Space Fall Game Development

Erick Rivas
School of Engineering
ITESM CSF
México, D.F.
Email: erick.cecyt@gmail.com

Alejandra Espidio
School of Engineering
ITESM CSF
México, D.F.
Email: alejandra.espidio@gmail.com

Jonathan Ginsburg
School of Engineering
ITESM CSF
México, D.F.
Email: jginsburgn@gmail.com

***Abstract*—This document intends to put forward an explanation for the creation of the game called Space Fall. It is a detailed report and is accompanied by images and C# code. The game uses a new technique of path selection for characters in a three-dimensional environment; it can be categorised as both, an RPG and an RTS, among others. The developing game engine was Unity 5.**

Keywords: path selection, ai, unity, rpg, rts, game development.
Palabras Clave: selección de caminos, ia, unity, desarrollo de videojuegos.

I. Introducción

Space Fall es un videojuego para ordenadores de simulación. La primer versión de este juego está para el computador aplicando las teclas ?WASD? para su movilidad. Del género RPG, RTS, Action, Adventure, Fighting, Puzzle, Shooter, Sports y Simulación arcade, el videojuego está desarrollado en una época futurista donde la misión es recolectar información de diversos planetas para que la especie humana prevalezca. El único objetivo del juego es el de entretener a los usuarios de cualquier género y edad, y que disfruten combatiendo obstáculos para ser héroes. Cada misión consiste en una expedición aérea muy similar a un salto en caída libre. El personaje principal es un humanoide, manipulado por el usuario, será el encargado de cumplir las tres diferentes misiones, una misión por cada planeta. Para crear el videojuego se usó el engine Unity versión 5.0 y se tomaron en cuenta diferentes aspectos como: el diseño gráfico de la interfaz, personaje y objetos, y un algoritmo utilizado para su desarrollo.

E.R., A.E., y J.G.
Diciembre 3, 2015

II. Procedimiento

A. Menú

El menú se realizó utilizando objetos de la librería de interfaz de usuario UI de Unity. Algunos de estos objetos fueron botones, páneles y textos. Adicionalmente se siguió la guía referida en bibliografía para la creación de menús. Por otra parte el Asset Collection de Galactic Objects fue utilizado para toda la parte visual; éste se puede descargar de la Asset Store de Unity.

Básicamente se colocaron los objetos aludidos anteriormente en la escena de tal manera que la cámara pudiera observarlos como es convencional para los menúes de videojuegos. Posteriormente, los botones, se conectaron a una función que tiene como argumento un número entero el cual hace referencia al índice de escena de las opciones del videojuego de unity (Build Settings). Dicha función hace el cambio de escena correspondiente al nivel seleccionado o a la pantalla de puntajes altos.

B. Algoritmo de Selección de Caminos

El algoritmo de selección de caminos fue diseñado mediante técnicas de diseño que se separan modularmente. Estos fueron: algoritmos ávidos y backtracking. Algoritmos ávidos se refiere al paradigma de diseño que propone tomar la solución óptima local. Backtracking es el nombre que se le da a las implementaciones recursivas en grafos por profundidad.

De manera breve, el algoritmo establece, en un espacio tridimensional, una esfera con centro en el personaje principal. Dicha esfera es cortada por planos en varias direcciones con vértice de unión en el mismo centro que la misma. Esto produce

un número de intersecciones, las cuales se toman como nodos de un grafo y las aristas se establecen de manera que los nodos no queden desconectados, sin llegar a ser una malla.

Respecto a los algoritmos utilizados en el videojuegos, se utilizará una adaptación del algoritmo Dijkstra con el objetivo de brindar mayor realismo a las trayectorias seguidas por los enemigos. De igual manera, se empleará el Algoritmo de interpolación de Newton para ejecutar aproximaciones que permitan ponderar la dificultad del juego.

La metodología que utilizaremos durante el desarrollo del videojuego será Scrum. Para adaptar esta metodología al videojuegos, diseñaremos cada Sprint para que se adecuen a las etapa del juego, ya sea lógica (Programación y testing), gráfica (Modelado, animación, etc) o integradora. Asimismo, asignaremos los roles del equipo de acuerdo a las características de la iteración , por ejemplo, si estamos en un Sprint que requiera muchas cuestiones gráficas el ScrumMaster será el diseñador (Alejandra) y los programadores formarían parte del equipo de desarrollo.

C. Diseño Gráfico

El personaje principal que se usa en el videojuego, es un humanoide en 3D para crear animaciones cercanas a la realidad de la caída libre; cuando corre desde la nave espacial de Neways, cuando está cayendo por el espacio y cuando esquiva hacia la izquierda y derecha. Con el fin de que el usuario se sienta atraído y a gusto con el diseño del personaje, se escogió un humanoide que fuera amigable y de la mano con la historia del juego. El nombre del humanoide que se obtuvo de la página web Creative Crash es roboter 2.0, animado en el programa de Autodesk, Maya 2015.

Asimismo los demás objetos están creados con el mismo programa, sus colores y formas fueron escogidos así para que tuvieran armonía con el personaje principal de la historia y la historia. La nave espacial está basada en los diseños de las naves futuristas que aparecen en películas como Star Trek o Star Wars.

Para que el personaje obtenga la energía, necesita capturar las latas de aceite que el programa Neways le pone en su camino y poder terminar la misión

Fig. 1. Humanoide

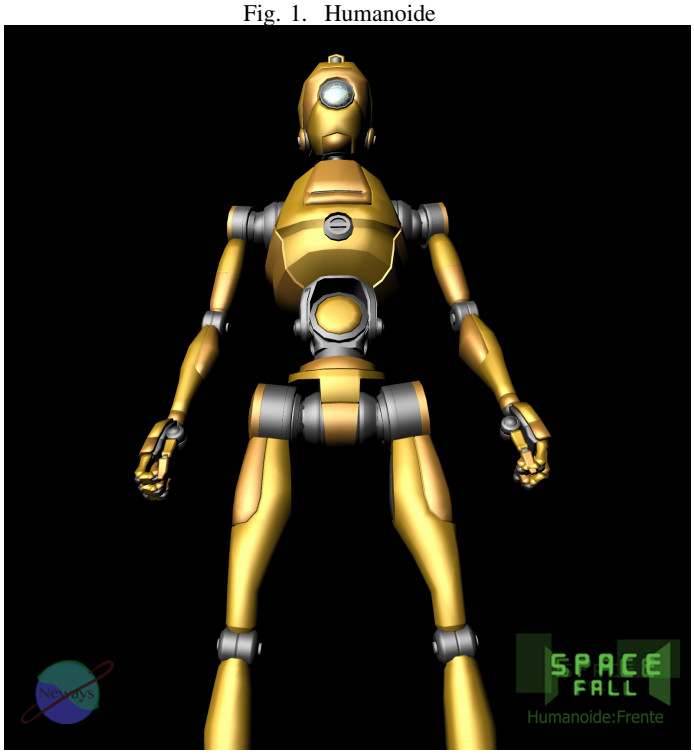
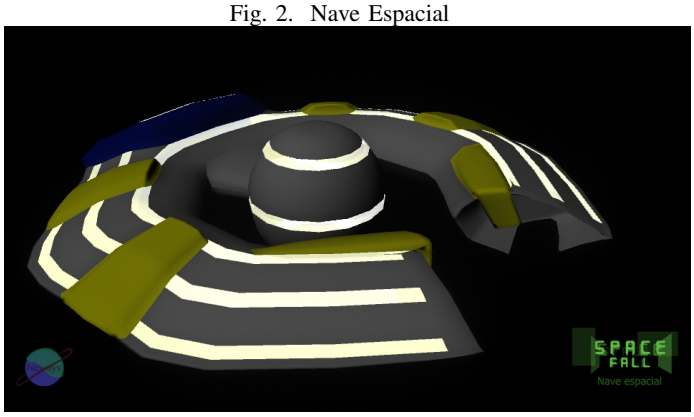


Fig. 2. Nave Espacial



Cuando el humanoide termine la misión aterrizará en una plataforma de Neways

Los logotipos están creados con el programa de Adobe Illustrator. Neways es creado por la NASA para cumplir la misión de que la especie humana prevalezca, por eso su, logo simula un planeta y contiene un aro, haciendo que el logo se parezca al de la NASA.

El logotipo del videojuego fue creado de acuerdo con la idea futurista y espacial. La tipografía que se usa, SPACE AGE, y el color verde hacen énfasis a la idea además de que crea armonía con la interfaz

Fig. 3. Lata de Aceitel

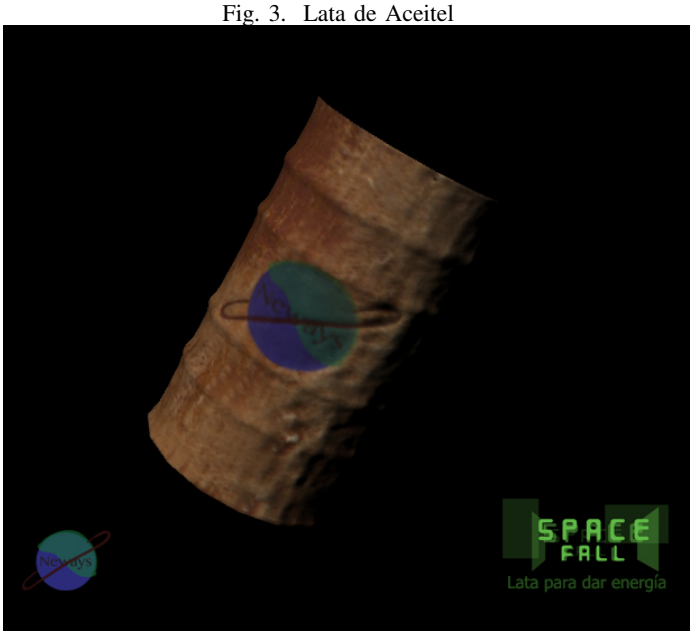


Fig. 4. Plataforma de Aterrizaje

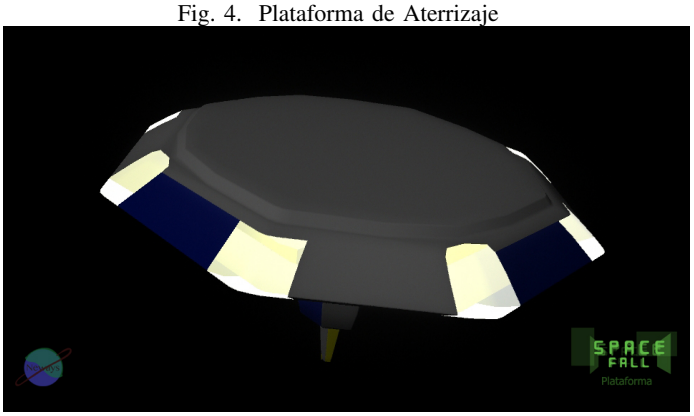


Fig. 5. Logo de Neways

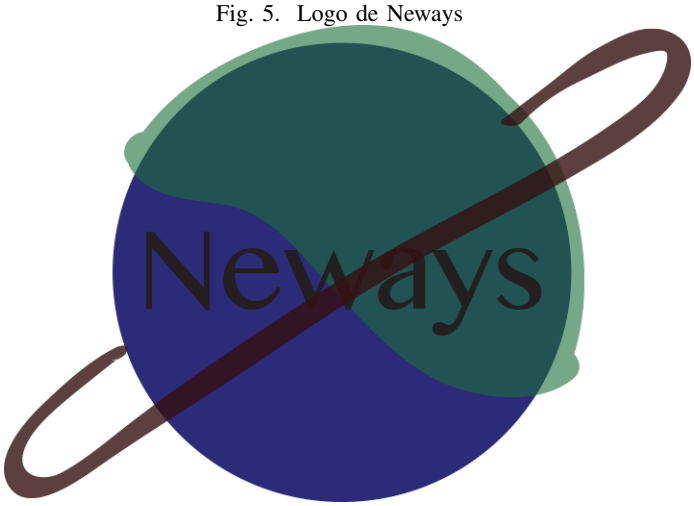


Fig. 6. Logo de Spacefall

Se usa de los assets de Unity el skybox en 2D del espacio y un mundo en 3D del que el humanoide tiene que recolectar la información pero mientras lo hace aparecen asteroides que tiene que esquivar y hay unos aros azules que cambian de color a verde cuando el humanoide entra en ellos para obtener puntos

Para administrar y almacenar la vida del personaje se crea un control. Su diseño está basado en la cara del humanoide, su ojo es donde se almacena la vida poniéndolo de color azul o blanco cuando está vacía, todo para generar armonía en la interfaz

III. Conclusión

La realización del presente videojuego fue altamente multidisciplinaria. Como equipo central, se unieron las habilidades de una aspirante a la Licenciatura de Artes Digitales, Alejandra Espidio, y las de dos aspirantes a la Ingeniería en Tecnologías Computacionales, Erick Rivas y Jonathan Ginsburg. También se utilizaron los conocimientos de terceros a los que se alude en los agradecimientos. Por tanto, Space Fall es un juego de arte e ingeniería en armonía. Tiene implementaciones de algoritmos innovadores y modelos con mucho toque artístico.

Adicionalmente, el algoritmo de selección de caminos en un espacio tridimensional abierto es de autoría propia y vuelve a plantear las posibilidades de hacer dicha tarea. Básicamente consiste en considerar un grafo, del cual los nodos se obtienen por cruces de planos en una esfera con centro en el punto pivote del personaje principal, y las aristas son seleccionadas de manera que dos nodos no queden separados sin necesidad de hacer una maya. Por lo tanto, se tiene un punto apuntando a casi todas las direcciones discretas en las que se puede mover el personaje. Esto genera que los puntos puedan tomar valores correspondientes a lo cerca o lejos que se

Fig. 7. Vida Llena

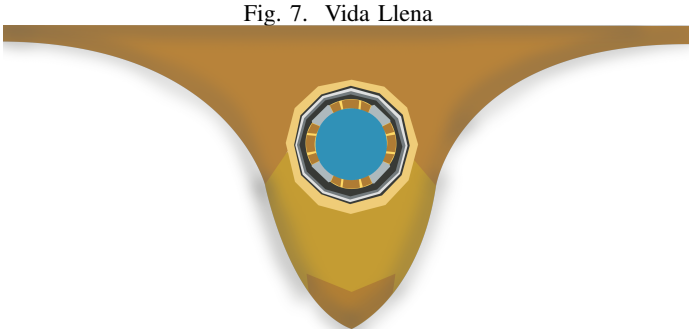
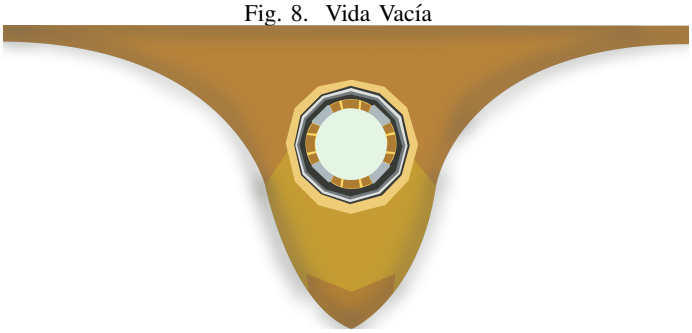


Fig. 8. Vida Vacía



encuentran de objetivos que se quieran alcanzar y que se quieran evadir. De esta manera el personaje tiene una tendencia a ir por los nodos que tengan peso favorable.

Se espera continuar con el desarrollo del algoritmo central de nuestro videojuego para el futuro. Así también, consideramos la posibilidad de publicar el videojuego en alguna plataforma para poder conocer el pensamiento de los jugadores.

IV. Bibliografía

Unity Tutorials. *Creating a Scene Selection Menu.* https://unity3d.com/learn/tutorials/modules/beginner/live-training-archive/creating-a-scene-menu. Julio 24 de 2015. Web.

NASA. Kepler-452b: Earth's Bigger, Older Cousin – Briefing Materials. https://www.nasa.gov/keplerbriefing0723.

Wikipedia. Gleise 667 Cc. https://en.wikipedia.org/wiki/Gliese_667_Cc.

The Guardian. Kepler 438b: Most Earth-like planet ever discovered could be home for alien life. http://www.theguardian.com/science/2015/jan/06/earth-like-planet-alien-life-kepler-438b.

http://www.creativecrash.com/maya/downloads/character-rigs/c/roboter

Agradecimientos

Para la realización del presente videojuego fue crucial contar con la ayuda y el entusiasmo de varias personas. La experiencia con las diversas herramientas que fueron utilizadas por parte de los que ayudaron fue imprescindible. Por otra parte, su amor por la materia correspondiente nos llevó a aprender de ellos. Mencionamos a algunas de estas personas: Samuel Ginsburg, Moises Alencastre y, el estudiante de doctorado, Octavio.

V. Anexos

Cdigo de seleccin de caminos en C#:

```
using System;
using System.Collections;



/**
 *   Handle the algorithms and behavior related to the Artificial Inteligent
 *   of the player
 */

public class GraphPath
{
    private float[] nodeValues;
    private float[, ] nodeEdges;
    private int   LEVELS;
    private int   NUM_NODES;
    private float INF = 100000000;


    public GraphPath(float[] nodeValues, float[, ] nodeEdges, int levels)
    {
        this.nodeValues = nodeValues;
        this.nodeEdges  = nodeEdges;
        NUM_NODES       = nodeValues.Length;
        LEVELS          = levels;
    }

    public ArrayList getRouteBFS()
    {
        ArrayList path = new ArrayList();

        Boolean[] visited = new Boolean[NUM_NODES];
        for (int i = 0; i < NUM_NODES; i++) { visited[i] = false; }

        Queue queue = new Queue();
        queue.Enqueue(0);          //Add origin

        int it = 0;
        while (queue.Count != 0 && it <= LEVELS)
        {
            int parent = (int)queue.Dequeue();
            path.Add(parent);
            visited[parent] = true;


      ArrayList childs = new ArrayList();
      for (int i = 0; i < NUM_NODES; i++) {
          int   childPos   = i;
          float childValue = nodeValues[childPos];
          if (((nodeEdges[parent, childPos] == INF) && (nodeEdges[childPos, parent] == INF)) || //No edges
              visited[childPos]) {                                                             //If the node isn't visited
              childValue = INF;
          }
          childs.Add(childValue);
      }
      ArrayList values = new ArrayList(childs);
      values.Sort();
      float minValue = (float)values[0];
      int   indexMax = childs.IndexOf(minValue);

      if ((indexMax != -1) && (minValue < INF)) {
          queue.Enqueue(indexMax);
      }

      it++;
  }

  return path;
}

private float     minSum = 100000000;
private ArrayList dfsPath;

public ArrayList getRouteDFS()
{
    Boolean[] visited = new Boolean[NUM_NODES];
    for (int i = 0; i < NUM_NODES; i++) { visited[i] = false; }
    executeDFS(0, 0, visited, 0, new ArrayList());
    return dfsPath;
}

private void executeDFS(int parent, int level, Boolean[] lastVisited, float sum, ArrayList lastPath)
{
    if (level <= LEVELS) {
        ArrayList path    = new ArrayList(lastPath);
        Boolean[] visited = new Boolean[NUM_NODES];
        for (int j = 0; j < NUM_NODES; j++) { visited[j] = lastVisited[j]; }
        visited[parent] = true;
        path.Add(parent);


        ArrayList childs = new ArrayList();
        for (int i = 0; i < NUM_NODES; i++) {
            int   childPos   = i;
            float childValue = nodeValues[childPos];
```

```
            ArrayList childs = new ArrayList();
            for (int i = 0; i < NUM_NODES; i++) {
                int   childPos   = i;
                float childValue = nodeValues[childPos];

                if (!((nodeEdges[parent, childPos] == INF) && (nodeEdges[childPos, parent] == INF)) ||
                      visited[childPos])) {
                    executeDFS(childPos, (level + 1), visited, (sum + childValue), path);
                }
            }
        }
        else{
            if (level == LEVELS + 1) {
                if (sum < minSum) {
                    minSum  = sum;
                    dfsPath = lastPath;
                }
            }
        }
    }
}
```