Brian Porter

Major: Bachelor of Science in Computer Science (Cybersecurity Minor)

Advised by: Dr. Hayes
Expected Date of Graduation: May 2021

Senior Project Defense (CSCI 499)

Index:

Problem Statement:

How many chess games exist in this world? If you were thinking of some estimate. You are likely wrong. Chess has been made and redone over and over since its creation. However, likely every chess board you have seen online or physically, usually fits the standard format of an 8x8 chess set with two players. However, that is not the only format of chess to exist. There are formats like 3-player chess in which the board can be shaped like a Y, a circle, or a hexagon and piece moves vary according to board shape. There is 4-player chess which is shaped like a plus sign and can have a variety of chess piece starting positions. Then there is 3D chess which has several different variants. While the normal 2-Player chess set is popular throughout the world, the average chess player knows little to nothing about the other variants of chess. This presents a problem not only for people that play chess regularly but also for people that do not know much about chess. There is so much untapped potential for these other variants of chess to rise in popularity. I believe this to be true given the popularity in many forms of simple games like Tetris that have arisen over the past few years. There are not only tournaments for Nintendo Tetris, but also tournaments for Tetris 99 which while holds to the basic rules of Tetris are both drastically different games (For reference Nintendo Tetris is made in the 90s and Tetris 99 is made in 2019).

Therefore, I believe that I can increase the popularity of some of these other chess formats in the same way that Tetris has been growing in popularity. The solution that I propose is a game that allows players to experience a variety of forms of chess and is accessible to a variety of people. This game is more like multiple games where players can play normal chess, 3-player, 4-player, and a few forms of 3D chess. This is an ideal solution because players can play with their friends for normal chess and new varieties of chess.

Description of Project:

This project in simplest terms, a “catalogue” of different chess games and will be called “War of Kings”. The idea is that two or more players will be able to play different chess formats. Each player will have set pieces under their control and will act on their given turn to checkmate their opponent’s kings. Rules may change different according to the format and there may be options for different rules. For instance, for 3-player and 4-player chess, when one player checkmates another player’s king there will be preset rules in which that defeated players pieces will either disappear or become controlled by the victor. There will be a main menu where the player can host a game or join a game and when hosting they can load a previously saved game. While a normal 2-player chess game seems simple to program, there are some rather complicated aspects, and it becomes even more difficult particularly for 3-player and 4-player chess (due to the shape of the board and placement) and 3D chess (due to the size of the board and potential formats). Speaking of which, 3D chess will have a few formats options for the players to select which changes the boards and positions of pieces.

Proposed Implementation Language(s): Blueprints. Future C++ implementation pending after completion of at least 90% of requirements.

Any software/equipment needed: Unreal Engine, one desktop PC or laptop (for testing online multiplayer capabilities can be achieved using one machine using the Unreal Engine Interface.)

Motivation: I find this project to be interesting and within my current knowledge to complete. In the past I have wanted to make a chess like game but making normal 2-player chess game would likely take me a month to do so one chess format alone is too simple for a project like this. I have considered developing an artificial intelligence (AI). However, that is not as interesting to me. Also, the definition of a good AI for this is too broad and I am not sure how qualified and skilled

I would be in making an advanced AI. Therefore, I wanted to focus on the other forms of chess. There are many 2-player chess and a couple 4-player chess games, but I have yet to see any 3-player or 3D chess games. The plan is to develop this first for local multiplayer on windows and then slowly but surely port the game to mobile and update it for online multiplayer capabilities. This is ideal considering I lack knowledge on many important networking techniques. Therefore, it is likely that online multiplayer will not be fully functioning for an extended amount of time considering what I am proposing right now is to develop up to five different chess game formats each with its own set of rules and logic. Making one game of chess would already be a difficult proposition considering the difficulty of later components such as functions for checkmating or stalemating and opponent's king. Still, despite my lack of knowledge on the subject, whether it is unreal engine, networking, or mobile game development. I will try my best.

Outline of Future Research Efforts: Currently I have a basic understanding of unreal engine blueprints and C++. I need to a little more research on game states and online systems along with collision systems for pieces.

Development Schedule:

*Note that all chess boards will be implemented as local multiplayer game first. Networking and certain Android mobile functionality was attempted early on but due to difficulties, lack of knowledge, and lack of time it was shifted to later development. Online Multiplayer Implementation was created alongside the single player implementation, but quick off schedule tests shown that it should be focused on after the base game is made. Also, this is a general schedule and is updated as needed. The Test Plan Schedule will be different as it is focused on

Testing things after implementations are made and is not shifted as often. Also, keep in mind this schedule is a reflection partially of what I have achieved currently and what I intend to achieve. The schedule milestones are on a weekly basis.

Term Explanations:

- Board State: The grid attachment to a board which allows a piece to be placed on the board
- Game State: The state of a player. Whether or not it is a particular player's turn
- Controller: Each game has its own controller. The input mappings and functions are the same but the way the implementations work is different. Each controller by the end should allow Xbox, Android, and Windows support. Xbox, Android, and Windows all will have camera and player movement but Android player movement was developed first. Controllers should also be able to cause a visual cue on a board cell when the cell is touched or a cursor is on the cell. Controllers should be able to select pieces owned to the current player and select a valid cell for the selected piece to move. More information is described in the Test Plan.
- Visual Representation: Some boards must be procedurally generated or created as a single asset does not exist to represent the board.

2020/2021

December 23rd: Create 2-Player Chess Board State. Begin creation of 2-Player Controller Input for Windows. Create 4-Player Chess Visual Representation of Board

December 30th: Create 4-Player Chess Board state and 4-Player Controller Input. Create piece selection, and game state change for 2-Player and 4-Player formats.

January 6th: Create validated movement system, and player HUDs for 2-Player and 4-Player formats. Set up testing for opening 2-Player and 4-Player Chess levels more efficiently

January 13th: Create a basic main menu that can exit game and open levels to 2-Player and 4-Player Formats. Add Android Mobile Input to 2-Player Controller and 4-Player Controller (player movement and camera control with visual gamepad)

January 20th: Attempt online multiplayer implementation for the current chess board set ups (In case implementations are not made at that point). Create visual representation of 3-Player Chess Board, Board state for placement of pieces, and HUD

January 27th: Finish visual representation of 3-Player Chess Board, Board state for placement of pieces, and HUD, Attempt to complete piece selection and map cursor and touch to grid. Disable Online Multiplayer functionality to test local multiplayer functionality.

February 6th: Creation of Alice Chess Board, Game State, Board State, and HUD. Improve main menu to attempt to allow players to host a 2-Player Chess game to have other Players join the Game.

February 13th: Creation of Alice Player Controller with Android and Windows Input. Create Validated Piece System for 2-Player and 4-Player Chess Boards

February 20th: Creation of Millennium Chess Board, Game State, Board State, and HUD

February 27th: Creation of Millennium Player Controller with Android and Windows Input

March 3rd: Adjust Android and Windows Controllers so both have player movement and camera movement. Start Validated Movement for Alice Chess and Millennium Chess.

March 10th: Finish Validated Movement for Alice Chess and Millennium Chess. Create pause menu and update all HUDs with a button to access the pause menu

March 17th: Improve Main Menu to allow player to play 2-Player, 3-Player, 4-Player, Alice, and Millennium Chess on local multiplayer. Create Check for local multiplayer for ease of testing

March 24th: Start special move cases for 2-Player, 4-Player, Alice, and Millennium Chess

March 31st: Start cases to see if King is in Check, is in Checkmate, or is in Stalemate for 2-Player, 4-Player, Alice, and Millennium Chess. Add Help Section to Main Menu and Pause Menu and ability for users to exit to Main Menu or Exit game through the Pause Menu

April 7th: Finish special move cases for 2-Player, 4-Player, Alice, and Millennium Chess. Start Functionality to remove players from the game. Create Game Ending Menus.

April 9th – April 28th: Break to focus on other classes

May 5th: Fix 3-Player Grid to allow for cursor and touch input for controller. Begin 3-Player Validated movement and Game State

May 19th: Start special move cases for 3-Player Chess. Debug any none bugs for current failed tests except for those involving Online Multiplayer Support and Android Mobile Support.

May 26th: Finish Check and Checkmate Cases for 2-Player, 3-Player, 4-Player, Alice, and Millennium Chess, and finish functionality to remove players from the game.

June 2nd: Add game ending events to 2-Player, 3-Player, 4-Player, Alice, and Millennium Chess and use previously made Game Ending Menus. Shift focus on Online Multiplayer Implementation

June 16th: Finish Online Multiplayer Implementation and start Android Mobile Implementation and add controller functionality for Xbox One Controllers. Update Project Requirements Document for future enhancement section

June 23rd: Add implementation to all chess games to show available moves when a piece is selected. Add Options Menu to Main Menu and Pause Menu

June 30th: Add options for chess board themes and Music.

July 28th: Create Artificial Intelligence for 2-Player Chess

August 25th: Create Artificial Intelligence for 3-Player Chess

September 29th: Create Artificial Intelligence for 4-Player Chess

October 27th: Create Artificial Intelligence for Alice Chess

November 24th: Create Artificial Intelligence for Alice Chess

December 1st: Convert Code to C++. Code generated by Unreal Engine in this manner usually looks very bulky so it may need to be cleaned up.

December 8th-29th: Break

2022

January 5th: Decide new formats of Chess to add. Make new schedule.

January 6th-Onward: Pending

Test Milestones:

***Note**: Test plan milestones shown does not detail success or failure. Only test attempt dates.
2020/2021

December 30th: 2-Player Chess Board Piece placement on board, Camera Controls for 2-Player Controller, Cursor interaction with board to highlight cells.

January 6th: 4-Player Chess Board Piece placement on board, Camera Controls for 2-Player Controller, Cursor interaction with board to highlight cells. 2-Player and 4-Player piece selection and change player turns

January 20th: 2-Player and 4-Player HUDs with player amount, color, and current turn. Main Menu button functionality, Android Mobile Input for 2-Player and 4-Player Controller allows player movement and camera control and touching board to select pieces

January 27th: Opening individual levels for 2-Player Chess and 4-Player Chess simulate 2 and 4 players playing game. Same functionality in previous tests for local multiplayer

February 6th: 3-Player Board is visually correct as described by resources, Piece placement on board. Camera Controls for 3-Player Controller, HUD is active

Creation of Alice Chess Board, Game State, Board State. Improve main menu to attempt to allow players to host a 2-Player Chess game to have other Players join the Game

February 20th: Alice Chess Board Piece placement on board, Camera Controls for Alice Player Controller for Windows and Android, Cursor interaction with board to highlight cells. Alice piece selection and change player turns. Validated Movement for 2-Player and 4-Player Chess

March 3rd: Millennium Chess Board Piece placement on board, Camera Controls for Millennium Player Controller for Windows and Android, Cursor interaction with board to highlight cells. Millennium Chess player controller cursor and touch interaction for piece selection and change player turns.

March 10th: All Controllers can move player and move camera on Windows and with mobile input.

Finish Validated Movement for Alice Chess and Millennium Chess. Create pause menu and update all HUDs with a button to access the pause menu

March 17th: Validated Movement for Alice Chess and Millennium Chess. Pressing Button on HUDs leads to Pause Menu.

Improve Main Menu to allow player to play 2-Player, 3-Player, 4-Player, Alice, and Millennium Chess on local multiplayer. Create Check for local multiplayer for ease of testing

March 24th: Main Menu Button Work. Local Multiplayer Activation when local game is selected in menu

Start Functionality to remove players from the game. Create Game Ending Menus.

April 9th – April 28th: Break to focus on other classes

May 5th: Special Moves on all Chess games. Help Section in menus have correct information. Can exit to Main Menu or quit game from Pause Menu. Game Ending Menus display.

May 19th: 3-Player Controller can highlight over cells accurately, can select pieces and moves are validated.

May 26th: Retests for previously failed tests. Special Moves for 3-Player Chess.

June 2nd: Get if King is in Check, if player causes Checkmate on another player, if Player causes Stalemate on another player for 2-Player, 4-Player, Alice, and Millennium Chess. Remove Player Functionality for 3-Player and 4-Player Chess.

June 16th: Game ending events and Menus.

June 23rd: Xbox controllers on Chess Boards allow for camera and player movement along with piece selection.

June 30th: Previous tests applied for Online Multiplayer (May take more time). Current player selecting piece on any board shows potential moves by highlighting cells. Options menu is accessible from Main Menu and Pause Menu

July 28th: Players can change what boards look like and can change music volume and choose different songs from options menu.

August 25th: Artificial Intelligence for 2-Player Chess

September 29th: Artificial Intelligence for 3-Player Chess

October 27th: Artificial Intelligence for 4-Player Chess

November 24th: Artificial Intelligence for Alice Chess

December 1st: Artificial Intelligence for Millennium Chess

2022

January 5th: Check Cleaned up code

January 6th-Onward: Pending

Research and Background

Due to my initial lack of knowledge of Unreal Engine Blueprints and programming chess formats I had to do a lot of research for different aspects of the game. In my research a found a few tutorials on making chess in Unreal Engine. The main methods I found are listed at the bottom of the research section. The primary methods I found came from three YouTube channels and were improved by the research shown in the other areas. The YouTube channels in question are Regal City Media, Tefel Dev, and Reids Channel. With these implementations, Regal City Media and Reids Channel are rather similar to each other in implementation while Tefel Dev choses a different approach. Firstly, Tefel Dev uses an approach with boards that are made of several pieces instead of a board that is one whole piece.

Therefore, you can select each board cell individually. He also kept a very simple approach to managing chess players and pieces. While I found his approach interesting, he had made no attempt at validating chess moves to make sure players cannot make invalid moves. Therefore, I decided not to go for his implementation. However, I did use his creation of the board as inspiration for the 3-Player board as I was forced to use hexagons meshed and color coded to resemble a 3-Player Chess board.

As stated before, Reids Channel and Regal City Media held two rather similar methods and use similar functions and components. I have looked at a couple other Unreal Engine Chess formats and I have come to the conclusion that their approaches seem to be the most popular. Furthermore, Reids Channel and Regal City Media's methods were most applicable to what I was trying to achieve, and each have benefits over the other. Reids Channel creates a chess game that utilizes online multiplayer. However, there are weaknesses with his implementation. First, his implementation is far from complete. While his implementation does set up the game and allow validated selection of pieces and capturing pieces. Reids Channel does not set up any special move cases like castling, some of his functionality that I looked at was unnecessary, and he does not have any check, checkmate, or stalemate implementation.

Regal City Media is a more complete Chess game, but it does lack online multiplayer. Regal City Media utilizes a lot of methods that Reids Channel regarding changing game state, selecting pieces, and validating moves. Secondly, his networking only works in a testing phase setup as he did not make a main menu that allows players to join a game. However, I do recognize that the way he implemented networking in his game seems to be an established method within Unreal Engine as most games that I have looked at (Unreal Engine Examples and developer made games) use very similar practices. In the end, I chose to build on Reids Channel's implementation partially for setting up the board, game, and piece selection. The reason is precisely that his implementation is unfinished. I did not want to use a full implementation for my work because I feel like at that point it would be more their implementation than mine. While I am making more chess games and special variants, I wanted to make sure that most of the work is mine and did not want to rely only on another's work. There is plenty that Reids Channel did not do in his implementation which would include checking and checkmate for the king, pawn promotion, special moves like castling, and a few other important components. Therefore, in my implementation I made many changes in Reids Channel's designs and added on some ideas from Regal City Media and Tefel Dev and I can safely say that while there are a couple things that I left roughly the same, most of the math related functions I changed significantly as I used my own board assets that I paid for and made assets when necessary which changed the way certain aspects of the games worked.

You will see more in the challenges section the types of changes I had to make. Considering the different formats of chess that I have been creating, functionality had to be changed significantly to match the new board styles and in some cases piece movements had to be changed. Also, since I added android input, the player and player controller classes had to be changed to accommodate that. Also, I made many changes that you will notice in the code and in my interface for my own purposes as I noticed some things in his video that looked kind of odd so I wanted to make improvements on it. Still, I do give partial credit to Reids Channel, Tefel Dev, Regal City Media, and all of the tutorials that allowed me to speed up my work significantly. This is also why I chose to work on 5 formats of chess simultaneously partially to challenge myself.

There are other sources that I used in my implementation which is detailed below but I wanted to give some credit where credit is due. Another important component is the main menu system and

pause menu systems. I used a lot of tutorials to help with creating buttons and setting up the code for them to work. To my knowledge I did not use any code from those tutorials except for some networking related stuff with IP addresses which is specifically for one button. However, that code does not work, and I am in the process of testing. However, Matt Aspland, the YouTube channel did help significantly in my understanding of main menus and widgets in general.

The rest of the resources I used were for smaller aspects of the game. For instance, to set up Android mobile support there are plugins I had to use, and software needed to make the conversions. A tutorial allowed me to find the settings and plugins. I did use Wikipedia as some resources for chess variants. Personally, I do not care if some of the variants I created existed or not. The point was to create something new. This is also why I did not just make a 2-Player chess game with some added flavor as that would be boring and too simple. Still, credit to the rest of these resources as well as they were very helpful in understanding many things about chess, Unreal Engine, and Blueprints.

Research

1. Android Mobile Setup:
    - https://www.youtube.com/watch?v=j9lpR5DCRbg&list=PLXeOdIeEeN6jmso2HhOQISDJtwHuCC94N&index=3
2. Adding Players to A Game:
    - https://forums.unrealengine.com/development-discussion/blueprint-visual-scripting/100117-creating-and-adding-a-playable-second-player
3. Adding Mobile Input to a Game:
    - https://www.youtube.com/watch?v=XOTtvnZjgio
    - https://docs.unrealengine.com/en-US/BlueprintAPI/Input/TouchInput/index.html
    - https://www.youtube.com/watch?v=sNmMH2Dicfc
4. Setting Up a level:
    - https://docs.unrealengine.com/en-US/Basics/Levels/index.html
5. Network Replication:
    - https://docs.unrealengine.com/en-US/InteractiveExperiences/Networking/Actors/index.html
    - https://docs.unrealengine.com/en-US/InteractiveExperiences/Networking/Actors/Properties/index.html
    - https://docs.unrealengine.com/en-US/Resources/ContentExamples/Networking/1_1/index.html
6. Server Activities and Client Activities:
    - https://docs.unrealengine.com/en-US/Resources/ContentExamples/Networking/1_1/index.html
7. Main Menu Creation:
    - https://www.youtube.com/watch?v=K1vVbwMJCTQ
8. Open Pause Menu In-Game:
    - https://docs.unrealengine.com/en-US/InteractiveExperiences/UMG/HowTo/CreatePauseMenu/index.html

9. Unreal Engine Interface:
    - https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Types/Interface/index.html
    - https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Blueprints/Editor/UIComponents/Menu/index.html
10. Set Up Chess Board:
11. 2-Player Chess Rules:
    - https://www.chess.com/learn-how-to-play-chess
12. 3-Player Chess Rules:
    - https://www.chessvariants.com/hexagonal.dir/echexs.html
    - https://www.chessvariants.com/hexagonal.dir/hexagonal.html
13. 4-Player Chess Rules:
    - https://en.wikipedia.org/wiki/Four-player_chess
14. Alice Chess Rules:
    - https://www.chessvariants.com/other.dir/alice.html
15. Millennium Chess Movement:
    - https://en.wikipedia.org/wiki/Millennium_3D_chess
16. Board Grid:
    - https://www.youtube.com/watch?v=Q4AOmT9aOEM
17. Camera Movement:
    - https://docs.unrealengine.com/en-US/InteractiveExperiences/UsingCameras/CameraComponents/index.html
    - https://docs.unrealengine.com/en-US/BlueprintAPI/Photography/SetCameraMovementSpeed/index.html
    - https://forums.unrealengine.com/development-discussion/blueprint-visual-scripting/30041-rotating-character-to-pitch-of-camera
18. Player Movement:
    - https://docs.unrealengine.com/en-US/InteractiveExperiences/HowTo/CharacterMovement/index.html
19. Piece Selection:
    - https://answers.unrealengine.com/questions/263314/picking-target-in-a-turn-based-rpg.html
20. Unreal Engine Blueprints:
    - https://www.youtube.com/watch?v=EFXMW_UEDco
21. Blueprint Widgets:
    - https://www.youtube.com/watch?v=JKjH2iz2ekQ

22. Hud:
    - https://docs.unrealengine.com/en-US/Resources/ContentExamples/Blueprints_HUD/1_1/index.html
    - https://www.youtube.com/watch?v=ra-G2qEWbuQ
23. Turn Based Game Set Up:
    - https://docs.unrealengine.com/en-US/BlueprintAPI/Online/TurnBased/index.html
24. Random Helpful Research:

- http://www.teal-game.com/blog/gettingstarted1/

25. Chess Implementations:
    - https://www.youtube.com/watch?v=cY8MsPYFT1U
    - https://www.youtube.com/watch?v=3JAvihk8uy8
    - https://www.youtube.com/watch?v=1LV3-zX4wME

Hardware, Software, and Languages:

Hardware:

- Windows: 4GB Ram, Any CPU or GPU

- Android: Tests done with 1GB Ram, 500mb will likely suffice

Software: Unreal Engine 4.24.1, Android

Works/NVIDIA_Nsight_Tegra_Release_3.5.18222.5935, Visual Studio 2019

Languages: Unreal Engine Blueprints. For later development C++

Problem Requirements:

Notes:

- Requirement Importance Range: 1-5. 1-2 is completely optional and likely are only visual improvements. 3-4 is a necessary component that can be modified

or minimized if primary functionality remains intact. 5 is essential and will likely not be changed during the development cycle

- PC is player character.
- Chess Rules are according to current standards
- Given that five variants of chess will be the initial chosen formats for the game and given the difficulty and time it takes to implement one variant it is unlikely that all five will be fully implemented for a rather long time. For this project to be fully implemented, may take upwards to a year to complete in its entirety. However, at least half of the requirements should be fulfilled in the first 4-month period and the rest of the requirements should at least be written up and in testing. Furthermore, after the currently chosen chess formats requirements are finished, more chess formats will be added, thereby potentially extending the development cycle.
- Functional Requirement Type List
    - Functional: Requirement that is an entity or event that progresses the game state in some manner
    - Performance: Requirement should be optimized to account for performance drops
    - Visual: Requirement in which the allows the system to be more user friendly and pleasing to the eye

Requirements:

1. Main Menu
2. Help Menu

3. Pause Menu
4. Camera Movement
5. Player Piece Selection
6. Highlight Board Cells
7. Highlight Potential Moves
8. Player Piece Movement
9. Board-Grid Layout
10. Online Multiplayer Functionality
11. Local Multiplayer Functionality
12. Board Piece Class
13. Player Controller Class
14. Player Character Base Class
15. Special Moves
16. Check Condition
17. Checkmate Condition
18. Stalemate Condition
19. Player Hud
20. Board Themes
21. Menu Music
22. In game music
23. Options Menu
24. Xbox Controller Support
25. Board Class Base

26. Remove Player

27. End Game

1. Main Menu

Requirement Importance: 4

Requirement Type: Functional, Visual

Description: A main menu system for players to select or create their game

Rationale: This system sets up the games between the one to four players depending on the game

Fit Criterion: Links to options menu, allows for opening local game, hosting game or for joining another game

Dependencies: Save state, options menu, local multiplayer, online multiplayer

2. Help Menu

Requirement Importance: 3

Requirement Type: Functional, Visual

Description: An help menu for players

Rationale: This menu helps player understand the game controls, and the rules of all the chess formats.

Fit Criterion: Sections for controllers, 2-Player Chess, 3-Player Chess, 4-Player Chess, Alice Chess and Millennium Chess

Dependencies: Main Menu, pause menu

3. Pause Menu

Requirement Importance: 4

Requirement Type: Functional, Visual

Description: A pause menu for players and connects to options menu and allows host player to save the machine state or quit the game

Rationale: This menu allows for a player to pause the game on their machine.

Fit Criterion: Pause menu brings up menu which leads to options menu. Does not pause machine state for either player. Able to save machine state and quit to main menu

Dependencies: Options menu, Main Menu.

4. Camera Movement

Requirement Importance: 4

Requirement Type: Functional, Visual

Description: Controller Functionality to allow players to look around

Rationale: Some players like to look at chess boards from different perspectives so this gives them the option to do so

Fit Criterion: Player can look up, down, left and right from their current position and can zoom in and out.

Dependencies: Player Controller Base, Player Base

5. Player Piece Selection

Requirement Importance: 5

Requirement Type: Functional, Visual

Description: Allows the player on their turn to select one of their ally pieces or a cell where they want their already selected piece to move

Rationale: The players need to be able to select pieces they want to move. You cannot play the game otherwise

Fit Criterion: Current player can select a piece that they own. If they have already selected the piece then they can click a board cell where they want that piece to go and if the move is a valid move then the piece will move there

Dependencies: Boardstate, Highlight Board Cells, Highlight Potential Moves, Player piece Movements

6. Highlight Board Cells

Requirement Importance: 2

Requirement Type: Functional, Visual, Performance

Description: Current player, when on windows, cursor highlights the cell it is hovering on or the cell that is clicked on if it holds a valid piece. On mobile only on selection is area highlighted

Rationale: While the game can function without the visual representation, it does help the player to know what piece they have selected

Fit Criterion: Active player's cursor will highlight a cell on the board. If the active player selects a piece the cell located will be highlighted in a different color.

Dependencies: Player Controller Base, Board Base Class, Board Piece Class

7. Highlight Potential Moves

Requirement Importance: 1

Requirement Type: Functional, Visual

Description: When a player selects their piece, board cells are highlighted given by the piece movement

Rationale: Not necessary but it is helpful to see what moves are available to the player

Fit Criterion: When active player selects a piece, Cells are highlighted where the piece can move and what pieces the ally piece can take

Dependencies: Player Controller Base, Player Piece Movement, Board-Grid Layout, Player, Board Piece Class

8. Player Piece Movement

Requirement Importance: 4

Requirement Type: Functional

Description: A defined system for pieces and valid moves

Rationale: This keeps players from moving pieces incorrectly

Fit Criterion: Each type of piece has a predefined set of moves which detail the kind of moves that the piece can make

Dependencies: Player Controller Class, Board Piece Class

9. Board-Grid Layout

Requirement Importance: 4

Requirement Type: Functional

Description: This details board and piece locations and placements in the 3D space

Rationale: This is necessary for almost every other aspect of the game to function as we need a way to attach the pieces to the board.

Fit Criterion: A grid is attached to an array of pieces which use a mathematical formula to express locations in the 3D space

Dependencies: Board Class Base, Board Piece Base

10. Online Multiplayer Functionality

Requirement Importance: 4

Requirement Type: Functional, Performance, Visual

Description: The ability to play online with other people

Rationale: This is an important piece of the game, but online networking is very difficult so local multiplayer should be focused on first.

Fit Criterion: A player can join a hosting player's game

Dependencies: All

11. Local Multiplayer Functionality

Requirement Importance: 5

Requirement Type: Functional, Performance, Visual

Description: This is arguably the most important component of the game. At the bare minimum you should be able to create and simulate a game on one system.

Rationale: Without local multiplayer, we do not have a game.

Fit Criterion: A player is able to create a game that simulates multiple players by switch player indexes.

Dependencies: All

12. Board Piece Class

Requirement Importance: 4

Requirement Type: Functional, visual

Description: This sets a class which helps create movement logic for the board pieces

Rationale: We need this class to help validate movement and keep players from making invalid moves

Fit Criterion: A player is only allowed to make moves that are described in the rules of a given chess format.

Dependencies: Board Class Base, Board Piece Base

13. Player Controller Class

Requirement Importance: 4

Requirement Type: Functional, Performance, Visual

Description: A controller to attach to a player. Pairs with the player base class

Rationale: This is a necessity as we can attach certain functions to the character class such as checks for special move types or seeing if player king is in danger. We also need the controller for android mobile support

Fit Criterion: The controller allows the player to move the player and the camera, the player can select and move pieces. And the controller accepts touch input.

Dependencies: Board Class Base, Board Piece Base, Game Mode, Game State

14. Player Character Base Class

Requirement Importance: 4

Requirement Type: Functional, Visual

Description: A player character class with a camera attachment

Rationale: There is not a lot of requirements for this class other than that it exists. But we can use this to regulate camera and player movement types and speed.

Fit Criterion: Camera is mounted to player correctly and camera and player movement speed is set to an acceptable level.

Dependencies: Player Controller Class

15. Special Moves

Requirement Importance: 4

Requirement Type: Functional, Performance, Visual

Description: Each chess game has specific special cases that should be considered. One of the most well known is castling.

Rationale: Not all chess games have the same movement or special moves, to fully create the game these must be worked on as well.

Fit Criterion: Each game should be able to perform the special moves that are listed in their rules.

Dependencies: Board Class Base, Board Piece Base

16. Check Condition

Requirement Importance: 4

Requirement Type: Functional, Performance

Description: This shows that the king is in Check at a given moment.

Rationale: To have a Checkmate or Stalemate function work correctly, you must have this work first since you must see if a given king is in Check.

Fit Criterion: The function can find the king's location and will check every direction from its location to see if there are any specific enemies on those paths.

Dependencies: Board Class Base, Board Piece Base, Player Controller Class

17. Checkmate Condition

Requirement Importance: 5

Requirement Type: Functional, Performance, Visual

Description: See if a player's king is in check and if the player currently has no valid moves which would save the king.

Rationale: Checkmate is one of the most well-known aspects of chess. To end the game one player must take the other player's king so this is a necessary component.

Fit Criterion: The formula must use the Check Condition to find and see if the king is in check. If he is in check, the formula must see if the king can move out of the way of the attacking piece. If he cannot, we must then check if an ally piece can either block the path or kill the attacking piece. If this is impossible, we have checkmate.

Dependencies: Board Class Base, Board Piece Base, Player Controller

18. Stalemate Condition

Requirement Importance: 5

Requirement Type: Functional, Performance, Visual

Description: A condition if a player's King is not in Check but the player has no valid moves

Rationale: This is a condition that is basically a tie since the only way to win is to cause Checkmate

Fit Criterion: See if the given king is in Checkmate but has no pieces that can make a move that would not result in the king being in check.

Dependencies: Board Class Base, Board Piece Base, Player Controller Class

19. Player Hud

Requirement Importance: 3

Requirement Type: Functional, Visual

Description: An overlay that gives players extra information about the game and allows them to go to the pause menu

Rationale: This is a useful quality of life component as it helps get a better grasp of information in-game

Fit Criterion: An overlay that allows the player to know what piece color is tied to which player, to know who's turn it is, and a button that opens up a pause menu

Dependencies: Player Controller Class, Player Base Class, Pause Menu

20. Board Themes

Requirement Importance: 1

Requirement Type: Visual

Description: Change how boards look by theme

Rationale: This is not a necessary component, but it adds some flare to the game and entertains players.

Fit Criterion: Players will be able to access an options menu where they can change themes on boards.

Dependencies: Board Class Base, Main Menu, Pause Menu, Options Menu

21. Menu Music

Requirement Importance: 1

Requirement Type: Functional

Description: Add music to the menus

Rationale: Not a necessary component to the game but it is a nice improvement and is a simple quality of life improvement for the game.

Fit Criterion: Game music can be activated or deactivated in the Options Menu

Dependencies: Main Menu, Pause Menu, Options Menu

22. In game music

Requirement Importance: 1

Requirement Type: Functional

Description: Music that is played in the level.

Rationale: Naturally, this is not a very important piece of the game, but it can add a little flare and excitement in a chess match so this should be implemented eventually.

Fit Criterion: Music that plays in chess levels which can be activated and deactivated in the Main Menu or Pause Menu

Dependencies: Main Menu, Pause Menu, Options Menu

23. Options Menu

Requirement Importance: 2

Requirement Type: Functional, Visual

Description: A menu in game that allows players to change certain options

Rationale: While there likely will not be a need for an Options Menu initially, it would be a good idea to create one later when there are actual options to change.

Fit Criterion: An options menu on the Main Menu and Pause Menu which allows players to change board themes, and activate or deactivate either Main Menu music, or Level music

Dependencies: Main Menu, Pause Menu, In-Game Music, Menu Music, Board Themes

24. Xbox Controller Support

Requirement Importance: 4

Requirement Type: Functional

Description: Allows for players to use an Xbox one controller to play the game

Rationale: This is necessary for almost every other aspect of the game to function as we need a way to attach the pieces to the board.

Fit Criterion: A grid is attached to an array of pieces which use a mathematical formula to express locations in the 3D space

Dependencies: Board Class Base, Board Piece Base

25. Board Class Base

Requirement Importance: 4

Requirement Type: Functional, Visual

Description: An object that represents a chess board which allows piece placement, piece movement, and other important functions.

Rationale: Without this component there is nowhere to place the chess pieces or interact with the game.

Fit Criterion: The board prevents interaction of the world with any other object, pieces can only be on the board and can only move within the board. Touching a piece will highlight the cell it is on.

Dependencies: Board Piece Base, Board Grid Layout

26. Remove Player

Requirement Importance: 4

Requirement Type: Functional, Visual

Description: In a game with more than 2-players, functionality that deletes all of a player's pieces at the when a loses

Rationale: This component is necessary to end certain games as the game does not end for 3-player and 4-player chess until there is only one player remaining.

Fit Criterion: A player who's king is lost by Checkmate has all pieces deleted from the board and the game continues without them.

Dependencies: Board Piece Base, Board Grid Layout, Board Base Class, Player Controller Class, End Game

27. End Game

Requirement Importance: 4

Requirement Type: Functional, Visual

Description: An event that ends the game based on who won or lost the game

Rationale: All games need a game ending event and there are a couple of ways of ending a chess game.

Fit Criterion: When a player loses by Checkmate or stalemate in a 2-player chess format, or when only one player is left in a chess format with more than 2-players

Dependencies: Check Condition, Checkmate Condition, Stalemate Condition

Functionality Progress

Functionality Tests:

- Planning Phase: Implementation not started
- In Progress: Implementation started but not complete
- Testing Phase: Implementation finished and in testing
- Complete: Testing Yields satisfactory results
- 2: 2-Player Chess Progress
- 3: 3-Player Chess Progress
- 4: 4-Player Chess Progress
- Alice: Alice Chess Progress
- Millennium: Millennium Chess Progress

Note: Most requirements have been started in implementation but not complete enough to test adequately. Also, some notes added in functionality test for clarification. I may also give a percentage value to mark how close I believe the components are to completion. Also keep in mind that while something may be marked as Complete, this does not mean that I am necessarily done with the components. I may come back and make some minor adjustments as needed.

1. Main Menu: Testing Phase
2. Help Menu: Complete

3. Pause Menu: Complete
4. Camera Movement: Complete
5. Player Piece Selection
    a. 2: Complete
    b. 3: Complete
    c. 4: Complete
    d. Alice: Complete
    e. Millennium: Complete
6. Highlight Board Cells:
    a. 2: Complete
    b. 3: Testing Phase
    c. 4: Complete
    d. Alice: Complete
    e. Millennium: Complete
7. Highlight Potential Moves
    a. 2: Planning Phase
    b. 3: Planning Phase
    c. 4: Planning Phase
    d. Alice: Planning Phase
    e. Millennium: Planning Phase
8. Player Piece Movement:
    a. 2: Complete
    b. 3: Testing Phase

    c. 4: Complete
    d. Alice: Complete
    e. Millennium: Complete

9. Board-Grid Layout:
    a. 2: Complete
    b. 3: Complete-90%
    c. 4: Complete
    d. Alice: Complete
    e. Millennium: Complete

10. Online Multiplayer Functionality:
    a. 2: In Progress
    b. 3: In Progress
    c. 4: In Progress
    d. Alice: In Progress
    e. Millennium: In Progress

11. Local Multiplayer Functionality:
    a. 2: Testing Phase
    b. 3: Testing Phase
    c. 4: Testing Phase
    d. Alice: Testing Phase
    e. Millennium: Testing Phase

12. Board Piece Class
    a. 2: Complete

b. 3: In Progress

c. 4: Complete

d. Alice: Complete

e. Millennium: Complete

13. Player Controller Class:

a. 2: Testing Phase

b. 3: Testing Phase

c. 4: Testing Phase

d. Alice: Testing Phase

e. Millennium: Testing Phase

14. Player Character Base Class:

a. 2: Complete

b. 3: Complete

c. 4: Complete

d. Alice: Complete

e. Millennium: Complete

15. Special Moves

a. 2: Testing Phase

b. 3: In Progress

c. 4: Testing Phase

d. Alice: Testing Phase

e. Millennium: Testing Phase

16. Check Condition:

a. 2: Complete
b. 3: Planning Phase
c. 4: Testing Phase
d. Alice: Testing Phase
e. Millennium: Testing Phase

17. Checkmate Condition
    a. 2: Testing Phase
    b. 3: Planning Phase
    c. 4: Testing Phase
    d. Alice: Testing Phase
    e. Millennium: Testing Phase

18. Stalemate Condition
    a. 2: Testing Phase
    b. 3: Planning Phase
    c. 4: Testing Phase
    d. Alice: Testing Phase
    e. Millennium: Testing Phase

19. Player Hud
    a. 2: Complete
    b. 3: Testing Phase
    c. 4: Testing Phase
    d. Alice: Testing Phase

e. Millennium: Testing Phase

20. Board Themes

a. 2: Planning Phase
b. 3: Planning Phase
c. 4: Planning Phase
d. Alice: Planning Phase
e. Millennium: Planning Phase

21. Menu Music

a. 2: Planning Phase
b. 3: Planning Phase
c. 4: Planning Phase
d. Alice: Planning Phase
e. Millennium: Planning Phase

22. In game music

a. 2: Planning Phase
b. 3: Planning Phase
c. 4: Planning Phase
d. Alice: Planning Phase
e. Millennium: Planning Phase

23. Options Menu

    a. 2: Planning Phase
    b. 3: Planning Phase
    c. 4: Planning Phase
    d. Alice: Planning Phase
    e. Millennium: Planning Phase

24. Xbox Controller Support

    a. 2: Planning Phase
    b. 3: Planning Phase
    c. 4: Planning Phase
    d. Alice: Planning Phase
    e. Millennium: Planning Phase

25. Board Class Base

    a. 2: Complete
    b. 3: In Progress
    c. 4: Complete
    d. Alice: Complete
    e. Millennium: Complete

26. Remove Player

    a. 3: In Progress
    b. 4: Testing

27. End Game

    a. 2: In Progress

    b. 3: In Progress

    c. 4: In Progress

    d. Alice: In Progress

    e. Millennium: In Progress

Project Features:

Menus:

Figure 1: Main Menu

Figure 2: Help Menu

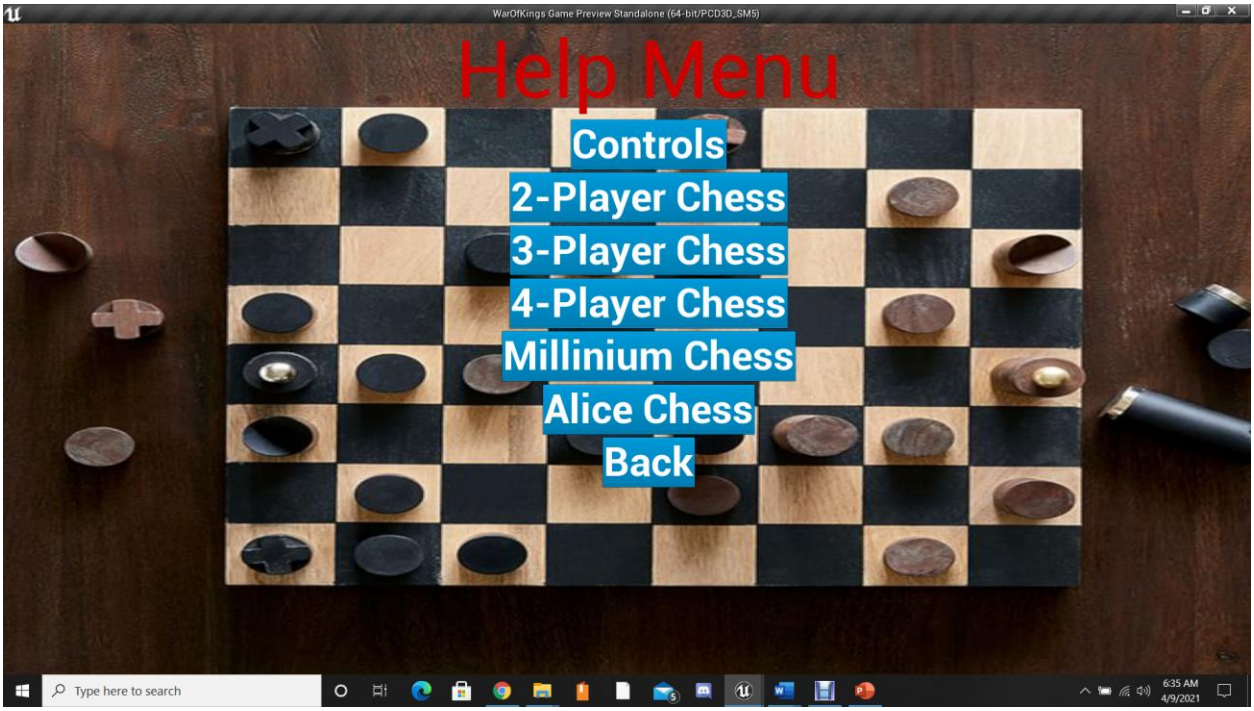


Figure 3: Controls Help

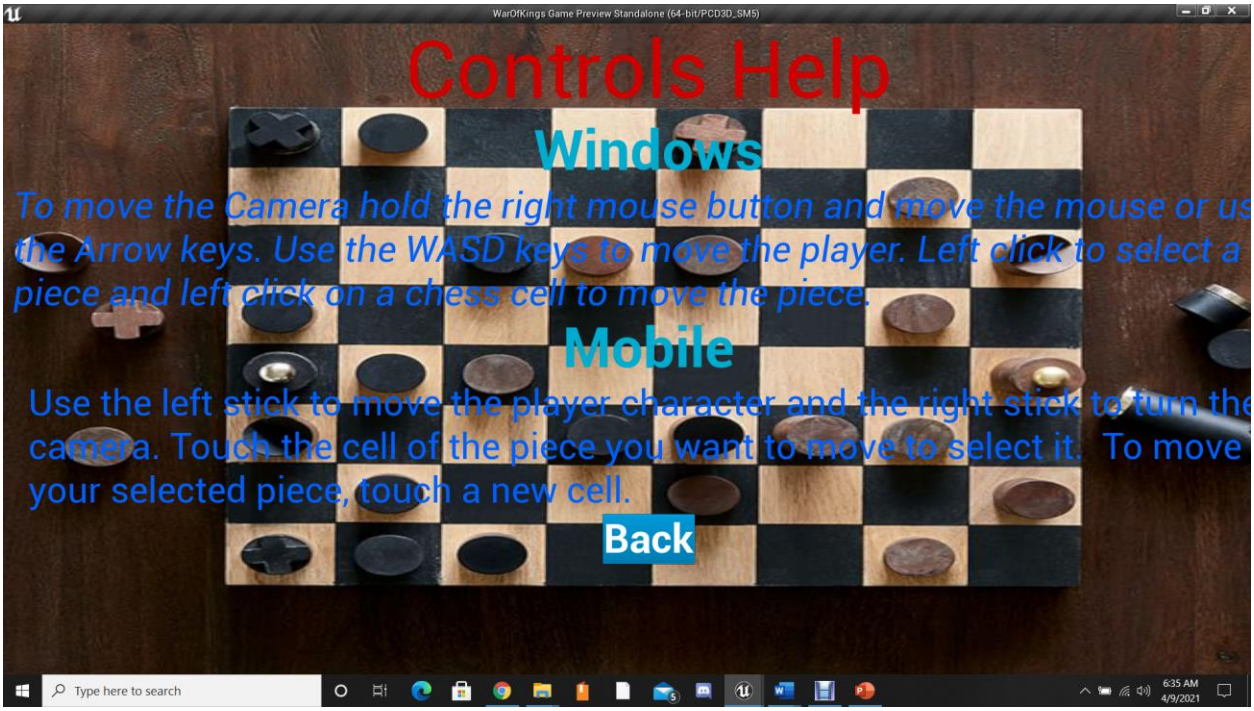


Figure 4: 2-Player Chess Rules

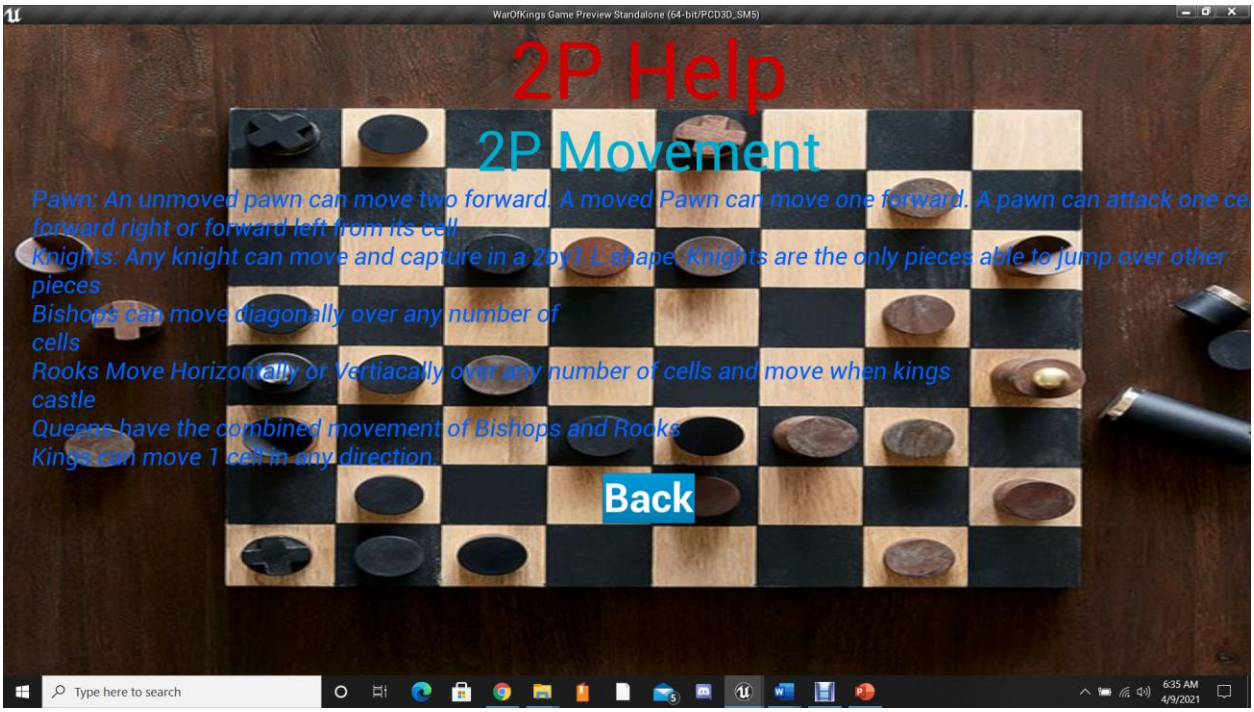


Figure 5: 3-Player Chess Rules

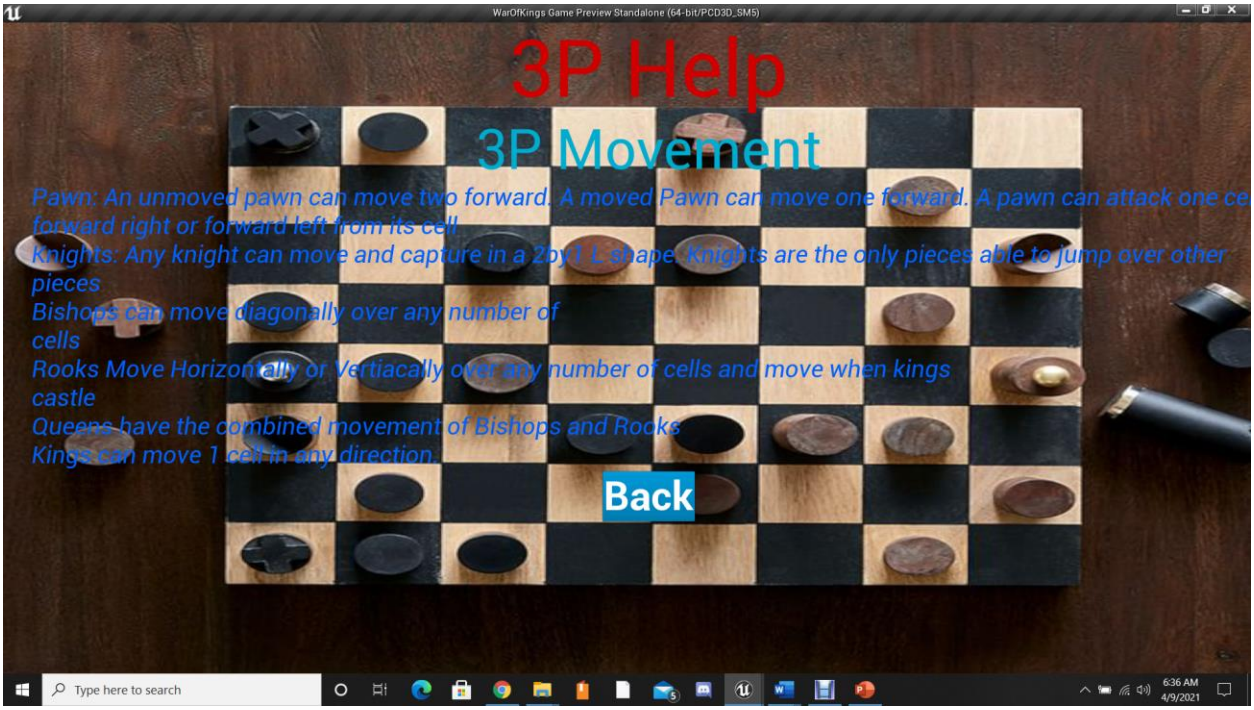


Figure 6: 4-Player Chess Rules

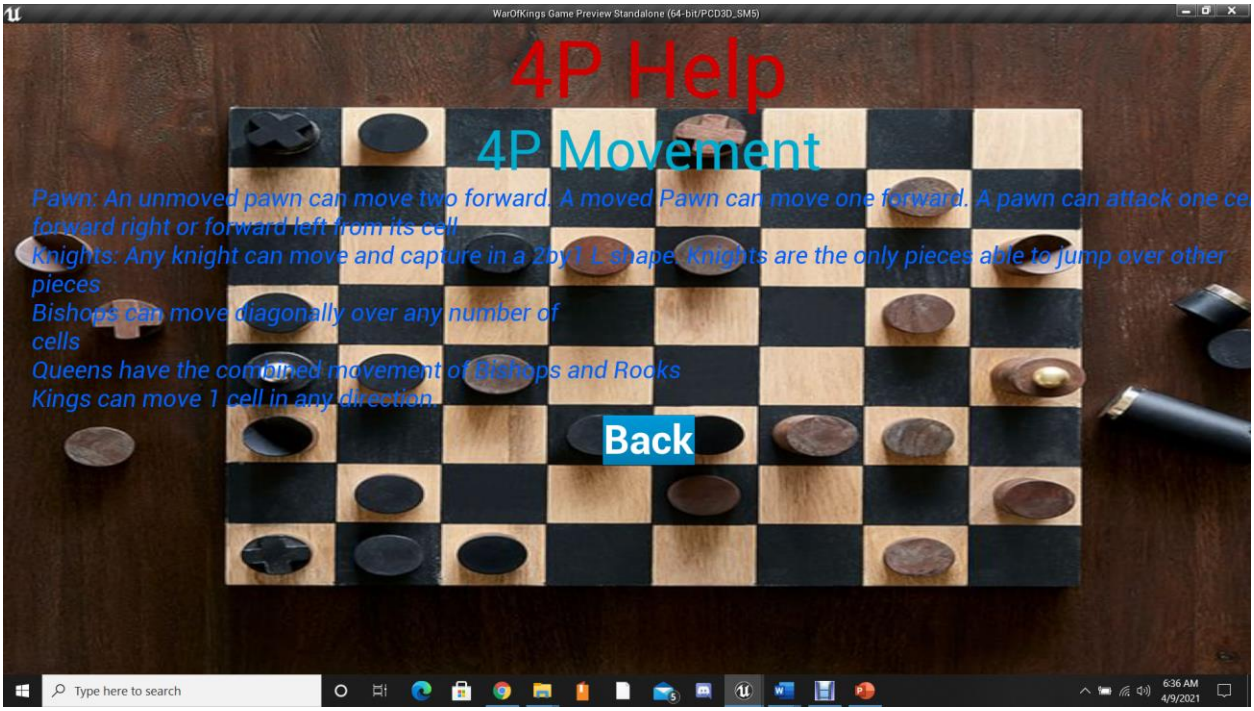


Figure 7: Millinium Chess Rules

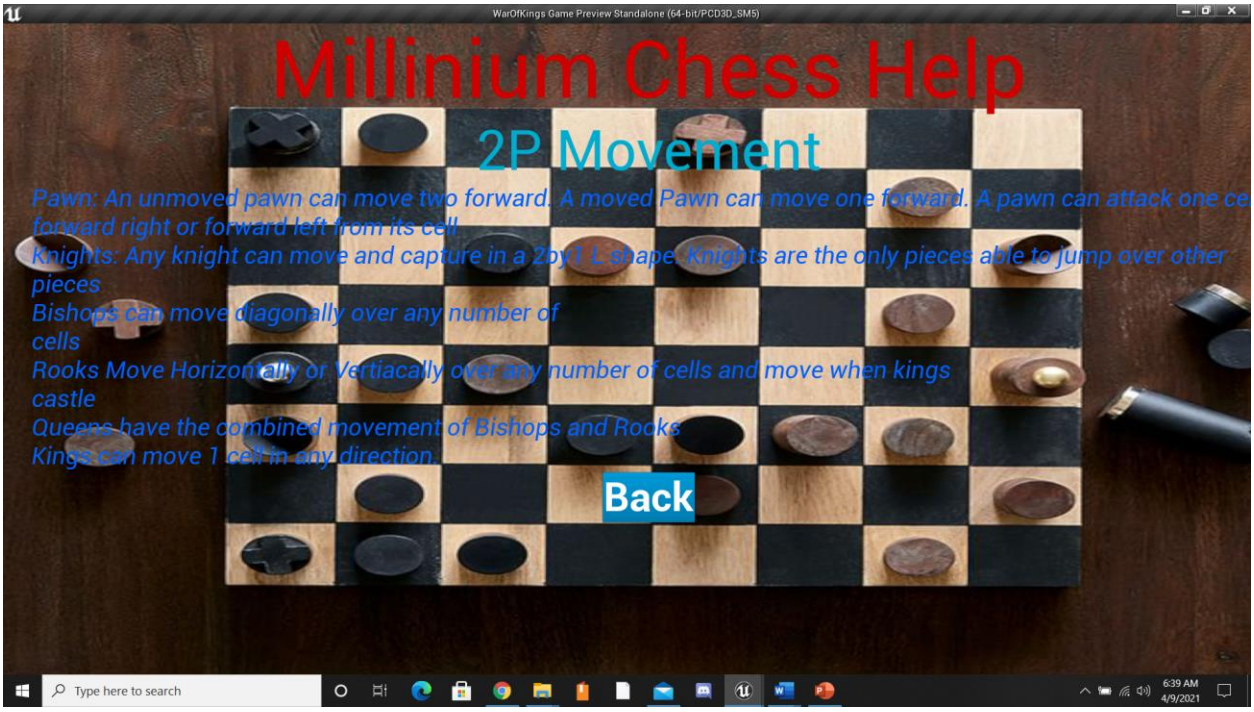


Figure 8: Alice Chess Rules

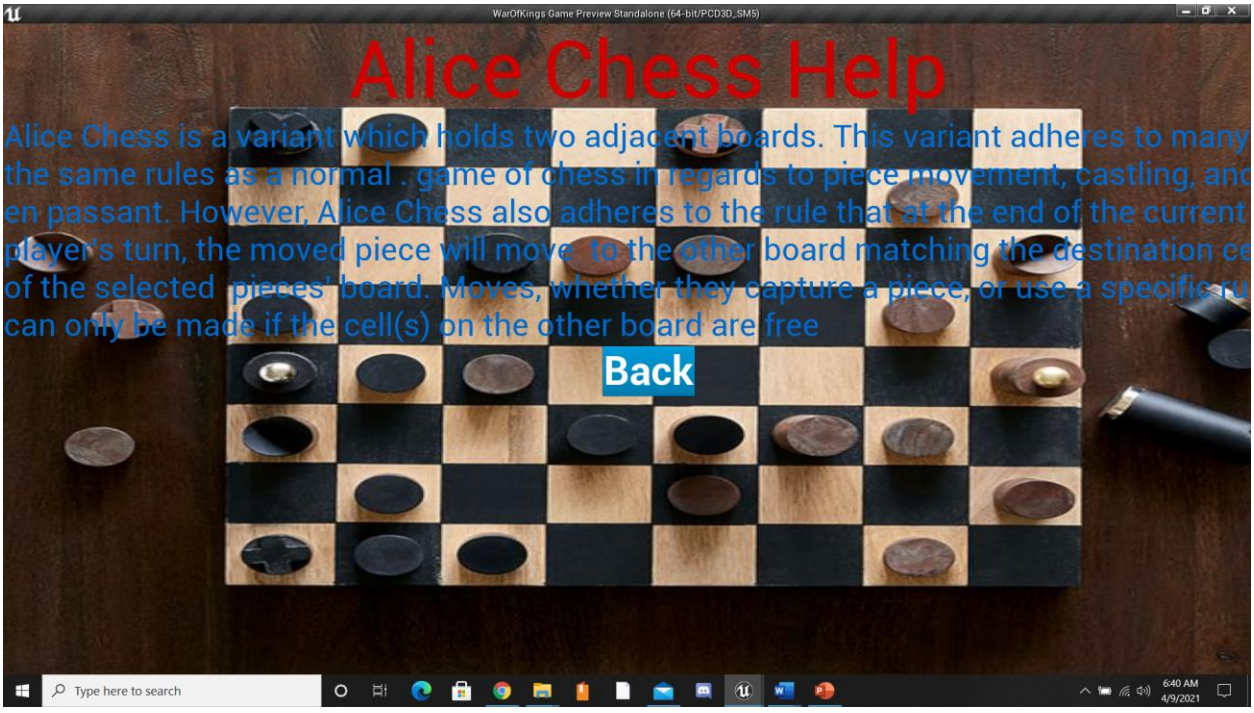


Figure 9: Pause Menu

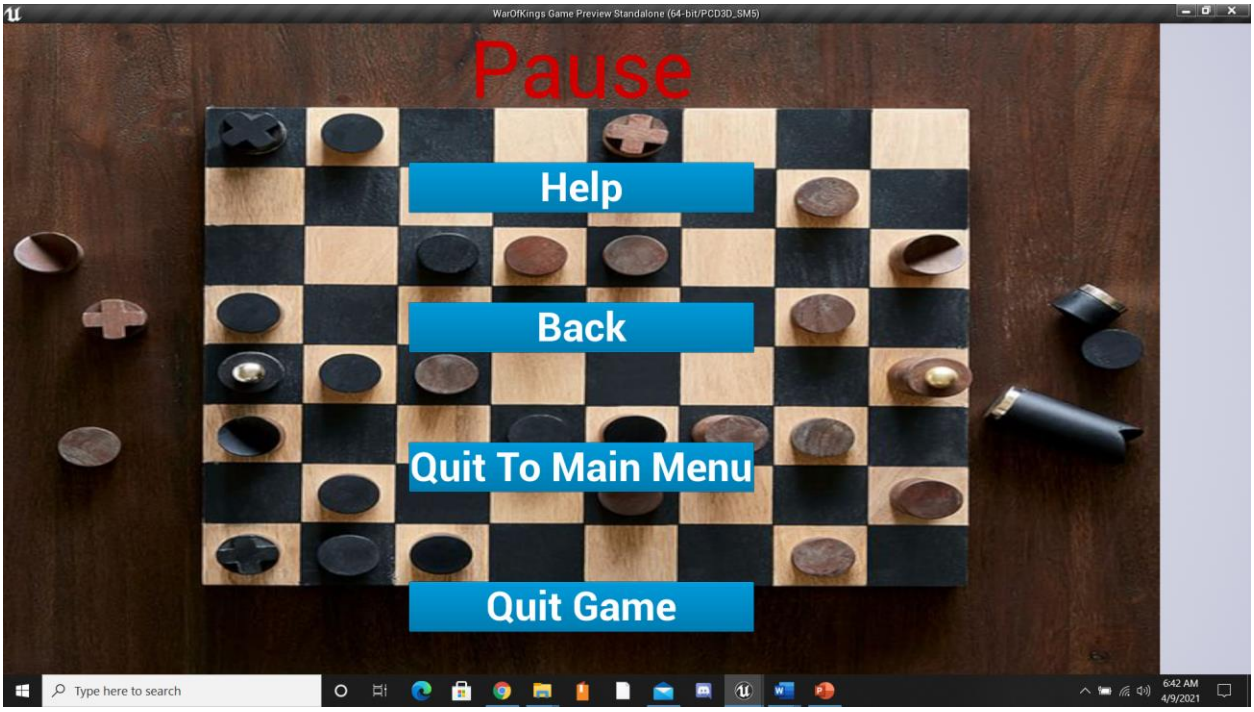


Figure 10: Promote Pawn Menu

Figure 11: Win Menu

Figure 12: Lose Menu

Figure 13: Stalemate Menu

2-Player Chess:

Figure 14: 2-Player Board

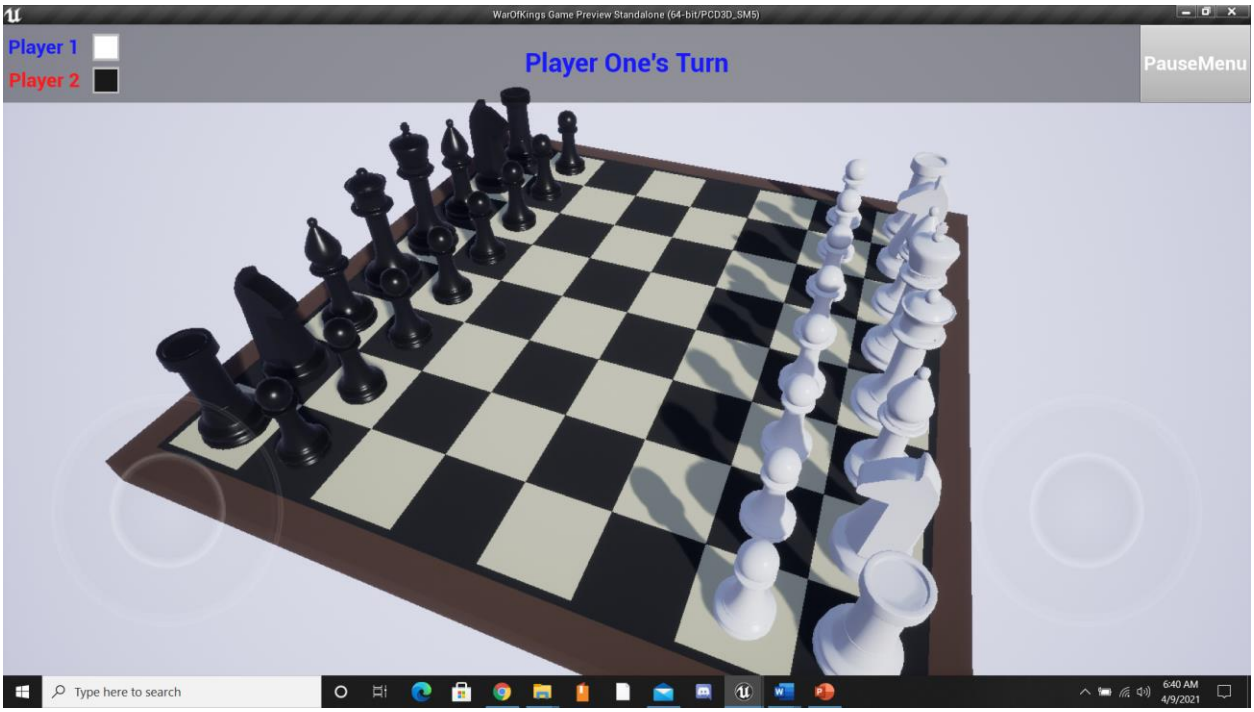


Figure 15: Cursor and touch interaction

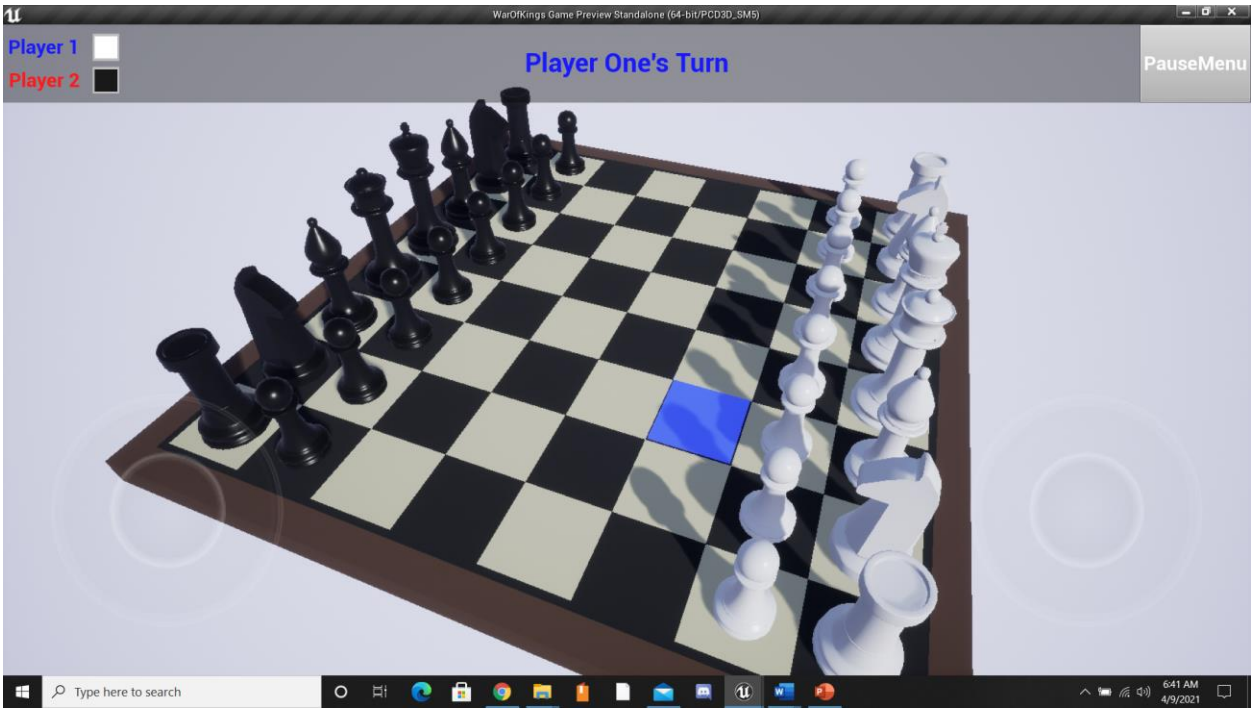


Figure 16: Select Piece

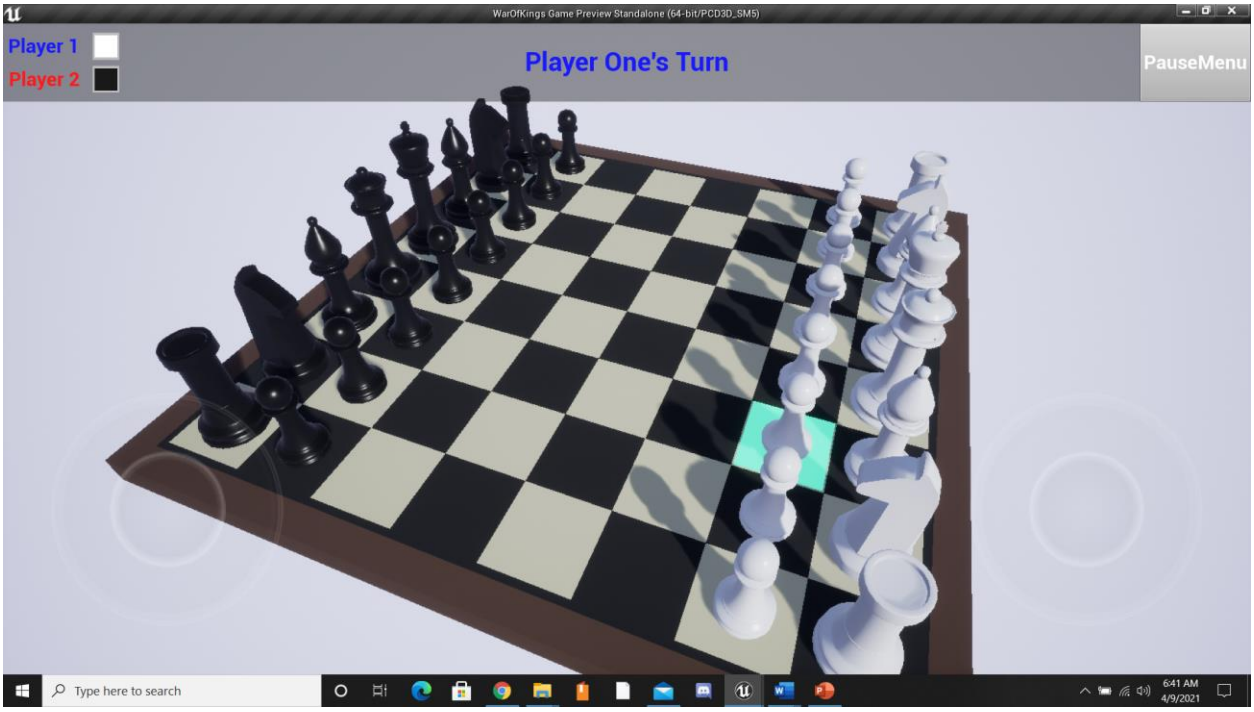


Figure 17: Move Piece

3-Player Chess:

Figure 18: 3-Player Chess networking trial run

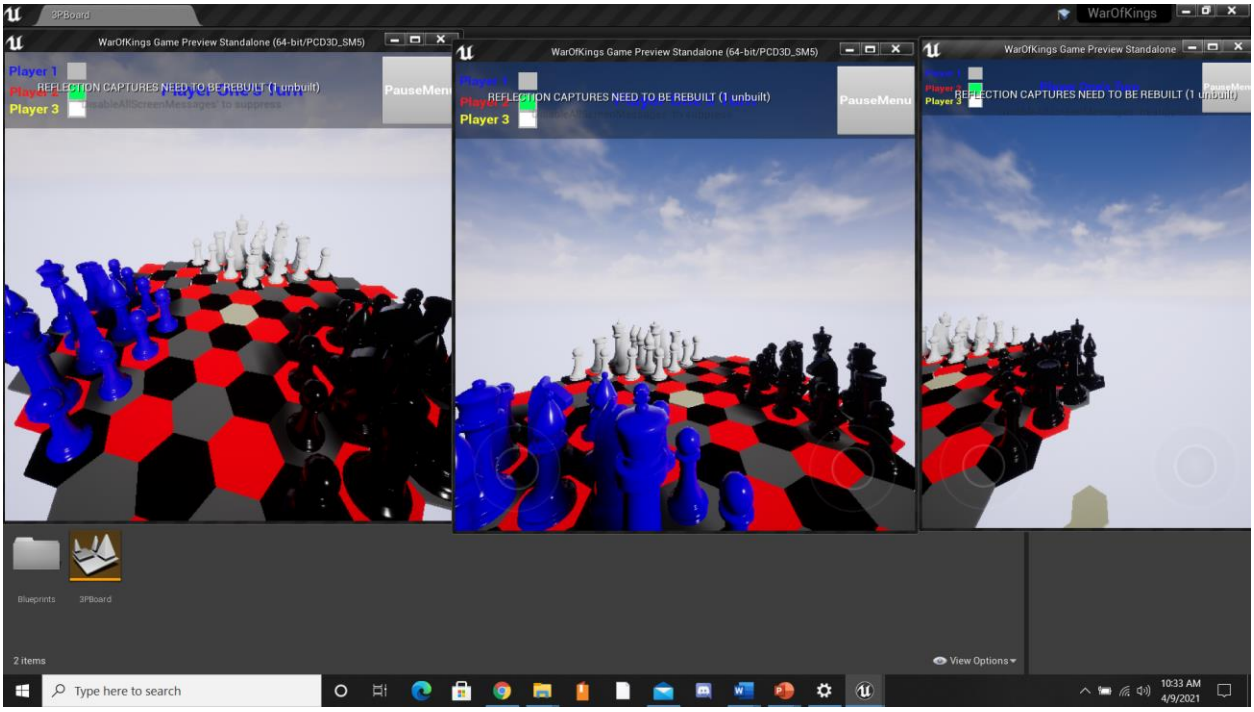
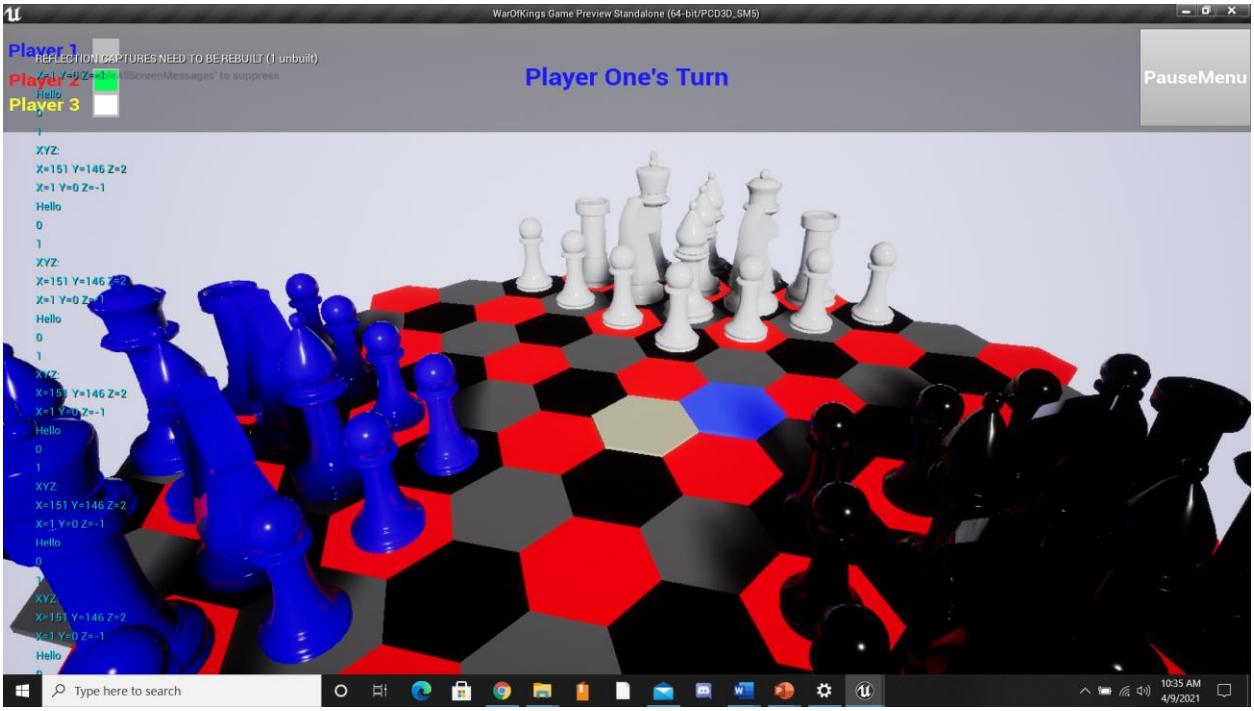


Figure 19: 3-player chess board with cursor and touch input



4-Player Chess:

Figure 20: 4-Player chess networking trial run

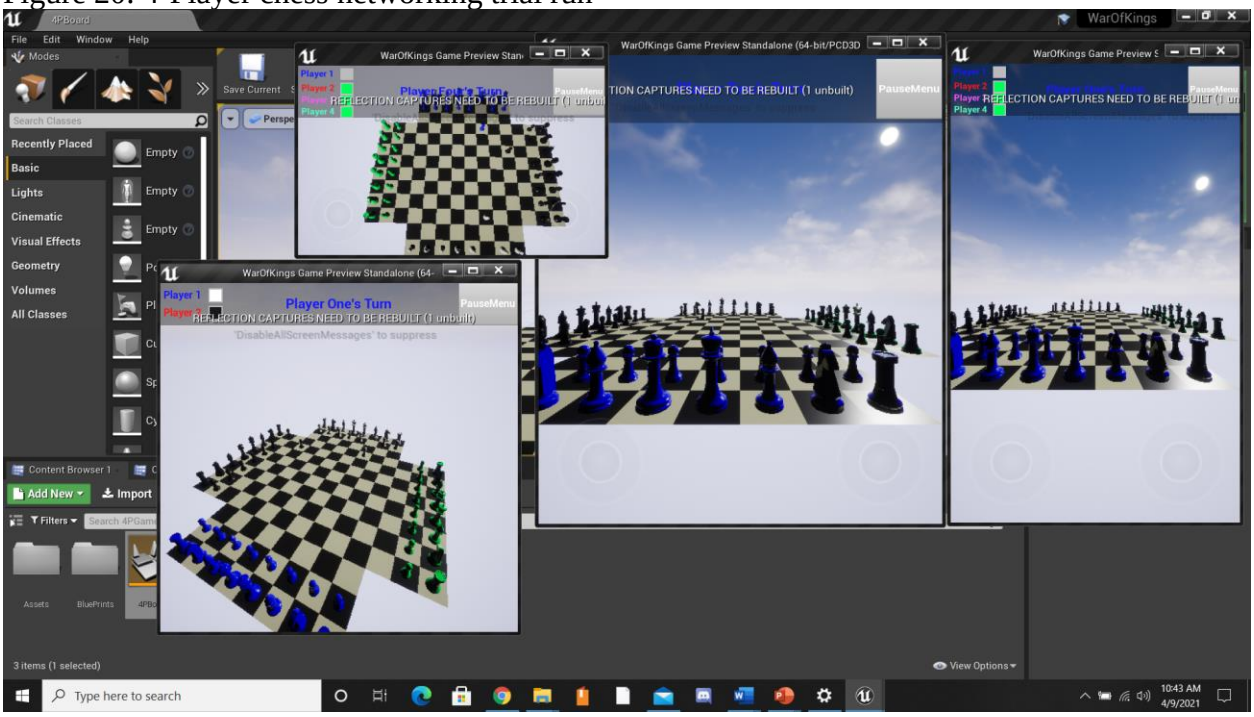


Figure 21: 4-Player Chess Board

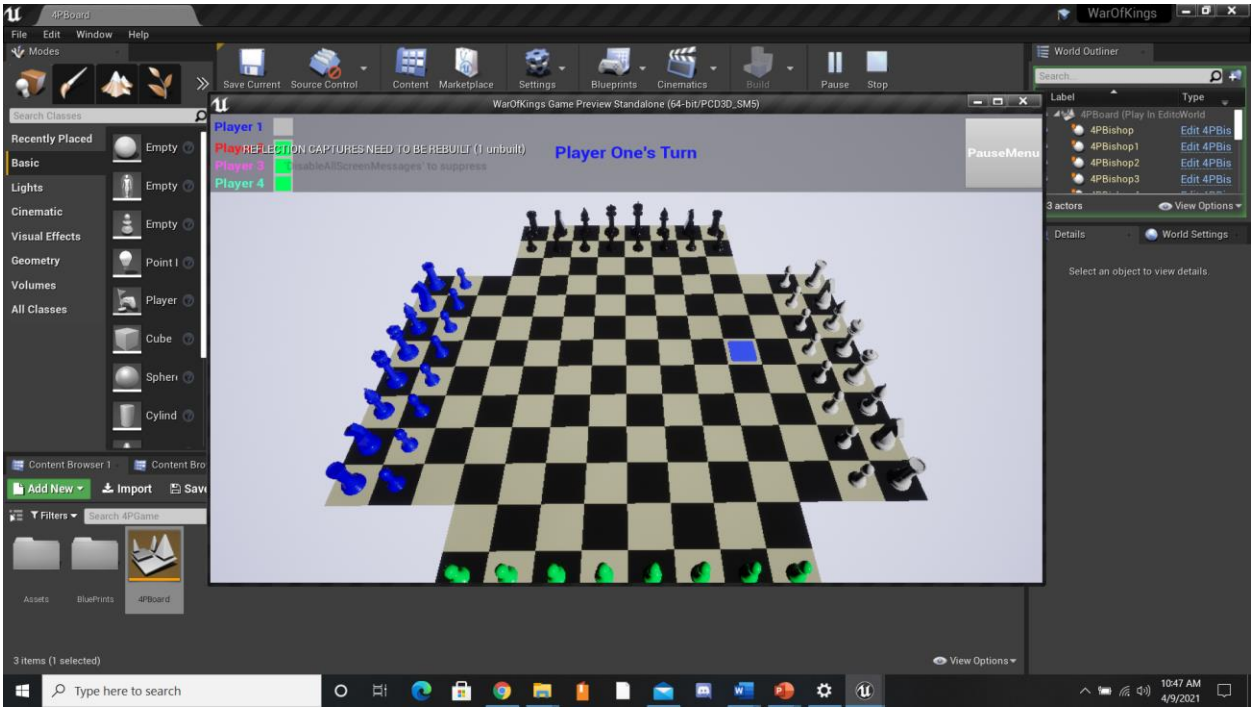


Figure 22: Black pawn prepares to capture white.

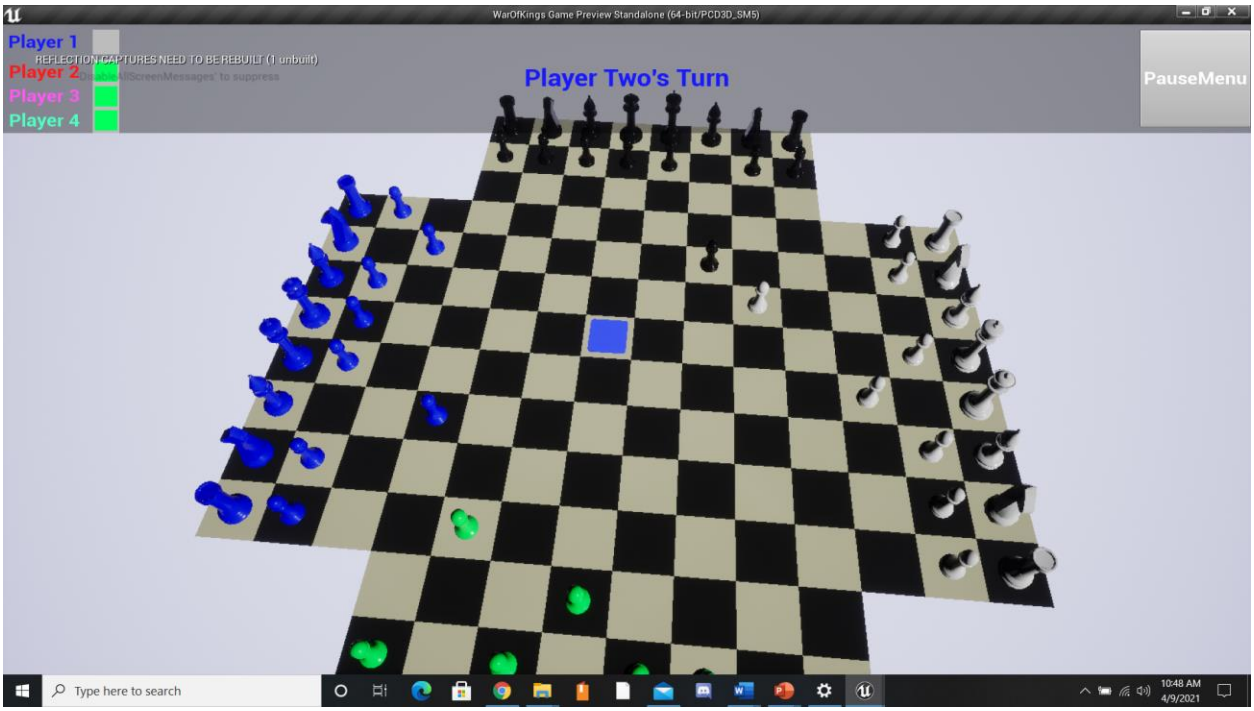
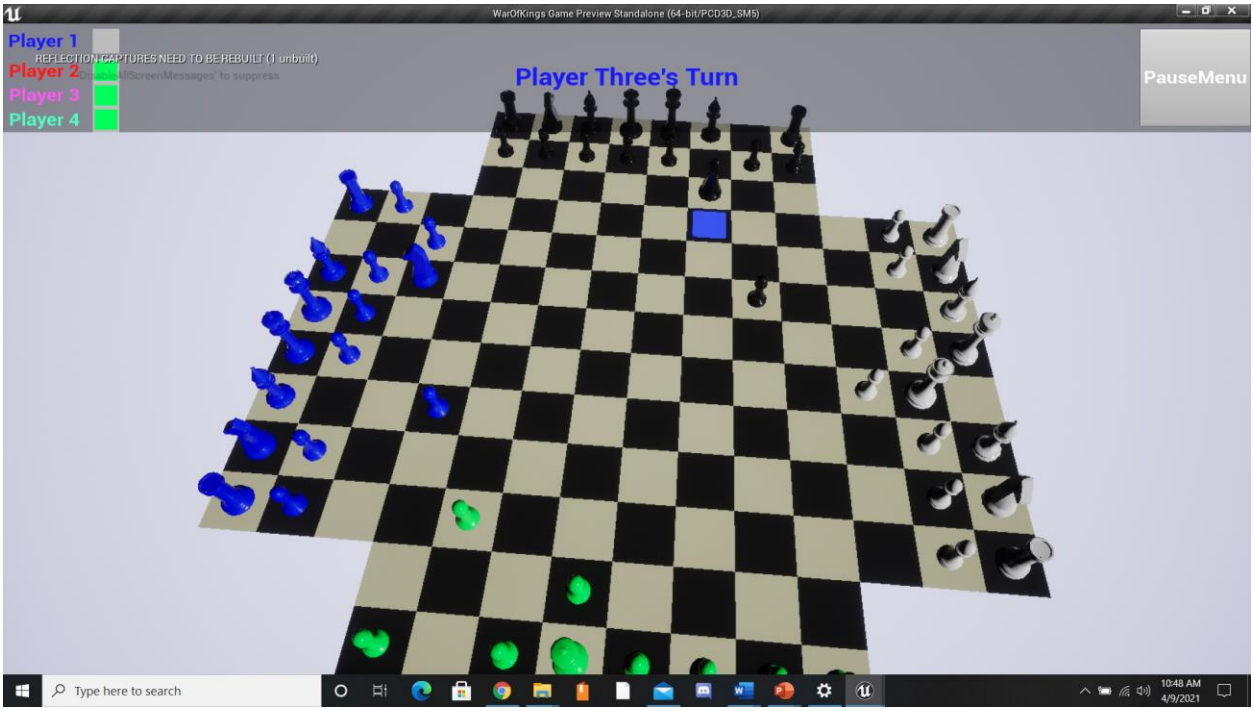


Figure 23: Black Pawn Captures White Pawn



Alice Chess:

Figure 24: Networking Trial Run

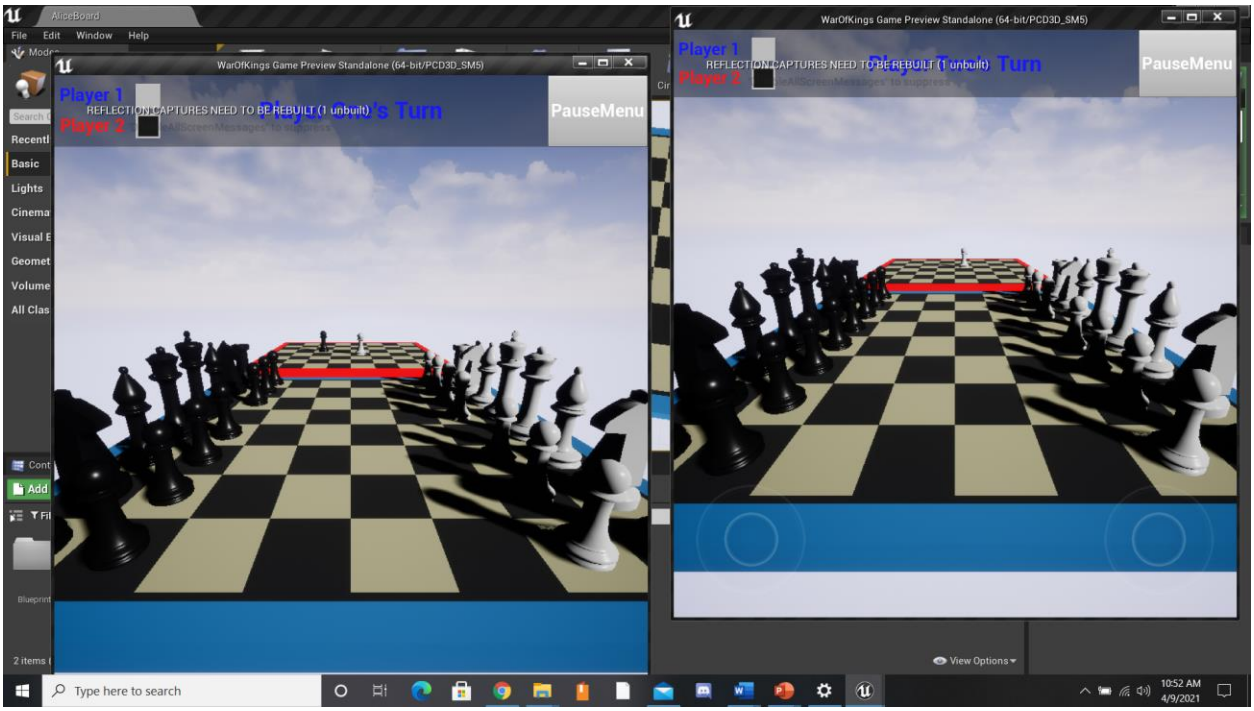


Figure 25: Piece Selection

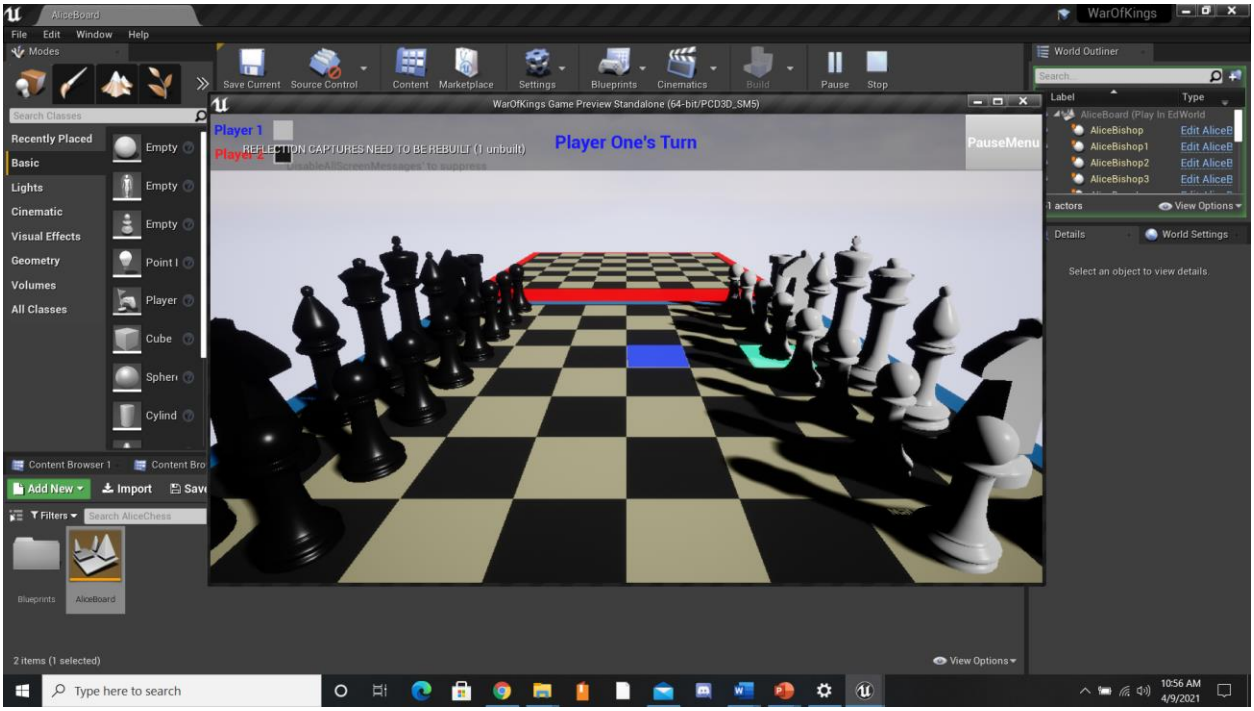


Figure 26: Piece moved

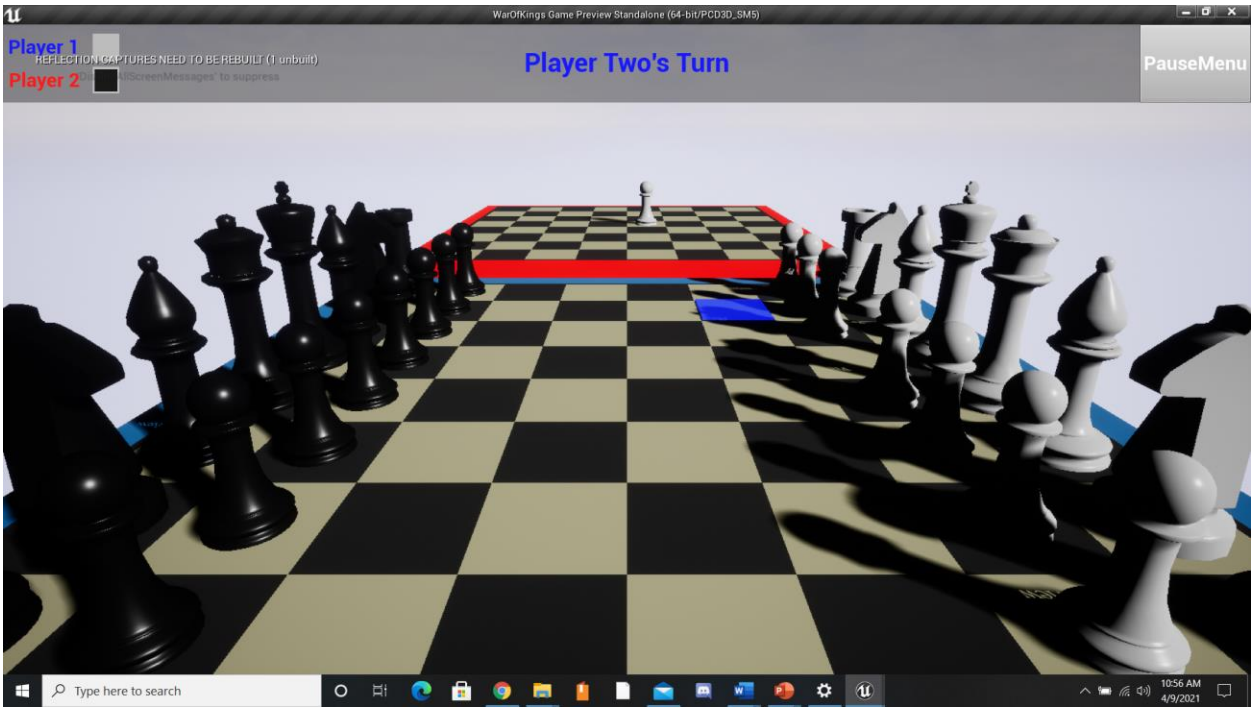


Figure 27: Alice Board

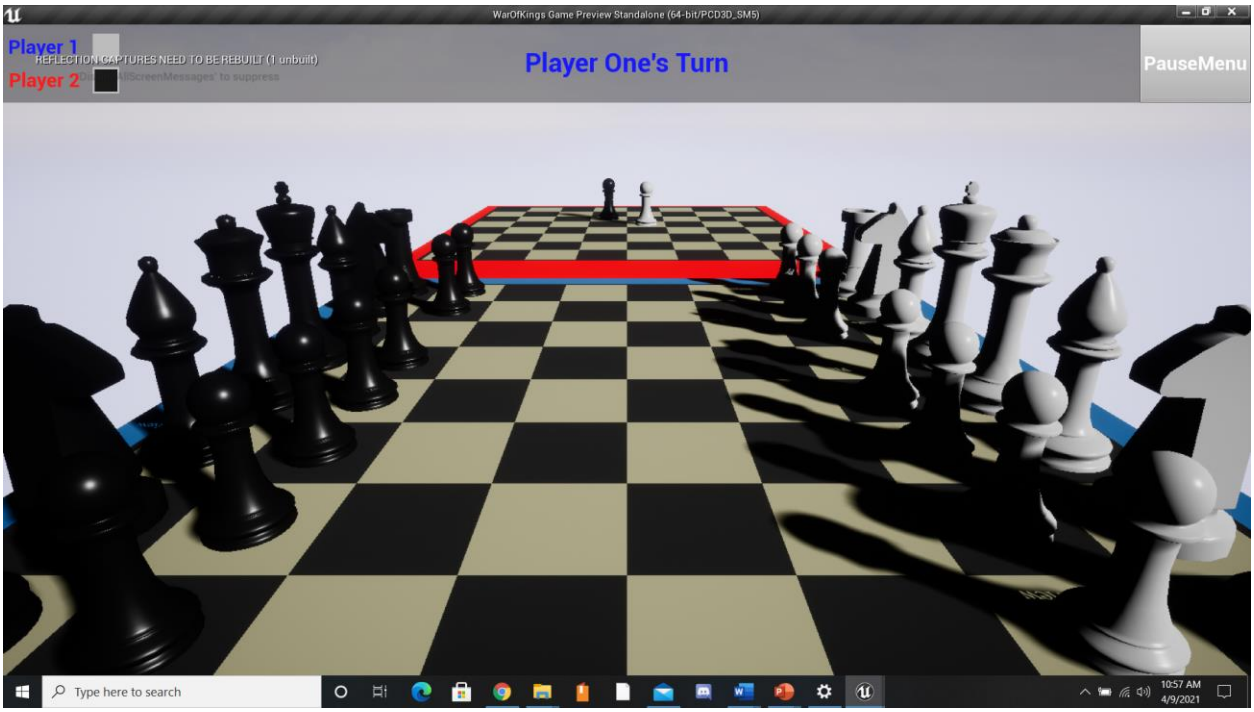


Figure 28: Piece Capture

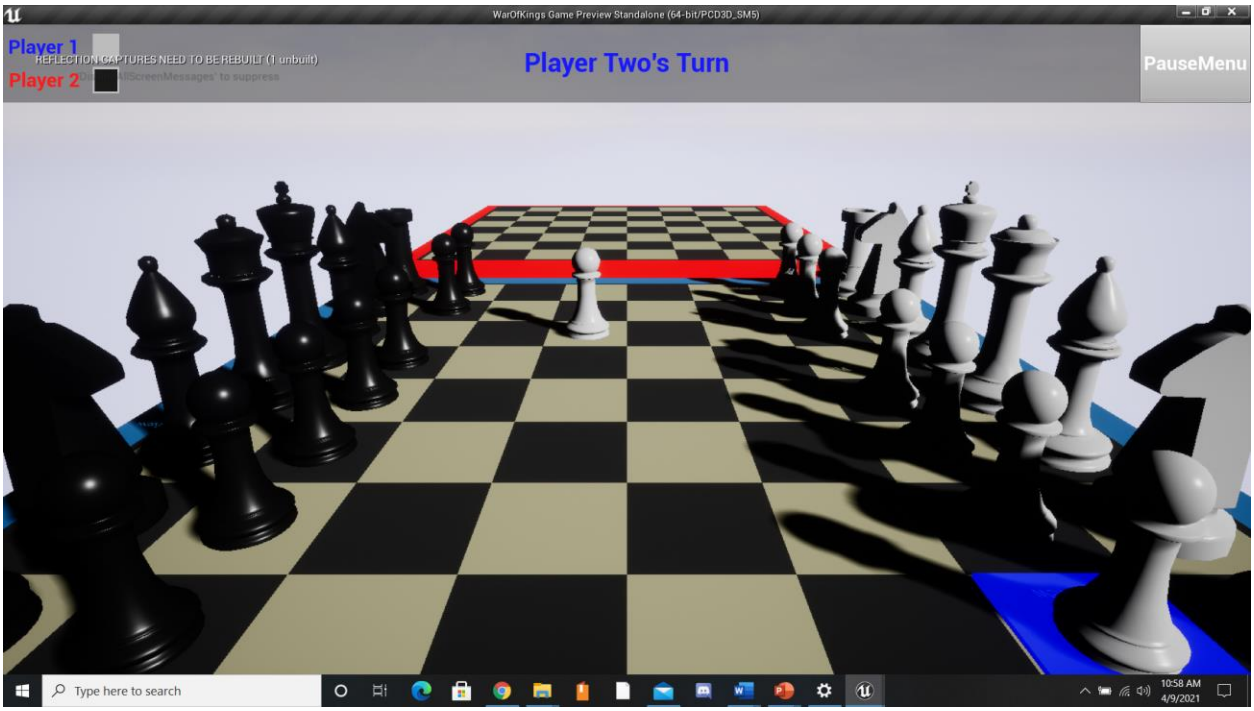
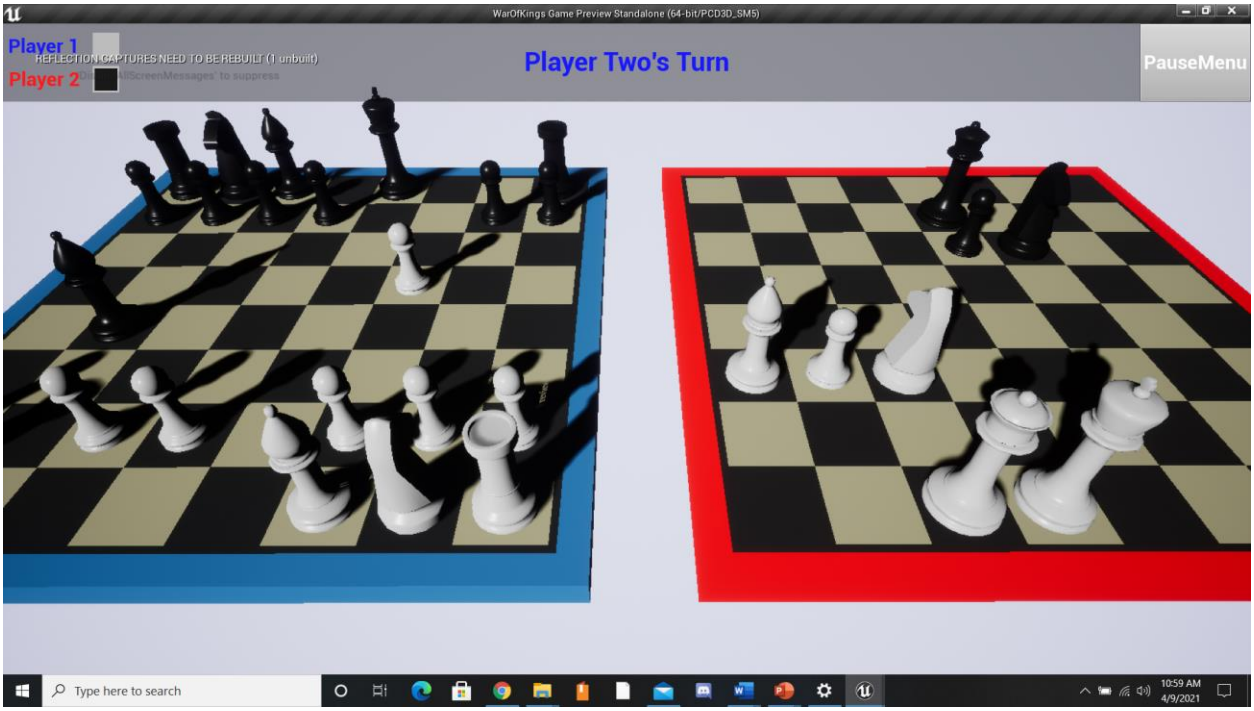


Figure 29: Move Pieces



Millennium Chess

Figure 30: Networking Trial Run

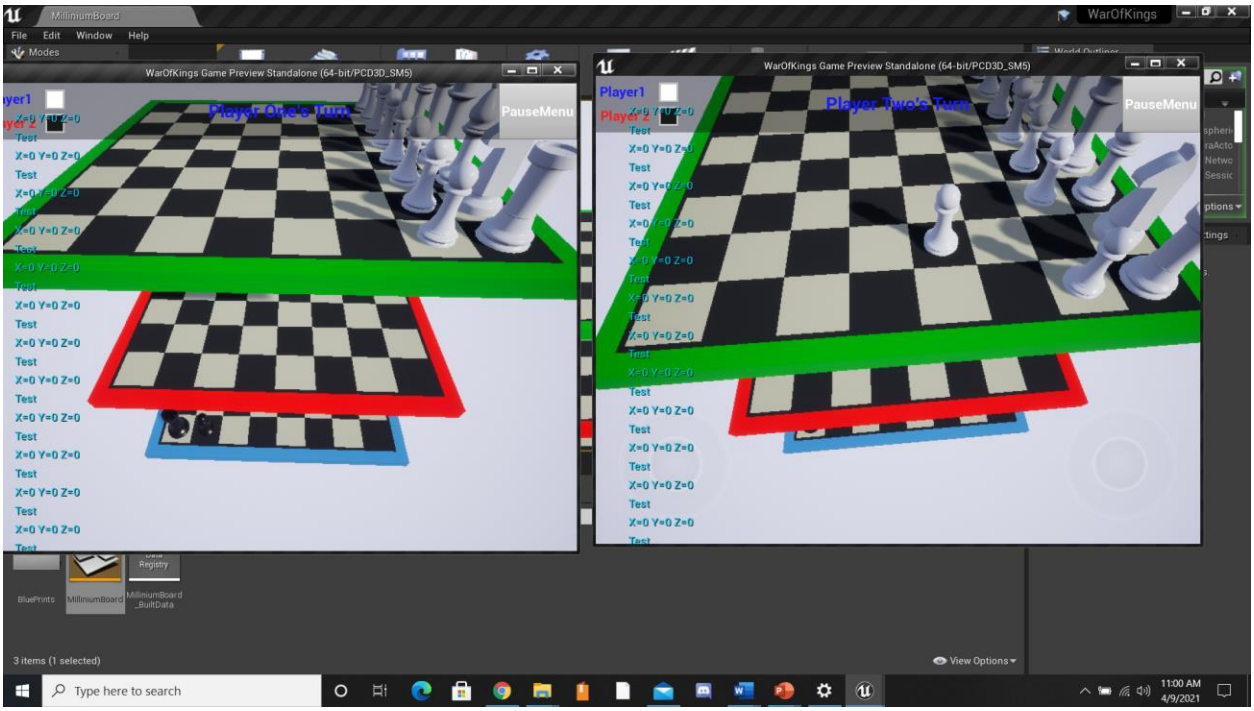


Figure 31: Piece Selection

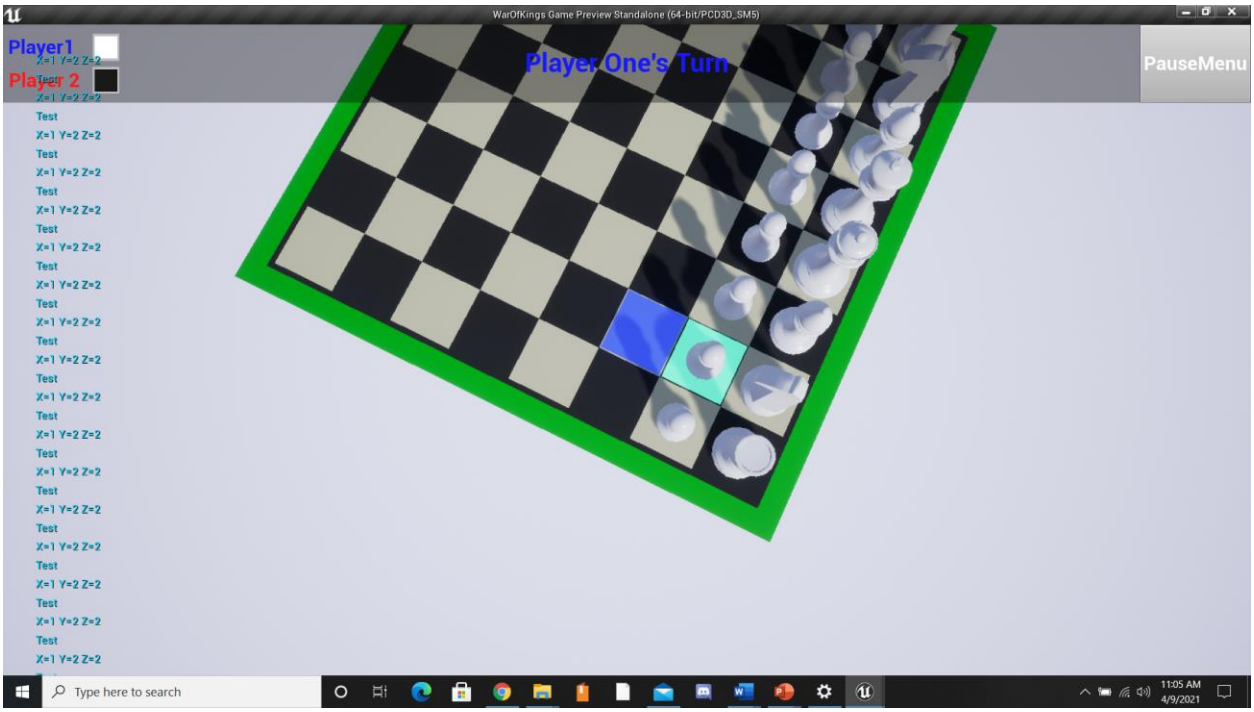
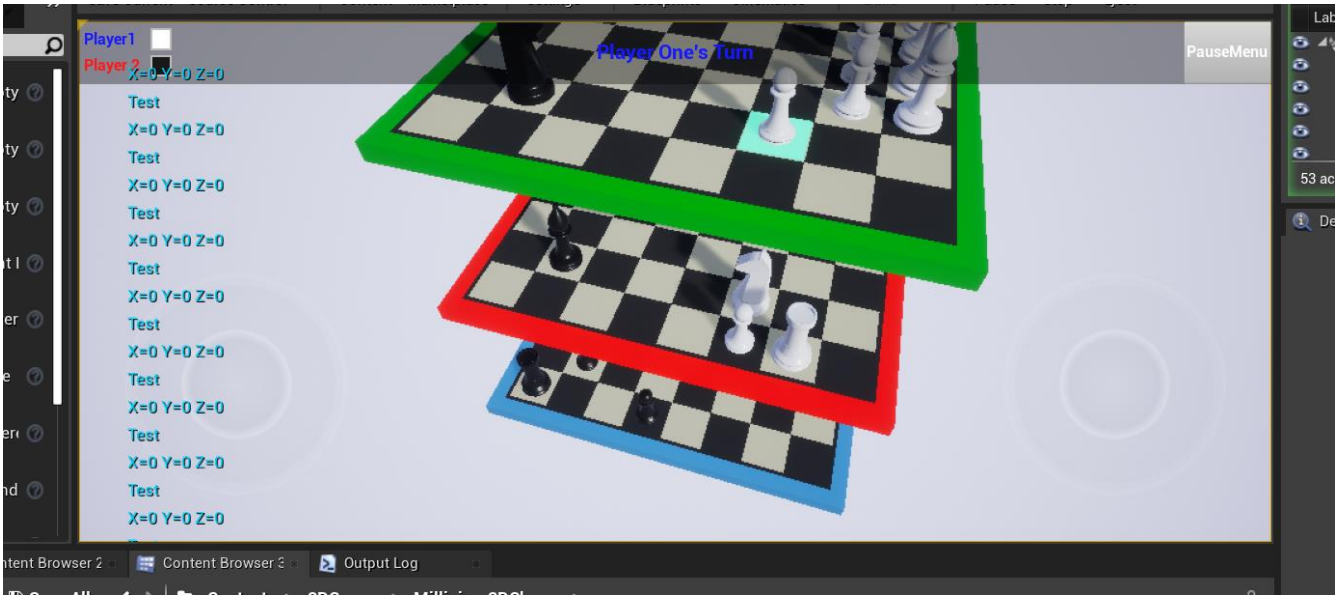


Figure 32: Millenium Board with moved pieces



Quick Explanation, Conclusion, and Links:

Earlier in development I halted focus on online multiplayer and effectively disabled what I had in my code as it was not working properly anyway. In the networking attempts you see above, multiple players are in my game. However, the appear to open in separate levels. Almost like local multiplayer. When Local multiplayer is finished, I will reactivate the necessary code and work on the networking implementations. Also, some menus have been made but not added into the base game yet.

Down Below is my project link and YouTube Link

Youtube Link

https://youtu.be/gj8RBjM4d3w

Project Github

Master Test Plan:

Introduction:

This form of testing will occur concurrently with development. Due to time constraints the project manager will handle testing. Given that in the development of this project, certain pieces rely on the functionality of previously developed components, testing must be continuous. Therefore, test phases will not be set in a linear fashion, but instead will be linked by components. Components in this case can either refer to specific functions, objects, levels, or assets. With this in mind, the goal of the system in its finality will be to have a fully functioning main menu with a way to host and join games for online multiplayer games, the ability to play a local multiplayer game on a single system, to achieve basic set ups for the chosen chess game variants which will include camera movement, piece movement, special movement according to chess variant, a hud for players with an accompanying functioning pause menu, the ability to capture opponent pieces, checkmate and stalemate for all formats, and specific rules related to the particular chess variants. The test plan will include android mobile support, windows support, and online multiplayer. However, due to technical constraints and potentially a lack of knowledge, it is likely that mobile and online multiplayer support may not be fully implemented until the other constraints have been implemented.

Constraints

- A desktop pc or laptop that has at least four gigabytes of ram.
- One Player: Unreal Engine can create testing systems that represent multiple players on one system.
- Internet Connection: For online multiplayer testing after packaging project
- The presence of free data packs with the Unreal Engine Marketplace for static meshes and animations for environments and characters
    - Purchasing cosmetic assets is allowed if necessary
- One Main Menu necessary for access to all levels and player can exit from the game
- Each player will have a HUD that shows the player colors and who's turn it is along with a button for a pause menu.
- Pause Menu and Main Menu will have access to a help section that details control for controllers and rules for all relevant chess formats.
- Players on android will have a touch controller on screen to allow them to gain a better visual of the board.
- Each game will have a game ending event on screen when the prerequisites to end the game are met for a particular chess format
- For online games, the one hosting the game will act as a listener server for the other players and those players will be clients joining the server.
- Online games will use an IP address system where the joining players will input the IP address of the hosting player.

Notes:

This game is simply a catalogue of chess games and is nothing more than an experiment. There are not many chess game implementations within Unreal Engine, let alone any games utilizing

the formats that I plan to implement. In fact, some formats I have chosen to have not been attempted to begin with. For sections that are difficult to explain with words I will provide visual representations at the bottom of the page for reference.

Test Items:

At least one Desktop PC or laptop with Windows 10 which both have the Unreal Engine 4.24 application as well as the files for the "War of Kings" game. An Android phone is required to complete mobile testing but is not required to test certain requirements.

Features to be Tested:

Note: If possible, some tests may be tested concurrently if there are no obvious in one test. Secondly, since this game holds currently 5 variants of chess, certain requirements are applied to all variants of chess.

- Main Menu System that Correctly links to the help menu and leads to the ability to extension host or join a game, make a local game on one machine, or exit the game and quit to desktop.
- Each Menu can be controlled by mouse and keyboard on windows, or touch on Android
- Applied to all formats of chess:
    - Each map will have the relevant chess board and pieces placed in correct locations
    - Players on their turn will be able to select a piece that they own and move that piece as long as the requirements for their chosen location are correct (spot is not held by an ally, spot is a valid move, etc...)
    - Players can check their opponents king by placing the correct type of chess board piece in the path of the opponent's king.
    - Stalemate in games can be achieved when a player has no valid moves but is also not currently in check.
    - Players, regardless of whether they are on Android or Windows, will be able to move their camera in a way to allow them to look at the chess board in a way that is preferential to the player.
    - Board pieces will be always within confines of the board-state unless piece is captured, in which case the piece will disappear.
    - Valid selected pieces or locations will be marked to show that they are selected.
    - Pawns when reaching a certain destination will promote to a new piece
- The current variants that will be implemented are: 2-Player, 3-Player, 4-Player, Alice, and Millennium Chess
- Boards, piece movements, and special case moves will be designed according to the rules given for the selected formats.
- For 2-Player Chess
    - En Passant and En Passant Capture
    - Castling implementation
- For3-Player Chess
    - Most conventional chess moves not applicable
- For 4-Player Chess
    - En Passant and En Passant Capture
    - Castling is allowed

- For Alice Chess
    - At end of turn, all current player pieces involved with move during turn move to other board
- For Millennium Chess
    - Piece movement is 3-dimensional
    - En Passant and En Passant Capture
    - Castling is allowed

Approach:

Level One: Testing core aspects.

- These are features that are essential for the game itself to function.
- These features will be tested by the project manager using one laptop/Desktop PC and one Android phone
- Features in this category include but are not limited to:
    - Local and Online multiplayer functionality (note that online multiplayer will take significantly longer to achieve)
    - Functionality of all menu buttons (Main Menu, Huds, and Pause Menu)
    - The ability for the current player to select a piece to move
    - The ability for players to move camera
    - The ability for players to select a valid cell on the board for the selected piece to move
    - The ability for the current player to capture pieces of opponents.
    - The ability for the current player to cause the Check event whenever the correct type of piece that the current player owns is placed on the path of cells to an opponent's king
    - The ability for current player to cause the Checkmate event against another player by putting the opponent's king in check and the targeted player is unable to get their king out of check on their turn.
    - The ability for the players to cause a Stalemate event by causing a player to no longer have any valid moves on their turn.
    -

Level Two: Testing added playability of game manually.

- These features are to increase usability for players and to make the game overall more entertaining
- These features will be tested by the project manager and may be assisted by previously chosen testers using multiple laptops/desktop pcs
- Features in this category include but are not limited to:
    - Visual improvements to maps, and menus, or assets
    - Potentially adding more variants of chess for players to choose from
    - Improving camera movement and piece selection to make transitions smoother.

Item Pass/Fail Criteria:

- Main Menu Button Functionality:
    - Pass:
        - Help Button when clicked leads to options menu.
            - Help Menu has multiple sections
            - Section 1: Controller Information
            - Section 2: 2-Player Chess Rules
            - Section 3: 3-Player Chess Rules
            - Section 4: 4-Player Chess Rules
            - Section 5: Alice Chess Rules
            - Section 6: Millennium Chess Rules
        - Quit Button when clicked exits the project
        - Join Game Button when clicked leads and when the player gives an IP address, will attempt to join the game and if successful and the correct amount of player have entered the game, the game will start
        - Host Game Button when clicked gives a menu with buttons for each game mode. Clicking the game mode will host the game for that mode (ex: 2-Player Chess)
        - Local Multiplayer Button when clicked gives a menu with buttons for each game mode. Clicking one of those buttons will lead to the mode and start a gem
- HUD:
    - Applied to players on all game formats
    - Shows the number of players and colors
    - Shows current player's turn
    - Pause menu button will be on top right of screen
- Pause Menu:
    - Has same Help Menu as the main menu
    - Has a back button to go back to the game
    - Has a quit button to exit game
- Functionality that applies to each chess game:
    - Pieces in all games:
        - We will have, King, Queen, Bishop, Knight, Rook, and Pawn. Each game will have these pieces
        - Each piece in each game will have a set type of movement
        - Pawns when reaching a certain section of the board will be promoted to a piece chosen by the player
    - All Boards:
        - Boards will be laid out on grids where the board state will be mapped to vector positions in the 3D coordinate systems.
        - The board state will also be an Array of pieces which is also mapped.
        - Every board cell will have a width and length to maintain consistency
        - 2-Player, Alice, and Millennium will use premade board assets.
        - 3-Player and 4-Player will use a mix of other assets to generate the correct board format.

- For all players:
  - Can hold the right mouse button or use arrow keys to turn screen.
  - Can use WASD keys to move the player
  - For Android mobile, the left stick can move camera position and right stick can move where the camera looks
  - Game ends if:
    - One Player cannot make any valid moves but their king is not in Check. This is known as Stalemate
    - One Player is in Check(Meaning that the opposing player has a piece that can capture the current player's king), Current player cannot defend their king and the defeated player would result in only one player remaining
  -
- During current player's turn:
  - Current player on Windows will be able to hover over the board and a section will highlight according to cursor position
  - Mobile users will not constantly highlight section but will highlight when the touch the screen on a section of the board
  - Player can then select a position
    - If the position has an ally piece, it will be selected
  - Player then can select a new position to make a move
    - Move can be made if:
      - Position is a spot that the selected piece can make according to the piece is valid (ex: pawn can move forward one)
      - Position selected is empty
      - Position selected holds an enemy piece
      - If the Move is in valid according to special rules of the variant(ex: En Passant Capture or Castling)
      - Making move does not lead to current player being in check
      - Making the move does not resolve the check condition if the player's king was put in Check before the start of their turn
  - End Game Functionality: Note that some chess games may vary and any changes to the functionality will be detailed in the relevant chess format
    - Find King: Get the index of the King that is owned by the current player
    - Get If Index is Vulnerable: Function must check for enemy Knights in specific locations around selected index. Certain paths will check for Bishops and Queens. Some paths will check for Rooks and Queens. Certain adjacent cells to the index selected will be checked for Pawns
    - Get If Index is Protected: Like the previous function but will see if ally pieces are on the selected indexes' path.
    - Get if Checkmate: See if king of current player is in check. If so, Function must check if the king can move to and adjacent cell. To check that the cell must have an unprotected enemy or empty cell. If the king cannot

move then we need to use the previous functions to see if an ally piece can block or attack the attacking piece
- Get if Stalemate: To check for stalemate we must see if the King is in check. If the King is not in check then we need to find all of the current player's pieces and see if any of them can make a valid move that does not cause that player's King to activate the Check status.

- 2-Player Functionality
  - Board:
    - An 8 x 8 board
  - Piece Movement and Capture:
    - Pawn: Can Move one forward. Can Capture diagonally forward
    - Rook: Can move and capture forward, backwards, side to side throughout board
    - Bishop: Can move and capture diagonally throughout board.
    - Knight: Can move and capture in an L shape (involves to cell movements in one direction and one cell movement in a sideways direction from that spot). Knight can jump over pieces.
    - Queen: Combines capture and movements of Bishop and Rook
    - King: Moves and captures one cell in any direction
  - Special Moves
    - Castling: Selecting king and selecting a spot two places away from an unmoved ally Rook will cause the castle to move if the King and Rook have not been moved in the game yet and the path between the two is clear. Capturing pieces are not allowed for these moves.
    - En Passant: An unmoved pawn can move two cells forward. Capturing is not allowed for this move.
    - En Passant Capture: When a player makes an En passant move and another player responds with an En Passant Move and places their pawn in front of the first player's pawn, on the next turn only, the first player's pawn may capture the second player's pawn forward even thought that is normally an invalid move. Capture is required for this move.
- 3-Player Functionality:
  - Board: A Hexagonal Board made of Hexagonal Cell that is color coded by 3 Colors. 11 rows and 11 Columns but each Row and Column has differing amounts of Elements
  - Piece Movement and Capture
    - See figures 1-6
  - Special Moves
    - 3-Player Chess has no special moves
- 4-Player Functionality:
  - Board: A 14x14 board with 3x3 corner sections invalidated.

        - See figure 7 below for more information
        - Piece Movement and Capture:
            - Movement is identical to 2-Player Chess. However, at the board corners, only knights can jump around corners
        - Special Moves
            - Castling: Selecting king and selecting a spot two places away from an unmoved ally Rook will cause the castle to move as long as the King and Rook have not been moved in the game yet and the path between the two is clear. Directions are changed for players two and four.
            - En Passant: An unmoved pawn can move two cells forward
            - En Passant Capture: Due to the size of the board, En Passant Capture is impossible.

- Alice Chess
    - Board:
        - Two 8x8 adjacent boards
    - Piece Movement:
        - Identical to 2-Player Chess movement initially
            - Players will make initially moves as one would in 2-Player Chess.
            - After the move is made, an additional move causes the piece in its position to go to the board that it is not on. (For example: Rook on (0,0) on left board moves to (7,0) on the same board. The move is made, then after Rook is moved to (7,0) on right board)
            - In addition to the rules described earlier for 2-Player Chess movement, moves are only valid if the relevant cell on the adjacent board is empty. This holds true for special moves.
            -
    - Special Moves:
        - Castling:
            - In addition to the rules stated in the previous explanations of Castling, the Rook and King involved with the move must have empty spots that they can move to on the adjacent board.
            - En Passant: In addition to the rules stated before, a spot where the Pawn would go on the adjacent board should be empty
            - En Passant Capture: In addition to the rules stated before, the spot after the capture is made, the cell on the adjacent board should be empty.
- Millennium Chess:
    - Board:
        - 8x8x3 board layout. Boards are hovering above each other.
    - Piece Movement:
        - Piece Movement is similar to 2-Player Chess but adds Z coordinates to the list of potential moves

        - See Figure 8 below for more information
    - Special Moves:
        - En Passant: In addition to the rules stated in the initial explanations of Castling detailed in 2-Player Chess, En Passant adds the option to move 2 cells down or 2 cells up as long as the move would be on the board.
        - En Passant Capture: In addition to the rules stated in the initial explanations of Castling detailed in 2-Player Chess, En Passant Capture allows the player to capture forward up or forward down in addition to directly capturing forward.
        - Castling: Functions Identical to 2-Player Chess Castling.

Test Environment:

- Constant Test Phase: One laptop with Unreal Engine 4.24. One Android Phone. One Micro-USB cable

Schedule:

- See Schedule and Test Milestones under Problem Statement and Description of Project

Responsibilities:

Project Manager: Brian Porter. Responsible for all development/documentation and testing core aspects of project

Assumptions and Dependencies:

- Assumptions:
    - The Game will always have at least one player(assuming a player decides to play against themself)
    - Machines used in testing will have at minimum 4 GB of ram on a Windows 10 operating system
- Dependencies:
    - Tests require Unreal Engine 4.24, GitHub desktop app with the accompanying project files. Visual Studio Version 1.47 is necessary for certain stages of development and compiling certain classes.

References:

- 3-Player Chess movement

Figure 1.

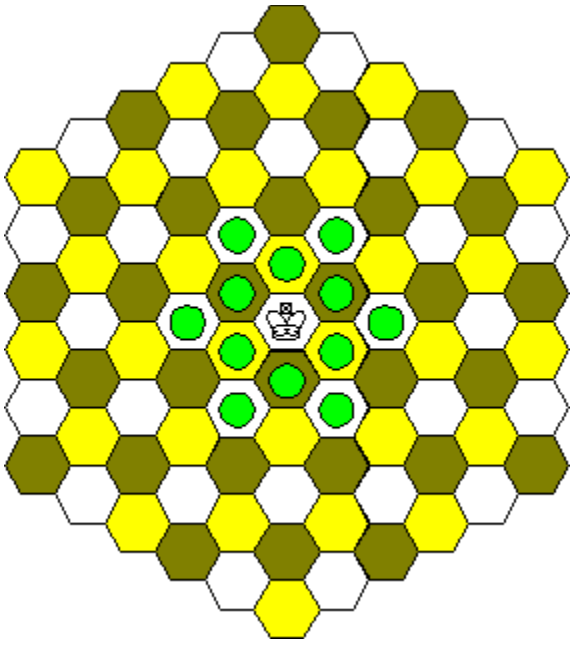


Figure 2

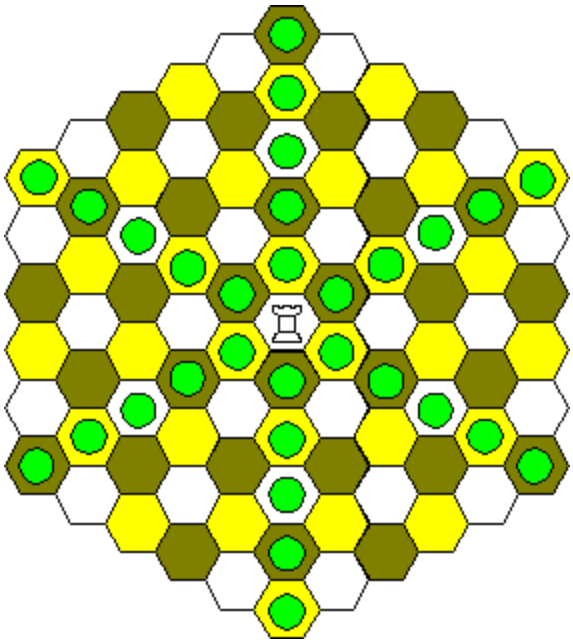


Figure 3.

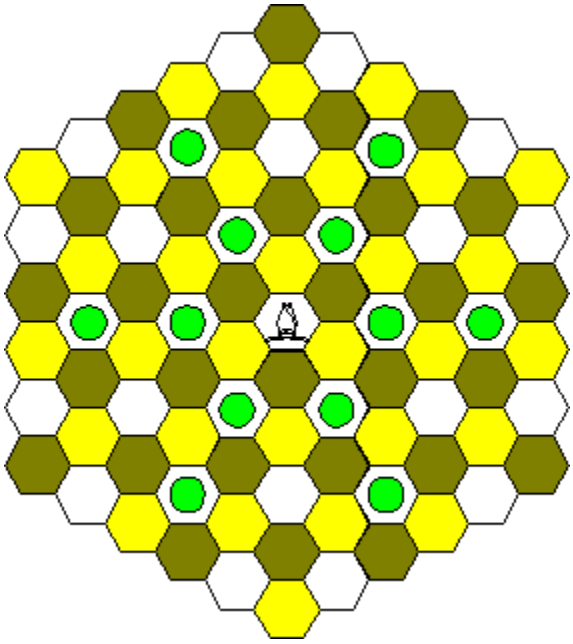


Figure 4.

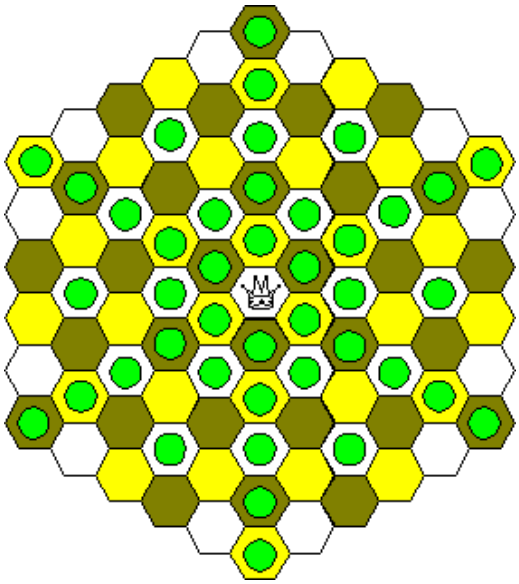


Figure 5

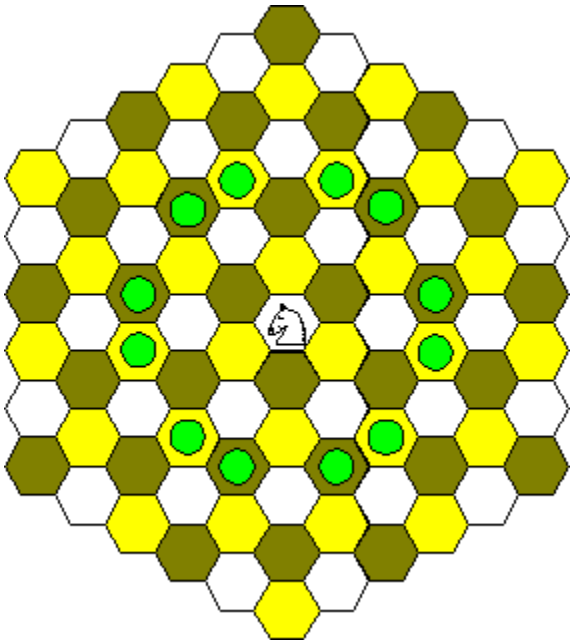
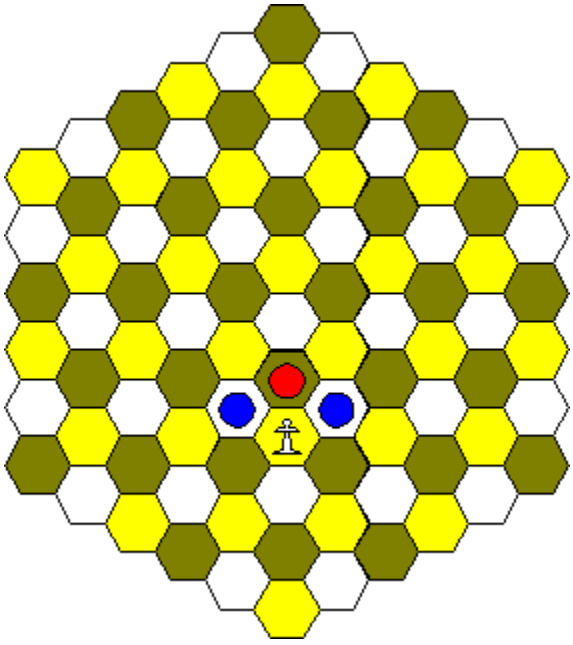
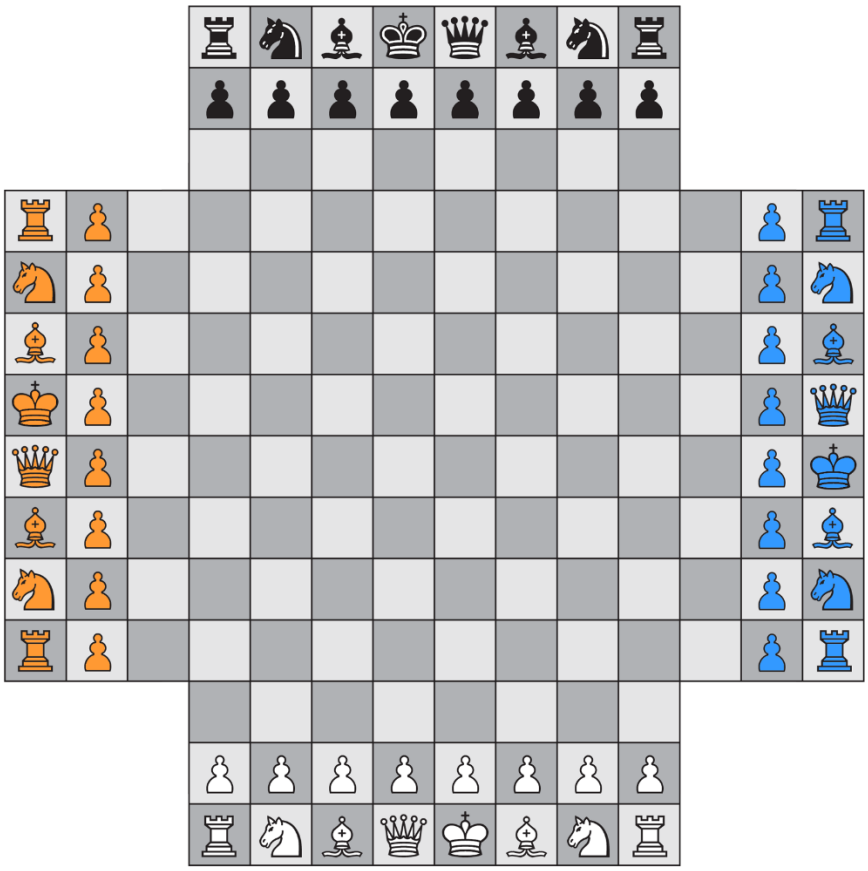


Figure 6



- 4-Player Board:

Figure 7



- Millinium Chess:
  - Figure 9
    - http://www.oocities.org/william_dagostino/millennium_3d_chess.pdf

Master Test Results:

Column Sections:

Considering that this game technically holds five separate games, each with similar rules, and as there is local and online multiplayer tests, the test results will have a couple different labels. I will also be using a rubric for simplicity. We will have several rubric categories. We have 2-Player Chess, 3-Player Chess, 4-Player Chess, Alice Chess, and Millennium Chess will be labeled as follows: "**2**", "**3**", "**4**" "**A**" "**M**". For local multiplayer and Online Multiplayer, we have "**L**" and "**O**". Tests involving Mobile Input will be labeled, "**MI**" and tests involving Mobile Support will be labeled "**MS**". If the test involves menus or HUDs, we will label the section "Widgets" or "**W**"If the test in question does not involve any of these particular sections, it will be known as "Other" and will be labeled "**OTH**"

Row Sections:

This section are the labels for the tests. This will be a numbered list as the test requirements were extensive. As stated above these tests may be repeated or overlapping in sections. For instance, camera movement applies to all formats of chess but there could be an error with it working on Millennium Chess for example. The rows will be labeled by numbers for simplicity. To label that a particular test works, we will give the cell "**P**". If the test fails in each section, then the cell is given "**F"**. Note that a failed test may also be incomplete but marked as "**F**" if the test has been attempted. If the test is irrelevant for a section, we leave the cell empty. If a test is incomplete, we give the cell "**I".** Lastly, here is the list of features as described in the test plan.

- Can hold the right mouse button or use arrow keys to turn screen.
- Can use WASD keys to move the player
- For Android mobile, the left stick can move camera position and right stick can move where the camera looks
- Help Button When Clicked Leads to Options Menu
- Sections Detailed in Test Plan are Displayed
- Quit Button on any menu on clicked exits the project
- Join Game Button when clicked leads and when the player gives an IP address, will attempt to join the game and if successful and the correct amount of player have entered the game, the game will start
- Host Game Button when clicked gives a menu with buttons for each game mode. Clicking the game mode will host the game for that mode (ex: 2-Player Chess)
- Local Multiplayer Button when clicked gives a menu with buttons for each game mode. Clicking one of those buttons will lead to the mode and start a game
- HUD is applied on all players in a game mode
- HUD shows the number of players and colors for players
- HUD shows the current player's turn
- HUD has a pause menu button on the top right of screen that can open pause menu
- Pause Menu has Help Menu
- Pause Menu has back button to go back to the game
- Pause Menu has quit button to exit the game

- Games have Kings, Queens, Bishops, Knights, Rooks, and Pawns
- Pawn Promotion
- Boards are mapped to a grid with a board-state for pieces and a 3D coordinate system
- Board Cells have set width and length
- Board uses premade board assets
- Board uses other assets of formatting for creation
- Current player on Windows will be able to hover over the board and a section will highlight according to cursor position
- Mobile users will not constantly highlight section but will highlight when the touch the screen on a section of the board
- Player can then select a position
- Player then can select a new position to make a move
- Find King:
- Get If Index is Vulnerable:
- Get If Index is Protected:
- Get if Checkmate:
- Get if Stalemate:
- Creation of Board and Placement of Pieces
- Validated Normal Moves
- Castling:
- En Passant:
- En Passant Capture

2: 2-Player Chess

3: 3-Player Chess

4: 4-Player Chess

A: Alice Chess

M: Millennium Chess

L: Local Support

O: Online Support

MI: Mobile Input

MS: Mobile Support

W: Widget

OTH: Other

Rubric:

	2	3	4	A	M	L	O	MI	MS	W	OTH
1	P	P	P	P	P	P	P				
2	P	P	P	P	P	P	P				
3	P	P	P	P	P	P	P	P			
4										P	
5										P	
6										P	
7	I	I	I	I	I	I	I			I	
8	I	I	I	I	I	I	I				
9	P	P	P	P	P	P	P			P	
10	P	P	P	P	P	P	P			P	
11	P	F	F	P	P	P	F			P	
12	P	P	P	P	P	P	P			P	
13	P	F	P	P	P					P	
14										P	
15										P	
16										P	

17	P	P	P	P	P						
18	F	I	F	F	F	I	I			I	
19	P	P	P	P	P	P	I				
20	P	P	P	P	P	P	I				
21	P			P	P						
22		P	P								
23	P	I	P	P	P	I	I				
24	P	F	P	P	P	I	I	P			
25	P	F	P	P	P	I	I	P			
26	P	F	P	P	P	I	I	P			
27	P	F	P	P	P	P	P				
28	F	I	F	F	F	I	I				
29	F	I	F	F	I	F	I				
30	F	I	F	F	I	F	I				
31	F	I	F	F	I						
32	P	P	P	P	P						
33	P	I	P	P	P	P	I				
34	F	I	F	F	F	F	I				
35	F	I	F	F	F	F	I				
36	F	I	F	F	F	F	I				

Current Thoughts:

When considering one chess game. People underestimate the complexities and the number of hidden rules that simple two player chess games have. I realized this early on given that I decided to work on 5 different formats, each with their own additions to their rules. Therefore, my main enemy was mostly time. Restarting my project from scratch in December and my overambition did not help. However, I would say I have achieved more in this project than I think anyone could have given the time and knowledge that I had. Definite partial credit to Regal City Media, Tefel Dev, Reids Channel, and Matt Aspland for help on menus, changing gamestates, creating boards, validating movement, and selecting pieces. While there were many other resources I used, I give these ones the most credit as some of my implementation come from their combined work. These channels and the other resources also helped me learn Unreal Engine Blueprints almost from scratch and I now have a good understanding of the interface itself.

Challenges Faced:

Introduction:

Over time as I design these documents, I recollect on the problems I have faced with this project. Unfortunately, problems arose from the very conception of the project. Given past faults of my own, problems I have faced in life at the time, and problems I face even now, I was forced to change my project completely in mid-December 2020 thereby making all my work for the previous two semesters mostly obsolete. I have had to struggle and fight to catch up and while it is accurate to say I am behind on my goals. I have achieved far more than anyone else would in this kind of project given the circumstances. Therefore, here I will present the challenges I have faced and overcome, and the challenges that are still in front of me. For ease of understanding I will categorize the challenges according to project components and themes. The components will be: 2-Player Chess, 3-Player Chess, 4-Player chess, Alice Chess, Millennium Chess, Main Menu, Pause Menu. Each Chess variant can be subcategorized by: Board, Pieces, Game State, Player Controller, Game Mode, Hud, Networking, Mobile Support. Sometimes I may reference the subcategories in their entirety (for instance if I were to mention Game Mode, I may be referring to more than one chess format). The themes are Unreal Engine, Input, and Project Settings. Some things may be a part of each format but does not fit in each category so I will just specifically state those.

Challenges Overcome with Solutions:

- **Chess**:
  - **Setting board with pieces in proper location**: This applies to all boards, but the solutions may vary. However, the overall solution involves holding an array of cells. And the cells will hold the X, Y, and sometimes Z coordinates. Which can be translated into a vector for a 3D coordinate on the map. Each board needs different math to make it work.
  - **Camera Movement**: I wanted to have a movement system that gives the player the option to look from many perspectives as I like looking at chess boards from multiple perspectives. One of the other issues is also using the camera with multiple methods. Fortunately, you can map movement axis to a button and the combination of buttons can provide more complicated movements. For instance, I have the Arrow keys which can provide the player with the ability to look up, down, left, or right.
  - **Player Movement**: Most chess games have no form of player movement at all and it is not necessary. However. I wanted to give players the option, and it is more useful for the 3D chess games that use larger boards. This is like Camera Movement, but you have to map settings differently. There is a component that allows for player movement input that you can use, and you can program an action that whenever a certain button is pressed, you can move along a specific axis
  - **Selecting and moving pieces**: Each chess game needs this functionality, but it changes somewhat according to shape boards. The main issue is deciding how you want to select the piece and by what kind of assets you are using. It is also good to make sure that pieces cannot make invalid moves. The main way to do

it is to have board pieces that all inherit from the same class and you can override functions. You can then give them a function that returns an array of the possible board locations that they can move. This seems to be a very popular way of achieving this method as I noticed a couple different people use a system like this.

- **Changing game state**: This part is not as difficult to achieve but it is a necessary component. From the implementations I have seen it is more of a timing thing. You can set a function to change state when a piece is selected, when the spot for it to move is selected, and to start and end the turn before and after those events.

- **Hud**:
  - **Show Player Colors**: This is supposed to mark the colors of the players. One of the issues is that binding objects with this function sometimes inexplicably does not work sometimes. But other than that. There is a way to bind colors to objects. Which is all that I really did here.
  - **Show Current Player Turn**: This a small quality of life improvement but it is nice to know which turn it is. It just gets the player turn according to current player index and outputs a string on the HUD.
  - **Pause Menu Button**: A simple button on the HUD that allows the player to open the pause menu. It is a little tricky though as you must disable the game controller and activate the menu controller temporarily. Making the menu itself wasn't hard but it does take time.
- **4P Chess**:
  - **Mapping Grid to Board:** The math to make the board is not very complex. Its in essence a large version of a 2-player board. But since the board shape is like a plus sign. I had to have extra checks which invalidate certain XY coordinates as they are not part of the board. There is an Unreal Engine blueprint that gets ranges which was useful for this.
  - **Board Piece Placement**: The piece movement is like 2-Player Chess. But finding the indexes for the piece placement took more time because of how big the board is and the number of players.
  - **Piece Movement System Adjustment**: This mostly applies to pawns as they can only move to one direction. For player 2 and player 4. If you do not change the movement system, it will be impossible for those player to move their pawns. There is a check in which details direction for pieces according to player. So, in this case I had to add left and right directions for it to work.
- **Alice Chess**:
  - **Mapping Grid to Boards**: In this case there are two boards. However, I still decided to use only one way because it was simpler. The issue though was initially my implementation was accepting input from the space in between the board even though pieces are not allowed to move there. I had to do some math to figure out how many columns should be invalidated. This also does cause some issues later but its more to add on a check to see which board pieces are on.

    - **Checking Valid Moves**: Alice Chess movement is like 2-Player chess. But there is a specific rule that I had to keep in mind. At the end of a player's turn a player's piece will move to the board that it is not currently on. However, a move can also only be made if the location of the other board is empty. For instance. If I move a rook from the bottom left position to the top left position, the top left position will need to be empty for the move to be valid.
- **Millennium Chess**:
    - **Creating a 3D grid**: Changing from a 2D grid to a 3D grid initially seems easy but there are things you have to consider later on. First, I need a significantly larger set of arrays for the coordinates and for the pieces. Secondly, I needed to space the boards out consistently so that the math on the z axis would map correctly. This is also important for placing the pieces on the middle and top map.
    - **Allowing Pieces to move along Z axis**: I learned about enumerations where you can simulate directions. This is used for all the boards. But with 3D chess with the Z component, I had to add more directions to account for up and down movement. I've seen a few implementations use a format like this and it is pretty smart. For instance, if you have UP, Forward. You can use 1, 0, or -1, to figure out piece movement.
- **Main Menu**:
    - **Opening Levels**: Opening levels is somewhat finicky. It took a little bit of time to realize that level names are case sensitive so when you try to open a level you have to give the exact name. Its not a big deal but it was somewhat of a setback.
    - **Widget Button Actions**: This was an issue early in the project construction. This is a basic thing that I did not know because I had little experience with interfaces like this and my experience with Unreal Engine was poor. But overall things like buttons in widgets have mappable actions such as "On-Hover" or "On-Touch" actions. Unreal Engine handles the contact. The programmer handles the action
- **Pause Menu**:
    - **Opening Pause Menu**: This a quality-of-life improvement for the game. I wanted to give the players the option to exit the game, go back to the menu, or look at rules to the game. It took more time making the menu itself than opening it. I made a button on the player HUDS which allow them to open the menu. It is a little tricky though because you must disable the game controller and enable the menu when you open the menu. Otherwise, you can still interact with the game.
- **Mobile Support**:
    - **Mobile Input**: Mobile input is a little weird to test. For instance, to test 2 touch controls you must have a device to test on. But for 1 touch you can mimic it with the mouse. I needed to watch a lot of tutorials to figure it out. It isn't hard to set up mobile touch support, but the controls do function differently when you go to test functionality. For instance, functions like hovering over objects do not work unless your finger is touching the screen on the object.

    - **Mobile Controller**: I wanted to allow camera and player movement, but it does not work quite the same as on windows. By using a tutorial, I figured out that there is a default controller pad. This allows me to bypass the animation side of things. But I still had to program the controls to map to the controller. Which did take some time. But this is discussed in other parts of my documentation.
- **Networking**:
    - **Replicating Board to Clients**: Replication in general is not too hard when things are static and unchanging. Since our board does not move this was more an issue of changing settings in the class than actual coding.
    - **Replicating Pieces to Clients**: By using a node called Switch on Authority, you can declare something to happen on a client, server, or both. By combining this feature with replication settings in objects allows a listener server replicate everything on the server once and leave no accidental copies.
    - **Creating Players:** This was somewhat difficult early on as I did not know much about Unreal Engine. Luckily however there is a node called “on player login” which is activated in an online game when a player joins a session. Using this allows me to start the game when the correct number of players join the game.
- Other:

Current Challenges:

    - **Time**: First, I had to restart my project Mid-December which means that all of my work and documentation became obsolete. But this needed to be done. My first project was too ambitious and simply impossible for me to do with the time that I had. Time is one of the prime factors for the other challenged you will notice below.
    - **Ambition**: Honestly, while my previous project was too ambitious. This one is as well to a lesser extent. Making Five fully functioning chess games on its own in this amount of time was already a very risky gamble but then to attempt to add mobile support along with trying to integrate networking.
    - **Special moves**: Most chess games have special moves which take extra checks to function correctly. There will be more details in the chess formats listed below.
    - **Checking Kings**: Every game needs to be able to recognize that a king is in danger. This is difficult since we have to check every direction and to make sure that the direction of the attack is caused by the right kind of piece. The algorithm I made basically checks every cell that the king could be attacked from and if the piece on that path is the correct type (for instance and adjacent Bishop cannot attack a king).
    - **Checkmate Kings**: Every game needs to be able to checkmate a player. This is difficult since there are a lot of things that need to be checked. The direction of the attack, whether the king can move out of the way, or whether an ally piece

can attack the attacking piece or block its path. Of course, this relies on showing that the king is in check. You also then need to check if the king can move which involves having a function that checks of a particular index is vulnerable to attack. By using that function you can check each adjacent cell to see if the king can move out of the way. If not, you need to see if any ally pieces can block the path of the attacking piece or attack the piece. That also utilizes the previous function. So, if the player can block or kill the piece, the king is safe. If not, it is checkmate.

- **Stalemate Kings**: Stalemate is also a difficult thing to achieve as you have to make sure the king is not in check and none of the pieces are able to make valid moves. What I did here was to get the potential moves of all ally pieces and put it into an array. However, there is an issue. I had to mark certain pieces as immovable because their movement would cause the king to be in check so they cannot move. Those piece movements were not added into the array. After we get the array we try to find out if the array is basically empty. If it is, then the king is in stalemate.
- **Player Removal**: This is only applicable to boards with more than 2-players, but we need to make sure that when a player loses its king, that we get rid of the pieces that the player owned and effectively remove the player from the game. This part was not too hard to make but it is necessary. All I must do is get the player index who was eliminated and find all their pieces and remove them from the board. Not too much of an issue there.
- **Pawn Promotion**: This is applicable to all the games and we need to be able to switch out a pawn that has moved to the correct location with a piece that the current player has selected. This is not too hard. But the basic understanding is to pop up a menu when a player pawn reaches a certain spot on the board. The menu will have buttons to click which will allow the player to change the piece to a new piece.
- **En Passant Capture**: En Passant Capture is a special move in chess where whenever a pawn uses an En Passant move (moving 2 forward as their first move) and the opponent responds immediately by moving their pawn En Passsant style in front of the first pawn. The first player can attack forward and take that pawn. But only immediately after the other player made that specific move. Right now I have implementations for 2-Player, 4-Player, Millennium, and Alice Chess. 2-Player Chess implementation is the most complete, but they do not work quite yet. This is partially due to difficulty and lack of time.
- **Castling**: Castling is somewhat difficult to check but it is applicable in all but the 3-Player chess mode and changes somewhat according to the mode. I implemented the check for this move in the player controller. There is a function that can move a piece by setting an array index. After it is shown that the player is trying to castle their king and their attempt to do so is under the correct conditions (king and rook has not been moved and there are no pieces in between the king and the rook) then I use the function to first move the king

and then move the rook. Each Chess board has most of this function written up but I am still testing the function.

- **3P Chess**:
  - **Mapping the board to a grid**: This was rather difficult. Arguably this is the most difficult format for me to make which is why I am so far behind on it in comparison. I have had to reset this a couple times as I have had issues with other aspects of the chess format. The method to create the board is different then the rest. I have a hexagonal asset. I also have an array. Now to map the array I have to have a separate formula for each row as each row as a different number of elements and must not only be mapped according to the array but also to a vector coordinate in the map itself. You will notice in the 3PBoard there is code in the construction section. This is the code that builds the board itself.
  - **Placing Pieces on the map**: Placing pieces on the board takes more advanced math than it does for the other grid formats. The problem is that each row is offset slightly due to the hexagonal shape of the board. Luckily, by having an array of pieces called board state, I can place the pieces where I need to. The main issue though is finding the right index for the piece to be placed.
  - **Hovering or touching a board cell and having the correct cell be marked**: This is where 3-Player chess gets rather tricky. Right now, this section is not complete and when it is, I will likely move on to more challenges with 3-Player chess. It's not that 3-Player chess if impossible to do. I know there is a way and I have plans. But it is more of a problem of time. The issue here is that I initially used an XYZ coordinate system on a 2-dimensional plane. While initially this isn't a problem at least for placing the pieces and creating the board, it becomes a problem when you want to mark the board and select pieces. The issues is, you can easily get the XYZ coordinate from the array by using a mathematical formula which I created. However, you cannot get the array index from the XYZ coordinate. This is because there are too many unknown variables at the same time. The formula cannot go backwards and get the unknown variable. You have to use a coordinate system that only holds two variables. Therefore, I am currently changing my implementation to fix this.

- **Main Menu**:
  - **Setting Online and Local Multiplayer Modes**: This is surprisingly difficult. The problem is to set a local or online multiplayer I need Boolean variable. The issue is that the variable gets reset as I open levels. So, whenever I open a level and set local mode on for instance, it may get turned off the moment the main menu widget is destroyed to open the level. And so far, I have been unsuccessful in setting local multiplayer on when the level opens according to when it was set in the main menu.
- **Mobile Support**:

    - **Mobile Game Support**: This part is somewhat unfinished mostly due to lack of time. I have seen the game somewhat function on a phone. But it remains untested until I have main functionality of the game working on windows. The issue is that you need very specific software and settings to
- **Networking**:
    - **Changing Game-state and Selecting Pieces:** When dealing replication, it is crucial to make sure that actions happen on the client or server at the right time. One issue I am having is that somewhere in some of my implementation, the board-state of the chess game is being resized to 0. Because of this I cannot access the board pieces on after the first time. I know this is a replication and networking issues because the local multiplayer version of the games works.
    - **Hosting Game:** One problem I am having with hosting the game is that while testing in a single level that simulates players who already joined a game works. But trying to join the game from the main menu does not work.
- Other:

My Thoughts on the Challenges of this Project:

I knew the moment I chose this project that many features would not be completed by the set time. To have as many features as I want to have. To have attempted as much as I have tried to attempt. The experimentation with many implementations. I knew that a lot of the tasks I assigned myself would not be finished in time. But I chose these tasks anyway. This is because I refuse to see this as simply a Senior Project. This is my project. And I have spent hundreds of hours on it. And I will complete it. Do I wish I could have got more done with the time that I had? Yes. Yes, I do. However, I did and will continue to do the best I can with what I have.

Future Enhancements:

Preface:

When working on this project there were many things that I wanted to achieve in a rather short amount of time. Therefore, in making this list, I knew immediately that there would be a lot of unfinished work left by this deadline. Therefore, in the other documents you may notice that my schedule extends over a very long period. This is because there is a lot I a have to fix and a lot I plan to add in when I get the chance.

Unfinished Features Section:

Considering the size and number of tasks I set for this project, there are still a lot of tasks that are unfinished at this moment that I am planning on working on. The features in this are those that I have started at some point and are either incomplete or are in testing. Some sections apply to all chess formats while some will specifically reference which chess format is having the issue. These issues are core functionality mostly and should be addressed.

- **Android Mobile Support**:
  - Mobile Input Functions and a demo of Android Works but full implementation is incomplete
  - More work is needed for it to work on online multiplayer as well
- **Chess**:
  - These are applied at least half the chess formats. If it is listed here than all the formats need work on this in some way
    - **Promoting Pawns**: When a player's pawn gets to the center mark from their position the pawn gets the choice to become a new piece
    - **Checking King**: A function to see if the player's king is in check
    - **Checkmate Function**: A function using the Check King function to see if a player's king is in check and they have no valid move.
    - **Stalemate Function**: If the player is not in check and has no valid move
    - **Deleting A Player**: When a player gets checkmate. This deletes the player's pieces. This is only applicable to 3-Player and 4-Player Chess
    - **Ending Game**: When there is only one army remaining on the board, the game ends. This is only applicable for 3-Player and 4-Player Chess.
    - **Castling**: This is applicable for all formats except 3-Player Chess
    - **En Passant Capture**: This is applicable for all formats except 3-Player
    - **Fix Player Huds**: Applicable to 3-Player and 4-Player Chess. Colors are displayed incorrectly and need to be fixed.

- **Online Multiplayer Compatibility**:
  - I need to research more on network capabilities
  - My Main Menu features the ability to input an IP address and joining a hosted game, but testing is incomplete, so it does not currently work.
  -
- **3-Player Chess**:
  - **3-Player Selecting Pieces**:
    - **Mapping Grid to Array**:
      - You can get the grid coordinates from the array index but with the chosen grid type does not allow getting an array index from the grid coordinates. A new grid coordinate system must be choses.
    - **Selecting Pieces**:
      - Previous issue needs to be fixed before we can select pieces
    - **Promoting Pawns**: When a player's pawn gets to the center mark from their position the pawn gets the choice to become a new piece
    - **Checking King**: A function to see if the player's king is in check
    - **Checkmate Function**: A function using the Check King function to see if a player's king is in check and they have no valid move.
    - **Stalemate Function**: If the player is not in check and has no valid move
    - **Deleting A Player**: When a player gets checkmate. This deletes the player's pieces
    - **Ending Game**: When there is only one army remaining on the board, the game ends

Future Features Section:

This section details features that have not been added into the game yet. They are not exactly necessary for the game to function, but they are mostly quality of life improvements for the players. None of these features are required but they are a nice touch and will be implemented after the unfinished features' implementations are complete.

- **Future Features**:
  - **Showing Available Moves**:
    - When a player clicks on their piece spots are highlighted with green for empty moveable spots and red for potential capture pieces.
  - **Options Menu**:
    - Added implementation to main menu and pause menu to allow players to change settings.
  - **Themes for chess boards**:
    - In an options menu the hosting player can choose new themes

- **Music**:
  - Music for games, and menus, and the ability to change music.
- **Static Camera Option**:
  - While I do prefer the ability to move the camera, a lot of chess games use a top-down camera for the chess boards. So, I will add static camera controls option for Chess games that are applicable.
- **New chess game variants (choices pending)**:
  - Since this game is in essence a catalogue of chess game formats, I do want to add more variants to choose from
- **Artificial Intelligence**:
  - This currently is undecided. On one hand it would be nice to implement this but making AI for chess is significantly harder than other games. That and with the extra formats it may not be implemented for a very long time if ever.
- **Convert Game to C++ from blueprints:**
  - This is so both forms of the game exist, and code is accessible and usable to everyone
- **Xbox One Controller support and by extension all controller support**
- **Alice Chess**:
  - Add the ability to press the space bar to switch back and forth between the boards
- **4-Player Chess**:
  - Decreasing movement speed of player (camera and player movement sensitivity is a little too high)