TRABAJO FINAL

MAZEJUMPER

Alumnos:

Brocardo sofia — comision 7

Tolosa nahuel

ÍNDICE

EXPLICACION DEL PROYECTO

¿De qué se trata?

El proyecto consiste en un juego llamado “MazeJumper”, cuyo objetivo es completar diez niveles en el menor tiempo posible, en los cuales el jugador obtendrá un puntaje en base a cuánto tiempo le llevó finalizarlo. Posteriormente, el puntaje será almacenado en un archivo con los registros del resto de los usuarios. De esta forma se obtendrá una persistencia de los datos.

¿Cómo surge la idea?

Entre amos miembros del grupo decidimos casi al instante que realizaríamos un juego como proyecto final, intentando de esta forma aplicar la mayor cantidad de conocimiento en C que tenemos. Fue así como comenzamos a realizar pruebas para verificar si era posible realizar un videojuego. Efectivamente, al nota que podríamos hacerlo comenzamos con el diseño.

¿Por qué el nombre “Mazejumper”?

Es una combinación de las palabras inglesas *“maze”* y *“jump”*, que significan laberinto y saltar respectivamente. La palabra saltar está implementada debido a la sensación de saltos que va dando el jugador mientras recorre el laberinto.

¿Cómo se compone el programa?

Nuestro juego está compuesto de:

- Un sistema de logueo.
- Un sistema de menús de opciones.
- Diez niveles prediseñados.

GUIA DE LIBRERIAS

Internamente, el código del juego está dividido en un total de nueve librerías. A continuación, se hará una breve descripción de cada una:

- **“Admin.c / Admin.h”:** Se encuentran todas las funciones relacionadas a las acciones que puede realizar el usuario administrador. Aquí se encuentra el menú del admin, funciones de baja y alta de ususarios y una función de penalización.

- **“Audio.c / Audio.h”:** Funciones que le dan sonido al juego.

- **“Game.c / Game.h”:** Librería que reúne todas las demás para luego ser incluida en el main de manera más simple y clara. Aquí se encuentran funciones esenciales para el funcionamiento del juego ya que se encuentra el flujo del programa, es decir el llamado a la creación de los niveles, el ciclo que recibe el movimiento del jugador por el mapa y la creación de demás pantallas (como menús).

- **“Levels.c / Levels.h”:** Creación y dibujo de las matrices que se implementaran para mostrar todas las pantallas del juego. Es una librería con una finalidad totalmente relacionada al apartado gráfico del programa.

- **“LogIn.c / LogIn.h”:** Funciones relacionadas a la validación (de correo y contraseña) y creación de usuarios para ingresar al juego como tal.

- **“Menu.c / Menu.h”:** Librería que está conformada con los textos que aparecerán en todos los menús del juego.

- **“Player.c / Player.h”:** Funciones relacionadas a las características (color, skin, posición de inicio) y movimiento del personaje en el juego.

- **“Positions.c / Positions.h”:** Contiene la función que permite el posicionamiento del cursor en cualquier lugar de la consola. Esta función es imprescindible en este proyecto ya que la totalidad del apartado gráfico está posicionado de forma predefinida.

- **“TDAs.c / TDAs.c”:** Funciones relacionada a la implementación de tipos de datos abstractos, ya sean árboles o listas simples.

IMPLEMENTASION DE TEMAS

Para este proyecto se utilizaron tres estructuras trabajadas durante la cursada:

- Listas simplemente vinculadas.
- Árboles binarios.
- Lista de listas simplemente vinculadas.

La lista simplemente vinculada es implementada para almacenar los datos de los diferentes usuarios registrados en el juego.

La estructura es la siguiente:

```
typedef struct _node1{
        char userName[50];
         char nickName[50];
         char email[50];
        char password[50];
         int playerID;
         int condition;
        int active;
        struct _node1* nextUser;
}userNode;
```

Originalmente la estructura compuesta iba a ser un “arreglo de listas simplemente vinculadas”. Sin embargo, conforme se avanzó con el diseño del juego, notamos que la implementación de la misma sería ineficiente en esta oportunidad. Esto se debe a que estaríamos manipulando una cantidad de registros desconocida, dado que pueden registrarse jugadores nuevos constantemente. Es por este motivo que implementamos una “lista de listas simplemente vinculadas”, y de esta manera evitar un manejo erróneo de la información.

Las estructuras son las siguiente:

```
typedef struct _node3{
        int playerID;
        lvlNode* lvlList;
        struct _node3* nextPly;
}playerNode;
```

```
typedef struct _node3{
        int playerID;
        int lvl;
        int score;
        struct _node2* nextlvl;
}lvlNode;
```

Se observa una relación a partir del ID del jugador.

En la propuesta de trabajo se habló de implementar una fila en nuestro proyecto como tercer TDA. Sin embargo, y luego de algunas pruebas, nos comenzó a traer problemas. Es por ello que optamos por la implementación árbol como tercera estructura. De este modo le dimos una utilidad a la hora de mostrar las estadísticas de puntajes globales.

La estructructura es la siguiente:

```
typedef struct _node4{
        char nickName[50];
        int playerID;
        int score;
        struct _node* left;
        struct _node* right;
}treeNode;
```

A continuación, se mostrará la manera en la cual se organizan los datos luego de ser leídos del archivo:

El archivo del cual se obtienen los datos está compuesto por una serie de registros creados a partir de la siguiente estructura:

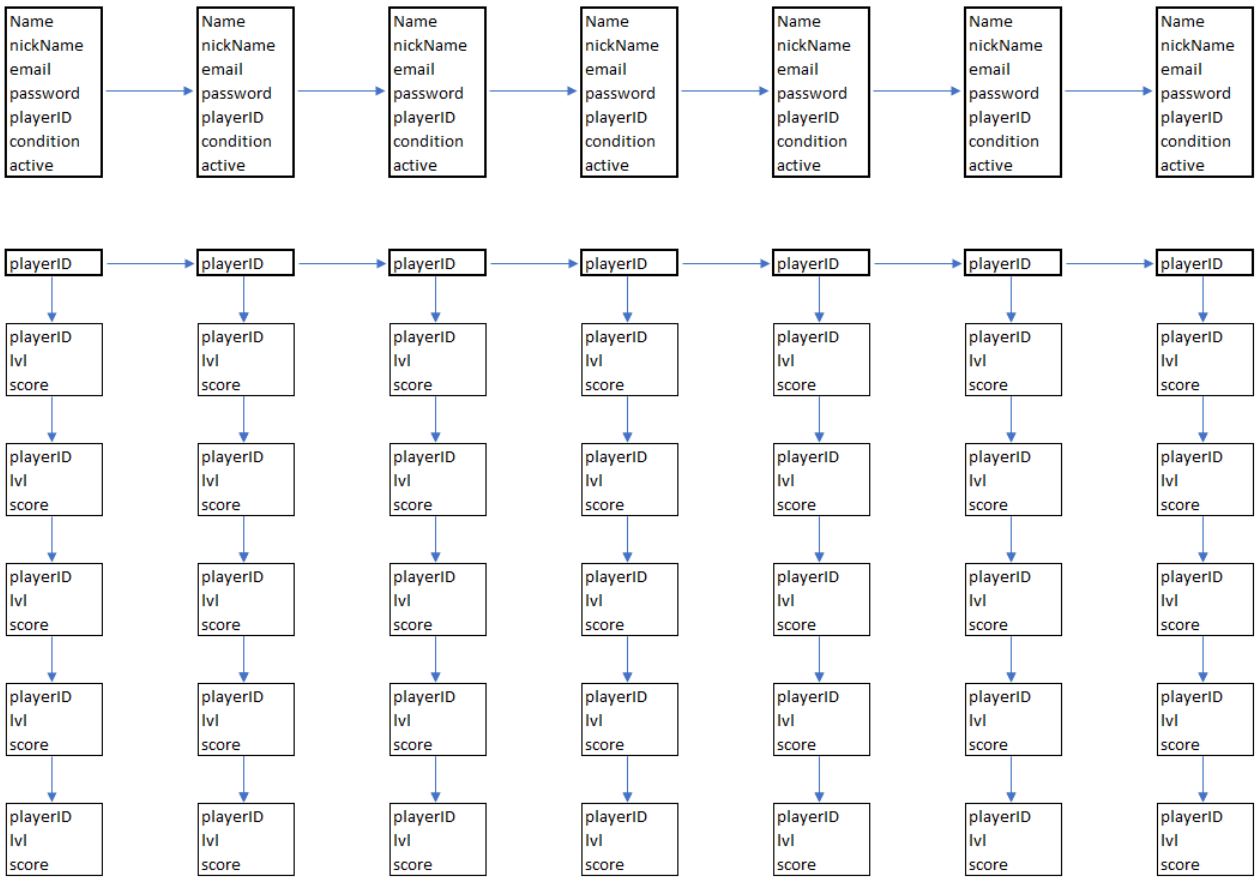
```
typedef struct{
        char userName[50];
        char nickName[50];
        char email[50];
        char password[50];
         int playerID;
        int condition;
        int active;
        int lvl;
        int score;
}stPlayerRecord;
```

Habrá una función en la librería “TDAs” encargada de la lectura de los datos. Su funcionamiento es el siguiente:

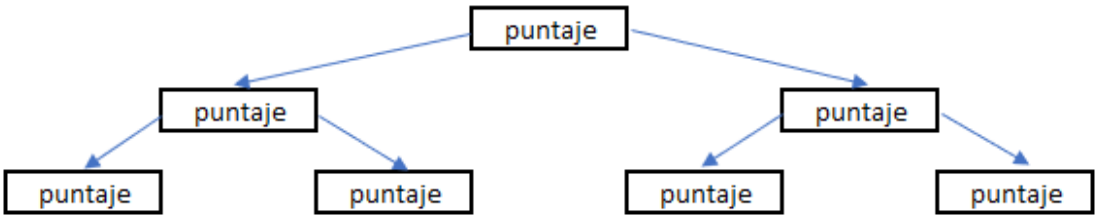
- Lee un registro.
- Comprueba si el usuario ya se encuentra cargado en las listas principales.
- Si ya fue cargado simplemente carga la información de los puntajes.
- Si no fue cargado, crea un nuevo nodo con la información en la lista “userNode” y “playerNode”. Posteriormente carga la información de los puntajes.

Este proceso se repite hasta leer la totalidad de los registros almacenados.

Aquí hay una muestra ilustrativa de cómo quedarían ambas listas:



Por último, el árbol es cargado con la suma de todos los puntajes de cada jugador, quedando así un árbol binario pudiendo estar o no balanceado. Esto depende del orden en el cual se ingresen los datos.



SISTEMA DE LOGUEO

Para ingresar al juego hay que loguearse o bien crear un usuario nuevo. En caso de logiearse, el usuario deberá ingresar el email con el cual se registró y la contraseña. Para que esto sea posible hay una serie de funciones encargadas de validar toda la información para habilitar o negar el ingreso al sistema.

Si el usuario es nuevo, será dirigido a un menú en el cual se le solicitarán los datos que el sistema requiere para la creación de un nuevo jugador. Una vez que los ingresa, hay funciones encargadas de generar "inicializar" al usuario. Podría decirse que se lo agrega a la lista principal, a la lista de listas y se setea en cero su puntaje. De esta forma el usuario podrá ingresar en un futuro dado que los datos se almacenan al salir del programa.

Hay una tercera opción de ingreso: como Admin. El administrador del juego podrá ingresar con un usuario y una contraseña que se encuentran almacenadas en el programa y no en el archivo con los registros de los jugadores. Podría decirse que el programa lo reconoce como un usuario más, con la diferencia de que podrá realizar tareas diferentes. El correo y clave son únicos y no es posible modificarlos. Esto se debe a que el ingreso del admin no es constante como el de un jugador.

A continuación, se dará un usuario de un jugador y el del admin para poder ingresar:

USUARIOS		ADMINISTRADOR	
EMAIL	CLAVE	EMAIL	CLAVE
jnt98@gmail.com	123	user@admin.com	admin123

DISEÑO Y CREACION DE NIVELES

Todos y cada uno de los niveles están diseñados cuidadosamente, dado que es importante tener una relación entre el mapa y la consola para que todo funcione.

¿Cómo funciona?

Se crea una matriz, la cual es cargada con el mapa del menú o nivel que se ejecutará. En el caso de los menús, lo que se hace es completar las celdas de la matriz de forma tal que simule que hay un texto escrito en ella.

En la imagen anterior se aprecia a modo de ejemplo cómo se “dibujó” la introducción del juego. Podemos ver cómo están todas las celdas numeradas. Esto es debido a que se debe conocer la ubicación donde irá el texto y así “pintar” esas celdas.

Esto se hizo con cada pantalla del juego que implica un apartado gráfico.

En el caso de los niveles, esta tarea es mucho más difícil dado a la cantidad de paredes que contienen los laberintos. Conforme avanzan los niveles, el diseño adquiere más complejidad y por ende la codificación del mismo también. Junto a este documento hay otro en *Excel* donde se ve el proceso de diseño de los niveles y menús que utilizamos. Es necesario aclarar que se utilizaron como “borradores”, pero se aprecia el hecho de trabajar con cada celda en específico.

Toda esta información se ve reflejada en la librería “Levels”, que es la más extensa del sistema.

A la hora de mostrar por pantalla los resultados del diseño, ya sea de un menú o un nivel, simplemente se utilizan funciones que se encargan de “printear” el contenido de la matriz.

MOVIMIENTO DEL PERSONAJE

Dentro de la librería “Game” hay una función en particular que contiene la función GetAsyncKeyState(). Esta es la encargada de recibir las órdenes del jugador, es decir la dirección en la que quiere que avance su avatar. El funcionamiento es el siguiente:

- Se crea un bucle que se repetirá hasta que el jugador alcance la salida del nivel.
- Dentro del bucle habrá condicionales con la función anteriormente mencionada.
- Conforme el usuario presione las teclas de movimiento se ejecutará una instrucción que será detallada más adelante.
- A medida que este bucle se ejecuta, la variable del tiempo va disminuyendo, dando la sensación de que el mismo se está agotando.
- Una vez encontrada la salida, el bucle se detiene y se eliminan las instrucciones que no pudieron ejecutarse para evitar entorpecer el programa.

En el tercer punto de la lista anterior se mencionó una instrucción. La misma consiste en chequear si el usuario puede avanzar o no. Esta se encuentra en la librería de “player” y consiste en comprobar si hay una pared del laberinto, es decir si la matriz posee un carácter de “muro”, en el casillero próximo. Si no hay tal muro se elimina el avatar de la posición actual y se crea uno nuevo al lado. Así se consigue la sensación de movimiento del personaje. Esta es la causa de que la matriz esté perfectamente alineada con el extremo superior izquierdo de la consola. De este modo concuerdan los ejes de la consola con los números de las celdas de la matriz.

Nuevamente se hace un llamado a la función “gotoxy()” para posicionar el cursor en la posición donde se desea desplazar el jugador.

MATRIZ DE SOLUCIONES

A lo largo del proyecto nos encontramos con tres grandes problemáticas que requirieron un tiempo considerablemente largo para su resolución.

La primera problemática fue cómo crear el laberinto. Inicialmente se crearía una matriz y el personaje estaría dentro de ella. Se podría decir que se desplazaría dentro de las celdas de la misma. El problema que esto ocasionaba es que por cada movimiento que hacía el personaje, habría que cargar la matriz de nuevo para apreciar los cambios. Este inconveniente sobre cómo poder "dibujar" el mapa y moverse sin tener que cargar todo de nuevo casi nos lleva a abandonar la idea del juego y optar por la propuesta de cátedra. Sin embargo, investigando se descubrió la existencia de una función que permite cambiar el posicionamiento del cursor en la consola. De esta manera nos dimos cuenta que podríamos utilizar la matriz para dibujar el nivel y la función de desplazamiento del cursor para movernos dentro del laberinto. Al comprobar que efectivamente este plan funcionaba presentamos la propuesta de nuestra a la cátedra y fue aprobada.

El resto del desarrollo fue largo, pero no se presentaron mayores complicaciones. Bastaba con revisar las líneas de código donde podrían estar los errores y se solucionaban de forma sencilla. Estamos hablando, por ejemplo, de la lógica de punteros, implementación de TDAs, etc.

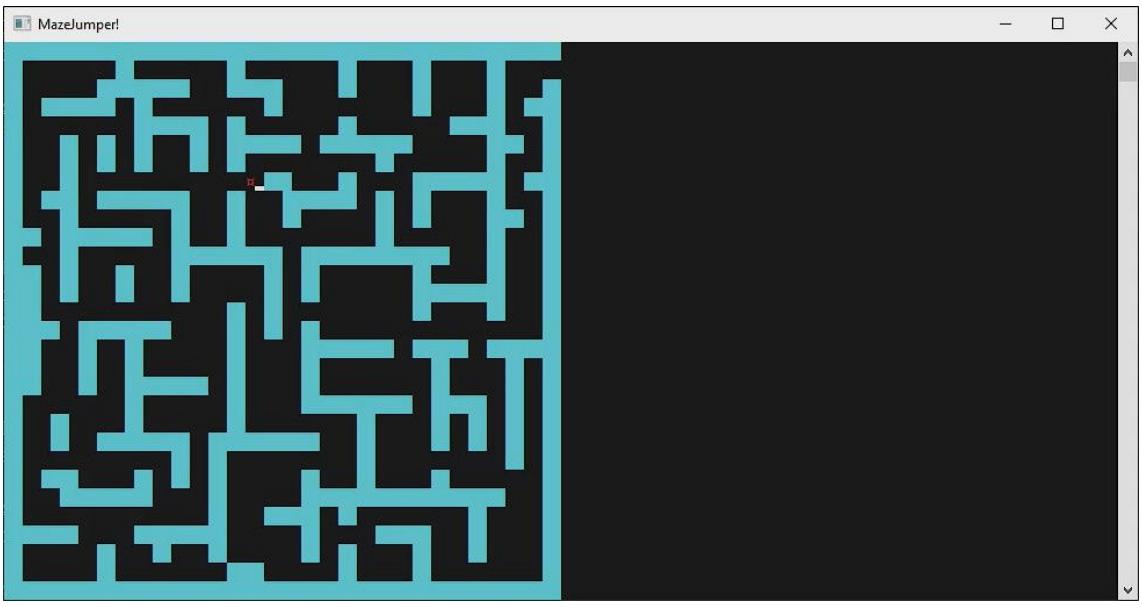
No fue hasta la ultima semana de desarrollo donde se presentó nuestro segundo gran inconveniente. Una vez que logramos integrar todos los menús y el juego en sí, notamos que había problemas con la función de movimiento del personaje. Resulta que al tratarse de un bucle infinito en el cual el jugador presiona teclas para dar instrucciones de movimiento, al finalizar el nivel era tanta la cantidad de teclas que no llegaron a ejecutarse que se almacenaban en un buffer. Entonces, cuando el juego cambiaba de pantalla, estas letras se escribían en la consola. Intentamos con fflush(), pero no funcionaba. Buscamos por muchísimos foros y las soluciones que brindaban los usuarios no tenían un efecto positivo en el juego. Al ver que se nos agotaban las opciones buscamos en foros de habla inglesa alguna persona que haya experimentado un problema similar. Así fue como encontramos la función "FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE))", la cual realiza exactamente lo que necesitábamos. De esta forma se solucionó este problema que nos llevó casi un día de búsqueda e investigación.

Por último, dos días antes de la entrega, cuando llegó el momento de incluir la música, nos encontramos con nuestro tercer gran problema. Resulta que desconocíamos en su totalidad esta funcionalidad, por lo que un miembro del grupo se dedicó a estudiarla mientras el otro continuaba con los últimos detalles de juego. Fue aquí donde intentamos implementar el audio por primera vez y el programa dejó de compilar en su totalidad. Decidimos deshacer esto último, pero por algún motivo el juego dejó de funcionar. Entonces se decidió crear de cero un nuevo proyecto donde se colocaron todas las funciones en el mismo main para luego volver a dividirlas en librerías. El paralelo, la música seguía sin funcionar. Intentamos con la implementación de hilos, creando uno que encendiera la canción y otro que la apagara. A pesar de esto, no funcionaba. Cuando estábamos por rendirnos con el aspecto de la música, un miembro del grupo se dio cuenta que quitando una de las modificaciones que había que hacer para que supuestamente funcione la música, se podría reproducir sin problemas. Fue así como logramos reproducir canciones sin problemas. Todo lo había ocasionado una palabra, literalmente D:

DIARIO DE TRABAJO

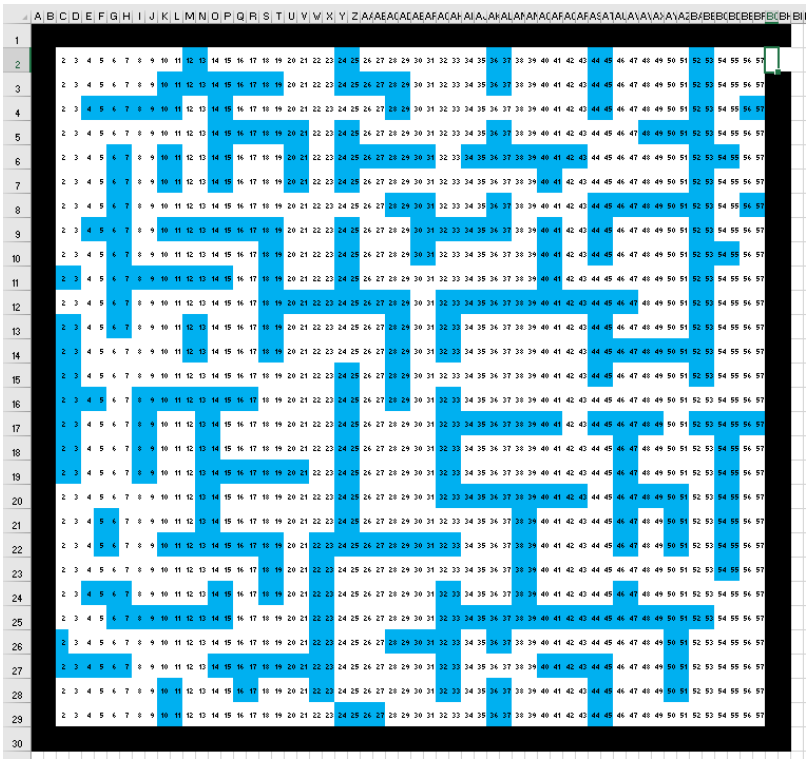
Semana 1 (18 a 24 de octubre):

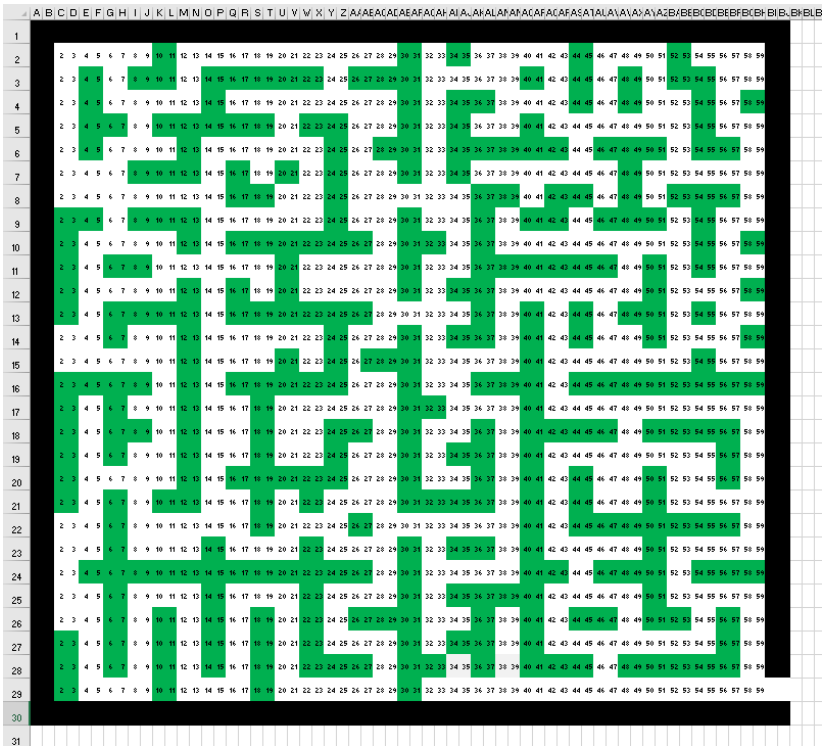
- Inicialmente consideramos optar por la propuesta de cátedra. Sin embargo, se nos ocurrió darle un enfoque relacionado con el entretenimiento. Así fue como decidimos desarrollar juego como proyecto final. El motivo es que a ambos integrantes del grupo nos gustan los videojuegos.
- **Nombre:** El nombre tiene origen en un videojuego llamado “The Lasto f Us”, done había un grupo de personas que se denominaban “Firefly”. No tiene más significado que ese.
- Debido a nuestras limitaciones de tiempo, ya sea por trabajo o por cursar otra carrera en simultaneo, la organización se vio dificultada, así como la asistencia a las clases por Zoom.
- El día **miércoles** se comenzó con el desarrollo de un modelo de juego. El motivo fue evaluar si es posible crear todo lo que se tenía en mente con un lenguaje de programación estructurado. Efectivamente, y luego de horas de investigación, se logró desarrollar un primer modelo del mismo.



- El día **jueves** se confeccionó el PDF con la propuesta donde se detalla el proyecto en cuestión y se especifica en qué puntos se utilizarán los temas trabajados durante la cursada. Rápidamente pudimos solucionar dicha aplicación y darle un sentido en el proyecto.

- El **viernes** fue aprobado el proyecto y se comenzó con el diseño oficial del juego. Primeramente, se reutilizó parte del código creado para generar las imágenes vistas arriba. Comenzó el diseño de *niveles* y *CountDown* en un Excel:





- El día **sábado** se codificó el sistema de puntajes y cronómetro: funcionamiento, posicionamiento. Sistema de logueo. Validación de email y contraseña.

Semana 2 (25 a 31 de octubre):

- El día **lunes** se diseñaron los niveles 2 3 y 4 del juego. Se perfeccionaron algunos aspectos de la pantalla de logueo.
- El día **martes** y **miércoles** no se trabajó en el proyecto.
- El día **jueves** se codeó el nivel 3.
- El día **viernes** se diseñaron los niveles restantes.

Semana 2 (1 a 7 de noviembre):

- El día **lunes** se codearon niveles 4 y 5.
- El día **martes** no se trabajó en el proyecto.
- El día **miércoles** se codearon niveles 6 y 7.
- El día **jueves** no se trabajó en el proyecto.
- El día **viernes** se codeo nivel 8.

Semana 3 (8 a 14 de noviembre):

- Desde el día **lunes** al día **jueves** no se trabajó en el juego debido a que fue una semana con muchos parciales.
- El día **viernes** se codearon los niveles 9 y 10. Con esto finalizó el diseño y codeo de los mismos.

Semana 4 (15 al 21 de noviembre)

- El día **lunes** se trabajó en la finalización de los menús y la implementación de la estructura compuesta **lista de listas**.
- El día **martes** se finalizó con el sistema de logueo y creación de nuevo usuario.
- El día **miércoles** se codearon las funciones de cambio de contraseña, alias del jugador, cambio de color y skin del usuario.
- El día **jueves** se incorporaron los menús y se creó el árbol de estadísticas.
- El día **viernes** se codificó y e incorporó todo el apartado de sonido.
- El día **sábado** se creó el apartado gráfico de todos los menús.

Semana 5 (22 y 23 de noviembre)

- El día **domingo** se hizo todo lo relacionado a las funciones del admin.
- El día **lunes**, último día de trabajo, se hizo una revisión general del funcionamiento del juego. Todo anduvo correctamente.