UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare
SPECIALIZAREA: Calculatoare Încorporate

3D Multiplayer Checkers with Leap Motion

Joc de dame 3D multiplayer utilizând tehnologia Leap Motion

PROIECT
PROIECTAREA SISTEMELOR
DE
INTERACȚIUNE OM-CALCULATOR

Coordonator științific,
Conf.dr.ing. Robert-Gabriel Lupu

Studenți:
Bogdan Bălău
Vlad Puf

Iași, 2020

Contents

3D Multiplayer Checkers with LeapMotion

Rezumat

Proiectul constă în realizarea unui joc de dame (eng. Checkers) 3D în care cei doi concurenți realizează mutările, pe rand, cu ajutorul unui controller Leap Motion. Procesarea datelor oferite de Leap Motion s-a realizat în mediul de dezvoltare Unity, folosind limbajul C#.

Motivația pentru a face acest proiect este reprezentată de dorința de a crea un joc de strategie multiplayer interactiv. De asemenea, acest tip de aplicație ar putea fi folosita și de către persoanele care suferă de afecțiuni ce țin de controlul membrelor superioare, afecțiuni care afectează capacitatea de a prinde un obiect și de a-l tine în mână. Cu puțin mai multa procesare a datelor provenite de la Leap Motion (spre exemplu o mediere a ultimelor poziții ale mâinii), o persoana cu boala Parkinson ar putea folosi acest mijloc de interacțiune cu calculatorul pentru a juca acest joc, fapt greu de realizat folosind un joc convențional cu piese fizice sau un dispozitiv de tip pointer obișnuit (mouse).

Dezvoltarea aplicației a cuprins două etape. Prima etapă a fost cea de realizare efectivă a jocului folosind controale obișnuite (mouse, tastatura) iar cea de-a doua etapă, cea de integrare a părții de interacțiune prin intermediul Leap Motion. Astfel, toate mutările din joc se realizează prin gesturile mâinii, interpretate de către modulul Leap Motion și traduse în mișcări pe tabla de joc.

Pentru crearea jocului s-a folosit mediul de dezvoltare Unity, iar pentru designul tablei de joc precum și al pieselor s-a folosit un asset gratuit disponibil pe „Unity Asset Store”.

În ceea ce privește interacțiunea om-calculator, mutarea pieselor aferente fiecărei runde se realizează prin intermediul unui program ce se folosește de modulele software oferite de producătorul de dispozitive electronice și senzori, Leap Motion. Aceste module sunt destinate platformei de dezvoltare software Unity și sunt folosite pentru a recunoaște gesturi executate de către utilizator în câmpul vizual al camerelor infra-roșu ale controller-ului folosit. Altfel spus, aplicația identifică poziția palmei jucătorului, iar pe baza mișcării acesteia și a mișcărilor degetelor se realizează manipularea pieselor.

Capitolul 1. Planificarea proiectului pe parcursul semestrului

Săptămânile 1-3

- Documentare cu privire la potențialele teme (au intrat in discutie urmatoarele: Lego 3D, Sah 3D, Tetris etc) respectiv dispozitivele ce pot fi folosite pentru dezvoltărea proiectului (Kinect, Leap Motion, Tobii etc). De asemenea, au prezentat interes mediile de programare ce ar putea fi folosite precum și limbajele de programare.
- Alegerea proiectului - implementarea unui joc de dame care sa poată fi jucat de către doi jucători, dezvoltat în Unity, interacțiunea om-calculator bazându-se pe folosirea controller-ului Leap Motion.
- Instalarea mediului de dezvoltare Unity si testarea cu exemple de baza.
- Acomodarea cu mediul de dezvoltare folosit si cu facilitatile/limitarile pe care acesta le prezinta dar si cu limbajul de programare pentru scripturile ce stau la baza functionalitatii aplicatiilor (C#).
- S-a căutat un asset potrivit pentru compunerea efectivă a jocului și importarea acestuia în mediul de programare folosit. Assetul ales contine tabla de joc cu rama, piesele pentru jocul de dame cu o fata ce reprezinta piesa obisnuita si o fata ce reprezinta piese – rege (i se permite si mutari inapoi) dar si piese de sah care nu au fost folosite in cadrul acestui proiect.
- Achiziționarea controller-ului Leap Motion, configurarea și testarea acestuia. La fel ca mediul de dezvoltare Unity, Leap Motion a reprezentat un dispozitiv nou pentru noi cu care nu mai intrasem in contact prin prisma dezvoltarii concrete de aplicatii pe baza acestuia. Astfel a fost nevoie de o perioada de acomodare, de rulare de aplicatii cu scop demostrativ si construirea unor proiecte simple care sa verifice functionalitatile necesare.
- Planificarea muncii pentru restul săptămânilor rămase.

Săptămânile 4-7

- Documentare amănunțită cu privire la tema propusă – identificarea unui set de reguli ce urmeaza a fi implementat pentru joc si modul de desfasurare a acestuia.
- Realizarea unui proiect pilot.
- Alegerea unui sistem de control a versiunilor pentru un mod de lucru mai facil in echipa, GitHub. Astfel am putut lucra in paralel la proiect fara a fi nevoie sa transferam fisiere intre noi sau a lucra pe rand.
- Crearea de Repository in GitHub

Săptămânile 8-12

- Realizarea mediului de joc – amplasarea facila a tablei de joc intr-o pozitie centrata fata de originea mediul de dezvoltare Unity si in cadrul camerei ce reprezinta perspectiva pe care o are jucatorul
- Așezarea pieselor pe tabla de joc, pe mijlocul casutelor corespunzatoare si in functie de tipul de reguli ales (exista versiuni cu cate 2 si cu cate 3 randuri de piese).
- Implementarea mutărilor pieselor pe tabla de joc folosind initial ca input dispozitivul de tip pointer mouse. La apasarea butonului din stanga a mouseului se selecta piesa de la pozitia indicata de pointer si la eliberarea butonului se incerca mutarea piesei selectate la noua pozitie indicata de catre pointer.
- Adaugarea regulilor de joc astfel incat sa se execute doar mutari valide – la incercarea unei mutari se verifica toate conditiile necesare (vor fi prezentate intr-un capitol separat).
- Importarea modulelor de Leap Motion si Interaction Engine in proiectul de Unity si utilizarea modulelor de modele de maini si scripturile potrivite pentru a facilita functionalitatile puse la indemana de catre Interaction Engine.
- Definirea spatiului de interactiune virtual pe baza unghiului de la camera modulului Leap Motion si ajustarea acestuia pentru a acoperi toata tabla de joc, oferind o sensibilitate potrivita la miscare
- Inlocuirea interactiunii cu piesele de joc de la mouse la gesturi interpretate de Leap Motion – pozitia piesei de joc este aleasa pe baza pozitiei varfului degetului aratator iar ridicarea piesei se face prin apropierea oricaror 2 degete (pinch). Incercarea de asezare a piesei se face prin departarea degetelor folosite pentru ridicare (release pinch).
- Întocmirea unui schelet pentru documentatie.
- Finalizarea părții practice și testarea acesteia pentru a identifica posibile buguri si situatii neasteptate.

Săptămânile 13-14

- Prezentarea inițială a aplicației (Saptamana 13).
- Pregătirea prezentării Power Point și a discursului.
- Finalizarea tehnoredactării documentației
- Sustinerea prezentarii si a demo-ului.

Capitolul 2. Implementarea jocului de dame

2.1. Despre jocul de dame

Jocul de dame este un joc de strategie, dezvoltat în Rusia, ce așează față în față doi jucători. În poziția de start, piesele sunt așezate în capete diferite ale tablei de joc, fiecare dintre oponenți având câte doisprezece piese. Jocul se încheie în momentul în care unul dintre jucători rămâne fără nici o piesă sau în momentul când nu se mai pot efectua mutări, rezultatul fiind remiza.

Fiecare rundă presupune ca jucătorul al cărui rând este să mute una dintre piese pe una din pozițiile diagonale libere din fața acesteia. În cazul în care locația anterior menționată este ocupată de o piesă a adversarului, iar locația următoare acesteia este liberă, mutarea trebuie făcută pe acea locație (saritura peste piesa adversarului). Această mutare are ca rezultat eliminarea piesei adversarului, de asemenea, jucătorul curent va trebui să mai facă o mutare daca in continuare mai are piese adverse peste care poate sari.

Un eveniment special este momentul în care o piesă a unui jucător ajunge pe ultimul rând al tablei din sensul opus al poziției de start. Această piesă devine rege și are proprietatea că poate fi mutată în orice direcție diagonală (adauga posibilitatea de mutare inapoi pe tabla de joc).

2.2. Mediul de dezvoltare Unity

Unity este un mediu de dezvoltare realizat de compania „Unity Technologies” și lansat în iunie 2005 în care se pot realiza jocuri sau aplicații 2D sau 3D. Acest mediu de dezvoltare a cunoscut un succes deosebit în special în industria jocurilor video, dar și în alte industrii precum cea a filmului, automotive, în inginerie și arhitectură. [1]

Un asset este o reprezentare a unui obiect al cărui design a fost realizat într-un program de editare. Acest obiect poate fi un model 3D, un fișier audio, o imagine sau orice tip de fișier ce poate fi interpretat de Unity. [2]

Design-ul 3D pentru tabla de joc precum și pentru piesele de joc fac parte dintr-un proiect de pe o platforma online[3][5]. Un set complet de modele ce include o tabla de sah, piese de sah, de dame, zaruri etc se gaseste in asset-uri gratuite precum cel numit „Free Little Games Asset Pack” disponibil pe „Unity Asset Store”.

Următorii pași au fost urmați pentru implementarea jocului de dame:

- Pentru început s-a creat tabla de joc. În acest sens, a fost descărcat asset-ul corespunzător și importat în cadrul proiectului ce a fost creat în prealabil. După acest pas, unui obiect de tip tablă i s-au definit aceleași caracteristici ca și ale componentei corespunzătoare din asset.
- În ceea ce privește piesele de joc, pentru acestea au fost create două tipuri de obiecte cărora li s-au atribuit caracteristicile pieselor din asset. Mai mult, aceste obiecte au primit un nume sugestiv – black piece si white piece (piesa neagra si piesa alba).
- După realizarea tipurilor de obiecte pentru tablă și piese s-au creat obiectele efective și s-au așezat în spațiul jocului respectând anumite coordonate ce au fost stabilite în prealabil (coordonatele unuia dintre colțurile tablei de joc a fost ales chiar centrul spațiului jocului (x, y, z) = (0, 0, 0)).
- Pentru fiecare tip de obiect din spațiul jocului a fost realizat câte un script în limbajul de programare C#. Conținutul acestor scripturi este disponibil în anexe.
- Restul dezvoltării proiectului a constat în realizarea interacțiunii om-calculator prezentată în capitolul următor.

Capitolul 3. Realizarea interacțiunii om-calculator

3.1. Elemente de introducere

Interactiunea dintre om si calculator se realizeaza in cadrul aceste aplicatii prin intermediul unui dispozitiv Leap Motion. Acesta este un modul dotat cu doua camere ce receptioneaza doar razele luminoase infrarosii emise de catre LED-uri in infrarosu montate pe dispozitiv si reflectate de obiectele din raza acestuia de actiune.

Pe baza viziunii stereo oferita de catre cele doua camere, acest dispozitiv cauta forma mainilor in imagini si stabileste pozitia si orientarea acestora in spatiu. Pe baza acestor informatii se poate reprezenta apoi mana pe ecranul unui computer sub forma unui schelet de mana care mimeaza miscarile operatorului uman.

Leap Motion pune la dispozitie 4 module pentru a crea interactiunea dintre utilizator si aplicatia dezvoltata:

- ***Core*** este modulul de baza al lucrului cu Leap Motion, fara acesta neputandu-se utiliza dispozitivul. Totodata **Leap Motion Core Assets** furnizeaza cheia pentru interactiunea dintre celelalte module Leap Motion. Utilizand doar modulul Core, se poate randa un set de palme grafice ce pot fi urmarite intr-o scena Unity si sa se ataseze acestora diverse obiecte de interactiune
- ***Interaction Engine*** ofera o reprezentarea mai fidela a palmelor si un control imbunatatit printr-o interactiune euristica cu obiectele din scena Unity. Acest modul permite utilizarea diverselor interactiuni fizice cu obiectele din scena, cum ar fi: apucarea acestora, aruncarea lor, oferirea de feedback la atingerea obiectelor, precum si determinarea proximitatii dintre palme si celelalte obiecte din scena. Astfel, Interaction Engine devine cheia interactiunii dintre utilizator si lumea 3D virtuala creata in Unity.
- ***Graphic Renderer*** permite randarea foarte fidela a curburilor 3D ale mainilor oferind si o interfata dinamica pentru interactiunea mainilor cu obiectele din scena. Acest modul poate fi folosit impreuna cu Interaction Engine pentru interactiunea dintre utilizator si o aplicatie cu efecte tridimensionale.
- Modulul ***Hands*** furnizeaza instrumentele necesare pentru a vizualiza 3D palmele cu care utilizatorul executa interactiunea cu obiectele.

3.2. Modul de interactiune in proiectul curent

Pentru proiectul de fata, au fost integrate in cadrul aplicatiei Unity, toate modulele mentionate anterior: Leap Motion Core Assets, Interaction Engine, Graphic Renderer si Hands Module.

Pentru a putea utiliza API-ul pus la dispozitie de Interaction Engine [6] a fost nevoie de a elimina mainile rigide din modulul Core si folosirea mainilor din Interaction Engine, la care sa fie atasat un script manager care sa preia informatiile de la controller.

Interactiunea cu piesele de joc nu se face prin intermediul unui control de tip obiecte rigide ce intra in contact unele cu altele. In acest proiect, interactiunea se face doar pe baza pozitiei varfului degetului aratator de la prima mana prezenta in cadru si pe baza gesturilor.

Mutarea pieselor de joc si orice update de pozitie a acestora se face prin software, operand in mod direct pe vectorul de coordonate al pieselor in concordanta cu pozitia mainii si pozitia necesara pe tabla de joc.

In cadrul unei ture, jucatorul trebuie sa apropie varful degetului aratator de piesa pe care intentioneaza sa o mute. Cu o apropiere suficienta a oricaror doua degete de la aceeasi mana, jucatorul isi exprima intentia de a muta acea piesa.

Piesa va urmari varful degetului pana cand jucatorul distanteaza suficient de mult degetele cu care a ridicat piesa de joc. In acel moment se incearca mutarea piesei din pozitia initiala in pozitia finala. Se verifica toate regulile elaborate si in cazul in care mutarea este permisa, piesa se muta la noua pozitie si runda acestui jucator se incheie.

Camera se muta din perspectiva oponentului iar modelele ce tin de Leap Motion sunt si ele rotite astfel incat mainile sa apara din noua perspectiva. In cazul rundei jucatorului negru, din cauza acestei rotatii, indicii din spatiu a pozitiei mainii si indicii de pe tabla de joc sunt oglinditi si necesita o corectie de remapare. Dupa acesti pasi, oponentul isi incepe tura de joc.

In cazul in care regulile de mutare a piesei nu sunt respectate, piesa va fi mutata in mod automat pe pozitia de pe care a fost luata iar jucatorul trebuie sa reia tura de la inceput.

De observat este faptul ca in cazul in care jucatorul are disponibila in runda sa o mutare ce presupune saltul peste o piesa a adversarului, acesta este obligat sa joace aceasta mutare. Astfel, piesele care nu indeplinesc conditia de a executa un salt nu vor fi validate pentru mutare si nu vor urmari degetul jucatorului.

3.3. Identificarea pozitiei varfului degetului pe tabla de joc

Intrucat Leap Motion nu pune la dispozitie o interfata prin care sa ofere coordonatele mainii raportate la spatiul din Unity si respectiv al jocului, a fost necesara o mapare de la spatiul fizic de interactiune de deasupra dispozitivului la spatiul de joc. Astfel, pozitia varfului degetului in spatiul fizic este reprezentata in milimetri fata de dispozitiv (pe cele 3 axe, X, Y si Z).

Pe baza masuratorilor efectuate in colturile tablei de joc si la inaltimea la care mana urmeaza sa interactioneze cu piesele, s-a dedus o functie de mapare pe fiecare dintre cele 3 axe. La fiecare frame primit de la Leap Motion, primul pas este de a transforma pozitia varfului degetului in pozitie in planul jocului.

Dupa filtrari, corectii si aproximari asupra datelor de intrare in milimetri s-a reusit obtinerea pozitiei deasupra unei casute de joc sub forma unei matrici de dimensiune 8 x 8. Astfel ne putem da seama in software de piesa ce se doreste a fi mutata si pozitia pe care se doreste sa ajunga aceasta.

3.4. Simularea unui eveniment de tipul „mouse click” si „click release”

Pentru o compatibilitate cu modul de joc prin intermediul unui mouse conventional, s-a dorit ca modul de interactiune pe baza mainilor sa fie unul identic cu cel pe baza mouseului.

Astfel s-a pus problema simularii unui eveniment de apasarie a butonului si a unui eveniment de eliberare a butonului.

Gradul de apropiere intre degete a fost ales ca mod de selectare a piesei de joc, dupa cum urmeaza:

- Daca jucatorul nu a selectat inca o piesa si apropie degetele suficient de mult, selecteaza piesa de la pozitia varfului degetului aratator.
- Cat timp degetele sunt aproapiate, muta piesa dupa deget pentru feedback vizual in ceea ce priveste deplasarea acesteia pe placa de joc.
- In momentul in care degetele se distanteaza suficient de mult, incearca mutarea piesei la pozitia indicata in acest moment de catre varful degetului aratator.

Capitolul 4. Probleme întâlnite pe parcursul dezvoltării proiectului

4.1. Dificultati

Dificultatea principala a realizat-o detectarea piesei care se doreste a fi mutata la un moment dat. Aceasta dificultate s-a datorat lipsei unei informatii de pozitie a mainii in spatiul de joc virtual.

Depasirea acestei probleme a presupus masuratori manuale in diferite puncte ale spatiului de joc si implementarea unei mapari pe baza relatiei dintre punctele din spatiul real (in milimetri fata de Leap Motion) si punctele din spatiul virtual.

Un alt pas dificil a fost stabilirea gestului care urma sa joace rolul de selectare a pisei de joc. In discutie au intrat urmatoarele variante:

- Apropierea degetelor de la aceeasi mana (pinch)
- Aparitia in scena a celeilalte maini
- Stationarea pentru mai mult timp (2 secunde) asupra aceleasi piese de joc
- Atingerea efectiva a piesei de joc cu degetul aratator

Pentru consistenta, rapiditate si logica, varianta finala a fost folosirea aceleasi maini prin apropierea a oricare doua degete. Avantajele acestei variante sunt reprezentate de rapidite (mana este deja urmarita in cadru si nu se asteapta 2 secunde pentru alegerea piesei), logica (in cadrul jocului in format fizic piesa se apuca intre degetul mare si degetul aratatot) si stabilitate si acuratete crescute.

4.2. Directii de imbunatatire

Pe baza experientei cumulate am identificat si posibile modalitati de imbunatatire a experiente in cadrul acestui joc. Dintre acestea amintim:

- Medierea ultimelor valori ale pozitiei mainii pentru a scapa de senzatia de tremurat
- Crearea unei functii de mapare care sa tina cont si de pozitia pe inaltime a mainii astfel incat sa se poata interactiona cu piesele si de la o distanta mai mare cu o acuratete mai buna
- Adaugarea de gesturi pentru rotirea camerei in spatiul de joc
- Implementarea unui meniu interactiv (exemplu in palma)
- Implementarea de socketuri care sa permita jocul intre doua calculatoare aflate in aceeasi retea
- Posibilitatea de a juca importiva calculatorului

Capitolul 5. Bibliografie

[1] https://en.wikipedia.org/wiki/Unity_(game_engine), accesat ianuarie 2020.

[2] https://docs.unity3d.com/Manual/AssetWorkflow.html, accesat ianuarie 2020.

[3] https://www.dropbox.com/s/s8h0kd0dhjqkv1q/checkers_art.rar?dl=0, accesat ianuarie 2020.

[4] https://www.youtube.com/playlist?list=PLLH3mUGkfFCVXrGLRxfhst7pffE9o2SQO, accesat ianuarie 2020.

[5] https://www.epitome.cc/#/asset, accesat ianuarie 2020

[6] https://leapmotion.github.io/UnityModules/namespace_leap_1_1_unity.html, accesat ianuarie 2020.

Capitolul 6. Anexe.

Anexa 1 – Piece.cs

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

public class Piece : MonoBehaviour
{
    public bool isWhite;
    public bool isKing;

    public bool IsForceToMove(Piece[,] board,int x, int y)
    {
        if(isWhite || isKing)
        {
            // Top left
            if(x >= 2 && y <= 5)
            {
                Piece p = board[x - 1, y + 1];
                // Is there a piece to kill?
                if(p != null && p.isWhite != isWhite)
                {
                    // Is there a free space to jump?
                    if (board[x - 2, y + 2] == null)
                        return true;
                }
            }
            // Top right
            if (x <= 5 && y <= 5)
            {
                Piece p = board[x + 1, y + 1];
                // Is there a piece to kill?
                if (p != null && p.isWhite != isWhite)
                {
                    // Is there a free space to jump?
                    if (board[x + 2, y + 2] == null)
                        return true;
                }
            }
        }
        if(!isWhite || isKing)
        {
            // Bottom left
            if (x >= 2 && y >= 2)
            {
                Piece p = board[x - 1, y - 1];
                // Is there a piece to kill?
                if (p != null && p.isWhite != isWhite)
                {
                    // Is there a free space to jump?
                    if (board[x - 2, y - 2] == null)
                        return true;
                }
            }
            // Bottom right
            if (x <= 5 && y >= 2)
            {
                Piece p = board[x + 1, y - 1];
                // Is there a piece to kill?
                if (p != null && p.isWhite != isWhite)
```

```
                {
                    // Is there a free space to jump?
                    if (board[x + 2, y - 2] == null)
                        return true;
                }
            }
        }
        return false;
    }
    public bool ValidMove(Piece[,] board, int x1, int y1, int x2, int y2)
    {
        //If you are moving on top of another piece
        if (board[x2, y2] != null)
            return false;

        int deltaMove = Mathf.Abs(x1 - x2);
        int deltaMoveY = y2 - y1;

        // For white
        if (isWhite || isKing)
        {

            if(deltaMove == 1)
            {
                if (deltaMoveY == 1)
                    return true;
            }
            else if(deltaMove == 2)
            {
                if (deltaMoveY == 2)
                {
                    Piece p = board[(x1 + x2) / 2, (y1 + y2) / 2];

                    if (p != null && p.isWhite != isWhite)
                        return true;
                }
            }
        }
        // For black
        if (!isWhite || isKing)
        {

            if(deltaMove == 1)
            {
                if (deltaMoveY == -1)
                    return true;
            }
            else if(deltaMove == 2)
            {
                if (deltaMoveY == -2)
                {
                    Piece p = board[(x1 + x2) / 2, (y1 + y2) / 2];

                    if (p != null && p.isWhite != isWhite)
                        return true;
                }
            }
        }

        return false;
    }
}
```

Anexa 2 – CheckersBoard.cs

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

public class CheckersBoard : MonoBehaviour
{

    public Piece[,] pieces = new Piece[8, 8];

    public GameObject whitePiecePrefab;
    public GameObject blackPiecePrefab;
    public GameObject LMC; // Leap Motion Controller
    private Vector3 boardOffset = new Vector3(-4.0f, 0, -4.0f);
    private Vector3 pieceOffset = new Vector3(0.5f, 0, 0.5f);

    private Vector3 CameraPosW = new Vector3(0f, 6f, -5f);
    //private Vector3 CameraRotW = new Vector3(60f, 0f, 0f);
    private Vector3 CameraPosB = new Vector3(0f, 6f, 5f);
    private Vector3 CameraRot = new Vector3(0f, 180f, 0f);

    public bool isWhite;
    private bool isWhiteTurn;

    private Vector2 mouseOver;
    private Vector2 leapOver;
    private Vector2 startDrag;
    private Vector2 endDrag;

    public float pinchStrength = 0;
    public int clickStatus = 0; //0 - nothing happend  1 - left click pressed  2 - left click
released
    public int lastClickStatus = 0; //0 - nothing happend  1 - left click pressed  2 - left
click released

    private Piece selectedPiece;
    private List<Piece> forcedPieces;
    private bool hasKilled;

    bool k = true;
    Vector3 cameraPosition = new Vector3(0, 0, 0);
    Vector3 indexFingerTipBoardPos = new Vector3(0, 0, 0);
    public Collider targetCollider;

    Leap.Controller c;

    // Start is called before the first frame update
    private void Start()
    {
        LMC = GameObject.Find("Leap Motion Controller");

        isWhiteTurn = true;
        forcedPieces = new List<Piece>();
        GenerateBoard();

        c = new Controller();

        cameraPosition = Camera.main.gameObject.transform.position;
        Debug.Log("Camera Position    x: " + cameraPosition.x + " y: " + cameraPosition.y + "
z: " + cameraPosition.z);
    }
```

```
// Update is called once per frame
private void Update()
{
    //Press ESC button to exit the game
    if (Input.GetKey("escape"))
    {
        Application.Quit();
    }
    //Press N button for new game (Restart)
    if (Input.GetKeyDown(KeyCode.N))
    {
        Application.LoadLevel(0);
    }
    UpdateMouseOver();

    if((isWhite)?isWhiteTurn:!isWhiteTurn)
    {
        int x = (int)leapOver.x;
        int y = (int)leapOver.y;

        if (!isWhite)
        {
            leapOver.x = 7 - leapOver.x;
            leapOver.y = 7 - leapOver.y;
            x = 7 - x;
            y = 7 - y;
            indexFingerTipBoardPos.x = 8 - indexFingerTipBoardPos.x;
            indexFingerTipBoardPos.z = 8 - indexFingerTipBoardPos.z;
        }

        if (selectedPiece != null)
            UpdatePieceDrag(selectedPiece);

        GetClickStatus();
        if(clickStatus == 1)
        {
            SelectPiece(x, y);
        }
        if(clickStatus == 2)
        {
            TryMove((int)startDrag.x, (int)startDrag.y, x, y);
        }
    }
}
private void GetClickStatus()
{
    switch(clickStatus)
    {
        case 0:
            if (lastClickStatus == 1 && pinchStrength < 0.4)
            {
                clickStatus = 2;
            }
            else
                if (lastClickStatus == 0 && pinchStrength > 0.7)
                {
                    clickStatus = 1;
                }
            break;
        case 1:
            clickStatus = 0;
            lastClickStatus = 1;
```

```
                break;
            case 2:
                clickStatus = 0;
                lastClickStatus = 0;
                break;
            default:
                break;
        }
    }
    private void UpdateMouseOver()
    {
        if (!Camera.main)
        {
            Debug.Log("Unable to find main camera");
            return;
        }

        RaycastHit hit;
        if (Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out hit, 25.0f,
LayerMask.GetMask("Board")))
        {
            mouseOver.x = (int)(hit.point.x - boardOffset.x);
            mouseOver.y = (int)(hit.point.z - boardOffset.z);
        }
        else
        {
            mouseOver.x = -1;
            mouseOver.y = -1;
        }
        Leap.Frame frame = c.Frame(0);

        List<Hand> handList = new List<Hand>();
        for (int h = 0; h < frame.Hands.Count; h++)
        {
            Hand leapHand = frame.Hands[h];
            handList.Add(leapHand);
        }

        Leap.Finger indexFinger;
        Vector3 fingerTipPosUnity = new Vector3(0, 0, 0);
        Vector3 fingerTipDirUnity = new Vector3(0, 0, 0);
        Vector3 fingerToScreen = new Vector3(0, 0, 0);

        Vector3 closestPoint = new Vector3(0, 0, 0);


        if (handList != null && frame.Hands.Count > 0)
        {
            indexFinger = frame.Hands[0].Fingers[(int)Finger.FingerType.TYPE_INDEX];
            pinchStrength = frame.Hands[0].PinchStrength;
            //Debug.Log("Pinch strength: " + pinchStrength);
            if (k)// && indexFinger.IsExtended)
            {
                Vector fingerTipPos = indexFinger.TipPosition;
                indexFingerTipBoardPos.x = (fingerTipPos.x / 34) + 4;
                indexFingerTipBoardPos.y = (fingerTipPos.y / 34) + 4 - 9;
                indexFingerTipBoardPos.z = ((fingerTipPos.z * (-1)) / 34) +4;
                //Debug.Log("Tip Position    x: " + ((fingerTipPos.x/34)+4) + " y: " +
((fingerTipPos.y / 34) + 4) + " z: " + (((fingerTipPos.z*(-1)) / 34) + 4));//
                leapOver.x = (int) Math.Floor((fingerTipPos.x / 34) + 4);
                leapOver.y = (int) Math.Floor(((fingerTipPos.z * (-1)) / 34) + 4);
                if (leapOver.x < 0 || leapOver.x > 7 || leapOver.y < 0 || leapOver.y > 7)
                {
```

```
                leapOver.x = -1;
                leapOver.y = -1;
            }
            Debug.Log("Finger on board position: " + leapOver.x + " " + leapOver.y);
        }
    }
}
private void UpdatePieceDrag(Piece p)
{
    if(!Camera.main)
    {
        Debug.Log("Unable to find main camera");
        return;
    }
    indexFingerTipBoardPos.x = indexFingerTipBoardPos.x + boardOffset.x;
    indexFingerTipBoardPos.z = indexFingerTipBoardPos.z + boardOffset.z;
    p.transform.position = indexFingerTipBoardPos;
}

private void SelectPiece(int x, int y)
{
    //out of bounds
    if (x < 0 || x >= 8 || y < 0 || y >= 8)
        return;

    Piece p = pieces[x, y];

    if (p != null && p.isWhite == isWhite)
    {
        if (forcedPieces.Count == 0)
        {
            selectedPiece = p;
            //startDrag = mouseOver;
            startDrag = leapOver;
        }
        else
        {
            // Look for the piece under our forced pieces list
            if (forcedPieces.Find(fp => fp == p) == null)
                return;

            selectedPiece = p;
            //startDrag = mouseOver;
            startDrag = leapOver;
        }
    }
}
private void TryMove(int x1, int y1, int x2, int y2)
{
    forcedPieces = ScanForPossibleMove();
    // Multipayer Support
    startDrag = new Vector2(x1, y1);
    endDrag = new Vector2(x2, y2);
    selectedPiece = pieces[x1, y1];

    // Out of bounds
    if (x2 < 0 || x2 >= 8 || y2 < 0 || y2 >= 8)
    {
        if (selectedPiece != null)
            MovePiece(selectedPiece, x1, y1);

        startDrag = Vector2.zero;
        selectedPiece = null;
```

```
            return;
        }
        if(selectedPiece != null)
        {
            // If it has not moved
            if(endDrag == startDrag)
            {
                MovePiece(selectedPiece, x1, y1);
                startDrag = Vector2.zero;
                selectedPiece = null;
                return;
            }

            // Check if its a valid move
            if(selectedPiece.ValidMove(pieces,x1,y1,x2,y2))
            {
                // Did we kill anything
                // If this is a jump
                if(Mathf.Abs(x2 - x1) == 2)
                {
                    Piece p = pieces[(x1 + x2) / 2, (y1 + y2) / 2];

                    if(p!= null)
                    {
                        pieces[(x1 + x2) / 2, (y1 + y2) / 2] = null;
                        Destroy(p.gameObject);
                        hasKilled = true;
                    }
                }

                // Were we suposed to kill anything?
                if(forcedPieces.Count != 0 && !hasKilled)
                {
                    MovePiece(selectedPiece, x1, y1);
                    startDrag = Vector2.zero;
                    selectedPiece = null;
                    return;
                }

                pieces[x2, y2] = selectedPiece;
                pieces[x1, y1] = null;

                MovePiece(selectedPiece, x2, y2);

                EndTurn();
            }
            else
            {
                MovePiece(selectedPiece, x1, y1);
                startDrag = Vector2.zero;
                selectedPiece = null;
                return;
            }
        }
    }
    private void EndTurn()
    {
        int x = (int)endDrag.x;
        int y = (int)endDrag.y;

        // Promoting to king
        if(selectedPiece != null)
        {
```

```
        if(selectedPiece.isWhite && !selectedPiece.isKing && y == 7)
        {
            selectedPiece.isKing = true;
            selectedPiece.transform.Rotate(Vector3.right * 180);
        }
        else if (!selectedPiece.isWhite && !selectedPiece.isKing && y == 0)
        {
            selectedPiece.isKing = true;
            selectedPiece.transform.Rotate(Vector3.right * 180);
        }
    }
    selectedPiece = null;
    startDrag = Vector2.zero;

    if (ScanForPossibleMove(selectedPiece, x, y).Count != 0 && hasKilled)
        return;

    isWhiteTurn = !isWhiteTurn;
    isWhite = !isWhite;
    hasKilled = false;
    CheckVictory();
    ChangeCameraView(isWhiteTurn);
}
private void ChangeCameraView(bool isWhiteTurn)
{
    if(isWhiteTurn)
    {
        Camera.main.gameObject.transform.position = CameraPosW;
        //Camera.main.gameObject.transform.Rotate(0, 180, 0);
        Camera.main.gameObject.transform.eulerAngles = new Vector3(
            Camera.main.gameObject.transform.eulerAngles.x,
            Camera.main.gameObject.transform.eulerAngles.y + 180,
            Camera.main.gameObject.transform.eulerAngles.z);
    }
    else
    {
        Camera.main.gameObject.transform.position = CameraPosB;
        //Camera.main.gameObject.transform.Rotate(120, 180, 0);
        Camera.main.gameObject.transform.eulerAngles = new Vector3(
            Camera.main.gameObject.transform.eulerAngles.x,
            Camera.main.gameObject.transform.eulerAngles.y + 180,
            Camera.main.gameObject.transform.eulerAngles.z);
    }
    LMC.transform.eulerAngles = new Vector3(
        LMC.transform.eulerAngles.x,
        LMC.transform.eulerAngles.y + 180,
        LMC.transform.eulerAngles.z);
}
private void CheckVictory()
{
    var ps = FindObjectsOfType<Piece>();
    bool hasWhite = false;
    bool hasBlack = false;

    for(int i = 0; i < ps.Length; i++)
    {
        if (ps[i].isWhite)
            hasWhite = true;
        else
            hasBlack = true;
    }
    if (!hasWhite)
        Victory(false);
```

```
        if (!hasBlack)
            Victory(true);
    }
    private void Victory(bool isWhite)
    {
        if (isWhite)
            Debug.Log("White has won!");
        else
            Debug.Log("Black has won!");
    }
    private List<Piece> ScanForPossibleMove(Piece p, int x, int y)
    {
        forcedPieces = new List<Piece>();
        if (pieces[x, y].IsForceToMove(pieces, x, y))
            forcedPieces.Add(pieces[x, y]);
        return forcedPieces;
    }
    private List<Piece> ScanForPossibleMove()
    {
        forcedPieces = new List<Piece>();

        // Check all the pieces
        for (int i = 0; i < 8; i++)
            for (int j = 0; j < 8; j++)
                if (pieces[i, j] != null && pieces[i, j].isWhite == isWhiteTurn)
                    if (pieces[i, j].IsForceToMove(pieces, i, j))
                        forcedPieces.Add(pieces[i, j]);

        return forcedPieces;
    }
    private void GenerateBoard()
    {
        //Generate white team
        for (int y = 0; y < 3; y++)
        {
            bool oddRow = (y % 2 == 0);
            for (int x = 0; x < 8; x += 2)
            {
                //Generate the piece
                GeneratePiece((oddRow) ? x : x+1, y);
            }
        }
        //Generate black team
        for (int y = 7; y > 4; y--)
        {
            bool oddRow = (y % 2 == 0);
            for (int x = 0; x < 8; x += 2)
            {
                //Generate the piece
                GeneratePiece((oddRow) ? x : x + 1, y);
            }
        }
    }
    private void GeneratePiece(int x, int y)
    {
        bool isPieceWhite = (y > 3) ? false : true;
        GameObject go = Instantiate((isPieceWhite) ? whitePiecePrefab : blackPiecePrefab) as
GameObject;
        go.transform.SetParent(transform);
        Piece p = go.GetComponent<Piece>();
        pieces[x, y] = p;
        MovePiece(p, x, y);
    }
```

```
    private void MovePiece(Piece p, int x, int y)
    {
        p.transform.position = (Vector3.right * x) + (Vector3.forward * y) + boardOffset +
pieceOffset;
    }
}
```