CheckARs Final Report

John Matienzo *(995456420)*, Valentin Berbenetz *(996156671)*

Word Count: (1,896)

1. INTRODUCTION

Modern day mobile devices come equipped with powerful processors, high-resolution screens and cameras, all of which allow the device to participate in the emerging field of Augmented Reality (AR). This technology gives developers a whole new set of tools, which they can use to expand their creativity when creating new applications and games. Given this, CheckARs was created to add a modern spin to a classic board game, Checkers.

The overall goal of the CheckARs application was to create a more realistic way of playing Checkers on a mobile device. Instead of moving virtual game pieces on the device via either touching the screen or pressing buttons, the app requires both players to move a virtual game piece with their finger in front of the camera, mimicking an action used with physical pieces. This was created with the help of Qualcomm's AR API which would recognize a target (an image printed on a physical piece of paper) via the camera, and would overlay the virtual board and game pieces on the device's screen. This method of gaming gives the user a "real" gaming experience on the go, without needing to carry the physical board game.

2. OVERALL DESIGN

The app can be broken down into three distinctive top-level blocks: the Image Processor, the Renderer, and the Game Engine (Appendix A). The Image Processor uses the Qualcomm AR API to detect an image (referred to as the image target) using the device's camera. Then the Image Processor positions the renderer's graphic objects correctly on the device's screen. This feature is also dynamically adaptive - when the camera is moved around, the board always stays fixed at the same position over top of the image target. Additionally, the API is responsible for detecting the user's finger in front of the device's camera and determines

its position relative to the target. This is a key feature since the game board created for the app is a series of virtual buttons (virtual buttons are triggers floating in free space relative to the image target that activates events), which are triggered when the API detects the finger's position overtop of them.

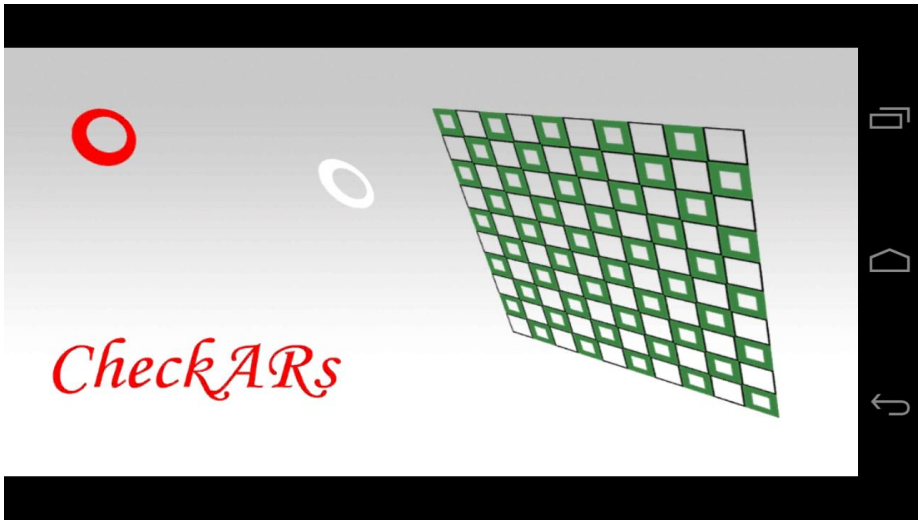
The second block, the Renderer, is responsible for rendering the virtual game board along with the 3D game pieces on the device's screen. The rendering is accomplished by using the OpenGL library, which gives the programmer a way to map 2-dimensional textures to 3-dimensional models. The library is written in C++ which only accepts function calls in that language. Android however requires source code to be written in Java, and as a result, JNI (Java Native Interface) has to be used to allow the programmer to bridge their C++ OpenGL code to the app's Java source code. To avoid the complexity of using OpenGL and the JNI, the team decided to use the Unity Development Environment, a 3rd party development environment which aids in the placement of graphics by performing all the OpenGL function calls in the background whilst reading the game objects' position updates from the programmer's scripts (created using C# or JavaScript).

The final block, the Game Engine, relates to the core functionality of the game. This block is responsible for managing the rules of the game, and maintaining game state. Here, the inputs are taken in from the image processing block, and from them, the current player's possible moves are determined and displayed to the player via signaling the renderer block. Once a selection is made, the game engine then signals the Renderer block again to update the user's move on the screen. Since this is a multiplayer game, the Game Engine block is divided into two sub-blocks. The first sub-block is responsible for performing "move" updates on the host's device, as well as sending the game state to the opponent's device across a WIFI network, in order to update the game on their device upon completion of the turn. The opponent player possesses the second sub-block, which is simply responsible for updating the game state with the information received from the host device, and acknowledging that it has completed. All of this functionality is implemented using scripts in C# within the Unity Development Environment, which is formatted to run on the Android Java SDK.

3. Statement of Functionality

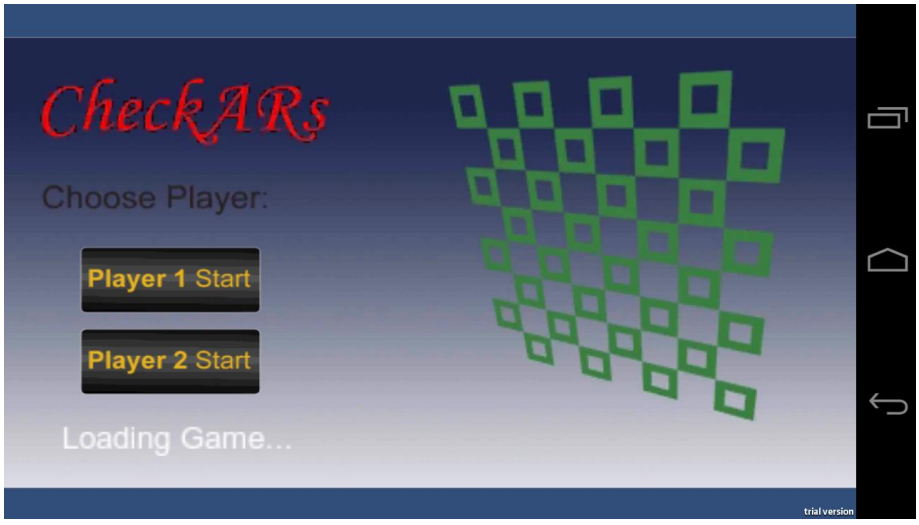
The working features of our application can best be demonstrated through an illustrated walkthrough:

3.1 Application Launch



The screenshot above is the splash screen that is displayed as the application loads dynamic libraries and the Unity Player (used for rendering).

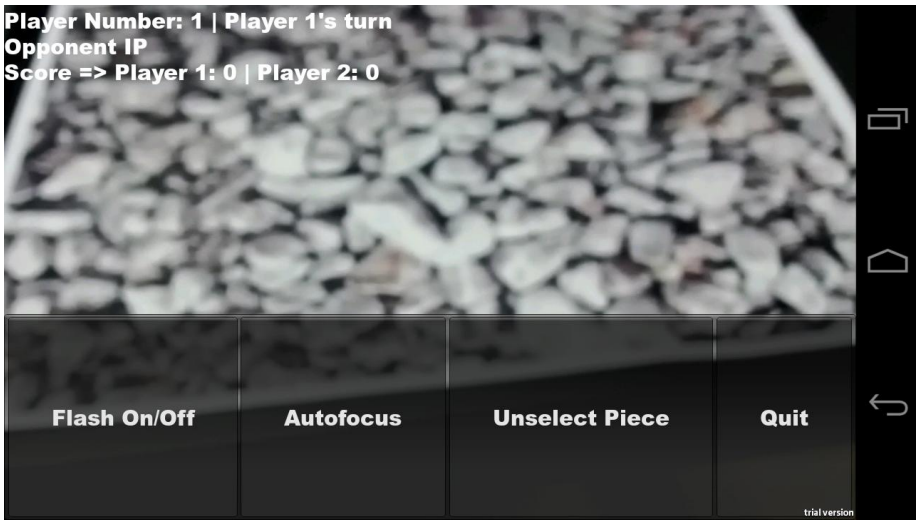
3.2 Application Main Menu



The menu above allows a user to select a player number. Once a player number is selected, the application waits on the main menu screen until an opponent is discovered on the network (discovery happens automatically). In the event that

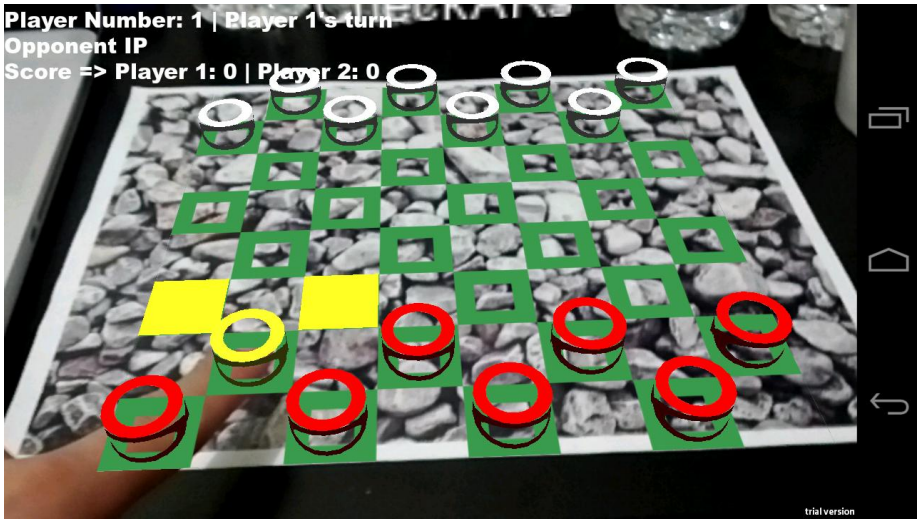
there is no network to connect to, there is also the option of using just one phone to play the game (this mode is activated if the user taps the checkered pattern on the menu screen)

3.3 Start of Game Play / In-Game Menu



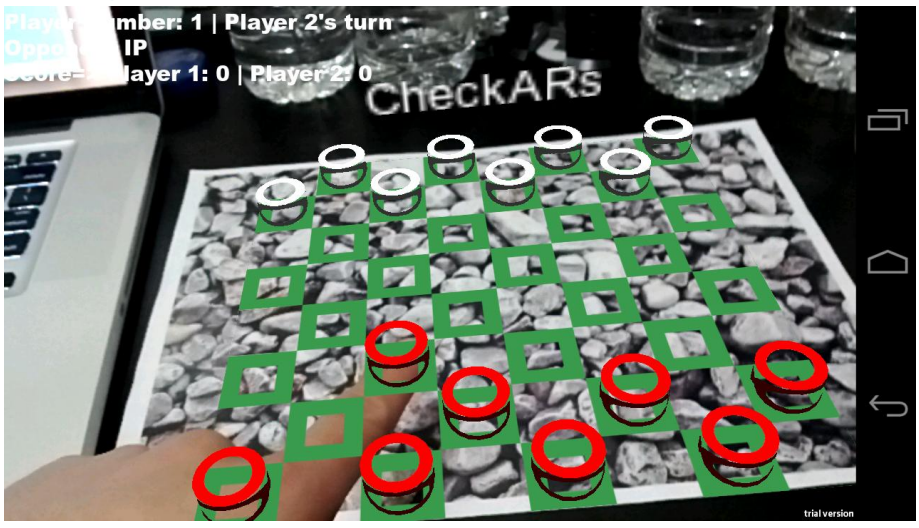
When the game first loads from the main menu screen, an in-game menu is overlaid on top of a camera preview screen. The user should use the "Auto-Focus" feature of the menu to ensure that the image target is clear and sharp. If the lighting conditions are dim, the user should turn "on" the phone's FLASH to ensure the image target is visible. To hide the menu, the user simply needs to tap anywhere on the screen.

3.4 Game Piece Selection



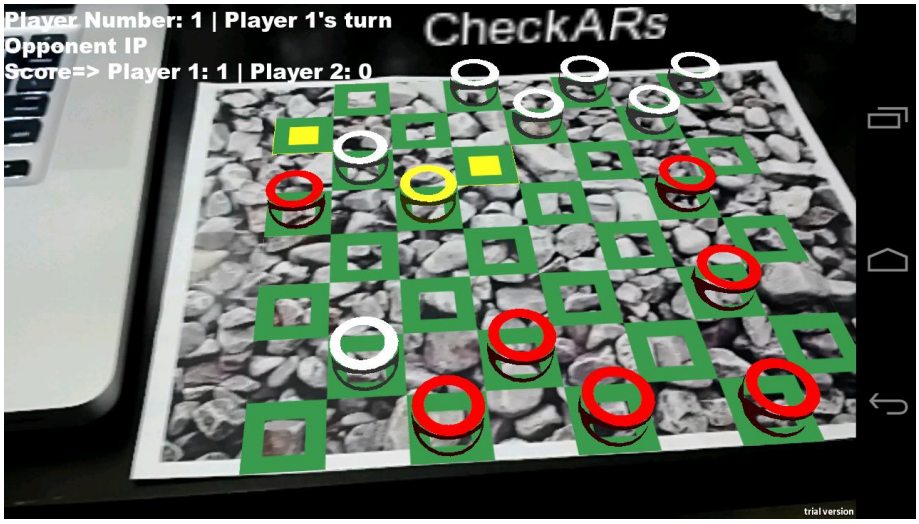
After the image target is in focus, the virtual checkers game board will be overlaid onto it. To select a piece, the user simply needs to place their finger on the desired piece (in front of the camera). A user knows when a piece has been selected when it turns yellow (as pictured above). Note: the game allows only valid movable pieces to be selected.

3.5 Moving Selected Piece to a valid tile



Once a piece is selected, a player simply moves the piece by sliding it onto a yellow tile (yellow tiles indicate a valid destination).

3.6 Scoring



There are two ways in which a player can obtain points:

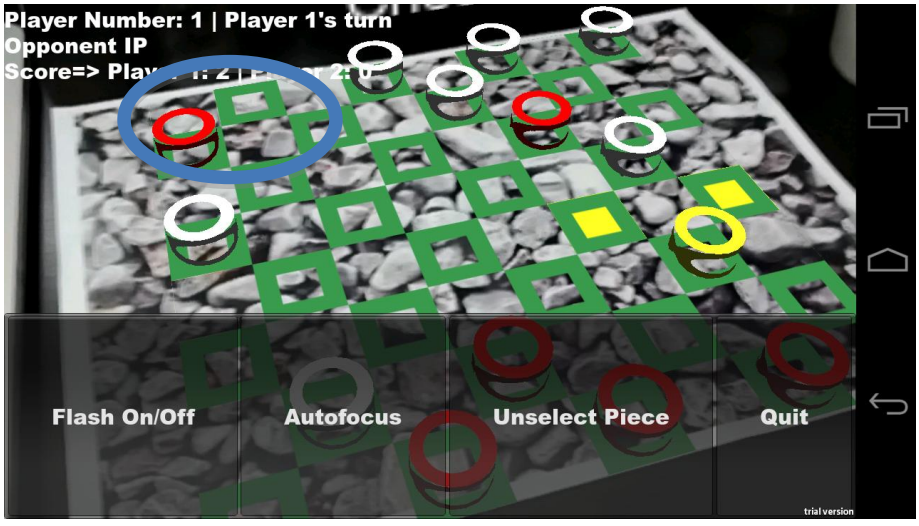
3.6.1 Opponent Kill

1 point - (the red player above [highlighted in yellow] can perform this move on his current turn)

3.6.2 Reaching the end

2 points - (the white player can perform this move on his next turn)

3.7 Unselect Piece



In the event that a player mistakenly selects a wrong piece, they can deselect the piece by activating the in-game menu (by tapping anywhere on the screen) and pressing “Unselect Piece.” In the above screenshot, player 1 would want to

unselect his current piece as his other piece (circled in blue) can move to the end of the board to obtain two points.

4 Key Learning Experiences

Over the course of developing this app, the team had the opportunity to learn how to work with many interesting and unique aspects of both game development, as well as the emerging field of Augmented Reality. First, the team had the opportunity to use a powerful game development environment, Unity, which gives the programmer more of a "what you see is what you get" programming experience as opposed to blind coding OpenGL in native code. The team found that when using this environment, developers are able to create and debug their games quickly and efficiently because they are presented with a visual representation of their game as they write their code. Unfortunately, we began coding our app using native code (JNI) to program the graphics in OpenGL, which added large volumes of complexity when it was paired with Qualcomm's AR API. Here, the team needed to manage the 3D textures (game board and pieces), along with all the virtual buttons positioned on the game board. Due to this, the team decided to use Unity fairly early in the project lifecycle to complete our application on time.

Another fundamental lesson that we learned over the course of this project was that the quality of the image target used by the AR API is important. The image target needs to be complex, and should contain many random edges with large contrast differences. Initially, the team created an image target that contained simple black text on a white background. Since such a target is sensitive to light changes, the team continuously ran into strange anomalies upon running the app, where virtual buttons would randomly be triggered (as shadows caused by the phone would be interpreted as a finger gesture from the user by the API). In addition, a poor quality target resulted in the API not properly fixing the graphic overlay on that target; in other words, the board and pieces were constantly jittering and inadvertently triggering the virtual buttons.

5 Contribution by Group Members

John

- Created Textures as well as game board and game piece layout on device screen (Unity)
- Laid out the virtual button grid and created associated button-triggered functions to allow player interaction with game pieces
- Created a game manager to maintain game state as well as signal a player's turn
- Created main menu as well as in-game menu
    - Integrated auto-focus, flash and quit options within the in-game menu
- Integrated WIFI/network communications into the game that allows automatic discovery of opponent player in the same subnet and updates game state on both devices

Valentin

- Created Textures as well as game board and game piece layout on device screen (OpenGL/JNI)
- Created methods to determine the current player's possible piece selection
- Created methods to determine what possible moves the selected piece can perform (attack of regular move)
- Created attack functionality where the opponents pieces would be flagged for removal and the current player's pieces were updated correctly

6 Future Work

There are several features which the team would like to implement for the app in the future. First and foremost, the accuracy of game piece selection via finger gestures needs to be improved. Currently, if a user attempts to select a piece located within the interior of the board, they may inadvertently select the wrong piece by mistake due to their hand triggering the wrong virtual buttons. To solve this issue, the team proposes to use a probabilistic method whereby the algorithm would approximate the shape of the hand based on the virtual buttons currently pressed, and attempt to determine where the finger is located. Another method would be to record the time that all the virtual buttons are pressed, and the piece that would

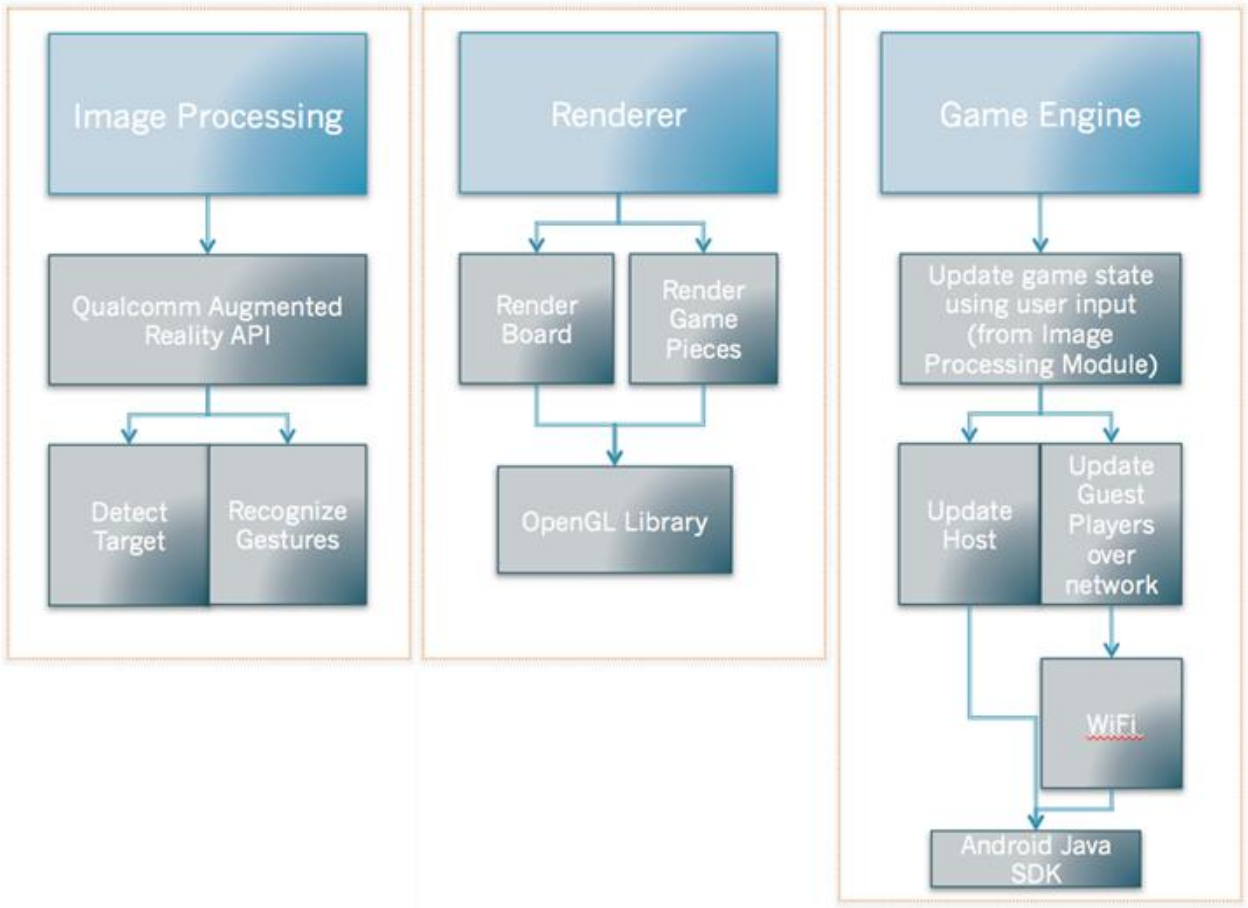
be selected would be the one that has had their virtual button held down for the shortest time (since the user would select that button last).

Another piece of functionality which the team would like to add would be artificial intelligence (AI), so that this game could also be played by a single player versus the computer. Finally, to make the game complete, we would like to implement the "King" functionality, where a piece that reaches the opposite end can become a super piece that can move both forwards and backwards.

7 Business School Consideration

We would be interested in having a Business School class on marketing and/or entrepreneurship take up our application.

Appendix A



Application Block Diagram