Cairo University
Faculty of Graduate Studies for Statistical Research
Department of Information Systems & Technology

Smart Online 3D-Chess

A Project Presented for Fulfillment
For Diploma Project in Computer Science

Submitted by

1.Ahmed Atef El-Habbak
2.Mohamed Saad Metwaly

Supervised by

Dr. Ahmed Hamzah

Cairo, Egypt
July 2022

Table of Contents

Abstract

Chess can be a very fun and mind-expanding game, created centuries ago. Though it's beneficial to the mind and played by millions of people around the world, therefore the level of development in computer chess programming is complicated, yet interesting as well. in this project we were supposed to develop a chess playing game by using Microsoft Visual Studio as an Integrated Development Environment for C# programming language and Unity as a Game engine. we will explain the methods used for developing a chess playing game in the following sections in this documentation.

Acknowledgement

First, we would like to thank ALLAH who helped us to complete this project. Second, we would like to express our sincere gratitude to our advisor Dr. Ahmed Hamza for his continuous support, for his patience and motivation. Third, we would like to express our sincere gratitude to our college for giving us this opportunity to exert our effort to develop this project as a teamwork.

List of Figures

Page

List of Tables

Page

List of Abbreviations

G.U.I – Graphical User Interface

L.A.N – Local Area Network

P2P – Peer 2 Peer

3D – Three Dimensional

I.D.E – Integrated Development Environment

2D – Two Dimensional

XML – Extensible Markup language

HTML – Hypertext Markup Language

U.M.L – Unified Modelling Language

OOP – Object-Oriented Programming

AI – Artificial Intelligence

IP-Address – Internet Protocol Address

CHAPTER 1: Introduction

1.1.Problem Statement :

Chess is an abstract strategy board game, it is the most popular board game which attracted people from all walks of life such as scholars, strategists, men, women, and children. chess game was originated in India in the 7th century, but the rules and appearance of pieces reached today developed in the 19th century. the increasing development in computers and INTERNET encourages our team to work on this project. Thereby, in this project we will design and implement a version of chess game with G.U.I. each piece in this game follows the basic rules of chess including capturing, check, and checkmate, encoding them by using Object-Oriented Programming. The game can be played by two players in other words human against human at the same time, also can be played by two players through Local Area Network (L.A.N).

1.2.Scope:

Our project focuses on the technical and G.U.I implementation. It follows all the basic rules of the chess such as check, checkmate and capturing. Also through our project allow players to play chess through windows operating system and play chess game over LAN network. Dut to project scope and time constraint, our chess game cannot play chess against computer, but we will search the proper algorithm and we will be implemented as can as it feasible to be implemented.

1.3.Purpose:

We can conclude the project purpose in two mandatory points:

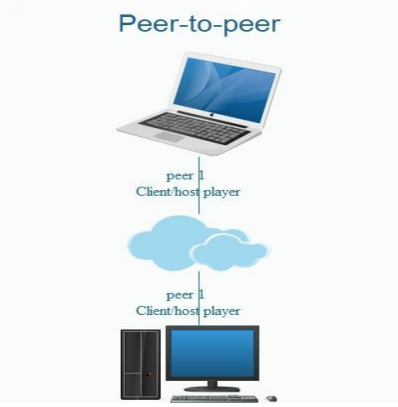
The first one: The first purpose for development this project lies at the deep desire to understand the game better. This purpose leading us to develop more efficient algorithms for implementing basic rules of chess in the game.

The second one: the second purpose for development this project is implementing the courses have been undertaking during our study this will enable us to challenge knowledge we have gained and an opportunity to strengthen them.

CHAPTER 2: System Analysis

2.1. System Architecture

In this section we encounter a briefly talk about system architecture used in this project , this system is P2P . according to this system a group of computers are linked together with equal permissions and responsibilities for processing data. Unlike traditional client-server networking, no devices in a P2P network are designated solely to serve or to receive data



2.2. System requirements specification:

Due to project scope, in this section we will discuss requirements analysis through functional and non-functional requirements.

2.3. Functional requirements:

2.3.1. Requirement specifications:

Table 2.3.1: Requirement specifications table

Identifier	Priority	Requirement
REQ-01	5	System shall allow local player to play local game
REQ-02	3	System shall allow local player to move chess piece
REQ-03	4	System shall allow local player to check whether move is valid or not
REQ-04	3	System shall allow local player to capture piece
REQ-05	2	system shall allow local player to perform castling
REQ-06	1	System shall allow local user to perform promotion
REQ-07	2	System shall allow local player to perform EnPassant
REQ-08	4	System shall allow local player to perform checkmate

REQ-09	3	System shall allow local player to perform exit from game.
REQ-10	2	System shall allow local player to rematch chess game.
REQ-11	2	System shall allow online player to play online game

2.3.2. User stories:

Table 2.3.2: User stories table

Identifier	User story	Size
ST-01	As a local player I can play a local game.	2pt
ST-02	As a local player I can move a piece.	4pt
ST-03	As a local player I can check whether moving a piece is valid or not	6pt
ST-04	As a local player I can capture piece.	2pt
ST-05	As a local player I can perform a checkmate.	4pt
ST-06	As a local player I can perform castling.	2pt
ST-07	As a local player I can perform EnPassant.	4pt
ST-08	As a local player I can perform promotion	6pt
ST-09	As a local player I can perform exit form game	4pt
ST-10	As a local player I can perform rematch	2pt
ST-11	As an online player I can play chess game online	4pt

2.3.3 work backlog:

Table 2.3.3: Work backlog table

Work items	User story	Iteration no	Estimated work duration
01	ST-01 Start local game	Iteration 1	2pt (1 day)
02	ST-02 Move piece	Iteration 2	4pt (2 day)
03	ST-03 Checking valid move	Iteration 3	6pt (2 day)
04	ST-04 Capturing piece	Iteration 4	2pt (1 day)
05	ST-05 Perform checkmate	Iteration 5	4pt (1 day)
06	ST-06 Perform castling	Iteration 6	2pt (1 day)
07	ST-07 Perform EnPassant	Iteration 7	4pt (2 day)
08	ST-08 Perform promotion	Iteration 8	6pt (1 day)
09	ST-09 Perform exit from game	Iteration 9	4pt (2 day)
10	ST-10 Rematching game	Iteration 11	2pt (1 day)
11	ST-11 Play online game	Iteration 12	4pt (1 day)

2.3.4 project duration

Work duration = path size / travel velocity

Path size = total work size So, work size = 40 / 2 = 20 days

2.3.5 Use-Case Diagram:

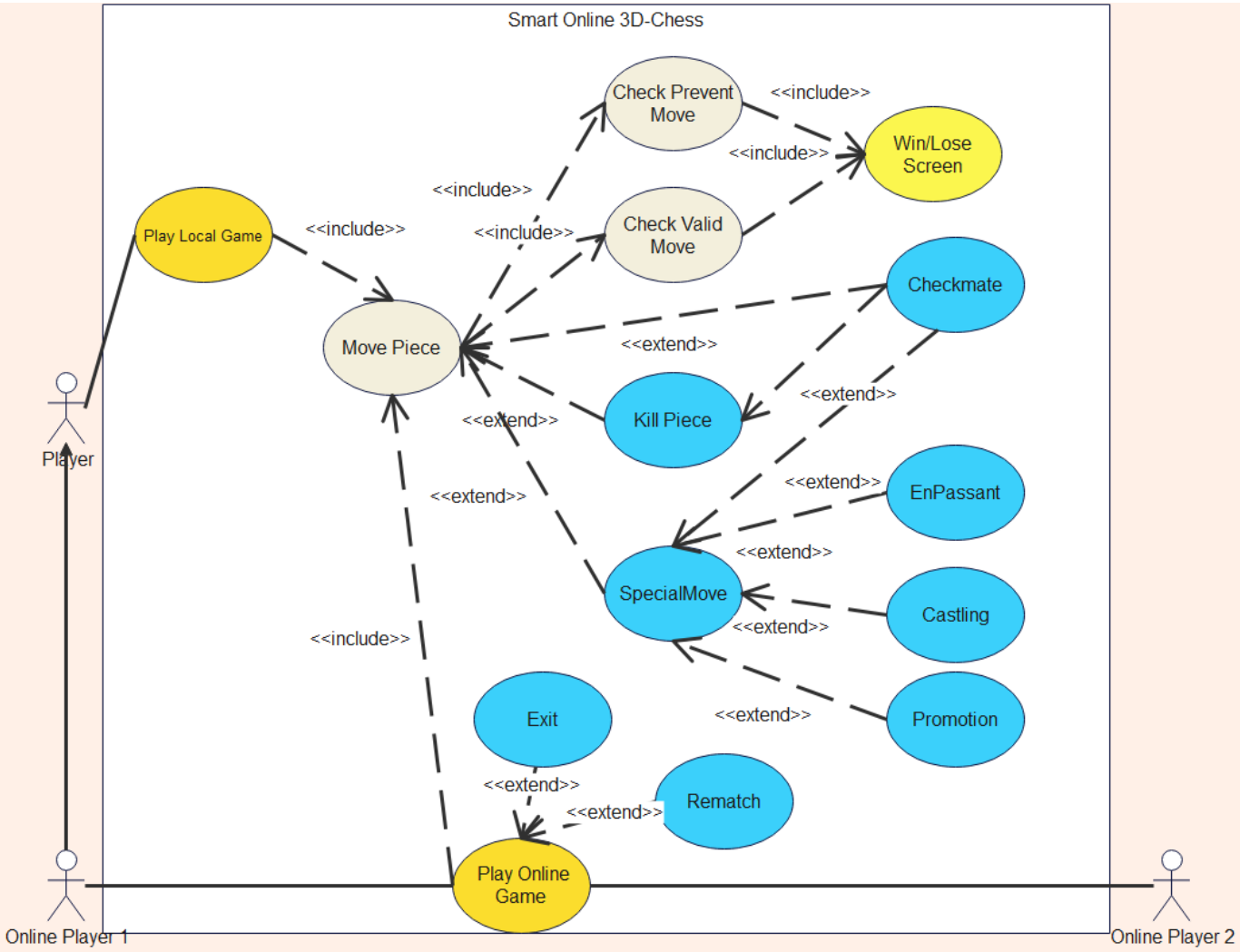


Figure 2.3.5: Use-Case diagram

2.3.6 Actor Description:

Table 2.3.6: Actor description table

Actor	Actor's Goal	Use-Case name
Local player	▪ Play a local chess game ▪ Moving a chess piece ▪ Checking validity of moving a piece ▪ Capturing piece ▪ Perform checkmate ▪ Perform castling ▪ Perform EnPassant ▪ Making promotion ▪ Exit from game	UC-01 play local game UC-02 moving piece UC-03 validity check UC-04 piece capture UC-05 checkmate UC-06 castling UC-07 EnPassant UC-08 promotion UC-09 exit game UC-10 rematch

	▪ Rematch chess game	
Online player	▪ Playing online chess game	UC-11 online game

2.3.7 Use-case description:

Table 2.3.7: Use-Case 1

USE-CASE 1 (UC-1)	
Related requirements	REQ-01
Initial actor	Local player
Actor's goal	Starting a new local game
Participating actor	
Pre-condition	Local player has properly turned on the system and connected to Chess Game.
Post-condition	Successfully starting a local chess game with all authorized operations
Flow of events for main success scenarios	
1. local player turn the system on 2. System response running appropriate functions and tasks 3. local player click on local game button 4. System response by displaying main game screen	
Flow of events for extension (Alternate scenario)	
2 system didn't run properly 1 local player repeat step 1	

Table 2.3.7: Use-Case 2

USE-CASE 2 (UC-2)	
Related requirements	REQ-02
Initial actor	Local player
Actor's goal	local player moving any chess piece
Participating actor	
Pre-condition	Local player successfully turn the system on and properly start a new local game .
Post-condition	Local player successfully moving a piece .
Flow of events for main success scenarios	

1. Local playe click on a piece to move 2. System responds by highlighting the possible positions to move 3. Local player successfully moved piece to destination position
Flow of events for extension (Alternate scenario)
2 System didn't highlight the possible positions. 1 local player repeat step 1

Table 2.3.7: Use-Case 3

USE-CASE 3 (UC-3)	
Related requirements	REQ-03
Initial actor	Local player
Actor's goal	Checking whether the piece moving is valid or not
Participating actor	
Pre-condition	Local player successfully turned the system on and started a local game
Post-condition	Local player properly performing a check for moving piece
Flow of events for main success scenarios	
1. Local player try to move a chess piece 2. System responds by checking validity of this move 3. Local user moved a piece	
Flow of events for extension (Alternate scenario)	
2 System refused local player move 1 Local player repeat step move	

Table 2.3.7: Use-Case 4

USE-CASE 4 (UC-4)	
Related requirements	REQ-04
Initial actor	Local player
Actor's goal	Local player capturing an opponent piece
Participating actor	
Pre-condition	Local player properly turn the system on , successfully started a local game

Post-condition	Local player successfully capturing piece
Flow of events for main success scenarios	
1. local player wants to capture an opponent piece 2. System responds by capturing the piece	
Flow of events for extension (Alternate scenario)	
2 System responds by refusing the piece capturing 3 Local player repeat step 1.	

Table 2.3.7: Use-Case 5

USE-CASE 5 (UC-5)	
Related requirements	REQ-05
Initial actor	Local player
Actor's goal	The local player performs castling move
Participating actor	
Pre-condition	Local player properly turn the system on , successfully started a local game
Post-condition	Local player properly perform a castling move
Flow of events for main success scenarios	
1. Local player try to perform a castling move 2. System responds by checking validity (check whether it is the first move for the king) , then perform the appropriate move	
Flow of events for extension (alternate scenario)	
2 System responds by refusing the move. 3 Local player try to perform another move.	

Table 2.3.7: Use-Case 6

USE-CASE 6 (UC-6)	
Related requirements	REQ-08
Initial actor	Local player
Actor's goal	Local player performs a checkmate move
Participating actor	

Pre-condition	Local player properly turn the system on , successfully started a local game
Post-condition	Local player successfully performed a checkmate
Flow of events for main success scenarios	
	1. Local player try to perform a checkmate 2. System responds by checking validity of move (king can't move to any position) 3. Local player performed a checkmate move 4. System responds by displaying win/lose screen
Flow of events for extension (Alternate scenario)	
	2 System responds by refusing checkmate move 3 local player try to perform another move.

Table 2.3.7: Use-Case 7

USE-CASE 7 (UC-7)	
Related requirements	REQ-07
Initial actor	Local player
Actor's goal	Local player performs EnPassant move
Participating actor	
Pre-condition	Local player properly turn the system on, successfully started a local game
Post-condition	Local player successfully perform EnPassant move
Flow of events for main success scenarios	
	1. Local player try to perform EnPassant move 2. System responds by checking validity for this move
Flow of events for extension (Alternate scenario)	
	2 System responds by refusing this move 3 local player try to performs another move

Table 2.3.7: Use-Case 8

USE-CASE 8 (UC-8)	
Related requirements	REQ-06
Initial actor	Local player
Actor’s goal	Local player perform a pawn promotion
Participating actor	
Pre-condition	Local player properly turn the system on, successfully started a local game
Post-condition	Local player successfully promoted a pawn
Flow of events for main success scenarios	
1. Local players perform a promotion 2. System reponds by checking validity for this move	
Flow of events for extension (Alternate scenario)	
2. System responds by refusing a promotion move 3. local player try to perform another move	

Table 2.3.7: Use-Case 9

USE-CASE 9 (UC-9)	
Related requirements	REQ-09
Initial actor	Local player
Actor’s goal	Local player exit from a chess game
Participating actor	
Pre-condition	Local player properly turn the system on, successfully started a local game
Post-condition	Local player successfully exit from game
Flow of events for main success scenarios	
1. Local players exit form current game 2. System responds by exiting form current game	
Flow of events for extension (Alternate scenario)	
2 System can’t exit from the current game . 3 local player kill the process .	

Table 2.3.7: Use-Case 10

USE-CASE 10 (UC-10)	
Related requirements	REQ-10
Initial actor	Local player
Actor's goal	Local player perform a rematch for a chess game
Participating actor	
Pre-condition	Local player properly turns the system on, successfully started a local game
Post-condition	Local player successfully perform a rematch
Flow of events for main success scenarios	
1. Local player rematch a chess game 2. System responds by starting a new local game	
Flow of events for extension (Alternate scenario)	
2 System can't start a new local game 1 local player will repeat step 1	

Table 2.3.7: Use-Case 11

USE-CASE 11 (UC-11)	
Related requirements	REQ-11
Initial actor	Online player
Actor's goal	Online player start an online game
Participating actor	
Pre-condition	online player properly turn the system on
Post-condition	Online player successfully started a new online game
Flow of events for main success scenarios	
1. Online player 1 insert internet protocol address (IP_Address) to start online game . 2. System responds by delivering IP_Address to online player 2 3. The online player 2 approved the play request	

Flow of events for extension (Alternate scenario)
3 the online player 2 refused to approve play request 1 online player 1 repeat step 1

2.3.8 Traceability Matrix:

Table 2.3.8: Traceability matrix

REQ-ID	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11
REQ1	5	✖										
REQ2	3		✖									
REQ3	4			✖								
REQ4	3				✖							
REQ5	2					✖						
REQ6	1								✖			
REQ7	2							✖				
REQ8	4						✖					
REQ9	4									✖		
REQ10	2										✖	
REQ11	2											✖
Max Priority Weight		5	3	4	3	2	4	2	1	4	2	2
Total Priority weight		5	3	4	3	2	4	2	1	4	2	3

According to traceability matrix:

Table 2.3.8: Table illustrates order of implementation based on traceability matrix

Use case	Max Priority weight	Total Priority weight	Order of implementation
UC-01	5	5	1
UC-02	3	3	5
UC-03	4	4	2
UC-04	3	3	6
UC-05	2	2	7
UC-06	4	4	3
UC-07	2	2	8
UC-08	1	1	11
UC-09	4	4	4
UC-10	2	2	9
UC-11	2	2	10

2.3.9 Activity Diagrams:

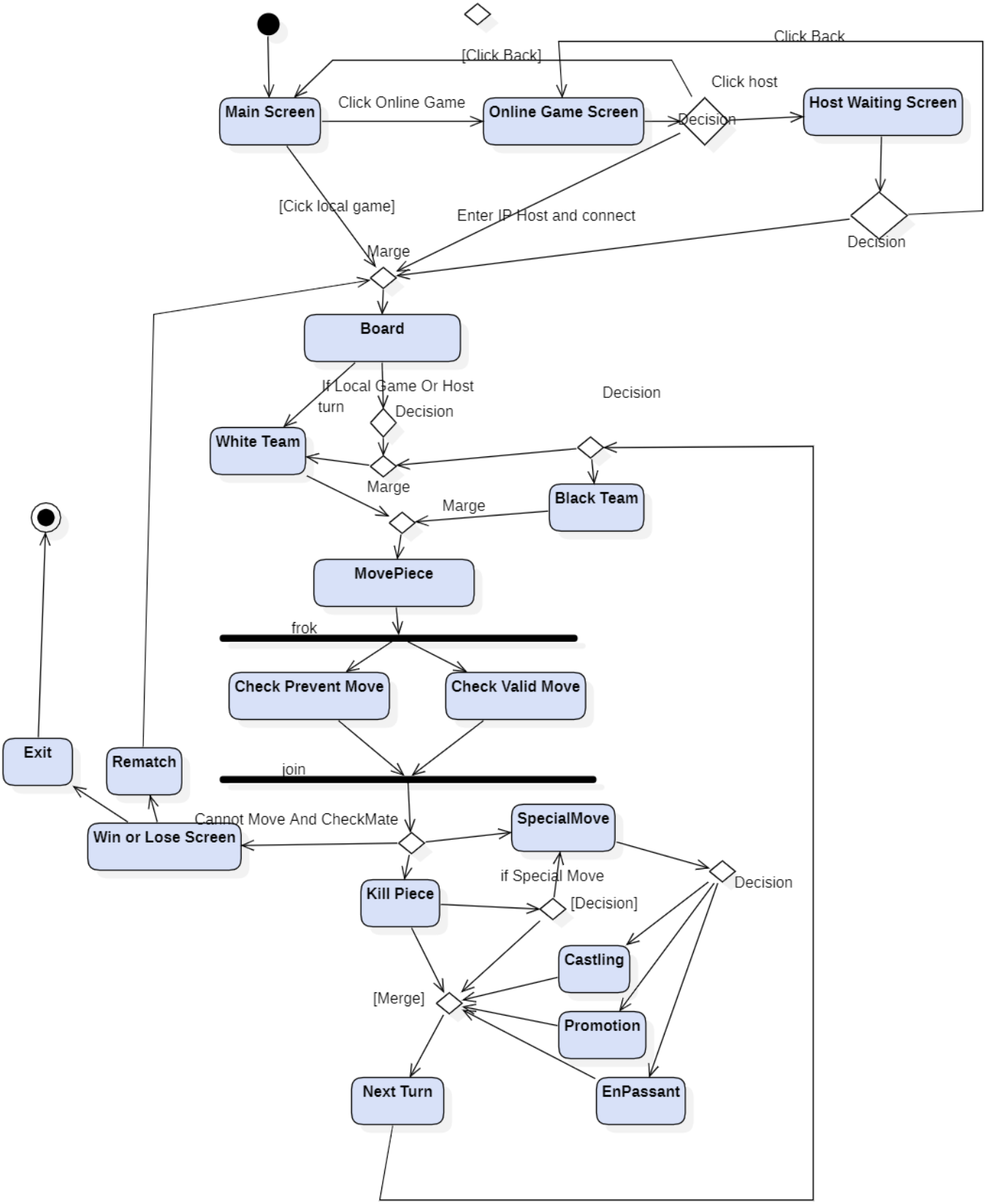


Figure 2.3.9: Activity diagram

2.3.10 Sequence Diagram:

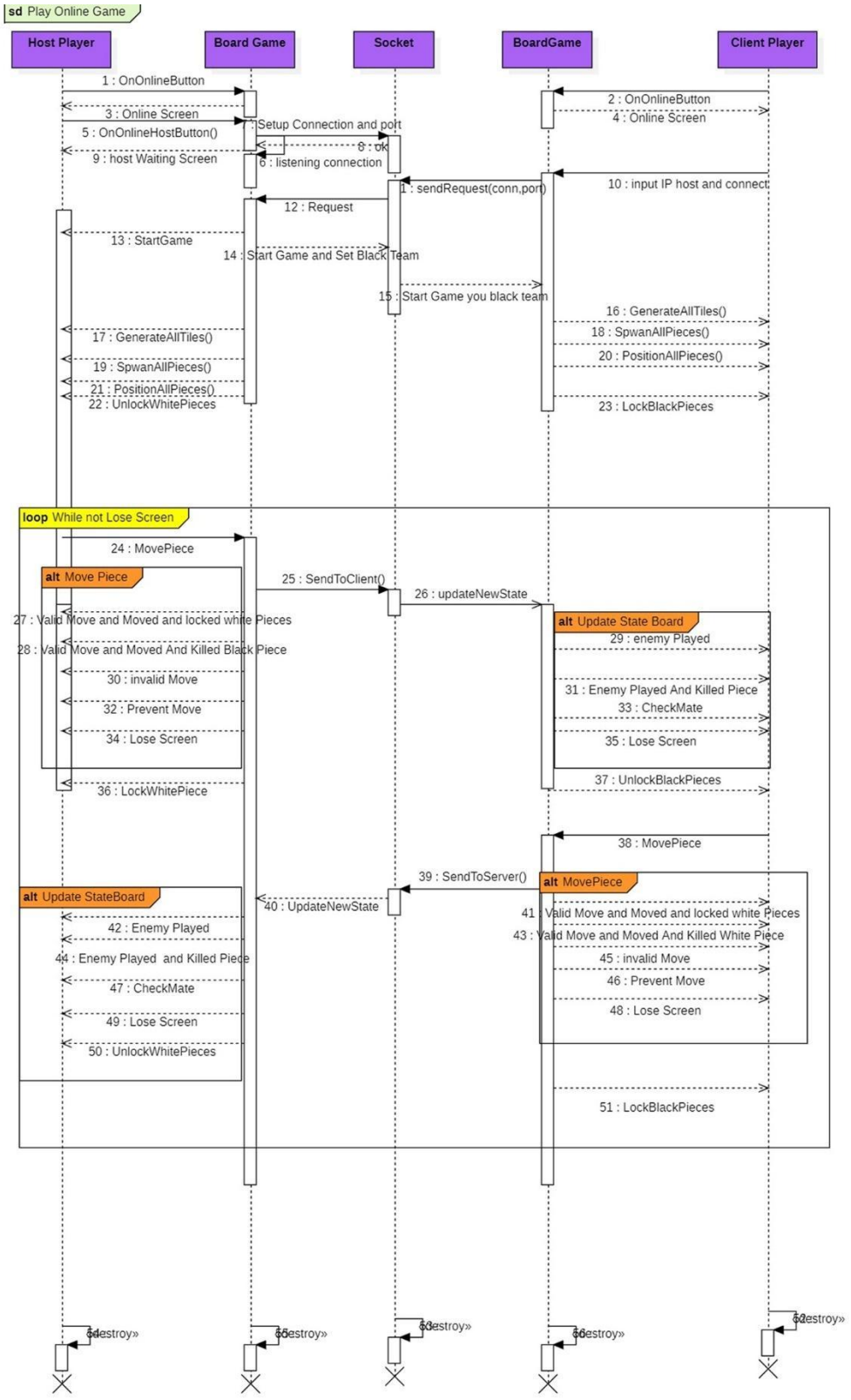


Figure 2.3.10: Sequence diagram

2.4. Non-functional requirements

Non-functional requirement describe services provide by system under which the system operational constraints thereby we can summarize non-functional requirements at the following points:

- Base and pieces must not be overly large.
- User interface must be easy to understand and use by players.
- peace movement system must move pieces in properly right way.
- The game will have no more than a 3 second lag between user input and response.
- Application will not be being crashed while it's used.
- Application can be installed on any windows platform.

CHAPTER 3: System Design

3.1 Component Diagram:

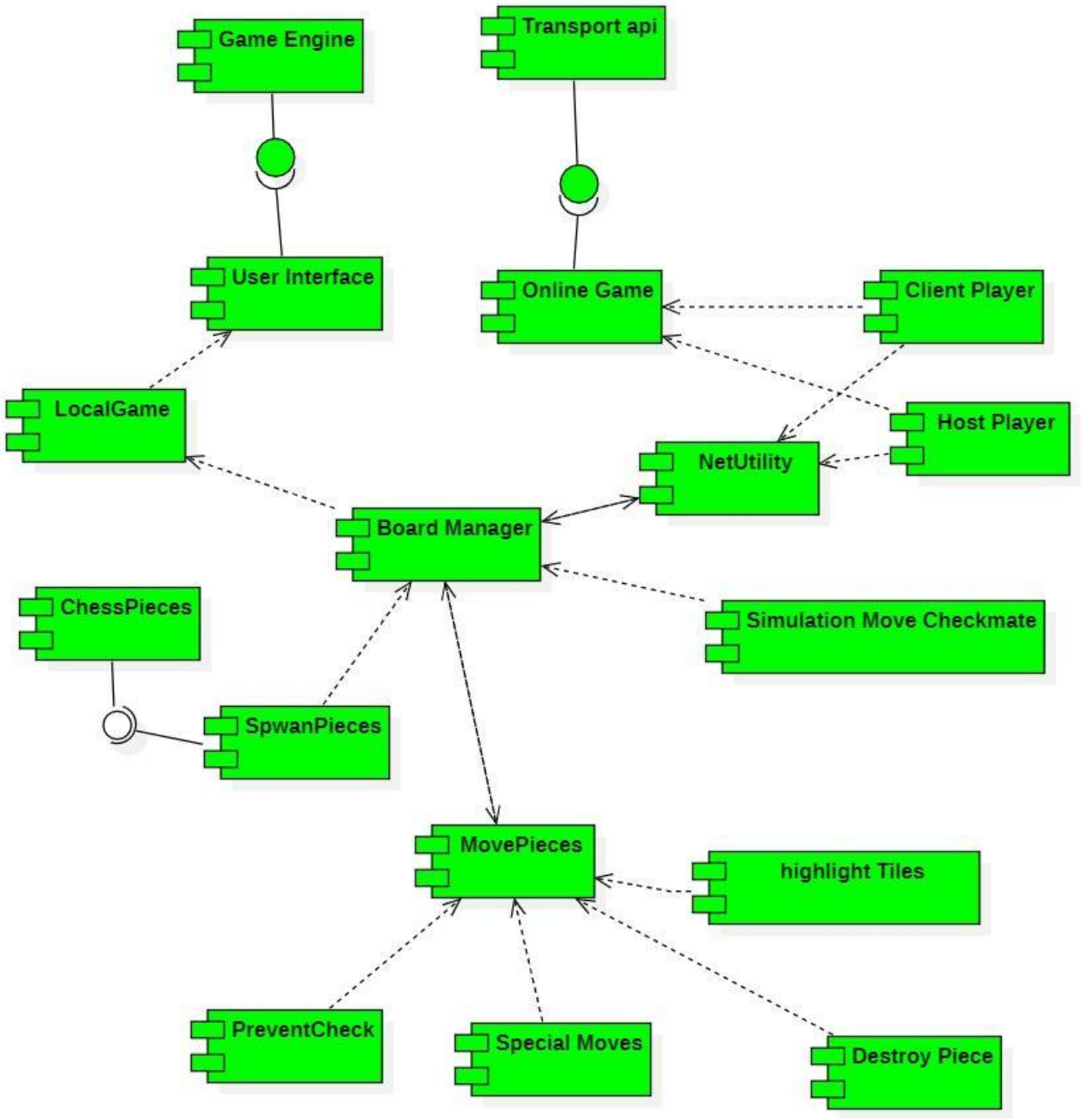


Figure 3.1: Component diagram

3.2 Class Diagram:

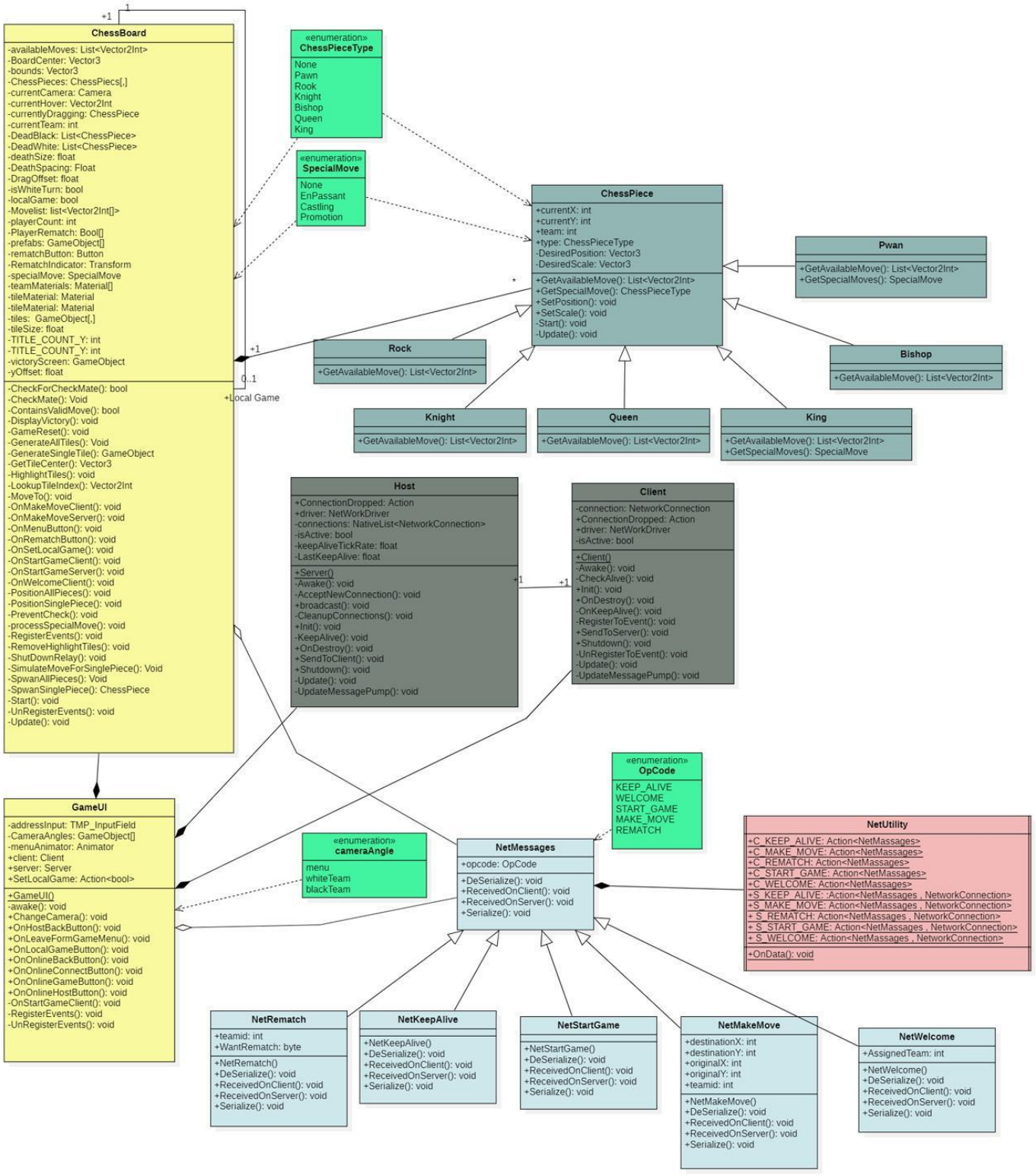


Figure 3.2: Class diagram

3.3 Testing (User Acceptance Test)

3.3.1 Test Cases:

Table 3.3.1: Test case-1

Test case identifier	TC-1
Use case tested	UC-1
Pass/fail criteria	The test passes if local player successfully start a local game
Input data	Processes and functions which turn on the system
Test procedures	**Expected result**
local player turn system on	System will accept by displaying main game screen

Table 3.3.1: Test case-2

Test case identifier	TC-2
Use case tested	UC-2
Pass/fail criteria	The test passes if local player successfully move a chess piece
Input data	Processes and functions which turn on the system
Test procedures	**Expected result**
local player moves a chess piece	System will respond by highlighting possible positions

Table 3.3.1: Test case-3

Test case identifier	TC-3
Use case tested	UC-3
Pass/fail criteria	The test passes if local player successfully move a piece
Input data	Processes and functions which turn on the system
Test procedures	**Expected result**
Local player move a chess piece	System will respond by checking validity for this move, then allow this move if appropriate piece properly move .

Table 3.3.1: Test case-4

Test case identifier	TC-4
Use case tested	UC-4
Pass/fail criteria	The test passes if local player properly capture a piece
Input data	Processes and functions which turn on the system
Test procedures	**Expected result**
Local user capture an opponent piece	System will respond by checking validity then capturing piece

Table 3.3.1: Test case-5

Test case identifier	TC-5
Use case tested	UC-5
Pass/fail criteria	The test passes if local player successfully perform castling move
Input data	Processes and functions which turn on the system
Test procedures	**Expected result**
Local player perform a castling move	System will accept by checking validity then perform this move

Table 3.3.1: Test case-6

Test case identifier	TC-6
Use case tested	UC-6
Pass/fail criteria	The test passes if local player successfully perform a checkmate
Input data	Processes and functions which turn on the system
Test procedures	**Expected result**
Local player performs a checkmate move	System will responds by checking validity of move , then display win/lose screen

Table 3.3.1: Test case-7

Test case identifier	TC-7
Use case tested	UC-7
Pass/fail criteria	The test passes if local player successfully perform EnPassant move
Input data	Registered user credentials: username, password
Test procedures	**Expected result**
Local player perform EnPassant move	System will respond by performing appropriate move

Table 3.3.1: Test case-8

Test case identifier	TC-8
Use case tested	UC-8
Pass/fail criteria	The test passes if local player properly perform pawn promotion
Input data	Processes and functions which turn on the system
Test procedures	**Expected result**
Local player perform promotion	System will responds by performing appropriate move

Table 3.3.1: Test case-9

Test case identifier	TC-9
Use case tested	UC-9
Pass/fail criteria	The test passes if local player successfully perform exit from game
Input data	Processes and functions which turn on the system
Test procedures	**Expected result**

Local player perform exit from game	System will responds by ending the game

Table 3.3.1: Test case-10

Test case identifier	TC-10
Use case tested	UC-10
Pass/fail criteria	The test passes if local player successfully rematch a chess game
Input data	Processes and functions which turn on the system
Test procedures	**Expected result**
Local player perform rematch	System will respond by restart current chess game

Table 3.3.1: Test case-11

Test case identifier	TC-11
Use case tested	UC-11
Pass/fail criteria	The test passes if online player successfully play online game
Input data	Online plyer insert internet protocol address (IP_Address)
Test procedures	**Expected result**
Online player 1 send play request to online player 2	System will responds by delivering online player 1 to online player 2

CHAPTER 4: Implementation Details

4.1 Report Writing Tool:

The Report Writing Tool is Microsoft Office Word 2022

4.2. System Development Tool:

- Unity Game engine as Game engine for creating game.
- Microsoft Visual Studio as I.D.E and code editor for C# programming Language.

4.2.1 Unity Game Engine:

Unity is a cross-platform game engine developed by unity technologies, this engine has been gradually extended to support a variety of desktop, mobile and virtual reality platforms . this engine can be used to develop 3D and 2D games.

4.2.2 Microsoft Visual Studio:

An I.D.E used in develop computer programs as well as web-applications , web-sites and mobile applications . Microsoft Visual Studio support 36 different programming language such as C, C++, C#, F#, JavaScript , XML.HTML.

4.3 Hints and Notes about different versions for tools:

Our team highly recommends that any development team must agree for accepted version of any development software tool and unify that version among the whole team members to save their efforts and time.

4.4 Form Design

Through this project, user login form divided into two sections , first section is local player form (local game) and the other one is online player form(online game) in which two players can play over LAN or INTERNET

4.4.1 Start Menu Screen

While user press local game , this will allow two players playing chess game with all moves for all pieces at the same time and same device .following figure will illustrate this process .

Or While user press online game He's allowed two choices first choice he can play as host and listening for client connection . second choice he can play as client and insert IP host

Figure 4.4.1: Start Menu Screen

4.4.2 Host menu Screen

Listening for client connection to go in game screen

Figure 4.4.2: Host menu screen

Figure 4.4.2: Waiting screen

Figure 4.4.2:In game screen

4.5 Class design

It is well known that the class diagram is the most significant diagrams in U.M.L so it can be used in describing the game implementation and relationship between game classes

4.5.1 ChessBoard.cs Class:

This class considered the main class in our project in which, countian's the most significant attributes and methods used in game operation processes

Class Attributes:

- tileSize: a variable which have a float datatype for determining the tile size of chess board.
- tile_Count_X: a variable which have integer data type used for determining the count of horizontal tiles.

- tile_Count_Y: a variable which have integer data type used for determining the count of vertical tiles.

Class Methods:

- Start (): a method with no return data type which responsible for initiate the values and objects in start frame in chess game.
- GenerateALLTiles (): a method with no return data type which responsible for generate the tile in chess board .
- SpwanAllPieces (): a method with no return data type which responsible for create all pieces objects and pieces positions in the chess game .
- PositionAllPieces (): a method with no return data type which responsible for placing all pieces on the chess board.
- HighlightTiles(): a method responsible for creating highlight on a specific position which player directed the cursor to it .

4.5.2 Class ChessBoard Code:

```
using System;
using System.Collections.Generic;
using Unity.Networking.Transport;
using UnityEngine;
using UnityEngine.UI;
public enum SpecialMove
{
    None = 0,
    EnPassant,  // ref https://www.youtube.com/watch?v=c_KRIH0wnhE
    Castling,  // https://www.youtube.com/watch?v=FcLYgXCkucc
    Promotion // https://www.youtube.com/shorts/Tt8VTZFPFa4
}
public class ChessBoard : MonoBehaviour
{
    [Header("Art Stuff")]
    [SerializeField] private Material tileMaterial;
    [SerializeField] float tileSize = 1.0f;
    [SerializeField] float yOffset = 0.2f;
    [SerializeField] private Vector3 boardCenter = Vector3.zero;
    [SerializeField] private float DeathSpacing = 0.3f;
    [SerializeField] private float deathSize = 0.3f;
     [SerializeField] private float dragOffset = 1f; // to fix magrge while two piece while moves
and that value controle y to make it up chess board
    [SerializeField] private GameObject victoryScreen;
    [SerializeField] private Transform rematchIndicator;
    [SerializeField] private Button rematchButton;
    [Header("Prefabs\" Please look ChessPieces.cs\" && Materials")]
    [SerializeField] private GameObject[] prefabs;
```

```
    [SerializeField] private Material[] teamMaterials;


    //LOGIC
    private const int TITLE_COUNT_X = 8;
    private const int TITLE_COUNT_Y = 8;
    private GameObject[,] tiles;
    private Camera currentCamera;
    private Vector2Int currentHover;
    private ChessPiece[,] chessPieces;
    private ChessPiece currentlyDragging; // for method drag mouse
        private List<ChessPiece> DeadWhite = new List<ChessPiece>(); //remove dead White pieces
    private List<ChessPiece> DeadBlack = new List<ChessPiece>(); //remove dead black pieces
    private Vector3 bounds;
        private bool isWhiteTurn; //  turn mechanic to  make white play firast and swaping between 2 player white and black

      private List<Vector2Int> availableMoves = new List<Vector2Int>(); ////to make hightlight tiles

     private List<Vector2Int[]> movelist = new List<Vector2Int[]>();        //to save movement for each pieces "for special moves"
    private SpecialMove specialMove;
    // MultiPlayer Logic
    private int playerCount = -1; // for server
    private int currentTeam = -1;  // for server and client
    private bool localGame = true; // To local Game Fix
    private bool[] PlayerRematch = new bool[2];
```

```
private void Start()
{
    isWhiteTurn = true;

    GenerateAllTiles(tileSize, TITLE_COUNT_X, TITLE_COUNT_Y);
    SpwanAllPieces();
    PositionAllPieces();
    RegisterEvents();  // Listening for net Welcome Massage
}
private void Update()
{
    if (!currentCamera)
    {
        currentCamera = Camera.main;
        return;
    }
    RaycastHit info;
    Ray ray = currentCamera.ScreenPointToRay(Input.mousePosition);
    if (Physics.Raycast(ray, out info, 100, LayerMask.GetMask("Tile", "Hover", "Highlight")))
    {   // Get the index of the tile i have hit
        Vector2Int hitPosition = LookupTileIndex(info.transform.gameObject);

        if (currentHover == -Vector2Int.one)
        {
            // First time Hovering
            currentHover = hitPosition;
            tiles[hitPosition.x, hitPosition.y].layer = LayerMask.NameToLayer("Hover");
        }
```

```
        //if we were already hovering a tile ,change the prevous one
        if (currentHover != hitPosition)
        {
            tiles[currentHover.x, currentHover.y].layer = (ContainsValidMove(ref availableMoves,
currentHover)) ? LayerMask.NameToLayer("Highlight") : LayerMask.NameToLayer("Tile");
            currentHover = hitPosition;
            tiles[hitPosition.x, hitPosition.y].layer = LayerMask.NameToLayer("Hover");
        }
        // drag mouse pics  "if we down on mouse "
        if (Input.GetMouseButtonDown(0))
        {
            if (chessPieces[hitPosition.x, hitPosition.y] != null)
            {
                //is it out turn?
                        if ((chessPieces[hitPosition.x, hitPosition.y].team == 0 && isWhiteTurn &&
currentTeam ==0) || (chessPieces[hitPosition.x, hitPosition.y].team == 1 && !isWhiteTurn &&
currentTeam ==1))
                {
                    currentlyDragging = chessPieces[hitPosition.x, hitPosition.y];

                    //to make hightlight tiles
                    // Get aList of where i can go and hightlight tiles
                            availableMoves = currentlyDragging.GetAvailableMove(ref chessPieces,
TITLE_COUNT_X, TITLE_COUNT_Y);
                    // Get a List Special moves
                            specialMove = currentlyDragging.GetSpecialMoves(ref chessPieces, ref
movelist, ref availableMoves);
                    PreventCheck(); // Lock if CheckMate!
```

```
                HighlightTiles();
            }
        }

    }
    //if we releasing the mouse button and draging
    if (currentlyDragging != null && Input.GetMouseButtonUp(0))
    {
                Vector2Int previousPosition = new Vector2Int(currentlyDragging.currentX, currentlyDragging.currentY);

                if (ContainsValidMove(ref availableMoves, new Vector2Int(hitPosition.x, hitPosition.y)))
        {
                MoveTo(previousPosition.x, previousPosition.y, hitPosition.x, hitPosition.y); // check if can move
            // Net Implementation
            NetMakeMove makemove = new NetMakeMove();
            makemove.originalX = previousPosition.x;
            makemove.originalY = previousPosition.y;
            makemove.destinationX = hitPosition.x;
            makemove.destinationY= hitPosition.y;
            makemove.teamId = currentTeam;
            Client.Instance.SendToServer(makemove);
        }
        else
        {
```

```
                                currentlyDragging.SetPosition(GetTileCenter(previousPosition.x,
previousPosition.y)); //back prev postion
                currentlyDragging = null;
                RemoveHighlightTiles();
            }
        }
    }
    else
    {
        if (currentHover != -Vector2Int.one)
        {
            tiles[currentHover.x, currentHover.y].layer = (ContainsValidMove(ref availableMoves,
currentHover)) ? LayerMask.NameToLayer("Highlight") : LayerMask.NameToLayer("Tile");
            currentHover = -Vector2Int.one;
        }
          if (currentlyDragging && Input.GetMouseButtonUp(0)) //Fix Bug Dropping reference
on release
        {
                        currentlyDragging.SetPosition(GetTileCenter(currentlyDragging.currentX,
currentlyDragging.currentY));
            currentlyDragging = null;
            RemoveHighlightTiles();
        }
    }
    // we are dragging a piece
    if (currentlyDragging)
    {
        Plane horizonatalPlane = new Plane(Vector3.up, Vector3.up * yOffset);
```

```
        float distance = 0.0f;
        if (horizonatalPlane.Raycast(ray, out distance))
            currentlyDragging.SetPosition(ray.GetPoint(distance) + Vector3.up * dragOffset);
              // fix magrge between two pieces and make movment whith incressing in y-axis 0.9float
        // postion change while dragging
      }
   }
   // Generate the board
   private void GenerateAllTiles(float tileSize, int tileCountX, int tileCountY)
   {
      yOffset += transform.position.y;
           bounds = new Vector3((tileCountX / 2) * tileSize, 0, (tileCountX / 2) * tileSize) + boardCenter;
      tiles = new GameObject[tileCountX, tileCountY];
      for (int x = 0; x < tileCountX; x++)
         for (int y = 0; y < tileCountY; y++)
            tiles[x, y] = GenerateSingleTile(tileSize, x, y);
   }
   private GameObject GenerateSingleTile(float tileSize, int x, int y)
   {
      GameObject tileObject = new GameObject(string.Format("X:{0}", "Y:{1}", x, y));
      tileObject.transform.parent = transform;
      Mesh mesh = new Mesh();
      tileObject.AddComponent<MeshFilter>().mesh = mesh;
      tileObject.AddComponent<MeshRenderer>().material = tileMaterial;

      Vector3[] vertices = new Vector3[4];
```

```
        vertices[0] = new Vector3(x * tileSize, yOffset, y * tileSize) - bounds;
        vertices[1] = new Vector3(x * tileSize, yOffset, (y + 1) * tileSize) - bounds;
        vertices[2] = new Vector3((x + 1) * tileSize, yOffset, y * tileSize) - bounds;
        vertices[3] = new Vector3((x + 1) * tileSize, yOffset, (y + 1) * tileSize) - bounds;

        int[] tris = new int[] { 0, 1, 2, 1, 3, 2 };
        mesh.vertices = vertices;
        mesh.triangles = tris;

        mesh.RecalculateNormals();
        tileObject.layer = LayerMask.NameToLayer("Tile");
        tileObject.AddComponent<BoxCollider>();
        return tileObject;

    }
    // Spwan of the Pieces
    private void SpwanAllPieces()
    {
        chessPieces = new ChessPiece[TITLE_COUNT_X, TITLE_COUNT_Y];
        int whiteTeam = 0, blackTeam = 1;
        // white team using ref in 2nd day in google drive image
        chessPieces[0, 0] = SpwanSinglePiece(ChessPieceType.Rook, whiteTeam);
        chessPieces[1, 0] = SpwanSinglePiece(ChessPieceType.Knight, whiteTeam);
        chessPieces[2, 0] = SpwanSinglePiece(ChessPieceType.Bishop, whiteTeam);
        chessPieces[3, 0] = SpwanSinglePiece(ChessPieceType.Queen, whiteTeam);
        chessPieces[4, 0] = SpwanSinglePiece(ChessPieceType.King, whiteTeam);
        chessPieces[5, 0] = SpwanSinglePiece(ChessPieceType.Bishop, whiteTeam);
        chessPieces[6, 0] = SpwanSinglePiece(ChessPieceType.Knight, whiteTeam);
```

```
        chessPieces[7, 0] = SpwanSinglePiece(ChessPieceType.Rook, whiteTeam);
        for (int i = 0; i < TITLE_COUNT_X; i++) // spwan pwan white team
        {
            chessPieces[i, 1] = SpwanSinglePiece(ChessPieceType.pawn, whiteTeam);
        }
        // black team using ref in 2nd day in google drive image
        chessPieces[0, 7] = SpwanSinglePiece(ChessPieceType.Rook, blackTeam);
        chessPieces[1, 7] = SpwanSinglePiece(ChessPieceType.Knight, blackTeam);
        chessPieces[2, 7] = SpwanSinglePiece(ChessPieceType.Bishop, blackTeam);
        chessPieces[3, 7] = SpwanSinglePiece(ChessPieceType.Queen, blackTeam);
        chessPieces[4, 7] = SpwanSinglePiece(ChessPieceType.King, blackTeam);
        chessPieces[5, 7] = SpwanSinglePiece(ChessPieceType.Bishop, blackTeam);
        chessPieces[6, 7] = SpwanSinglePiece(ChessPieceType.Knight, blackTeam);
        chessPieces[7, 7] = SpwanSinglePiece(ChessPieceType.Rook, blackTeam);
        for (int i = 0; i < TITLE_COUNT_X; i++) // spwan pwan black team
        {
            chessPieces[i, 6] = SpwanSinglePiece(ChessPieceType.pawn, blackTeam);
        }
    }
    private ChessPiece SpwanSinglePiece(ChessPieceType type, int team)
    {
                                    ChessPiece cp = Instantiate(prefabs[(int)type - 1],
transform).GetComponent<ChessPiece>(); // to spwan piecs from chessPieces
        cp.type = type;
        cp.team = team;
        // note if team 0 i add 0 if team black i add 6
            cp.GetComponent<MeshRenderer>().material = teamMaterials[((team == 0) ? 0 : 6) +
((int)type - 1)]; // to adding material in GUI Unity for every piecs
```

```
        return cp;
    }
    // Positioning of spwaned pieces
    private void PositionAllPieces() // to set position for all pieces
    {
        for (int x = 0; x < TITLE_COUNT_X; x++)
            for (int y = 0; y < TITLE_COUNT_Y; y++)
                if (chessPieces[x, y] != null)
                    PositionSinglePiece(x, y, true);
    }
     private void PositionSinglePiece(int x, int y, bool force = false)  // to use it in for loop in
method positionAllPieces to set position for pieces
    {
        chessPieces[x, y].currentX = x;
        chessPieces[x, y].currentY = y;
        chessPieces[x, y].SetPosition(GetTileCenter(x, y), force);
    }
    private Vector3 GetTileCenter(int x, int y)
    {
         return new Vector3(x * tileSize, yOffset, y * tileSize) - bounds + new Vector3(tileSize / 2,
0, tileSize / 2);
    }
    //HighLight Tiles
    private void HighlightTiles()
    {
        for (int i = 0; i < availableMoves.Count; i++)
        {
```

```
                                        tiles[availableMoves[i].x,  availableMoves[i].y].layer  =
LayerMask.NameToLayer("Highlight");
        }
    }
    private void RemoveHighlightTiles()
    {
        for (int i = 0; i < availableMoves.Count; i++)
        {
            tiles[availableMoves[i].x, availableMoves[i].y].layer = LayerMask.NameToLayer("Tile");
        }
        availableMoves.Clear(); //to clear list
    }
    // CheckMate
    private void CheckMate(int team)
    {
        DisplayVictory(team);
    }
    private void DisplayVictory(int winningTeam) // to display who win
    {
        victoryScreen.SetActive(true); // to show Vectory Screen UI game opject
         victoryScreen.transform.GetChild(winningTeam).gameObject.SetActive(true); // to show
text who win i do it in gui by adding text
    }
    public void OnRematchButton()
    {
        if (localGame)
        {
            NetRematch whiterematch = new NetRematch();
```

```
            whiterematch.teamId = 0;
            whiterematch.WantRematch = 1;
            Client.Instance.SendToServer(whiterematch);
            NetRematch blackrematch = new NetRematch();
            blackrematch.teamId = 1;
            blackrematch.WantRematch = 1;
            Client.Instance.SendToServer(blackrematch);
        }
        else
        {
            NetRematch rematch = new NetRematch();
            rematch.teamId = currentTeam;
            rematch.WantRematch = 1;
            Client.Instance.SendToServer(rematch);
        }
    }
    public void GameReset()
    {
        //UI Canvas
        rematchButton.interactable = true;
        rematchIndicator.transform.GetChild(0).gameObject.SetActive(false);
        rematchIndicator.transform.GetChild(1).gameObject.SetActive(false);
            victoryScreen.transform.GetChild(0).gameObject.SetActive(false); // to hide text if
winner team white
            victoryScreen.transform.GetChild(1).gameObject.SetActive(false); // to hide text if
winner team white
        victoryScreen.SetActive(false); // to hide VectoryScreen "Gameobject"
        //Field Reset
```

```
        currentlyDragging = null;
        availableMoves.Clear();
        movelist.Clear(); // updated to clear move list
        PlayerRematch[0] = PlayerRematch[1] = false;
        // Cleen up
        for (int x = 0; x < TITLE_COUNT_X; x++)
        {
            for (int y = 0; y < TITLE_COUNT_Y; y++)
            {
                if (chessPieces[x, y] != null)
                { Destroy(chessPieces[x, y].gameObject); }
                chessPieces[x, y] = null;
            }
        }
        for (int i = 0; i < DeadWhite.Count; i++)
        {
            Destroy(DeadWhite[i].gameObject);
        }
        for (int i = 0; i < DeadBlack.Count; i++)
        {
            Destroy(DeadBlack[i].gameObject);
        }
        DeadWhite.Clear(); DeadBlack.Clear();
        // spwan all pieces
        SpwanAllPieces();
        PositionAllPieces();
        isWhiteTurn = true;
    }
```

```
    public void OnMenuButton()
    {
        NetRematch rematch = new NetRematch();
        rematch.teamId = currentTeam;
        rematch.WantRematch = 0;
        Client.Instance.SendToServer(rematch);
        GameReset();
        GameUI.Instance.OnLeaveFromGameMenu();
        Invoke("ShutdownRelay", 1.0f);
         //reset Some Values
         playerCount = -1;
        currentTeam = -1;
    }
    //Special Moves
    //to check and and destroy enemy
    private void ProcessSpecialMove()
    {
        if (specialMove == SpecialMove.EnPassant)
        {
            var newMove = movelist[movelist.Count - 1];              // keep track position
            ChessPiece myPawn = chessPieces[newMove[1].x, newMove[1].y];
            var TargetPawnPosition = movelist[movelist.Count - 2];     // keep track position
                                    ChessPiece    enemyPwan    =    chessPieces[TargetPawnPosition[1].x,
TargetPawnPosition[1].y];
            if (myPawn.currentX == enemyPwan.currentX)
            {
                        if  (myPawn.currentY  ==  enemyPwan.currentY  -  1  ||  myPawn.currentY  ==
enemyPwan.currentY + 1)
```

```
            {
                if (enemyPwan.team == 0)
                {
                    DeadWhite.Add(enemyPwan);
                    DeadWhite.Add(enemyPwan);
                    enemyPwan.SetScale(Vector3.one * deathSize); // to Decrease he scale
                    enemyPwan.SetPosition(new Vector3(8 * tileSize, yOffset, -1 * tileSize)
                            - bounds + new Vector3(tileSize / 2, 0, tileSize / 2) + (Vector3.forward *
DeathSpacing) * DeadWhite.Count); // change postion dead white outside

                }
                else
                {
                    DeadBlack.Add(enemyPwan);
                    DeadBlack.Add(enemyPwan);
                    enemyPwan.SetScale(Vector3.one * deathSize); // to Decrease he scale
                    enemyPwan.SetPosition(new Vector3(8 * tileSize, yOffset, -1 * tileSize)
                            - bounds + new Vector3(tileSize / 2, 0, tileSize / 2) + (Vector3.forward *
DeathSpacing) * DeadBlack.Count); // change postion dead black outside
                }
                chessPieces[enemyPwan.currentX, enemyPwan.currentY] = null;
            }
        }
    }
    if(specialMove == SpecialMove.Promotion)
    {
        Vector2Int[] lastmove = movelist[movelist.Count - 1];
        ChessPiece TargetPwan = chessPieces[lastmove[1].x, lastmove[1].y];
```

```
        if(TargetPwan.type == ChessPieceType.pawn)
        {
            if(TargetPwan.team == 0 && lastmove[1].y == 7) // white team
            {
                ChessPiece newQueen = SpwanSinglePiece(ChessPieceType.Queen, 0); // spwan gameobject queen and material white
                newQueen.transform.position = chessPieces[lastmove[1].x, lastmove[1].y].transform.position;
                Destroy(chessPieces[lastmove[1].x, lastmove[1].y].gameObject); // destroy pwan
                chessPieces[lastmove[1].x, lastmove[1].y] = newQueen;
                PositionSinglePiece(lastmove[1].x, lastmove[1].y); // set new position for queen
            }
            if (TargetPwan.team == 1 && lastmove[1].y == 0) // black team
            {
                ChessPiece newQueen = SpwanSinglePiece(ChessPieceType.Queen, 1); // spwan gameobject queen and material black
                newQueen.transform.position = chessPieces[lastmove[1].x, lastmove[1].y].transform.position;
                Destroy(chessPieces[lastmove[1].x, lastmove[1].y].gameObject); // destroy pwan
                chessPieces[lastmove[1].x, lastmove[1].y] = newQueen;
                PositionSinglePiece(lastmove[1].x, lastmove[1].y); // set new position for queen
            }
        }
    }
    if (specialMove == SpecialMove.Castling)
    {
        var lastMove = movelist[movelist.Count - 1];
        //left rook
```

```
        if (lastMove[1].x == 2)
        {
            if (lastMove[1].y == 0) // white team
            {
                ChessPiece rook = chessPieces[0, 0];
                chessPieces[3, 0] = rook;
                PositionSinglePiece(3, 0);
                chessPieces[0, 0] = null;
            }
            else if (lastMove[1].y == 7) // black team
            {
                ChessPiece rook = chessPieces[0, 7];
                chessPieces[3, 7] = rook;
                PositionSinglePiece(3, 7);
                chessPieces[0, 7] = null;
            }
        }
        //right rook
        else if (lastMove[1].x == 6)
        {
            if (lastMove[1].y == 0) // white team
            {
                ChessPiece rook = chessPieces[7, 0];
                chessPieces[5, 0] = rook;
                PositionSinglePiece(5, 0);
                chessPieces[7, 0] = null;
            }
            else if (lastMove[1].y == 7) // black team
```

```
            {
                ChessPiece rook = chessPieces[7, 7];
                chessPieces[5, 7] = rook;
                PositionSinglePiece(5, 7);
                chessPieces[7, 7] = null;
            }
        }
    }
}
private void PreventCheck() // to check if prevent "Before Move"
{
    ChessPiece TargetKing = null;
    for (int x = 0; x < TITLE_COUNT_X; x++)
    {
        for (int y = 0; y < TITLE_COUNT_Y; y++)
        {
            if (chessPieces[x, y] != null)
                if (chessPieces[x,y].type == ChessPieceType.King)
                {
                if (chessPieces[x, y].team == currentlyDragging.team)
                    TargetKing = chessPieces[x, y];
                }
        }
          // Since we sending ref availablemoves , we will be deleting moves that are putting us
in check
    }
    SimulateMoveForSinglePiece(currentlyDragging, ref availableMoves, TargetKing);
}
```

```
                private void SimulateMoveForSinglePiece(ChessPiece cp , ref List<Vector2Int>moves,ChessPiece targetKing ) // to Simulate move
  {
    // Save the current values  , to reset after function call
    int actualX = cp.currentX;
    int actualY = cp.currentY;
    List<Vector2Int> movesToRemove = new List<Vector2Int>();
    // going through all the moves , simualte them and check if we are in check
    for (int i = 0; i < moves.Count; i++)
    {
      int simX = moves[i].x;
      int simY = moves[i].y;
                    Vector2Int kingPositionThisSim = new Vector2Int(targetKing.currentX, targetKing.currentY);
      // Did we simulate the king move
      if(cp.type == ChessPieceType.King)
      {
        kingPositionThisSim = new Vector2Int(simX, simY);
      }
      // copy the [][]array and not reference
        ChessPiece[,] simulation = new ChessPiece[TITLE_COUNT_X,TITLE_COUNT_Y]; // to simulate The board
      List<ChessPiece> simAttackingPieces = new List<ChessPiece>(); // another team
      for (int x = 0; x < TITLE_COUNT_X; x++)
      {
        for (int y = 0; y < TITLE_COUNT_Y; y++)
        {
          if (chessPieces[x, y] != null)
```

```
                {
                    simulation[x, y] = chessPieces[x, y];
                    if (simulation[x, y].team != cp.team)
                    {
                        simAttackingPieces.Add(simulation[x, y]);
                    }
                }
            }
        }
        //Simulate that move
        simulation[actualX, actualY] = null;
        cp.currentX = simX;
        cp.currentY = simY;
        simulation[simX, simY] = cp;
        // Did one of the piece got taken down during our simulation
            var deadPiece = simAttackingPieces.Find(c => c.currentX == simX && c.currentY ==
simY); //lambda expression
        if (deadPiece != null)
        {
            simAttackingPieces.Remove(deadPiece);
        }
        // Get All Simulated Attacking pieces Moves
        List<Vector2Int> simMoves = new List<Vector2Int>();
        for (int a = 0; a < simAttackingPieces.Count; a++)
        {
                    var pieceMoves = simAttackingPieces[a].GetAvailableMove(ref simulation,
TITLE_COUNT_X, TITLE_COUNT_Y);
            for (int b = 0; b < pieceMoves.Count; b++)
```

```
                simMoves.Add(pieceMoves[b]);
            }
            // If king is trouble ? then remove the move
            if(ContainsValidMove(ref simMoves, kingPositionThisSim))
            {
                movesToRemove.Add(moves[i]);
            }
            // Restore the actual chesspiecs "cp " data
            cp.currentX = actualX;
            cp.currentY = actualY;
        }
        // Remove from the current available move list
        for (int i = 0; i < movesToRemove.Count; i++)
        {
            moves.Remove(movesToRemove[i]);
        }
    }
    //
    private bool CheckForCheckMate() // after move
    {
        var lastMove = movelist[movelist.Count - 1];
        int targetTeam = (chessPieces[lastMove[1].x, lastMove[1].y].team == 0) ? 1 : 0;
        List<ChessPiece> attackingPieces = new List<ChessPiece>();
        List<ChessPiece> defendingPieces = new List<ChessPiece>();
        ChessPiece TargetKing = null;
        for (int x = 0; x < TITLE_COUNT_X; x++)
        {
            for (int y = 0; y < TITLE_COUNT_Y; y++)
```

```
        {
            if (chessPieces[x, y] != null)
            {
                if (chessPieces[x, y].team == targetTeam)
                {
                    defendingPieces.Add(chessPieces[x, y]);
                    if (chessPieces[x, y].type == ChessPieceType.King)
                    {
                        TargetKing = chessPieces[x, y];
                    }
                }
                else
                {
                    attackingPieces.Add(chessPieces[x, y]);
                }
            }
        }
    }
    // is the king  attacked rightnow ?
    List<Vector2Int> currentAvailableMoves = new List<Vector2Int>();
    for (int i = 0; i < attackingPieces.Count; i++)
    {
                    var pieceMoves = attackingPieces[i].GetAvailableMove(ref chessPieces,
TITLE_COUNT_X, TITLE_COUNT_Y);
        for (int b = 0; b < pieceMoves.Count; b++)
            currentAvailableMoves.Add(pieceMoves[b]);
    }
    // Are we in check right now ?
```

```
        if(ContainsValidMove(ref currentAvailableMoves , new Vector2Int(TargetKing.currentX , TargetKing.currentY)))
        {
            //King in under attack , can we move something to help thim ?
            for (int i = 0; i < defendingPieces.Count; i++)
            {
                        List<Vector2Int> defendingMoves = defendingPieces[i].GetAvailableMove(ref chessPieces,TITLE_COUNT_X,TITLE_COUNT_Y);
                    //Since we are sending ref AvailableMoves , we will be deleting moves that are putting us in check
                SimulateMoveForSinglePiece(defendingPieces[i],ref defendingMoves , TargetKing);
                if (defendingMoves.Count != 0)
                {
                    return false;
                }
            }
            return true; // CheckMate exit
        }
        return false;
    }
    //Operations
    private bool ContainsValidMove(ref List<Vector2Int> moves, Vector2Int position)
    {
        for (int i = 0; i < moves.Count; i++)
            if (moves[i].x == position.x && moves[i].y == position.y)
                return true;
        return false;
    }
```

```
        private void MoveTo(int originalX, int originalY, int x, int y)
        {
            ChessPiece cp = chessPieces[originalX, originalY];
            Vector2Int previousPosition = new Vector2Int(originalX, originalY);
            // Is There another piece on the target position
            if (chessPieces[x, y] != null)
            {
                ChessPiece otherCp = chessPieces[x, y];
                if (cp.team == otherCp.team)
                    return ;
                //if its the enemy team
                if (otherCp.team == 0) // white team
                {
                    if (otherCp.type == ChessPieceType.King) // Win Condition
                    {
                        CheckMate(1); // black team
                    }
                    DeadWhite.Add(otherCp);
                    otherCp.SetScale(Vector3.one * deathSize); // to Decrease he scale
                    otherCp.SetPosition(new Vector3(8 * tileSize, yOffset, -1 * tileSize)
                                - bounds + new Vector3(tileSize / 2, 0, tileSize / 2) + (Vector3.forward *
DeathSpacing) * DeadWhite.Count); // change postion dead white outside
                }
                else if (otherCp.team == 1) // black team
                {
                    if (otherCp.type == ChessPieceType.King) // Win Condition
                    {
                        CheckMate(0); // White team
```

```
            }
            DeadBlack.Add(otherCp);
            otherCp.SetScale(Vector3.one * deathSize); // to Decrease he scale
            otherCp.SetPosition(new Vector3(-1 * tileSize, yOffset, 8 * tileSize)
                        - bounds + new Vector3(tileSize / 2, 0, tileSize / 2) + (Vector3.back *
DeathSpacing) * DeadBlack.Count); // change postion dead black outside
          }
        }
        chessPieces[x, y] = cp;
        chessPieces[previousPosition.x, previousPosition.y] = null;
        PositionSinglePiece(x, y);
        isWhiteTurn = !isWhiteTurn;
      if (localGame)
      {
        currentTeam = (currentTeam == 0) ? 1 : 0; // fix local game to swaping between 0,1;
      }
         movelist.Add(new Vector2Int[] { previousPosition, new Vector2Int(x, y) }); // store prev
postion and new positon to make special moves
        ProcessSpecialMove();  // do check and do to speacial moves
      if (currentlyDragging)
      {
        currentlyDragging = null;
      }
        RemoveHighlightTiles();
      if (CheckForCheckMate())
      {
        CheckMate(cp.team); // cp Current team
      }
```

```
            return ;
        }
        private Vector2Int LookupTileIndex(GameObject hitInfo)
        {
            for (int x = 0; x < TITLE_COUNT_X; x++)
                for (int y = 0; y < TITLE_COUNT_Y; y++)
                    if (tiles[x, y] == hitInfo)
                        return new Vector2Int(x, y);
            return -Vector2Int.one; //-1-1 Invalid
        }
    // Networking
    #region
    private void RegisterEvents()
    {
        NetUtility.S_WELCOME += OnWelcomeServer;
        NetUtility.S_MAKE_MOVE += OnMakeMoveServer;
        NetUtility.S_REMATCH += OnRematchServer;
        NetUtility.C_MAKE_MOVE += OnMakeMoveClient;
        NetUtility.C_WELCOME += OnWelcomeClient;
        NetUtility.C_START_GAME += OnStartGameClient;
        NetUtility.C_REMATCH += OnRematchClient;
        GameUI.Instance.SetlocalGame += OnSetLocalGame;
    }
    private void UnRegisterEvents()
    {
        NetUtility.S_WELCOME -= OnWelcomeServer;
        NetUtility.S_MAKE_MOVE -= OnMakeMoveServer;
        NetUtility.S_REMATCH -= OnRematchServer;
```

```
    NetUtility.C_MAKE_MOVE -= OnMakeMoveClient;
    NetUtility.C_WELCOME -= OnWelcomeClient;
    NetUtility.C_START_GAME -= OnStartGameClient;
    NetUtility.C_REMATCH -= OnRematchClient;
    GameUI.Instance.SetlocalGame -= OnSetLocalGame;
}
//Server
private void OnWelcomeServer(NetMessages msg, NetworkConnection cnn)
{
    // Client Has Connected  , Assign The Team and return the message back to him
    NetWelcome nw = msg as NetWelcome;
    //Assign Team
    nw.AssignedTeam = ++playerCount;
    //Return Back to The Client
    Server.Instance.SendToClient(cnn,nw);
    //If Full, start The Game
    if(playerCount == 1)
    {
        Server.Instance.Broadcast(new NetStartGame());
    }
}
//Client
private void OnWelcomeClient(NetMessages msg)
{
    // Receive the Connection Message
    NetWelcome nw = msg as NetWelcome;
    //Assign Team Team
    currentTeam = nw.AssignedTeam;
```

```
        Debug.Log($"My Assign Team is {nw.AssignedTeam}");
        if(localGame && currentTeam == 0)
        {
            Server.Instance.Broadcast(new NetStartGame());
        }
    }
    private void OnStartGameClient(NetMessages msg)
    {
        //Change The Camera
                GameUI.Instance.ChangeCamera((currentTeam == 0) ? cameraAngle.whiteTeam :
cameraAngle.blackTeam);
        }
    private void OnMakeMoveServer(NetMessages msg, NetworkConnection cnn)
    {
        // Receive the message , broadcast it back
        NetMakeMove makemove = msg as NetMakeMove;
        //This is Where you could do some validation check later
        //-- code
        // Receive , and just broadcast  it back
        Server.Instance.Broadcast(makemove);
    }
    private void OnRematchServer(NetMessages msg, NetworkConnection cnn)
    {
        Server.Instance.Broadcast(msg);
    }
    private void OnMakeMoveClient(NetMessages msg)
    {
        NetMakeMove makemove = msg as NetMakeMove;
```

```
                                    Debug.Log($"makemove : {makemove.teamId}
{makemove.originalX}:{makemove.originalY} --->{makemove.destinationX}
{makemove.destinationY}");
        if (makemove.teamId != currentTeam)
        {
            ChessPiece target = chessPieces[makemove.originalX, makemove.originalY];
                                        availableMoves = target.GetAvailableMove(ref
chessPieces,TITLE_COUNT_X,TITLE_COUNT_Y);
                    specialMove = target.GetSpecialMoves(ref chessPieces, ref movelist, ref
availableMoves);
                MoveTo(makemove.originalX, makemove.originalY, makemove.destinationX,
makemove.destinationY);
        }
    }
    private void OnRematchClient(NetMessages msg)
    {
        // Receive the connection message
        NetRematch rematch = msg as NetRematch;
        // Set the boolean for rematch
        PlayerRematch[rematch.teamId] = rematch.WantRematch == 1;
        // Active the piece of UI
        if(rematch.teamId != currentTeam)
        {
                    rematchIndicator.transform.GetChild((rematch.WantRematch == 1) ? 0 :
1).gameObject.SetActive(true);
            if(rematch.WantRematch != 1)
            {
                rematchButton.interactable = false;
```

```
            }
        }
        // if both wants to rematch
        if (PlayerRematch[0] && PlayerRematch[1])
        {
            GameReset();
        }
    }
    //
    private void ShutdownRelay()
    {
        Client.Instance.Shutdown();
        Server.Instance.Shutdown();
    }
        private void OnSetLocalGame(bool value)
    {
        playerCount = -1;
        currentTeam = -1;
        localGame = value;
    }
    #endregion
```

4.5.3 Class GAMEUI Code:

```
using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
```

```
using UnityEngine;
public enum cameraAngle
{
    menu = 0,
    whiteTeam = 1,
    blackTeam =2
}
public class GameUI : MonoBehaviour
{
    // Start is called before the first frame update
    public static GameUI Instance { set; get; }
    public Server server;
    public Client client;
    [SerializeField] private Animator menuAnimator;
     [SerializeField] private TMP_InputField addressInput; // text mesh pro inputfield who in
GUI
    [SerializeField] private GameObject[] cameraAngles;
    // Camera
    public Action<bool> SetlocalGame;
    public void ChangeCamera(cameraAngle index)
    {
        for (int i = 0; i < cameraAngles.Length; i++)
        {
            cameraAngles[i].SetActive(false);
            // to off all cameras
        }
        // TO ENABLE CAMERA INDEX
        cameraAngles[(int)index].SetActive(true);
```

```
    }
    private void Awake()
    {
        Instance = this;
        RegisterEvents();
    }
    public void OnlocalGameButton()
    {
        menuAnimator.SetTrigger("InGame");
        SetlocalGame?.Invoke(true);
        server.Init(8007); // server initialize at port 8007
        client.Init("127.0.0.1", 8007); // local host client
    }
    public void OnOnlineGameButton()
    {
        menuAnimator.SetTrigger("OnlineMenu");
    }
    public void OnOnlineHostButton()
    {
        SetlocalGame?.Invoke(false);
        server.Init(8007); // server initialize at port 8007
        client.Init("127.0.0.1",8007); // local host client
        menuAnimator.SetTrigger("HostMenu");
    }
    public void OnOnlineConnectButton()
    {
        SetlocalGame?.Invoke(false);
        client.Init(addressInput.text, 8007);  // input address form InputField
```

```
    }
    public void OnOnlineBackButton()
    {
        menuAnimator.SetTrigger("StartMenu");
    }
    public void OnHostBackButton()
    {
        server.Shutdown();
        client.Shutdown();
        menuAnimator.SetTrigger("OnlineMenu");
    }
    public void OnLeaveFromGameMenu()
    {
        ChangeCamera(cameraAngle.menu);
        menuAnimator.SetTrigger("StartMenu");
    }
    #region
    private void RegisterEvents()
    {
        NetUtility.C_START_GAME += OnStartGameClient;
    }
    private void UnRegisterEvents()
    {
        NetUtility.C_START_GAME -= OnStartGameClient;
    }
    private void OnStartGameClient(NetMessages obj)
    {
        menuAnimator.SetTrigger("InGame");
```

```
    }
    #endregion
}
```

4.5.4 Class ChessPiece Code:

```
using System.Collections.Generic;
using UnityEngine;
public enum ChessPieceType
{
    none = 0,
    pawn = 1,
    Rook =2,
    Knight = 3,
    Bishop = 4,
    Queen = 5,
    King = 6,
}
public class ChessPiece : MonoBehaviour
{
    public int team;
    public int currentX;
    public int currentY;
    public ChessPieceType type;

    private Vector3 desiredPosition;
    private Vector3 desiredScale = Vector3.one; // vector3 defuat value =0 we change it to 1
```

```
    private void Start()
    {
        // to fix black Knight rotation in start game while he spwaning
        //     white team no change       black team rotation 180 dagree
         transform.rotation = Quaternion.Euler((team == 0) ? Vector3.zero : new Vector3(0, 180,
0));
    }

    private void Update()
    {
        transform.position = Vector3.Lerp(transform.position, desiredPosition, Time.deltaTime *
10);
        transform.localScale = Vector3.Lerp(transform.localScale, desiredScale, Time.deltaTime *
10);
    }
     public virtual List<Vector2Int> GetAvailableMove(ref ChessPiece[,] board ,int tileCountX,int
tileCountY)//to make hightlight tiles
    {
        List<Vector2Int> r = new List<Vector2Int>();

        //test hightlight {center postion } and change it later
        r.Add(new Vector2Int(3, 3));
        r.Add(new Vector2Int(3, 4));
        r.Add(new Vector2Int(4, 3));
        r.Add(new Vector2Int(4, 4));
        return r;
    }
```

```
    public virtual SpecialMove GetSpecialMoves(ref ChessPiece[,] board , ref List<Vector2Int[]>
movelist , ref List<Vector2Int> availableMoves)
    {
        return SpecialMove.None;
    }

        public virtual void SetPosition(Vector3 position , bool force = false) // to movement
Smooth
    {
        desiredPosition = position;
        if (force)
            transform.position = desiredPosition;
    }
    public virtual void SetScale(Vector3 scale, bool force = false) // to movement Smooth
    {
        desiredScale = scale;
        if (force)
            transform.localScale = desiredScale;
    }
}
```

4.5.5 Class Bishop Code:

```
using System.Collections.Generic;
using UnityEngine;

public class Bishop : ChessPiece
```

```
{
    public override List<Vector2Int> GetAvailableMove(ref ChessPiece[,] board, int tileCountX,
int tileCountY)
    {
        List<Vector2Int> r = new List<Vector2Int>();
        // TOP RIGHT MOVEMENT
        for (int x = currentX+1 , y = currentY + 1; x < tileCountX && y <tileCountY; x++,y++)
        {
            if (board[x, y] == null)
            {
                r.Add(new Vector2Int(x, y));
            }
            else
            {
                if (board[x, y].team != team)
                {
                    r.Add(new Vector2Int(x, y));
                }
             break;
            }
        }
        // TOP LEFT MOVEMENT
        for (int x = currentX - 1, y = currentY + 1; x >= 0 && y < tileCountY; x--, y++)
        {
            if (board[x, y] == null)
            {
                r.Add(new Vector2Int(x, y));
            }
```

```
        else
        {
            if (board[x, y].team != team)
            {
                r.Add(new Vector2Int(x, y));
            }
         break;
        }
    }
    // Bottom Right MOVEMENT
    for (int x = currentX + 1, y = currentY - 1; x < tileCountX && y  >= 0; x++, y--)
    {
        if (board[x, y] == null)
        {
            r.Add(new Vector2Int(x, y));
        }
        else
        {
            if (board[x, y].team != team)
            {
                r.Add(new Vector2Int(x, y));
            }
        break;
        }
    }
    // Bottom LEFT MOVEMENT
    for (int x = currentX - 1, y = currentY - 1; x >= 0 && y >= 0; x--, y--)
    {
```

```
            if (board[x, y] == null)
            {
                r.Add(new Vector2Int(x, y));
            }
            else
            {
                if (board[x, y].team != team)
                {
                    r.Add(new Vector2Int(x, y));
                }
                break;
            }
        }
        return r;
    }
}
```

4.5.6 Class King Code:

```
using System.Collections.Generic;
using UnityEngine;
public class King : ChessPiece
{
    public override List<Vector2Int> GetAvailableMove(ref ChessPiece[,] board, int tileCountX,
int tileCountY)
    {
        List<Vector2Int> r = new List<Vector2Int>();
```

```
// MOVEMENT RIGHT
if (currentX + 1 < tileCountX)
{
   //RIGHT
   if (board[currentX + 1, currentY] == null)
   {
      r.Add(new Vector2Int(currentX + 1, currentY));
   }
   else if (board[currentX + 1, currentY].team != team)
   {
      r.Add(new Vector2Int(currentX + 1, currentY));
   }
   //TOP RIGHT
   if (currentY + 1 < tileCountY)
   {
      //RIGHT
      if (board[currentX + 1, currentY + 1] == null)
      {
         r.Add(new Vector2Int(currentX + 1, currentY + 1));
      }
      else if (board[currentX + 1, currentY + 1].team != team)
      {
         r.Add(new Vector2Int(currentX + 1, currentY + 1));
      }
   }
   //Bottom RIGHT
   if (currentY - 1 > 0)
   {
```

```
            //RIGHT
            if (board[currentX + 1, currentY - 1] == null)
            {
                r.Add(new Vector2Int(currentX + 1, currentY - 1));
            }
            else if (board[currentX + 1, currentY - 1].team != team)
            {
                r.Add(new Vector2Int(currentX + 1, currentY - 1));
            }
        }
    }
    // MOVEMENT LEFT
    if (currentX - 1 >= 0)
    {
        //LEFT
        if (board[currentX - 1, currentY] == null)
        {
            r.Add(new Vector2Int(currentX - 1, currentY));
        }
        else if (board[currentX - 1, currentY].team != team)
        {
            r.Add(new Vector2Int(currentX - 1, currentY));
        }
        //TOP LEFT
        if (currentY + 1 < tileCountY)
        {

            if (board[currentX - 1, currentY + 1] == null)
```

```
            {
                r.Add(new Vector2Int(currentX - 1, currentY + 1));
            }
            else if (board[currentX - 1, currentY + 1].team != team)
            {
                r.Add(new Vector2Int(currentX - 1, currentY + 1));
            }
        }
        //Bottom LEFT
        if (currentY - 1 > 0)
        {
            //Left
            if (board[currentX - 1, currentY - 1] == null)
            {
                r.Add(new Vector2Int(currentX - 1, currentY - 1));
            }
            else if (board[currentX - 1, currentY - 1].team != team)
            {
                r.Add(new Vector2Int(currentX - 1, currentY - 1));
            }
        }
    }
    // MOVEMENT UP
    if (currentY + 1 < tileCountY)
        if (board[currentX, currentY + 1] == null || board[currentX,currentY+1].team != team)
        {
            r.Add(new Vector2Int(currentX, currentY + 1));
        }
```

```
        // MOVEMENT DOWN
        if (currentY - 1 >= 0)
            if (board[currentX, currentY - 1] == null || board[currentX, currentY - 1].team != team)
            {
                r.Add(new Vector2Int(currentX, currentY - 1));
            }
        return r;
    } // Movement King


            public override SpecialMove GetSpecialMoves(ref ChessPiece[,] board, ref
List<Vector2Int[]> movelist, ref List<Vector2Int> availableMoves)
    {
        SpecialMove r = SpecialMove.None;
         var kingMove = movelist.Find(m => m[0].x == 4 && m[0].y == ((team == 0) ? 0 : 7));   // 4
x-axis king  If White  y-axis = 0 else = 7 //lambda expression to check movement
         var LeftRook = movelist.Find(m => m[0].x == 0 && m[0].y == ((team == 0) ? 0 : 7));   // 0
x-axis rook If White  y-axis = 0 else = 7
         var RightRook = movelist.Find(m => m[0].x == 7 && m[0].y == ((team == 0) ? 0 : 7));   // 7
x-axis rook If White  y-axis = 0 else = 7
        if (kingMove == null && currentX == 4)
        {
            //white team
            if(team == 0)
            {
                //left rook
                if (LeftRook == null)
                    if(board[0,0].type == ChessPieceType.Rook)
```

```
            if(board[0,0].team==0)
                if(board[3,0]==null)
                    if(board[2,0]==null)
                        if (board[1, 0] == null)
                        {
                            availableMoves.Add(new Vector2Int(2, 0)); // movment king
                            r = SpecialMove.Castling;
                        }
    //left rook
    if (RightRook == null)
        if (board[7, 0].type == ChessPieceType.Rook)
            if (board[7, 0].team == 0)
                if (board[5, 0] == null)
                    if (board[6, 0] == null)
                    {
                        availableMoves.Add(new Vector2Int(6, 0)); // movment king
                        r = SpecialMove.Castling;
                    }
}
else
{
    //left rook
    if (LeftRook == null)
        if (board[0, 7].type == ChessPieceType.Rook)
            if (board[0, 7].team == 1)
                if (board[3, 7] == null)
                    if (board[2, 7] == null)
                        if (board[1, 7] == null)
```

```
                    {
                        availableMoves.Add(new Vector2Int(2, 7)); // movment king
                        r = SpecialMove.Castling;
                    }
            //left rook
            if (RightRook == null)
                if (board[7, 7].type == ChessPieceType.Rook)
                    if (board[7, 7].team == 1)
                        if (board[5, 7] == null)
                            if (board[6, 7] == null)
                            {
                                availableMoves.Add(new Vector2Int(6, 7)); // movment king
                                r = SpecialMove.Castling;
                            }
        }
    }
    return r;
  }
}
```

4.5.7 Class Knight Code:

```
using System.Collections.Generic;
using UnityEngine;


public class Knight : ChessPiece
{
```

```
    public override List<Vector2Int> GetAvailableMove(ref ChessPiece[,] board, int tileCountX,
int tileCountY)
    {
        List<Vector2Int> r = new List<Vector2Int>();
        // TOP RIGHT MOVEMENT
        int x = currentX + 1;
        int y = currentY + 2;
        if(x < tileCountX && y < tileCountY)
        {
            if (board[x, y] == null || board[x,y].team != team)
            {
                r.Add(new Vector2Int(x, y));
            }
        }
        // TOP RIGHT MOVEMENT
        x = currentX + 2;
         y = currentY + 1;
        if (x < tileCountX && y < tileCountY)
        {
            if (board[x, y] == null || board[x, y].team != team)
            {
                r.Add(new Vector2Int(x, y));
            }
        }
        // TOP LEFT MOVEMENT
        x = currentX - 1;
        y = currentY + 2;
        if (x >= 0 && y < tileCountY)
```

```
        {
            if (board[x, y] == null || board[x, y].team != team)
            {
                r.Add(new Vector2Int(x, y));
            }
        }
        // TOP LEFT MOVEMENT
        x = currentX - 2;
        y = currentY + 1;
        if (x >= 0 && y < tileCountY)
        {
            if (board[x, y] == null || board[x, y].team != team)
            {
                r.Add(new Vector2Int(x, y));
            }
        }
        // Bottom RIGHT MOVEMENT
         x = currentX + 1;
         y = currentY - 2;
        if (x < tileCountX && y >= 0)
        {
            if (board[x, y] == null || board[x, y].team != team)
            {
                r.Add(new Vector2Int(x, y));
            }
        }
        // Bottom RIGHT MOVEMENT
        x = currentX + 2;
```

```
y = currentY - 1;
if (x < tileCountX && y >= 0)
{
   if (board[x, y] == null || board[x, y].team != team)
   {
      r.Add(new Vector2Int(x, y));
   }
}
// Bottom LEFT MOVEMENT
x = currentX - 1;
y = currentY - 2;
if (x >= 0 && y >= 0)
{
   if (board[x, y] == null || board[x, y].team != team)
   {
      r.Add(new Vector2Int(x, y));
   }
}
// Bottom LEFT MOVEMENT
x = currentX - 2;
y = currentY - 1;
if (x >= 0 && y >= 0)
{
   if (board[x, y] == null || board[x, y].team != team)
   {
      r.Add(new Vector2Int(x, y));
   }
}
```

```
        return r;
    }
}
```

4.5.8 Class Pawn Code:

```
using System.Collections.Generic;
using UnityEngine;
public class Pawn : ChessPiece
{
    public override List<Vector2Int> GetAvailableMove(ref ChessPiece[,] board , int tileCountX,
int tileCountY)
    {
        // Rule for Pawn
        List<Vector2Int> r = new List<Vector2Int>();
        int direction = (team == 0) ? 1 : -1;  // One in Front
        // one step in front
        if (board[currentX, currentY + direction] == null)
            r.Add(new Vector2Int(currentX, currentY + direction)); // to move one step
        // two steps in front if pwan is 1st movement
        if (board[currentX, currentY + direction] == null)
        {
            //WhiteTeam                    0      Y     +  1*2
            if(team == 0 && currentY == 1 && board[currentX,currentY+direction * 2] == null)
                r.Add(new Vector2Int(currentX, currentY + direction *2));
            //WhiteTeam                    0      Y     +  1*2          empty
            if (team == 1 && currentY == 6 && board[currentX, currentY + direction * 2] == null)
```

```
            r.Add(new Vector2Int(currentX, currentY + direction * 2));


        }
        //Kill Move
        if(currentX !=tileCountX-1)
        {    //      x+1     y       1                       //check if enemy team
                if(board[currentX+1,currentY+direction]!=null && board[currentX + 1, currentY +
direction].team != team)
            {
                r.Add(new Vector2Int(currentX + 1, currentY + direction));
            }
        }
        if (currentX != 0)
        {    //      x-1          y       1                  //check if enemy team
             if (board[currentX - 1, currentY + direction] != null && board[currentX - 1, currentY +
direction].team != team)
            {
                r.Add(new Vector2Int(currentX - 1, currentY + direction));
            }
        }
        return r;
    }
             public override SpecialMove GetSpecialMoves(ref ChessPiece[,] board, ref
List<Vector2Int[]> movelist, ref List<Vector2Int> availableMoves)
    {
        int direction = (team == 0) ? 1 : -1;
         if((team ==0 &&currentY==6)|| (team == 1 && currentY == 1)) // check if pawn if need
promotion
```

```
        {
            return SpecialMove.Promotion;
        }
        //En Passant movement
        if (movelist.Count > 0)
        {
            Vector2Int[] lasttMove = movelist[movelist.Count - 1];
            // if its 1st move and type pwan
              if(board[ lasttMove[1].x,lasttMove[1].y ].type == ChessPieceType.pawn) // if the last
piece moved is pwan
            {
                 if (Mathf.Abs(lasttMove[0].y - lasttMove[1].y ) == 2) // if the last move was a +2 in
either direction
                {
                     if(board[lasttMove[1].x, lasttMove[1].y].team != team) // if the move was from
another team
                    {
                        if (lasttMove[1].y == currentY)  // if both pawns are one in same Y-axis
                        {
                            if (lasttMove[1].x == currentX - 1) // landed left
                            {
                                availableMoves.Add(new Vector2Int(currentX - 1, currentY + direction));
                                return SpecialMove.EnPassant;
                            }
                            if (lasttMove[1].x == currentX + 1) // landed right
                            {
                                availableMoves.Add(new Vector2Int(currentX + 1, currentY + direction));
                                return SpecialMove.EnPassant;
```

```
                    }
                }
            }
        }
    }
    return SpecialMove.None;
  }
}
```

4.5.9 Class Queen Code:

```
using System.Collections.Generic;
using UnityEngine;
public class Queen : ChessPiece
{
   // we copy code from Rook.cs and Bishop
    public override List<Vector2Int> GetAvailableMove(ref ChessPiece[,] board, int tileCountX,
int tileCountY)
   {
      List<Vector2Int> r = new List<Vector2Int>();
      //Down Diraction movement
      for (int i = currentY - 1; i >= 0; i--)
      {
         //if not empty
         if (board[currentX, i] == null)
         {
```

```
            r.Add(new Vector2Int(currentX, i));
        }
        if (board[currentX, i] != null)
        {
            if (board[currentX, i].team != team)
            {
                r.Add(new Vector2Int(currentX, i));
            }
            break;
        }
    }
    //Up Diraction movement
    for (int i = currentY + 1; i < tileCountY; i++)
    {
        //if not empty
        if (board[currentX, i] == null)
        {
            r.Add(new Vector2Int(currentX, i));
        }
        if (board[currentX, i] != null)
        {
            if (board[currentX, i].team != team)
            {
                r.Add(new Vector2Int(currentX, i));
            }
            break;
        }
    }
```

```
        //Left Diraction movement
        for (int i = currentX - 1; i >= 0; i--)
        {
            //if not empty
            if (board[i, currentY] == null)
            {
                r.Add(new Vector2Int(i, currentY));
            }
            if (board[i, currentY] != null)
            {
                if (board[i, currentY].team != team)
                {
                    r.Add(new Vector2Int(i, currentY));
                }
                break;
            }
        }
        //Right Diraction movement
        for (int i = currentX + 1; i < tileCountX; i++)
        {
            //if not empty
            if (board[i, currentY] == null)
            {
                r.Add(new Vector2Int(i, currentY));
            }
            if (board[i, currentY] != null)
            {
                if (board[i, currentY].team != team)
```

```
            {
                r.Add(new Vector2Int(i, currentY));
            }
            break;
        }
    }
    // we no need special kill like pwan ;)
    // TOP RIGHT MOVEMENT
    for (int x = currentX + 1, y = currentY + 1; x < tileCountX && y < tileCountY; x++, y++)
    {
        if (board[x, y] == null)
        {
            r.Add(new Vector2Int(x, y));
        }
        else
        {
            if (board[x, y].team != team)
            {
                r.Add(new Vector2Int(x, y));
            }
            break;
        }
    }
    // TOP LEFT MOVEMENT
    for (int x = currentX - 1, y = currentY + 1; x >= 0 && y < tileCountY; x--, y++)
    {
        if (board[x, y] == null)
        {
```

```
            r.Add(new Vector2Int(x, y));
        }
        else
        {
            if (board[x, y].team != team)
            {
                r.Add(new Vector2Int(x, y));
            }
            break;
        }
    }
    // Bottom Right MOVEMENT
    for (int x = currentX + 1, y = currentY - 1; x < tileCountX && y >= 0; x++, y--)
    {
        if (board[x, y] == null)
        {
            r.Add(new Vector2Int(x, y));
        }
        else
        {
            if (board[x, y].team != team)
            {
                r.Add(new Vector2Int(x, y));
            }
            break;
        }
    }
    // Bottom LEFT MOVEMENT
```

```
        for (int x = currentX - 1, y = currentY - 1; x >= 0 && y >= 0; x--, y--)
        {
            if (board[x, y] == null)
            {
                r.Add(new Vector2Int(x, y));
            }
            else
            {
                if (board[x, y].team != team)
                {
                    r.Add(new Vector2Int(x, y));
                }
                break;
            }
        }
        return r;
    }
}
```

4.5.10 Class Rook Code:

```
using System.Collections.Generic;
using UnityEngine;

public class Rook : ChessPiece
{
```

```
    public override List<Vector2Int> GetAvailableMove(ref ChessPiece[,] board, int tileCountX,
int tileCountY)
    {
        List<Vector2Int> r = new List<Vector2Int>();

        //Down Diraction movement
        for (int i = currentY - 1; i >=0 ; i--)
        {
            //if not empty
            if (board[currentX, i] == null)
            {
                r.Add(new Vector2Int(currentX, i));
            }
            if (board[currentX, i] != null)
            {
                if (board[currentX, i].team != team)
                {
                    r.Add(new Vector2Int(currentX, i));
                }
                break;
            }
        }
        //Up Diraction movement
        for (int i = currentY + 1; i < tileCountY; i++)
        {
            //if not empty
            if (board[currentX, i] == null)
            {
```

```
            r.Add(new Vector2Int(currentX, i));
        }
        if (board[currentX, i] != null)
        {
            if (board[currentX, i].team != team)
            {
                r.Add(new Vector2Int(currentX, i));
            }
            break;
        }
    }
    //Left Diraction movement
    for (int i = currentX - 1; i >= 0; i--)
    {
        //if not empty
        if (board[i, currentY] == null)
        {
            r.Add(new Vector2Int(i, currentY ));
        }
        if (board[i, currentY] != null)
        {
            if (board[i, currentY].team != team)
            {
                r.Add(new Vector2Int(i, currentY));
            }
            break;
        }
    }
```

```
        //Right Diraction movement
        for (int i = currentX + 1; i < tileCountX; i++)
        {
            //if not empty
            if (board[i, currentY] == null)
            {
                r.Add(new Vector2Int(i, currentY));
            }
            if (board[i, currentY] != null)
            {
                if (board[i, currentY].team != team)
                {
                    r.Add(new Vector2Int(i, currentY));
                }
                break;
            }
        }
        // we no need special kill like pwan ;)
        return r;

    }
}
```

4.5.11 Class NetUtility Code:

```
using System;
using Unity.Networking.Transport;
```

```
using UnityEngine;
public enum OpCode
{
   KEEP_ALIVE = 1,
   WELCOME = 2,
   START_GAME = 3,
   MAKE_MOVE = 4,
   REMATCH = 5
}
public static class NetUtility
{
   // Net messages
   public static Action<NetMessages> C_KEEP_ALIVE; // Receive massages on client
   public static Action<NetMessages> C_WELCOME;    // Receive massages on client
   public static Action<NetMessages> C_START_GAME; // Receive massages on client
   public static Action<NetMessages> C_MAKE_MOVE;  // Receive massages on client
   public static Action<NetMessages> C_REMATCH;    // Receive massages on client
        public static Action<NetMessages,NetworkConnection> S_KEEP_ALIVE;    // Receive
massages on Server
       public static Action<NetMessages, NetworkConnection> S_WELCOME;      // Receive
massages on Server
       public static Action<NetMessages, NetworkConnection> S_START_GAME; // Receive
massages on Server
       public static Action<NetMessages, NetworkConnection> S_MAKE_MOVE;   // Receive
massages on Server
        public static Action<NetMessages, NetworkConnection> S_REMATCH;     // Receive
massages on Server
   //Methods
```

```
        public static void OnData(DataStreamReader stream ,NetworkConnection cnn, Server
server = null)
    {
        NetMessages msg = null;
        var opCode = (OpCode)stream.ReadByte();
        switch (opCode)
        {
            case OpCode.KEEP_ALIVE: msg = new NetKeepAlive(stream);break;
            case OpCode.WELCOME: msg = new NetWelcome(stream); break;
            case OpCode.START_GAME: msg = new NetStartGame(stream); break;
            case OpCode.MAKE_MOVE : msg = new NetMakeMove(stream); break;
            case OpCode.REMATCH: msg = new NetRematch(stream); break;
            default: Debug.LogError("Message reseived has no Opcode "); break;
        }
        if(server != null){ msg.ReceivedOnServer(cnn); }
        else { msg.ReceivedOnClient(); }
    }
}
```

4.5.12 Class Server Code:

```
using System;
using Unity.Collections;
using Unity.Networking.Transport;
using UnityEngine;

public class Server : MonoBehaviour
{
```

```
// to make treads for request
#region Singleton implementation
public static Server Instance { set; get; }


private void Awake()
{
   Instance = this;
}
#endregion
//Rules
public NetworkDriver driver;
private NativeList<NetworkConnection> connections;
private bool isActive = false;
private const float keepAliveTickRate = 20.0f; // to keep send message every 20 second
private float lastKeepAlive;
public Action connectionDropped;
//Methods
public void Init(ushort port) // initalize port
{
   driver = NetworkDriver.Create();
   NetworkEndPoint endPoint = NetworkEndPoint.AnyIpv4;
   endPoint.Port = port;


   if (driver.Bind(endPoint) != 0)  // 0 is success !=0 not success
   {
      Debug.Log("Unable to Bind on port " + endPoint.Port); // test
      return;
   }
```

```
        else
        {
            driver.Listen();
            Debug.Log("Currently listening on port" + endPoint.Port); // test
        }
            connections = new NativeList<NetworkConnection>(2, Allocator.Persistent); // max
player 2
        isActive = true;
    }
    public void Shutdown() // close off the server
    {
        if (isActive)
        {
            driver.Dispose();
            connections.Dispose();
            isActive = false;
        }
    }
    public void OnDestroy()
    {
        Shutdown();
    }
    private void Update()
    {
        if (!isActive)
        {
            return;
        }
```

```
    KeepAlive(); // send msg every 20 sec to sure connection between server and client


    driver.ScheduleUpdate().Complete(); // to make job system queue of msgs
    CleanupConnections(); // any body is not connected to us but we still have reference
    AcceptNewConnections(); // is there somebody knocking on the door to enter our server
    UpdateMessagePump();  // is are they sending us message and if so  do we have to reply
}
private void KeepAlive()
{
    if(Time.time - lastKeepAlive > keepAliveTickRate) // every 20 sec
    {
        lastKeepAlive = Time.time;
        Broadcast(new NetKeepAlive());
    }
}
 private void CleanupConnections()
{
    for (int i = 0; i < connections.Length; i++)
    {
        if (!connections[i].IsCreated)
        {
            connections.RemoveAtSwapBack(i);
            --i; // for don't break loop
        }
    }
}
 private void AcceptNewConnections()
{
```

```
        //Accept new Connections
        NetworkConnection c;
        while ((c = driver.Accept()) != default(NetworkConnection))
        {
            connections.Add(c); // add connection to list
        }
    }
    private void UpdateMessagePump()
    {
        DataStreamReader stream;
        for (int i = 0; i < connections.Length; i++)
        {
            NetworkEvent.Type cmd;
                    while((cmd = driver.PopEventForConnection(connections[i], out stream)) !=
NetworkEvent.Type.Empty)
            {
                if (cmd==NetworkEvent.Type.Data)
                {
                    NetUtility.OnData(stream, connections[i], this);
                }
                else if(cmd == NetworkEvent.Type.Disconnect)
                {
                    Debug.Log("Client Disconnect from server ");
                    connections[i] = default(NetworkConnection);
                    connectionDropped?.Invoke();
                        Shutdown(); // this does not happen usually , its just because we are in a two
player  game
                }
```

```
            }
        }
    }
    // Server specific
    public void SendToClient(NetworkConnection connection , NetMessages msg)
    {
        DataStreamWriter writer;
        driver.BeginSend(connection, out writer);
        msg.Serialize(ref writer);
        driver.EndSend(writer);
    }
    public void Broadcast(NetMessages msg)
    {
        for (int i = 0; i < connections.Length; i++)
        {
            if (connections[i].IsCreated)
            {
              //  Debug.Log($"Sending {msg.Code} To : {connections[i].InternalId}");
                SendToClient(connections[i], msg);
            }
        }
    }
}
```

4.5.13 Class Client Code:

```
using System;
using Unity.Networking.Transport;
```

```
using UnityEngine;
public class Client : MonoBehaviour
{
    //Rules
    public NetworkDriver driver;
    private bool isActive = false;
    private NetworkConnection connection;

    public Action connectionDropped;

    #region Singleton implementation
    public static Client Instance { set; get; }

    private void Awake()
    {
        Instance = this;
    }

    #endregion
    //Methods
    public void Init(string ip, ushort port) // initalize port
    {
        driver = NetworkDriver.Create();
        NetworkEndPoint endPoint = NetworkEndPoint.Parse(ip,port);
        connection = driver.Connect(endPoint);
        Debug.Log("Attemping to connect to server on " + endPoint.Address);
        isActive = true;
        RegisterToEvent();
```

```
    }
    public void Shutdown() // close off the server
    {
        if (isActive)
        {
            UnRegisterToEvent();
            driver.Dispose();
            isActive = false;
            connection = default(NetworkConnection);
        }
    }
    public void OnDestroy()
    {
        Shutdown();
    }
    private void Update()
    {
        if (!isActive)
        {
            return;
        }
        driver.ScheduleUpdate().Complete(); // to make job system queue of msgs
        CheckAlive();
        UpdateMessagePump();  // is are they sending us message and if so  do we have to reply
    }
    private void CheckAlive()
    {
        if (!connection.IsCreated && isActive)
```

```
    {
        Debug.Log("SomeThing went worng, we lost connection to server");
        connectionDropped?.Invoke();
        Shutdown();
    }
}
private void UpdateMessagePump()
{
    DataStreamReader stream;
        NetworkEvent.Type cmd;
        while ((cmd = connection.PopEvent(driver,out stream)) != NetworkEvent.Type.Empty)
        {
            if (cmd == NetworkEvent.Type.Connect)
            {
             SendToServer(new NetWelcome());
            Debug.Log("we are connected!");
            }
            else if (cmd == NetworkEvent.Type.Data)
            {
               NetUtility.OnData(stream, default(NetworkConnection));
            }
            else if (cmd == NetworkEvent.Type.Disconnect)
            {
                Debug.Log("Client Disconnect from server ");
                connection = default(NetworkConnection);
                connectionDropped?.Invoke();
                Shutdown();
            }
```

```
            }
        }
        public  void SendToServer(NetMessages msg)
        {
            DataStreamWriter writer;
            driver.BeginSend(connection, out writer);
            msg.Serialize(ref writer);
            driver.EndSend(writer);
        }
        //Event parsing
        private void RegisterToEvent()
        {
             NetUtility.C_KEEP_ALIVE += OnKeepAlive;
        }
        private void UnRegisterToEvent()
        {
            NetUtility.C_KEEP_ALIVE -= OnKeepAlive;
        }
        private void OnKeepAlive(NetMessages nm)
        {
            //Sent it Back , to Keep Both side alive
            SendToServer(nm);
        }
    }
```

4.5.14 Class NetKeepAlive Code:

```
```

```
using Unity.Networking.Transport;


public class NetKeepAlive : NetMessages
{
  public NetKeepAlive() // constractor <-- Making the box
  {
    Code = OpCode.KEEP_ALIVE;
  }
  public NetKeepAlive(DataStreamReader reader) // constractor <-- Receiving the box
  {
    Code = OpCode.KEEP_ALIVE;
    DeSerialize(ref reader);
  }
  public override void Serialize(ref DataStreamWriter writer)
  {
    writer.WriteByte((byte)Code);
  }
  public override void DeSerialize(ref DataStreamReader reader)
  {
    // defualt reader in class NetUtility
  }
  public override void ReceivedOnClient()
  {
    NetUtility.C_KEEP_ALIVE?.Invoke(this);
  }
  public override void ReceivedOnServer(NetworkConnection cnn)
  {
    NetUtility.S_KEEP_ALIVE?.Invoke(this,cnn);
```

```
    }
}
```

4.5.15 Class NetMakeMove Code:

```
using Unity.Networking.Transport;
public class NetMakeMove : NetMessages
{
    public int originalX;
    public int originalY;
    public int destinationX;
    public int destinationY;
    public int teamId;
    public NetMakeMove() // constractor <-- Making the box
    {
        Code = OpCode.MAKE_MOVE;
    }
    public NetMakeMove(DataStreamReader reader) // constractor <-- Receiving the box
    {
        Code = OpCode.MAKE_MOVE;
        DeSerialize(ref reader);
    }
    public override void Serialize(ref DataStreamWriter writer)
    {
        writer.WriteByte((byte)Code);
        writer.WriteInt(originalX);
        writer.WriteInt(originalY);
```

```
        writer.WriteInt(destinationX);
        writer.WriteInt(destinationY);
        writer.WriteInt(teamId);
    }
    public override void DeSerialize(ref DataStreamReader reader)
    {
        originalX = reader.ReadInt();
        originalY = reader.ReadInt();
        destinationX = reader.ReadInt();
        destinationY = reader.ReadInt();
        teamId = reader.ReadInt();
    }
    public override void ReceivedOnClient()
    {
        NetUtility.C_MAKE_MOVE?.Invoke(this);
    }
    public override void ReceivedOnServer(NetworkConnection cnn)
    {
        NetUtility.S_MAKE_MOVE?.Invoke(this, cnn);
    }
}
```

4.5.16 Class NetMessages Code:

```
using Unity.Networking.Transport;
public class NetMessages
{
  public OpCode Code { set; get; }
```

```
    public virtual void Serialize(ref DataStreamWriter writer) // to write bits data
    {
        writer.WriteByte((byte)Code);
    }
    public virtual void DeSerialize(ref DataStreamReader reader) // to read bits data
    {


    }
    public virtual void ReceivedOnClient() // Only address server
    {
        NetUtility.C_KEEP_ALIVE?.Invoke(this);
    }
        public virtual void ReceivedOnServer(NetworkConnection cnn) // to Receive Msg and
Attach who send to us
    {
        NetUtility.S_KEEP_ALIVE?.Invoke(this,cnn);
    }
}
```

4.5.17 Class NetRematch Code:

```
using Unity.Networking.Transport;
public class NetRematch : NetMessages
{
    public int teamId;
```

```
public byte WantRematch; // to check another player want Rematch or exit
public NetRematch() // constractor <-- Making the box
{
    Code = OpCode.REMATCH;
}
public NetRematch(DataStreamReader reader) // constractor <-- Receiving the box
{
    Code = OpCode.REMATCH;
    DeSerialize(ref reader);
}
public override void Serialize(ref DataStreamWriter writer)
{
    writer.WriteByte((byte)Code);
    writer.WriteInt(teamId);
    writer.WriteByte(WantRematch);
}
public override void DeSerialize(ref DataStreamReader reader)
{
    teamId = reader.ReadInt();
    WantRematch = reader.ReadByte();
}
public override void ReceivedOnClient()
{
    NetUtility.C_REMATCH?.Invoke(this);
}
public override void ReceivedOnServer(NetworkConnection cnn)
{
    NetUtility.S_REMATCH?.Invoke(this, cnn);
```

```
    }
}
```

4.5.18 Class NetStartGame Code:

```
using Unity.Networking.Transport;
using UnityEngine;
public class NetStartGame : NetMessages
{
    public NetStartGame()
    {
        Code = OpCode.START_GAME;
    }
    public NetStartGame(DataStreamReader reader)
    {
        Code = OpCode.START_GAME;
        DeSerialize(ref reader);
    }
    public override void Serialize(ref DataStreamWriter writer)
    {
        writer.WriteByte((byte)Code);
    }
    public override void DeSerialize(ref DataStreamReader reader)
    {
    }
    public override void ReceivedOnClient()
    {
```

```
        NetUtility.C_START_GAME?.Invoke(this);
    }
    public override void ReceivedOnServer(NetworkConnection cnn)
    {
        NetUtility.S_START_GAME?.Invoke(this, cnn);
    }
}
```

4.5.19 Class NetWelcome Code:

```
using Unity.Networking.Transport;
using UnityEngine;

public class NetWelcome : NetMessages
{

    public int AssignedTeam { set; get; }
    public NetWelcome()
    {
        Code = OpCode.WELCOME;
    }
    public NetWelcome(DataStreamReader reader)
    {
        Code = OpCode.WELCOME;
        DeSerialize(ref reader);
    }
    public override void Serialize(ref DataStreamWriter writer)
    {
```

```
        writer.WriteByte((byte)Code);
        writer.WriteInt(AssignedTeam);
    }
    public override void DeSerialize(ref DataStreamReader reader)
    {
        // we already read the byte in the NetUtility :: Data
        AssignedTeam = reader.ReadInt();
    }
    public override void ReceivedOnClient()
    {
        NetUtility.C_WELCOME?.Invoke(this);
    }
    public override void ReceivedOnServer(NetworkConnection cnn)
    {
        NetUtility.S_WELCOME?.Invoke(this, cnn);
    }
}
```

CHAPTER 5: Conclusion and Future work

5.1 Conclusion:

Now we reached to the final section of our project report, so we can summarize our project in the following main points :

- We successfully develop a chess game by using OOP principals and G.U.I
- Allow players to play chess game simultaneously, in other words playing chess game interactively at the same time .
- Allow players to play chess game over a network through LAN

5.2 Future Work:

As for the available time and due to the Covid-19 pandemic. we decided to accomplish this project as possible as we can do. hopefully in the future we can develop an AI such as minimax with alpha beta pruning algorithm in which player of chess game can play chess with his device whether this device is PC or smart phone.

References

1. https://www.udemy.com/course/games_mathematics/Udemy

   Mathematics for Computer Games Development using Unity

2. https://www.udemy.com/course/math-for-games/

   Math For Video Games: The Fastest Way To Get Smarter At Math

3. https://www.udemy.com/course/intro-to-data-oriented-tech-stack-dots-ecs-in-unity/

   Intro To Data Oriented Tech Stack (DOTS) & ECS In Unity

4. https://www.youtube.com/playlist?list=PLNrYPbOVrc-vvtUIsjD8WKVTsS4_08HSI

   Unity Courses for IUG students

5. https://www.youtube.com/playlist?list=PL5jV059fe9MYeTIU9ZWFZK0Wke8Lw5wxy

   Chess Game Tutorial

6. https://www.researchgate.net/publication/319390201_APPLYING_ALPHA-APPLYING

   ALPHA-BETA ALGORITHM IN A CHESS ENGINE