An Environment For Graphics Programming As A Spectator Sport - Final Report

Sponsor:
James Geller

Team:
Gary Cheng
Doug Kreiss
Ernesto Martinez
Mohammed Raza

CS 491-103
Dr. Osama Eljabiri

Chapter 1

Project Background:

Why make programming a spectator sport? We are interested in increasing the participation of underrepresented minorities and women in computer science education and in the computing job market. We are proposing a new approach based on the observation that many young people are attracted by spectator sports to become themselves active in the sport disciplines of their heroes.

Problem Definition:

We are arguing that hackathons are not sufficiently effective for broadening participation, e.g. of women, in computing. The same applies to an even higher degree to “adults” with family and job obligations. We propose a contrasting model model for creating interest in computing that is based on making it a spectator sport. We present a detailed design of the game, the physical layout of the venue for a computing competition as a spectator event, the structure and scoring of rounds and matches and the organization of tournaments.

Glossary:

Hackathons - An event, typically lasting 48 hours or more hours, for computer programmers, hackers, and enthusiasts to engage in collaborative computer programming.

Broadening Participation - An attempt to interest the common person into programming and making them participate in programming with a graphics coding game.

Coding Education - An approach to teaching computer science through coding.

Spectator Sport - A sport that many people find entertaining to watch.

Revision Update:

The latest version of the program includes

-Ability to manually enter values into the time (minutes) and constraint (percentage) boxes

-Help button for referee screen located underneath player status boxes

-Help & grid button for each player

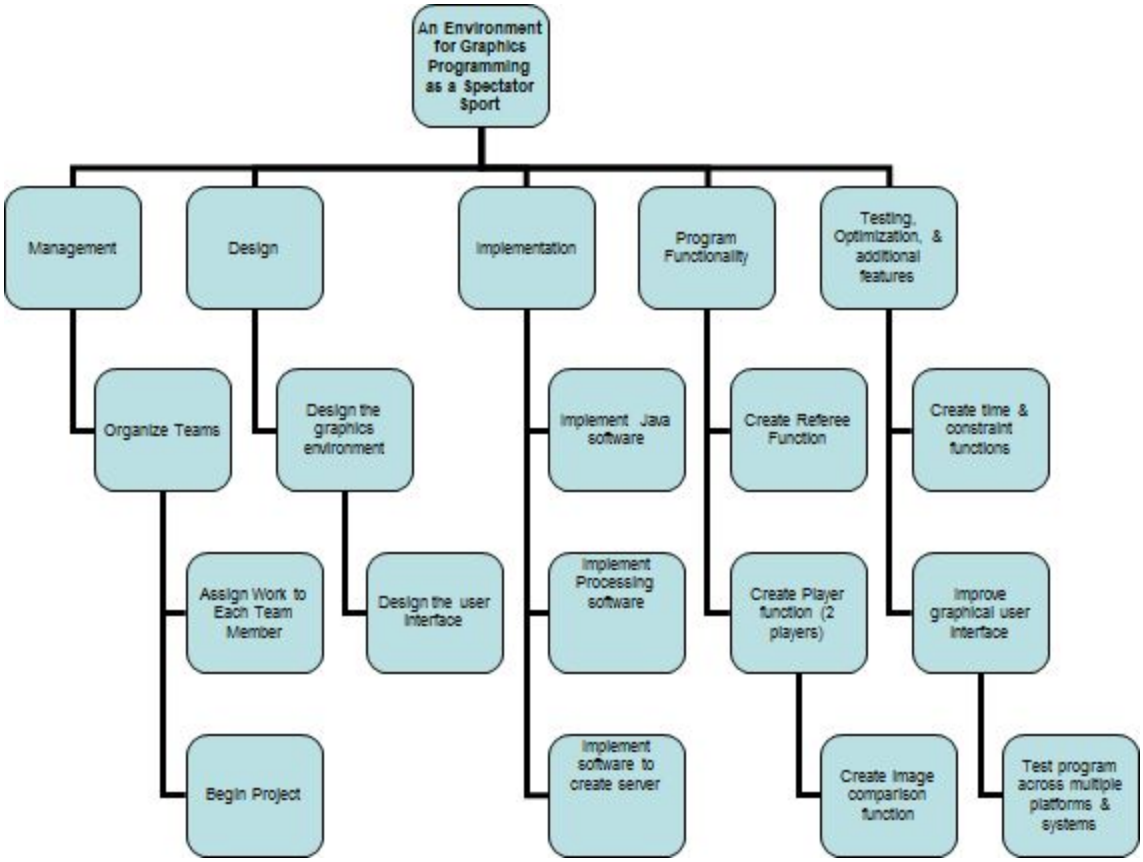
-Revisions to the graphical user interface (GUI)

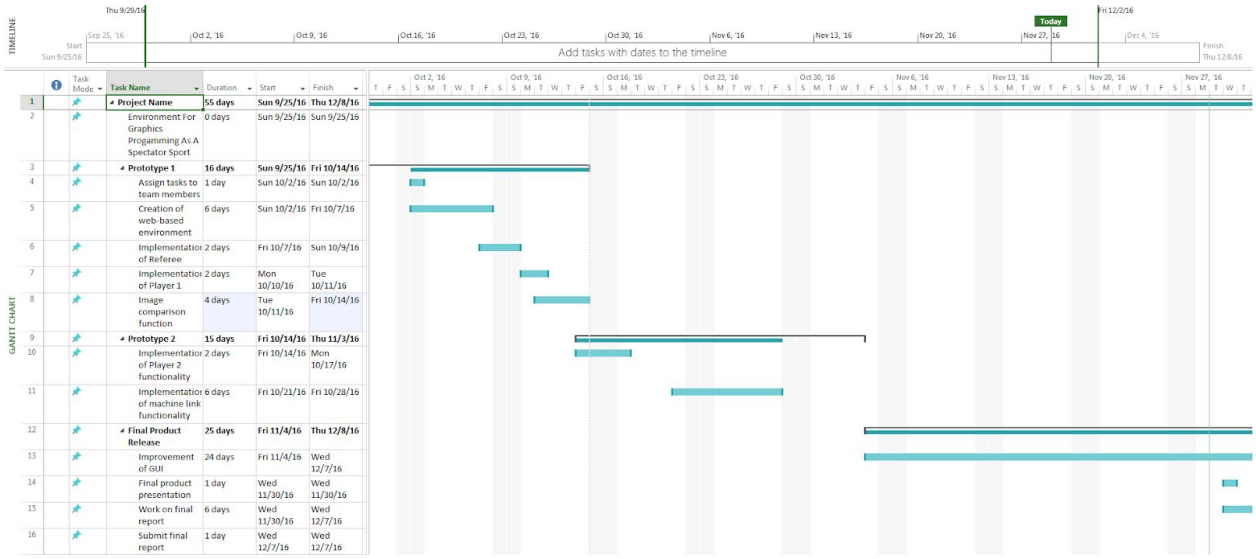
Chapter 2

Task Analysis:

Since the project was such a big undertaking, there was careful and strategic analysis in figuring out what to do first. The first task was exclusive to the project manager, who had to organize the teams, and assign work based on individual strengths. The next task was the design process, which entailed designing the program visually, and designing the look of the graphical user interfaces. The next task was implementing the Java code and Processing languages and making sure the server (which ran on Java) could connect. The next task was the functionality of the program, which included creating the referee and player functions. The final task was testing (especially across different platforms and systems), optimization (making sure the program runs in an efficient manner), and creating any supplemental functions that we feel would benefit the users, such as a constraint feature.

WBS/Gantt:





Risk Management:

The biggest risk that was determined was whether or not each machine could communicate with one another, since this program required 3 computers to exchange data. The player computers needed to communicate with the referee computer and also needed to connect to the referee's server. In order to mitigate this risk, we made it top priority that each machine was able to communicate with one another through rigorous testing.

Chapter 3

Stakeholders:

An Environment for Graphics Programming as a Spectator Sport involves three main types of stakeholders. First is Dr. James Geller, our sponsor from NJIT. The project stems from his idea of an alternative to a hackathon that is more inviting to casual programmers and appreciable by fans of spectator sports in general. As such, a successful project means implementing a solution that satisfies the requirements of our sponsor. Furthermore, Dr. Geller has sponsored past projects for the Capstone Program. It is within our interest to do well in the project to ensure that Dr. Geller remains a sponsor for future projects for the Capstone Program.

As project members, we are also stakeholders. We are the ones implementing the solution and given that hackathons have been popular for quite some time, it is paramount that our project has specific goals in mind and a solution that is simple and intuitive to match our target audience—the beginner and casual fans of programming. Failure to accomplish this could dissuade event organizers interested in hosting a hackathon-like event. Without capturing the interests of the general audience, the project will not be able to achieve its goal of programming as a spectator sport.

Lastly, participants and spectators of the game we have designed are stakeholders. The participants invest their time to take part in the game and if the game is boring, lacking a competitive aspect, or doesn't highlight one's programming skills, for example, then their time is wasted and the game will not be a good alternative to hackathons. Their feedback is essential because they are the target audience who have a unique perspective given that they are likely participants of hackathons. Since the nature of our project heavily depends on captivating viewers of the game, spectators of the game are also stakeholders.

Requirements Gathering:

Through the Scrum methodology, constant communication between the project members and the project sponsor naturally means project members have a good understanding of what the

project sponsor expects. To establish specific goals and requirements, we have asked some of our friends at NJIT and Rutgers who have participated in a hackathon what they would envision programming as a spectator sport to be in the form of short interviews. We asked them whether they would be willing to be both a participant and a spectator of a game involving two players. In addition, electronic surveys were sent out to all students of the Capstone Program to gauge the students' perception of hackathons and its alternative—a computer graphics tournament.

From our first meeting, Dr. Geller shared with us how the spectator game we will be implementing will look like. He described the setup of the environment, which will involve a referee who is in charge of setting the time limit for the duration of the game and will be selecting from a set of images an image that two players will try to replicate using programming code. Whichever player best matches the master image in the given time frame will be the winner. The three people involved in the game will each have a computer—the two players will literally face each other on their computers while the referee's computer will face the audience, simulating a traditional spectator sport. Behind the platform or area where the game is taking place will be three massive screens—two to show the players' screens so that we can see their attempts in coding in real-time and a third screen to show the referee's screen, which will let us see the collection of images that can be "played" with as well as the timer. At the end of the game, the referee's screen will show the results of the players—the percentage of the image that each player was able to match the original image.

While Dr. Geller has specific ideas on what the product should look like, we were given some degree of freedom in how we choose to implement the solution. We took this opportunity to research and determine what technologies to use to ensure we are not limited by them as we progress further into the project. We also needed to take into account certain constraints when we looked into specific technologies, such as the lack of funding. Due to the lack of funding, we decided it is best to build the game as a desktop application as opposed to a web application, which would require financial investment as well as maintenance. Given that the game is a spectator sport and requires three computers regardless, the limitations of a desktop application do not restrict us in any significant manner. In fact, the only requirement that we need is that the three computers are under the same network, which is a non-issue based on our requirements.

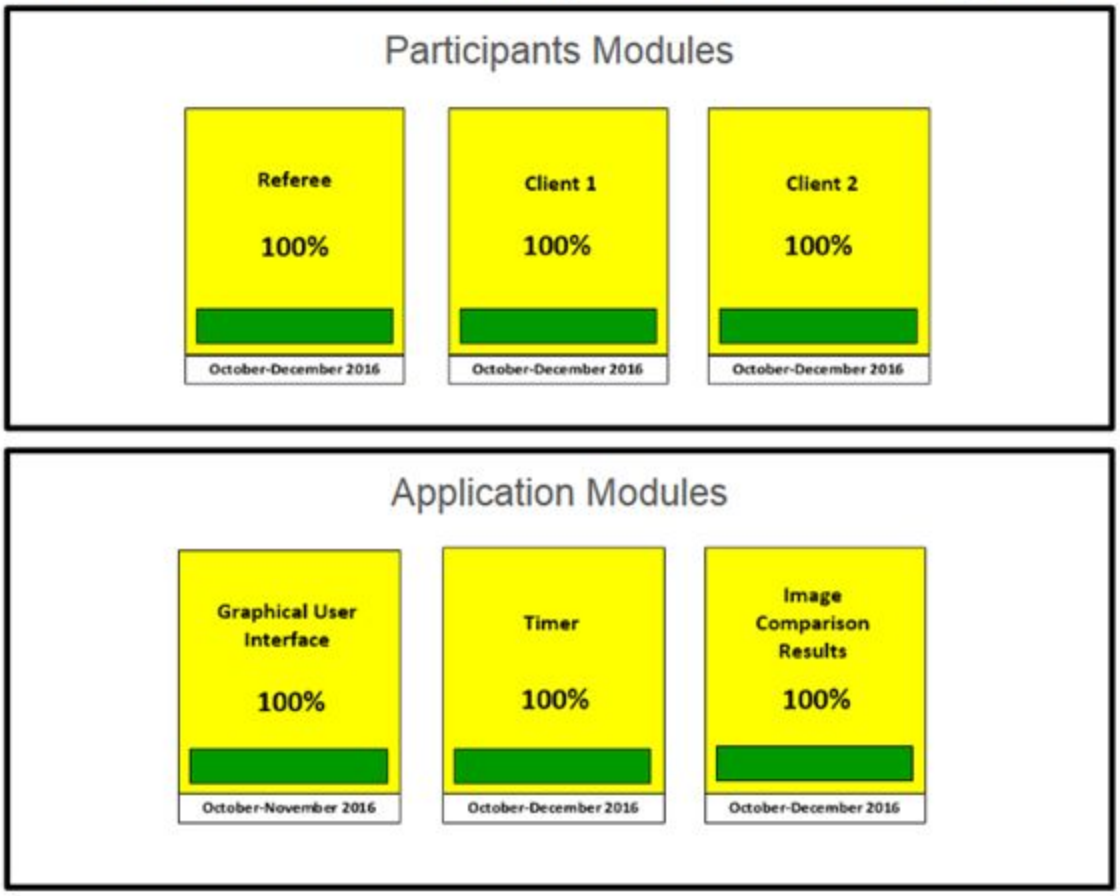
Project Scope:

As mentioned earlier, we decided to build a desktop application due to financial constraints (and to avoid potential security risks as well as constant maintenance). The client applications, which will be the applications that the players will run, will need an area to type their code in as well as an area to draw the image based on their code. The referee application will select the image the players will try to match as well as establish a time limit. A server application will need to be implemented to keep track of the client and referee applications on the network.

Since the game involves programming graphics, we needed to find a way to do image rendering and analysis. Specifically, image rendering involves displaying the image the player is generating using his or her programming code (ideally in real-time), while image analysis involves comparing the images accurately, but more importantly in such a way that best gauges a player's programming skill. Image comparison analysis is a complicated field and given the time constraints and scope of the project, using existing libraries and framework is expected.
We also needed to decide on a language for the players to use. Factors to consider are: simplicity of the language's syntax, the language's familiarity or popularity, and whether how suitable the language is for generating graphics without much abstractions. We chose to use the Processing language because it is a language for learning to code within the context of visual arts and is based on simplified Java syntax, a language that many beginner programmers tend to be exposed to.

To provide a general sense of what our project entails in terms of features that need to be implemented, the following is a functional decomposition diagram reflecting the progress of the features that were implemented by the end of semester:

Functional Decomposition Diagram



Chapter 4

Design:

When we first analyzed the project in detail we determined there would be three major hurdles:

1. Connecting all clients (referee, player one, player two)

2. Finding a language that would facilitate drawing images on the screen using code
3. Finding a method to compare referee and player images and quantify the result

For hurdle 1, we chose to pursue the program as a web application. For hurdle 2, we followed Dr. Geller's advice and selected the Processing language as our bridge between code and image rendering. For hurdle 3, we selected Resemble.js, an existing JavaScript library, for its image comparison functions.

As previously mentioned, our initial approach was to build a web application to take advantage of the internet as a de facto server framework that would allow the referee and player clients to communicate in real time. However, after building a mockup, it was revealed that real-time communication between clients would be impossible due to the restrictions NJIT has in place that disallows the use of websockets. Therefore, our initial plan for client connectivity had to be discarded and it also meant we had to significantly alter our strategy.

After some discussions to select a new strategy, we settled on building the program using Java. Java would allow real-time communication between clients by connecting them through sockets protocols not disallowed by NJIT. Java would also provide us with rich graphics tools to build functional and attractive graphical user interfaces for clients.

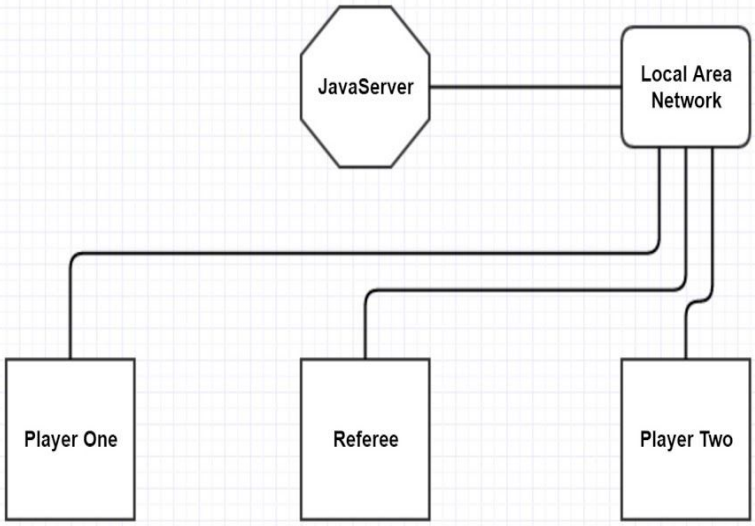
With a new strategy in place, our first course of action was to build the network framework. We built the JavaServer class first, as well as some basic player classes and connected them over a local network. This would be our proof-of-concept for player connectivity and distribution of data in real time.

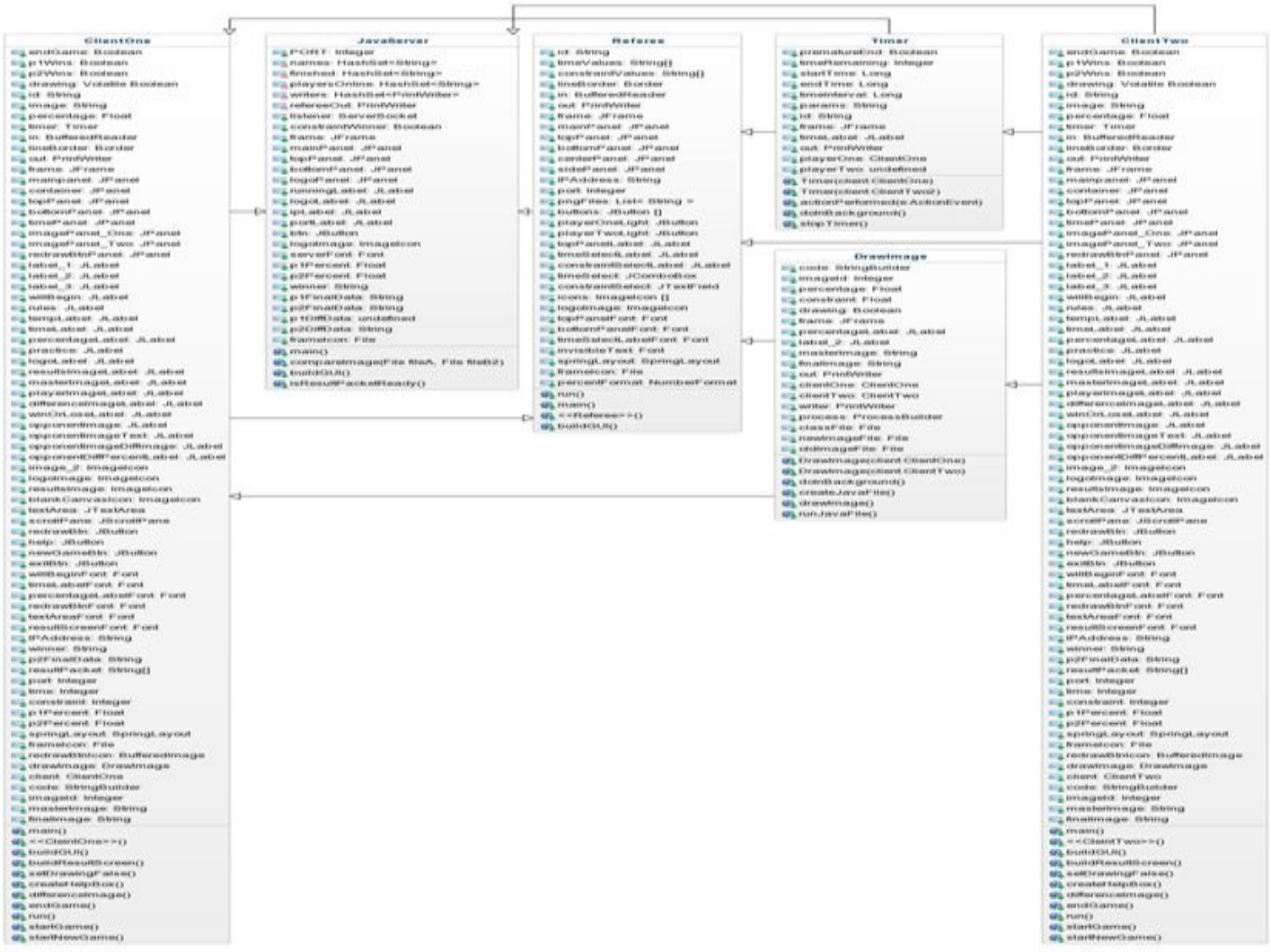
Our next major hurdle was implementing Processing into our program to allow users to draw images. This was accomplished by extracting the user's code, throwing it to an external Java code file, compiling said Java file, saving the output as a PNG image file, and then inserting this new image into the player interface. This meant the user was able to freely code using Processing language syntax and see their result with only a few seconds of delay.

When it came to implementing the system to compare images, we had to discard our initial plan of using Resemble.js, as it proved to be incompatible with Java's JavaScript emulator. We decided to write an algorithm that would compare both images and produce a percent value that represented the similarity between two images. This was accomplished by reading each image into a data array that gives us access to each individual pixel. Then, each pixel in the corresponding location in both images is evaluated, if their RGB value is identical, the pixels are recolored blue. If the pixels are not identical, they are recolored red. At the end of the game, the players are provided with a blue and red image that serves as a visual map of their results.

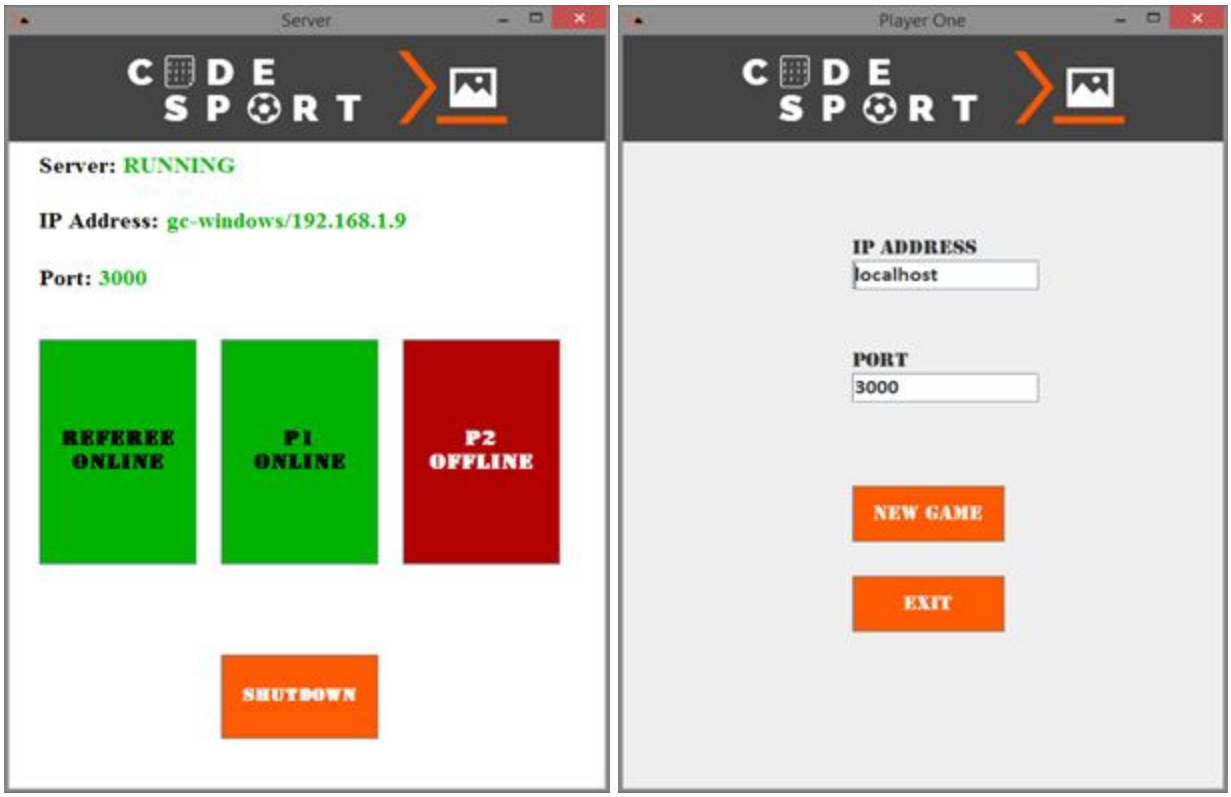
Despite coming across some unexpected obstacles, we were successful in implementing all three systems: connectivity, image rendering, and image comparison. That meant we could provide the client with a fully functional product that met all his major expectations.

NETWORK DIAGRAM




ClientOne
endGame: Boolean
p1Wins: Boolean
p2Wins: Boolean
drawing: Volatile Boolean
id: String
image: String
percentage: Float
timer: Timer
in: BufferedReader
lineBorder: Border
out: PrintWriter
frame: JFrame
mainpanel: JPanel
container: JPanel
topPanel: JPanel
bottomPanel: JPanel
timePanel: JPanel
imagePanel_One: JPanel
imagePanel_Two: JPanel
redrawBtnPanel: JPanel
label_1: JLabel
label_2: JLabel
label_3: JLabel
wlBegin: JLabel
rules: JLabel
tempLabel: JLabel
timLabel: JLabel
percentageLabel: JLabel
practice: JLabel
logoLabel: JLabel
resultsImageLabel: JLabel
masterImageLabel: JLabel
playerImageLabel: JLabel
differenceImageLabel: JLabel
winOrLoseLabel: JLabel
opponentImage: JLabel
opponentImageText: JLabel
opponentImageDiffImage: JLabel
opponentDiffPercentLabel: JLabel
image_2: ImageIcon
logoImage: ImageIcon
resultsImage: ImageIcon
blankCanvasIcon: ImageIcon
testArea: JTextArea
scrollPane: JScrollPane
redrawBtn: JButton
help: JButton
newGameBtn: JButton
exitBtn: JButton
wlBeginFont: Font
timLabelFont: Font
percentageLabelFont: Font
redrawBtnFont: Font
textAreaFont: Font
resultScreenFont: Font
IPAddress: String
winner: String
p2FinalData: String
resultPacket: String[]
port: Integer
time: Integer
constraint: Integer
p1Percent: Float
p2Percent: Float
springLayout: SpringLayout
frameIcon: File
redrawBtnIcon: BufferedImage
drawImage: DrawImage
client: ClientOne
code: StringBuilder
imageId: Integer
masterImage: String
finalImage: String
main()
<<ClientOne>>()
buildGUI()
buildResultScreen()
setDrawingFalse()
createHelpBox()
differenceImage()
endGame()
run()
startGame()
startNewGame()
JavaServer
PORT: Integer
names: HashSet<String>
finished: HashSet<String>
playersOnline: HashSet<String>
writers: HashSet<PrintWriter>
refereeOut: PrintWriter
listener: ServerSocket
constrainWinner: Boolean
frame: JFrame
mainPanel: JPanel
topPanel: JPanel
bottomPanel: JPanel
logoPanel: JPanel
runningLabel: JLabel
logoLabel: JLabel
ipLabel: JLabel
portLabel: JLabel
btn: JButton
logoImage: ImageIcon
serverFont: Font
p1Percent: Float
p2Percent: Float
winner: String
p1FinalData: String
p2FinalData: String
p1DiffData: undefined
p2DiffData: String
frameIcon: File
main()
comparelmage(File fileA, File fileB2)
buildGUI()
isResultPacketReady()
Referee
id: String
timeValues: String[]
constrainValues: String[]
lineBorder: Border
in: BufferedReader
out: PrintWriter
frame: JFrame
mainPanel: JPanel
topPanel: JPanel
bottomPanel: JPanel
centerPanel: JPanel
sidePanel: JPanel
IPAddress: String
port: Integer
pngFiles: List< String >
buttons: JButton []
playerOneLight: JButton
playerTwoLight: JButton
topPanelLabel: JLabel
timeSelectLabel: JLabel
constrainSelectLabel: JLabel
timeSelect: JComboBox
constrainSelect: JTextField
icons: ImageIcon []
logoImage: ImageIcon
topPanelFont: Font
bottomPanelFont: Font
timeSelectLabelFont: Font
invisibleText: Font
springLayout: SpringLayout
frameIcon: File
percentFormat: NumberFormat
run()
main()
<<Referee>>()
buildGUI()
Timer
premadeEnd: Boolean
timeRemaining: Integer
startTime: Long
endTime: Long
timeInterval: Long
params: String
id: String
frame: JFrame
timLabel: JLabel
out: PrintWriter
playerOne: ClientOne
playerTwo: undefined
Timer(client:ClientOne)
Timer(client:ClientTwo2)
actionPerformed(e:ActionEvent)
dotInBackground()
stopTimer()
DrawImage
code: StringBuilder
imageId: Integer
percentage: Float
constraint: Float
drawing: Boolean
frame: JFrame
percentageLabel: JLabel
label_2: JLabel
masterImage: String
finalImage: String
out: PrintWriter
clientOne: ClientOne
clientTwo: ClientTwo
writer: PrintWriter
process: ProcessBuilder
classFile: File
newImageFile: File
oldImageFile: File
DrawImage(client:ClientOne)
DrawImage(client:ClientTwo)
dotInBackground()
createJavaFile()
drawImage()
runJavaFile()
ClientTwo
endGame: Boolean
p1Wins: Boolean
p2Wins: Boolean
drawing: Volatile Boolean
id: String
image: String
percentage: Float
timer: Timer
in: BufferedReader
lineBorder: Border
out: PrintWriter
frame: JFrame
mainpanel: JPanel
container: JPanel
topPanel: JPanel
bottomPanel: JPanel
timePanel: JPanel
imagePanel_One: JPanel
imagePanel_Two: JPanel
redrawBtnPanel: JPanel
label_1: JLabel
label_2: JLabel
label_3: JLabel
wlBegin: JLabel
rules: JLabel
tempLabel: JLabel
timLabel: JLabel
percentageLabel: JLabel
practice: JLabel
logoLabel: JLabel
resultsImageLabel: JLabel
masterImageLabel: JLabel
playerImageLabel: JLabel
differenceImageLabel: JLabel
winOrLoseLabel: JLabel
opponentImage: JLabel
opponentImageText: JLabel
opponentImageDiffImage: JLabel
opponentDiffPercentLabel: JLabel
image_2: ImageIcon
logoImage: ImageIcon
resultsImage: ImageIcon
blankCanvasIcon: ImageIcon
testArea: JTextArea
scrollPane: JScrollPane
redrawBtn: JButton
help: JButton
newGameBtn: JButton
exitBtn: JButton
wlBeginFont: Font
timLabelFont: Font
percentageLabelFont: Font
redrawBtnFont: Font
textAreaFont: Font
resultScreenFont: Font
IPAddress: String
winner: String
p2FinalData: String
resultPacket: String[]
port: Integer
time: Integer
constraint: Integer
p1Percent: Float
p2Percent: Float
springLayout: SpringLayout
frameIcon: File
redrawBtnIcon: BufferedImage
drawImage: DrawImage
client: ClientTwo
code: StringBuilder
imageId: Integer
masterImage: String
finalImage: String
main()
<<ClientTwo>>()
buildGUI()
buildResultScreen()
setDrawingFalse()
createHelpBox()
differenceImage()
endGame()
run()
startGame()
startNewGame()

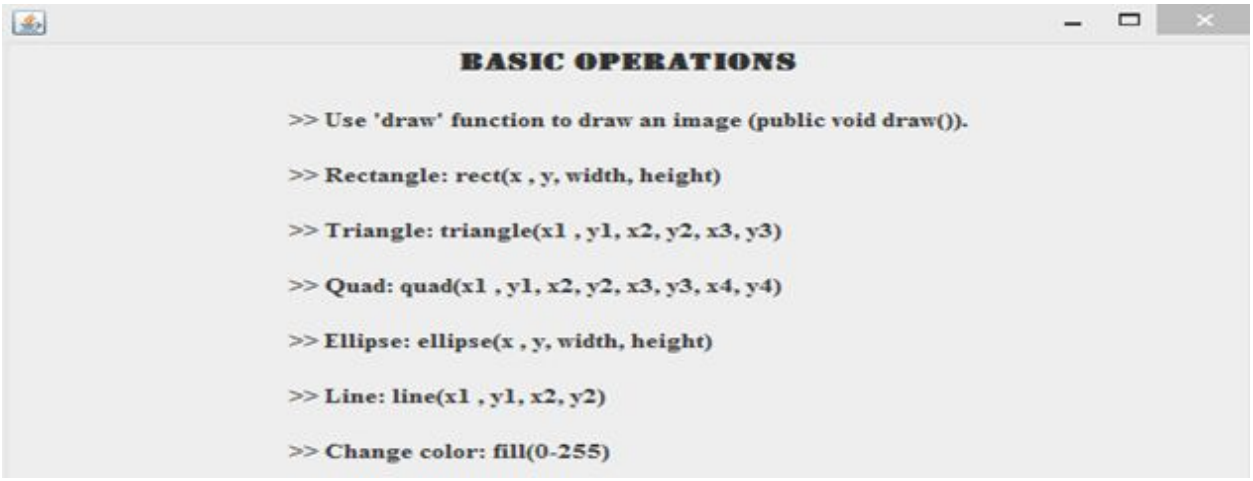
Chapter 5

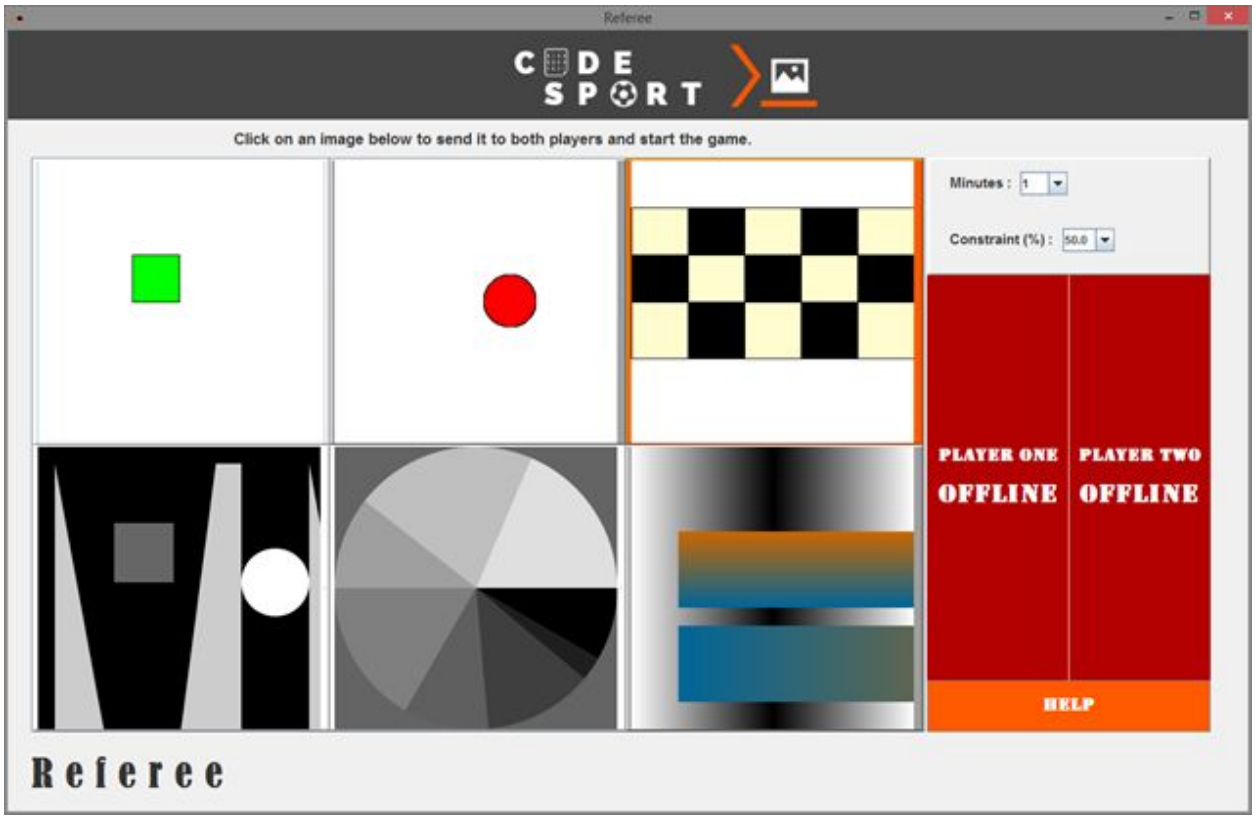


To the left is an image of the server application's GUI. It provides a visual on whether the players and the referee are connected so that communication can be established. Additional details such as the IP address and port number are displayed for players and the referee to specify in order to enable a connection.

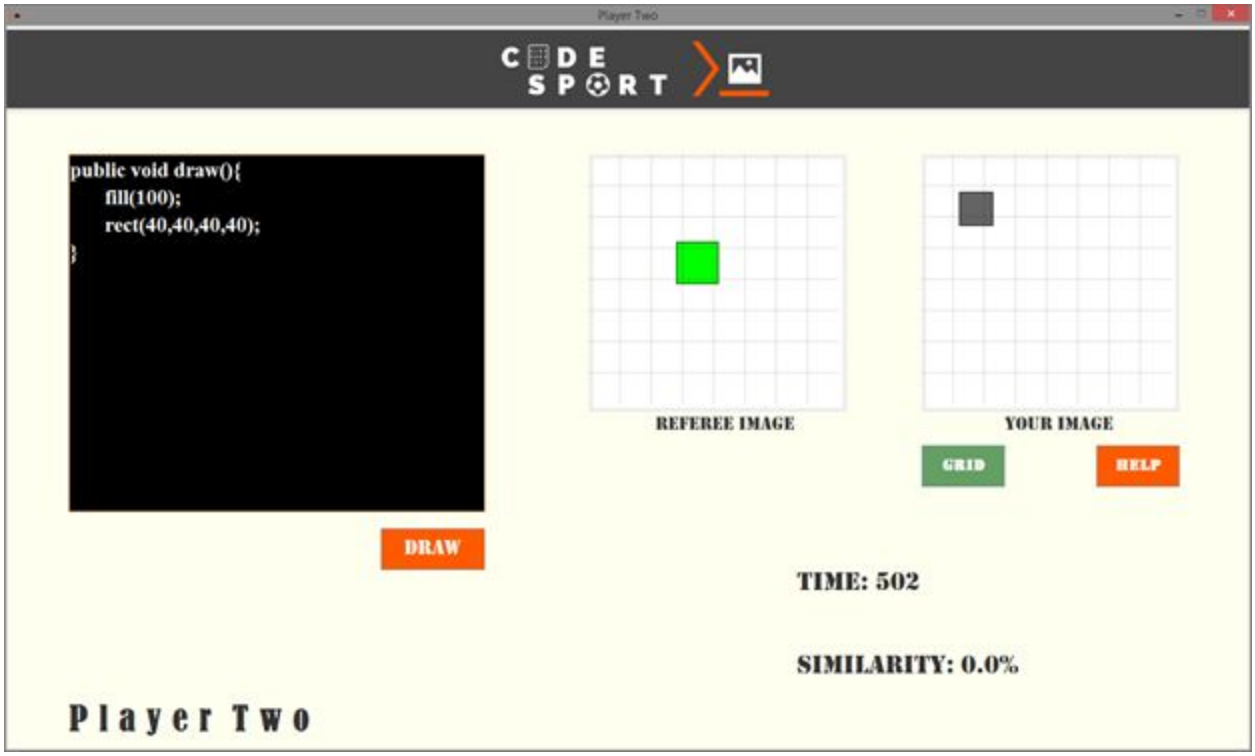
To the right is an image of a GUI that a player would see. The player has the option to specify the IP address or port number he or she wishes to connect to as well as the option to start the game when he or she is ready.

Below is an image of a brief help menu that players can access when they play the game to get them started.





Above is an image of a referee's GUI, where the referee can also see whether the two players are ready to play the game. He or she selects an image for the players to try to reproduce using code. A time limit can be specified as well as the percentage of similarity that needs to be achieved for a player to win before the time is up.



Above is an image of a player's GUI. To the left is the area where the player types the programming code. The middle area displays the master image that the referee has selected for the players to attempt to match. The right area shows the image rendered by the player's code when he clicks "DRAW" during the game. which will process the code. The time and similarity of the image is shown for the players to keep track of their progress. An option to enable or disable grid is available to aid in matching the image as close as possible. The help button will display the help menu.

User Manual

Player:

1. When the Referee has launched the server and the Referee client, execute the PlayerOne.jar

or PlayerTwo.jar file depending on whether you are player one or player two, as designated

by the referee.

2. Specify the IP address and the port number that the Referee used in order to establish a connection. When you are ready, click "NEW GAME".

3. The game will start when both players are ready. You will see the image the referee has selected. In the area to the left, type your code to try to match this image. Click "DRAW" at

any time throughout the game to render your image based on the code—it will be shown in the area to the right.

Tips:

- You may practice your coding skills any time before the referee starts the game.
- A help menu is provided to help you get started on the Processing language—the language you will use to attempt to reproduce the image. Click “HELP”.
- Click “GRID” to enable or disable the grid overlay, which may aid in your effort to match the master image.
- Remember to keep track of your time and progress!

Referee:

1. Launch JavaServer.jar followed by Referee.jar, changing the IP address and port number if necessary. Assign a player as player one and another as player two so that they may launch their respective executable files.

2. On the Referee client, a visual indicator shows whether the two players are connected. When they are, specify the constraint percentage and the time limit for the duration of the game.

3. Select an image to begin the game.

Chapter 6

The problem being solved in the case of the project was to design a program that would make software coding more interesting and accessible to individuals who may not have known about it previously, or had the time and patience to participate in Hackathons. Our sponsor, Dr. Geller, had the idea of developing a game that would allow these individuals to easily become more involved in a fun and exciting way. A development specification was presented to us, and we proceeded with this as a template for research and development of the product.

In the first stage of research into the project, a proper graphics programming environment was to be selected. There are several options available, but the Processing programming language seemed to be the best fit for the requirements set forth by the specification. There are several versions of the Processing programming language available for use since it is an open-source graphics programming environment. Our research led us to Processing.js. This is a version that is specifically designed for use with web browsers.

A significant problem we faced when developing the solution was that it needed to be compatible with NJIT’s online file system, AFS. The problem is that in order to have direct communication between clients, we would need to utilize web sockets, which are limited for use by students at the University for security purposes. Our solution to this problem was to use a

database to store the master and player images and allow the server to access them in order to run the code used for image analysis and comparison. This implementation was a combination of both desktop and web-based technologies. In testing this version of the program we found it to be a very cumbersome and convoluted approach. The software was using a Java Server to control the communication, but there was little direct communication between the clients. Therefore, options for further development features were extremely limited.

Another significant factor in the development of this version of the software involved the use of an open-source JavaScript library called Resemble.js. An intuitive way to find the difference between two images was to be developed. During our early research stages, we discovered this web-based solution and attempted to incorporate it into our program. In order to accomplish this, we needed to use the images stored in our database. Overall, although this solution was able to handle most of the requirements, it may not have been the most efficient approach. Further development would have required resources that were out of the scope of this project.

The next solution involved the implementation of an entirely desktop-based version of the program, which was accomplished by modifying the first version. Similar to the previous version, this one used the Processing language library, a Java Server, and Java Sockets for communication. However, the web-based and database elements were discarded. Since the communication is being done over the same network, we found there to be little use for a database, as any files generated by the players could just as easily be stored locally as files on the clients' computers. The Image analysis and comparison feature was also included in the code, as opposed to using the Resemble.js JavaScript library.

The first step in Unit testing this solution involved making sure that the communication between the players and referee classes was optimal. Each class of the program was treated as a stand-alone component and tested individually for errors. The server class was implemented first, along with the referee and client classes, and tested to ensure packets were being sent and received between the player and referee classes. Next, a timer class was designed to set a specific time limit to the game. In the early stages of development, a time limit was hard coded into the software and later revised to include a selection box to allow the referee to choose the time limit. Another significant modification to this version of the program was to allow for the player to modify their Processing code several times, and for this to be displayed on the screen. In order for this to be possible, the drawimage class was included. To test this class, we needed to make sure that the file generated by the players code was accurate, since it must be updated each time the player attempts to re-draw his or her image. After the player's code is compiled, the class helps to generate the image that is displayed.

Another important feature included in this version was the ability for the referee to have a visual representation of the players, as this allows for a more seamless interaction between clients and referee. In testing this module, we discovered the optimal way was to use Java input/output streams to indicate whether each player was ready for file transfer over the network. Finally, the last stage of unit testing involved the user interface elements of the software. Each module was tested individually first. As each additional module was developed, it was ensured that each one was free of errors and tested several times.

Once the system was developed, and with the completion of module testing, the next phase was program testing. Since this program involves the use of networking technologies, we were tasked with testing the program under several circumstances. First and foremost, the

software needed to run on the NJIT wireless network, and all modules must be on the same network to function appropriately. Although the program can also be modified to work on a local area network, we found the most convenient approach to be one that involves a wireless network and the use of laptops. The individual computers used by the server, referee, and client classes, must all be able to compile Java code, as well as being able to execute it. Testing this involved ensuring that the individual computers had the proper elements installed on them. For example, we needed to make sure all of the computers used to run the system had the same version of Java, as well as the components necessary to compile programs written using Java.

We utilized both top-down and bottom-up testing strategies for the program testing phase. With top-down testing, we tested the communication of the main module to each individual client class which allowed us to identify errors and fix them. The top-down approach was important in identifying any issues with the server, timer, and drawimage classes. Next, the bottom-up program testing strategy was used to identify errors with the clients.

In the System testing phase of development, we needed to make sure that the software was able to compile and run on different systems. For this stage, we decided to make each module into an executable .jar file that can be run on different systems. Once again, some requirements were necessary, such as the ability of the system to run and compile Java files, as well as the need for all machines to be connected to the same network. However, more testing is required, as this phase is primarily done by individuals outside of the development team. Errors may arise from further testing, however, at the current stage, the system is stable.

During the verification stage of software testing, we must ensure that the system is being developed according to the requirements. We have successfully designed a distributed three computer system that is able to handle communication, file transfer, and image analysis, as specified by our stakeholder. Throughout the development process, we have met with our sponsor for the project, Dr. Geller, and he has informed and guided us in the different options we have in development, and how to solve issues. Each component of the program was thoroughly discussed with the sponsor and each team member before the final product was submitted. The system that has been developed could be modified further to include several elements. The specification document mentions the inclusion of a point system and the possibility of a web-based version of the game, which may be developed in the future.

In the validation stage, we must rely heavily on testing. Our program is designed to be used by individuals who are not familiar with graphics programming, and possibly without any previous knowledge of coding at all. However, we have not been able to distribute it to the public, as this is beyond the current requirements. This would allow large groups of people to test and evaluate the product. The next step is to allow the finished program to be played by the people it was intended for. Since it is designed to be a spectator sport, an audience whom would be able to observe the game being played may also be required for testing. Through their analysis and feedback, we would be able to further alter the product.

Conclusion:

This project has shown us the importance of reaching out to the general public in regards to computer science and coding. When it comes to Hackathons, many efforts have been made to increase participation and interest of minorities and individuals who might not have time, but this has garnered limited success. Our project takes a different approach by making the idea of participating fun as well as competitive. The idea of a game suits this need extremely well and the product that has been developed may be the next step to getting more people interested in the field of computer science.

The game can be looked at as a stepping stone into the world of coding. In order to familiarize someone with programming quickly, there needs to be a visual representation of the code at work and how it changes depending on the user's source code. The game we have developed does this through the use of graphics programming. Many creative minds would have a chance to express themselves using the product, as it is a combination of art and computer science. Furthermore, instead of Hackathons, a game that could be played by anyone in their spare time, as well as in a competitive setting, could ultimately be the key to drawing more interest in this field.

Throughout the semester, we have had the opportunity to work with our Sponsor, Dr. Geller, on a real-world example of Client-Server architecture using Java. In addition, we have learned to use the Processing programming language and have learned how to use it's libraries in a desktop-based application. The implementation of file input/output streams, image analysis and comparison, and Graphical User Interface design also played significant roles in the research and development, and we have gained knowledge in each of these areas.

We have faced several obstacles throughout the development process and devised ingenuitive ways to overcome them. For example, one of the major factors was switching from a partially web-based application to a completely desktop-based version. This allowed us to include additional features that could not have been possible using a web-based version. We have also gained significant knowledge into how to properly manage the development of the project. The work was distributed to the various team members based on their strengths and we have used visual representations of the management in the form of Work Breakdown Structure and Gantt charts.

If given the opportunity to think about the project in a different way and without the limitations imposed upon us by AFS, we may have decided to develop an entirely web-based application instead. We have also discussed the possibility of making the game into a player versus player application, without the need of a referee. This would let us have an online game that would be very easily accessible by a more wide audience of people.

We believe that the final product sufficiently performs the tasks as described under the requirements specification. We have successfully developed an environment for graphics programing as a spectator sport. With Dr. Geller's support and guidance of the next team which may continue the project, we look forward to seeing it further evolve into his vision.