.UBAfiuba
FACULTAD DE INGENIERÍA

ALGORITMOS Y PROGRAMACIÓN I - CURSO CAMEJO

Trabajo Práctico Nº 2

Enanitos Valley

Fecha presentación	28/09/2023
Fecha entrega	26/10/2023

Segundo cuatrimestre 2023

1. Introducción

Desde que llegó Blancanieves a vivir con los enanitos, les contagió el amor por cuidar las plantas y pasar tiempo al aire libre. Y como ya no les alcanza el trabajo en la mina y les está costando llegar a fin de mes, decidieron invertir en comprar unos lotes de campo para ponerse a cosechar verduras, para luego venderlas y así poder ahorrar unas monedas.

2. Objetivo

El presente trabajo práctico tiene como objetivo que el alumno:

- Diseñe y desarrolle las funcionalidades de una biblioteca con un contrato preestablecido.
- Se familiarice con y utilice correctamente los tipos de datos estructurados.
- Desarrolle una interfaz gráfica amigable y entendible para el usuario.

Por supuesto, se requiere que el trabajo cumpla con las buenas prácticas de programación profesadas por la cátedra.

Se considerarán críticos la modularización, reutilización y claridad del código.

3. Enunciado

Como desarrolladores de este juego, debemos ayudar a Blancanieves a cultivar su huerta para llegar a su objetivo de ahorro, que es juntar 1000 monedas. Si Blancanieves se queda sin monedas, pierde el juego.

El juego consiste en un terreno donde se tendrán 3 huertas de 3x3 con distintas verduras, donde se deben plantar semillas, luego cosecharlas una vez que crecieron, y llevarlas al depósito para venderlas y así ganar monedas.

Plantar cada tipo de verdura cuesta un monto distinto, así como cada verdura tiene un precio distinto al venderla. Se deberá esperar a que estén disponibles para cosechar para poder hacerlo.

Para inicializar, primero deberán ubicarse las huertas, luego los obstáculos, las herramientas, Blancanieves y la puerta del depósito.

3.1. Huertas

Se tendrán 3 huertas de tamaño 3x3 (9 cultivos). Se podrá sembrar cualquier verdura en sus cultivos. Se deberá posicionar de manera aleatoria en el terreno el cultivo del medio y a partir de ese cultivo, se generarán los de alrededor. Si un cultivo excede límites del terreno, no deberá inicializarse.

3.2. Obstáculos

Los obstáculos deben posicionarse aleatoriamente en el terreno. Cabe destacar que no puede posicionarse un obstáculo en la misma posición que otro objeto.

3.2.1. Plagas

Aparecen aleatoriamente sobre una posición del terreno cada 10 movimientos. Si aparecen arriba de alguna de las 3 huertas, tendrás 10 movimientos para salvarlas, sino se pudrirán y desaparecerán.

3.2.2. Espinas

Se posicionan 5 espinas aleatoriamente en el terreno al iniciar el juego, cada vez que Blancanieves pasa por arriba de una pierde 5 monedas.

3.3. Herramientas

Habrá herramientas para ayudar a Blancanieves con su misión.

3.3.1. Fertilizante

Aparece un fertilizante cada 15 movimientos de forma aleatoria en el terreno. Cabe aclarar que no puede estar ubicado arriba de otro objeto o del personaje. Al activarlo sobre un cultivo de una huerta, hace que crezcan todas las verduras de esa huerta. Se agarran de a uno.

3.3.2. Insecticidas

Al iniciar el juego, Blancanieves tiene 3 insecticidas disponibles para usar. Al activarse sobre un cultivo, curarán toda esa huerta de la plaga.

3.4. Depósito

Se debe inicializar en una posición aleatoria. No puede superponerse con ningún objeto. Blancanieves debe poner ahí la cosecha para que le dé dinero. Una vez que está a distancia Manhattan 2 del depósito, se intercambian las verduras que haya cosechado por monedas.

3.5. Canasta

Para cosechar las verduras, Blancanieves deberá posicionarse sobre un cultivo donde la verdura esté lista para cosechar y ésta se deberá agregar a la canasta de Blancanieves.

La canasta puede llevar como máximo 6 verduras. Al llevarlas al depósito, la canasta se vacía y se podrá volver a cosechar más verduras.

Aclaración: Una vez cosechadas, las verduras no se pudren.

3.6. Plantas

Todas las plantas tienen un precio a la hora de plantarse, y crecen después de una determinada cantidad de movimientos. Si alguna no se cosecha a tiempo desaparece y el cultivo queda disponible para plantar una nueva verdura.

3.6.1. Tomate

Sembrar semillas de tomates cuesta 5 monedas. Cada cosecha de tomate (cada cultivo) te da 30 monedas. Vive durante 30 movimientos, pasados los 20 ya crece y se puede cosechar, pero pasados 10 movimientos después de los primeros 20, se pudre. Es decir, si planto un tomate y camino 20 movimientos, el tomate ya está maduro y lo puedo ir a buscar, a partir de este momento si tardo más de 10 movimientos ya no lo puedo cosechar y desaparece.

3.6.2. Zanahoria

Sembrar semillas de zanahoria cuesta 10 monedas. Cada cosecha de zanahoria (cada cultivo) te da 50 monedas. Vive durante 25 movimientos, pasados los 15 ya crece y se puede cosechar, pero pasados 10 movimientos después de los primeros 15, se pudre.

3.6.3. Brócoli

Sembrar semillas de Brócoli cuesta 15 monedas. Cada cosecha de Brócoli (cada cultivo) sale 70 monedas. Vive durante 20 movimientos, pasados los 10 ya crece y se puede cosechar, pero pasados 15 movimientos después de los primeros 10, se pudre.

3.6.4. Lechuga

Sembrar semillas de Lechuga cuesta 20 monedas. Cada cosecha de Lechuga (cada cultivo) sale 80 monedas. Vive durante 15 movimientos, pasados los 10 ya crece y se puede cosechar, pero pasados 5 movimientos después de los primeros 10, se pudre.

3.7. Terreno

El terreno será representado con una matriz de 20x20, donde estarán dispersos todos los elementos mencionados anteriormente.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0																				
1		B	B	B															C	L
2		C	C	C															C	L
3		T	L	C															C	B
4																				
5			E																	
6																		E		
7									S											
8					F															
9																				
10																				
11																				
12							C	C	Z											
13							L	C	C											
14							C	T	C											
15																		E		
16	E						E													
17	D																			
18																				
19																				

3.8. Modo de juego

El personaje se podrá mover en 4 direcciones:

- **Arriba**: W
- **Abajo**: S
- **Derecha**: D
- **Izquierda**: A

El personaje no podrá salir del terreno, osea que si Blancanieves se encuentra en la primera fila y trata de moverse para arriba no moverá y tampoco contará como movimiento.

Para **sembrar**, Blancanieves debe estar sobre un cultivo de una huerta y se usará la letra **Z** para sembrar zanahoria, **T** para sembrar tomate, **L** para sembrar lechuga, y **B** para sembrar brócoli.

Para **usar el insecticida**, Blancanieves debe estar sobre un cultivo de una huerta atacada por una plaga y se usará la letra **I** para activarlo sobre toda la huerta.

Para **usar el *fertilizante***, Blancanieves debe estar sobre un cultivo de una huerta y se usará la letra **F** para activarlo sobre toda la huerta.

Para cosechar lo sembrado, se podrá hacer posicionándose sobre los cultivos cuando crezcan las plantas y cuando no haya pasado el tiempo límite para hacerlo. Si se posiciona sobre un cultivo que aún no está listo, no debe suceder nada. No se podrán cosechar verduras que tengan plaga.

Para ganar el juego, Blancanieves tiene que ganar un total de 1000 monedas. La cantidad de monedas inicial dependerá del enanito resultante del TP1:

- Gruñón: 150 monedas

- Dormilón: 200 monedas
- Sabio: 250 monedas
- Feliz: 300 monedas

4. Especificaciones

4.1. Convenciones

- **Blancanieves**: S.
- **Cultivo vacío**: C.
- **Depósito**: D.

Se deberá utilizar la siguiente convención para los obstáculos:

- **Espinas**: E.
- **Plagas**: P.

Para las herramientas:

- **Fertilizantes**: F.

Para las verduras:

- **Lechuga**: L.
- **Tomate**: T.
- **Zanahoria**: Z.
- **Brócoli**: B.

4.2. Funciones y procedimientos

Para poder cumplir con los pedidos, se pedirá implementar las siguientes funciones y procedimientos.

```
#ifndef __GRANJA_H__
#define __GRANJA_H__

#include <stdbool.h>

#define MAX_CANASTA 10
#define MAX_OBJETOS 100
#define MAX_HUERTA 3
#define MAX_PLANTAS 9

typedef struct coordenada {
    int fila;
    int columna;
} coordenada_t;

typedef struct cultivo {
    int movimiento_plantado;
    coordenada_t posicion;
    char tipo;
    bool ocupado;
} cultivo_t;

typedef struct huerta {
    int movimientos_plagado;
    bool plagado;
    cultivo_t cultivos[MAX_PLANTAS];
    int tope_cultivos;
} huerta_t;

typedef struct objeto {
    coordenada_t posicion;
    char tipo;
```

```c
} objeto_t;

typedef struct personaje {
    coordenada_t posicion;
    bool tiene_fertilizante;
    int cant_insecticidas;
    int cant_monedas;
    char canasta[MAX_CANASTA];
    int tope_canasta;
} personaje_t;

typedef struct juego {
    int movimientos;
    objeto_t objetos[MAX_OBJETOS];
    int tope_objetos;

    huerta_t huertas[MAX_HUERTA];

    coordenada_t deposito;

    personaje_t jugador;
} juego_t;

/*
 * Inicializará el juego, cargando toda la información inicial de las huertas, los obstáculos, las herramientas y el personaje.
 * El enanito corresponde al enanito obtenido en el TP1.
 */
void inicializar_juego(juego_t* juego, char enanito);

/*
 * Realizará la acción recibida por parámetro. Puede ser un movimiento, con todas sus consecuencias, o sembrar, poner insecticida o fertilizante.
 * La acción recibida deberá ser válida.
 */
void realizar_jugada(juego_t* juego, char accion);

/*
 * Imprime el juego por pantalla
 */
void imprimir_terreno(juego_t juego);

/*
 * El juego se dará por ganado si se llega a 1000 monedas. Si Blancanieves se queda
 * sin monedas, se considerará perdido.
 * Devuelve:
 * --> 1 si es ganado
 * --> -1 si es perdido
 * --> 0 si se sigue jugando
 */
int estado_juego(juego_t juego);

#endif /* __GRANJA_H__ */
```

También se deberá usar la biblioteca implementada en el TP1 para obtener el enanito y así poder definir la cantidad inicial de monedas.

Observación: Queda a criterio del alumno/a hacer o no más funciones y/o procedimientos para resolver los problemas presentados. No se permite agregar funciones al .h presentado por la cátedra, como tampoco modificar las funciones dadas.

5. Resultado esperado

Se espera que se creen las funciones y procedimientos para que el juego se desarrolle con fluidez. Así mismo, se espera que se respeten las buenas prácticas de programación.

Muchas de las funcionalidades quedan a criterio del alumno, solo se pide que se respeten las estructuras y especificaciones brindadas.

El trabajo creado debe:

- Interactuar con el usuario.

- Mostrarle al usuario de forma clara el terreno y todos sus elementos.
- Mostrarle al jugador la información del juego a cada momento.
- Informarle al jugador correctamente cualquier dato que haya sido ingresado incorrectamente.
- Informarle al jugador si ganó o perdió.
- Cumplir con las buenas prácticas de programación.
- Mantener las estructuras propuestas actualizadas a cada momento.

6. Compilación y Entrega

El trabajo práctico debe ser realizado en un archivo llamado granja.c, lo que sería la implementación de la biblioteca granja.h. También se deberá incluir en la entrega la biblioteca del TP1. El trabajo debe ser compilado sin errores al correr el siguiente comando:

```
gcc *.c -o juego -std=c99 -Wall -Wconversion -Werror -lm
```

Aclaración: El main tiene que estar desarrollado en un archivo llamado juego.c, el cual manejará todo el flujo del programa.

Lo que nos permite ***.c** es agarrar todos los archivos que tengan la extensión .c en el directorio actual y los compile todos juntos. Esto permite que se puedan crear bibliotecas a criterio del alumno, aparte de las exigidas por la cátedra.

Por último debe ser entregado en la plataforma de corrección de trabajos prácticos **AlgoTrón** (patente pendiente), en la cual deberá tener la etiqueta **¡Exito!** significando que ha pasado las pruebas a las que la cátedra someterá al trabajo.

IMPORTANTE! Esto no implica necesariamente haber aprobado el trabajo ya que además será corregido por un colaborador que verificará que se cumplan las buenas prácticas de programación.

Para la entrega en **AlgoTrón** (patente pendiente), recuerde que deberá subir un archivo **zip** conteniendo únicamente los archivos antes mencionados, sin carpetas internas ni otros archivos. De lo contrario, la entrega no será validada por la plataforma.

7. Anexos

7.1. Obtención de números aleatorios

Para obtener números aleatorios debe utilizarse la función **rand()**, la cual está disponible en la biblioteca stdlib.h.

Esta función devuelve números pseudo-aleatorios, esto quiere decir que, cuando uno ejecuta nuevamente el programa, los números, aunque aleatorios, son los mismos.

Para resolver este problema debe inicializarse una semilla, cuya función es determinar desde dónde empezarán a calcularse los números aleatorios.

Los números arrojados por **rand()** son enteros sin signo, generalmente queremos que estén acotados a un rango (queremos números aleatorios entre tal y tal). Para esto, podemos obtener el resto de la división de **rand()** por el valor máximo del rango que necesitamos.

Aquí dejamos un breve ejemplo de como obtener números aleatorios entre 10 y 30.

```c
#include <stdio.h>
#include <stdlib.h> // Para usar rand
#include <time.h>   // Para obtener una semilla desde el reloj

int main(){
        srand ((unsigned)time(NULL));
        int numero = rand() % 20 + 10; // la amplitud del rango es 20 y el valor mínimo es 10.
        printf("El valor aleatorio es: %i\n", numero);

        return 0;
}
```

7.2. Limpiar la pantalla durante la ejecución de un programa

Muchas veces nos gustaría que nuestro programa pueda verse siempre en la pantalla sin ver texto anterior.

Para esto, podemos utilizar la llamada al sistema **clear**, de esta manera, limpiaremos todo lo que hay en nuestra terminal hasta el momento y podremos dibujar la información actualizada.

Y se utiliza de la siguiente manera:

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
        printf("Escribimos algo\n");
        printf("que debería\n");
        printf("desaparecer...\n");

        system("clear"); // Limpiamos la pantalla

        printf("Solo deberiamos ver esto...\n");
        return 0;
}
```

7.3. Distancia Manhattan

Para obtener la distancia entre 2 puntos mediante este método, se debe conocer a priori las coordenadas de dichos puntos.

Luego, la distancia entre ellos es la suma de los valores absolutos de las diferencias de las coordenadas. Se ve claramente en los siguientes ejemplos:

- La distancia entre los puntos (0,0) y (1,1) es 2 ya que: $|0 - 1| + |0 - 1| = 1 + 1 = 2$
- La distancia entre los puntos (10,5) y (2,12) es 15 ya que: $|10 - 2| + |5 - 12| = 8 + 7 = 15$
- La distancia entre los puntos (7,8) y (9,8) es 2 ya que: $|7 - 9| + |8 - 8| = 2 + 0 = 2$