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 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.

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.

**¿Cómo uno puede alcanzar un diseño modular simple?**

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.

**¿Cómo podría el juego inicializarse y procesar la lógica detrás de N en raya?**

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 las siguientes dos ilustraciones puede uno apreciar la implementación abstracta de las clases `Tablero` y `Casilla`:

[Capte la atención de los lectores mediante una cita importante extraída del documento o utilice este espacio para resaltar un punto clave. Para colocar el cuadro de texto en cualquier lugar de la página, solo tiene que arrastrarlo.]

Ilustración 3: Implementación de la clase `Casilla`

[Capte la atención de los lectores mediante una cita importante extraída del documento o utilice este espacio para resaltar un punto clave. Para colocar el cuadro de texto en cualquier lugar de la página, solo tiene que arrastrarlo.]

Ilustración 2: Implementación de la clase `Tablero`

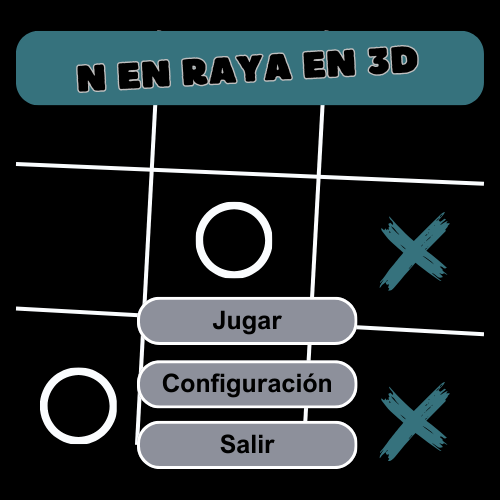


Enfocándonos exclusivamente en la experiencia del usuario e ignorando los detalles de la implementación, logramos definir el siguiente flujo de programa para el momento que el juego es inicializado:

Bloque de Código 2: Esquema general del flujo de programa de iniciar\_juego()

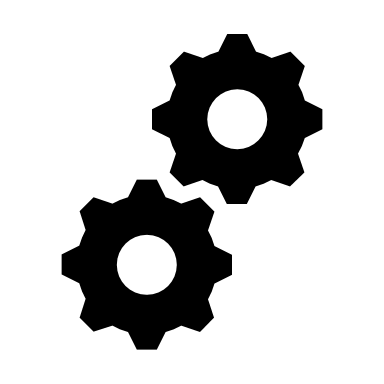
**Diseño del proyecto:**

**Menú principal:**

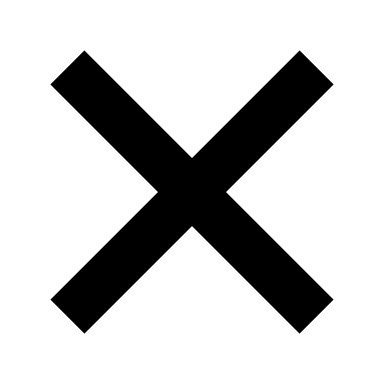
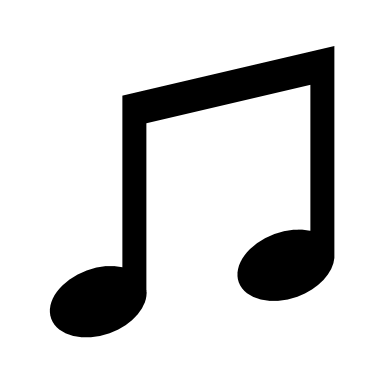
****

N en Raya 3D

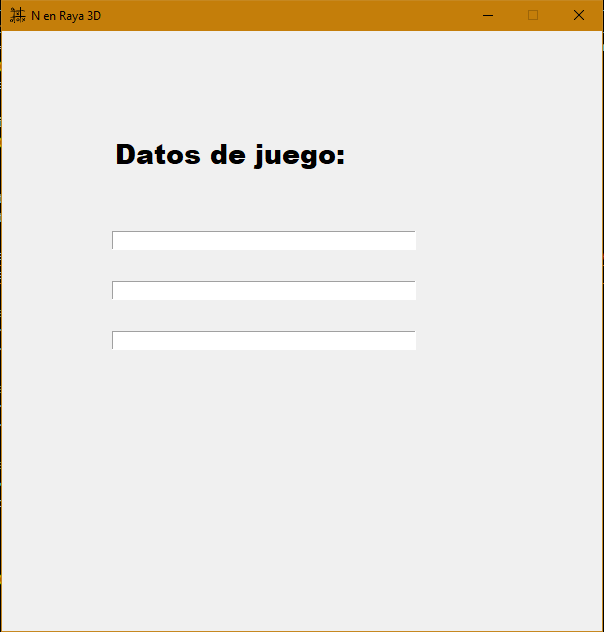
Jugar

****

Configuración

****

Salir

****

**Menú pre-juego:**

Datos de juego

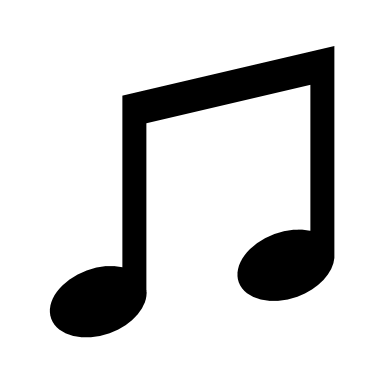
Ingrese el nombre del jugador 1:

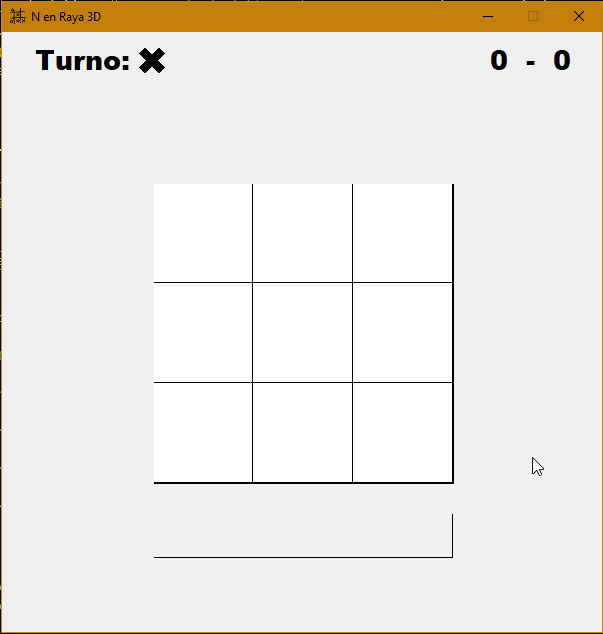
Ingrese el nombre del jugador 2:

Ingrese la dimensión del tablero:

Regresar

Iniciar

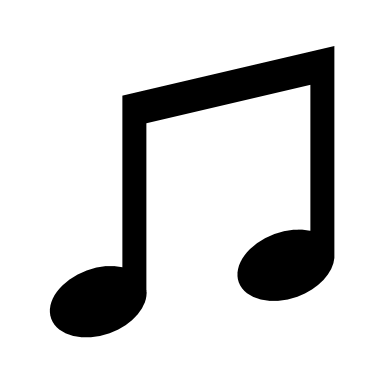
****

**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