilot test on some selected people to ensure everything is working correctly.

Location:

The Cognitive Science Laboratory is located on the third floor of the Eolas building. It is a state-of-the-art lab filled with eye-tracking equipment and associated hardware. It is painted white so as to have no distraction bias during the undertaking of any experimentation. Access is also limited with only one entrance and no external windows. It is used mainly for psychological and educational experiments. Access was granted in early December to begin preparation. Running this experiment in any other location would have been very challenging, so it was very fortunate that the department has such a valuable provision.

Experimental Setup and Participant Acquisition:

The Cognitive Science Lab was booked for periods of time over a two week timeframe. During the first week, the focus was on getting everything relating to the experiment just right. Other external variables that the room itself does not counter, needed to be eliminated. In particular:

1. The participants would all need to be seated comfortably, while having their heads at the same relative level. The EyeTribe calibration software made this an easy task (see Fig 4.7).
2. All participants would need to be the same distance from the screen in order for the analysis software to be accurate. The chair used was kept in a fixed position to ensure that any participant was approximately 61cm from the screen when sitting upright.
3. The EyeTribe would have to remain in the same position (apart from small vertical angle changes for participants of different heights) throughout all experiments. The product was placed directly below the monitor and was not moved over the course of the experiments.
4. Every participant would need to have close to 20/20 vision (20/20 corrected vision was acceptable as the EyeTribe does not have a problem with most lenses).

In terms of participants, it was important to get people of different types. In order to gauge interest in undertaking the experiment, a sign-up sheet was sent around the 4th year lab and the research labs. A range of times were offered between 10am and 7pm in order to accommodate everybody. About 30 interested candidates were found of which 24 were chosen to participate. The rest were placed on a reserve list as there were only 24 timeslots available. Two reserves were called upon. Of the final 24, 19 were male and 5 female. Some mature students volunteered as well as the expected majority in the 20-23 age range. Another variable that was tested was glasses wearers. These were taken to verify the EyeTribe developers claim that glasses work with their device.

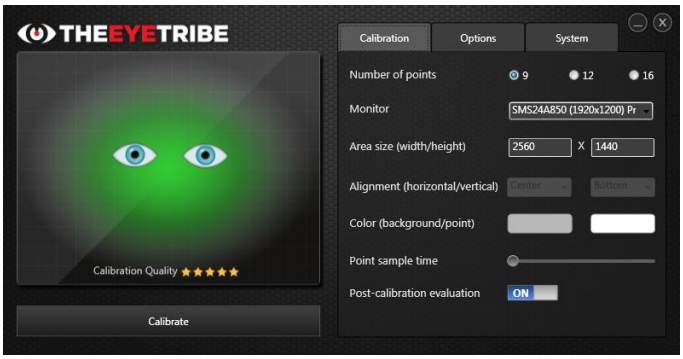

Fig 4.7: The EyeTribe Calibration Tool

Pilot Testing:

The main body of participants were tentatively scheduled. A number of days before commencing the main body of work, a pilot testing phase was ran with 5 chosen subjects. A number of those tested at this stage had previous experience in experiment running and they were able to give me some invaluable feedback. A number of small errors in the code test cases were also identified and remedied before they caused any data issues. No data for these participants was analysed, the log files were merely verified as containing reasonably accurate data and all their data was then deleted. This phase also solidified that all external variables were being dealt with appropriately and gave practice time in detailing every step of the set-up with participants.

Main Experiment:

The main experimentation phase took phase on Tuesday 8th and Wednesday 9th of December 2015. Twelve participants were brought in each day for a period of approximately 30 minutes each (some people took less than the allocated time). The exact rundown of participant timeslots (no identifiable information is provided) and a summary of their efficacy results can be seen in Appendix B and is also further discussed in Chapter 5.

The order of events for each participant was as follows:

1. Bring the participant to the Cognitive Science Laboratory.
2. Get the individual seated and comfortable.
3. Run the EyeTribe calibration tool and ensure that a high quality calibration is achieved. Also ensure that the monitor is set to 60Hz at this point for best data collection.
4. Inform the individual about what to expect in the experiment (without giving away any information as to why they are taking on the task, so as not to introduce a bias).
5. Open the OpenSesame environment and run the experiment for the participant. Participants were “named” based on their entry order; “subject-1” and so on.
6. Have the participant make the necessary agreements and talk them through to the second calibration phase. Allow the participant to take as much time as needed to complete the experiment while quietly ensuring it was running properly from the other side of the room.

After the two days of experimentation was complete, the experimentation phases were finished. Analysing the data would be performed next and drawing conclusions from the collected data.

Chapter 5: Evaluation

Section 5.1: Experiment Output

The experiment produced two different output files for each participant. One file contained eye-tracking data (.tsv) that was purely produced by the EyeTribe and logged by PyGaze. The other data file (.csv) was produced by OpenSesame and contained information such as survey responses, responses to each trial, whether a response was correct or not, how long it took to answer each question, overall experiment runtime and a copy of the correct answer for quick reference.

All of this produced data was collected in raw form. The goal of this Chapter is to present an overview of elements of both data files in such a way as to make some conclusions about what was learned. All original raw data files as well as all produced results are included in the supporting section of this report and descriptions of how to interpret them are also included.

Section 5.2: Eye-tracking Analysis

Interpreting the eye-tracking data file is not easily done manually. A toolkit was to be used that would read in the data file, partition it based on trials and produce visual outputs that would make the results clearer to analyse. The first tool that was tested was called Ogama [18]**.** This tool however was not built specifically for the data files produces by this experiment. After some further searching, PyGazeAnalyser [19] was discovered. This toolkit was written by a member of the OpenSesame team, Edwin Dalmaijer, and was built to directly interact with OpenSesame output files. Some difficulties were experienced in getting this toolkit functional, but through liaison on the development forum, a working analysis script was modified for use with this experiment. The final analysis script is included in the supporting section.

Fig 5.1 contains example representation of the four types of output files produced by this analysis script. From the top right corner moving left to right these are:

a) A fixation map, showing where the participants gaze rested for a period longer than half a second. The bigger the circle, the longer the fixation.
b) A heat map, showing the areas most fixated on.
c) A raw data file, simply showing all areas the participant rested their gaze on.
d) A scan path, showing what are known as “saccades”. These are the rapid movements of the eye from one fixation point to the next. It can be seen in the example that the first fixation began in the middle of the screen and then moved to the top left of the code. This is due to the fact that participants were asked to focus on the fixation dot in the centre of the screen in between trials. This acts as verification for the correctness of the produced outputs.

A set of images like these was produced for each test case in each run of the experiment. This produced 960 individual pieces of analysis. As such, the discussion of potential beacons that follows is merely possibilities. Full verification of results would take a much longer period of time than is available. Work will be continuing in this regard over the coming weeks in preparation for publication of a paper.

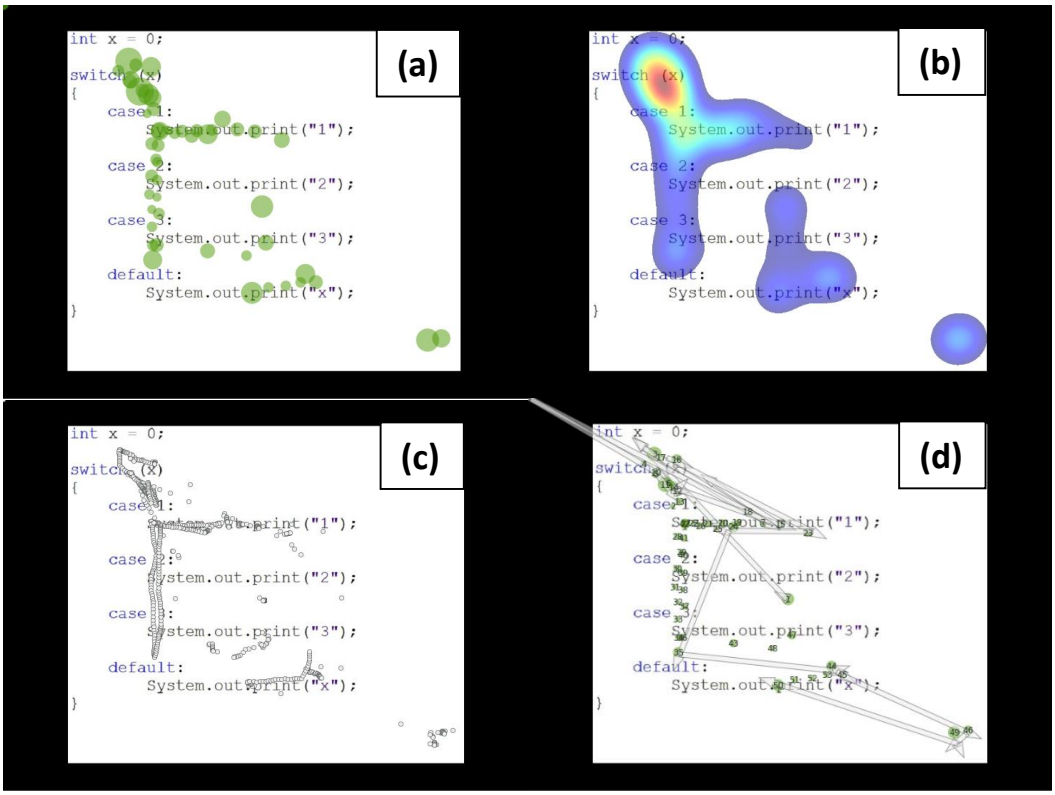

Fig 5.1: (a): Fixations, (b): Heat Map, (c): Raw Data, (d): Scan Path

The following is the beacon information found for each test case (as presented in Appendix A):

- Question 1: Most successful candidates looked at the int declarations frequently and then glanced at each if condition and possible answer briefly. Most did not read the final else statement as it wouldn't need to be executed. The if conditions are potential beacons here.
- Question 2: Some candidates looked through each case statement individually. Others recognised that the answers were linear and jumped straight to the default case. As such, no clear beacon emerged in this question.
- Question 3: Fixations were widespread during this question, but the largest time was spent on line 1 looking at the string definition. This would imply that people are manipulating the string in their mind. This does not necessarily point to it being a beacon however.
- Question 4: This question had a lot of code and was quite a difficult problem. As such, much data was produced during this test case. No clear beacons can be identified.
- Question 5: This question further inferred that the if statement conditions could be beacons.
- Question 6: Most participants realised that the array declaration was just the numbers 1-10 and they did not fixate here long. Many did however fixate on the loop calculation (result += Array[i]) for an extended period of time. This has the potential of being a beacon.
- Question 7: Contrary to Wiedenbeck's work, less time was spent looking as the "swap space" code than other areas of this question. This does not directly disprove her theory, but the findings do not point to it being a beacon. Most focus was on the array initialisation.
- Questions 8/9/10: These questions were all nested loop based. As was expected, successful candidates spent most time fixated on the innermost loop calculation as well as the bounds of each loop. The bounds of the loop may be a beacon in these test cases.

Section 5.3: Efficacy Analysis

When efficacy is mentioned in relation to this experiment, it refers simply to the number of questions each participant answered correctly. This data alone is quite powerful, and when combined with the results of the psychological survey, statistical statements can be made about whether particular groups were able to perform better than others. To do this, mean analyses were performed on a subset of the possible "types" of person. If combined with p-testing, this would give us a statistical figure about whether established results are significant or not. The full results sheet is provided as Appendix B, this contains the raw experiment data as well as a table of analyses done.

The first point of note is that the mean number of correct questions formed an upper bounded normal distribution with 2 participants getting 3 or 4 questions right, 10 people getting 5 or 6 right, 10 people getting 7 or 8 right and 2 people getting 9 or 10 right. This infers that the experiment was fair and a good proportion of participant types were chosen. Following from this, it was determined that the mean number of correct answers across all participants was 6.458333.

The following are some of the analyses that were performed on the dataset:

- Males had a mean result of 6.421053 while females had a mean result of 6.6. These results are negligibly different from the overall average and as such a p-test was not performed.
- Due to the small sample sizes, specific breakdowns of the "Morning Bird or Night Owl" question did not progress past mean testing. However, some interesting trends did begin to show. People who claimed to be morning people seemed to perform worse in the morning than in the evening, which is contrary to expectations. This can be seen in Fig 5.2.
- Left handed people had a mean result of 6 (sample size 4) and right handed people had a mean result of 6.55 (sample size 20). Calculation of a p-score via a two tailed t-test resulted in p = 0.516203. This implies that there is no statistically significant difference between left and right handed participants.
- One interesting result was that as age group got higher, mean result got higher also. The 18-20 age group had a mean result of 5.4, 20-23 year olds had a mean result of 6.4 and 23+ year olds had a mean score of 8.
- There was no significant difference between participants with 2 or 3 year Java programming experience (mean 6.5) and those with 4 or 5 years' experience (mean 6.4375).
- Participants who answered "No Difference" to the Morning bird/ Night owl question performed significantly worse than the overall mean result (mean 5.857143).

Type of Person vs. Time Experimented	Mean	Count
Morning People vs. Morning / Lunch	6.333333	3
Morning People vs. Afternoon	7	1
Morning People vs. Evening / Night	7	2
Neither People vs. Morning / Lunch	6	1
Neither People vs. Afternoon	5.5	2
Neither People vs Evening / Night	6	4
Night People vs. Morning / Lunch	8	2
Night People vs. Afternoon	6.6	5
Night People vs. Evening / Night	6.25	4

Fig 5.2: Mean Analysis of Person Types

Chapter 6: Conclusions

The field of eye-tracking has become more prevalent in recent years and there is growing interest in using eye-tracking within Computer Science Education. This project attempted to verify multiple pieces of software, hardware and researcher's theories relating specifically to eye-tracking and programming comprehension. It has provided a large corpus of data from experienced programmers that has been analysed and has made some solid findings relating to beacons. The dataset still contains more information that will be analysed as part of future work. The implementations could easily be modified to ask many other eye-tracking related questions also.

As future work, a paper will be published on the initial findings of these beacons before work can begin on running the second phase of the experimentation on 1st year students. This could be run in a similar fashion to the second phase of the "Duncker's tumour" experiment [4] described in Chapter 2. During this phase, a group of students would be shown the test cases with the beacons highlighted. The results would be compared to a control group of students to see if learning in novice programmers can be improved through (accurate) directed attention. This is an important issue due to the ever increasing need for competent technology graduates in industry.

The undertaking of this project has been a very beneficial learning experience. If this project could be undertaken from the beginning again, some things would have been best approached differently. Some elements of the experiment itself could have been better built to make things easier in the analysis stage and to improve the focus of the participants. For example, counterbalancing of the test cases would be performed in order to reduce the chance of participant fatigue. In the current design, the test cases get harder incrementally but this may act as a deterrent once people get past a certain point. Fatigue is a strong possibility that may have resulted in some negative data from perfectly capable individuals. A number of the test cases also involved some basic (but potentially complex) arithmetic. A suggestion arose in the pilot testing phase for implementation into future experiments – namely, a "doodle pad" area could be provided inside the experiment builder to allow the participant to write some basic notes. While this would help with fatigue, it may distract the eyes from their main task and hence interfere with the data.

Sourcing of an eye-tracking analysis toolkit would also be done much earlier. One of the largest time sinks in the project went into fixing up the analysis script. If this had been vetted before data collection, everything would have run much smoother and trial separation log elements could have been better implemented.

Now that this analysis script is functional however, it is ready made for any future work that needs to be analysed in eye-tracking by the author. A PhD in eye-tracking will be undertaken from the next academic year, so this is invaluable to have completed. The same is the case for all the experience in experiment building and research skills that have been gained.

To conclude, this project has helped answer many questions about beacons, but there is still much to learn. With future research, we can harness this knowledge for the benefit of education.

Bibliography

[1] S. Wiedenbeck, "Beacons in computer program comprehension," *Int. J. Man. Mach. Stud.*, vol. 25, no. 6, pp. 697 – 709, 1986.

[2] T. Busjahn, C. Schulte, B. Sharif, Simon, A. Begel, M. Hansen, R. Bednarik, P. Orlov, P. Ihantola, G. Shchekotova, and M. Antropova, "Eye tracking in computing education," *Proc. tenth Annu. Conf. Int. Comput. Educ. Res.*, pp. 3–10, 2014.

[3] C. O'Brien, J. Humphreys, and N. Ide McAuliffe, "Concern over drop-out rates in Computer Science courses," *Irish Times*, 2016. [Online]. Available: http://www.irishtimes.com/news/education/concern-over-drop-out-rates-in-computer-science-courses-1.2491751. [Accessed: 10-Feb-2016].

[4] E. R. Grant and M. J. Spivey, "Eye movements and problem solving," *Psychol. Sci.*, vol. 14, no. 5, pp. 462–466, 2003.

[5] R. Brooks, "Towards a theory of the comprehension of computer programs," *Int. J. Man. Mach. Stud.*, vol. 18, no. 6, pp. 543–554, 1983.

[6] E. Soloway and K. Ehrlich, "Empirical Studies of Programming Knowledge.pdf," no. 5, pp. 595–609, 1984.

[7] E. Soloway, B. Adelson, and K. Ehrlich, "Knowledge and Processes in the Comprehension of Computer Programs," in *The Nature of Expertise*, 1988, pp. 129–152.

[8] C. Aschwanden and M. Crosby, "Code Scanning Patterns in Program Comprehension," *Proc. 39th Hawaii Int. Conf. Syst. Sci.*, 2006.

[9] M. E. Crosby, J. Scholtz, and S. Wiedenbeck, "The Roles Beacons Play in Comprehension for Novice and Expert Programmers," in *14th Workshop of the Psychology of Programming Interest Group*, 2002, pp. 58–73.

[10] Tobii, "Tobii X60 and X120 Eye Tracker," *www.tobii.com*, 2014. [Online]. Available: http://www.tobii.com/en/eye-tracking-research/global/products/hardware/tobii-x60x120-eye-tracker/.

[11] "The EyeTribe." [Online]. Available: https://theeyetribe.com/. [Accessed: 14-Jan-2016].

[12] J. W. Peirce, "PsychoPy-Psychophysics software in Python," *J. Neurosci. Methods*, vol. 162, no. 1–2, pp. 8–13, 2007.

[13] S. Mathôt, E. Godefroid, F. De Groot, L. Van Der Linden, and E. Ort, "Step-by-step tutorial (beginner)," 2016.

[14] E. S. Dalmaijer, S. Mathôt, and S. Van der Stigchel, "PyGaze: An open-source, cross-platform toolbox for minimal-effort programming of eyetracking experiments.," *Behav. Res. Methods*, pp. 1–16, 2013.

[15] R. Bednarik, "Eye Movements in Programming Datasets 2014." [Online]. Available: http://emipws.org/datasets-2014/. [Accessed: 20-Mar-2016].

[16] E. S. Dalmaijer, "Is the low-cost EyeTribe eye tracker any good for research?," *PeerJ Prepr.*, vol. 4, no. 606901, pp. 1–35, 2014.

[17] S. Mathôt, D. Schreij, and J. Theeuwes, "OpenSesame: An open-source graphical experiment builder for the social sciences," *Behav. Res. Methods*, vol. 44, no. 2, pp. 314–324, 2012.

[18] A. Vosskühler, V. Nordmeier, L. Kuchinke, and A. M. Jacobs, "OGAMA (Open Gaze and Mouse Analyzer): open-source software designed to analyze eye and mouse movements in slideshow study designs.," *Behav. Res. Methods*, vol. 40, no. 4, pp. 1150–1162, 2008.

[19] E. Dalmaijer, "PyGaze Analyser," 2015. [Online]. Available: http://www.pygaze.org/2015/06/pygaze-analyser/. [Accessed: 15-Mar-2016].