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

Cuprins

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 rând, 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 ține în mână. Cu puțin mai multă procesare a datelor provenite de la Leap Motion (spre exemplu o mediere a ultimelor poziții ale mâinii), o persoană 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 în discuție următoarele: Lego 3D, Șah 3D, Tetris etc) respectiv dispozitivele ce pot fi folosite pentru dezvoltarea 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 și testarea cu exemple de bază.

- Acomodarea cu mediul de dezvoltare folosit și cu facilitățile/limitările pe care acesta le prezintă, dar și cu limbajul de programare pentru scripturile ce stau la baza funcționalității aplicațiilor (C#).

- S-a căutat un asset potrivit pentru compunerea efectivă a jocului și importarea acestuia în mediul de programare folosit. Assetul ales conține tabla de joc cu ramă, piesele pentru jocul de dame cu o față ce reprezintă piesa obișnuită și o față ce reprezinta piese – rege (i se permite și mutări înapoi), dar și piese de șah care nu au fost folosite în 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 în contact prin prisma dezvoltării concrete de aplicații pe baza acestuia. Astfel a fost nevoie de o perioada de acomodare, de rulare de aplicații cu scop demonstrativ și construirea unor proiecte simple care să verifice funcționalitățile necesare.

- Planificarea muncii pentru restul săptămânilor rămase.

Săptămânile 4-7

- Documentare amănunțită cu privire la tema propusă – identificarea unui set de reguli ce urmează a fi implementat pentru joc și modul de desfășurare a acestuia.

- Realizarea unui proiect pilot.

- Alegerea unui sistem de control a versiunilor pentru un mod de lucru mai facil în echipa, GitHub. Astfel am putut lucra în paralel la proiect fără a fi nevoie sa transferam fișiere între noi sau a lucra pe rand.

- Crearea de Repository în GitHub

Săptămânile 8-12

- Realizarea mediului de joc – amplasarea facilă a tablei de joc într-o poziție centrata față de originea mediul de dezvoltare Unity și în cadrul camerei ce reprezintă perspectiva pe care o are jucătorul.
- Așezarea pieselor pe tabla de joc, pe mijlocul căsuțelor corespunzătoare și în funcție de tipul de reguli ales (exista versiuni cu cate 2 și cu cate 3 rânduri de piese).
- Implementarea mutărilor pieselor pe tabla de joc folosind inițial ca input dispozitivul de tip pointer mouse. La apăsarea butonului din stânga a mouseului se selecta piesa de la poziția indicată de pointer și la eliberarea butonului se încercă mutarea piesei selectate la noua poziție indicata de către pointer.
- Adăugarea regulilor de joc astfel încât să se execute doar mutări valide – la încercarea unei mutări se verifică toate condițiile necesare (vor fi prezentate într-un capitol separat).
- Importarea modulelor de Leap Motion și Interaction Engine în proiectul de Unity și utilizarea modulelor de modele de mâini și scripturile potrivite pentru a facilita funcționalitățile puse la îndemână de către Interaction Engine.
- Definirea spațiului de interacțiune virtual pe baza unghiului de la camera modulului Leap Motion și ajustarea acestuia pentru a acoperi toată tabla de joc, oferind o sensibilitate potrivita la mișcare.
- Înlocuirea interacțiunii cu piesele de joc de la mouse la gesturi interpretate de Leap Motion – poziția piesei de joc este aleasa pe baza poziției vârfului degetului arătător iar ridicarea piesei se face prin apropierea oricăror 2 degete (pinch). Încercarea de așezare a piesei se face prin depărtarea degetelor folosite pentru ridicare (release pinch).
- Întocmirea unui schelet pentru documentație.
- Finalizarea părții practice și testarea acesteia pentru a identifica posibile erori și situații neașteptate.

Săptămânile 13-14

- Prezentarea inițială a aplicației (Săptămâna 13).
- Pregătirea prezentării Power Point și a discursului.
- Finalizarea tehnoredactării documentației.
- Susținerea prezentării și 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 (săritura peste piesa adversarului). Această mutare are ca rezultat eliminarea piesei adversarului, de asemenea, jucătorul curent va trebui să mai facă o mutare dacă în 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 (adaugă posibilitatea de mutare înapoi 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 șah, piese de șah, de dame, zaruri etc se găsește în 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

Interacțiunea dintre om și calculator se realizează în cadrul aceste aplicații prin intermediul unui dispozitiv Leap Motion. Acesta este un modul dotat cu două camere ce recepționează doar razele luminoase infraroșii emise de către LED-uri în infraroșu montate pe dispozitiv și reflectate de obiectele din raza acestuia de acțiune.

Pe baza viziunii stereo oferita de către cele două camere, acest dispozitiv cauta forma mâinilor în imagini și stabilește poziția și orientarea acestora în spațiu. Pe baza acestor informații se poate reprezenta apoi mâna pe ecranul unui computer sub forma unui schelet de mână care mimează mișcările operatorului uman.

Leap Motion pune la dispoziție 4 module pentru a crea interacțiunea dintre utilizator și aplicația dezvoltata:

- ***Core*** este modulul de baza al lucrului cu Leap Motion, fără acesta neputându-se utiliza dispozitivul. Totodată **Leap Motion Core Assets furnizează** cheia pentru interacțiunea dintre celelalte module Leap Motion. Utilizând doar modulul Core, se poate randa un set de palme grafice ce pot fi urmărite într-o scena Unity și să se atașeze acestora diverse obiecte de interacțiune

- ***Interaction Engine*** **oferă** o reprezentarea mai fidela a palmelor și un control îmbunătățit printr-o interacțiune euristica cu obiectele din scena Unity. Acest modul permite utilizarea diverselor interacțiuni fizice cu obiectele din scena, cum ar fi: apucarea acestora, aruncarea lor, oferirea de feedback la atingerea obiectelor, precum și determinarea proximității dintre palme și celelalte obiecte din scena. Astfel, Interaction Engine devine cheia interacțiunii dintre utilizator și lumea 3D virtuala creata în Unity.

- ***Graphic Renderer*** permite randarea foarte fidela a curburilor 3D ale mâinilor oferind și o interfața dinamica pentru interacțiunea mâinilor cu obiectele din scena. Acest modul poate fi folosit împreuna cu Interaction Engine pentru interacțiunea dintre utilizator și o aplicație cu efecte tridimensionale.

- Modulul ***Hands*** furnizează instrumentele necesare pentru a vizualiza 3D palmele cu care utilizatorul executa interacțiunea cu obiectele.

3.2. Modul de interacțiune în proiectul curent

Pentru proiectul de față, au fost integrate în cadrul aplicației Unity, toate modulele menționate anterior: Leap Motion Core Assets, Interaction Engine, Graphic Renderer și Hands Module.

Pentru a putea utiliza API-ul pus la dispoziție de Interaction Engine [6] a fost nevoie de a elimina mâinile rigide din modulul Core și folosirea mâinilor din Interaction Engine, la care să fie atașat un script manager care să preia informațiile de la controller.

Interacțiunea cu piesele de joc nu se face prin intermediul unui control de tip obiecte rigide ce intră în contact unele cu altele. În acest proiect, interacțiunea se face doar pe baza poziției vârfului degetului arătător de la prima mână prezenta în cadru și pe baza gesturilor.

Mutarea pieselor de joc și orice update de poziție a acestora se face prin software, operand în mod direct pe vectorul de coordonate al pieselor în concordanta cu poziția mâinii și poziția necesara pe tabla de joc.

În cadrul unei ture, jucătorul trebuie să apropie vârful degetului arătător de piesa pe care intenționează să o mute. Cu o apropiere suficienta a oricăror două degete de la aceeași mână, jucătorul își exprima intenția de a muta acea piesa.

Piesa va urmări vârful degetului pana când jucătorul distanțează suficient de mult degetele cu care a ridicat piesa de joc. În acel moment se încearcă mutarea piesei din poziția inițială în poziția finala. Se verifica toate regulile elaborate și în cazul în care mutarea este permisa, piesa se muta la noua poziție și runda acestui jucător se încheie.

Camera se muta din perspectiva oponentului iar modelele ce țin de Leap Motion sunt și ele rotite astfel încât mâinile să apară din noua perspectiva. În cazul rundei jucătorului negru, din cauza acestei rotații, indicii din spațiu a poziției mâinii și indicii de pe tabla de joc sunt oglindiți și necesita o corecție de remapare. După acești pași, oponentul își începe tura de joc.

În cazul în care regulile de mutare a piesei nu sunt respectate, piesa va fi mutata în mod automat pe poziția de pe care a fost luata iar jucătorul trebuie să reia tura de la început.

De observat este faptul că în cazul în care jucătorul are disponibila în runda sa o mutare ce presupune saltul peste o piesa a adversarului, acesta este obligat să joace aceasta mutare. Astfel, piesele care nu îndeplinesc condiția de a executa un salt nu vor fi validate pentru mutare și nu vor urmări degetul jucătorului.

3.3. Identificarea poziției vârfului degetului pe tabla de joc

Întrucât Leap Motion nu pune la dispoziție o interfață prin care să ofere coordonatele mâinii raportate la spațiul din Unity și respectiv al jocului, a fost necesara o mapare de la spațiul fizic de interacțiune de deasupra dispozitivului la spațiul de joc. Astfel, poziția vârfului degetului în spațiul fizic este reprezentata în milimetri față de dispozitiv (pe cele 3 axe, X, Y și Z).

Pe baza măsurătorilor efectuate în colturile tablei de joc și la înălțimea la care mână urmează să interacționeze cu piesele, s-a dedus o funcție de mapare pe fiecare dintre cele 3 axe. La fiecare frame primit de la Leap Motion, primul pas este de a transforma poziția vârfului degetului în poziție în planul jocului.

După filtrări, corecții și aproximări asupra datelor de intrare în milimetri s-a reușit obținerea poziției deasupra unei căsuțe de joc sub forma unei matrici de dimensiune 8 x 8. Astfel ne putem da seama în software de piesa ce se dorește a fi mutata și poziția pe care se dorește să ajungă aceasta.

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

Pentru o compatibilitate cu modul de joc prin intermediul unui mouse convențional, s-a dorit ca modul de interacțiune pe baza mâinilor să fie unul identic cu cel pe baza mouseului. Astfel s-a pus problema simulării unui eveniment de apăsare a butonului și a unui eveniment de eliberare a butonului.

Gradul de apropiere intre degete a fost ales ca mod de selectare a piesei de joc, după cum urmează

- Dacă jucătorul nu a selectat încă o piesa și apropie degetele suficient de mult, selectează piesa de la poziția vârfului degetului arătător.
- Cat timp degetele sunt apropiate, muta piesa după deget pentru feedback vizual în ceea ce privește deplasarea acesteia pe placa de joc.
- În momentul în care degetele se distanțează suficient de mult, încearcă mutarea piesei la poziția indicata în acest moment de către vârful degetului arătător.

Capitolul 4. Probleme întâlnite pe parcursul dezvoltării proiectului

4.1. Dificultăți

Dificultatea principala a realizat-o detectarea piesei care se dorește a fi mutata la un moment dat. Aceasta dificultate s-a datorat lipsei unei informații de poziție a mâinii în spațiul de joc virtual.

Depășirea acestei probleme a presupus măsurători manuale în diferite puncte ale spațiului de joc și implementarea unei mapări pe baza relației dintre punctele din spațiul real (în milimetri față de Leap Motion) și punctele din spațiul virtual.

Un alt pas dificil a fost stabilirea gestului care urma să joace rolul de selectare a piesei de joc. În discuție au intrat următoarele variante:

- Apropierea degetelor de la aceeași mână (pinch)
- Apariția în scena a celeilalte mâini
- Staționarea pentru mai mult timp (2 secunde) asupra aceleași piese de joc
- Atingerea efectiva a piesei de joc cu degetul arătător.

Pentru consistenta, rapiditate și logica, varianta finala a fost folosirea aceleași mâini prin apropierea a oricare două degete. Avantajele acestei variante sunt reprezentate de rapiditate (mâna este deja urmărită în cadru și nu se așteaptă 2 secunde pentru alegerea piesei), logica (în cadrul jocului în format fizic piesa se apucă între degetul mare și degetul arătător) și stabilitate și acuratețe crescute.

4.2. Direcții de îmbunătățire

Pe baza experientei cumulate am identificat și posibile modalități de îmbunătățire a experiente în cadrul acestui joc. Dintre acestea amintim:

- Medierea ultimelor valori ale poziției mâinii pentru a scapă de senzația de tremurat
- Crearea unei funcții de mapare care să tina cont și de poziția pe înălțime a mâinii astfel încât să se poată interacționa cu piesele și de la o distanta mai mare cu o acuratețe mai buna
- Adăugarea de gesturi pentru rotirea camerei în spațiul de joc
- Implementarea unui meniu interactiv (exemplu în palma)
- Implementarea de socket-uri care să permită jocul intre două calculatoare aflate în aceeași rețea
- Posibilitatea de a juca împotriva 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;
                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];
    if (x2 < 0 || x2 >= 8 || y2 < 0 || y2 >= 8) // Out of bounds
    {
        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 + pi-
eceOffset;
    }
}
```