Universidad Simón Bolívar.

Ingeniería de la Computación.

CI2691 Laboratorio de algoritmos y estructuras I.

PROYECTO DEL LABORATORIO DE ALGORITMOS Y ESTRUCTURAS I: LA VIEJA N-DIMENSIONAL.

Alumnos:

Alejandro Salazar, 16-11080.

Cristian Inojosa, 17-10285.

Profesor:

Fernando Lovera.

CONTENIDO DEL INFORME

INTRODUCCIÓN

1.1 Breve descripción del problema.

La idea de este proyecto es crear un juego multijugador (2) sencillo, mediante las buenas prácticas de programación aprendidas en los cursos de CI2611 y CI2691, así como el desarrollo de una interfaz gráfica para este. El juego consiste en una versión N-dimensional de la popular ´´La Vieja´´, pero en vez de ser 1 tablero de 3*3, serán N tableros de N*N.

En La Vieja Tradicional se hace punto al colocar una línea horizontal, vertical o diagonal de tres fichas, en este juego se buscara que los puntos se hagan al hacer una línea horizontal, vertical o diagonal de N fichas y cuando se marque una ficha en la misma posición de los N tableros.

1.2 Motivación.

Aprender nuevas formas de diseñar y crear interfaces gráficas mediante el uso librerías, en este caso, se usará la librería Pygame del lenguaje de programación Python. También se buscará dividir el problema principal de crear La Vieja N-dimensional en sub-problemas más pequeños y sencillos para ejercitar el análisis descendente de problemas, técnica vista en el curso teórico.

1.3 Objetivos planteados.

En principio, se buscara desarrollar la parte lógica del juego, esto es, un programa que guarde las posiciones en las que han jugado los participantes, y que de acuerdo a esto, el programa lleve la cuenta del puntaje de cada jugador, según las reglas del juego.

Luego, se plantea representar la parte lógica del juego de manera gráfica, esto es, mostrar N tableros de N*N, donde queden marcadas las fichas ya jugadas por cada jugador, se muestren los nombres de ambos, se publique el puntaje en vivo, y al finalizar se dé la opción jugar nuevamente.

1.4 Alcance de la solución.

En cuanto al alcance de este juego, se plantea cumplir con todo lo demandado por el enunciado, y se supera las expectativas en algunos ámbitos, por ejemplo, en principio, el máximo N es 11, nuestra versión está prevista y funciona para N mucho mayores, se cumplirá con la posibilidad de jugar nuevamente guardando el resultado previo. No se plantea la posibilidad de una cantidad de jugadores distinta de 2.

DISEÑO

2.1 Resultado del análisis descendiente.

En cuanto al porque decidimos separar el programa de la manera en la que los hicimos:

Primeramente Tablero y Jugador porque son clases, y se hace más sencillo manejarlas por separado, y es una buena práctica que las clases estén en su propio archivo.

También buscamos separar los archivos donde se declaran las funciones de aquellos donde se utilizan, en nuestro caso se declaran las funciones en gui.py, tablero.py, y logic.py, esto nos trae ventajas como al momento de modificar el código, solo lo hagamos una vez y de manera ordenada.

Asimismo, distinguimos main.py de main2.py, dado que en main.py es un programa sin la interfaz gráfica, donde se probó primeramente que el juego funcionara y que la lógica de nuestras funciones en los archivos gui.py y logic.py estuviera correcta, luego creamos main2.py, donde le colocamos la interfaz gráfica con la ayuda PyGame y la lógica de las funciones en los archivos tablero.py y logic.py.

Sintetizando, podemos decir que principalmente buscamos separar los archivos donde creamos las funciones (gui.py, tablero.py y logic.py) de aquellos donde las utilizamos (main.py y main2.py).

2.2 Especificación y desarrollo de los programas, procedimientos y funciones.

En los códigos de esta parte se excluyenlas definiciones de variables y la importación de librerías y clases y se centra en las funciones de los archivos.

2.21 Tablero.py: Este clase crea la interfaz (Pro) de los tableros en main2.py, y define las funciones _init_, dibujar, esta_adentro, dibujar_fichas. Código en Python (PyGame no existe en GCL).

```
1.  class Tablero:
2.
3.      def __init__(self, x:int, y:int, ancho:int, alto:int, n:int):
4.          self.x = x
5.          self.endx = x + ancho
6.          self.y = y
```

```
7.          self.endy = y + alto
8.          self.ancho = ancho
9.          self.alto = alto
10.         self.n = n
11.         self.ancho_celda = ancho // n
12.         self.alto_celda = alto // n
13.
14.     def dibujar(self, superficie, color, borde = 2):
15.
16.         for i in range(1, self.n):
17.
18.             x = self.x + self.ancho_celda*i
19.             y = self.y + self.alto_celda*i
20.
21.             pygame.draw.aaline(superficie, color, (self.x, y), (self.endx, y), bor
    de)
22.             pygame.draw.aaline(superficie, color, (x, self.y), (x, self.endy), bor
    de)
23.
24.     def esta_adentro(self, punto):
25.         if self.x <= punto[0] <= self.endx and self.y <= punto[1] <= self.endy:
26.             self.ultima_casilla = ((punto[1] -
     self.y) // self.alto_celda, (punto[0] - self.x) // self.ancho_celda)
27.             return True
28.
29.         return False
30.
31.     def dibujar_fichas(self, superficie, tablero, colores):
32.         for i in range(0, self.n):
33.             for j in range(0, self.n):
34.                 y = self.y + self.alto_celda // 2 + self.alto_celda * i
35.                 x = self.x + self.ancho_celda // 2 + self.ancho_celda * j
36.                 r = min(self.ancho_celda, self.alto_celda) // 4
37.                 if tablero[i][j] != 0:
38.
39.                     pygame.gfxdraw.filled_circle(superficie, x, y, r, colores[tabl
    ero[i][j]-1])
40.                     pygame.gfxdraw.aacircle(superficie, x, y, r, colores[tablero[i
    ][j]-1])
```

2.22 Gui.py: Esta función sirve para definir funciones que usaremos para dibujar en tablero en nuestra versión básica del juego y sus funciones son: (main.py). Código en Python porque se usa la función Input y Print. (No existen en GCL).

```
1.  def dibujar_tablero(tab:[[int]], N:int) -> None:
2.
3.      for i in range(N):
4.          line = ""
5.          for j in range(N):
6.              line += ' x ' if tab[i][j] == 1 else (' o ' if tab[i][j] == 2 else '
     ')
7.              line += '|' if j < N-1 else ''
8.
9.          print(line)
10.         if(i < N-1):
```

```
            print('-'*(4*N-1))

def dibujar_supertablero(supertab:[[[int]]], N:int) -> None:
    for index,tab in enumerate(supertab):
        print('\n Tablero {}:'.format(index))
        dibujar_tablero(tab, N)

def borrar():
    os.system('cls' if os.name=='nt' else 'clear')

def obtener_jugada(nombre:str) -> None:
    print("Juega {}".format(nombre))
    return int(input("Ingrese el número del tablero en el que desea jugar \n")), \

    int(input("Ingrese el número de la fila en la que desea jugar \n")), \
    int(input("Ingrese el número de la columna en la que desea jugar \n"))

def imprimir_puntaje(jugadores:[Jugador]) -> None:
    print("Puntaje:")
    for jugador in jugadores:
        print("{}: {}".format(jugador.nombre, jugador.puntaje))

def preguntar_seguir_jugando() -> bool:
    seguir = str(input("Desea seguir jugando? [y] \n"))
    return True if(seguir.lower() == "y") else False
```

2.23 Jugador.py: En este archivo se crea la clase jugador, y sirve para guardar el de los jugadores e inicializar su puntaje. Código en Python porque es una clase (No existe en GCL).

```
class Jugador:
    def __init__(self, nombre:str):
        self.nombre = nombre
        self.puntaje = 0
```

2.24 Logic.py: Esta es la pieza fundamental de los programa main.py y main2.py, dentro del código aparece como comentario los parámetros, lo que retorna y el comportamiento de cada función. (PseudoCódigo en GCL)

```
func esValida(T: array [0..N)x[0..N)x[0..N) of int, N, tab, fila, col:int)
{pre N > 0}
{post result = 0 <= tab <= N /\ 0 <= fila <= N /\ 0 <= col <= N /\ (T[tab][fila][c
ol] = 0)}
|[
  var result:bool;

  result := 0 <= tab <= N /\ 0 <= fila <= N /\ 0 <= col <= N /\ (T[tab][fila][col]
 = 0);

  >> result
]|
```

```
11.
12. func hayLineaHorizontal(A:array[0..N)x[0..N)x[0..N), N, tab, fila:int)
13. { pre N>0 /\ 0 <= tab < N /\ 0 <= fila < N }
14. { post result = (%forall i | 0 <= i < N-1 : A[i]=A[i+1]) }
15. |[
16.   var result: bool;
17.   var i: int;
18.   result, i := true, 0;
19.
20.   { inv result = (%forall j | 0 <= j < i : A[j] = A[j+1]) }
21.   { cota N-i }
22.   do (i < N-1 /\ result) -> result, i := A[i] = A[i+1], i+1 od
23.
24. ]|
25. def quedan_fichas(fichas:int) -> bool:
26.
27.     """
28.     Parámetros:
29.     fichas (int): el número de fichas que quedan por jugar
30.     Retorna:
31.     (bool) Verdadero si fichas es mayor que 0, Falso en caso contrario
32.     """
33.
34.     return True if fichas > 0 else False
35.  # funciones esvalida y hay linea horizontal en python.
36. def es_valida(T:[[[int]]], N:int, tab:int, fila:int, col:int) -> bool:
37.
38.     """
39.     Parámetros:
40.     T ([[[int]]]): lista de lista de lista de enteros, representando los tableros
41.     N (int): el número de tableros,filas y columnas
42.     tab (int): el tablero de la celda jugada
43.     fila (int): la fila de la celda jugada
44.     col (int): la columna de la celda jugada
45.     Retorna:
46.     (bool) Verdadero si la jugada es válida, Falso en caso contrario
47.     Comportamiento:
48.     Checa que tanto el tablero como la fila y la columna sean mayores o iguales qu
    e 0 y menores que N,
49.     y que la celda jugada no haya sido jugada aún (T[tab][fila][col] == 0)
50.     """
51.
52.     return 0 <= tab < N and 0 <= fila < N and 0 <= col < N and T[tab][fila][col] =
    = 0
53.
54. def hay_linea_horizontal(T:[[[int]]], N:int, tab:int, fila:int) -> bool:
55.
56.     """
57.     Parámetros:
58.     T ([[[int]]]): lista de lista de lista de enteros, representando los tableros
59.     N (int): el número de tableros,filas y columnas
60.     tab (int): el tablero de la celda jugada
61.     fila (int): la fila de la celda jugada
62.     Retorna:
63.     (bool) Verdadero si todas las celdas de la fila tienen el mismo valor Falso en
     caso contrario
64.     """
65.
```

```
66.     return all(T[tab][fila][i] != 0 and T[tab][fila][i] == T[tab][fila][i+1] for i
     in range(N-1))
67.
68. def hay_linea_vertical(T:[[[int]]], N:int, tab:int, col:int) -> bool:
69.
70.     """
71.     Parámetros:
72.     T ([[[int]]]): lista de lista de lista de enteros, representando los tableros
73.     N (int): el número de tableros,filas y columnas
74.     tab (int): el tablero de la celda jugada
75.     col (int): la fila de la celda jugada
76.     Retorna:
77.     (bool) Verdadero si todas las celdas de la columna tienn el mismo valor, Falso
     en caso contrario
78.     """
79.
80.     return all(T[tab][i][col] != 0 and T[tab][i][col] == T[tab][i+1][col] for i in
     range(N-1))
81.
82. def hay_linea_diagonal(T:[[[int]]], N:int, tab:int, fila:int, col:int) -> int:
83.
84.     """
85.     Parámetros:
86.     T ([[[int]]]): lista de lista de lista de enteros, representando los tableros
87.     N (int): el número de tableros,filas y columnas
88.     tab (int): el tablero de la celda jugada
89.     fila (int): la fila de la celda jugada
90.     col (int): la fila de la celda jugada
91.     Retorna:
92.     (int) El número de lineas diagonales que hay
93.     Comportamiento:
94.     Checha si la fila y la columna son iguales, en dado caso, la celda forma parte
     de la diagonal
95.     principal, por lo que checa que todas las celdas cuyas filas y columnas son ig
    uales tengan el mismo valor.
96.     Si suecede, suma 1 a result
97.     Checa que la suma de la fila con la columna sea igual a N-
    1, en dado caso, la celda forma parte de la
98.     diagonal secundaria, por lo que checha que todas las celdas cuyas filas y colu
    mnas suman N-1 tengan el mismo valor.
99.     Si sucede, suma 1 a result
100.          Retorna result
101.          """
102.          result = 0;
103.          if(fila==col):
104.              result += 1 if all(T[tab][i][i] != 0 and T[tab][i][i] == T[tab][i+
    1][i+1] for i in range(N-1)) else 0
105.          if(fila+col == N-1):
106.              result += 1 if all( T[tab][i][N-1-i] != 0 and T[tab][i][N-1-
    i] == T[tab][i+1][N-2-i]  for i in range(N-1)) else 0
107.
108.          return result
109.
110.      def hay_linea_tableros(T:[[[int]]], N:int, fila:int, col:int) -> bool:
111.
112.          """
113.          Parámetros:
114.          T ([[[int]]]): lista de lista de lista de enteros, representando los t
    ableros
```

```
115.            N (int): el número de tableros,filas y columnas
116.            fila (int): la fila de la celda jugada
117.            col (int): la fila de la celda jugada
118.
119.            Retorna:
120.            (bool) Verdadero si todas las celdas [fila][col] de los diferentes tab
    leros tienen el mismo valor,
121.            Falso en caso contrario
122.            """
123.
124.            return all(T[i][fila][col] != 0 and T[i][fila][col] == T[i+1][fila][co
    l] for i in range(N-1))
125.          #
126.        def hay_linea(T:[[[int]]], N:int, tab:int, fila:int, col:int) -> bool:
127.
128.            """
129.            Parámetros:
130.            T ([[[int]]]): lista de lista de lista de enteros, representando los t
    ableros
131.            N (int): el número de tableros,filas y columnas
132.            tab (int): el tablero de la celda jugada
133.            fila (int): la fila de la celda jugada
134.            col (int): la fila de la celda jugada
135.            Retorna:
136.            (bool) Verdadero si se formó una línea en la fila, columna, diagonal o
     columna tridimensional de la celda
137.            jugada, Falso en caso contrario
138.            """
139.
140.            return any([hay_linea_horizontal(T, N, tab, fila), hay_linea_vertical(
    T, N, tab, col),\
141.                hay_linea_diagonal(T, N, tab, fila, col), hay_linea_tableros(T, N,
     fila, col)])
142.
143.        def sumar_lineas(T:[[[int]]], N:int, tab:int, fila:int, col:int, turno:int
    , jugadores:[Jugador]) -> None:
144.
145.            """
146.            Parámetros:
147.            T ([[[int]]]): lista de lista de lista de enteros, representando los t
    ableros
148.            N (int): el número de tableros,filas y columnas
149.            tab (int): el tablero de la celda jugada
150.            fila (int): la fila de la celda jugada
151.            col (int): la fila de la celda jugada
152.            Retorna:
153.            result (int): el número de líneas formadas al jugar la celda
154.
155.            Comportamiento:
156.            Inicializa result en 0, suma 1 por cada línea formada
157.            """
158.
159.            result = 0
160.            result += 1 if hay_linea_horizontal(T, N, tab, fila) else 0
161.            result += 1 if hay_linea_vertical(T, N, tab, col) else 0
162.            result += hay_linea_diagonal(T, N, tab, fila, col)
163.            result += 1 if hay_linea_tableros(T, N, fila, col) else 0
164.
165.            jugadores[turno].puntaje += result
```

```
166.
167.       def reflejar_jugada(T:[[[int]]], tab:int, fila:int, col:int, turno:int) -
    > None:
168.
169.           """
170.           Parámetros:
171.           T ([[[int]]]): lista de lista de lista de enteros, representando los t
    ableros
172.           N (int): el número de tableros,filas y columnas
173.           tab (int): el tablero de la celda jugada
174.           fila (int): la fila de la celda jugada
175.           col (int): la fila de la celda jugada
176.           turno (int): 0 si es el turno del jugador 1, 1 si es el turno del juga
    dor 2
177.           Comportamiento:
178.           Cambia la celda T[tab][fila][col] a 1 si es el turno del jugador 1, 2
    si es el turno del jugador 2
179.           """
180.
181.           T[tab][fila][col] = turno+1
182.
183.       def cambiar_jugador(turno:int) -> int:
184.
185.           """
186.           Parámetros:
187.           turno (int): 0 si es el turno del jugador 1, 1 si es el turno del juga
    dor 2
188.           Retorna:
189.           (int) 0 si es el turno del jugador 2, 1 si es el turno del jugador 1
190.           """
191.
192.           return 0 if turno else 1
```

2.25 Main.py: Versión sin interfaz gráfica del juego, para ser utilizada desde la terminal (PseudoCódigo en GCL, excepto una parte no Traducible a GCL)

```
1.  # partes código se mantiene en python porque se usa la funciones no definidas en
    GCL
2.
3.  Do (otra_partida)->
4.
5.      tablero = [[[0 for k in range(N)] for j in range(N)] for i in range(N)]
6.      turno = empieza
7.      empieza = cambiar_jugador(empieza)
8.      fichas = N**3
9.
10.     Do (quedan_fichas(fichas))->
11.
12.         dibujar_supertablero(tablero, N)
13.
14.         Do (swap)->
15.
16.             tab, fila, col = obtener_jugada(jugadores[turno].nombre)
17.
18.             if(es_valida(tablero, N, tab, fila, col)):
```

```
                Swap:=False
            [] (!es_valida(tablero, N, tab, fila, col:
                print("Jugada inválida")
        od
        reflejar_jugada(tablero, tab, fila, col, turno)

        borrar()

        if(hay_linea(tablero, N, tab, fila, col)):
            sumar_lineas(tablero, N, tab, fila, col, turno, jugadores)

        imprimir_puntaje(jugadores)
        turno = cambiar_jugador(turno)

        fichas = fichas - 1
  od
    borrar()
    imprimir_puntaje(jugadores)
    otra_partida = preguntar_seguir_jugando()
```

2.26 Main2.py: Versión final del juego, con la interfaz gráfica incluida (Código en Python porque usa PyGame, que no está disponible en GCL).

```
import sys, pygame, pygame.freetype
from Tablero import Tablero
from logic import *
from Jugador import Jugador

# Configuración de Pygame
pygame.init()
tamaño = ancho, alto = 800, 600
pantalla = pygame.display.set_mode(tamaño)

font = pygame.freetype.SysFont('Arial', 24)

# Colores
negro = 0, 0, 0
blanco = 255, 255, 255
azul = 0, 102, 255
rojo = 255, 0, 0

# Configuración pantalla introductoria
intro = True
jugadores = [Jugador(""), Jugador("")]
turno = 0
N = ""
n_error = False

# Pantalla introductoria
while intro:

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

```

```
33.             sys.exit()
34.
35.         #elif event.type == pygame.MOUSEBUTTONDOWN:
36.             #intro = False
37.
38.         elif event.type == pygame.KEYDOWN:
39.
40.             if event.key == pygame.K_RETURN:
41.
42.                 if turno < 2:
43.
44.                     turno += 1
45.
46.                 else:
47.                     try:
48.
49.                         assert int(N) > 0
50.                         intro = False
51.
52.                     except:
53.                         n_error = True
54.
55.
56.             elif turno < 2:
57.
58.                 if event.key == pygame.K_BACKSPACE:
59.
60.                     jugadores[turno].nombre = jugadores[turno].nombre[:-1]
61.
62.                 else:
63.
64.                     jugadores[turno].nombre += event.unicode
65.
66.             else:
67.
68.                 if event.key == pygame.K_BACKSPACE:
69.
70.                     N = N[:-1]
71.
72.                 else:
73.                     valid_keys = "0123456789"#["0". "1", "2", "3", "4", "5", "6",
    "7", "8", "9"]
74.
75.                     if event.unicode in valid_keys:
76.                         N += event.unicode
77.
78.     pantalla.fill(negro)
79.
80.     font.render_to(pantalla, (50, 50), "Ingrese el nombre del jugador 1:", blanco)

81.     font.render_to(pantalla, (50, 80), jugadores[0].nombre, blanco)
82.
83.     if turno > 0:
84.         font.render_to(pantalla, (50, 110), "Ingrese el nombre del jugador 2:", bl
    anco)
85.         font.render_to(pantalla, (50, 140), jugadores[1].nombre, blanco)
86.
87.     if turno > 1:
```

```
88.          font.render_to(pantalla, (50, 170), "Ingrese el número de tableros,", blan
    co)
89.          font.render_to(pantalla, (50, 200), "filas y columnas:", blanco)
90.          font.render_to(pantalla, (50, 230), N, blanco)
91.
92.      if n_error:
93.          font.render_to(pantalla, (50, 500), "N debe ser mayor que 0", rojo)
94.
95.      pygame.display.flip()
96.
97. # Configuración inicial del juego
98.
99. N = int(N)
100.        tablero_display = Tablero(100, 100, 400, 400, N)
101.        primer_jugador = 0
102.        otra_partida = True
103.
104.        while otra_partida:
105.
106.            # Configuración inicial de la partida
107.            turno = primer_jugador
108.            primer_jugador = cambiar_jugador(primer_jugador)
109.            tab = 0
110.            tablero = [[[0 for k in range(N)] for j in range(N)] for i in range(N)
    ]
111.            fichas = N**3
112.
113.
114.            # Pantalla Principal del Juego
115.            while quedan_fichas(fichas):
116.
117.                for event in pygame.event.get():
118.
119.                    if event.type == pygame.QUIT:
120.
121.                        sys.exit()
122.
123.                    elif event.type == pygame.MOUSEBUTTONDOWN:
124.
125.                        pos = pygame.mouse.get_pos()
126.
127.                        if pygame.mouse.get_pressed()[0]:
128.
129.                            if tablero_display.esta_adentro(pos):
130.                                fila, columna = tablero_display.ultima_casilla
131.
132.                                try:
133.                                    assert es_valida(tablero, N, tab, fila, column
    a)
134.
135.                                    reflejar_jugada(tablero, tab, fila, columna, t
    urno)
136.
137.                                    if hay_linea(tablero, N, tab, fila, columna):

138.                                        sumar_lineas(tablero, N, tab, fila, column
    a, turno, jugadores)
139.
```

```
                                    #print("Puntaje: \n {jugador1}: {puntaje1} \n {jugador2}: {puntaje2}".format(jugador1 = jugadores[0].nombre, jugador2 = jugadores[1].nombre, puntaje1 = jugadores[0].puntaje, puntaje2 = jugadores[1].puntaje))

                                    fichas -= 1
                                    turno = cambiar_jugador(turno)

                                except:
                                    pass
                                    #print("Jugada inválida")


                            elif 274 <= pos[0] <= 326:
                                if 49 <= pos[1] <= 88:
                                    tab -= 1 if tab > 0 else 0
                                elif 512 <= pos[1] <= 551:
                                    tab += 1 if tab < N-1 else 0

                pantalla.fill(blanco)
                pygame.draw.rect(pantalla, (236, 240, 241), (580, 0, 800, 600))
                pygame.draw.polygon(pantalla, azul, [(300, 49), (274, 88), (326, 88)])
                pygame.draw.polygon(pantalla, azul, [(300, 551), (274, 512), (326, 512)])
                tablero_display.dibujar(pantalla, negro)
                tablero_display.dibujar_fichas(pantalla, tablero[tab], [azul, rojo])
                font.render_to(pantalla, (255, 10), "Tablero {}".format(tab), negro)
                font.render_to(pantalla, (600, 10), "Puntaje:", negro)
                font.render_to(pantalla, (600, 40), jugadores[0].nombre, negro)
                font.render_to(pantalla, (600, 70), str(jugadores[0].puntaje), negro)
                font.render_to(pantalla, (600, 100), jugadores[1].nombre, negro)
                font.render_to(pantalla, (600, 130), str(jugadores[1].puntaje), negro)
                font.render_to(pantalla, (600, 160), "Turno: {}".format(jugadores[turno].nombre), negro)
                pygame.display.flip()

            outro = True
            # Pantalla Final
            while outro:

                for event in pygame.event.get():
                    if event.type == pygame.QUIT:
                        sys.exit()
                    if event.type == pygame.MOUSEBUTTONDOWN:
                        pos = pygame.mouse.get_pos()
                        print(pos)

                        if pygame.mouse.get_pressed()[0]:
                            if 230 <= pos[1] <= 260:
                                if 50 <= pos[0] <= 80:
                                    outro = False
                                elif 100 <= pos[0] <= 130:
                                    otra_partida = False
                                    outro = False

```

```
                pantalla.fill(negro)
                font.render_to(pantalla, (50, 50), "Puntaje:", blanco)
                font.render_to(pantalla, (50, 80), jugadores[0].nombre, blanco)
                font.render_to(pantalla, (50, 110), str(jugadores[0].puntaje), bla
    nco)
                font.render_to(pantalla, (50, 140), jugadores[1].nombre, blanco)
                font.render_to(pantalla, (50, 170), str(jugadores[1].puntaje), bla
    nco)
                font.render_to(pantalla, (50, 200), "¿Desea seguir jugando?", blan
    co)
                font.render_to(pantalla, (50, 230), "Si", blanco)
                font.render_to(pantalla, (100, 230), "No", blanco)
                pygame.display.flip()
```

2.3 Demostraciones de correctitud.

En esta sección haremos la demostración de correctitud de las funciones solicitadas (esValida y HayLineaHorizontal) por el enunciado, con su respectiva completa traducción a GCL

- Función EsValida

```
func esValida(T: array [0..N)x[0..N)x[0..N) of int, N, tab, fila,
col:int)
{pre N > 0}
{post result = 0 <= tab <= N /\ 0 <= fila <= N /\ 0 <= col <= N /\
(T[tab][fila][col] = 0)}
|[
  var result:bool;

  result := 0 <= tab <= N /\ 0 <= fila <= N /\ 0 <= col <= N /\
(T[tab][fila][col] = 0);

  >> result
]|
```

Prueba:

{N > 0} S { result = 0 <= tab <= N /\ 0 <= fila <= N /\ 0 <= col <= N /\ (T[tab][fila][col] = 0) }

(result = 0 <= tab <= N /\ 0 <= fila <= N /\ 0 <= col <= N /\ (T[tab][fila][col] = 0))[result:= 0 <= tab <= N /\ 0 <= fila <= N /\ 0 <= col <= N /\ (T[tab][fila][col] = 0)]

= <Substitución Textual>
0 <= tab <= N /\ 0 <= fila <= N /\ 0 <= col <= N /\ (T[tab][fila][col] = 0) = 0 <= tab <= N /\ 0 <= fila <= N /\ 0 <= col <= N /\ (T[tab][fila][col] = 0)
= <true=q=q>
true
<= <p=>true>
N > 0

- **Función Hay Línea horizontal.**

```
func hayLineaHorizontal(A:array[0..N)x[0..N)x[0..N), N, tab,
fila:int)
{ pre N>0 /\ 0 <= tab < N /\ 0 <= fila < N }
{ post result = (%forall i | 0 <= i < N-1 : A[i]=A[i+1])  }
|[
  var result: bool;
  var i: int;
  result, i := true, 0;

  { inv result = (%forall j | 0 <= j < i : A[j] = A[j+1]) /\ 0 <=
i <= N-1 }
  { cota N-i }
  do (i < N-1 /\ result) -> result, i := A[i] = A[i+1], i+1 od

]|
```

Prueba {H} S {P}:

(result = (%forall j | 0 <= j < i : A[j] = A[j+1]) [result, i := True, 0]
= <Sustitución Textual>
(True = (%forall j | 0 <= j < 0 : A[j] = A[j+1]

= <Rango vacío 0 <= j < 0 >
true=true

= <true=q=q>

true
<= <p => true >
N>0 /\ 0 <= tab < N /\ 0 <= fila < N

Prueba: {P /\ B /\ t=c} S {t<c}

T<N-i [result, i := A[i] = A[i+1], i+1]
= <Sustitución textual>

T<N-i-1

<= < propiedad de desigualdad>

T<= N-i

= < def de desigualdad>

T=N-i V T<N-i

<= < p=>p v q>

T=n-i

<= < p /\ q=>p >

result = (%forall j | 0 <= j < i : A[j] = A[j+1]) /\ 0 <= i <= N-1 /\ T=n-i

Prueba: P /\ B => T>=0

result = (%forall j | 0 <= j < i : A[j] = A[j+1]) /\ 0 <= i <= N-1 /\ N>0 /\ i<N-1

=> < p /\ q => p , aritmetica>

N>0 /\ N-i>1

=> < p /\ q => p , p /\ True=p , 1>=0 0 true >

N-i>1 /\ 1 >= 0

=> < Transitividad>

N-i>=0

Prueba {P /\ B} S {P}:

(i < N-1 /\ result /\ result = (%forall j | 0 <= j < i : A[j] = A[j+1]))[result, i := A[i] = A[i+1], i+1] => inv

H0: i+1 < N-1
H1: A[i] = A[i+1]
H2: A[i] = A[i+1] = (%forall j | 0 <= j < i + 1 : A[j] = A[j+1]

A[i] = A[i+1] = (%forall j | 0 <= j < i + 1 : A[j] = A[j+1]
  <H1>
true = (%forall j | 0 <= j < i + 1 : A[j] = A[j+1]

<H0 /\ (a < b /\ b < c => a < c) >
true = (%forall j | 0 <= j < N-1 : A[j] = A[j+1]
<def S.T>
P[result, i:= true, N-1]

Prueba: P /\ ¬B => Q

result = (%forall j | 0 <= j < i: A[j] = A[j+1]) /\ (N-1<=i V !result) /\ 0 <= i <= N-1

=> <Transitividad j<i /\ i<=N-1>

result = (%forall j | 0 <= j < N-1: A[j] = A[j+1]) /\ (N-1<=i V !result) /\ 0 <= i <= N-1

=> < p /\ q=>p >

result = (%forall j | 0 <= j < N-1: A[j] = A[j+1])

= <Change of dummy j:=i >

result = (%forall j | 0 <= i < N-1: A[i] = A[i+1])

2.4 Optimizaciones.

Para la realización de este juego realmente no se hizo ninguna optimización del concepto original al resultado final.

Sin embargo, algunas optimizaciones que están fuera del alcance del curso, que se le podrían añadir al juego son:

- Un ajuste del tamaño de la ventana automático, que dependa del tamaño de la pantalla del usuario.
- Un algoritmo que pueda competir inteligentemente contra el usuario en una versión de un solo jugador.
- Crear una base de datos que registre un ranking de las victorias por cada usuario.

DETALLES DE IMPLEMENTACIÓN

3.1 Tipos de datos.

En nuestro programa usamos datos de tipo booleano, entero, y string.

Las funciones de *es_valida, quedan_fichas* y las de *hay_linea* (verical, horizontal, diagonal y en todos los tableros), *tablero_display_esta_adentro,* y *pregunta_seguir_jugando* son funciones de tipo booleano.

Para captar los nombres de los jugadores (input) y reflejarlos (output) en la pantalla se usaron variables de tipo string, igualmente

Para obtener el N (input) de las dimensiones del juego solo permite colocar variables enteras y positivas, para cambiar de turno entre jugadores se usa una variable llamada *turno* que es de tipo entero, de igual manera para reflejar el puntaje se usan enteros.

También se usaron procedimientos y funciones de la librería PyGame en *tablero.py* y *main2.py* que sirven para hacer la interfaz, pero estos no pertenecen a los booleanos, enteros, o strings

3.2 Estructuración del código.

El código se estructura principalmente en dos partes, una sin la interfaz gráfica llamada main.py y otra con ella main2.py.

Decidimos, como buena práctica, separar los archivos donde se declaran las funciones de donde se utilizan. Y un archivo para cada clase creada (jugador.py y tablero.py).

El archivo main.py, depende de los archivos gui.py, logic.py y jugador.py. En el caso de gui,py, este guarda las funciones necesarias dibujar el tablero en la terminal. Para logic.py, este tiene las funciones para llevar la parte lógica del juego y decidir cuando se hace punto y los turnos. Y jugador.py sirve para guardar los nombres y el puntaje. Todas estas son armadas en main.py para mostrar la versión simple del juego.

Para el archivo main2.py, este depende de los archivos tablero.py, logic.py y jugador.py. En el caso de tablero,py, este presenta las funciones necesarias dibujar la interfaz gráfica del tablero. Para logic.py, es parecido a main.py, este

tiene las funciones para llevar la parte lógica del juego y decidir cuando se hace punto y los turnos. Y jugador.py sirve para guardar los nombres y el puntaje. Similarmente a main.py, en este caso se arman las funciones para crear el juego, con la diferencia del uso de la librería PyGame para crear la interfaz**.**

3.3 Comentario adicional con respecto a la traducción del programa.

En principio, el programa fue realizado en Python3, y luego se realizó el informe por eso, en algunas partes de las sección *2.2*, no fueron traducidos por completos los archivos de Python a GCL, puesto que ciertas funciones o procedimientos son propios de Python, PyGame o simplemente no están definidas en GCL. Sin embargo, todas las funciones solicitadas por el enunciado han sido traducidas en GCL para sus pruebas de correctitud.

ESTADO ACTUAL

4.1 Operatividad del programa.

Actualmente el juego funciona con total normalidad, y cumple con las siguientes precisiones:

- Permite colocar los nombres/nicknames de ambos participantes.
- Se puede colocar el valor N, con la restricción que N tiene que ser un entero mayor que N
- En todo momento, durante el juego, se muestran los nombres de ambos jugadores en la ventana.
- Se muestra los puntajes en vivo de ambos jugadores en la ventana.
- Al finalizar la partida se da la opción para jugar nuevamente

No se reportan anomalías.

4.2 Manual de operación.

1. Descargue todos los archivos de la vieja N-dimensional en una misma carpeta.
2. Si no tiene Python 3, descárguelo.
3. Abra la terminal de su computadora.
4. Ubique el archivo en su terminal.
5. Abra main2.py con Python 3.
6. En la ventana que se abrirá introduzca los nombres de los jugadores, y el N.
7. Listo, disfrute del juego.
8. Al finalizar el juego, seleccione ´´SI´´ o ´´NO´´ de acuerdo a si desea o no seguir jugando.

CONCLUSIONES

5.1 Resultados obtenidos.

Se logró el cometido de realizar un juego basado en La Tradicional Vieja de 1 tablero de 3*3, pero adaptado a N tableros de N*N casillas. La parte lógica funciona correctamente y sin imprevistos, en esta se dividió el juego en varias funciones acorde a lo demandado en el enunciado. En cuanto a la parte gráfica, se muestra el tablero de N*N, con las opción de cambiar a cualquiera de los otros N-1 tableros, se reflejan los nombre y puntaje de cada participante.

5.2 Experiencias adquiridas.

Podemos destacar principalmente tres experiencias adquiridas con este proyecto.

Primeramente, se mejoró nuestra destreza en el uso de la librería Pygame. Previo a este proyecto nuestro conocimiento de dicha librería era básicamente nulo, y a lo largo de este proyecto fuimos mejorando, hasta alcanzar resultados satisfactorios en el proyecto.

En el mismo orden de ideas, podemos destacar que se hizo uso del análisis descendente, para dividir un problema grande en varios sub-problemas, esto por medio de clases, y funciones, que luego fueron unidas.

Por último, la resiliencia, durante el desarrollo de este proyecto, se presentaron diversos contratiempos por las condiciones de la universidad u del país, que nos ponían en aprietos como equipo, sin embargo, siempre tuvimos la meta de sacar adelante este trabajo y aprender de las dificultades.

5.3 Dificultades presentadas.

Durante la realización de este proyecto se presentaron algunos inconvenientes logísticos.

Hubo bastantes apagones que entorpecían la labor desarrollar el juego, de igual manera la falta de internet fue un contratiempo, a pesar que en las instalaciones de la Universidad Simón Bolívar, cuenta con este servicio, llegar a ella es sumamente complicado por su nulo servicio de transporte. Estos problemas se superaron gracias a una constante comunicación entre el equipo.

5.4 Recomendaciones.

Nuestras recomendaciones pueden estar enfocadas principalmente en dos partes, recomendaciones para algún desarrollador que desee continuar el juego y recomendaciones al usuario.

Para algún desarrollador, sugeriríamos la creación de un modo nocturno, que es algo muy común hoy día para todo tipo de aplicaciones, crear un sistema de torneos para más de dos jugadores, y también se podría agregar una versión contrareloj y un algoritmo que compita con el jugador.

Para el usuario se recomienda tener instalado Python 3, y la librería pygame en su computador para poder correr el programa, y para el consumo del mismo tener un mouse, pues este es más cómodo que un touchpad para el juego.

BIBLIOGRAFIA

- ✓ PyGame: PyGame Documentation. Consultado en mayo del 2019. Disponible en línea en: https://pygame.org/docs/

- ✓ Stack Overflow: How to create a text input box with pygame? (2017). Consultado en mayo del 2019. Disponible en línea en: https://stackoverflow.com/questions/46390231/how-to-create-a-text-input-box-with-pygame

- ✓ YouTube: Tic-tac-toe multiplayer in pygame (2019). Consultado en mayo del 2019. Disponible en línea en : https://www.youtube.com/playlist?list=PL1P11yPQAo7pJT26yr1_cmfS1g_RX7b4d