Chinese chess game design

Introduction:

Chinese chess is a popular form of strategic chess in that country. It has a rich historical and cultural history and is recognized as one of the oldest and most intricate chess pieces in the world. Chinese chess has its roots in China and dates back thousands of years to the Warring States and Spring and Autumn eras. Chinese chess is played on a nine-palace grid divided into the red and black kingdoms, respectively. Generals, officials, elephants, horses, chariots, artillery, and infantry are among the 16 pieces that each group has. Each piece moves and is constrained in a different way. The object of the game is to defeat your opponent by eating their will.Players of Chinese chess must evaluate situations and tactic. Based on each piece's worth and movement style, players must attack foes. A chess game's progression frequently depends on the mutual game between each side's attack and defense. A top chess player must have strong chess talents, superior computing capabilities, and a wealth of experience. Chinese chess is extremely popular and fiercely competitive. In China and many other nations, it has a large community of hobbyists and professional players. Chinese chess is an excellent activity for developing one's mind as well as their capacity for thought, judgment, and focus. It is frequently employed in the field of education and is regarded as a crucial strategy for developing students' intellect and character. Chinese chess is a sophisticated and age-old form of chess that combines mental aptitude, strategic thinking, and cultural components. It serves as both entertainment and a means of developing character and intelligence. It also captures the spirit of Chinese culture. Chinese chess is an intriguing and enjoyable game for both experienced players and casual enthusiasts. In the project, I designed a Chinese chess game that can meet the requirements of human-machine combat. This game offers different modes, including human-machine combat and player to player combat. And players can set difficulty levels during human-machine combat. In my testing, the playability of this game is very high, and it also ensures that players can enhance their chess skills in battles.







Method:

In this project, I first built a chessboard, which is the foundation of all the code, and then all the movements of the chess pieces are related to it. Afterwards, when constructing each piece, I determined their position through the corresponding coordinates of the chessboard and set the movement rules for each piece. Afterwards, it is also the key to all, different from human to human combat, human-machine combat requires judgment on the operating rules of the computer. Here we used decision tree judgment and set different priority levels for different pieces based on priority. I will explain all my script files below.

Checkmate: This is a determination whether the Grand Marshal has been cornered. Afterwards, the code also needs to determine whether the Grand Marshal will be eaten by any type of chess piece.

public Checkmate()

{

gameManager = GameManager.Instance;

uiManager = UIManager.Instance;

}

/// <summary>

/// whether threat general

/// </summary>

public void JudgeIfCheckmate()

{

GetKingPosition();

//If there is no will in the index position obtained by traversing from the above method, it will not exist and has been eaten

if (gameManager.chessBoard[jiangX,jiangY]!=1)

{

uiManager.ShowTip("red win");

gameManager.gameOver = true;

return;

}

//general does not exist, has been eaten

else if (gameManager.chessBoard[shuaiX,shuaiY]!=8)

{

uiManager.ShowTip("黑色棋子胜利");

gameManager.gameOver = true;

return;

}

ChessOrGrid：This file is responsible for managing many rules related to the movement of chess pieces. For example, by setting the color of the chess pieces, the order of the players can be judged. Secondly, trigger detection when clicking on a piece or grid to judge the rules of the piece's movement and ensure that the piece's movement is logical. And during this process, the code will also separate the player from the player and the player from the computer. When the player is in a war with the player, it provides thinking time. During a war between the player and the computer, the computer will directly play chess according to the set logic.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class ChessOrGrid : MonoBehaviour

{

public int xIndex, yIndex; // Coordinates of the cell on the grid

public bool isRed; // Indicates if the cell belongs to the red player

public bool isGrid; // Indicates if the cell is part of the grid or contains a chess piece

private GameManager gameManager; // Reference to the GameManager script

private GameObject gridGo; // Reference to the current game object

void Start()

{

gameManager = GameManager.Instance; // Assign the GameManager instance to the gameManager variable

gridGo = gameObject; // Assign the current game object to the gridGo variable

}

public void ClickCheck()

{

if (gameManager.gameOver) // If the game is already over, return

return;

int itemColorId = isGrid ? 0 : (isRed ? 2 : 1); // Determine the itemColorId based on the cell type

GridOrChessBehavior(itemColorId, xIndex, yIndex); // Call the GridOrChessBehavior method with the determined parameters

}

private void GridOrChessBehavior(int itemColorID, int x, int y)

{

int FromX, FromY, ToX, ToY; // Variables to store coordinates

gameManager.HideCanEatUI(); // Hide the UI indicating potential moves

switch (itemColorID)

{

case 0:

// Behavior for an empty cell

// Add your code here

break;

case 1:

// Behavior for black pieces

// Add your code here

break;

case 2:

// Behavior for red pieces

// Add your code here

break;

default:

// Invalid itemColorId

// Add error handling or fallback behavior here

break;

}

}

}

ChessReseting：This code is responsible for managing the repentance function. Firstly, use a counter to record the number of moves taken by the chess player. Afterwards, create an array to store all the steps that have already been taken for repentance. After obtaining the position of each step, simply clicking the repentance button will subtract 1 from the array index, causing the chess piece to move to the position of the previous step, completing the repentance task。

GameManager：This code, as the core code for managing games, implements numerous functions. Firstly, it sets up a combat mode (player to player or player to computer). Secondly, the code sets some classes to control the progress of the game, including which side is in turn to play chess, and whether the game ends. Afterwards, the code processed the resources in the game, such as grids, chess pieces, and the UI for the movement of chess pieces. For example, the chess pieces that have been eaten will disappear, and players can choose to replay the game (which will initialize the chessboard), etc.

using System. Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

/// <summary>

/// Store game data, game references, game resources, mode switching and control

/// </summary>

public class GameManager : MonoBehaviour

{

public static GameManager Instance { get; private set; }

public int chessPeople;//The current number of players, PVE 1 PVP 2 Networking 3

public int currentLevel;//Current difficulty 1. Easy 2. Normal 3. Difficult

/// <summary>

/// data

/// </summary>

public int[,] chessBoard;//The status of the current chessboard

public GameObject[,] boardGrid;//All grids on the board

private const float gridWidth = 69.9f;

private const float gridHeight = 71.8f;

private const int gridTotalNum = 90;

/// <summary>

/// switch

/// </summary>

public bool chessMove;//Which way to go, red true black false

public bool gameOver;//The end of the game can not move chess

private bool hasLoad;//Whether the current game has been loaded

/// <summary>

/// resource

/// </summary>

public GameObject gridGo;//grid

public Sprite[] sprites;//The sprites of all pieces

public GameObject chessGo;//chess piece

public GameObject canMovePosUIGo;//Position that can be moved to display

/// <summary>

/// Quote

/// </summary>

[HideInInspector]

public GameObject boardGo;//board

public GameObject[] boardGos;//0. Standalone 1. Networking

[HideInInspector]

public ChessOrGrid lastChessOrGrid;//The last clicked object (grid or chess piece)

public Rules rules;//rule class

public MovingOfChess movingOfChess;//moving class

public Checkmate checkmate;//General detection class

public ChessReseting chessReseting;//Repent chess

public SearchEngine searchEngine;//search engine

public GameObject eatChessPool;//The chess piece pool where the eaten chess pieces are stored

public GameObject clickChessUIGo;//UI display of selected chess pieces

public GameObject lastPosUIGo;// UI display of the position of the chess piece before moving

public GameObject canEatPosUIGo;//The UI display that can eat the chess piece

private Stack<GameObject> canMoveUIStack;//Move the position UI to display the object pool of the game object

private Stack<GameObject> currentCanMoveUIStack;//The game object stack that has been displayed by the UI when moving the position

MovingOfChess：This is a code that controls the movement rules and queries of chess pieces. Firstly, create an array to store a list of all legal paths. The first index represents the search depth, while the second index represents the corresponding number at the current depth. When the chess piece moves, its own position and the direction it wants to move will appear. So if encountering other pieces that need to be eaten, the code will turn the piece into a subclass of the eaten piece, inherit its position, and remove the eaten piece. Of course, before eating the chess pieces, we need to traverse the entire chessboard to ensure that its marching rules meet the requirements. After confirming that everything is correct, we can perform the above operations.

Rules:This is a code that controls the movement rules of chess pieces. If moving chess is a code that controls the movement of chess pieces and determines whether there is a function to eat chess pieces, then Rules are the rules that control the movement of chess pieces, such as cars can only walk in a straight line, generals cannot walk out of the nine squares, and so on.

public bool IsValidMove(int[,] position, int FromX, int FromY, int ToX, int ToY)

{

// Check if the move is valid based on the current position

// and the specified source and destination coordinates

// Return true if the move is valid, otherwise false

pass;

}

public bool IsSameSide(int x, int y)

{

// Check if the two chess pieces specified by their IDs

// belong to the same side or player

// Return true if they belong to the same side, otherwise false

pass;

}

public bool IsBlack(int x)

{

// Check if the chess piece specified by its ID is black

// Return true if it is black, otherwise false

pass;

}

public bool IsRed(int x)

{

// Check if the chess piece specified by its ID is red

// Return true if it is red, otherwise false

pass;

}

public bool IsVaild(int moveChessID, int[,] position, int FromX, int FromY, int ToX, int ToY)

{

// Check if the move of the chess piece specified by its ID

// is valid based on the current position and the specified

// source and destination coordinates

// Return true if the move is valid, otherwise false

pass;

}

public bool KingKill(int[,] position, int FromX, int FromY, int ToX, int ToY)

{

// Check if the move of the king chess piece is a killing move,

// where the king captures an opponent's chess piece

// Return true if it is a killing move, otherwise false

pass;

}

Future improvement:

In this project, I believe the most important part is the design of AI. Due to the high requirements for AI operation in human-machine combat, many simple algorithms are insufficient. So I first improved the original deep exploration algorithm. The specific operation is to first assign different values to different pieces, such as using chariots, artillery, soldiers, and elephants. And then give different pieces different degrees of flexibility. By calculating their agility and value, calculate their danger value and determine the movement or attack target of the chess pieces. Finally, several algorithms (negative pole maximum algorithm, alpha beta pruning algorithm, merge sort and history inspired optimization pruning algorithm) are used to calculate each chess piece movement algorithm, calculate the maximum number of steps to win, and record the steps, that is, the operation mode of each chess piece, find the optimal algorithm, generate the corresponding chess spectrum, train the most powerful AI, and complete the man-machine battle. However, so far, the algorithm still has shortcomings. Due to the lack of many training parameters, the operation of computer chess is still too rigid. It only eats according to the threat level of the chess pieces, without judging the traps set by the opponent, and does not have higher thinking ability. Afterwards, I consider combining algorithms with AI to train a reasonable model for the computer, and increasing the training scale. For example, using some chess games used by masters for replay training can make the computer more intelligent.