Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Informática Aplicada

Trabajo Práctico N° 2

Grupo 1

AÑO: 2022

Nombre y Apellido	Nº de Legajo
Leonel Lingotti	L-3147/1
Luciano Raffagnini	R-4419/9
Sebastián Ignacio Rodríguez	R-4422/9

Modelización del Problema

Para realizar la MEF se utilizó un enfoque análogo al del TP1. Cada estado codifica qué jugador está jugando en ese momento o si el cronómetro está en reposo. Los valores de las variables Botón y Fin generan las transiciones entre estados.

Identificamos 8 estados distintos que asociamos con 8 situaciones distintas:

- **Reposo:** El cronómetro ha sido reseteado.
- **Turno J1 - Botón Pulsado:** Es el turno del Jugador 1, aún mantiene pulsado el botón.
- **Turno J1:** Es el turno del Jugador 1.
- **Turno J2 - Botón Pulsado:** Es el turno del Jugador 2, aún mantiene pulsado el botón.
- **Turno J2:** Es el turno del Jugador 2.
- **Termina Tiempo J1:** El Jugador 1 se ha quedado sin tiempo.
- **Termina Tiempo J2:** El Jugador 2 se ha quedado sin tiempo.

De está forma podemos, a partir de la lista de eventos generados en forma aleatoria, inferir de forma inequívoca el comportamiento que debería tener cronómetro.

A su vez, está modelización, nos permite descartar fácilmente los eventos aleatorios no válidos, ya que cada estado tiene un conjunto finito y bien definido de eventos que pueden provocar cambios de estado.

Desarrollo del programa

Utilizamos un programa en lenguaje C++ para la solución del problema que cuenta con:

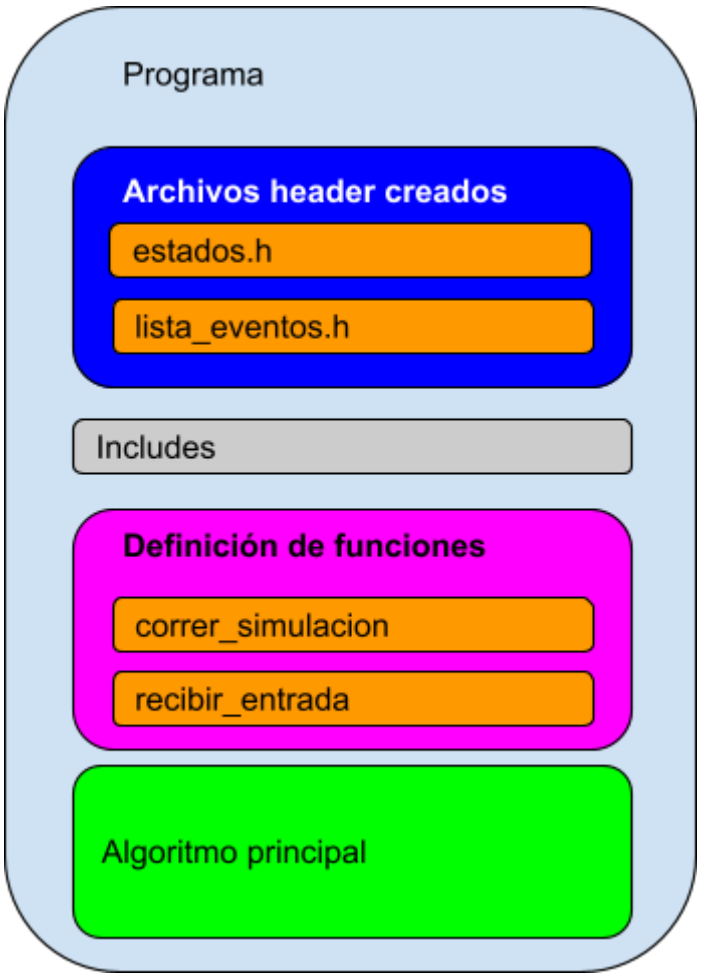


Figura 1: Estructura funcional del programa.

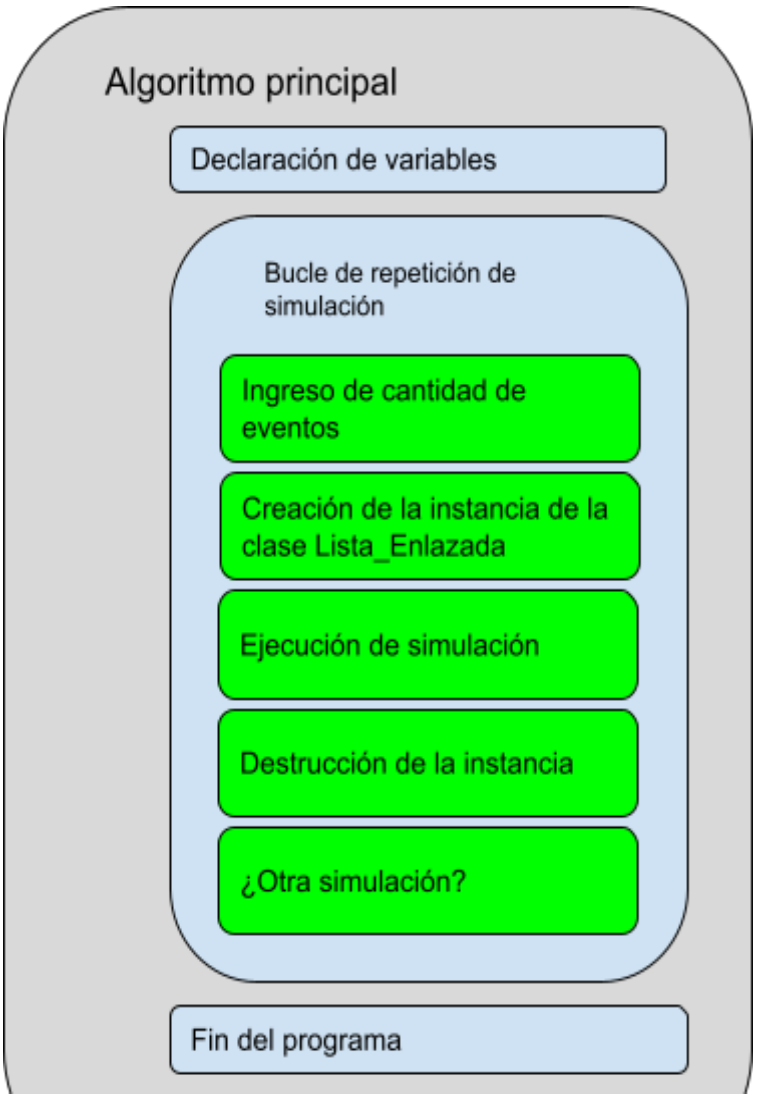
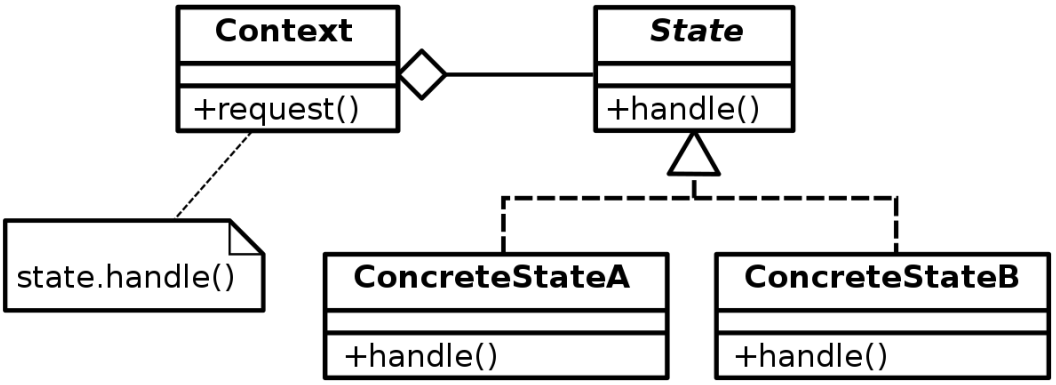


Figura 2: Estructura funcional del algoritmo principal (main).

Patrón State

Para la implementación de la MEF utilizamos un patrón utilizado en programación orientada a objetos conocido como “State”.



Consiste en definir una interfaz *State* que contenga un método *handle*, que procese el estado actual. Luego definiremos una clase para cada estado de la MEF, que implementará la interfaz State, con la correspondiente implementación del método *handle*. De está forma, cada clase se encarga de realizar las tareas pertinentes a cada estado.

En el caso específico de nuestro programa, cada clase se encarga de imprimir por pantalla los datos correspondientes al estado actual, dependiendo de las transiciones que ocurran, la clase de Estado instanciada será distinta.

Encapsulamiento

Decidimos ocultar los detalles de implementación de la lista enlazada detrás de una clase *Lista_Enlazada*. Esta clase recibe en su constructor, el largo de la lista y se encarga de crearla dinámicamente, generando los eventos aleatorios correspondientes. A su vez, al llamar a su destructor, se libera la memoria reservada.
De está forma, expone solamente el contenido de los eventos generados, que serán utilizados para realizar la simulación.

Por otro lado, tuvimos que tomar una medida adicional para que la clase *Lista_Enlazada* no filtre los detalles de implementación. Esta clase contiene un método llamado *get_nodo*() que se encarga de retornar un nodo de la lista y avanzar una posición dentro de la misma. Si retornamos un nodo de tipo *Evento*, como se utiliza en la lista enlazada, un programador podría acceder a través de la variable *sig* del nodo, al siguiente elemento de la lista, y por lo tanto a toda la lista. Para evitar esto, la clase *Evento* implementa la interfaz *Mediciones*, que solo expone 2 métodos, *get_boton*() y *get_fin*(), que retornan los valores generados aleatoriamente para ese nodo. De esta forma, *get_nodo*() retorna un objeto de tipo *Mediciones*, haciendo un “cast” del nodo de tipo *Evento*, evitando así dar acceso a los otros elementos de la lista.

Sobre el funcionamiento del programa

Al iniciar el programa se recibe un mensaje de bienvenida y se pregunta si se quiere observar la lista de eventos aleatorios que se crean. Esto lo agregamos considerando el caso que se quiera comprobar de forma visual su correcto funcionamiento.

```
       Bienvenido a la simulacion de ajedrez

  Desea ver las listas de eventos aleatorios que se generen?
  Aplica a todas las simulaciones

Ingrese su respuesta (S/N):
```

Nota: Esta elección afecta toda la ejecución del programa, es decir, al terminar la simulación no se preguntará nuevamente si se desea ver la lista.

Posteriormente solicita la cantidad de eventos aleatorios a crearse. Se debe enviar como un número + tecla enter.

En caso de haber elegido S, luego de ingresar la cantidad de eventos el programa imprimirá la lista de eventos y luego la simulación, sino sólo mostrará la simulación.

```
       Bienvenido a la simulacion de ajedrez

  Desea ver las listas de eventos aleatorios que se generen?
  Aplica a todas las simulaciones

Ingrese su respuesta (S/N):
n
Ingrese la cantidad de eventos aleatorios que desea crear:
10
Cronometro en reposo:                    ENA1 = 1 ENA2 = 1
Tiempo 1 corriendo con boton apretado:   ENA1 = 1 ENA2 = 0
Cronometro en reposo:                    ENA1 = 1 ENA2 = 1
Tiempo 1 corriendo con boton apretado:   ENA1 = 1 ENA2 = 0
Cronometro en reposo:                    ENA1 = 1 ENA2 = 1

Fin de la simulacion

Desea realizar otra simulacion?
Ingrese su respuesta (S/N):
```

```
        Bienvenido a la simulacion de ajedrez

  Desea ver las listas de eventos aleatorios que se generen?
  Aplica a todas las simulaciones

Ingrese su respuesta (S/N):
s
Ingrese la cantidad de eventos aleatorios que desea crear:
10
Boton:1 Fin:0
Boton:0 Fin:1
Boton:0 Fin:1
Boton:1 Fin:0
Boton:1 Fin:1
Boton:1 Fin:0
Boton:1 Fin:0
Boton:0 Fin:0
Boton:0 Fin:0
Boton:1 Fin:0
Cronometro en reposo:                    ENA1 = 1 ENA2 = 1
Tiempo 1 corriendo con boton apretado:   ENA1 = 1 ENA2 = 0
Fin del tiempo 1
Cronometro en reposo:                    ENA1 = 1 ENA2 = 1
Tiempo 1 corriendo con boton apretado:   ENA1 = 1 ENA2 = 0
Tiempo 1 corriendo sin apretar boton:    ENA1 = 1 ENA2 = 0

Fin de la simulacion

Desea realizar otra simulacion?
Ingrese su respuesta (S/N):
```

Nota: La simulación finaliza abruptamente cuando se terminan los eventos aleatorios.

Con respecto al uso de funciones recursivas

Los métodos que utilizamos para crear y destruir la lista de eventos son de tipo recursivo. Esta metodología tiene la desventaja de que si intentamos crear una lista demasiado larga podemos desbordar el stack y que el programa deje de funcionar. Sin embargo consideramos que trabajaremos con listas de un largo razonable de forma que nunca se exceda este límite.

Realizamos una prueba en una de nuestras PCs para determinar el largo máximo soportado y encontramos que es de **32559** elementos, por lo menos para esa máquina en particular.