Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información

CI2691 – Laboratorio de Algoritmos y Estructuras I

Enero – Marzo 2024

**N en Raya – 3D**

Profesor: Estudiantes:

Amin Arriaga David Díaz (20-10019)

Alan Argotte (19-10664)

**Resumen**

El siguiente trabajo pretende describir el análisis descendente realizado para la implementación del juego *N en Raya 3D* escrito en Python 3.11.3 y haciendo uso del binding de la biblioteca gráfica Tk en Python, Tkinter. Esta descripción incluye la mención de los subproblemas que surgen naturalmente del enfoque aplicado para determinar una solución práctica y simple a susodichos problemas y el análisis de los módulos y clases involucrados en la implementación de estas soluciones. También uno planea resaltar las consideraciones operativas del programa acerca de su funcionamento y los errores encontrados, los resultados obtenidos y las opiniones de sus creadores sobre la experiencia del desarrollo del proyecto y recomendaciones para los futuros programadores que planeen alterar de una u otra forma el juego.

**Diseño Descendente de *N en Raya 3D*:**

El **bloque de código 1** adjunto a continuación describe de forma sucinta en CGS el flujo del programa definitivo del juego:

Bloque de Código 1: Esquema general de solución del juego

* : inicializa los elementos gráficos de la Graphical User Interface (GUI) correspondientes al menú principal de un juego (botón iniciar, botón salir) y sus respectivas funciones lógicas.
* : inicializa los elementos gráficos de la GUI correspondientes al menú prejuego. Es decir, define los nombres de los jugadores, la dimensión de los tableros de juego y muestra los botones iniciar y regresar al menú principal.
* : inicializa los elementos gráficos de la GUI correspondientes a los tableros de juego, el puntaje de cada jugador, el marcador, las fichas de los jugadores dado un movimiento y despliega los botones continuar juego y regresar al menú principal.

A continuación, en *bloque de código 2* y *bloque de código 3* definimos el diseño descendente de las funciones e , respectivamente:

* : determina la posición y aspecto de los botones con funciones lógicas del programa y el fondo de pantalla de la GUI.

Bloque de Código 2: Diseño descendente de iniciar\_menu\_principal()

* activa la función lógica del botón jugar en la GUI. Permite que el usuario entre al menú prejuego y siga con el flujo del programa
* : activa la función lógica del botón salir en la GUI. Sale del constructor iteración del flujo del programa y lo termina cambiando el estado de la variable global .

Bloque de Código 3: Diseño descendente de iniciar\_menu\_prejuego()

* : determina la posición y aspecto de los botones *regresar* e *iniciar*, el fondo del menú prejuego y los *entries* para el nombre del jugador 1, jugador 2 y la dimensión del tablero de juego.
* : define la función lógica del botón regresar. Actualiza el aspecto de la GUI y la cambia al menú principal.
* : define la función lógica del botón iniciar. Inicializa el juego si y solo si se introducen nombres válidos para jugador 1, jugador 2 y una dimensión adecuada para inicializar los tableros.
* : solicita al usuario del programa un nombre válido para el jugador 1.
* : solicita al usario del programa un nombre válido para el jugador 2.
* solicita al usuario del programa un entero positivo mayor o igual que 3 para inicializar los tableros del juego.

La aparente simplicidad de este análisis descedente es severamente opacado por las necesidades operativas sucitadas durante la implementación del juego. Entre las problemáticas ocurridas, surgían las siguientes preguntas: ¿cómo uno puede alcanzar un diseño modular simple? ¿cómo podría el juego inicializarse y procesar la lógica detrás de *N en raya*? El restante de este punto en este informe se dedicará a responder estas preguntas.

Debido a la clara distinción de características intrínsecas de los elementos del juego (uno o más tableros y sus correspondientes casillas) y a la íntima relación entre éstos, se crearon las clases `Tablero` y `Casilla` con sus respectivos atributos, métodos y funciones auxiliares para reflejar el estado de una partida del juego y sus elementos gráficos. Motivado a que el enfoque de este informe es dedicado a la descripción detallada de las estructuras de datos utilizadas, en el apartado ‘Estructuras de Datos’ puede uno apreciar la implementación abstracta de las clases `Tablero` y `Casilla`. Sin embargo, enfocándonos exclusivamente en la experiencia del usuario, logramos definir el siguiente flujo de programa para el momento que el juego es inicializado. Aunque se ha de hacer especial énfasis que este flujo es solo posible a partir de las llamadas a los métodos entre las clases `Tablero` y `Casilla` que se modifican entre sí:

Bloque de Código 3: Esquema general del flujo de programa de iniciar\_juego()

* dada la dimensión ingresada por el usuario del programa, define objetos de tipo `Tablero` y los introduce en una variable global que es una lista de tipo `Tablero`.
* : determina la posición y aspecto de un display de mensajes en la parte inferior de una cuadrícula que muestra los tableros de juego. Indica cuando un jugador haya ganado el tablero o se haya llegado a un empate.
* : determina la posición y aspecto de un display que muestra el puntaje alcanzado por el jugador 1 y el jugador 2 junto a la ficha que cada uno utiliza. Por defecto, al jugador 1 le corresponden las cruces; al jugador 2, los círculos.
* : determina la posición, aspecto y función lógica del botón regresar durante la partida de juego. Elimina los elementos gráficos inicializados en el juego y el estado de la partida para regresar al menú principal. Hace esto cambiando el estado de una variagle global llamada `regresar`.
* : cambia el estado lógico y gráfico de la casilla seleccionada por el jugador 1 o el jugador 2 en uno de los tableros.
* cambia el estado de la variable global llamada `turno` la cual determina si el siguiente cambio en el estado gráfico de la casilla seleccionada por el jugador corresponde al jugador 1 o al jugador 2. Es decir, decide si en la casilla debe ser dibujada una cruz o un círculo. Asimismo, al final de cada partida, se intercambia el turno del jugador que comenzó. Es decir, si inicia la partida el jugador 1, la siguiente la inicia el jugador 2.
* : por cada cambio en el estado de una casilla realizado por un jugador, el tablero es procesado para identificar un patrón de victoria como viene definido en las especificaciones del programa o si se empató la partida. En caso de que uno de los jugadores haya creado un patrón de victoria o un patrón de empate, cambia el estado de una variable global llamada `partida\_ganada` o no.
* : de haber identificado un patrón ganador en los tableros, identifica las casillas pertenecientes a ese patrón ganador.
* : cambia el estado del contador para reflejar los cambios en la puntuación entre los jugadores.
* : crea un mensaje mostrado en el display de mensajes indicando el estado de la partida tras haber alcanzado un patrón de victoria o de empate.
* determina la posición, aspecto y función lógica de un botón en el momento que una partida se gana o se empata. Cambia el estado de una variable global `continuar\_partida`.
* reinicia el estado lógico y gráfico de las casillas en los tableros

**Estructuras de Datos**

**Diseño Modular**

La cuestión del diseño modular se presentó como una solución por sí misma debido a las características de Tkinter: la ventana principal no se actualizaba adecuadamente con el conjunto de elementos gráficos correspondientes a cada menú y, consecuentemente, no podía contenerse en un mismo módulo su implementación. Por ende, se creó un módulo único que contendría a la ventana principal del juego, generando así las dependencias entre módulos presentes en la ilustración 1:

Ilustración 1: Depedencia intermodular de la implementación de N en raya 3D

* : contiene las características gráficas constantes de la ventana principal de la GUI del juego.
* : contiene las funciones lógicas y elementos gráficos del menú principal del juego.
* : contiene las funciones lógicas y elementos gráficos del menú prejuego.
* : contiene las funciones lógicas y elementos gráficos de *N en Raya 3D*.
* : contiene el flujo del programa del juego.

Como se mencionó en el apartado del diseño descendente del juego, en la implementación del juego, las siguientes clases esquematizadas y sus métodos que no solo alteran los atributos de sus propias instancias, sino una a la otra, controlan el flujo del programa mediante sus inicializaciones y cambios efectuados por el usuario del programa. Para ilustrar este punto, considere que no es una función la que inicializa el tablero y las casillas como elementos distintos e independientes entre sí, sino que solamente inicializa varias instancias de la clase `Tablero` que, a su vez, inicializan instancias de `Casilla` que son también atributos de las instanciaciones de `Tablero`.

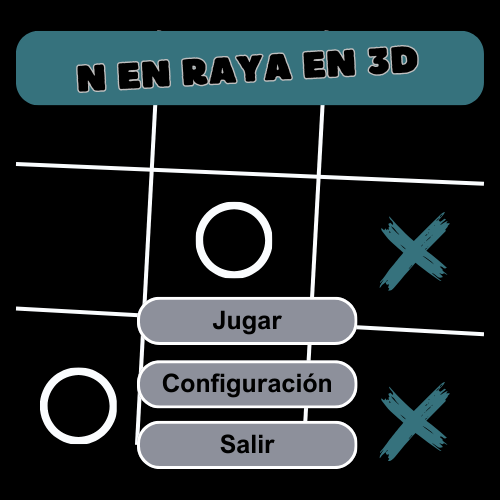
**Clase Tablero:**

* : dimensión del tablero como indicado por el usuario

**Clase Casilla:**

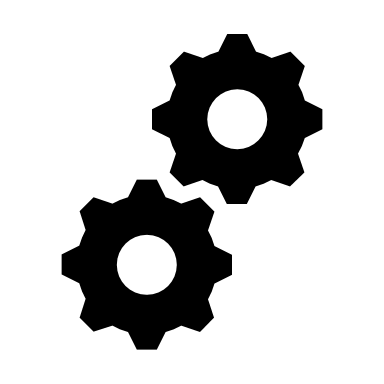
**Diseño del proyecto:**

**Menú principal:**

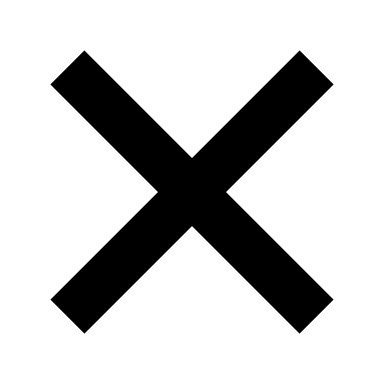
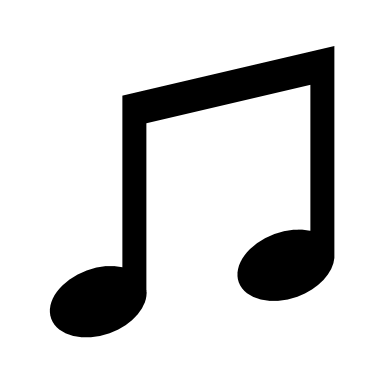
****

N en Raya 3D

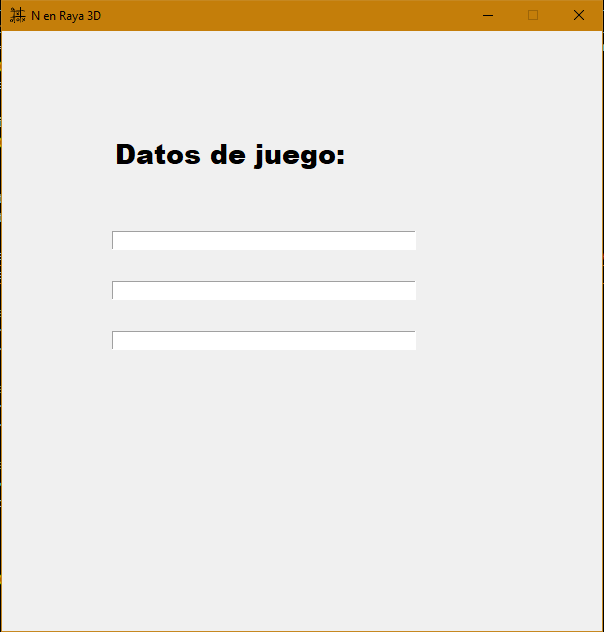
Jugar

****

Configuración

****

Salir

****

**Menú pre-juego:**

Datos de juego

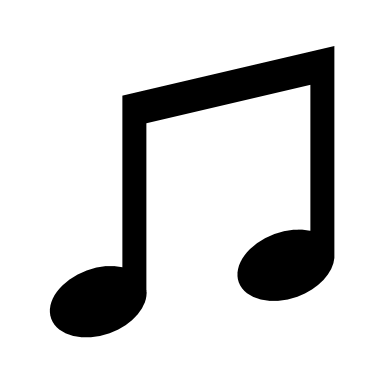
Ingrese el nombre del jugador 1:

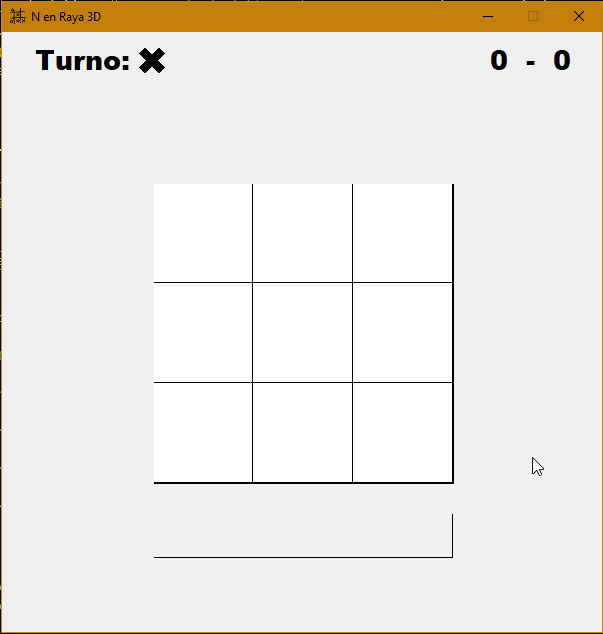
Ingrese el nombre del jugador 2:

Ingrese la dimensión del tablero:

Regresar

Iniciar

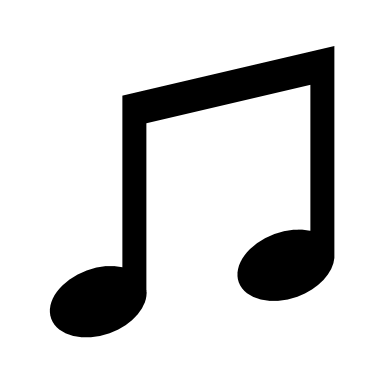
****

**Interfaz de juego:**

Turno:

0 – 0

David – Alan

****

David ha ganado (como era de esperarse)

**IDEAS:**

*Creación del tablero:*

Creamos una ‘clase madre’ llamada Tablero y una ‘clase hija’ llamada Casilla. Tablero se compone de Casillas.

* Tablero:
  + Atributos:
    - List[List[Casilla]]
    - Estado: “Ganado” o “En Juego”
    - Lado: int
    - Turno: int
  + Métodos:
    - Dibujar\_Tablero():
      * Crear\_Casilla():
    - Eliminar\_Tablero():
    - Ganar\_Tablero():
    - Empatar\_Tablero():
    - Reiniciar\_Tablero():
* Casilla:
  + Atributos:
    - Estado: int
    - Posicion: Tuple[int,int]
    - Lado: int
    - Tablero: Tablero
  + Métodos:
    - Dibujar\_Casilla():

Sobre los métodos de la clase Casilla:

* Dibujar\_Casilla():
  + Crear\_cuadrado(args\*, color)
    - Crear\_Evento(<Botton-1>)
      * Editar\_Estado()
* Editar\_Estado():
  + Condicionales (por definir)
    - Al\_cambiar() #llama a Cambiar\_Turno

idea

1. Crear un **lienzo** que contenga las dimensiones esperadas. (Vamos a poder mover el canvas)
2. Crear **cuadros** con las dimensiones que deberían tener las casillas.
3. Agregar un evento a esos cuadrados, de modo que cuando se pulsen dibujen una cruz o un círculo según en qué turno estamos.

**Cuadro**

**Lienzo**

**¿Cómo resolver el problema de la creación de las diversas casillas?**

Se me ocurre que podríamos crear una clase que sea la plantilla de las casillas.

Cada casilla debe tener una serie de atributos: lienzo, x, y, ancho, alto, estado.

También por lo menos el método crear\_casilla(lienzo, x, y, ancho, alto)

**Distribución de las casillas:**

Supongamos que el lienzo tiene tamaño M de píxeles.

Cuando el usuario indique cuántas casillas desea (N), lo que haremos es dividir M sobre N. Eso determinará el ancho de las casillas. Una vez tenemos el ancho, se me ocurre crear las casillas con una función que sea, por ejemplo, **generar\_tablero().**

En ella debería haber un bucle parecido a este:

Siendo