Técnicas avanzadas de HCI con uso de rastreadores de ojos y manos

Trabajo Práctico Especial

Leap Motion

Instituto Tecnológico de Buenos Aires

Buenos Aires, Argentina

2015

Alumnos

• Lucas Andrés Farré	lfarre@itba.edu.ar	53180
• Kevin Hirschowitz Kraus	khirscho@itba.edu.ar	52539
• Alejandro Bezdjian	abezdjia@itba.edu.ar	52108

Docentes

- Marcela Guerrero
- Robert Pucher

1. Introducción

Mediante el uso del dispositivo de *input Leap Motion* desarrollamos un juego que consiste básicamente en colocar cubos de madera en distintas cajas. El usuario se puede mover a lo largo del mapa si se acerca o aleja del dispositivo con su mano a lo largo del eje *z*.

También mediante distintos gestos puede interactuar con los diferentes objetos con los que se enfrenta, las distintas leyendas que aparecen a lo largo del juego ayudan al usuario para sepa qué acciones debe realizar.

2. Funcionalidades

- Arrojar objetos
- Agarrar objetos
- Desplazarse en el mapa
- Talar árboles mediante el uso de *toolkit*
- Escalar montañas
- Manejo de *toolkits*
- Colocar objetos en cajas
- Contador de puntajes

Una de las principales funcionalidades que programamos es el movimiento de la cámara, de acuerdo a la posición de la mano que el dispositivo detecta, decidimos hacia dónde se desplazará la misma con el fin de darle al usuario un mayor panorama y una posibilidad de recorrer y moverse a través del mundo virtual creado. Hablaremos con más detalle acerca de esta implementación en el apartado *Desarrollo*.

Existen objetos que se pueden agarrar con una mano y también el juego cuenta con la funcionalidad de arrojar los mismos. Estos objetos que son "*agarrables*" por el usuario son fácilmente reconocibles para el mismo ya que la misma física del juego los hace identificables en el piso gracias a la gravedad que le otorgamos a los objetos en general. Al crear un mundo virtual similar a nuestro mundo real, las acciones y tipos de movimiento que puede realizar el usuario son casi intuitivas. Es coherente que si hacemos el gesto de agarrar un objeto la representación en el juego realice esta misma acción, lo mismo al girar la cámara o soltar un objeto.

Si el usuario se dirige hacia una zona montañosa tiene permitido subirse a la misma, simulando el comportamiento de escalarla. Dándole así un panorama general de todo el universo virtual creado.

Si el usuario cuando se acerca a un árbol toma un lápiz o cualquier elemento de tipo *toolkit*, el juego transforma este objeto en su mundo en un hacha. Una vez que el usuario se encuentra con el hacha y comienza a realizar gestos de tipo *swipe* entonces los distintos árboles se caen y nos arrojan cubos de madera. Dependiendo del tipo de árbol que talemos nos arrojará distintos tipos de madera que habrá que colocar en las diferentes cajas.

Existen dos tipos de cajas, la caja que espera recibir madera de color más oscuro y otra que recibe bloques más claros. Si arrojamos bloques que no corresponden a una de estas cajas entonces se descontará puntaje, caso contrario el mismo aumentará.

3. Affordances

Decidimos con el fin de guiar al usuario en la usabilidad de nuestro juego agregar captions que le informan al usuario que acciones tiene permitido realizar de acuerdo en que posición del mapa se encuentre.

El objetivo principal del juego es ordenar la totalidad de los objetos con un puntaje máximo, para darnos cuenta de que tan bien lo estamos realizando contamos con contadores encima de cada una de las cajas que se corresponden a los tipos de madera que le ponemos a cada una. Si le pones un cubo que no corresponde con el esperado se descuenta un punto del contador y si le colocamos uno correcto suma al mismo.

4. Gestos

El gesto que diseñamos está enfocado en la tala de árboles, si nos acercamos a uno y empezamos a realizar gestos de tipo *swipe* podemos talar los árboles luego de repetir el movimiento alrededor de 5 veces. Podemos mencionar que este gesto es extremadamente intuitivo para el usuario ya que en el mundo real, cuando uno desea talar un árbol con un hacha, los movimientos que realiza son prácticamente los mismos.

Se hizo uso además del gesto de tipo *circle*, utilizado para reiniciar el juego una vez finalizado. También resulta muy intuitivo ya que el dibujo del mismo coincide con el clásico ícono de *refresh* utilizado por ejemplo en los navegadores web.

Otros gestos que desarrollamos en el juego, pero que no hacen uso de los gestos incluidos en la *API* de *Leap Motion*, son los utilizados para la navegación a través del mapa. Si el usuario mueve su mano hacia la derecha, la cámara se desliza en esa dirección y lo mismo ocurre hacia el otro lado. Si el usuario quiere avanzar, solo debe estirar su brazo

simulando que quiere llegar alcanzar algún objeto. Similar a cualquier gesto deíctico que realiza un bebé cuando quiere alcanzar algún objeto fuera de su alcance.

5. Ergonomía

Intentamos a lo largo de este trabajo práctico realizar un juego con movimientos cotidianos de los usuarios con el fin de evitar movimientos incómodos o que pudieran darle fatiga luego de jugar un determinado tiempo. Los movimientos y habilidades que requiere el juego desarrollado pueden ser fácilmente comprendidas a su vez por adultos como por niños menores de edad, y, en poco tiempo cualquiera de los dos se puede adaptar a la totalidad de las funcionalidades y movimientos.

6. Desarrollo

Se eligió como motor de desarrollo de videojuegos a *Unity 3D*, ya que *Leap Motion* ofrece un *plugin* para el mismo con el cual es posible colocar las manos en la escena de una manera fácil y rápida. Las mismas ya tienen incorporada la física de agarrar, arrojar y empujar objetos. Además, como los scripts de *Unity* se programan en *C#*, *Leap Motion* cuenta también con una librería para este lenguaje que nos permite hacer uso de la *API* vista en clase.

Mostraremos los *scripts* que más hacen uso de la *API*. A continuación podemos ver *CameraMovement.cs*, el mismo es el responsable de que la cámara se desplace al mover las manos, dando como resultado que el usuario se mueva libremente por el mapa.

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

public class CameraMovement : MonoBehaviour {

    private Controller controller;
    // Use this for initialization
    void Start () {
        controller = new Controller ();
        Physics.gravity = Physics.gravity * 2;
    }

    // Update is called once per frame
    void Update () {
                    float   height   =   Terrain.activeTerrain.SampleHeight
(transform.position);
```

```
         transform.position = new Vector3(transform.position.x, height + 8,
transform.position.z);
        Frame frame = controller.Frame ();
        foreach (Hand hand in frame.Hands) {
            float z = hand.PalmPosition.z;
            float x = hand.PalmPosition.x;
            bool handUp = z < -120;
            bool handDown = z > 120;
            Vector3 directionXZ = transform.forward;
            directionXZ.y = 0;
            if(handUp) {
                transform.position += directionXZ * Time.deltaTime * 8;
            } else if(handDown){
                transform.position -= directionXZ * Time.deltaTime * 8;
            }
            if(x < -120) {
                transform.Rotate(0, -30 * Time.deltaTime, 0, Space.World);
            } else if(x > 120) {
                transform.Rotate(0, 30 * Time.deltaTime, 0, Space.World);
            }
        }
    }
}
```

El *controller* lo inicializamos en el método *Start()*. Como el método *Update()* es llamado por cada *frame* del juego, simplemente llamamos a *controller.Frame()* y obtenemos así el *frame* del *Leap* asociado al *frame* del juego. Iteramos por cada una de las manos y nos fijamos la posición de las mismas para determinar si vamos a mover o no la cámara.

Si la posición de la mano en el eje *z* es menor a *-120* entonces, entonces la mano se encuentra delante del Leap y movemos la cámara en la dirección que estaba apuntando. Si en cambio es mayor a *120*, entonces la mano está detrás del *Leap* y desplazamos la cámara en la dirección opuesta de la que estaba apuntando, dando como resultado la sensación de *marcha atrás*. Definimos el número *120* luego de varias pruebas, para darle al jugador la posibilidad de estar quieto en un lugar para poder agarrar los objetos cercanos a él. Es decir, si la mano se encuentra entre *-120* y *120*, la cámara permanecerá quieta. Esto mismo hicimos con el eje *x*, si la mano está posicionada a menos de *-120*, la cámara rota hacia la izquierda y si está ubicada a más de *120*, la misma rota hacia la derecha.

Para lograr esto, además, hicimos que el *HandController*, objeto de *Unity* responsable de hacer aparecer las manos en la escena sea hijo del objeto *Camera*, de esta manera si la cámara se mueve, el *HandController* la acompaña y las manos continúan presentes en la pantalla.

Pasaremos a mostrar ahora *CutTree.cs*, el mismo hace uso de la *API*, para detectar el gesto *swipe*.

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

public class CutTree : MonoBehaviour {

    public int woodQuantity = 25;
    public GameObject woodType;
    private Controller controller;

    // Use this for initialization
    void Start () {
        controller = new Controller ();
        controller.EnableGesture (Gesture.GestureType.TYPE_SWIPE);
    }

    // Update is called once per frame
    void Update () {
        if (woodQuantity <= 0 && transform.rotation.eulerAngles.x < 80) {
            transform.Rotate(Vector3.right, 50 * Time.deltaTime);
        }
    }

    void OnTriggerStay(Collider collider) {
        if (woodQuantity > 0 && collider.tag.Equals("Axe")) {
            Frame frame = controller.Frame ();
            // Get gestures
            GestureList gestures = frame.Gestures ();
            for (int i = 0; i < gestures.Count; i++) {
                Gesture gesture = gestures [i];
                if (gesture.Type == Gesture.GestureType.TYPESWIPE) {
                    Object wood = Object.Instantiate (woodType,
gameObject.transform.position, new Quaternion ());
                    ((GameObject)
wood).GetComponent<Rigidbody>().isKinematic = false;
                    woodQuantity--;
                }
            }
        }
    }
}
```

Los árboles en el juego poseen un componente de *Unity* llamado *Collider*, los mismos son utilizados para detectar colisiones entre objetos. De esta manera, mientras un objeto se mantenga en contacto con el *collider*, el método *OnTriggerStay()* será llamado por cada

frame del juego. Así, una vez detectada la colisión, nos fijamos si el objeto con el que colisionó es el hacha, y en este caso le pedimos a la *API* que nos informe si hubo algún gesto *swipe*, si es así, decrementamos el contador de la cantidad de madera que dispone el árbol. Si la cantidad de madera es menor a 0, entonces en el método *Update()* hacemos rotar el árbol 90 grados, dando la sensación de que el mismo se cae.

Finalmente mostraremos un fragmento del *script UpdateLeaderboard.cs*, ya que el mismo en su totalidad es muy extenso.

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

public class UpdateLeaderboard : MonoBehaviour {

    public string playerName;

    private static int maxNameLength = 10;
    private Controller controller;

    // Use this for initialization
    void Start () {
        controller = new Controller ();
        controller.EnableGesture (Gesture.GestureType.TYPE_CIRCLE);
    }
    // Update is called once per frame
    void Update () {
        ...
        // Get gestures
        Frame frame = controller.Frame ();
        GestureList gestures = frame.Gestures ();
        for (int i = 0; i < gestures.Count; i++) {
            Gesture gesture = gestures [i];
            if (gesture.Type == Gesture.GestureType.TYPE_CIRCLE) {
                Application.LoadLevel(Application.loadedLevel);
                break;
            }
        }
    }
    ...
}
```

Inicializamos el *controller* en el método *Start()* y habilitamos la detección del gesto *circle*. En este momento, la *UI* del juego le indica al usuario que si desea reiniciar el juego

realice un gesto de un círculo con su dedo. Así, en el método *Update()*, por cada *frame* nos fijamos si hubo un gesto de este tipo y en ese caso cargamos nuevamente el mismo nivel.

La *API* fue utilizada también para detectar la mano del jugador al principio del juego en el script *HideIntro.cs*. La *UI* en este caso indica al usuario que coloque su mano sobre el *Leap* una vez que esté listo para comenzar el juego. Otro *script* que hace uso de la *API* es *Creator.cs*, el mismo permitía generar cubos de madera color claro haciendo un gesto *circle* hacia la izquierda, y de color oscuro si el mismo se realizaba hacia la derecha. Decidimos deshabilitarlo ya que hacía perder la gracia del juego.

7. Conclusión

La realización de este trabajo práctico especial fue muy preciado para nosotros porque no solo pudimos realizar un juego funcional y atractivo, sino que para lograr esto tuvimos que aprender a desarrollar para un nuevo dispositivo haciendo uso de su *API*, del motor de videojuegos *Unity 3D* y del lenguaje *C#*, que aunque sean muy populares, nosotros no estábamos muy familiarizados con los mismos, por lo que fue desafiante pero muy valioso utilizarlos. Además de esto, pudimos repasar los conocimientos adquiridos en *HCI*, llevando nuevamente a la práctica los conceptos de usabilidad que uno debe tener muy en cuenta a la hora de hacer un nuevo desarrollo para que el mismo sea entendido y apreciado por los usuarios del mismo, y en este caso mayor cuidado había que tener aún ya que el juego está principalmente dirigido a los niños y adolescentes.

8. Bibliografía

- Integración de *Leap Motion* con *Unity 3D* https://developer.leapmotion.com/getting-started/unity
- Documentación del *SDK* de *Leap Motion* para *C#* https://developer.leapmotion.com/documentation/csharp/index.html
- Documentación de la *Scripting API* de *Unity 3D* http://docs.unity3d.com/ScriptReference/index.html
- Presentaciones de las clases subidas a *sakai*.