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
* función global que, dado un patrón de victoria en el tablero o intertableros, actualiza la puntuación, muestra el patrón de victoria a los jugadores pintando las casillas a los que pertenece y coloca el botón continuar.
* : función global que analiza la condición de las casillas pertenecientes a cada tablero para detectar un patrón de victoria intertablero.
* : número natural único que identifica al tablero.
* : función global que, una vez ganada una partida, cambia el estado de todas las casillas en todos los tableros para evitar que cualquiera de los jugadores altere la configuración de los tableros al momento de una victoria.
* : arreglo de casillas que pertenecen al tablero.

**Clase Casilla:**

* : toda instanciación de una casilla debe pertenecer a un objeto de tipo `Tablero`.
* : método de `Tablero` que procesa todas las casillas pertenecientes al tablero. Así, cuando se cambie el estado de una de las casillas, este método es invocado para que procese el tablero y verifique la existencia de un patrón de victoria o no.
* **:** número natural que varía entre y que determinan si una casilla está vacía, fue tomada por el jugador 1 o el jugador 2, respectivamente. Es decir, identifica a una casilla si está vacía, tiene una cruz o un círculo. Este atributo es esencial para el procesamiento del tablero e intertableros.

Para la implementación del programa, las clases `Tablero` y `Casilla` reciben otros atributos cuyo uso es exclusivo de Tkinter y utilizados para la inserción de elementos gráficos esenciales para la jugabilidad del programa. Sin embargo, esos atributos fueron determinados arbitrariamente para la implementación del programa. En otras palabras, otro programador podría modificar a discreción las dimensiones de la ventana de juego y, consecuentemente, las del tablero de juego y, la de las casillas. Por ejemplo, la longitud del lado de cada tablero, considerado como un atributo, es función de la longitud en pixeles de uno de los lados de la ventana de Tkinter donde el juego toma lugar y la longitud del lado de cada una de las casillas es, a su vez, función de las dimensiones gráficas del tablero.

**Estado actual del proyecto:**

Inevitablemente, uno quisiera concluir la infabilidad de su trabajo después de probar algunos casos de prueba. Sin embargo, la experiencia y la implementación actual del programa nos impiden llegar a ese conclusión. En esta sección detallamos brevemente el error que encontramos en el estado actual del proyecto:

*Pobre o nula jugabilidad para*

Debido a la asignación de un valor constante para la dimensión de la ventana y las consecuentes funciones de las longitudes de los tableros y sus casillas como funciones de ésta, para una dimensión del *N en raya*mayor que 12, las casillas de juego comienzan a reducir dramáticamente su área, complicando así la apreciación del estado de las casillas y su precisa selección para cambiar su estado. O sea, ver si existe una cruz o un círculo o colocar cualquiera de ellos en una de las casillas. Además, para la casilla ubicada en la esquina inferior izquierda de cada uno de los tableros existe un cuadrado de área gris que desaparece el resto de los tableros para que así el usuario del juego pueda ver el tablero seleccionado en su totalidad y tener acceso a cada una de sus casillas; este cuadrado termina opacando la casilla, dificultando así su selección para cambiar su estado e inhibiendo en consecuencia el desarrollo del juego.

Naturalmente, el máximo valor de crece en proporción a la dimensión de la ventana, por lo que este error puede ser ‘ignorado’ a medida que uno agranda la ventana y aumenta el valor de prudentemente.

A pesar de este error, no es incorrecto afirmar que este programa no contiene ‘bugs’ que puedan alterar sobremanera la experiencia del usuario del programa. Asimismo, debido a la magnitud del proyecto trabajado y en vista del resultado final en función de las líneas de código, uno puede fácilmente divagar sobre la eficiencia en el uso de los recursos de la computadora y en la optimización del código del programa. Sin embargo, debido a que el asunto de la eficiencia y la optimización no fueron el enfoque en la implementación del programa y a la inversión de tiempo implícita en dichas tareas, decidimos deliberadamente mantenerlo en su estado actual.

**Conclusiones**

**Resultados obtenidos:**

Se creó un juego que cumple con las funcionalidades que cualquier usuario esperaría que tuviera un N en raya 3D; una interfaz gráfica atractiva que capta la atención del usuario sin abrumarlo con el color o elementos que despiten la atención del usuario de su estrategia en la partida. Además, debido a la naturaleza del juego y a la mismísima interfaz, el programa es sencillo de manejar.

**Recomendaciones:**

Para los siguientes programadores que planeen contribuir al desarrollo de este programa, sus autores les dejan sus recomendaciones que, por cuestiones de tiempo y capacidad, debieron permanecer como bocetos de ideas:

1. *Tamaño de la ventana*: el hecho de que el tamaño de la ventana no se puede modificar puede ser muy molesto para algunos usuarios, eso sin contar que este tipo de programas en ventanas lucen muy anticuados. Sería excelente que el programa permitiera incluso mostrarse en pantalla completa.
2. *Modo un jugador contra la computadora*: el juego es bastante entretenido, pero en ocasiones no se tiene otro jugador contra el que jugar; por lo que añadir un algoritmo contra el que pudieras jugar haría que el juego fuera más entretenido.
3. *On-line*: se podría implementar una modalidad online, de modo que se pudiera jugar contra otros jugadores alrededor del mundo.

**Dificultades presentadas:**

1. *Creación de casillas en el tablero*: este fue uno de los primeros problemas que se nos presentó durante la implementación del juego. Al principio no entendíamos cómo hacer que las casillas se mantuvieran en el tablero sin importar cuántas de ellas colocaras. Básicamente creamos el código a partir de pensar qué haríamos nosotros si fuéramos una máquina y nos pidieran que ordenáramos dichas casillas en un tablero de tamaño constante. Gracias a esto, colocar múltiples tableros en un mismo espacio resultó más sencillo.
2. *Modularización*: una de las dificultades más importantes fue construir el código de modo que fuera modular. Es decir, poder dividirlo en módulos más pequeños que se encargaran de cada una de las interfaces gráficas. El código se había escrito en un solo archivo que era bastante confuso. Lo realmente difícil fue traducir ese código a uno que fuera modular y ordenado.
3. *Diseño*: también fue algo tardado conseguir un diseño que nos gustara, ocupamos un tiempo importante probando distintos modelos en Canva.
4. *Resaltar líneas de victoria*: uno de los trabajos más tediosos y que postergamos hasta el final fue el de resaltar las líneas ganadoras. Desde el principio sabíamos que sería complicado resaltar las casillas porque nuestro programa podía identificar que había una victoria, pero no sabía reconocer qué tipo de victoria era. Finalmente, se resolvió creando una función que pintara las casillas y se le pasa un ID que identifica el tipo de victoria. El resultado nos gustó mucho.
5. *Verificación de empate*: en el caso 2D era bastante sencillo empatar y; por lo tanto, fue sencillo probar que nuestro código de empate funcionaba perfectamente, pero en el caso 3D es sumamente complicado conseguir un patrón de empate, tanto así que creemos que directamente no es posible (no hemos probado que sea matemáticamente imposible). Es por esto que verificar que nuestro algoritmo de empate ha resultado ser una odisea, cada vez que intentamos empatar alguien termina por ganar.

**Visión sobre la experiencia del desarrollo del proyecto:**

Trabajar en este proyecto fue una experiencia única. Al principio todo parecía muy abrumador y complicado, sin embargo, la lógica del juego la derivamos rápidamente. Eso nos motivó a seguir trabajando constantemente. Todos los días conseguíamos algo nuevo que modificar en el programa y ahora finalmente entendemos por qué los grandes proyectos requieren de tanto tiempo para su desarrollo.

Cada detalle que resolver se traducía en un cambio de lógica de todo el programa, pero era relativamente sencillo agregar cosas nuevas porque conocíamos muy bien cómo funcionaba nuestro código que está bien organizado. Así, a pesar de que uno de nosotros no hubiese estado presente durante el desarrollo de una determinada sección del programa, uno podría leer la documentación dejada por el otro en comentarios en el programa y contextualizar el código agregado con el resto de las funcionalidades ya creadas.

Además, visitamos muchísimos foros recónditos en Internet, chequeamos la documentación de Tkinter y Python, esto nos ayudó a comprender todo lo que se puede lograr hacer en el mismo.

Ahora sentimos que tenemos un conocimiento que realmente puede ser útil en el mercado y en nuestros proyectos. Además, nos sentimos emocionados porque por primera vez estamos experimentado el trabajo de código en conjunto con GitHub que era algo que teníamos pensando aprender y este proyecto fue la mejor excusa para lograrlo.