**Interactive Checkers Solver Application for Smart Phones**

**Submitted To**

**Emily Richardson**

**Yeojoon Kim**

**Dr. Michael Becker**

**Dr. Brian Evans**

**Rick Maule**

**Qualcomm**

**Prepared By**

**Farhad Abasov**

**Arthur Ishiguro**

**James Lee**

**Kevin Scholz**

**Alexander Yeh**

**EE464 Senior Design Project**

**Electrical and Computer Engineering Department**

**University of Texas at Austin**

**Fall 2011**

**CONTENTS**

**TABLES iv**

**FIGURES v**

**EXECUTIVE SUMMARY vi**

**1.0 INTRODUCTION 1**

**2.0 DESIGN PROBLEM STATEMENT 2**

**2.1 PROBLEM STATEMENT 2**

**2.2 SPECIFICATIONS AND CONSTRAINTS 2**

**2.3 DESIGN PARAMETERS 3**

**3.0 DESIGN PROBLEM SOLUTION 4**

**3.1 DESIGN SOLUTION 4**

**3.2 PRODUCT DESCRIPTION 5**

**3.3 SYSTEM OVERVIEW 6**

**3.3.1 Tracking 7**

**3.3.2 Game State Analysis 8**

**3.3.3 Checkers AI Algorithm 8**

**3.3.4 Output Rendering 9**

**3.3.5 User Interface 11**

**4.0 DESIGN IMPLEMENTATION 12**

**4.1 SUBSYSTEM IMPLEMENTATION 12**

**4.1.1 Checkerboard Tracking 13**

**4.1.2 Game State Analysis 13**

**4.1.3 Checkers AI 14**

**4.1.4 Output Rendering 16**

**4.1.5 User Interface 17**

**4.2 SYSTEM INTEGRATION 18**

**4.3 CHALLENGES 18**

**CONTENTS (Continued)**

**4.3.1 Checkerboard Detection 18**

**4.3.2 Light Reflection on Checkers Pieces 19**

**4.3.3 Omitted Checkers Path in Checkers AI Checkers AI 19**

**5.0 TEST AND EVALUATION 20**

**5.1 ACCURACY OF CHECKERBOARD TRACKING 20**

**5.2 SPEED AND ACCURACY OF GAME STATE ANALYSIS 21**

**5.2.1 Timing Analysis of Game State Analysis 21**

**5.2.2 Accuracy Analysis of Game State Analysis 22**

**5.3 FUNCTIONALITY OF CHECKERS AI ALGORITHM 23**

**5.3.1 Move Suggestion Quality 23**

**5.3.2 Input Handling 24**

**6.0 TIME AND COST CONSIDERATIONS 24**

**7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN 25**

**8.0 RECOMMENDATIONS 25**

**5.1 INTERACTIVE CHESS SOLVER 25**

**5.2 DEVELOPMENT OF AR APPLICATIONS 26**

**9.0 CONCLUSIONS 26**

**REFERENCES 27**

**APPENDIX A – TRACKING TESTING RESULTS A-1**

**APPENDIX B – BILL OF MATERIALS B-1**

**TABLES**

1 Accuracy of Game State Analysis in Various Conditions 23

A1 Continuous Tracking in an Indoors Environment A-1

A2 Continuous Tracking in an Outdoors Environment A-1

B1 Bill of Materials for the Project B-1

**FIGURES**

1 Operations of the Interactive Checkers Solver Application 5

2 System Level Block Diagram of the Interactive Checkers Solver 6

3 Input and Output of Game State Analysis 8

4 Inputs and Outputs of Output Rendering 10

5 Mini-max Tree 15

6 Tracking Rating System Results of the Unique Checkerboard 19

7 Time Required to Run the Game State Analysis Algorithm 22

**EXECUTIVE SUMMARY**

Augmented reality (AR) is an emerging technique used to overlay virtual graphics over an image of the physical world. Recent advances in smart phones have flourished opportunities for the development of AR smart phone applications. The Interactive Checkers Solver application development project discussed in this report demonstrates AR as an effective way for checkers players to learn game playing strategies.

This report documents various aspects of the development of the Interactive Checkers Solver application. The report first discusses the application as a solution towards the general need for checkers players to find intuitive and convenient ways to train themselves. The report also discusses the key operations of the five software subsystems of our design: tracking, game state analysis, checkers artificial intelligence (AI), output rendering, and the user interface. When the application loads, the user must first point the mobile phone’s camera at the checkerboard of an ongoing checkers game. The tracking subsystem will identify and track the checkerboard when the board is visible through the camera. The game state analysis subsystem will then sample color information of the display to determine the colors of existing checkers pieces and populates the current checkers game state. Subsequently, the checkers AI algorithm uses the determined game state to calculate the next effective move. The calculated move is then illustrated by the output rendering subsystem through 3D shapes such as arrows and circles. All settings and message displays are controlled through the user interface subsystem. This report also discusses the software implementation details of each subsystem, as well as challenges faced when designing a unique checkerboard and pieces. We also assess the application operations by analyzing test results. Such tests include the tolerable ranges of the tracking system, timing and accuracy of the game state analysis routine, and the functionality of the checkers AI algorithm through various inputs.

The development of the Interactive Checkers Solver application successfully completed. The application posed no major time, cost, environmental, and safety concerns during the development process. However, the application proved to be limited, requiring the creation of custom checkerboard and pieces suited for our design. In this report, we discuss the project as a platform for future AR application development projects, particularly an interactive chess solver. Overall, the Interactive Checkers Solver project demonstrated the use of AR graphics in mobile smart phones as a new method for checkers game training.

**1.0 INTRODUCTION**

This report provides a comprehensive documentation of the development of an augmented reality (AR) Interactive Checkers Solver application for mobile smart phones. Augmented reality is a technique used to overlay digital graphics over an image of the physical world. Our Interactive Checkers Solver identifies an ongoing checkers game through a smart phone’s camera and suggests effective moves to application users by illustrating 3D graphics over the captured checkerboard image. Farhad Abasov, Arthur Ishiguro, James Lee, Kevin Scholz, and Alexander Yeh undertook the project under supervision of Professor Brian Evans and Rick Maule from Qualcomm. The development of AR smart phone applications opened up new ways to utilize virtual graphics for entertainment systems. Similarly, our project expands the playing and training style of a checkers game. By using our application, checkers players can educate themselves in an interactive and entertaining way.

This report is organized in the following manner. The Design Problem Statement section introduces the design problem and delineates the requirements and design parameters of the Interactive Checkers Solver. The Design Problem Solution section explains the basic operation of the application, and its development and running environments. This section also illustrates key procedures of each software subsystem of our application: tracking, game state analysis, checkers artificial intelligence (AI), output rendering, and a user interface (UI). Subsequently, the Design Implementation section details the software implementation of each subsystem. The section also describes engineering challenges in our design, such as the checkers piece and board design and the calculation of the path to move. The Testing and Evaluation section then analyzes the results of the accuracy, timing, and functionality tests for our application’s operations. The subsequent section outlines the overall scheduling strategy and the financial cost of our design development. We also describe safety and environmental issues regarding our application design in the following section. Finally, our report will conclude with recommendation for future development of related AR smart phone applications.

**2.0 DESIGN PROBLEM STATEMENT**

The Interactive Checkers Solver application aims to implement a new way for checkers players to train themselves. This section outlines the problem for players to find effective and convenient training methods on a regular basis. The section will also provide design specifications and parameters that were considered to develop the application.

**2.1 PROBLEM STATEMENT**

Checkers players who wish to improve their proficiency need to play with skilled opponents to learn different strategies of the game and improve their skills. However, finding such an opponent regularly is difficult. The goal of our project is to provide users an intuitive way to learn effective strategies in the game of checkers through mobile smartphones. As a solution, our AR Interactive Checkers Solver application allows players to practice with an advanced computer AI on a real checkers board. To solve this problem, we have divided the solution to be composed of five different subsystems, which we will describe in the Design Solution section.

**2.2 SPECIFICATIONS AND CONSTRAINTS**

The application will have to meet certain specifications in order to provide the necessary functionality to the user. The AR checkers solver application will be restricted in its use of memory, which is a scarce resource on a mobile phone. The Qualcomm AR software development kit (SDK), which our application uses for tracking the checkerboard, supports all Android devices of version 2.1 and higher. One of the earliest released Android devices, the Motorola Droid, contained 256MB of random access memory (RAM) [1]. The application should use no more than 25% of the system RAM resources, or 64MB. This limit will ensure that the application runs smoothly while leaving enough memory for background applications.

**2.3 DESIGN PARAMETERS**

Various parameters were taken into consideration for this project. In our application, we defined the design parameters with respect to speed and accuracy, functionality, and flexibility of the user interface.

First, our application must produce results in an accurate and fast manner. In our application, we need the tracking of the checkerboard to be robust through various camera distances and angles. In addition, the analysis of the checkers game state must also be accurate and fast. Specifically, we require that the checkers game is correctly identified at least six out of ten trials. The identification must also take no more than five seconds. To achieve these goals, we use optimization techniques, which will be discussed in the Design Implementation section.

We also require a functional checkers solving algorithm to work within our application. The algorithm must provide effective moves for any given state that leads the user to a winning game. We require that the user can win against an average checkers player using the algorithm. While the user is running the application, we must also ensure that the algorithm does not halt or crash during its operation. In our application, we require that the algorithm can continuously run for one day without crashing.

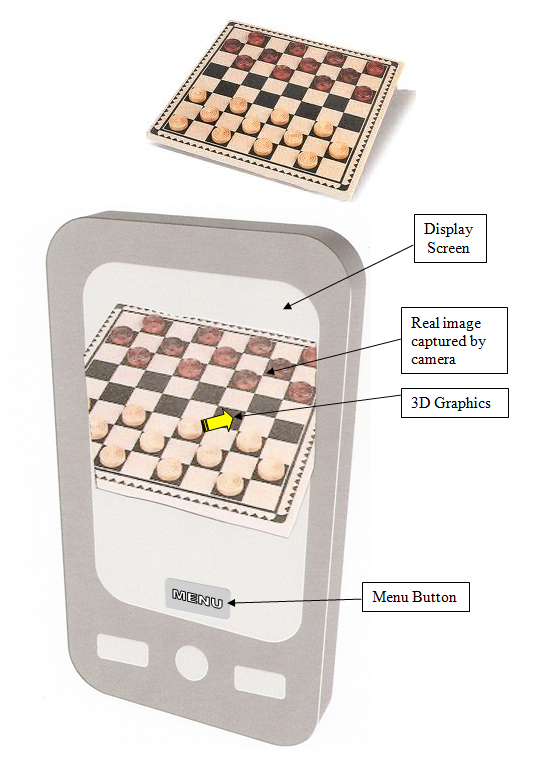
Finally, the application must also have a user interface with flexible features. For instance, the application needs to provide difficulty settings that users can modify. The difficulty settings will correspond to the amount of time the algorithm will require when calculating the next move. The more time allotted, the better the move suggestion will be. The user should be able to specify from a range of one to five seconds in more granular increments. The application must also have a user interface which will allow the user to specify whose turn it is. This will start the process of computing the next suggested move to display on the screen. While this is an important feature of the user interface to provide, we will be minimizing the set of features to the users. As mentioned in the beginning, our audience is checker players who want to improve their skills. We provide only the set of features that will be vital to that audience, and eliminate other unnecessary interfaces that will only make the application look unattractive and distracting. Our goal is to make the user understand how to use the application instantly.

**3.0 DESIGN PROBLEM SOLUTION**

As a solution to the need for checkers players to find ways to train themselves, our application uses the AR technology to display graphics to illustrate suggested moves. This section will first discuss how our application provides an intuitive training environment for a checkers player. In the subsequent product description section, we will describe the software and hardware environment of our application. We will then outline the description of the application system and its five software subsystems: tracking, game state analysis, checkers artificial intelligence (AI) algorithm, output rendering, and the user interface (UI).

**3.1 DESIGN SOLUTION**

Our project is an Android smart phone application that employs the augmented reality (AR) technology to interactively solve and illustrate advantageous moves to checkers players. AR is a technique used to overlay digital information over a visual interpretation of the physical world. In our project, the application renders virtual 3D arrows and checkers pieces to illustrate suggested playing moves. Figure 1 illustrates how our application will employ AR graphics to illustrate the playing moves. These AR graphics are displayed over the video image in real-time, as if they exist on the actual checkerboard. Our application will first recognize and identify an ongoing checkers game through a real-time video feed from a smartphone’s camera. Subsequently, a checkers solver algorithm calculates a move for the identified game state. The application will then illustrate the calculated move by rendering 3D shapes, such as arrows and circles, to represent the calculated move.



**Figure 1. Operations of the Interactive Checkers Solver Application**

**3.2 PRODUCT DESCRIPTION**

The interactive checkers solver will run as an Android smartphone application program. This section will provide the description of the application as a software product. We will describe specifics about the software development, including the chosen software languages and development environment. We will also describe the running environment of the application, including the required use of a custom checkerboard and pieces.

Our application was written in the C++ and Java software languages, and developed using the Eclipse software development environment. The C++ code describes the native operations of the application, including tracking, image analysis, checkers solver routine, and rendering graphics. The Java code interfaces the user interface and native operations with the Android software platform. Qualcomm also provides a SDK, which contains the architecture to implement AR techniques such as real-time tracking. Our application will be compiled using both the AR SDK, as well as Android’s native development kit (NDK).

Our application is designed to run on any Android smart phone with operating system 2.1 or higher, with an ARMv6 floating point unit (FPU) processor. For reasons discussed later in this document, the application will also be run using a custom checkerboard and pieces. The checkers pieces are red and black in colors for each player, and are green and blue on the opposite side. The green and blue colors indicate the crowned pieces of the red and black players, respectively.

**3.3 SYSTEM OVERVIEW**

The Interactive Checkers Solvers will run on an Android smart phone as a software application. To use the application, the user must first point the phone’s camera at the checkerboard of an ongoing checkers game. Once the checkerboard is visible, the application will begin its operations. Figure 2 illustrates the overall flow of the application’s procedures. The application will first analyze the captured camera image and determine the state of the game by detecting the checkerboard and the pieces. After identifying the game state, the application will run a checkers AI algorithm to calculate the next suggested move. Subsequently the output rendering subsystem will render 3D images illustrating the calculated move on the phone’s display. The application’s UI will allow users to specify application settings, and will also display text messages on the screen. The UI subsystem will also communicate data with the other subsystems as the application runs. At the system level, the UI subsystem will provide settings that users can modify and display messages to the display. This section will subsequently outline key operations of each of the described software subsystems.

**Settings**

**User**

**Interface (UI)**

**Camera**

**Tracking**

**Game State Analysis**

**Checkers AI**

**Output Rendering**

**APPLICATION**

**Display**

**INPUT**

**OUTPUT**

**Figure 2. System Level Block Diagram of the Interactive Checkers Solver**

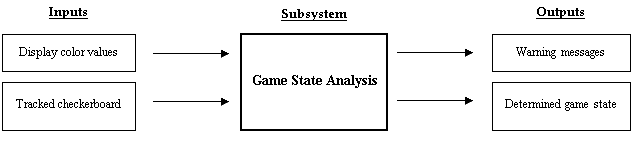
**3.3.1 Tracking**

The primary function of the tracking subsystem is to identify and track the checkerboard when visible through the camera and to notify the user when the checkerboard cannot be found. In addition, if the entire checkerboard is not visible or some checkers squares are not visible to the camera, then that particular square on the checkerboard will be highlighted to indicate that it cannot detect a majority of the area of that square. These notifications will allow users to make certain adjustment to ensure that the application will work properly. The input to the tracking subsystem is a live video feed from the mobile device’s camera. Using the video feed, the system will notify the game state analysis subsystem of the presence of a checkers game. This is accomplished by outputting the coordinates of the tracked checkerboard to the game state analysis. To simplify the process, our tracking subsystem will utilize Qualcomm’s SDK to track the checkerboard.

Qualcomm’s SDK provides android developers with a set of tools to easily track unique objects in their application. One such tool is the tracking rating system available on Qualcomm’s Development Network. By uploading image targets to the Qualcomm Development Network, the users can easily identify which images contain the highest probability of being detected by the SDK’s tracking algorithm. In addition, users can also download target configuration files from the Qualcomm Development Network and add them to their application. The SDK then adds the appropriate image details to a database for comparison at run-time. Moreover, the SDK provides many tracking functions such as identifying the quantity of currently tracked images. These functions will allow developers to concentrate on the unique features of their application.

**3.3.2 Game State Analysis**

Once the checkerboard is tracked, the game state analysis subsystem will determine the checkers game state by identifying the user’s and opponent’s pieces. Figure 3 shows an input and output diagram for this subsystem.



**Figure 3. Input and Output of Game State Analysis**

Using the tracked checkerboard, the subsystem will extract information about the checkerboard including its center, width, and length. The subsystem will then perform position arithmetic to find the location of the squares, and will sample the display’s color values associated with the square. In our application, the red and black checkers pieces will represent the user’s and opponent’s pieces. Pieces will also be crowned once they reach the opponent’s side, and will be represented by green and blue pieces. Using the sampled color information, our software will determine the game state by identifying existing pieces and their colors. If the calculation is unsuccessful, the subsystem will exit after displaying a warning message on the screen. If the routine is successful, the subsystem will store the game state in a data structure as an output.

**3.3.3 Checkers AI Algorithm**

The checkers AI is responsible for calculating the next suggested move. After the checkerboard has been detected and the game state has been computed, the game state analysis component will pass the game state to the checkers AI algorithm. The algorithm will then compute a suggested move to make. After computing the next move, this subsystem will convert the results into a suitable data structure to be used by the subsequent subsystem to display the move on the screen. In order to find such a move, our application will use an open source checkers AI algorithm which utilizes the mini-max search algorithm written in the C programming language.

The checkers AI algorithm allows us to calculate the next suggested move for both players. Before the algorithm calculates the next move, our application allows the user to choose which color he or she is playing as. Because the player may not always be playing on the same side of the board, this feature adds flexibility to our application. The checkers AI algorithm also allows the strength of the AI to be changed by adjusting the amount of time allotted for it to calculate the next move. The more time given to the algorithm, the stronger the move it suggests. Beginner players can decrease the time for an easier game, and more advanced players can set the allotted time to a higher setting to give them a challenge.

There are various approaches in finding an advantageous next move in a game of checkers. The checkers solver application will use an open source implementation of a mini-max search. The mini-max approach is an AI algorithm which builds a tree of next possible moves for the next several moves, and picks the move which leads to the best outcome. This algorithm works well for our needs as it can easily scale its difficulty from beginner to advanced players. Our team found an open source implementation of the algorithm to use in the application. We decided that it was best to use an already implemented solution rather than implement our own, because it is already well tested, and we have a limited time of one semester. If we tried to implement our own solution in one semester, it would likely contain many bugs or even be incomplete.

**3.3.4 Output Rendering**

The output rendering subsystem is responsible for rendering 3D graphics onto the phone’s display. Users can then see the calculated move through rendered 3D shapes such as arrows and cylinders. There are two objectives that this subsystem will perform: rendering graphics to correctly illustrate the calculated move and allow users to check the calculated game state.

Figure 4 shows the inputs and outputs of the output rendering subsystem. The subsystem first receives the current locations of all the detected checker pieces. Moreover, it will receive the location of the checker piece to move and its destination location from the checkers AI subsystem. Upon receiving these necessary coordinates, the output rendering subsystem will render 3D images of virtual checker pieces and arrows on the display. These images will illustrate the move calculated by the checkers AI subsystem. The end points of the arrow will signify the start and end positions of the checker piece. Some moves in checkers can involve multiple jumps. For these moves, the subsystem will render multiple arrows and multiple checkers pieces to indicate all the jumps.

Output Rendering

Checker piece locations

Destination location of suggested move

3D images of arrows and circles

**INPUT**

**SUBSYSTEM**

**OUTPUT**

**Figure 4. Inputs and Outputs of Output Rendering**

In addition to the moves, the output rendering subsystem will also display red and black circles over locations of all identified pieces. These circles will allow users to visually check that the application has identified the pieces correctly. The application will also render green and blue virtual checkers pieces on top of the pieces to indicate detected king pieces for the red and black players respectively.

Lastly, while the checkers AI is calculating the next move, this subsystem will render lines that scan across the board to create the illusion to the users that the AI is scanning the board and determining the next move.

**3.3.5 User Interface**

As a significant design requirement of our checkers solver application, the user must be able to effectively communicate with the system. In order to satisfy this constraint, we developed a user interface subsystem, which serves as a communication medium between the user and the individual subsystems within the application. Although not a requirement, we believe that the UI will also enhance the user’s experience while improving the user’s learning potential. The primary features of the UI consist of on-screen buttons, instructive pop-up messages, and a changeable difficulty setting that is accessible from the menu.

To adhere to the simplistic nature of our application, we have implemented two user-friendly buttons that overlay the real-time image from the camera displayed on the LCD. These buttons were developed to allow the user to easily start certain processes and change significant settings. The first button is referred to as the “Calculate” button and is used to signify that the user desires the application to render the next suggested move. By pressing the “Calculate” button, the UI notifies the game state analysis subsystem to identify the current game state, which subsequently calls the next appropriate subsystem until the results are displayed on the LCD. The second button is referred to as the “Change Player” button. This button allows the user to easily express his or her color to the application. Thus, pushing this button will modify whether the application renders the next suggested move for the black or red pieces.

To assist the user with anomalies in the expected application behavior, we have included informative messages that temporarily pop-up on the main screen when certain situations are identified by the application. For example, a warning message is displayed to the user if the tracking subsystem cannot identify the unique checkerboard from the live camera feed. Such a message informs the user that he or she should re-position the phone so that the application can properly identify the game state and render suitable results. Because these messages may not give exact instructions on how to fix particular errors, the user is informed that he or she should reference the user manual to find helpful hints on how to improve the application’s performance.

As a final feature of the UI, there exists a difficulty setting that allows the user to easily modify how long the application should spend searching for the next suggested move. While the setting to change the user’s color is easily accessible from the primary screen, this feature is only available from the menu button on the user’s android mobile device because we believe a typical user will not modify the difficulty setting often. In addition, the UI stores the difficulty setting indefinitely, allowing the user to modify the setting once, rather than applying his or her preference each time he or she opens the application.

**4.0 DESIGN IMPLEMENTATION**

To successfully implement a working prototype, it is necessary to effectively manage projects such that the solution is completed within the given time constraints and budget, yet still maintains consistency with all specifications and requirements. In order to do so, projects must often make use of other existing products or techniques to increase efficiency. In addition, it is often essential to break larger problems into smaller ones such that individual subsystems can first function independently. System level integration often then becomes easier to debug and render the desired results. The following section discusses the implementation of our Interactive Checkers Solver design solution as well as the challenges faced along the way.

**4.1 SUBSYSTEM IMPLEMENTATION**

As the first milestone of our design implementation, we decided to individually work on the five different subsystems of the Interactive Checkers Solver application and develop a robust solution before attempting any system level integration. These subsystems include the checkerboard tracking, the game state analysis, the checkers algorithm, rendering output, and a user interface.

**4.1.1 Checkerboard Tracking**

In order for the Interactive Checkers Solver application to function as desired, it is necessary for the application to identify a checkerboard. This is accomplished by the checkerboard tracking subsystem, which uses Qualcomm’s SDK to track objects such as a checkerboard. By uploading a picture of our designed checkerboard and adding the binaries to our project source code, Qualcomm’s SDK provides critical information such as the location of the center of the checkerboard as well as the orientation of the checkerboard. Knowing the orientation of the checkerboard is critical for our application, as it allows it to identify which direction the pieces are allowed to move. Using Qualcomm’s SDK to track the checkerboard rather than implementing our own tracking solution provided extra time to focus on other subsystems of the project as well as complete a qualitative assessment of the SDK’s tracking capabilities.

**4.1.2 Game State Analysis**

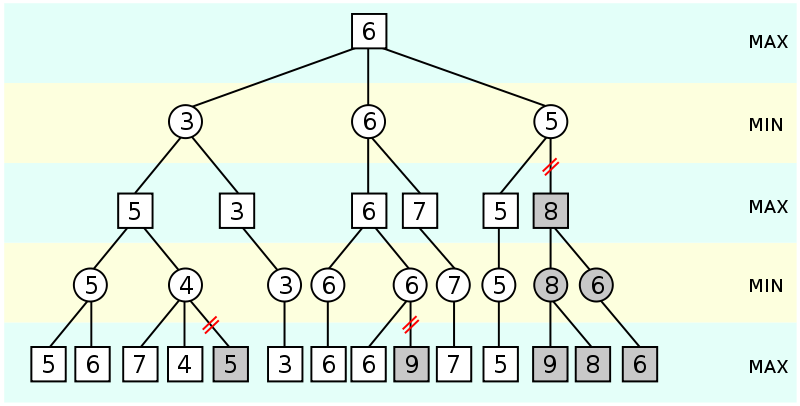
The game state analysis subsystem has two main goals: to sample the display’s color information and to determine existing checkers pieces. This section delineates the application’s software implementation of these routines.

The subsystem must first correctly identify a checkers square of interest on the display. For a tracked object, Qualcomm’s AR SDK provides its geometric attributes including its width, length, and center coordinates [2]. Using such attributes for the tracked checkerboard, the subsystem calculates the set of display coordinates that correspond to a particular square by performing positional arithmetic. The SDK also provides a method to extract the color information of a particular display pixel [2]. A pixel corresponds to a unit of display coordinates, and stores its color attribute as a set of red, green, and blue (RGB) intensities. For instance, a maximum R value with minimum B and G values indicates that a pixel contains a red color. If the user runs the application in a dark environment however, a red checkers piece may in fact appear black. To combat this defect in piece detection, the subsystem applies an image processing technique called histogram equalization. Histogram equalization is an image processing technique used to enhance the contrast of an image. Using the technique, color intensities are adjusted to a new value such that all intensities are well distributed in the intensity space. The operation is done separately for RGB values, and in effect brightens images captured in dark environments. For each pixel within the current checkerboard square, the subsystem stores the adjusted color value. If any pixel falls outside of the display’s range, the subsystem returns with a failure output.

The subsystem subsequently uses the extracted color information to determine the color of any existing checkers piece. Using each pixel’s color intensity, the subsystem applies a threshold technique to determine the RGB component with the highest intensity. For instance, if a pixel has an R value higher than some threshold, and G and B values lower than the threshold, the pixel is determined to be red. The subsystem applies the operation for pixels within the checkers square, and identifies a colored checkers piece if the majority of pixels were determined to be that color. For example, if the majority of the pixels of a square were red, then the subsystem concludes that a red checkers piece exists on that square. The described operation is repeated until all checkers squares have been processed, at which point the game state analysis exits after updating the game state.

**4.1.3 Checkers AI**

The checkers AI subsystem is one of the more important subsystems in the project; it is responsible for using the state of the game to calculate the next suggested move. In order to provide a robust solution, we researched different algorithms and discovered that the mini-max algorithm was a great solution to solve the game of checkers. The mini-max algorithm works by generating all the possible moves of the game for a certain number of moves ahead as seen in Figure 5. Once it has generated all the final moves, it assigns them values based on how good the state of the game is for the player, with higher values representing a better move. Next, the algorithm assumes that both the player and the opponent will play perfectly by alternating between finding the minimum and the maximum of all the moves generated. Finally, the algorithm picks the move that leads to the highest scored value. Alternate popular solutions in finding the next suggested checkers move include to create a pre-computed database of all the possible moves or use a combination of searching an endgame moves database. We chose the mini-max algorithm because it fits well in our specifications of finding the next move within one to five seconds while providing a fairly advanced move.



**Figure 5. Mini-max Tree[3]**

Although we originally planned to implement our own mini-max checker AI to solve the game of checkers, we ultimately used an open source implementation of the checkers AI, and have integrated it with the other subsystems. It is implemented in C and has been imported into our application using the Android native development kit. After the game state analysis determines the states of the checkerboard, it passes it into the AI, which calculates the next move and returns it back to the game state analysis to be used for rendering graphics on the screen. We chose to use an open source implementation because it would require more than a semester to develop the source code to work perfectly and to be optimized to run fast, even though the algorithm is fairly straight forward. Moreover, we would also require much more time to test the code, whereas the open source implementation has already been used in a few mobile applications.

**4.1.4 Output Rendering**

The two primary goals of the output rendering subsystem include rendering intuitive 3D graphics to illustrate the move and allowing users to double check that the application is functioning correctly. The following description describes the details regarding the implementation that achieves these goals.

There are two shapes that the subsystem renders frequently: arrows and cylinders. This subsystem makes use of OpenGL ES 2.0 to render these shapes. Everything in OpenGL is rendered using few primitive shapes. These primitive shapes are triangles, lines, and points. Hence to render a circle, the subsystem makes use of hundreds of tiny triangles to render a circle that appears to have smooth circular edges. To render an arrow, the subsystem renders two triangles to render a rectangle as well as a normal primitive triangle to complete the arrow. Finally, the subsystem renders these shapes in the 3D space, so the subsystem renders more triangles to give depth to the shapes. OpenGL provides an API to control how a primitive shape should be rendered. For example, a programmer can give three vertices to render a triangle in any way. This gives the programmer a lot of flexibility and control over how the shapes will be rendered in the final output. In addition to the circles and arrows, the subsystem also renders several lines to show that the checkers AI is scanning the board to calculate the next move. This is done by rendering several lines that decrease in alpha value, or the transparency.

After being able to render these shapes, the output rendering subsystem uses coordinates to render these shapes in any desired location. The subsystem also receives other information like whether the movement is a capture movement or a simple movement, and the subsystem implements the logic to take the appropriate action based on this information.

**4.1.5 User Interface**

To allow effective communication between the user and the Interactive Checkers Solver application, our design solution includes an intuitive user interface that provides game time buttons, instructive messages, and a difficulty setting menu. Because developing such features with an object oriented language such as Java would be a difficult task if creating all source code from scratch, we leveraged Android’s libraries to implement our user interface. Using Android’s libraries not only provides assurance of a well-tested solution, but also significantly reduces our project time requirements and increases our efficiency.

To reduce the amount of necessary Java code, we first developed the organization and layout of the user interface using an extensible markup language (XML) format. In addition to a reduction in code size, implementing the user interface in XML allows separation between the presentation of the application and the code that controls the behavior. A primary benefit of this organization includes the ease at which programmers can create XML layouts for different screen orientations, sizes, or languages without having to modify source code and recompile. Using the XML file format, we created layouts for the menu, difficulty setting and help pop-up dialogue boxes, and the buttons that appear in the main screen of the application.

As a second step to creating the user interface, we developed the functionality of the features listed in the XML files. To create basic objects such as buttons or checkboxes, we imported the android widget package. We could then set parameters such as the *OnClickListener* to perform specific functions after a widget is triggered by the user. Displaying text could also easily be realized with the widget package through a toast. To create the menu, we used such features as the *MenuInflater* and an *AlertDialog*. These android tools allowed for easy implementation of pop-up windows and displaying buttons when the menu button is pressed on the mobile device.

**4.2 SYSTEM INTEGRATION**

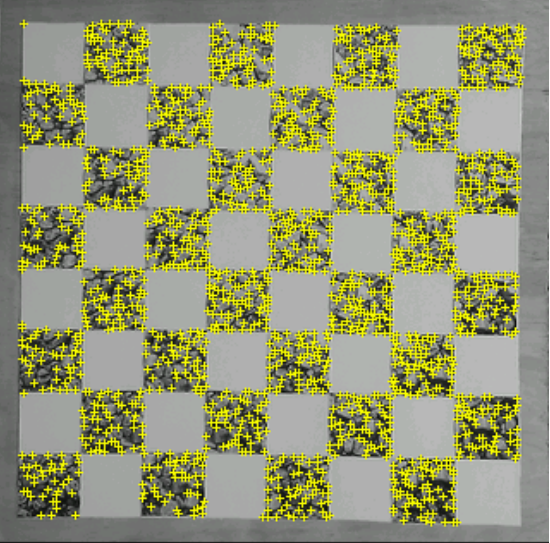
During the second phase of implementing our design, we integrated all the subsystems of our Interactive Checkers Solver application. Although the integration of all the subsystems was primarily a straightforward process, it was necessary to modify existing functions and write additional source code in order to complete the process. In addition, we chose to slowly integrate different portions of the individual subsystems such that only minor modifications would be made. Thus in the case of errors, we could easily identify the source of the erroneous behavior and make modifications as necessary. This approach of system integration worked well and allowed individual team members to work closely together.

**4.3 CHALLENGES**

During the implementation phase of our design of the Interactive Checkers Solver application, our team frequently met obstacles that needed to be addressed. The most significant challenges however included designing a checkerboard to meet the limitations of Qualcomm’s Augmented Reality SDK, handling erroneous data caused from reflected light from the checkers pieces, and the omitted path of the suggested move in the checkers AI.

**4.3.1 Checkerboard Detection**

Although a few limitations to Qualcomm’s Augmented Reality SDK exist, a critical limitation that effects our application is that the image targets should be rich in detail and not contain repetitive patterns. Tracking a unique object is essential because the application can thus identify the orientation of the board with regards to the position of the camera. Because normal checkerboards contain a repetitive pattern, the SDK cannot identify the checkers game using a normal black and red checkerboard. To alleviate this constraint, we developed a checkerboard using the existing stone image provided by the SDK with a white grid overlaid on top in order for the image to resemble a checkerboard. Because the existing stone image had tracked well using the SDK’s tracking algorithm, we believed that this checkerboard design had a high probability of being tracked. After experimentation, we were able to prove this solution to be robust. Figure 6 illustrates the unique features of our checkerboard identified by the tracking rating system.



**Figure 6. Tracking Rating System Results of the Unique Checkerboard**

**4.3.2 Light Reflection on Checkers Pieces**

During the game state analysis, the application depends on the color of the checkers pieces captured by the phone’s camera. Because the checkers pieces were made of reflective material, bright lighting conditions created glares at some angles. In the image captured by the camera, the glare caused pieces to appear white at some angles. Consequently, the game state analysis subsystem misses the identification of such pieces. To avoid the issue created by the glare, our team redesigned the checkers pieces. We placed colored circular stickers on both sides of the checkers pieces, corresponding to the original color. As a result, the modification enhanced the accuracy of the piece detection under bright environments.

**4.3.3 Omitted Checkers Path in Checkers AI**

One of the challenges in using the open source implementation was that only the final state of the checkerboard was calculated; the move or jump made by the checkers piece was not stored anywhere. However, it is important for our application to know exactly which piece moved, and how it moved, especially if it makes multiple jumps in one move. To solve the problem, we compared the original state of the game, to the one after the move had been calculated to find which pieces had moved. We were able to implement an algorithm that uses this information to find the path of the checkers pieces.

**5.0 TEST AND EVALUATION**

The application relies on inputs that vary based on how the user handles the application, such as the phone’s position. One of the challenges in building the application was to find the range of inputs the application would properly function under, and have it consistently give accurate results within the proper input bounds. Therefore, we tested and evaluated the application under different angles and distances from the board to ensure the user knows what the ideal position for the phone is. Next, it was tested under different lighting conditions to find the perfect place to play the game with the highest accuracy. Finally, the checkers AI sub-module was tested to see how well it plays against average checkers players and ensure proper functionality when given valid checkers inputs.

**5.1 ACCURACY OF CHECKERBOARD TRACKING**

In our application, the checkerboard must be tracked in order for any of the subsequent operations to run. To study the limits of the tracking system, we tested the tracking of the checkerboard through several variables: camera view angle, distance, and lighting.

We tested the subsystem by answering the binary question: “tracked” or “not tracked” for 10 degree angle increments starting from 0 to 70 degrees, and 2 inch distance increments away from the board. We performed experiments in both indoor and outdoor lighting environments. In the indoor experiments, testing was done by utilizing a lamp while varying the distance at 2 inch increments away from the light source. In the outdoor experiments, we conducted the tests under direct sunlight as well as in projected shadows.

After experimenting with the consistency of the tracking, we found that the checkerboard was tracked best in uniform lighting conditions. Also, the user should avoid playing in areas where glares or reflections from the light source on the checkerboard exist as these two conditions can affect the tracking of the checkerboard. In addition, we discovered that the user needs to point the camera above 50 degrees with respect to the board at a distance less than 7 inch away from the board for the checkerboard to be initially tracked. Once it is tracked, it will track the board continuously within the minimum and maximum distance and angle requirements, which are listed in Tables A1 and A2 in Appendix A. Even though the tracking subsystem is able to track the board within the minimum and maximum distances shown in the tables, it is highly suggested and recommended that the camera be held at an approximate distance of 5 to 8 inches and at angle of 50 to 70 degrees with respect to the checkerboard to achieve the best result and allow the entire checkerboard to be visible to the camera’s view.

**5.2 SPEED AND ACCURACY OF GAME STATE ANALYSIS**

In our application, we require that the game state analysis should take no more than five seconds. We also specify that the game state must be correctly identified at least six out of ten trials. This section will analyze the testing done to ensure these requirements.

**5.2.1 Timing Analysis of Game State Analysis**

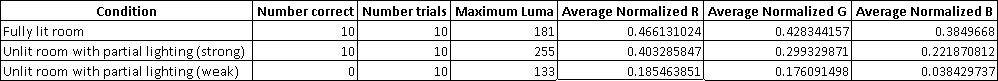
The game state analysis subsystem samples the display pixels in areas within a legal checkerboard square, adjusts the readings for lighting effects, and samples them a second time to identify existing pieces. To determine the required time, we used functions given by the C library to extract the number of seconds taken to run the software routine. For our experimentation, we extracted the number of seconds taken for a varied number of sampled pixels. The maximum number of pixels corresponds to an instance when the checkerboard fills up the display, and the minimum corresponds to an instance when the checkerboard is too far for tracking. Figure 7 is a plot illustrating the trends in the time required to run the game state analysis algorithm. The routine does not take any more than 0.5 seconds, and thus meets our specifications of a five second maximum.

**Figure 7. Time Required to Run the Game State Analysis Algorithm**

**5.2.2 Accuracy Analysis of Game State Analysis**

For our application, we required that the game state analysis must accurately determine the checkers game state at least six out of ten trials. To test the accuracy, we counted the number of times a game state was correctly identified in three room environments: a fully lit room and an unlit room with a strong and weak partial lighting using a 60W lamp. In each setting, each four colored pieces (red, green, blue, and black) were placed on the board. If the pieces were correctly identified and no empty square was mistakenly recognized, we determined the trial to be a success. Ten trials were completed for each setting, and the number of successful trials for each setting were counted. See table 1for an illustration of the result of this experiment. In each experiment, the maximum luma and the normalized average R, G, and B values were recorded. Luma is an image attribute representing the brightness of an image [4]. The average R, G, and B values were normalized by dividing the values by the maximum value, 255. From the table, we can see that the game state analysis was accurate 10 out of 10 times for a lit room and an unlit room with strong partial lighting. However, the blue piece was incorrectly detected as a black piece in all trials in an unlit room with weak lighting. The outcome can be explained by the low luminance value, and the low average B value as recorded. These attribute cause the captured image to appear dark in general, and blue colors to be less distinguishable. We thus concluded that the application must be played in a well-lit environment to meet our accuracy requirement, six out of ten trials.

**Table 1. Accuracy of Game State Analysis in Various Conditions**



**5.3 FUNCTIONALITY OF CHECKERS AI ALGORITHM**

Our application relies on the checkers AI subsystem to be consistent in the next move calculation quality, and to handle any valid input from the previous subsystems. For this subsystem, two testing topics include its performance against playing average players, and robustness in handling various inputs.

**5.2.1 Move Suggestion Quality**

For the application, we expected the checkers AI to consistently produce quality move suggestions. We tested the quality of the checkers AI by playing against humans and evaluating the results. The checkers AI was tested by playing human players online using services such as Yahoo Games which provide a means of playing checkers with other players from around the world. We tested the checkers AI by simulating the human player’s moves on the checkers AI interface. After the checkers AI suggested a move, the move was copied onto the board with the real human player; effectively, the human played versus the checkers AI. The checkers AI was allowed only a second to calculate the next move, which gave us a good base kill measurement. After playing with ten different players, the checkers AI successfully won nine of the games, and drew one. The checkers AI won 90% of the games which is greater than our initial specification of winning more than 75% of the games, and therefore the quality of the checkers AI was deemed successful.

**5.2.2 Input Handling**

To ensure that the checkers AI does not crash when computing the next move, the algorithm was tested by running with various valid input combinations. To start the calculation, a function is called with array parameters filled with numeric values representing the different pieces on the board. Since it is not practical to run all possible input combinations for all possible game states, we tested the algorithm by automating the test to play against itself. A simple program was written which ran the checkers AI over the course of twenty four hours. The program tested the checkers AI by switching the side it was playing for and thus played the game of checkers versus itself. Once a game was over, a new game was started and the testing continued. The test ran successfully over 24 hours, which gave us a good indication that the checkers AI code is able to properly handle most input states. Although this test is not conclusive on whether or not the checkers AI is completely bug free when handling input cases, it gave us good enough results during the tests to conclude that it is robust enough to use in the application.

**6.0 TIME AND COST CONSIDERATIONS**

We were able to meet our time and budget constraints for developing the Interactive Checkers Solver Application. Because we divided the system into individual subsystems and assigned each subsystem to primary and secondary owners, we did not have any issues with completing our solution within our time constraints. The cost required to develop this application prototype included the cost of the MDP device and the cost of creating the unique checkerboard and pieces used for our application. The MSM8655 Snapdragon MDP is provided for sale by BSQUARE Corporation at a cost of $995.00 [5]. Although we used Qualcomm’s AR SDK as a software development environment, the installation files can be downloaded free of charge; therefore, the SDK usage poses the team no addition charges for the application development. See Table B1 in appendix B for a detailed description of the required material costs in our project.

**7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN**

Our application is purely software and does not contain equipment, machineries, or materials that could pose any threats to the safety of the user. Therefore, using the Interactive Checker Solver application does not have significant safety concerns that could endanger human life or produce any environmental damage. Regarding the design’s ethical aspects, an engineer must make sure to follow the application development rules and guidelines of the system they are developing for. While writing code, it is important not to use code without permission. The Android and the Qualcomm SDKs are available for free under their respective licenses; therefore, our application is free to use them as we like. The checkers AI code was another major part of the application which was not written by us. However, the author of the code has open sourced the AI and allows anyone to use it as they wish. This software application will serve public interest of checkers players to improve their skills and knowledge, and learn effective checker playing strategy as well as provide enjoyment for the user when using our application.

**8.0 RECOMMENDATIONS**

The Interactive Checker Solver is a software application that provides the user an intuitive way to train themselves in order to improve their checker skills and knowledge. This section will provide recommendations on how others might enhance our application, such as developing an Interactive Chess Solver and possibly further development of AR applications.

* 1. **INTERACTIVE CHESS SOLVER**

Other senior design teams could possibly use our findings of the SDK and tracking from our project and complete an “interactive chess solver” application. The interactive chess solver will be similar to the Interactive Checkers Solver as it will allow users to improve their chess playing skills and knowledge, but could pose additional and more interesting challenges. One challenge of an interactive chess solver is that it would require the SDK to recognize the different chess pieces along with a more sophisticated set of rules to follow, such as what moves certain piece can make. Another challenge would be a creation of a new board, as the designers would have to take into account that in chess there can be pieces on each square. If future teams could use our information on what we know to overcome these interesting and challenging obstacles, then it should lead to a more interesting augmented reality application.

* 1. **DEVELOPMENT OF AR APPLICATIONS**

As previously mentioned, future teams could also look into further development of AR applications. While completing the checker solver, we discovered that the debugging was a hassle because we would need to load the application each time and manually test it out using Qualcomm’s SDK. Future teams could explore other possibilities of an automated or systematic debugging strategy like using an emulator when working with AR SDK applications. In addition, future teams could enhance Qualcomm’s AR system by eliminating the need to have highly feature-rich objects in order for the image to be tracked.

**9.0 CONCLUSIONS**

This report provided a comprehensive documentation of the development of an AR Interactive Checkers Solver application for mobile smart phones. We discussed the application as a solution to a need to provide checkers players an intuitive way to learn effective playing strategies. The application used the AR technology to illustrate effective next moves by rendering 3D graphics on a smart phone’s display. The report described the application’s requirements and specifications, including its operating accuracy and user-friendliness. Our project composed of five software systems: tracking, game state analysis, checkers AI, output rendering, and a user interface. We described key operations and implementation details and challenges of each subsystem. Through the application development, we faced challenges that required the design of custom checkerboard and pieces. In this report, we also assessed the accuracy, speed, and functionality of the application operations. The project did not cause major issues in areas including development time, cost, and safety and ethical issues. Finally, we discussed our project as a platform for future AR application development. The necessity in our project to develop custom checkerboard and pieces is a limitation that can be further investigated for improvement. However, the Interactive Checkers Solver project achieved its goal to provide an interactive experience in playing checkers games. The project demonstrated the AR technology as an innovative topic for future entertainment systems for smart phones.

**REFERENCES**

[1] “DROID by Motorola, A855”, http://developer.motorola.com/products/droid (current 20 Nov. 2011).

[2] “AR API – Qualcomm Augmented Reality,” Qualcomm Inc., https://

ar.qualcomm.com/qdevnet/api (current 20 Nov. 2011).

[3] “Algorithms,” Stanford, http://www.stanford.edu/~msirota/soco/minimax.html (current 20 Nov. 2011).

[4] Poynton, Charles. “YUV and luminance considered harmful,” http://poynton.com/ PDFs/YUV\_and\_luminance\_harmful.pdf (current 20 Nov. 2011).

[5] “Product Catalog : EA-MSM8655™-based Snapdragon™ Mobile Development Platform with Android™ OS,” BSQUARE Corporation, http://store.bsquare.com/

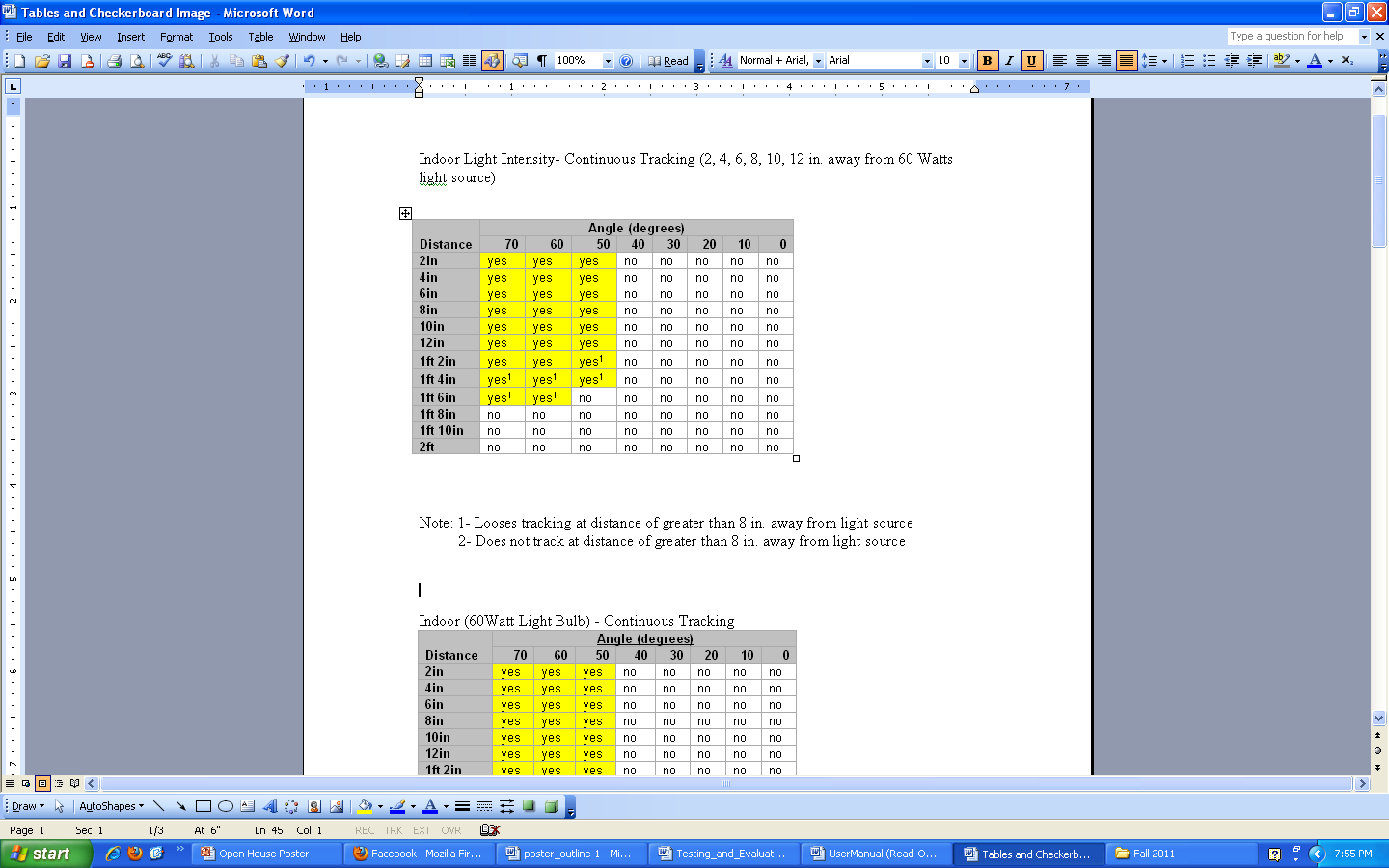
catalog/index.cfm?fuseaction=product&theParentId=160&id=2737 (current 20 Nov. 2011).

**APPENDIX A – TRACKING TESTING RESULTS**

**APPENDIX A – TRACKING TESTING RESULTS**

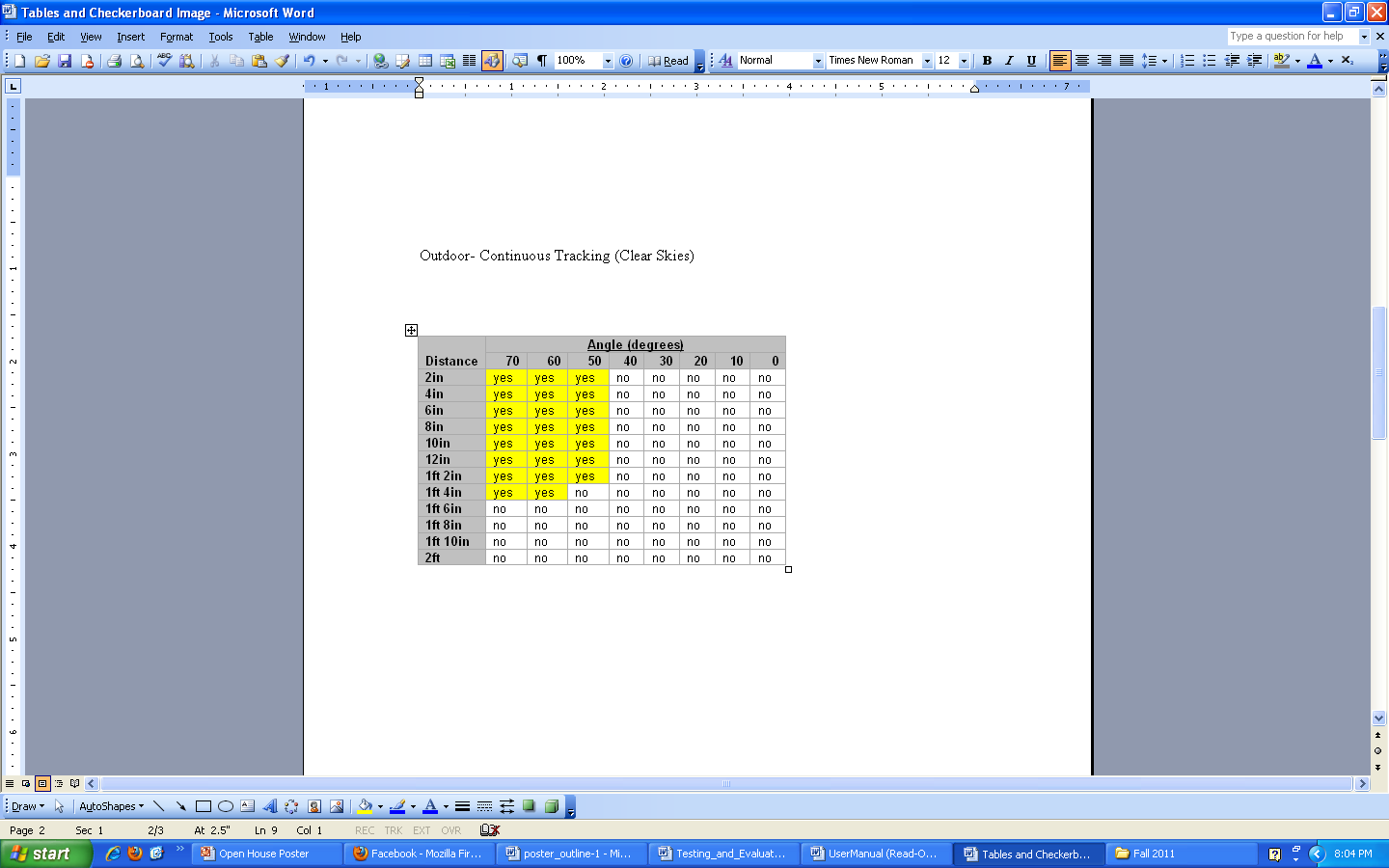
The following tables illustrate whether or not the unique checkerboard was tracked by our application under various distance and angles from the phone’s camera. A “yes” indicates that the checkerboard was tracked, and a “no” indicates that board was not tracked.

**Table A1. Continuous Tracking in an Indoors Environment**



Note 1. Does not track at distance >8ft away light source

**Table A2. Continuous Tracking in an Outdoors Environment**



**APPENDIX B – BILL OF MATERIALS**

**APPENDIX B – BILL OF MATERIALS**

The following diagram provides a description and cost of each materials used to develop the application software.

**Table B1. Bill of Materials for the Project**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Component** | **Description** | **Quantity** | **Performance Criteria** | **Price estimate** | **Prototype number** | **Vendor** |
| Snapdragon MDP | Smart phone device used to run application | 1 | Basic smart phone functionality | $995.00 [1] | MSM8655 | BSQUARE |
| Augmented Reality SDK | Software environment to develop application | 1 | Functionality of software compilation | $0.00 | Version 1.0.0 | Qualcomm |
| Checkerboard and pieces | A set of custom checkers components used to run the application and a wood for Checkerboard | 1 | Ability to be identified by the application | $10.00 (pieces)  $10.00 (wood) | N/A | N/A |