Proyecto final

Fundamentos de Computación 1

Carpeta de Campo

Documentación

Borges Marcelo Agustin

Vega Hernández Carlos David

**Descripción general del proyecto:**

El proyecto “**Mascota Virtual**” es un proyecto final desarrollado en el contexto de la asignatura de Fundamentos de la Computación 1. El objetivo principal del proyecto es crear una mascota virtual similar a un Tamagotchi, implementada en lenguaje C.

Para lograr este objetivo, se han utilizado diversas herramientas de programación, tales como estructuras, archivos, arrays y funciones. Estas herramientas permiten mantener un código organizado y modular, facilitando su comprensión y mantenimiento.

Además, se ha dado especial énfasis a la implementación de árboles binarios, uno de los temas principales de la asignatura. La integración de árboles binarios en el código del proyecto permite explorar y aplicar los conceptos teóricos aprendidos durante la cursada.



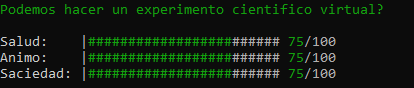
Características del proyecto:

1. Tema y Objetivo principal:

el proyecto “Mascota virtual” se centra en la creación de una mascota virtual interactiva, inspirada en el concepto del popular juego Tamagotchi. El objetivo es simular la responsabilidad de cuidar y mantener a una mascota virtual, proporcionando una experiencia de juego entretenida y educativa.

1. Barras de estado:

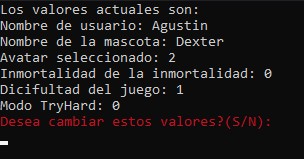
Se deben implementar barras de estado, como la barra de hambre, salud y ánimo de la mascota. Estas barras deben estar relacionadas entre sí, de modo que el nivel de hambre afecte la salud de la mascota, y el ánimo pueda influir en la interacción con minijuegos. El estado de la mascota se reflejará visualmente mediante las barras de estado, lo que permitirá al usuario evaluar y satisfacer las necesidades de la mascota.



1. Datos de usuario y niveles de dificultad:

El proyecto debe permitir a los usuarios ingresar los datos de la mascota, como también, el nombre de usuario.

Estos datos se utilizarán para personalizar la experiencia del juego. Además, se puede optar por niveles de dificultad, que varían la mortalidad de la mascota entre otras características, proporcionando desafíos adecuados a la preferencia y habilidad del usuario.

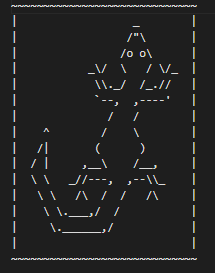
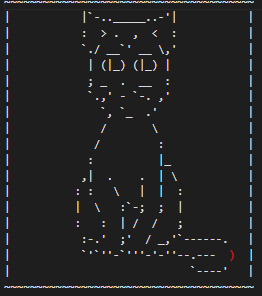


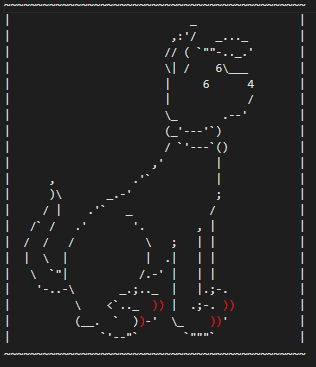
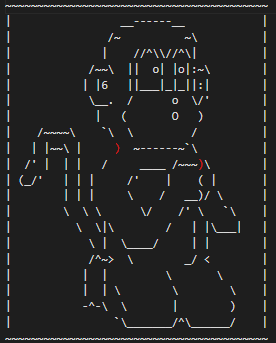
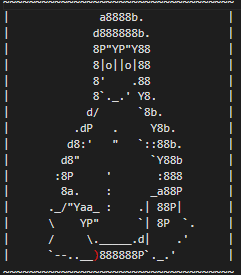


1. Avatares en ASCII:

El proyecto incluye avatares en ASCII para representar visualmente a la mascota virtual.

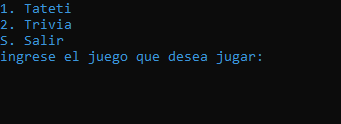
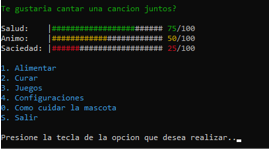
Estos avatares en ASCII agregan un componente visual atractivo y permiten al usuario conectarse emocionalmente con la mascota.



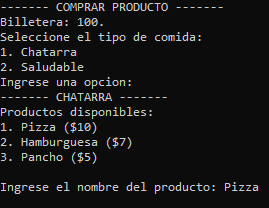


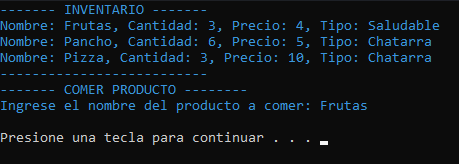
1. Menú interactivo y minijuegos:

Mediante la implementación de un menú interactivo, se proporciona al usuario diferentes opciones para interactuar con la mascota virtual. Estas opciones pueden incluir alimentar a la mascota, jugar con ella, tener un control de su salud, entre otros. Además, se incluyen minijuegos que permiten al usuario interactuar y entretenerse junto a la mascota virtual, afectando su estado de ánimo y brindando recompensas.



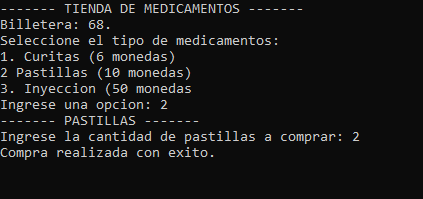
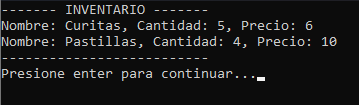
1. Sistema de inventario y tienda de alimentos:

El proyecto incluye un sistema de alimentación variado que permite al usuario seleccionar y proporcionar alimentos a la mascota. Además de una tienda de alimentos donde el usuario puede adquirir diferentes tipos de comida para la mascota. Esto agrega un componente de gestión y toma de decisiones al juego.



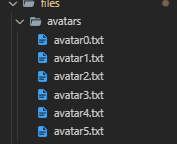
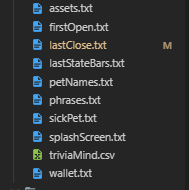
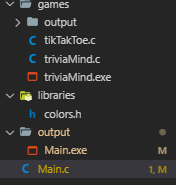
1. Sistema de inventario y tienda de Medicamentos:

El proyecto incluye un sistema de salud variado que permite al usuario seleccionar y proporcionar medicamentos a la mascota. Además de una tienda de medicamentos donde el usuario puede optar entre 3 variados productos, con sus respectivas sanidades.



1. Archivos externos:

Para ampliar la funcionalidad del proyecto, se utilizan archivos externos, asimismo su implementación permite la capacidad de leer y escribir archivos de texto (.txt) y archivos de valores separados por comas (.csv) para almacenar y recuperar datos relevantes del juego, como registros de puntación, datos del usuario, o información de la tienda de alimentos.



**Implementación del tema relacionado a la materia:**

Árboles Binarios: Inserción/Búsqueda/Eliminación.

Sistema de gestión e inventario de alimentación:

En nuestro proyecto de mascota virtual en C, hemos decidido utilizar árboles binarios para gestionar el inventario de alimentos y suministros de nuestra mascota. Esta implementación nos permitirá organizar y acceder eficientemente a los elementos del inventario, además de facilitar la inserción, búsqueda y eliminación de los mismos. A continuación, explicaré los detalles de esta implementación en base a la información proporcionada.

1. Para representar un nodo en nuestro árbol binario, utilizaremos una estructura llamada **‘node’** que contendrá la información relevante de cada artículo del inventario. Los campos de esta estructura incluyen:

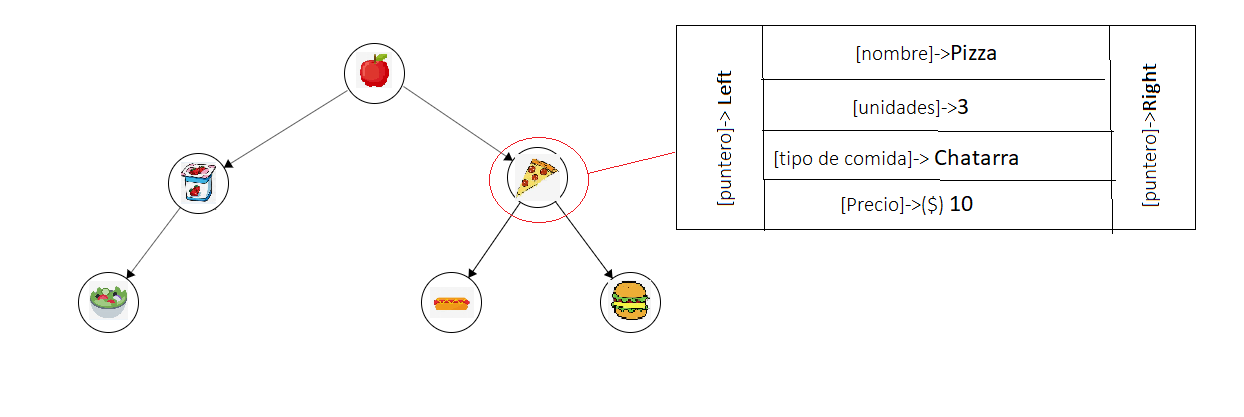
* ‘**itemName’**: El nombre del artículo.
* ‘**quantity’**: La cantidad disponible del artículo.
* ´**price**´: El precio del artículo.
* ‘**foodType’**: el tipo de comida.
* ‘**left’**: puntero a subárbol izquierdo.
* ‘**right’**: puntero a subárbol derecho.

1. La función **‘crearNodo’** se encarga de crear un nuevo nodo del árbol.

Toma valores del artículo y crea un nodo asignando memoria y copiando los valores proporcionados.

1. La función **‘insertarNodo’** se utiliza para insertar un nuevo nodo en el árbol. Toma como argumentos la raíz del árbol, los valores del artículo y realiza la inserción en la posición adecuada del árbol según el nombre del artículo. Si el artículo ya existe en el árbol, se incrementa la cantidad en lugar de insertar un nuevo nodo.
2. La función **‘buscarNodo’** se utiliza para buscar un nodo en el árbol según el nombre del artículo. Toma como argumentos la raíz del árbol y el nombre del artículo, y realiza una búsqueda recursiva en el árbol hasta encontrar el nodo correspondiente o llegar a una hoja.
3. La función **‘mostrarInventario’** se utiliza para mostrar el inventario en orden. Recorre el árbol en orden recursivamente, mostrando los detalles de cada artículo en el nodo actual.
4. La función **‘encontrarMinimo’** es una función auxiliar que encuentra el nodo con el valor mínimo en el subárbol. Se utiliza en la función **‘eliminarNodo’** para encontrar el sucesor inmediato de un nodo cuando se elimina.
5. La función **‘eliminarNodo’** se utiliza para eliminar un nodo del árbol según el nombre del artículo. Realiza una búsqueda recursiva para encontrar el nodo a eliminar y realiza la eliminación según los casos posibles (nodo sin hijos, nodo con un solo hijo o nodo con dos hijos). También actualiza la cantidad de dinero (**‘wallet**) al eliminar un nodo.

En la función ‘**alimentation’** se utiliza un bucle **‘do-while’** para mostrar un menú de opciones al usuario. Las opciones incluyen comprar un producto, mostrar el inventario, comer un producto y salir del programa.

En la opción “Comer producto”, se muestra el inventario actual y se ingresa el nombre del producto a comer. Se busca el nodo correspondiente y, si se encuentra, se actualiza la cantidad y se elimina si la cantidad llega a 0. Además, se actualiza el **‘monedero’** al agregar el precio del producto.

Sistema de gestión e inventario de medicamentos:

En esta sección del código, se utiliza la estructura de datos de arboles binarios para implementar el sistema de inventario y gestión de medicamentos. Dichos árboles se utilizan para almacenar medicamentos disponibles en el inventario de la mascota, permitiendo la búsqueda, inserción y eliminación eficiente de elementos.

1. Para representar un nodo en nuestro árbol binario, utilizaremos una estructura llamada **‘product’** que representará un medicamento en el inventario. Contiene los siguientes campos:

* **‘productName’:** El nombre del artículo.
* **‘quantity’:** La cantidad disponible del artículo.
* **‘price’:** El precio del artículo.
* **‘left’**: puntero a subárbol izquierdo.
* ‘**right’**: puntero a subárbol derecho.

1. La función **‘createNodePr’** se encarga de crear un nuevo nodo del árbol a partir de los parámetros proporcionados. Toma como argumentos el nombre del producto, la cantidad y el precio, y devuelve un puntero al nodo creado.
2. La función **‘InsertNodePr’** permite insertar un nuevo nodo en el árbol de medicamentos.

Recibe la raíz del árbol actual, el nombre del producto, la cantidad y el precio del medicamento a insertar. La función realiza una comparación entre el nombre del producto y el nombre almacenado en el nodo actual, y realiza la inserción en el subárbol correspondiente.

Si el nombre del producto es menor al nombre en el nodo actual, se inserta en el subárbol izquierdo, Si es mayor, se inserta en el subárbol derecho. En caso de que el nombre sea igual, se incrementa la cantidad del nodo actual.

1. La función **‘searchNode’** permite buscar un medicamento en el árbol a partir de su nombre.

Recibe la raíz del árbol actual y el nombre del producto a buscar. Realiza una comparación entre el nombre del producto y el nombre almacenado en el nodo actual.

Si el nombre coincide con el del nodo actual, se devuelve el nodo encontrado. En caso contrario, se realiza la búsqueda en el subárbol correspondiente (izquierdo o derecho) según el resultado de la comparación.

1. La función **‘showInventoryPr’** muestra el inventario de medicamentos en orden ascendente.

Recibe la raíz del árbol y realiza un recorrido **inorden** (izquierdo, nodo, derecho) para imprimir los detalles de cada medicamento almacenado en el árbol.

1. La función **‘deleteNodePr’** permite eliminar un medicamento del inventario. Recibe la raíz del árbol y el nombre del producto a eliminar. La eliminación se realiza siguiendo las reglas de los árboles binarios d búsqueda:

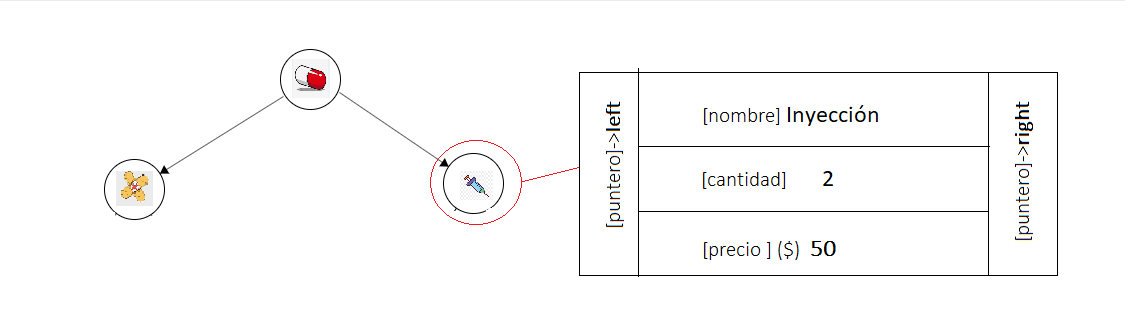
* Si el medicamento tiene una cantidad mayor a 1, se decrementa la cantidad en 1.
* Si el medicamento tiene una cantidad de 1, se busca el nodo con el valor mínimo en el subárbol derecho (o el valor máximo en el subárbol izquierdo), se copian los valores de ese nodo al nodo actual y se elimina el nodo mínimo (o máximo) del subárbol correspondiente.
* Si el medicamento no tiene subárboles, se elimina el nodo directamente.
* Si el medicamento tiene un solo subárbol, se reemplaza el nodo actual por el subárbol.

1. La función **‘consumeMedicine’** permite consumir medicamentos del inventario. Recibe la raíz del árbol, el nombre del medicamento a consumir y la cantidad deseada. La función busca el medicamento en el árbol y realiza las siguientes acciones:

* Si se encuentra el medicamento y la cantidad disponible es suficiente, se decrementa la cantidad en el nodo correspondiente y se elimina el medicamento si la cantidad llega a 0.
* Si el medicamento no se encuentra en el inventario o la cantidad deseada es mayor a la disponible, se devuelve 0 para indicar un consumo inválido.

1. La función **‘healing’** es la función principal del sistema de inventario y gestión de medicamentos. Contiene un bucle que muestra un menú al usuario y realiza las acciones correspondientes según la opción seleccionada.

El menú permite al usuario comprar medicamentos, mostrar el inventario, consumir medicamentos y salir del sistema. Se utiliza la estructura **walletData** para almacenar el saldo de monedas del usuario.



**Arquitectura del proyecto**

Lista de estructuras:

* **AssetsData:** contiene la información PRINCIPAL en tiempo de ejecución.
* **dataStateBars**: almacena los datos de las barras de estado.
* **elpasedTime**: almacena las horas/minutos/segundos que transcurrieron desde la última sesión de juego.
* **walletData**: contiene las monedas del usuario, más adelante se agregarán gemas.
* **node**: nodo del árbol binario relacionado a función alimentación.
* **product**: Estructura para representar los nodos del Árbol binario referido **Healy**.
* **question**: Estructura donde guardar de forma individual, la pregunta actual y su respuesta (0= falso, 1=verdadero).

Lista de funciones utilizadas en el proyecto:

* **firstTime**: chequea si es la primera vez que se abre el programa y pide las configuraciones iniciales.
* **assetsLoad**: esta función carga las configuraciones PRINCIPALES.
* **timeConverter**: convierte los segundos de la variable **timeResult** de la función **lastOpenGetterAndSaver** horas/minutos/segundos y almacena los datos procesados en la estructura **elapsedTime.**
* **stateBarsGetterAndSaver**: carga los estados de la barra anterior y calcula el valor real en base al tiempo transcurrido.
* **alimentation**: Implementa un sistema de gestión e inventario de alimentación.
* **healing**: Implementa un sistema de inventario y gestión de medicamentos.
* **colorBar**: determina el color de las barras dependiendo de su estado, se pasa solo la PROPIEDAD del puntero.

Ej: la propiedad **ptrDataStateBars->health** es un entero.

* **drawBars**: Dibuja las barras completas, lo que quiere decir que dibuja la barra en sí y también su parte restante, como parámetro. Recibe el valor de la barra que se quiere dibujar.
* **showStateBars**: Muestra finalmente las barras.
* **randomPhrases**: Arroja frases aleatorias contenidas en el archivo **"phrases.txt".**
* **splashScreen**: Muestra una pantalla de bienvenida al usuario.
* **showAvatar**: muestra el avatar seleccionado por el usuario.
* **randomPetName**: genera un nombre aleatorio para la mascota.
* -**settings**: permite modificar los ajustes del archivo **assets**, luego de ejecutada la función debe recargar la estructura "**AssetsData**" tiene dos modos, **mode** = 1 permite modificar los valores / **mode** = 0 es para cargar los valores iniciales.
* **walletGetterAndSaver**: guarda o carga la cantidad de monedas del usuario; **mode** = 1 guarda la información en el archivo wallet.txt, **mode** = 0 lee el contenido del archivo.
* **stateBarsDecrement**: determina en cuanto se deben decrementar las barras dependiendo del tiempo transcurrido y de la dificultad seleccionada por el usuario.
* **deathScreen**: muestra una pantalla de muerte cuando la salud es <= 0.
* **tutorial**: explica cómo usar y cuidar a la mascota.
* **gameExecute**: ejecuta los juegos seleccionados.
* **sickPet**: dependiendo del número generado, la mascota se enferma o no, si se enferma, se guarda uno en el archivo **sick.**

Además, la función se encarga de leer el estado desde el archivo **sickPet** (si ya estaba enferma o no) y guardar el nuevo valor en caso de que se enferme cuando se ejecute la enfermedad o que se cure con una vacuna como el valor no es importante en todo momento de la ejecución, no se crea una estructura para empaquetar el dato si no que solo se crea un puntero en el **main**. [**mode** 0 = lee **mode** 1 = guarda.]

* **separateQuestionsAnswers**: con esta función, separo en cada línea, las preguntas de las respuestas de cada línea leída. La función **strlok** divide la cadena en subcadenas utilizando un delimitador específico.
* **generateRandomNumber**: Función que genera un numero aleatorio dentro de un rango determinado entre **min** y **max**.
* **SaveReadInventoryA:** Esta función se encarga de lectura y escritura de datos entre los nodos del árbol de Alimentación y su respectivo archivo.
* **SaveReadInventoryB:** Esta función se encarga de lectura y escritura de datos entre los nodos del árbol de Medicamentos y su respectivo archivo.

Lista de Minijuegos:

* **tikTaktoe.c**: Minijuego de estrategia para 2 jugadores que se juega en un tablero 3x3 donde el objetivo es colocar 3 fichas en forma horizontal, vertical o diagonal.
* **triviaMind.c**: Serie de preguntas y respuestas (10 por sesión de juego), donde se responde con 0(falso) o 1(verdadero).

Lista de archivos:

* **Assets.txt:** Contiene Datos del usuario.
* **firstOpen.txt**: Contiene la fecha exacta (en segundos) en la que se abrió el juego por primera vez.
* **lastClose.txt**: Contiene la fecha exacta (en segundos) en la que se cerró el juego por última vez.
* **lastStateBars.txt**: Contiene el estado de las barras de estado cuando se cerró por última vez el programa.
* **petNames.txt**: Contiene los nombres de las mascotas.
* **phrases.txt**: Contiene lista de frases aleatorias que la mascota va diciendo durante la ejecución.
* **sickPet.txt**: Contiene el estado de enfermedad de la mascota.
* **splashScreen.txt**: Logo de inicio del programa
* **triviaMing.csv**: Contiene los datos de las preguntas y respuestas referidos al minijuego **TriviaMind.c**
* **wallet.txt**: Contiene los datos de las monedas.

Librerías incluida:

* **colors.h:** Crea una serie de constantes que representas colores.

**Seguimiento del proyecto:**

Seguimiento del proyecto en **Notion (**página web, inicio de sesión necesario para ver):

<https://www.notion.so/677b7fbe01a8454b97879e792bdbf1fd?v=650adc9bb58c4ed08e8c3f8b9805103f&pvs=4>

Seguimiento del proyecto en **GitHub** (página web):

<https://github.com/LoKk291/Tamagotchi-C.git>

**Debilidades:**

* El inventario de los árboles binarios, es volátil, es decir que dichos datos, se almacenan únicamente en la memoria RAM, y cuando se cierra el programa, y por consecuente, se pierden.
* En una sección ya identificada del minijuego **‘Triviamind’**, hay un crasheo inesperado, debido al tiempo limitado para el proyecto, no se ha solucionado.

**Bibliografías:**

* <https://es.stackoverflow.com/questions/381853/como-importar-un-include-de-otra-carpeta-en-lenguaje-c>
* <https://programavideojuegos.blogspot.com/2014/02/la-libreria-timeh.html>
* <https://ascii.co.uk/art/>
* <https://www.topster.es/texto-ascii/banner4.html>
* <https://www.youtube.com/watch?v=4hr4VFSfG-Q&ab_channel=DBClassroom>

**Conclusión:**

Mediante el presente proyecto se logra reafirmar conceptos dados en la materia (Fundamentos de la Computación 1) y en anteriores, además de afinar las habilidades de los participantes en los ámbitos de codificación y pensamiento lógico para la resolución de problemas mediante herramientas de programación.

Cabe destacar también, el desempeño en grupo, así como la responsabilidad individual para cumplir con los objetivos establecidos.