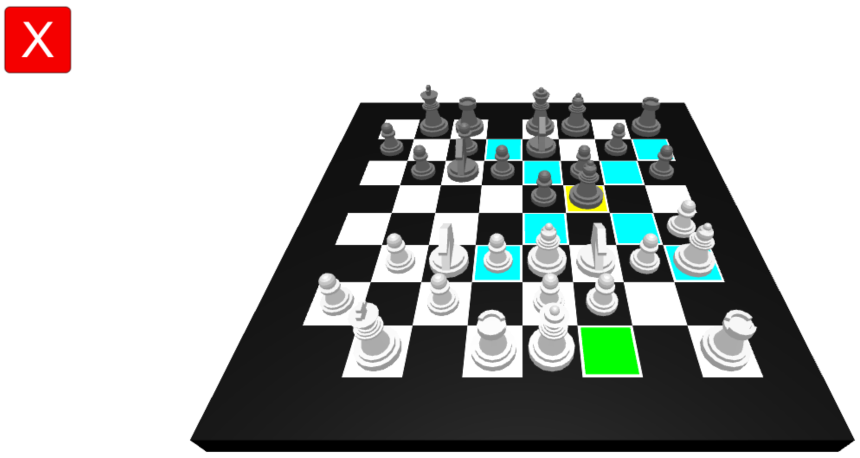
Chess Game

By WaveCave Games



Description

This is a very simple double chess game, which is played by 2 players. Control 16 pieces, eat enemy's king to win the game. Include full source code so you can easily make an individual mode (as known as player vs. computer).

Features:

- Easy to win
- Ready to use 3D chess pieces
- Include all basic piece movements and special movements (en passant, castling, pawn promotion (to queen / rook / knight / bishop)).
- Left / right king position setting
- Open source
- Clean script interface

How to Play

Open ‘WaveCaveGames’ -> ‘ChessGame’ -> ‘Scenes’, open the ‘Menu’ scene, click ‘Play’ button.

In the settings, you can change the king position and toggle show possible moves. 'Show Possible Moves' don't show the points that king will under attack after get there. This is a heavy function, it needs enough memory to run.

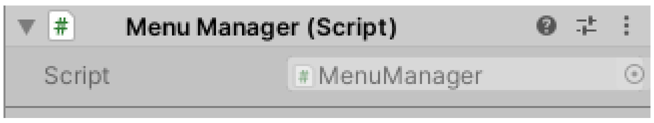
Control 16 pieces, eat enemy's king to win the game.

For further help, search ‘How to play chess’.

Component Guide

Menu Manager:

This script is used in button. It has no properties.

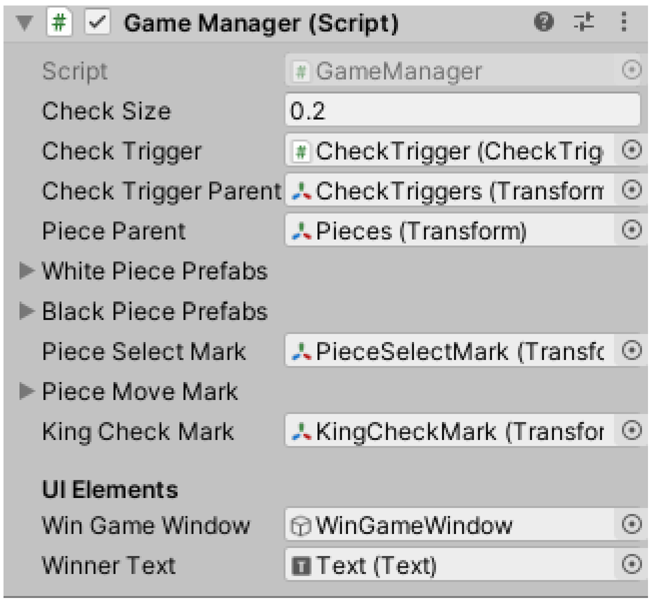


Menu Manager (Script)

Script	MenuManager

Game Manager:

Check Size:
The single check size. Chess board Surface size is check size * 8.
Check Trigger:
The check trigger. It automatically Create 64 triggers during Start() Function.
Check Trigger Parent:
The parent object for all triggers.
Piece Parent:
The parent object for all pieces.
Piece Prefabs:
The prefab for all pieces. It automatically create 32 pieces during Start() function.
Piece Select Mark:
The mark you used for select a piece.
Piece Move Mark:
The mark you used after moving a piece. Size must be 2.
King Check Mark:
The mark you used when the king is under attack.
Win Game Window:
The window displayed on UI. When you ‘eat’ enemy’s king, this window will appear.
Winner text:
The text inside the win game window, display whose win.

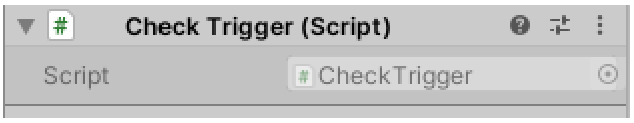


Game Manager (Script)

Script	GameManager
Check Size	0.2
Check Trigger	CheckTrigger (CheckTrig
Check Trigger Parent	CheckTriggers (Transform
Piece Parent	Pieces (Transform)
▶ White Piece Prefabs	
▶ Black Piece Prefabs	
Piece Select Mark	PieceSelectMark (Transfc
▶ Piece Move Mark	
King Check Mark	KingCheckMark (Transfor
UI Elements	
Win Game Window	WinGameWindow
Winner Text	Text (Text)

Check Trigger:

When you clicked a piece, the trigger And its marks will appear. It has no Properties in inspector.

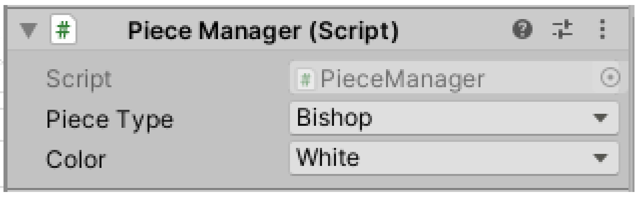


Check Trigger (Script)

Script	CheckTrigger

Piece Manager:

Piece Type:
The piece type (pawn / rook / Knight / bishop / queen / king).
Color:
The piece color (black / white).

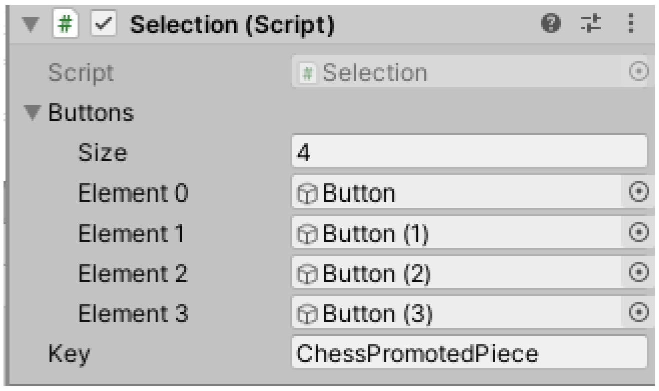

Piece Manager (Script)

Script: PieceManager
Piece Type: Bishop
Color: White

General Components

Selection:

Buttons:
All buttons inside this object.
Key:
The key string you used for storing the data.

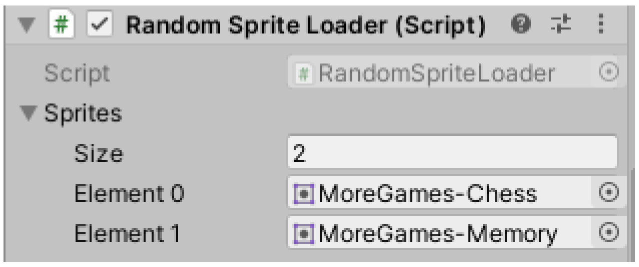


Selection (Script)

Script	Selection
Buttons	
Size	4
Element 0	Button
Element 1	Button (1)
Element 2	Button (2)
Element 3	Button (3)
Key	ChessPromotedPiece

Random Sprite Loader:

Sprites:
Sprites you want to load randomly.



Random Sprite Loader (Script)

Script	RandomSpriteLoader
Sprites	
Size	2
Element 0	MoreGames-Chess
Element 1	MoreGames-Memory

Scripting

```
namespace WaveCaveGames.ChessGame{

public class MenuManager : MonoBehaviour
{
public void LoadScene(string sceneName){}
public void Quit(){}
public void OpenURL(string url){}
}

public class GameManager : MonoBehaviour
{
[System.Serializable] public class PieceList{
public GameObject pawn;
public GameObject rook;
public GameObject knight;
public GameObject bishop;
public GameObject queen;
public GameObject king;
}
public float checkSize;
public CheckTrigger checkTrigger;
public Transform checkTriggerParent;
public Transform pieceParent;
public PieceList whitePiecePrefabs;
public PieceList blackPiecePrefabs;
public Transform pieceSelectMark;
public Transform[] pieceMoveMark;
public Transform kingCheckMark;
[Header("UI Elements")]
public GameObject winGameWindow;
public Text winnerText;
//check trigger is laid out regularly, laid 8 horizontal checks
out, then start laying next vertical checks. e.g. checkTriggers[0]
position is (-3.5, -3.5), checkTriggers[6] position is (-3.5, 2.5),
checkTriggers[25] position is (-0.5, -2.5)
[HideInInspector] public CheckTrigger[] checkTriggers;
[HideInInspector] public PieceManager clickedPiece;
//true = black turn, false = white turn
[HideInInspector] public bool isBlackTurn;
[HideInInspector] public CheckTrigger pawnPassedCheck;
[HideInInspector] public PieceManager passedPawn;
```

```
//0,7 = rook, 1,6 = knight, 2,5 = bishop, 3 = king, 4 = queen, 8-15
= pawn
[HideInInspector] public PieceManager[] whitePieces;
[HideInInspector] public PieceManager[] blackPieces;
public const float sqrt2 = 1.414214f;
public CheckTrigger FindCheck(Vector3 v){}
//intervalMultiplier: this value must be 1 or sqrt2 (1.414214),
if the selected piece is knight, you can ignore this value setting.
public void FindCheckAndLightUp(Vector3 v, float intervalMulti
plier){}
public void FindCheckAndLightUp(Vector3 v, bool castledPlace){}
//return check if can get to target, else return null. p: the
selected piece. v: the check vector. checkIfAttacked: if ticked,
it will give possible moves (excludes the check that king will under
attack after get there), don’t tick this to avoid crashes and
exceptions.
public CheckTrigger FindAndReturnCheckIfCanGetToTarget(PieceMa
nager p, Vector3 v, float intervalMultiplier, bool checkIfAtta
cked){}
public CheckTrigger FindAndReturnCheckIfCanGetToTarget(PieceMa
nager p, Vector3 v, bool castledPlace, bool checkIfAttacked){}
public void ClickPiece(PieceManager p){}
//return all possible targets for piece p. don’t tick
checkIfAttacked to avoid crashes and exceptions.
public CheckTrigger[] PossibleTargets(PieceManager p, bool che
ckIfAttacked){}
public void ChangePlayerTurn(){}
//checkIfAttacked: if ticked, it will give possible moves (excludes
the check that king will under attack after get there), don’t tick
this to avoid crashes and exceptions.
public bool IsKingUnderAttack(bool checkIfAttacked){}
//CheckIfKingUnderAttack must call after ChangePlayerTurn()
public void CheckIfKingUnderAttack(){}
//if show possible moves is ticked, it will end the game when a
player can’t protect his king, if not ticked, you need to ‘eat’
enemy’s king to win.
public void CheckIfWin(){}
public void BackToMenu(){}
public CheckTrigger[] FindActiveTriggers(){}
}

public class CheckTrigger : MonoBehaviour
{
[HideInInspector] public PieceManager piece;
```

```
[HideInInspector] public bool castledPlace;
public void ClickThis(){}
}

public enum PieceType{
Pawn, Rook, Knight, Bishop, Queen, King
}
public enum PieceColor{
White, Black
}
public class PieceManager : MonoBehaviour
{
public PieceType pieceType;
public PieceColor color;
[HideInInspector] public bool hasMoved;
public void ClickThis(){}
}
}

namespace WaveCaveGames.UI{

public class Selection : MonoBehaviour {
public GameObject[] buttons;
public string key;
public void SetKeyValue(int value){}
public void BackupValue(){}
public void SaveBackupValue(){}
}

public class RandomSpriteLoader : MonoBehaviour {
public Sprite[] sprites;
}
}

namespace WaveCaveGames.Utilities{

public class ArrayUtility {
//Return the size of the array.
public static int Size<T>(T[] array){}
//Add an object to target array.
public static void IncreaseArray<T>(ref T[] array, T item){}
//Add an object to target array, and place it to target 'itemI
ndex'. Index must be equal or smaller than the size of the array.
public static void IncreaseArray<T>(ref T[] array, T item, int
 itemIndex){}
//Add multiple objects to target array.
public static void IncreaseArray<T>(ref T[] array, params T[]
items){}
//Add a 'not null' object to target array.
public static void IncreaseArrayAvoidNull<T>(ref T[] array, T
item){}
//Add a 'not null' object to target array, and place it to tar
get 'itemIndex'.
public static void IncreaseArrayAvoidNull<T>(ref T[] array, T
item, int itemIndex){}
//Remove object 'index' in target array. If the index is 2, it
'll remove the second object. The index must be lower than the
 size of the array.
public static void DecreaseArray<T>(ref T[] array, int index){}
//Remove object from - to in target array. From must be lower
than to. If the from is equal to to, it'll remove this object
only.
public static void DecreaseArray<T>(ref T[] array, int from, i
nt to){}
//Remove 'item' in target array. If you have multiple 'item',
it'll remove the first one.
public static void DecreaseArrayObject<T>(ref T[] array, T ite
m){}
//Remove from - to in target array. Make sure not have same it
ems in the array, or it'll not work correctly.
public static void DecreaseArrayObject<T>(ref T[] array, T fro
m, T to){}
//Remove all 'item' in target array. If the item is null, it’s
recommend using RemoveEmptyItems.
public static void RemoveAll<T>(ref T[] array, T item){}
//Resize an array. This works same as the editor array resizin
g.
public static void Resize<T>(ref T[] array, int size){}
//Clear the array.
public static void ClearArray<T>(ref T[] array){}
//Remove all empty items in an array.
public static void RemoveEmptyItems<T>(ref T[] array){}
//Remove same items. Sometimes it'll not remove completely for
 some strange null reference. To remove completely, call Remov
eEmptyItems.
public static void RemoveSameItems<T>(ref T[] array){}
//Return the index of the object 'item'. If you have multiple
'item', it'll return the first one. If not found, it'll return
```

```
 -1.
public static int ObjectToIndex<T>(T[] array, T item){}
//Return all the array items' name string.
public static string ArrayItems<T>(T[] array){}
//Return the layouted array items' name string.
public static string ArrayItemsLayouted<T>(T[] array){}
}
}
```

Tips

- All scripts are free to use in commercial games, but you can’t sell it elsewhere, or publish this game without modification.
- You can easily use the provided method to create individual mode.
- If you don’t like my chess model, you can change another model. And don’t forget the check size configuration.
- If the check trigger is unresponsive, drag the camera closer to chess board,