5-2007

The Effects of Gaze Guidance on Educational Software

Brian Murphy
Clemson University, bjmurph@clemson.edu

THE EFFECTS OF GAZE GUIDING ON EDUCATION SOFTWARE

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Science

---

by
Brian Murphy
May 2007

Accepted by:
Dr. Brian Malloy, Committee Chair
Dr. Andrew Duchowski
Dr. Wayne Madison

Abstract

Eye tracking technology is on the brink of becoming ubiquitous. Thus, a need exists for new applications for this potent technology. There are currently three main kinds of eye tracking applications: gaze-responsive, gaze-aware, and gaze-contingent. A fourth classification, termed gaze-guiding, to our knowledge, coined and implemented for first time. The gaze-guiding technique is the use of motion, light, color, or other visual stimuli to modify the user's fixations to pre-determined locations when the user fixates on specific areas of interest. To test the technique, an education software program that teaches Newtonian physics through the use of gaze-guidance was developed. It is suggested that a natural mapping exists between gaze-guidance and the software's built-in lesson plan. It is also speculated that gaze-guidance reduces the extraneous cognitive load of associating written and visual problem elements. An experiment was conducted to evaluate the effectiveness of gaze-guidance. Although not found to significantly affect performance, most participants considered gaze-guidance helpful, especially for difficult problem examples.

Acknowledgments

To Amy Pope, for use of her students, Dr. D, for all the help, All the dorks,and to my family and parents.

Table of Contents

List of Figures

Chapter 1

Introduction

Eye tracking as a technology has reached a point where mainstream usage is only held back by the cost of hardware.There is now a wealth of interesting applications, pertaining to Human-Computer Interaction, Human-Centered Design and Training, and Computer Mediated Collaboration. Eye tracking applications can broadly be classified as *gaze-responsive*, *gaze-aware*, and *gaze-contingent*. A novel interactive style has been implemented, termed ***gaze-guiding***. Gaze-guiding applications are ones that attempt to direct gaze by drawing visual attention to locations deemed important by the software. Note carefully the distinction between such applications and *gaze-guided* applications. Gaze-guided applications are merely gaze-aware and gaze-responsive in that the software tracks real-time gaze location and responds to the user in some way. The archetypal gaze-guided application is eye typing, where the user guides the software cursor to select a word or phrase of choice and thus evokes a response from the software. In contrast, gaze-guiding applications reverse this interactive cycle: a visual stimulus cue is first presented by the software to draw the user's attention. Naturally, gaze-guiding applications must also be gaze-aware in order to ascertain whether the user fixated the intended visual stimulus.

The software attempts to guide the user's gaze to a predetermined location via attention drawing behavior. If the user has not yet fixated the currently cued location, the software escalates its attention-drawing behavior, e.g., instead of simply highlighting the cued location, blinking is turned on. This style of attention-cueing behavior persists until the current cue is fixated or the software times out. This is somewhat analogous to blinking web page banner ads, although banner ads are not gaze-aware. To be gaze-guiding, banner ads would need to turn off once fixated.

An example would be a document displaying application, similar to Adobe Acrobat.

The software deems that you should read this introduction before any other section of the document. To do so, the application may highlight the section heading. If you fixate the cued location, then the software proceeds to cue the next location (perhaps the next sentence, or next section heading having ascertained that you have dutifully read every word of the introduction), and so on.

To our knowledge, no gaze-guiding applications currently exist with the exception of this implementation. To demonstrate gaze guidance, a piece of education software for teaching Newtonian physics was implemented. The software is a collection of Newtonian physics word problems with accompanying visual representation. The words in the word part of the problem make use of gaze guidance to draw attention to the relevant part the visual portion. It is suggested that there is a natural mapping between gaze guidance and the software's built-in lesson plan.

It has been shown that visual representation is a valid and useful tool in teaching concepts in science to students. Students often find visual representations of abstract concepts, such as physics formulas helpful to learning these concepts [Cheng 1999]. However, novice users, because of their lack of knowledge, lack the skill to pick out the relevant portions of a visual display. Irrelevant information can confuse the novice learner, and distract them from the relevant portions of the display. Also, relevant information may not be as salient as irrelevant information, and missed. For example, in a problem about the trajectory of a dolphin leaping out of the water, the fact that the object in motion is a dolphin is irrelevant, however, that might be more interesting than the numbers pertaining to the dolphin's speed and angle [Hegarty et al. 1991; Linn 2003]. Novices expend much of their cognitive resources interpreting the visual and verbal information separately and are left with few resources to link the representations. The highly interconnective nature of word problems and their corresponding visual representation produces extraneous cognitive load. Extraneous cognitive load, as opposed to intrinsic cognitive load, can be reduced helping the student solve the problem faster. Novice learners can be overwhelm their working memory by extraneous cognitive load, the reduction of which can expedite learning [Sweller et al. 1998].

When students do not know what information to attend to, they are likely to draw in-

correct conclusions. Making relevant information more salient through gaze-guiding could free more cognitive resources for constructing an understanding of the underlying content. Because novice learners experience difficulty coordinating multiple visual components and interpreting animations, our gaze-guiding application serves to cue student attention to relevant information. Cueing, in this case, is defined as the manipulation of design elements to attract attention to specific portions of a graphic [Treisman 1986a; 1986b]. These manipulations take the form of movement, which should be picked up by the peripheral vision of the user. When the user reads a portion of the word problem and there is a relevant portion of the accompanied illustration, cueing occurs to induce the user to look at the relevant portion of the graphic.

Automatic cueing behavior is the crux of the gaze-guiding interaction style and is dependent on the availability of eye tracking technology in the classroom. For this reason the developed program is not ready for large scale use and implementation since it relies on the assumption of ubiquity of eye tracking devices. In order to achieve ubiquitous deployment, eye tracking technology must overcome the following hurdles:

- ease of use, require no calibration,
- unobtrusiveness, allow remote, automatic session startup, with free head motion,
- affordability, priced as a computer accessory, and
- ease of installation, be as easy to install as a plug-and-play mouse.

By briefly reviewing eye tracking's state-of-the-art, with one eye on technology and the other on applications, we will show that we are not isolated in our assumption, and that eye tracking researchers are on the verge of solving these problems. This implementation and the study conducted with it provides data on the usefulness of the technique of gaze guidance as a teaching tool, so that it may be deployed once the conditions are met.

Chapter 2

Background: Application Review

Many eye tracking applications exist. To our knowledge, none of them are gaze guiding. Several applications are similar, some in terms of goal, others in terms of implementation. To be called gaze guiding, and application must have the following features:

- the application must be gaze aware and gaze responsive,
- the application must provide visual cues to the user in real time,
- the application must use those cues to attempt to direct the user's gaze,
- the application must respond once gaze has been directed,
- the cues must be prepared ahead of time, not created dynamically.

Below, varying eye tracking application classes are discussed and shown how they differ from being gaze guiding.

2.1 Gaze-Based Selection

In early eye pointing work, the gaze pointing paradigm of "what you look at is what you get" was explored by Jacob [1990]. A particularly critical challenge is keeping "eye pointing" natural. Eye movement is normally associated with perception and not with action, and is often involuntary. Gaze-based target selection may accidentally trigger unintended actions, causing the "Midas touch" problem identified by Jacob.

While gaze guidance may have the user "select" objects with their eyes following visual cueing, the "Midas touch " problem is not as relevant. Gaze guidance attempts to direct a user's natural eye movements, rather than using the user's gaze to direct the program.

While the initial activation of gaze guidance might require some sort of selection, by taking advantage of a user's natural viewing habit, for example, starting at the top of the page in a paper, and distinguishing between fixations and saccades, there are no controls *per se* susceptible to the "Midas touch". The main reason for the distinction is that gaze guidance is not a means to control a device, be it a computer or some other gaze aware piece of technology. Rather, gaze guidance is a means for the technology to guide the user.

2.2 Eye Typing

The prohibitive cost of most eye trackers causes them to be, outside academic and corporate use, used to facilitate communication among the disabled. The majority of these provide a software keyboard, where dwell time serves as clicking. It should be noted that clear feedback is often given, as some disabled users are unaccustomed with controlling anything at all [Majaranta and Räihä 2002].

While gaze guidance could be used by the disabled, it was not specifically targeted to be. Gaze guidance is meant for the average computer user. For those that have difficulty moving their eyes, guidance of their scan path may not prove useful, and may result in frustration instead of intended clarity. Also, gaze guidance is not a system that obtains direction (input) **from** the user, as an eye typing system would. It is only meant to provide direction **to** the user, from the creator of the gaze guiding application. This is an important, if subtle distinction.

2.3 Gaming

Many newer commercial games include the concept of gaze-based interaction. Most obvious is the first-person shooter, where the field of view of the player's avatar is explicitly presented to the user. Other games build on natural human eye movements in the context of social interaction. For example, In The Legend of Zelda: The Wind Waker, a visual deictic reference is made by the player's avatar: the avatar indicates interest in a nearby object by looking at it. This is similar to gaze guidance but differs in an important way: the game

does not take any input from the user's eyes. While the game does attempt to direct the user's gaze, it does not measure actual gaze, it does not attempt to further alter the user's gaze after the initial cue, and it does not provide feedback once the user's gaze has shifted.

As demonstrated by Smith and Graham [2006], an eye tracker can increase one's sense of immersion in a video game and can significantly alter the gameplay experience. While gaze guidance could be used towards very similar ends, this usage is very similar to using the eyes as selecting tools. Here the eyes would be used in place of a manual controller. In gaze guidance, the eyes do not control the application, but rather move in response to the game's (visual) direction.

2.4 Mediating Human-Human Dialog

One application of eye tracking in computer-mediated human-human communication is the use of gaze to convey turn taking. Gaze direction can also communicate more accurately where participants in a video conference are looking. Vertegaal [1999] demonstrated a prototype implementation where images of participants rotated to reflect where they are looking. For a similar effect, eye gaze can also be used to drive the eyes of a virtual avatar.

A visual aid symbolizing a participant's gaze in a collaborative environment has been shown to be beneficial for disambiguating a deictic point of reference, especially when the user's line of sight is decoupled from their head direction [Duchowski et al. 2004].

Gaze guidance, while conveying information from a designer to a user, is not a person to person communication enhancing tool. While one could argue that the designer of the gaze guidance is providing information to the user, and thus communicating, it varies from gaze enhanced communication in several important ways. Gaze guidance must be programmed ahead of time. While the process of gaze guidance is dynamic, it remains static in the sense that modifications can not be made on the fly. Gaze guidance follows a pre-defined "script". Gaze guidance is intended for a single user, and communication, by definition, is intended for two or more. In gaze assisted communication, gaze based feedback is created and consumed by the users; in gaze guidance the user is not creating feedback. Finally, gaze assisted communication is not necessarily gaze responsive. The program that enhances the

communication does not necessarily react based on where the user is looking, but simply reports it in some fashion to the coparticipant.

2.5 Mediating Human-Machine Dialog

Because of the restraints imposed by the current limitations of artificial intelligence, conversations between humans and machines are very limited in terms of adaptability. Through the use of input from eye trackers, applications can seem more 'intelligent' and can take cues from the eyes instead of traditional modes of input, such as clicking, or verbal commands. For example Sibert et al. [2000] implemented *The Reading Assistant*, an eye tracking program that says words when a reader focuses on them over a certain threshold. This was to help remedial readers, who often had trouble with certain words. Programs such as these should not be confused with actual intelligence, as they are analogous to simply adding more pages to the book in Searle's [1985] *Chinese room problem*.

In a more complex example Qvarfordt and Zhai [2005] developed a system called *iTourist* that uses this principal. Much like how an attentive person would take advantage of the listener's eye gaze, iTourist reasons about the tourist's interest, shows with call-out pictures and with speech synthesis tells the tourist about different places of the city. An algorithm analyzes the eye gaze patterns and adapts the dialog accordingly, telling more if the tourist is intensely focused on the current place being talked about, or cutting the story short if the tourist's eye gaze wonders away.

Such applications are certainly gaze aware and gaze responsive. However, they are not gaze guiding. These applications are similar to gaze guiding application in that they seek to provide feedback based on gaze position. However they do not attempt to direct the user's gaze position, which gaze guiding applications seek to do.

2.6 Designing Web Pages

As the Internet becomes increasingly more interwoven into everyday life, the issue of web site design becomes more prevalent. Web designers need information on user habits in order

to improve site design. Currently, this is mostly done through the use of web server logs. However, this only allows for a very broad picture of user behavior. The logs can record to what pages and for how long a user looked at a page, but not what specific elements the user looked at on a page. Eye tracking can be used to compliment server log analysis used by web page designers by seeing which specific page content user are looking at and for how long. Eye tracking provides an ability to perform a detailed analysis of how users use a specific web page, and modify the layout of the web page in order to direct the user gaze [Beymer et al. 2005].

While the study of web page design attempts to direct a user's gaze, it does so in a passive way. A user's eye data is analyzed, then the page is modified and improved, and the cycle continues. Gaze guidance is a real-time process, modifying the user's gaze and providing feedback to the user in real time.

2.7 Training With Scanpaths

There are many jobs where the safety of the end user depends upon the visual inspection of the product, ranging from the food and drug industries, to the manufacturing of various products. Since not all defects are obvious, there are certain techniques that experts use in order to find them. The training of new personnel can be expedited through the comparison of the scan path between experts and novices. Expert scan paths tend to be systematic in nature. Novice scan paths tend to be more random. A systematic scan path to discover defects tends to be more efficient, in terms of both accuracy and speed. By providing novices feedforward training, both of these performance measures can be increased [Sadasivan et al. 2005].

Scanpath-based training is very similar to gaze guidance. Both attempt to direct the gaze of the user. It is in how they attempt to direct the user's gaze behavior that they differ. Generally, feedforward training is presented all at once, as in the entire scanpath of the expert is presented at one time allowing the novice user to follow it. In gaze guidance, the intended scanpath is not immediately visible. Scanpath-based training is not gaze responsive. The user is presented with a scan path to follow, however, no feedback is immediately

given on the scanpath that the user finally takes. Finally, the intent of scanpath-based training is to shape the scanpath of a user for a certain task. Eventually, the user is expected to perform similar tasks without the use of the expert scanpath. Gaze guidance, however, is intended to be used regardless of the level of skill of the user. It was never intended to be turned off.

Chapter 3

Implementation

To test gaze-guiding, a gaze-guiding software application was implemented. The implementation took the form of a piece of education software which was used to teach physics concepts. The program was written in Microsoft Visual C#. It was run on a AMD Opteron Processor 246, 1.99 Ghz with 2.00 GB of RAM. The operating system was Microsoft Windows XP. The coding was performed in Microsoft Visual Studio 2005.

Essential to the program, as well as most other eye tracking programs, is the ability to differentiate between fixations and saccades. Since the essence of gaze-guiding hinges on determining where a person is fixating, moving the fixation to a pre-defined place, and detecting the movement in real time, an algorithm was needed that was not only able to determine the difference accurately, it also had to run concurrently with the rest of the program, without noticeably slowing the entire program. In this implementation, velocity-based detection was used. The user's eye coordinates (x, y) are transformed from the eye tracker's normalized reference frame into degrees visual angle. In this implementation, the last (buffered) five measurements are passed through a filter [-2, -1, 0, 1 2], calculating the velocity of the user's gaze. A threshold was set at 200 deg/s, with filtered velocities above it denoted as saccades, and those below deemed fixations. This algorithm was chosen because it is fast, with constant run time (on the order of ten operations). Other algorithms were considered, such as position-variance based detection, and were rejected because of speed constraints.

Subjects were cued to look at objects in the visual portion of the problem through the use of motion. This motion was achieved through a 'shaking' effect. This was implemented by randomly selecting a number between 0 and 1/50, and adding that number to the object's OpenGL x and y coordinates (with $x, y \in [-1, 1]$) every time the scene was drawn. This

translates the object by between 0 and 20 pixels in the x direction and 0 to 25 in the y direction.

There are two distinct forms that a user will see when they run this program, the 'setup' form and the 'problem' form.

3.1 The Setup Form

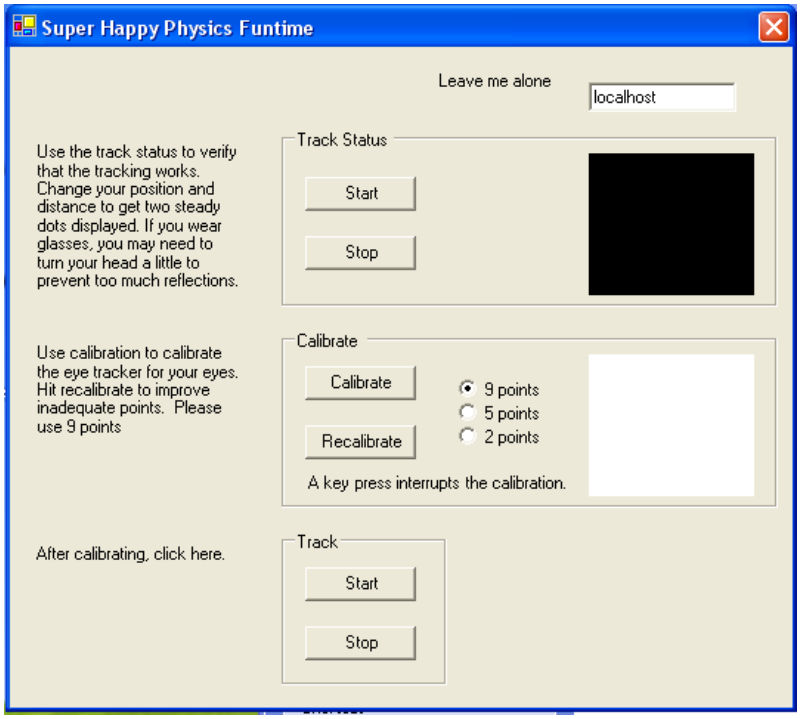

Figure 3.1: The setup form, the initial screen the user sees, used to calibrate the eye tracker

The setup form, shown in Figure 3.1, is the initial form that appears when the program is started up. Its primary function is to provide feedback to the user about the ability of the eye tracker to measure their gaze. No gaze- guiding is implemented on this form, however, it provides essential information for the eye tracker. Functionality for finding a comfortable and suitable position for the user, and calibrating their eyes is also found here. The functionality of this form is almost entirely implemented by the Tobii API, and as such, not discussed in detail here.

3.2 The Problem Form

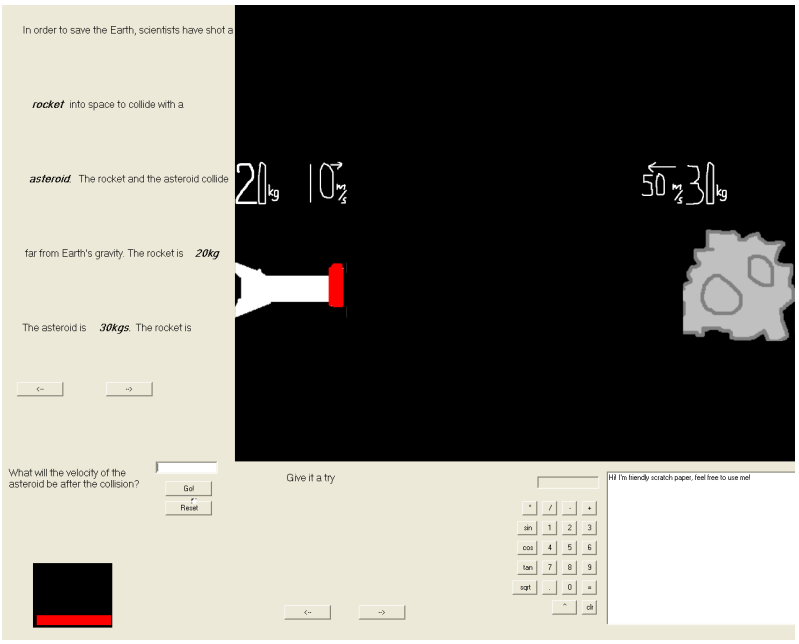

Figure 3.2: A typical problem

The problem form, Figure 3.2, is where gaze-guiding actually occurs. In the bottom left corner is the tracking position feedback and shows where the user's eyes are relative to the tracker's camera. Further to the right is a 'hint box', designed to help the user along with the problem, and prevent frustration if the user does not understand the problem. Next to the hint box is a calculator and 'scratch paper'. Since the eye tracker can only track the users eye if they are looking at the screen, the calculator and the 'scratch paper' are provided to keep the users gaze on screen. These parts of the problem are incidental to the main objective of the program, to test the concept of gaze-guiding. The two major sections that do this are the written component of the problem and the visual component of the problem.

3.2.1 The Written Component

Figure 3.3 shows the written component of the problem. Displayed here is the setup or 'word problem', and the specific question the program is posing to the user. Due to several limitations, only five lines of setup text can be displayed at a time. In order to allow more than five lines of text per problem, the arrow buttons at the bottom may be used. By clicking

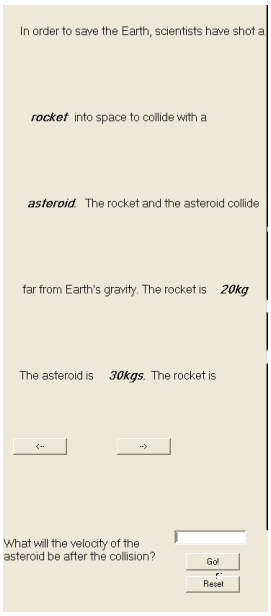

Figure 3.3: The written part of the problem

on them, the user may navigate backwards and forwards through several blocks of text. Throughout the text, there are italicized words. Each of these words has a corresponding element in the visual component of the problem. These italicized words, when gazed at, cue the user to look at their corresponding visual component. This implements the the gaze-guiding portion of the program.

Each italicized word has a bounding box around it, which checks for the gaze of the user. When the user gazes at the italicized word, such as during the course of reading the problem, the program detects this, and cues the relevant visual portion of the problem. In order to set this relationship between the words and the objects in a procedural fashion, as opposed to manually creating bounding boxes around these words for each word, the number of elements that can cue a response in the visual portion is limited to one per line. Manual bounding box creation is not provided and implemented, however, finding the correct coordinates manually is time consuming and requires a large amount of trial and error.

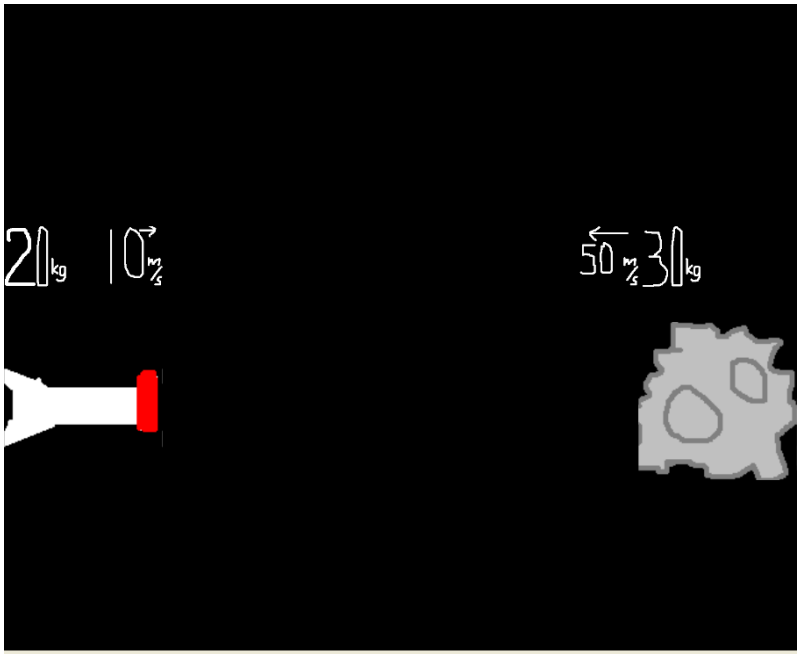

Figure 3.4: The visual part of the problem

3.2.2 The Visual Component

The visual component of the problem is realized using OpenGL. OpenGL is not native to C#, so an open source wrapper library called CSGL (`http://csgl.sourceforge.net/`) was required. CSGL is based upon OpenGL and works almost identically. However, some functionality, specifically blending did not work. For our experiment, all graphical objects are simple 2D boxes, with texture maps. More complicated objects are possible, but since the focus of the experiment was on gaze-guiding and not sophisticated visual components, they were not used. Each object in the visual component can be cued by a trigger in the written component, usually a word or words that are relevant to that visual component. Upon being triggered, the object shakes, in the hope of drawing the attention of the user. The object continues to shake until the user gazes at it, at which time it resumes its original position.

The visual component is also animated. It is also a rudimentary physics engine, which can detect collisions and react to those collisions. It takes into account mass, forces, acceleration, and velocity. The collision detection algorithm is bounding box-based. Collision reaction takes into account mass, velocity, and angle of impact. It is also possible to vary

an object's various physical attributes based on the answer given by the user. For example, in one of the problems in our experiment, ball is thrown into the air at a specific velocity, attempting to reach a certain height. Based on the answer the user inputs, the ball travels at a different speed. While the program was originally intended to teach physics, it is easily adaptable to other types of problems.

Chapter 4

Experimental Validation

4.1 Objectives

The objective of this experiment was to create a gaze-guiding application that would teach physics concepts similar to those taught in an introductory college level physics class. Upon completion of the experiment, it is hoped that the participant will have learned something about the concepts behind simple Newtonian physics. While this is being accomplished, the goal of the gaze-guiding application is to facilitate this process, by managing attentional distribution, and lessening the cognitive load required for the user to understand the problem.

There were several reasons for the choice of physics problems. First, a physics engine for the visual display could be implemented relatively easily. Second, physics is a system with well defined rules with unique quantitative answers. Physics problems can also be divided into textual and visual components, with strong connections between the two. Finally, although not everyone understands the exact mathematics of motion and collisions, most people are at least somewhat familiar with them from their everyday experiences.

Problems for the program were inspired by physics text books [Sayer 1970], [Fryshman 1970], [Chen 1974].

4.2 Hypothesis

It was hypothesized that participants that were aided by the use of gaze-guiding would on average, be more accurate and would complete the problems faster than those not assistedby

gaze-guiding. It was also expected that users with the gaze-guiding would subjectively find the application more enjoyable than those without.

4.3 Design

There were two groups in the experiment: one group with gaze guidance, the other without (the control group). A between-subjects design was therefore employed with gaze guidance as the sole independent variable.

4.4 Procedure

Each participant was first asked to sign an informed consent waiver. After consenting to participate in the experiment, the participant completed a short questionnaire to gather demographic and computer use data. After completing the questionnaire, the participant was given a paper handout with Newtonian physics formulas on it. The participant was instructed to read the handout in its entirety and told that there would be problems based on the handout during the computer portion of the experiment. The handout took three to five minutes to read, but no time limit was placed upon the user. After the participant read the handout, the eye tracking portion of the experiment began. The eye tracker was calibrated to each participant using nine calibration points. The participant was then given some practice sessions with the eye tracker. The practice sessions consisted of a problem very similar to the ones contained in the educational software to familiarize the participant with the interface of the educational software as well as they eye tracker. The participant was asked that if they required a short break, and took one if needed.

For participants with the gaze guiding condition, the education software makes use of gaze guiding to direct the user's gaze to specific points in the program, based upon what the participant looks at. Participants without the gaze guiding condition did not make use of gaze guiding. The only difference between the two conditions is in the software used. In both conditions, data on the participant's gaze location, and fixation, were collected by the Tobii Eye Tracker. Time to completion and correctness were gathered by the program.

For participants with the gaze guiding condition, they then started the educational software. The eye tracker was recalibrated using nine calibration points. The participant then went through ten Newtonian physics problems, assisted by the gaze guiding technique. The order of the problems was randomized for each user, but each user completed the same problems. Completion of these problems took between twenty minutes to an hour, though no time constraint was placed upon the participant. After the participant was finished, a short interview was conducted about the use of the program and their preferences. The participant was then be debriefed, thanked for their time, and allowed to leave.

In the case of participants without the gaze guiding condition, they would also start the educational software. The eye tracker was also recalibrated using nine calibration points. The participant then went through ten Newtonian physics problems. The order of the problems were randomized for each user, but each user completed the same problems. Completion of these problems took between twenty minutes to an hour, though no time constraint was placed upon the participant. After each participant was finished, a short interview was conducted about the use of the program and their preferences. The participant was then debriefed, thanked for their time, and allowed to leave.

4.5 Subjects

Participants were college undergraduate and graduate students, with knowledge of physics ranging from none at all to familiarity with basic physics. Participants were excluded if they did not know how to read or could not read at or above a high school level, were unable to see, or had great difficulty seeing, or had a strong grasp of the concepts behind Newtonian physics. There were ten participants in all, divided equally into the gaze guidance and non-gaze guidance groups. Three of the participants were female, and seven were male. The average and median age was 20. No participant had any serious eye related injury or illness, but one was farsighted and three were nearsighted. All participants were familiar with computers and used them daily. All but two had taken some sort of calculus and all had taken trigonometry. Nine had experience with educational software, though most of this came from playing with computer programs during childhood. Only one had any

Total Average Time Control (seconds)	Total Average Time Experimental (seconds)	p-value
2171.338	2435.532	0.5678

Table 4.1: Results giving completion time for all problems

Total Average Score Control	Total Average Score Experimental	p-value
23.2	26.6	0.6385

Table 4.2: Results giving total score for all problems where score is defined as the number of wrong answers.

experience with eye tracking, but only briefly as a participant for another experiment.

4.6 Results

Data describing accuracy and time taken were recorded for each individual problem, as well as pooled for total times and total score. ANOVA was performed for each individual problem and the combined totals from all problems on the number of incorrect answers, referred to as 'score'. Time elapsed from the start of the problem to the time the participant clicked the 'Next' button was tabulated. An incorrect answer was defined as entering an answer in the answer box that was not the number that was programed into the software as the correct answer. The score for the problem was defined as the sum of the incorrect answers. Results are given in Tables 4.1, 4.2, 4.3, and 4.4.

No statistical significance was found between the control and experimental groups, in aggregate or in individual problem statistics. The highest p-value was .9153, observed between the differences in score for the Super Fast man problem (see Figure B.10). The lowest p-value was 0.0872, found in time to complete the Rocket car problem (see Figure B.8). This value is the only value that tended towards significance.

4.7 Discussion

It is speculated that the lack of significant effect was obscured by other effects. The most prominent of these is most likely the small sample size. Due to time constraints and recruit-

Problem	Average Time Control (seconds)	Average Time Experimental (seconds)	p-value
Ants	43.322	64.628	.1285
Asteroid	378.874	418.81	.7392
Ball	203.346	241.722	.6935
Bunny	72.762	57.668	.3004
Hunter	541.77	487.04	.8543
Ice	226.634	178.628	.5094
Moose	185.006	145.702	.5858
Rocket Car	530.442	304.484	.0872
Squid	68.018	58.762	.7149
Very Fast Man	239.738	213.894	.76331

Table 4.3: Completion times for individual problems

Problem	Average Score Control	Average Score Experimental	p-value
Ants	0.2	1.2	0.1434
Asteroid	4.6	5	0.3466
Ball	1.6	1.8	0.8882
Bunny	0.8	.6	0.6666
Hunter	5	6.8	0.3852
Ice	1.6	2	0.7078
Moose	1.4	2.25	0.712
Rocket Car	4.4	4.2	0.8819
Squid	1.2	0.6	0.6666
Very Fast Man	2.4	2.6	0.9153

Table 4.4: Scores for individual problems, where score is defined as the number of wrong answers

ment problems, only five participants were recruited to each group. Another consideration is the variable proclivity of each participant toward physics. While efforts were made to recruit participants with the same knowledge level, not all participants possessed similar skill levels, test taking ability, and (perhaps most important) levels of interest. In hindsight, an experiment where the participant does gaze-guiding problems and non-gaze-guiding problems, and the two are compared, might show more significant results (in other words, a within-subjects design may be more suitable).

Some participants complained of fatigue from the number of problems. A smaller number of problems may have precluded fatigue. Some participants also complained about the lack of physical paper with which to perform calculations. It may be that unfamiliarity with the use of a text box as 'scratch paper' increased the cognitive load for these users, slowing them down, and possibly causing them to make mistakes. Four out of the ten participants made negative comments about either the calculator or the 'scratch paper'.

One measurement, the time to complete the Rocket car problem, tended towards significance. This may be because this problem has the largest number of visual elements. There are three vectors, each with angle and length visual representation, and the rocket car itself. In all, ten visual elements. The next largest number of visual elements is six, and some problems had as few as three. By this measure, the Rocket car problem was the most involved, complicated by the vector angles being close to each other, making it important to differentiate which values corresponded with which vectors. One participant referred to this problem specifically when asked if they found the gaze-guiding helpful, "The one with the car rocket was greatly clarified." It is speculated that a significant effect may emerge given more subjects. In future experiments, problems with smaller numbers of visual elements should be omitted.

It is also encouraging to note that only one participant in the gaze-guiding group responded negatively when asked if they found the gaze-guiding helpful, "No, It was kind of distracting." It should be noted that the participant was the least experienced with computers (though still familiar) and also commented "I'm happy I don't have to do physics again," perhaps indicating a predisposition to disliking the program as a whole. Four of thefive par-

ticipants indicated that the gaze-guiding helped them find things or clarified things in the picture.

Chapter 5

Conclusion

Although no statistically significant effect was found for gaze guidance, most users found the novel interaction style helpful, particularly for completion of the most involved problem (the Rocket car problem, see Figure B.8). It is possible that with a larger sample size the effect on performance may be more pronounced. In future studies, the current experiment may be improved through a smaller number of problems. It is recommended that those problems should have a large number of relevant visual/word components, at least ten. Screening for more motivated participants would also be helpful as well as ensuring a more uniform level of learned skill. Experimental design should invoke within-subjects testing, with half the problems performed with the use of gaze guidance and half without. Finally, a third group may be added, where in addition to the current gaze-guiding technique, a 'bouncing ball' would more strongly guide the user's gaze through the word part of the problem as well as its visual component.

APPENDICES

Appendix A

Subjective Answers

The questions and answers that follow are the results of the post-test interview conducted after the participant had completed interaction with the educational software. Although the answers are not quantitative, they do provide some insight into the participants' experiences. The answers are ordered by participant number.

A Gaze-Guiding Group

- What was your overall impression of the program?

    1. I don't like physics. It would have been easier if I had my glasses.
    2. That was torture. I really didn't like the calculator. Also splitting the problem between two pages was annoying.
    3. That was cool. It was pretty neat.
    4. I thought it was cool when you see the word the object moved. It helped me find things in the diagram quicker. It was hard to memorize all the formulas
    5. It was good. The moving graphics helped me locate things while I was reading, but after I was done, it was distracting.

- What was the hardest problem?

    1. The rocket car
    2. The monkey
    3. The meteor
    4. The monkey

5. The monkey followed by asteroid

- What was the easiest problem?

  1. The ants
  2. I don't remember
  3. The ball throwing one
  4. Very Fast man
  5. Squids

- Please rate the program on a scale from 1 to 10, 1 being the worst thing ever, 10 being the best in terms of educational software.

  1. 3 or 4
  2. Maybe like a 5
  3. 7 or 8
  4. 5 or 6 because the formula wasn't clear.
  5. 7, it could show you where to apply.

- Did you feel that your eyes were being drawn to certain places?

  1. When there was movement.
  2. I felt like you were trying to.
  3. Yah.
  4. Sometimes. I was attracted to moving things.
  5. Oh yah, when I was reading the problem.

- Did you find the gaze-guiding helpful?

  1. At times, Sometimes, the one with the car rocket was greatly clarified.
  2. No, It was kind of distracting.
  3. Yah, it was helpful for clarifying which numbers in the problem description correlated to the ones in the picture.

4. Somewhat, it only helped you find things faster if there was a bunch of numbers on there.
5. The moving graphics helped me locate things while I was reading, but after I was done, it was distracting.

- Do you feel more knowledgeable about physics?
  1. No, it makes me feel more dumb
  2. No.
  3. Yes.
  4. Not really.
  5. I feel like I've been refreshed on a few things.
- Would you like to see this type of program used in the classroom?
  1. I guess.
  2. I think it would be better for giving tests than for teaching.
  3. Yes.
  4. Maybe, it might work for some.
  5. Its better than a PowerPoint
- If you had a child who was learning physics, would you buy a program similar to this one for them?
  1. Yah.
  2. I'd probably just get them a tutor.
  3. Yes.
  4. Yes.
  5. Maybe.
  6. No, I'd rather work with them with a book.
- Comments

1. (None)
2. I'm happy I don't have to do physics again.
3. My only real problem was things were a little blurry because of my nearsightedness. Other than that it was fine.
4. The screen was a little bright.
5. I still think the monkey problem is screwed up.

B Non-Gaze-Guiding Group

- What was your overall impression of the program?

  1. The calculator needs a negative button, but other than that it was ok.
  2. That wasn't fun. I didn't understand some of the questions. I like to use a pencil and paper, you can't manipulate a computer like a piece of paper.
  3. Fine, I didn't like the formula immediately. Compress the problems. The calculator was a little hard use.
  4. It seems like it would be good for gathering information, but I can't see any purpose besides that. Its kinda creepy seeing those red lights there.
  5. Pretty good, I 'm not a very good test taker. The feedback in the bottom distracted me. Overall I thought it was pretty neat.

- What was the hardest problem ?

  1. The asteroid.
  2. The rocket problem, I just pressed 0 till it gave me the answer.
  3. The monkey problem.
  4. Very Fast man.
  5. The one with the monkey.

- What was the easiest problem ?

1. The moose.
2. Rocket.
3. (None)
4. Monkey.
5. The squid.

- Please rate the program on a scale from 1 to 10, 1 being the worst thing ever, 10 being the best.

1. 7
2. (undecided)
3. About a 5 it was pretty average.
4. About a 5
5. 9

- Did you feel that your eyes were being drawn to certain places?

1. No.
2. No.
3. No.
4. Kind of. The animation[after go was clicked] was too fast fast.
5. Yah, to the feedback.

- Do you feel more knowledgeable about physics?

1. Nope.
2. No, Id already encountered these formula before.
3. No, I knew these formulas before.
4. Slightly.
5. A little bit.

- Would you like to see this type of program used in the classroom?

    1. Possibly, it needs to be refined more.
    2. No.
    3. Probably.
    4. I don't know, probably not. It seems better for testing.
    5. Yah

- If you had a child who was learning physics, would you buy a program similar to this one for them?

    1. Maybe, depends on the physics they are in.
    2. It was kinda strenuous doing that many towards the end, and I was just putting in 0.
    3. Yah.
    4. Yes.
    5. Id considers it.

- Comments

    1. (none)
    2. I didn't know the notepad wouldn't erase each time.
    3. Yah.
    4. The animation was funny.
    5. (none)

Appendix B

Screenshots

Figure B.1: Problem “Ants”, a problem about vector addition

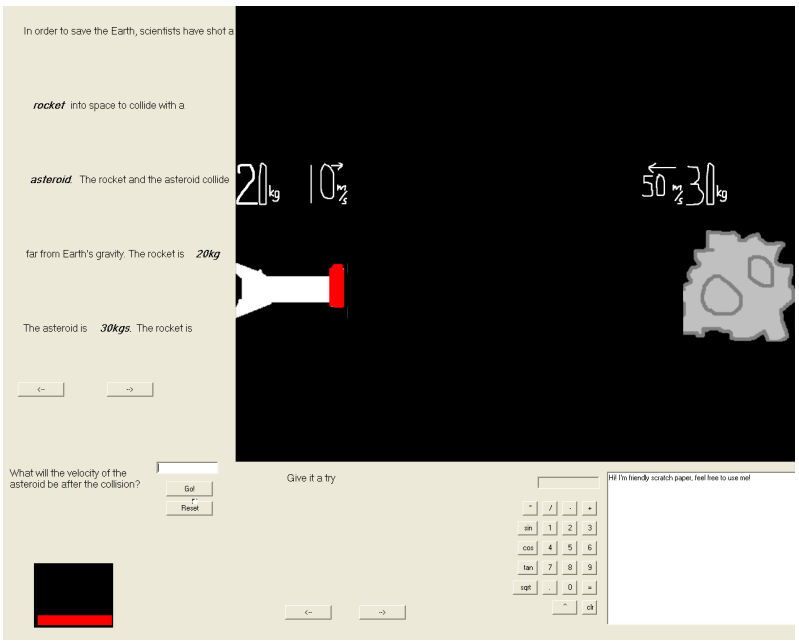

Figure B.2: Problem “Asteroid”, a problem about the collision of two moving objects

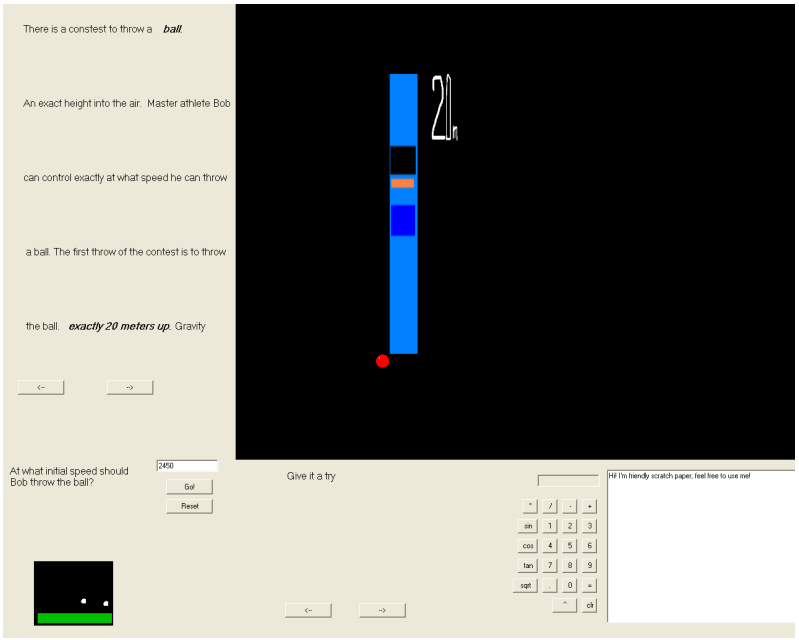

Figure B.3: Problem “Ball”, involving velocity, distance and constant acceleration

Figure B.4: Problem "Bunny", involving finding the kinetic energy of the fluffy bunny of doom

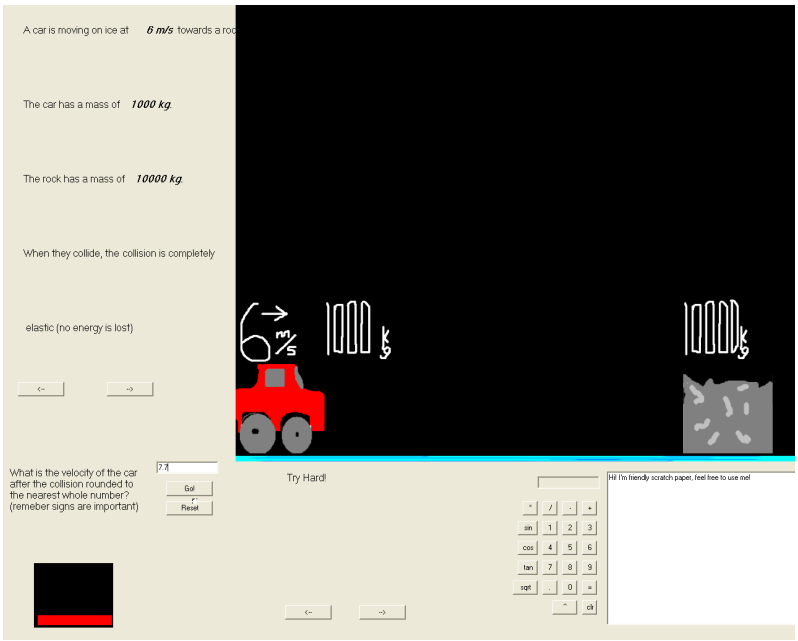

Figure B.5: Problem "Ice", about the collision of a moving object with a stationary one

Figure B.6: Problem “Hunter”, a problem about projectile motion

Figure B.7: Problem “Moose”, another problem about projectile motion

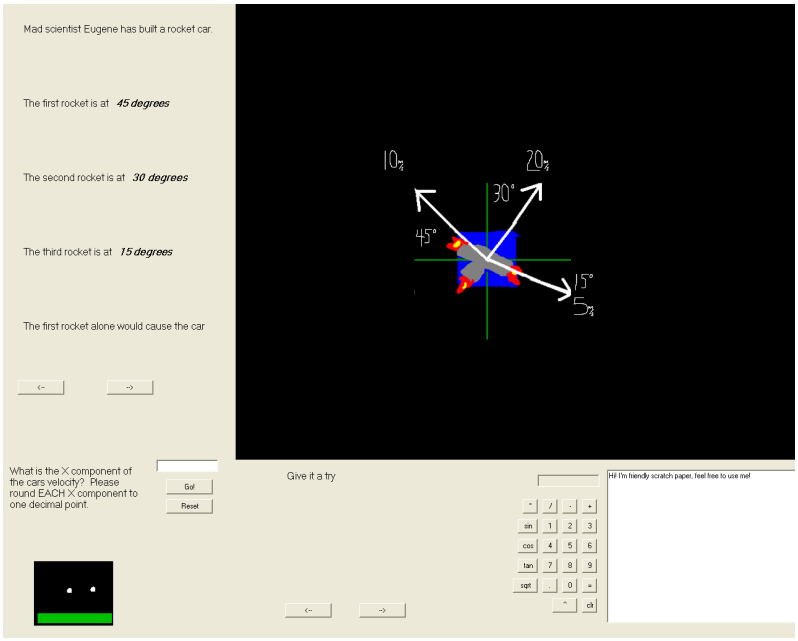

Figure B.8: Problem "Rocket Car", another vector addition problem

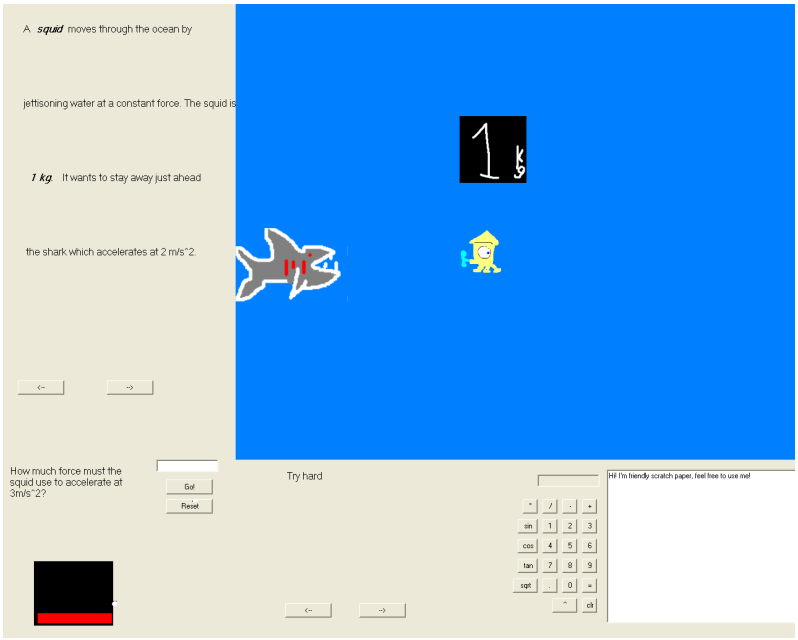

Figure B.9: Problem "Squid", involving force equal mass times acceleration

Figure B.10: Problem “Very Fast Man”, another problem involving $F = m \times a$ and kinetic energy

Appendix C

Tutorial

A Getting Started

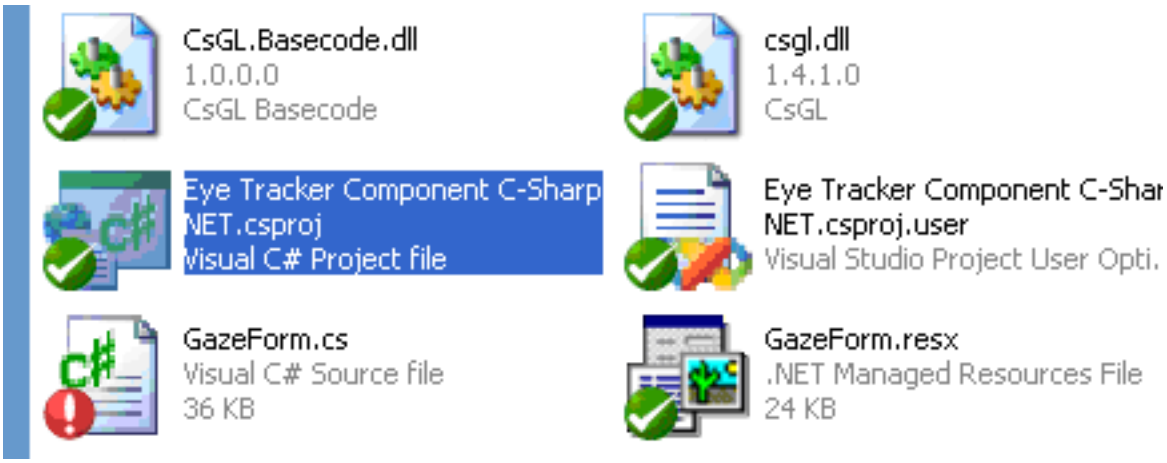

Figure C.1: Tutorial 1

To get started look for the Visual C# project file as seen in Figure C.1 . Double click on the icon. A screen like the one shown in Figure C.2 should open up. In the upper right menu, go to ”View” and then ”Class View”, as shown in Figure C.3 or hit ”ctrl+shift+C”. A window like one in Figure C.4 should pop up. Expand the "eye tracker component c-sharp net" and double click on "GazeForm" as shown in Figure C.5 You should see some code from Gazeform.cs. Scroll down until you see the comments as shown in Figure C.6. As the comments suggest, this is where we will be adding out code.

B Your First Problem

The first thing we need to do is declare a problem. We do this by adding the code

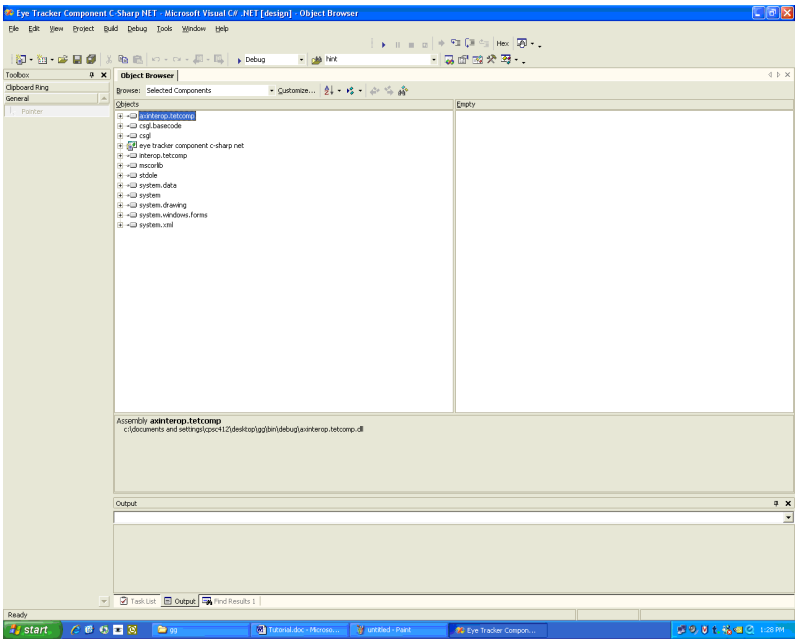

Figure C.2: Tutorial 2

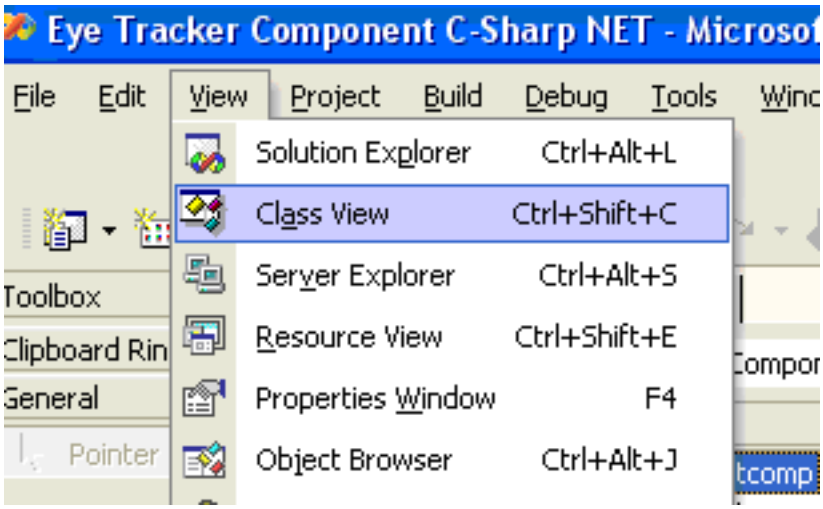

Figure C.3: Tutorial 3

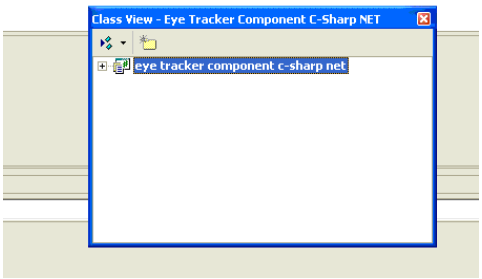

Figure C.4: Tutorial 4

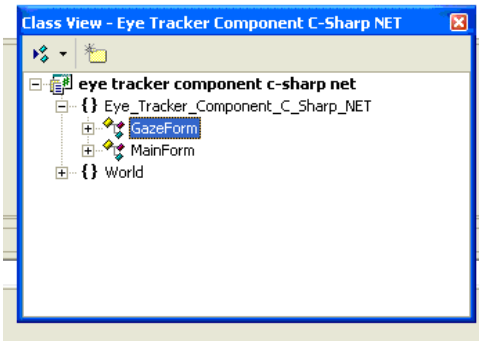

Figure C.5: Tutorial 5

```
bb1.setPhysics(false);
    bb2.setPhysics(false);
    bb3.setPhysics(false);
    bb4.setPhysics(false);
    bb5.setPhysics(false);

//  Here is where you add code.  Don't touch anything else


//  End of area where you add code.
```

Figure C.6: Tutorial 6

```
problem problemOne = new problem();
```

We now have a problem to work with. One of the first things we should decide is the scale of the problem. We do this with the setMetersPerPixel(float f) function. This function determines how many meters there are in pixel. The default value is 1, but this is not very useful. Since the monitor is 1280x1024, 1 meter per pixel is a very large area. Generally, it is a good idea to set it to however large you want it across, divided by (1280.0-376.0)). This value corresponds with the visible area of the physics engine. For example, if we wanted the visible area to be 40 meters across

```
problemOne.setMetersPerPixel((float)(40.0/(1280.0-376.0)));
```

would do so. Now that our scale is set up, let's add some things to the world.

C Things

Things are the objects that populate the physic engine. By adding things to a problem, we add objects the problem. To create things, we declare it like we declared a problem

```
thing ball = new thing()
```

"ball" is just the name of the thing, and does not confer any properties on the thing in question. We will add them now.

```
ball.setPhysics(true);
```

This gives the thing physics. If the thing does not have physics, it is invisible, and has no effect on other things in terms of physics. Things with no physics can be used as bounding boxes to active other things for gaze guidance, but more on that later.

Next we need to give the ball an IDs number. This number will allow us to refer to the ball in order to activate it for gaze-guiding. All integer values are valid for IDs, except for -1. -1 is used by the program to signify that no object is should be shaking. If an object is given the ID of -1, it may behave inappropriately. To give the ball an ID of 4

```
ball.setId(4);
```

If the thing has a picture associated with it, we can set that picture with setTex(LoadTextures(string filename)) function. The filename should be a bitmap that has a power of two dimensions (8x8 128x128, 512x512 etc.) If we had a bitmap called ball1.bmp that was in a folder called "images" which was in the .exe's folder we could set the texture like this

```
ball.setTex(LoadTextures("data/ball1.bmp");
```

Let's say we want other things to go through this particular thing, as opposed to colliding and bumping off of it. This is particularly useful for labels. To do this we have to set the setIntangible(bool b) to true. For example

```
lable.setIntangible(true);
```

creates a label that does not collide with other things. setIntangible is false by default, so you do not have to set it if you do want your thing to collide with other things.

If you want to make an immovable floor or wall object the setImmobile(bool) and setFloor(bool) or setWall(bool) flags can be set in the same way. setFloor and setWall must be used in conjunction with setImmobile in order for them to have any effect. Now we should set the mass of the thing. We do this with a call to setMass.

```
ball.setMass(1f);
```

setMass sets the mass of the thing in kilograms. Mass is used calculating physics. The default mass of a thing is 1 kg. Now let's position the thing in the world. We can do this one of two ways. We can do this by screen coordinates, or we can place it in relative meters. To place the thing with screen coordinates use the setBB(float topX, float topY, float bottomX, float bottomY). setBB is based upon screen coordinates. The top left of the screen is (0,0). The bottom right is (1,1). For example, to place a thing whose top left corner is at the exact middle of the screen, and extends to the bottom corner of the screen.

```
ball.setBB(.5f, .5f, 1f, 1f);
```

This type of placement is usually reserved for things that do not have physics. The other way of placing things is the setInMeters(float xPos, float YPos, float xSize, float ySize, float dimension) function. This function sets things relative to the top left visible portion of the visual part of the problem. The first two arguments are the x and y coordinates in meters of the top left corner of the thing. The next two arguments are how big in meters the thing is in the x and the y direction. The final argument is the scale of the world, which should be the same as the scale we set up earlier. For example, to make a 2 by 4 meter the ball 10 meters to the right and 5 meters down

```
setInMeters(10.0f, 5.0f, 2.0f, 4.0f, (float)(40.0/(1280.0-376.0)));
```

This method of positioning things is useful when the distance between things is important. If when the problem starts you want the object to be moving there are several options

Initial Velocity

- setInitialXVel(float meterspersecond)
- setInitialYVel(float meterspersecond)

Constant Force

- setConstantXForce(float joulespersecond)
- setConstantYForce(float joulespersecond)

Constant Acceleration

- setConstantXA(float meterspersecondsquared)
- setConstantYA(float meterspersecondsquared)

Initial velocity sets in meters per second how fast the thing is going. Constant force exerts a constant joules per second, causing the object to accelerate. Constant acceleration accelerates the object at a constant rate. This is useful for simulating gravity or forces like

it. Finally, when you have your thing the way you like it, add it to the problem with the addThing function. You must use the problem that you want the thing to appear in. To add the ball to the problem problemOne we declared above

```
problemOne.addThing(ball);
```

D The Word Problem

Now that the visual part of the problem is set up, we have to set up the word part of the problem. The program can only display five lines of text at a time. Each line is set by calling its own function. P1 refers to a problem.

```
p1.addSentenceLineOne("pre", "target", "post" , int IDofReactingThing);
p1.addSentenceLineTwo("pre", "target", "post" , int IDofReactingThing);
p1.addSentenceLineThree("pre", "target", "post" , int IDofReactingThing);
p1.addSentenceLineFour("pre", "target", "post" , int IDofReactingThing);
p1.addSentenceLineFive("pre", "target", "post" , int IDofReactingThing);
```

"target" refers to the words that you want to trigger the shaking of an object. "pre" and "post" refer to the words that are before and after the triggering words. IDofReactingThing is the ID of the thing that you want to react. Let's say you wanted to the sentence "I have a ball that bounces." to appear on the screen. When the user looks at the word ball, you want a thing with the id of 4 to react.

```
p1.addSentenceLineOne("I have a", "ball", "that bounces.", 4);
```

Note that you must call all five functions. To create an empty line

```
p1.addSentenceLineTwo(" ", " ", " ", -1);
```

To set the question for a problem, use the setQuestion(string s) function. For example.

```
p1.setQuestion("At what speed should the helicopter be moving in order for Peter
to land perfectly on the moose?");
```

To set the answer to the question use the setAnswer(float answer, int IDofAffected, int type) function. The answer parameter is the answer to the question. IDofAffected allows you to pick which item in the visual representation will be affected by the answer. type specifies how it will be affected. For example, to set the answer to 25, affecting the thing with the id of 1, and have it affects that things velocity in the x direction.

```
p1.setAnswer(25f, 1, p1.VELOCITYX);
```

There are several types of ways a thing can be affected, velocity in the x and y direction, acceleration, and force.

The last thing that we need to set is the hints. Hints are revealed after a user gets a problem wrong. Hints must be entered in the order that you wish to display them. Do this using the addHint(string s) function. For example.

```
p1.addHint("This is a hint");
```

After we have entered all the hints we want, we have to enter the problem into theproblem

vector. This is done by adding it to the vector called "problems". To add problem p1,

```
problems.add(p1);
```

Bibliography

Beymer, D., Russell, D., and Orton, P. Z. 2005. Wide vs. Narrow Paragraphs: An Eye Tracking Analysis. In *Proceedings of INTERACT 2005, IFIP Conference on Human-Computer Interaction*. 741–752.

Chen, M. 1974. *Universtiy of California, Berkeley Physics Problems with Solutions*. Prentice-Hall, Inc, Englewood Cliffs, New Jersey.

Cheng, P. C. H. 1999. Unlocking conceptual learning in mathematics and science with effective representational systems. *Computers & Education 33*, 109–130.

Duchowski, A. T., Cournia, N., Cumming, B., McCallum, D., Gramopadhye, A., Greenstein, J., Sadasivan, S., and Tyrrell, R. A. 2004. Visual Deictic Reference in a Collaborative Virtual Environment. In *Eye Tracking Research & Applications (ETRA)*. ACM, San Antonio, TX.

Fryshman, B. 1970. *Problem Solving in Physical Science: for Nonscience Majors*. Addison-Wesley Publishing Company, Reading, MA, USA.

Hegarty, M., Carpenter, P. A., and Just, M. A. 1991. Diagrams in the comprehension of scientific text. In *Handbook of reading research*, R. Barr, M. L. Kamil, P. Mosenthal, and P. Pearson, Eds. Longman, New York, NY, 641–668.

Jacob, R. J. 1990. What You Look at is What You Get: Eye Movement-Based Interaction Techniques. In *Human Factors in Computing Systems: CHI '90 Conference Proceedings*. ACM Press, 11–18.

Linn, M. 2003. Technology and science education: starting points, research programs, and trends. *International Journal of Science Education 25,* 6, 727–758.

Majaranta, P. and Räihä, K.-J. 2002. Twenty Years of Eye Typing: Systems and Design Issues. In *Eye Tracking Research & Applications (ETRA) Symposium*. ACM, New Orleans, LA.

Qvarfordt, P. and Zhai, S. 2005. Conversing with the user based on eye-gaze patterns. In *Proceedings of the ACM Conference on Human Factors in Computing Systems, CHI'2005*. 221–230.

Sadasivan, S., Greenstein, J. S., Gramopadhye, A. K., and Duchowski, A. T. 2005. Use of Eye Movements as Feedforward Training for a Synthetic Aircraft Inspection Task. In *Proc. CHI '05*. ACM Press.

Sayer, M. 1970. *Notes and Problems in Applied Physics*. Heinemann Education Books Ltd, London, Great Britain.

Searle, J. R. 1985. *Minds, brains, and programs*. MIT Press, Cambridge, MA, USA.

Sibert, J. L., Gokturk, M., and Lavine, R. A. 2000. The reading assistant: eye gaze triggered auditory prompting for reading remediation. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*. ACM Press, New York, NY, USA, 101–107.

Smith, J. D. and Graham, N. 2006. Use of Eye Movements for Video Game Control. In *ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE)*. ACM, Hollywood, CA.

Sweller, J., van Merrienboer, J., and Paas, F. 1998. Cognitive architecture and instructional design. *Educational Psychology Review 10*, 251–296.

Treisman, A. 1986a. Features and Objects in Visual Processing. *Scientific American 255,* 5 (November), 114B–125,140.

Treisman, A. 1986b. Properties, parts, and objects. In *Handbook of perception and human performance: Vol II*, K. Boff, L. Kaufman, and J. Thomas, Eds. Wiley, New York, NY, 1–70.

Vertegaal, R. 1999. The GAZE Groupware System: Mediating Joint Attention in Mutiparty Communication and Collaboration. In *Human Factors in Computing Systems: CHI '99 Conference Proceedings*. ACM Press, 294–301.