

Bishop's University

Computer Games Design

Final Project

Instructed By: Prof. Russell Butler

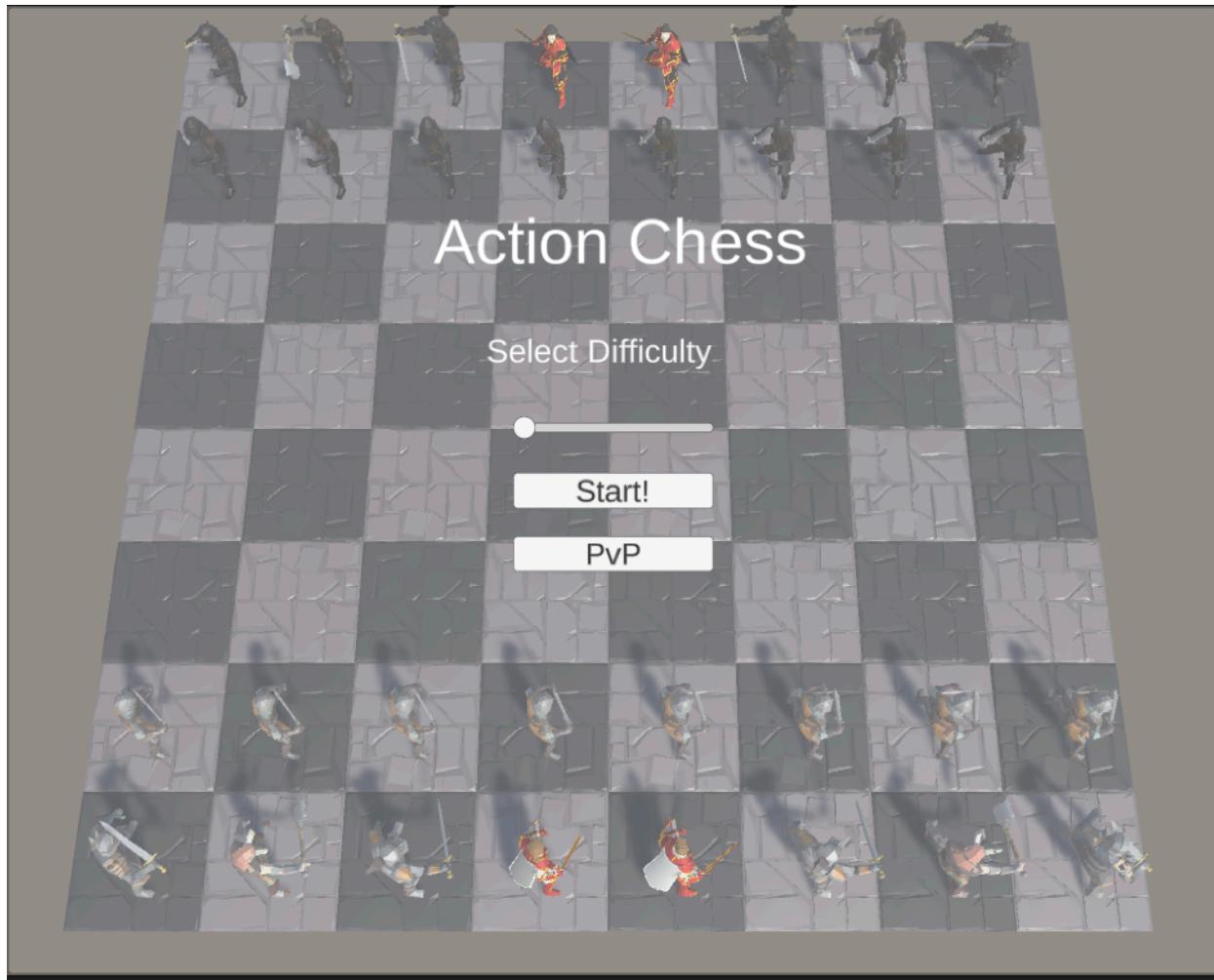
Submitted By:
Jagjot Singh

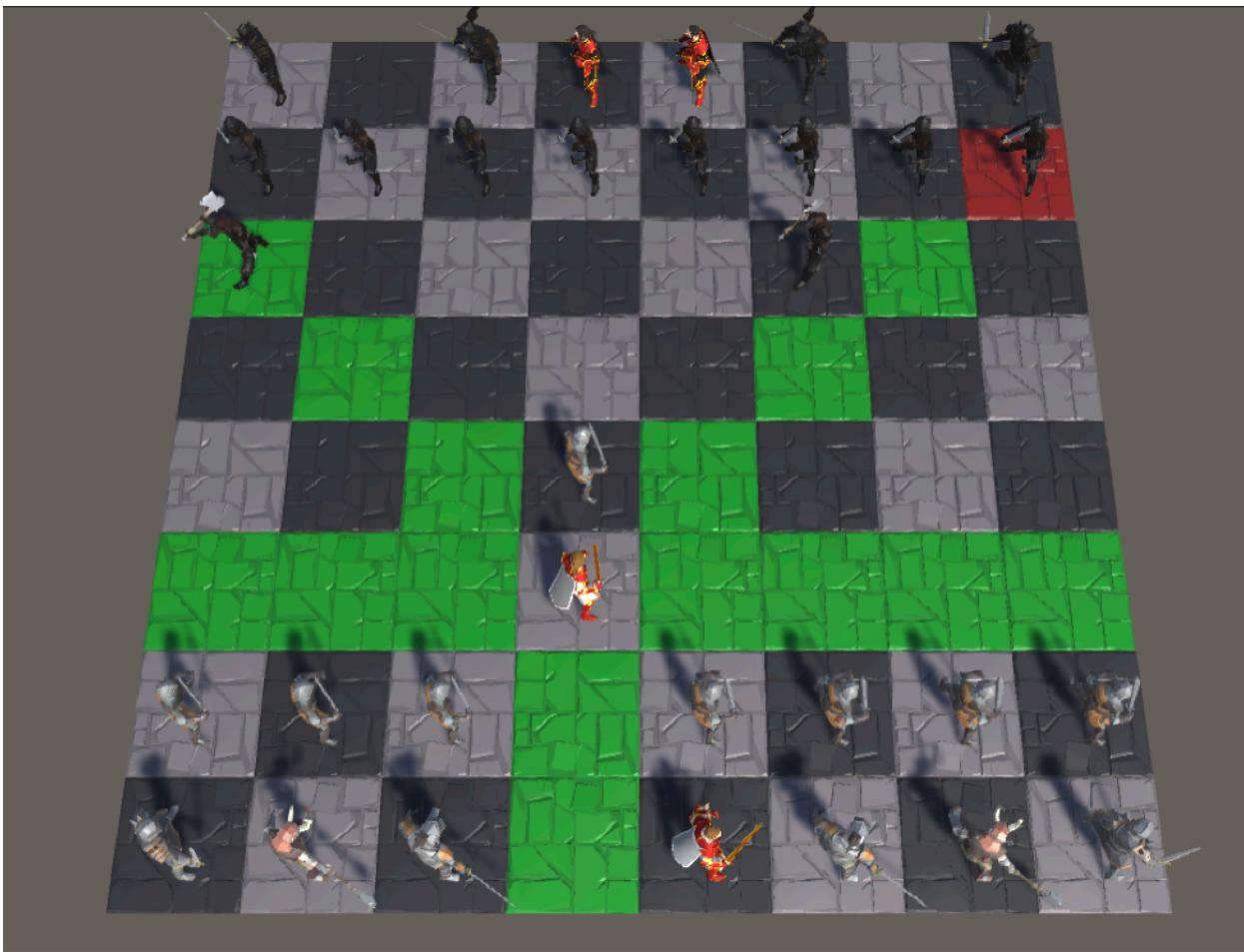
1) Game features implemented:

We tried to implement the following features in the our project:

- A) Chess rules.
- B) MiniMax With Alpha - Beta pruning.
- C) Piece selection and movement.
- D) Detailed attack collisions and piece destruction.
- E) Random player(At difficulty lvl 1) and MiniMax player.
- F) Difficulty slider.
- G) Sound effects for hit and death.
- H) Multiplayer with Unity NetCode and Relay.

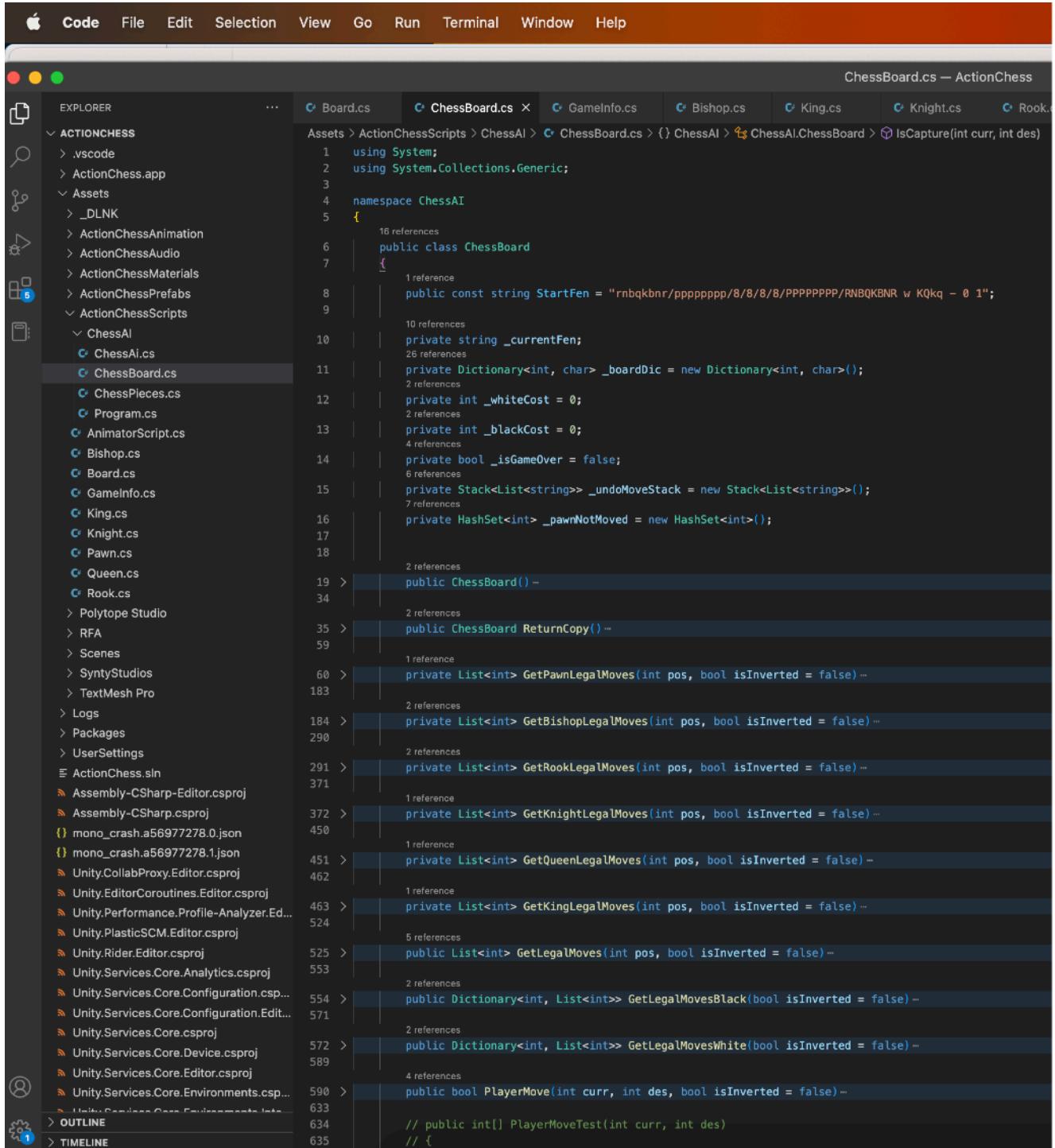
2) Game Screenshots:





Screenshot highlighting legal moves and pieces in danger

3) Chess base source code:



The screenshot shows a Code editor window with the following details:

- Menu Bar:** Code, File, Edit, Selection, View, Go, Run, Terminal, Window, Help.
- Explorer:** Shows the project structure under "ACTIONCHESS".
- Code Editor:** The file "ChessBoard.cs" is open, showing the following code:

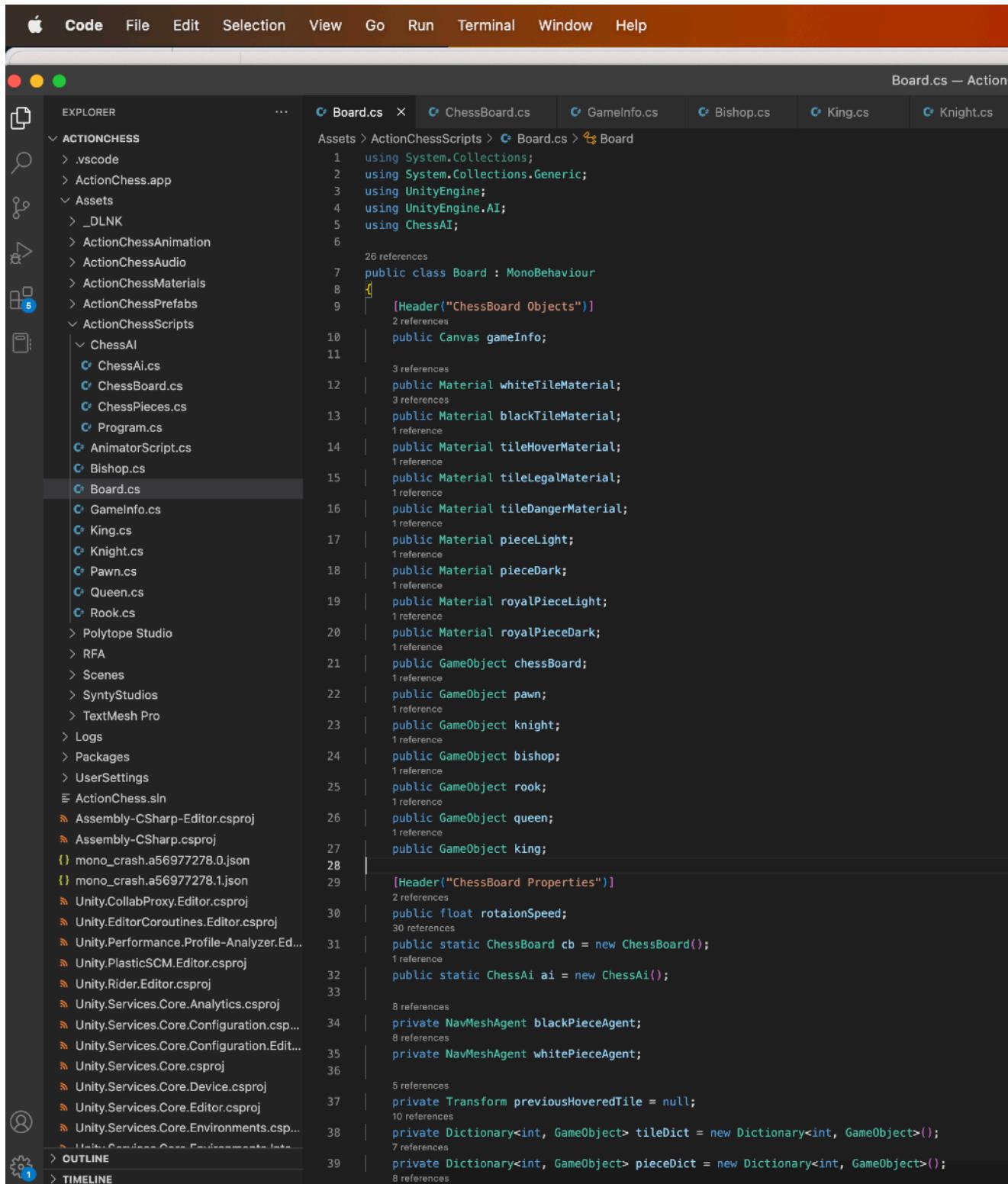
```
ChessBoard.cs — ActionChess

1  using System;
2  using System.Collections.Generic;
3
4  namespace ChessAI
5  {
6      public class ChessBoard
7      {
8          public const string StartFen = "rnbqkbnr/pppppppp/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1";
9
10         private string _currentFen;
11         private Dictionary<int, char> _boardDic = new Dictionary<int, char>();
12         private int _whiteCost = 0;
13         private int _blackCost = 0;
14         private bool _isGameOver = false;
15         private Stack<List<string>> _undoMoveStack = new Stack<List<string>>();
16         private HashSet<int> _pawnNotMoved = new HashSet<int>();
17
18         public ChessBoard()
19         {
20             _currentFen = StartFen;
21             _boardDic = CreateBoard();
22             _whiteCost = CalculateCost();
23             _blackCost = CalculateCost();
24         }
25
26         public ChessBoard ReturnCopy()
27         {
28             return new ChessBoard(_currentFen, _boardDic, _whiteCost, _blackCost);
29         }
30
31         private List<int> GetPawnLegalMoves(int pos, bool isInverted = false)
32         {
33             List<int> moves = new List<int>();
34             if (IsWhiteSquare(pos))
35             {
36                 if (IsWhiteSquare(pos + 8))
37                 {
38                     moves.Add(pos + 8);
39                 }
40                 if (IsWhiteSquare(pos + 7))
41                 {
42                     moves.Add(pos + 7);
43                 }
44                 if (IsWhiteSquare(pos + 9))
45                 {
46                     moves.Add(pos + 9);
47                 }
48                 if (IsWhiteSquare(pos + 6))
49                 {
50                     moves.Add(pos + 6);
51                 }
52                 if (IsWhiteSquare(pos + 5))
53                 {
54                     moves.Add(pos + 5);
55                 }
56                 if (IsWhiteSquare(pos + 4))
57                 {
58                     moves.Add(pos + 4);
59                 }
59             }
60             else
61             {
62                 if (IsBlackSquare(pos - 8))
63                 {
64                     moves.Add(pos - 8);
65                 }
66                 if (IsBlackSquare(pos - 7))
67                 {
68                     moves.Add(pos - 7);
69                 }
69                 if (IsBlackSquare(pos - 9))
70                 {
71                     moves.Add(pos - 9);
72                 }
72                 if (IsBlackSquare(pos - 6))
73                 {
74                     moves.Add(pos - 6);
75                 }
75                 if (IsBlackSquare(pos - 5))
76                 {
77                     moves.Add(pos - 5);
78                 }
78                 if (IsBlackSquare(pos - 4))
79                 {
80                     moves.Add(pos - 4);
81                 }
81             }
82             return moves;
83         }
84
85         private List<int> GetBishopLegalMoves(int pos, bool isInverted = false)
86         {
87             List<int> moves = new List<int>();
88             if (IsWhiteSquare(pos))
89             {
90                 if (IsWhiteSquare(pos + 9))
91                 {
92                     moves.Add(pos + 9);
93                 }
94                 if (IsWhiteSquare(pos + 7))
95                 {
96                     moves.Add(pos + 7);
97                 }
97                 if (IsWhiteSquare(pos - 9))
98                 {
99                     moves.Add(pos - 9);
100                }
100                if (IsWhiteSquare(pos - 7))
101                {
102                    moves.Add(pos - 7);
103                }
103            }
104            else
105            {
106                if (IsBlackSquare(pos - 9))
107                {
108                    moves.Add(pos - 9);
109                }
109                if (IsBlackSquare(pos - 7))
110                {
111                    moves.Add(pos - 7);
112                }
112                if (IsBlackSquare(pos + 9))
113                {
114                    moves.Add(pos + 9);
115                }
115                if (IsBlackSquare(pos + 7))
116                {
117                    moves.Add(pos + 7);
118                }
118            }
119            return moves;
120        }
121
122        private List<int> GetRookLegalMoves(int pos, bool isInverted = false)
123        {
124            List<int> moves = new List<int>();
125            if (IsWhiteSquare(pos))
126            {
127                if (IsWhiteSquare(pos + 8))
128                {
129                    moves.Add(pos + 8);
130                }
130                if (IsWhiteSquare(pos - 8))
131                {
132                    moves.Add(pos - 8);
133                }
133            }
134            else
135            {
136                if (IsBlackSquare(pos - 8))
137                {
138                    moves.Add(pos - 8);
139                }
139                if (IsBlackSquare(pos + 8))
140                {
141                    moves.Add(pos + 8);
142                }
142            }
143            return moves;
144        }
145
146        private List<int> GetKnightLegalMoves(int pos, bool isInverted = false)
147        {
148            List<int> moves = new List<int>();
149            if (IsWhiteSquare(pos))
150            {
151                if (IsWhiteSquare(pos + 1))
152                {
153                    moves.Add(pos + 1);
154                }
154                if (IsWhiteSquare(pos + 2))
155                {
156                    moves.Add(pos + 2);
157                }
157                if (IsWhiteSquare(pos - 1))
158                {
159                    moves.Add(pos - 1);
160                }
160                if (IsWhiteSquare(pos - 2))
161                {
162                    moves.Add(pos - 2);
163                }
163            }
164            else
165            {
166                if (IsBlackSquare(pos - 1))
167                {
168                    moves.Add(pos - 1);
169                }
169                if (IsBlackSquare(pos - 2))
170                {
171                    moves.Add(pos - 2);
172                }
172                if (IsBlackSquare(pos + 1))
173                {
174                    moves.Add(pos + 1);
175                }
175                if (IsBlackSquare(pos + 2))
176                {
177                    moves.Add(pos + 2);
178                }
178            }
179            return moves;
180        }
181
182        private List<int> GetQueenLegalMoves(int pos, bool isInverted = false)
183        {
184            List<int> moves = new List<int>();
185            moves.AddRange(GetRookLegalMoves(pos, isInverted));
186            moves.AddRange(GetKnightLegalMoves(pos, isInverted));
187            return moves;
188        }
189
190        private List<int> GetKingLegalMoves(int pos, bool isInverted = false)
191        {
192            List<int> moves = new List<int>();
193            if (IsWhiteSquare(pos))
194            {
195                if (IsWhiteSquare(pos + 1))
196                {
197                    moves.Add(pos + 1);
198                }
198                if (IsWhiteSquare(pos + 2))
199                {
200                    moves.Add(pos + 2);
201                }
201                if (IsWhiteSquare(pos - 1))
202                {
203                    moves.Add(pos - 1);
204                }
204                if (IsWhiteSquare(pos - 2))
205                {
206                    moves.Add(pos - 2);
207                }
207            }
208            else
209            {
210                if (IsBlackSquare(pos - 1))
211                {
212                    moves.Add(pos - 1);
213                }
213                if (IsBlackSquare(pos - 2))
214                {
215                    moves.Add(pos - 2);
216                }
216                if (IsBlackSquare(pos + 1))
217                {
218                    moves.Add(pos + 1);
219                }
219                if (IsBlackSquare(pos + 2))
220                {
221                    moves.Add(pos + 2);
222                }
222            }
223            return moves;
224        }
225
226        public List<int> GetLegalMoves(int pos, bool isInverted = false)
227        {
228            List<int> moves = new List<int>();
229            moves.AddRange(GetKingLegalMoves(pos, isInverted));
230            moves.AddRange(GetQueenLegalMoves(pos, isInverted));
231            moves.AddRange(GetRookLegalMoves(pos, isInverted));
232            moves.AddRange(GetBishopLegalMoves(pos, isInverted));
233            moves.AddRange(GetKnightLegalMoves(pos, isInverted));
234            moves.AddRange(GetPawnLegalMoves(pos, isInverted));
235            return moves;
236        }
237
238        public Dictionary<int, List<int>> GetLegalMovesBlack(bool isInverted = false)
239        {
240            Dictionary<int, List<int>> moves = new Dictionary<int, List<int>>();
241            foreach (var move in GetLegalMoves(0, isInverted))
242            {
243                moves.Add(move, new List<int>());
244            }
245            return moves;
246        }
247
248        public Dictionary<int, List<int>> GetLegalMovesWhite(bool isInverted = false)
249        {
250            Dictionary<int, List<int>> moves = new Dictionary<int, List<int>>();
251            foreach (var move in GetLegalMoves(1, isInverted))
252            {
253                moves.Add(move, new List<int>());
254            }
255            return moves;
256        }
257
258        public bool PlayerMove(int curr, int des, bool isInverted = false)
259        {
260            if (curr == 0)
261            {
262                if (des == 1)
263                {
264                    _currentFen = StartFen;
265                    _boardDic = CreateBoard();
266                    _whiteCost = CalculateCost();
267                    _blackCost = CalculateCost();
268                }
269            }
270            else
271            {
272                if (des == 0)
273                {
274                    _currentFen = StartFen;
275                    _boardDic = CreateBoard();
276                    _whiteCost = CalculateCost();
277                    _blackCost = CalculateCost();
278                }
279            }
280            return true;
281        }
282
283        // public int[] PlayerMoveTest(int curr, int des)
284        //{
285        //     return null;
286        //}
287    }
288}
```

The screenshot shows the Apple Xcode IDE interface. The top menu bar includes Code, File, Edit, Selection, View, Go, Run, Terminal, Window, and Help. The left sidebar (EXPLORER) displays the project structure for 'ACTIONCHESS' with files like .vscode, ActionChess.app, Assets, ActionChessScripts, and various chess-related classes such as Board.cs, ChessBoard.cs, GameInfo.cs, Bishop.cs, King.cs, and Knight.cs. The main code editor window shows the 'ChessBoard.cs' file with approximately 1000 lines of C# code, focusing on methods like UpdateCost(), GetWhiteCost(), GetBlackCost(), GetCurrentFen(), and IsCapture(). The code editor has syntax highlighting and line numbers.

Above is the screenshot of base chess, As the code is nearly 1000 lines of code, I've decided to post the screenshot of only methods used.

4) Source code for chessboard Unity C# Script:



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** Code, File, Edit, Selection, View, Go, Run, Terminal, Window, Help.
- Title Bar:** Board.cs — Action
- Explorer:** Shows the project structure under 'ACTIONCHESS'. The 'Board.cs' file is selected in the Explorer sidebar.
- Code Editor:** Displays the C# source code for 'Board.cs'. The code defines a 'Board' class that inherits from 'MonoBehaviour'. It includes fields for 'gameInfo' (Canvas), various materials (whiteTileMaterial, blackTileMaterial, tileHoverMaterial, tileLegalMaterial, tileDangerMaterial, pieceLight, pieceDark, royalPieceLight, royalPieceDark), game objects (chessBoard, pawn, knight, bishop, rook, queen, king), and properties for rotation speed and piece agents. The code uses Unity's [Header] attribute to group related properties.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.AI;
5  using ChessAI;
6
7  public class Board : MonoBehaviour
8  {
9      [Header("ChessBoard Objects")]
10     public Canvas gameInfo;
11
12     public Material whiteTileMaterial;
13     public Material blackTileMaterial;
14     public Material tileHoverMaterial;
15     public Material tileLegalMaterial;
16     public Material tileDangerMaterial;
17     public Material pieceLight;
18     public Material pieceDark;
19     public Material royalPieceLight;
20     public Material royalPieceDark;
21     public GameObject chessBoard;
22     public GameObject pawn;
23     public GameObject knight;
24     public GameObject bishop;
25     public GameObject rook;
26     public GameObject queen;
27     public GameObject king;
28
29     [Header("ChessBoard Properties")]
30     public float rotationSpeed;
31     public static ChessBoard cb = new ChessBoard();
32     public static ChessAI ai = new ChessAI();
33
34     private NavMeshAgent blackPieceAgent;
35     private NavMeshAgent whitePieceAgent;
36
37     private Transform previousHoveredTile = null;
38     private Dictionary<int, GameObject> tileDict = new Dictionary<int, GameObject>();
39     private Dictionary<int, GameObject> pieceDict = new Dictionary<int, GameObject>();
```

The screenshot shows the Unity Editor interface with the following details:

- Top Bar:** Apple logo, Code, File, Edit, Selection, View, Go, Run, Terminal, Window, Help.
- Left Sidebar (Explorer):** Shows the project structure under "ACTIONCHESS". Key items include ".vscode", "ActionChess.app", "Assets" (containing "_DLNK", "ActionChessAnimation", "ActionChessAudio", "ActionChessMaterials", "ActionChessPrefabs"), "ActionChessScripts" (containing "ChessAI" with files "ChessAi.cs", "ChessBoard.cs", "ChessPieces.cs", "Program.cs", "AnimatorScript.cs", "Bishop.cs", "Board.cs" (selected), "GameInfo.cs", "King.cs", "Knight.cs"), and various Unity packages like "Polytope Studio", "RFA", "Scenes", "SyntyStudios", "TextMesh Pro", "Logs", "Packages", "UserSettings", and "Assembly-CSharp-Editor.csproj", "Assembly-CSharp.csproj", "mono_crash.a56977278.0.json", "mono_crash.a56977278.1.json", "Unity.CollabProxy.Editor.csproj", "Unity.EditorCoroutines.Editor.csproj", "Unity.Performance.Profile-Analyzer.Ed...", "Unity.PlasticSCM.Editor.csproj", "Unity.Rider.Editor.csproj", "Unity.Services.Core.Analytics.csproj", "Unity.Services.Core.Configuration.cs...", "Unity.Services.Core.Configuration.Edit...", "Unity.Services.Core.csproj", "Unity.Services.Core.Device.csproj", "Unity.Services.Core.Editor.csproj", "Unity.Services.Core.Environments.cs...".
- Code Editor:** The "Board.cs" script is open. The code defines a class "Board" with fields for legal tiles, source and destination tiles, animators, and game objects for King and Queen. It includes methods for Start, Update, and FixedUpdate. The "Start" method initializes tile and piece dictionaries. The "Update" method checks for wins or draws and updates the game info component. The "FixedUpdate" method is empty.

```
private HashSet<int> legalTiles = new HashSet<int>();  
private int sourceTile = -1;  
private int destinationTile = -1;  
private Animator whiteAnimator;  
private Animator blackAnimator;  
public static bool isBlackTurn = false;  
private bool whiteTake = false;  
private bool blackTake = false;  
private GameObject blackKing;  
private GameObject whiteKing;  
private int cpuDifficulty = 1;  
  
void Start()  
{  
    // foreach (var item in cb.GetCurrentBoardDict())  
    // {  
    // }  
    Time.timeScale = 0f;  
    // cb.SetCurrentFen("rnb1kbnr/pp1p2pp/3N4/2pP1p2/2PpQ3/8/PP2PPP/R1q1KBNR b - - 99 50")  
    foreach (var item in chessBoard.GetComponentsInChildren<Transform>())  
    {  
        if (item.name.Contains("Tile"))  
        {  
            tileDict.Add(GetTileNumberFromTileTransform(item), item.gameObject);  
            pieceDict.Add(GetTileNumberFromTileTransform(item), RenderChessPiece(item));  
        }  
    }  
  
    // Update is called once per frame  
    void Update()  
{  
        // SetTileColor();  
        // MoveChessPiecePlayer();  
        if(blackKing == null){  
            var gi = this.gameInfo.GetComponent<GameInfo>();  
            gi.updateMsg("White Wins!");  
            gi.gameOver();  
        }  
        if(whiteKing == null){  
            var gi = this.gameInfo.GetComponent<GameInfo>();  
            gi.updateMsg("Black Wins!");  
            gi.gameOver();  
        }  
    }  
  
    void FixedUpdate()  
    {}
```

The screenshot shows a code editor interface with the following details:

- Header:** Apple, Code, File, Edit, Selection, View, Go, Run, Terminal, Window, Help.
- Title Bar:** Board.cs — Action
- Left Sidebar (EXPLORER):**
 - ACTIONCHESS
 - .vscode
 - ActionChess.app
 - Assets
 - _DLNK
 - ActionChessAnimation
 - ActionChessAudio
 - ActionChessMaterials
 - ActionChessPrefabs
 - ActionChessScripts
 - ChessAI
 - ChessAI.cs
 - ChessBoard.cs
 - ChessPieces.cs
 - Program.cs
 - AnimatorScript.cs
 - Bishop.cs
 - Board.cs
 - GamelInfo.cs
 - King.cs
 - Knight.cs
 - Pawn.cs
 - Queen.cs
 - Rook.cs
 - Polytope Studio
 - RFA
 - Scenes
 - SyntyStudios
 - TextMesh Pro
 - Logs
 - Packages
 - UserSettings
 - ActionChess.sln
 - Assembly-CSharp-Editor.csproj
 - Assembly-CSharp.csproj
 - { mono_crash.a56977278.0.json }
 - { mono_crash.a56977278.1.json }
 - Unity.CollabProxy.Editor.csproj
 - Unity.EditorCoroutines.Editor.csproj
 - Unity.Performance.Profile-Analyzer.Ed...
 - Unity.PlasticSCM.Editor.csproj
 - Unity.Rider.Editor.csproj
 - Unity.Services.Core.Analytics.csproj
 - Unity.Services.Core.Configuration.cs...
 - Unity.Services.Core.Configuration.Edit...
 - Unity.Services.Core.csproj
 - Unity.Services.Core.Device.csproj
 - Unity.Services.Core.Editor.csproj
 - Unity.Services.Core.Environments.cs...
 - Unity.Services.Core.EnvironmentalData...
- Right Panel (Code View):** The code for the `Board.cs` file is displayed. The current line of code is highlighted: `private GameObject RenderChessPiece(Transform tile) {`. The code includes methods for setting tile colors, moving pieces, attacking, and updating piece dictionaries.

5) MiniMax with Alpha- Beta pruning source code:

```

EXPLORER ... Board.cs ChessBoard.cs GameInfo.cs Bishop.cs King.cs Knight.cs Rook.cs
ACTIONCHESS ... Assets > ActionChessScripts > ChessAI > ChessAi.cs
.vscode
ActionChess.app
Assets
_DLNK
ActionChessAnimation
ActionChessAudio
ActionChessMaterials
ActionChessPrefabs
ActionChessScripts
ChessAI
ChessAi.cs
ChessBoard.cs
ChessPieces.cs
Program.cs
AnimatorScript.cs
Bishop.cs
Board.cs
GameInfo.cs
King.cs
Knight.cs
Pawn.cs
Queen.cs
Rook.cs
Polytope Studio
RFA
Scenes
SyntyStudios
TextMesh Pro
Logs
Packages
UserSettings
ActionChess.sln
Assembly-CSharp-Editor.csproj
Assembly-CSharp.csproj
mono_crash.a56977278.0.json
mono_crash.a56977278.1.json
Unity.CollabProxy.Editor.csproj
Unity.EditorCoroutines.Editor.csproj
Unity.Performance.Profile-Analyzer.Ed...
Unity.PlasticSCM.Editor.csproj
Unity.Rider.Editor.csproj
Unity.Services.Core.Analytics.csproj
Unity.Services.Core.Configuration.csp...
Unity.Services.Core.Configuration.Edit...
Unity.Services.Core.csproj
Unity.Services.Core.Device.csproj
Unity.Services.Core.Editor.csproj
Unity.Services.Core.Environments.csp...
Unity.Services.Core.EnvironmentalInt...
OUTLINE
TIMELINE
Assets > ActionChessScripts > ChessAI > ChessAi.cs
public class ChessAi
{
    private char _maximizingColor;
    // private ChessBoard _cb;
    //
    // public ChessAi(ChessBoard cb, char maximizingColor = 'w')
    // {
    //     this._cb = cb;
    //     this._maximizingColor = maximizingColor;
    // }

    public ChessAi(char maximizingColor = 'w')
    {
        this._maximizingColor = maximizingColor;
    }

    private int EvaluateHeuristic(ChessBoard cb)
    {
        if (_maximizingColor == 'w')
        {
            return cb.GetWhiteCost() - cb.GetBlackCost();
        }
        return cb.GetBlackCost() - cb.GetWhiteCost();
    }

    public int[] GetBlackNextMove(ChessBoard cb, int depth)
    {
        int[] bm = new int[3];
        return Minimax(cb, depth, int.MinValue, int.MaxValue, false, 0);
    }

    public int[] GetWhiteNextMove(ChessBoard cb, int depth)
    {
        return Minimax(cb, depth, int.MinValue, int.MaxValue, true, 0);
    }

    public int[] Minimax(ChessBoard cb, int depth, int alpha, int beta, bool isMaximizing, int s)
    {
        if (depth == 0 || cb.GetIsGameOver())
        {
            int[] bm = new int[3];
            bm[2] = EvaluateHeuristic(cb);
            return bm;
        }

        var rand = new Random();

        if (isMaximizing)
        {
            var maxEval = int.MinValue;
            var pieces = _maximizingColor == 'w' ? cb.GetLegalMovesWhite() : cb.GetLegalMovesBlack();
            int[] bm = new int[3];
            HashSet<int> set = new HashSet<int>();
            List<int> lst = new List<int>();
            foreach (var val in pieces)
            {
                lst.Add(val.Key);
            }
        }
    }
}

```

```
EXPLORER ... Board.cs ChessBoard.cs GameInfo.cs Bishop.cs King.cs Knight.cs Rook.cs  
ACTIONCHESS .vscode ActionChess.app Assets _DLNK ActionChessAnimation ActionChessAudio ActionChessMaterials ActionChessPrefabs ActionChessScripts ChessAI ChessAi.cs ChessBoard.cs ChessPieces.cs Program.cs AnimatorScript.cs Bishop.cs Board.cs GameInfo.cs King.cs Knight.cs Pawn.cs Queen.cs Rook.cs Polypipe Studio RFA Scenes SyntyStudios TextMesh Pro Logs Packages UserSettings ActionChess.sln Assembly-CSharp-Editor.csproj Assembly-CSharp.csproj mono_crash.a56977278.0.json mono_crash.a56977278.1.json Unity.CollabProxy.Editor.csproj Unity.EditorCoroutines.Editor.csproj Unity.Performance.Profile-Analyzer.Ed... Unity.PlasticSCM.Editor.csproj Unity.Rider.Editor.csproj Unity.Services.Core.Analytics.csproj Unity.Services.Core.Configuration.cs... Unity.Services.Core.Configuration.Edit... Unity.Services.Core.csproj Unity.Services.Core.Device.csproj Unity.Services.Core.Editor.csproj Unity.Services.Core.Environments.cs...  
Assets > ActionChessScripts > ChessAI > ChessAi.cs > () ChessAI > & ChessAI.ChessAi > GetBlackNextMove(ChessBoard cb, int  
52  
53     var maxEval = int.MinValue;  
54     var pieces = _maximizingColor == 'w' ? cb.GetLegalMovesWhite() : cb.GetLegalMovesBlack();  
55     int[] bm = new int[3];  
56     HashSet<int> set = new HashSet<int>();  
57     List<int> lst = new List<int>();  
58     foreach (var val in pieces)  
59     {  
60         lst.Add(val.Key);  
61     }  
62  
63     foreach (var t in pieces)  
64     {  
65         int index = rand.Next(lst.Count);  
66         while (set.Contains(index))  
67         {  
68             index = rand.Next(lst.Count);  
69         }  
70         set.Add(index);  
71         var curr = lst[index];  
72         foreach (var des in pieces[curr])  
73         {  
74             if (des != null)  
75             {  
76                 bm[des]++;  
77             }  
78         }  
79     }  
80     if (bm[0] > bm[1])  
81     {  
82         return bm[0];  
83     }  
84     else if (bm[1] > bm[0])  
85     {  
86         return bm[1];  
87     }  
88     else  
89     {  
90         var minEval = int.MaxValue;  
91         var pieces = _maximizingColor == 'w' ? cb.GetLegalMovesBlack() : cb.GetLegalMovesWhite();  
92         int[] bm = new int[3];  
93         HashSet<int> set = new HashSet<int>();  
94         List<int> lst = new List<int>();  
95         foreach (var val in pieces)  
96         {  
97             lst.Add(val.Key);  
98         }  
99  
100        foreach (var t in pieces)  
101        {  
102            int index = rand.Next(lst.Count);  
103            while (set.Contains(index))  
104            {  
105                index = rand.Next(lst.Count);  
106            }  
107            set.Add(index);  
108            var curr = lst[index];  
109            foreach (var des in pieces[curr])  
110            {  
111                if (des != null)  
112                {  
113                    bm[des]++;  
114                }  
115            }  
116        }  
117        if (bm[0] > bm[1])  
118        {  
119            return bm[0];  
120        }  
121        else if (bm[1] > bm[0])  
122        {  
123            return bm[1];  
124        }  
125    }  
126    return bm;
```

6) Player vs Player:

We have implemented the multiplayer functionality using Unity Netcode and Unity Relay service.



Players can request for the Unity Relay code and pass it to other player to join the multiplayer.

7) Source Code for Unity Relay Service:

```
private string joincode;
// Start is called before the first frame update
0 references
private async void Start()
{
    await UnityServices.InitializeAsync();
    AuthenticationService.Instance.SignedIn += () =>
    {
        Debug.Log("Signed In " + AuthenticationService.Instance.PlayerId);
    };

    await AuthenticationService.Instance.SignInAnonymouslyAsync();
}

1 reference
private async void CreateRelay()
{
    try
    {
        Allocation allocation = await RelayService.Instance.CreateAllocationAsync(1);
        joincode = await RelayService.Instance.GetJoinCodeAsync(allocation.AllocationId);
        // Debug.Log("JoinCode: " + joincode);

        NetworkManager.Singleton.GetComponent<UnityTransport>().SetHostRelayData(
            allocation.RelayServer.Ipv4,
            (ushort)allocation.RelayServer.Port,
            allocation.AllocationIdBytes,
            allocation.Key,
            allocation.ConnectionData
        );

        NetworkManager.Singleton.StartHost();
    }
    catch (RelayServiceException e)
    {
        Debug.Log(e);
    }
}

1 reference
private async void JoinRelay(string code)
{
    try
    {
        Debug.Log("Joining Relay with code: " + code);
        JoinAllocation joinAllocation = await RelayService.Instance.JoinAllocationAsync(code);

        NetworkManager.Singleton.GetComponent<UnityTransport>().SetClientRelayData(
            joinAllocation.RelayServer.Ipv4,
            (ushort)joinAllocation.RelayServer.Port,
            joinAllocation.AllocationIdBytes,
            joinAllocation.Key,
            joinAllocation.ConnectionData,
            joinAllocation.HostConnectionData
        );
        NetworkManager.Singleton.StartClient();
    }
    catch (RelayServiceException e)
    {
        Debug.Log(e);
    }
}
```

8) Files submitted:

a) ActionChess.zip:

Windows version of the game inside 'AC_Win' folder.

b) AC_Mac.zip: MacOS version on the game.

c) Scripts.zip: Contains the C# scripts.