CS412/512 Assignment 3: Wizard Chess

Student Name: Yuming Su

Student ID: 002344631

Professor: Russell Butler

Overview:

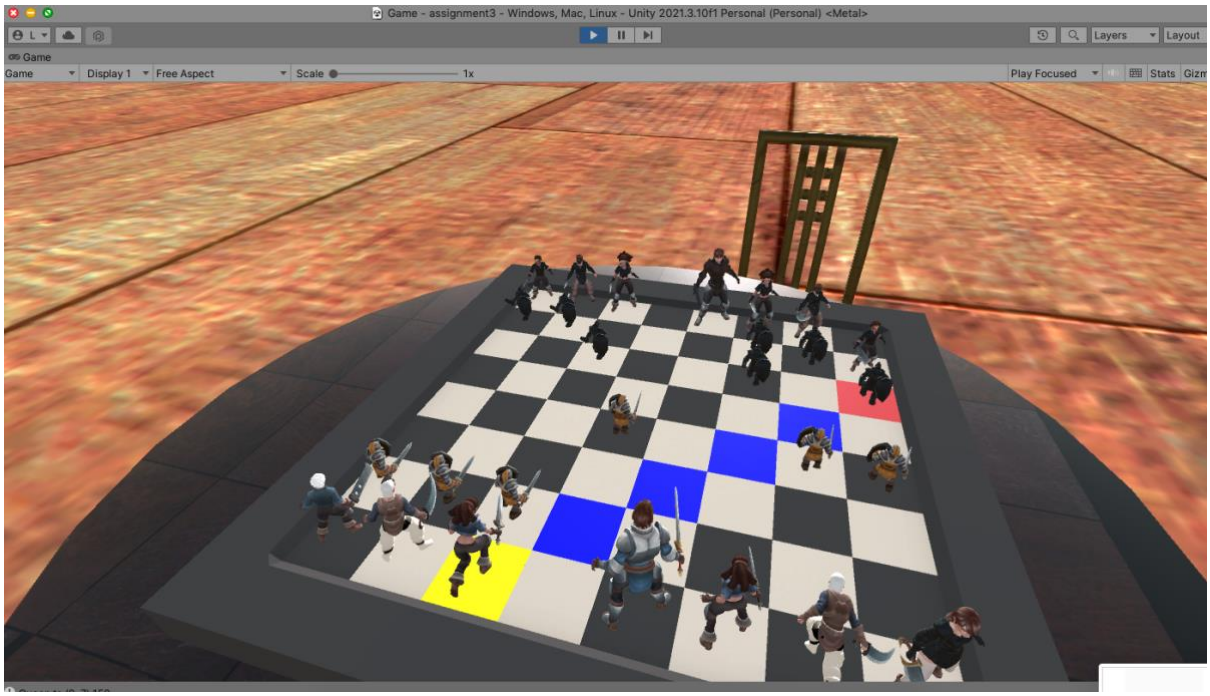
It is a chess game followed its rules. The control is easy only based on mouse. We control only white pieces, and the AI will be in black pieces. You can just click a chessman, and it become heighlighted. If it is selected, you will see the possible movement steps on chess board in different colors. The chessmen prefebs are replaced by some medieval cartoon warriors. Because the imported cartoon warriors cannot perfectly match our chess pieces, I set the different scales to identify them.

pieces Scales from large to small:

King=1, Queen=0.9, Bishop=0.8, Knight=0.7, Rook=0.6, Pawn=0.5

I locked the camera control with always focusing on chess board. There are avaliable UI to control. If you press “ESC”, it will be able to quit. If you are failed. It will show something. Please remember we are white, and the rivals are black. The background music was added!

When your chessman move on rival's chessman, you could kill the rival's chessman. The grid color will become red.



When the king is threated, you cannot move other pieces. You have to try to protect king first.

implement the minimax algorithm in C# with alpha-beta pruning:

In this project minimax algorithm is impletemented. I serached a lot of methods to implement on AI. At the heart of chess technology is the minimum-maximum local search of the game room. This technique tries to minimize your opponent's points and maximize your points. Check all possible movements at each depth. These points are distributed throughout the tree and you can choose the best movement at all depths. The larger the tree, the better decisions you can make (because the algorithm can see more progress).

alpha-beta pruning is used to significantly reduce the time of minimax search space. Basically, you can track the worst and best behaviors of each player and use these behaviors to completely avoid the branches that can give you worse results. This cut provides the same exact movement as using the mini-max (that is, no loss of accuracy). Ideally, you can double the depth of your search tree without increasing search time.

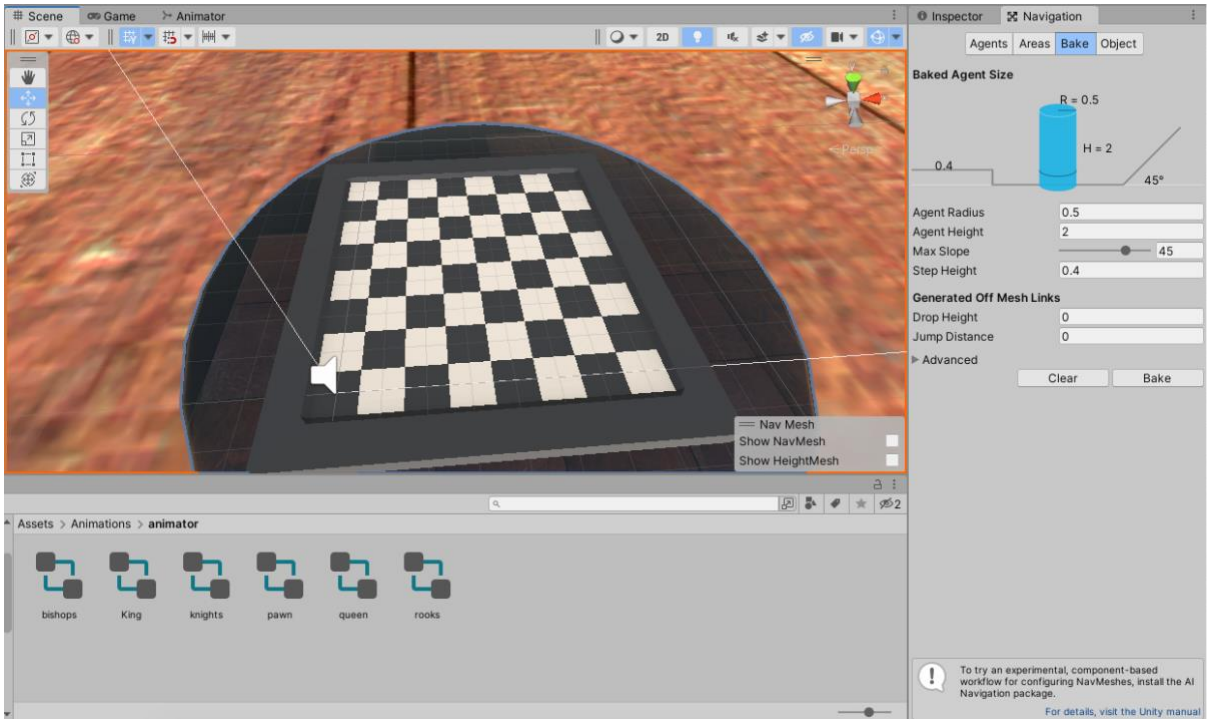
```
function minimax(position, depth, alpha, beta, maximizingPlayer)
  if depth == 0 or game over in position
    return static evaluation of position

  if maximizingPlayer
    maxEval = -infinity
    for each child of position
      eval = minimax(child, depth - 1, alpha, beta, false)
      maxEval = max(maxEval, eval)
    return maxEval

  else
    minEval = +infinity
    for each child of position
      eval = minimax(child, depth - 1, alpha, beta, true)
      minEval = min(minEval, eval)
    return minEval
```

```
ChessAI ▸ MiniMax(int depth, bool isMax)
144
145     private int MiniMax(int depth, bool isMax)
146     {
147         // If max depth is reached or Game is Over
148         if(depth == 0 || isGameOver())
149         {
150             // Static Evaluation Function
151             int value = StaticEvaluationFunction();
152
153             return value;
154         }
155
156         // string ActiveChessmansDetail = "";
157
158         // If it is max turn(NPC turn : Black)
159         if(isMax)
160         {
161             int hValue = System.Int32.MinValue;
162             // int ind = 0;
163             // Get list of all possible moves with their heuristic value
164             // For all chessmans
165             foreach(Chessman chessman in ActiveChessmans.ToArray())
166             {
167                 // ActiveChessmansDetail = ActiveChessmansDetail + "(" + ++ind + ")" + (chessman.isWhite?"White
168
169                 if(chessman.isWhite) continue;
170
171                 bool[,] allowedMoves = chessman.PossibleMoves();
172
173                 // detail = detail + "(" + ind + ") " + (chessman.isWhite?"White":"Black") + chessman.GetType()
174
175                 // For all possible moves
176                 for(int x=0; x<8; x++)
177                 {
178                     for(int y=0; y<8; y++)
179                     {
180                         if(allowedMoves[x, y])
181                         {
```

I do set navmesh agents and some animation controllers for all chess pieces including idle, walk, run, attack, die. It does work well very. Sometimes some parts may cause some bugs in MoveChessman method and AI script.



```
BoardManager ▸ MoveChessman(int x, int y)

152            // Allowed moves highlighted in blue and enemy in Red
153            allowedMoves = SelectedChessman.PossibleMoves();
154            BoardHighlights.Instance.HighlightPossibleMoves(allowedMoves, isWhiteTurn);
155
156
157             anim = SelectedChessman.GetComponent<Animator>();
158            agent = SelectedChessman.GetComponent<NavMeshAgent>();
159            /*!!agent.enabled = false;
160
161           agent.enabled = true;
162           anim = SelectedChessman.GetComponent<Animator>();
163           //Debug.Log(anim.name);
164
165           新的*/
166            /*!! 新的*/
167        }
168
169
170        public void MoveChessman(int x, int y)
171        {
172            if(allowedMoves[x,y])
173            {
174                Chessman opponent = Chessmans[x, y];
175
176                if(opponent != null)
177                {
178                    // Capture an opponent piece
179                    ActiveChessmans.Remove(opponent.gameObject);
180                    Destroy(opponent.gameObject);
181
182                }
183
184                if (EnPassant[0] == x && EnPassant[1] == y && SelectedChessman.GetType() == typeof(Pawn))
185                {
186                    if(isWhiteTurn)
187                        opponent = Chessmans[x, y + 1];
188                    else
```

```
ChessAI ▸ NPCMove()

46
47      // Funtion that makes NPC move
48      public void NPCMove()
49      {
50          // Start the StopWatch
51          System.Diagnostics.Stopwatch stopwatch  = new System.Diagnostics.Stopwatch();
52          stopwatch.Start();
53
54          // detail = "Start:\n";
55          // board = "Current Actual Board :\n";
56          // printBoard();
57
58          // New State History Stack
59          History = new Stack< State>();
60
61          /* ---------------------- Sense ---------------------- */
62
63          ActualChessmansReference = BoardManager.Instance.Chessmans;
64          ActualWhiteKing = BoardManager.Instance.WhiteKing;
65          ActualBlackKing = BoardManager.Instance.BlackKing;
66          ActualWhiteRook1 = BoardManager.Instance.WhiteRook1;
67          ActualWhiteRook2 = BoardManager.Instance.WhiteRook2;
68          ActualBlackRook1 = BoardManager.Instance.BlackRook1;
69          ActualBlackRook2 = BoardManager.Instance.BlackRook2;
70          ActualEnPassant = BoardManager.Instance.EnPassant;
71
72          ActiveChessmans = new List<Chessman>();
73          Chessmans = new Chessman[8, 8];
74
75          for(int x=0; x<8; x++)
76              for(int y=0; y<8; y++)
77              {
78                  if(ActualChessmansReference[x, y] != null)
79                  {
80                      Chessman currChessman = ActualChessmansReference[x, y].Clone();
81                      ActiveChessmans.Add(currChessman);
82                      Chessmans[x, y] = currChessman;
```

There are still some problems. In general, it is a very complicated project and challenging to me. I was tried to implement full animations in long time, but it doesn't performed very well. Since this cannot be done, I didn't start to set the difficulty levels of this game. I tried many

online resources too.

References:

https://assetstore.unity.com/packages/3d/characters/medieval-cartoon-

https://www.youtube.com/watch?v=cWgo0ak_8sE

https://assetstore.unity.com/packages/3d/props/chess-pieces-board-70092

https://github.com/juliangarnier/3D-Hartwig-chess-set

https://github.com/DarshanMaradiya/

https://www.youtube.com/watch?v=8W-MJlgAz8U

https://assetstore.unity.com/packages/3d/props/free-low-poly-chess-set-116856

https://assetstore.unity.com/packages/p/toon-rts-units-demo-69687