Juego de los palillos (NIM) - Timbiriche

Gabino Ruiz Ramirez
Benemérita Universidad autónoma de Puebla
FCC
202056050
gabino.ruizr@alumno.buap.mx

Jasmine Pérez Sánchez
Benemérita Universidad autónoma de Puebla
FCC
202073218
jasmine.perez@alumno.buap.mx

Jesús Huerta Aguilar
Benemérita Universidad autónoma de Puebla
FCC
202041509
jesus.huertaag@alumno.buap.mx

RESUMEN

Este ensayo académico analiza el desarrollo de dos programas interactivos en Python que implementan juegos clásicos utilizando la biblioteca Tkinter. El primero es el juego de Timbiriche, donde dos jugadores se turnan para dibujar líneas entre puntos, intentando completar cuadros para ganar puntos. El segundo es el juego de los Palillos, un reto estratégico donde los jugadores deben retirar palillos sin quedarse con el último. Ambos códigos presentan oportunidades para estudiar cómo se gestionan interfaces gráficas y se implementan las reglas de juegos mediante programación estructurada en Python..

PALABRAS CLAVE

Python, Juegos interactivos, Tkinter, Programación gráfica, Lógica de juegos, Timbiriche, Juego de los palillos.

ABSTRACT

This academic essay analyzes the development of two interactive programs in Python that implement classic games using the Tkinter library. The first is the game of Timbiriche, where two players take turns drawing lines between dots, attempting to complete squares to earn points. The second is the game of Sticks, a strategic challenge where players must remove sticks without being left with the last one. Both codes provide opportunities to study how graphical interfaces are managed and how game rules are implemented through structured programming in Python.

KEYWORDS

Python, Interactive games, Tkinter, Graphical programming, Game logic, Timbiriche, Sticks game.

INTRODUCCIÓN

La programación de juegos es una excelente manera de estudiar conceptos fundamentales de la informática, como la interacción usuario-máquina, el manejo de eventos y la lógica de control.

Este ensayo se enfoca en dos implementaciones de juegos clásicos, Timbiriche y el Juego de los Palillos, desarrollados en Python utilizando la biblioteca Tkinter. A lo largo del ensayo, se analizará cómo estos juegos interactivos gestionan las entradas de los usuarios, las reglas del juego y los mecanismos de puntaje, proporcionando un estudio de caso práctico para la enseñanza de la programación de interfaces gráficas y la lógica de juegos.

METODOLOGIA

El desarrollo de los dos juegos se llevó a cabo utilizando Python y la biblioteca Tkinter para la creación de interfaces gráficas. Tkinter fue elegida por su facilidad de uso y su capacidad para crear ventanas interactivas con elementos gráficos. Para el juego de **Timbiriche**, se diseñó una cuadrícula de puntos donde los jugadores pueden trazar líneas entre los mismos. Se implementaron eventos de clic para permitir que los jugadores seleccionen los puntos inicial y final de una línea.

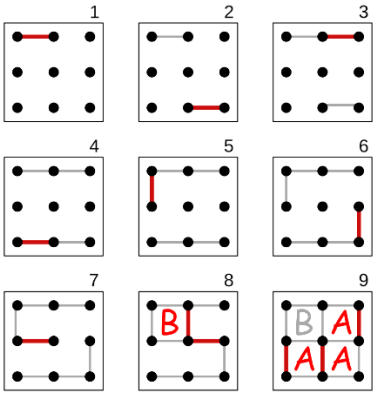


Ilustración 1: Tablero

El sistema de puntaje se construyó de manera que, cuando un jugador completa un cuadrado, se le asignan puntos y puede trazar otra línea en su turno.

Por otro lado, el **Juego de los Palillos** sigue una estructura similar, pero su lógica está orientada a la manipulación de una cantidad finita de palillos. El juego comienza con el jugador introduciendo el número de palillos totales y la cantidad máxima que se puede retirar en cada turno.

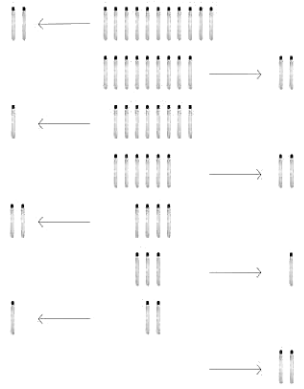


Ilustración 2: Juego de los palillos gameplay

Un ciclo de eventos maneja los turnos alternados entre el jugador y la computadora, validando que las jugadas sean correctas, es decir, que los jugadores no retiren más palillos de los permitidos. Si en algún punto quedan menos palillos de los que un jugador desea retirar, se le indica que debe elegir un número válido.

DESCRIPCION DEL CODIGO: JUEGO DE LOS PALILLOS (NIM)

`jugar_palillos`: Esta función maneja el turno del jugador. Verifica que el número ingresado por el jugador sea válido (dentro del rango permitido), actualiza la cantidad de palillos restantes y, si el jugador gana (es decir, no quedan palillos), muestra un mensaje de victoria y cierra el juego. Si no, le cede el turno a la computadora llamando a la función `tomar_programa`.

```
def jugar_palillos():
    global X, N, turno, label_palillos, entry_palillos
    try:
        tomar = int(entry_palillos.get())

        if tomar < 1 or tomar > N or tomar > X:
            messagebox.showwarning("Movimiento inválido", f"Debes
tomar entre 1 y {min(N, X)} palillos.")
            return
    except ValueError:
        messagebox.showwarning("Entrada inválida", "Debes ingresar
un número.")
        return

    X -= tomar

    label_palillos.config(text=f"Palillos restantes: {X}")

    if X == 0:
        messagebox.showinfo("¡Felicidades!", "¡Has ganado!")
        root.quit()
        return
    turno = 0
    tomar_programa()
```

`tomar_programa`: Controla el turno de la computadora. La función utiliza una estrategia basada en las propiedades matemáticas del juego para decidir cuántos palillos tomar. Actualiza el número de palillos restantes y determina si la computadora ha ganado. Si el juego sigue, devuelve el turno al jugador.

```
def tomar_programa():
    global X, N, turno, label_palillos

    if X % (N + 1) == 0:
        tomar = random.randint(1, N)
    else:
        tomar = X % (N + 1)

    messagebox.showinfo("Turno del programa", f"El programa toma
{tomar} palillos.")
    X -= tomar
    label_palillos.config(text=f"Palillos restantes: {X}")

    if X == 0:
        messagebox.showinfo("Fin del juego", "El programa gana.")
        root.quit()
    else:
        turno = 1
```

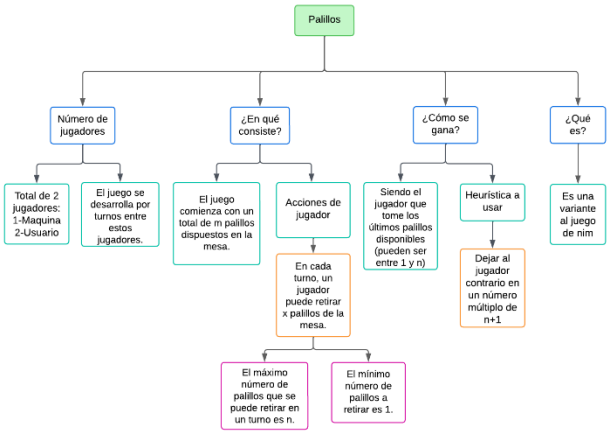
`iniciar_juego`: Esta función configura el inicio del juego. Recibe los valores iniciales, como la cantidad total de palillos y el máximo que se puede tomar por turno. Luego, construye la interfaz gráfica del juego, eliminando cualquier componente anterior y mostrando los controles para que el jugador ingrese su jugada. También inicia el juego con el turno del jugador.

```
def iniciar_juego():
    global X, N, turno, entry_palillos, label_palillos, root

    try:
        X = int(entry_palillos_inicial.get())
        N = int(entry_max_palillos.get())
    except ValueError:
        messagebox.showerror("Entrada inválida", "Introduce valores
válidos para los palillos.")
        return

    if X <= 0 or N <= 0:
        messagebox.showerror("Entrada inválida", "Los valores deben
ser mayores que cero.")

        return

    for widget in root.winfo_children():
        widget.destroy()
    label_palillos = tk.Label(root, text=f"Palillos restantes:
{X}", font=("Arial", 18))

    label_palillos.pack(pady=10)

    tk.Label(root, text=f"¿Cuántos palillos tomas (1 a {N})?",
font=("Arial", 14)).pack(pady=5)

    entry_palillos = tk.Entry(root, font=("Arial", 14))
    entry_palillos.pack(pady=5)

    tk.Button(root, text="Tomar palillos", command=jugar_palillos,
font=("Arial", 14)).pack(pady=10)

    turno = 1
```

`root.mainloop`: Es el bucle principal de `tkinter` que mantiene activa la ventana del juego. Sin esto, la interfaz gráfica no funcionaría, ya que es lo que permite al programa responder a las acciones del usuario (como hacer clic en botones o ingresar texto) hasta que el juego termine.

```
tk.Label(root, text="Introduce la cantidad de palillos inicial:",
font=("Arial", 14)).pack(pady=5)
entry_palillos_inicial = tk.Entry(root, font=("Arial", 14))
entry_palillos_inicial.pack(pady=5)

tk.Label(root, text="Introduce la cantidad máxima de palillos por
turno:", font=("Arial", 14)).pack(pady=5)
entry_max_palillos = tk.Entry(root, font=("Arial", 14))
entry_max_palillos.pack(pady=5)

tk.Button(root, text="Iniciar juego", command=iniciar_juego,
font=("Arial", 14)).pack(pady=20)

root.mainloop()
```

MAPA CONCEPTUAL: JUEGO DE LOS PALILLOS (NIM)



DESCRIPCION DEL CODIGO: TIMBIRICHE

`draw_grid`: Dibuja una cuadrícula de puntos en el área de juego. Cada punto se representa como un pequeño círculo negro.

```
    def draw_grid(self):

        for i in range(self.size):
            for j in range(self.size):
                x = (i + 0.5) * self.cell_size
                y = (j + 0.5) * self.cell_size
                self.canvas.create_oval(x-5, y-5, x+5, y+5,
fill="black")
```

`click_event:` Maneja los clics del usuario para trazar líneas. Verifica si el clic es válido y si corresponde a su turno. Registra el primer clic en un punto y espera el segundo clic para trazar una línea.

```
def click_event(self, event):
        if self.current_player == self.player_choice:
            x, y = event.x, event.y
            clicked_point = self.get_closest_point(x, y)

            if clicked_point:
                if self.first_click is None:
                    self.first_click = clicked_point

                else:
                    second_click = clicked_point

                    if self.is_valid_line(self.first_click,
second_click):
                        line = (self.first_click, second_click)

                        if line not in self.lines and

self.reverse_line(line) not in self.lines:

                            self.lines.add(line)
                            self.animate_line(line)
                            completed_squares =
self.check_square(line)

                            if not completed_squares:
                                self.current_player = 3 -
self.current_player
                            else:
                                self.update_score()

                            self.first_click = None

                            if self.current_player !=
self.player_choice:
                                self.root.after(1000,
self.computer_turn)
                    else:
                        self.first_click = None
```

`is_valid_line:` Verifica si la línea entre dos puntos es válida (debe ser adyacente horizontal o verticalmente, no diagonal).

```
    def is_valid_line(self, point1, point2):

        x1, y1 = point1
        x2, y2 = point2

        if (x1 == x2 and abs(y1 - y2) == self.cell_size) or (y1 ==
y2 and abs(x1 - x2) == self.cell_size):
            return True
        return False
```

`get_closest_point`: Devuelve el punto más cercano a las coordenadas donde se hizo clic, si está dentro de los límites del tablero.

```
    def get_closest_point(self, x, y):

        for i in range(self.size):

            for j in range(self.size):
                px, py = (i + 0.5) * self.cell_size, (j + 0.5) *
self.cell_size
                if abs(x - px) < 10 and abs(y - py) < 10:
                    return (px, py)
        return None
```

`reverse_line` :Invierte el orden de una línea, útil para verificar si una línea ya fue trazada (ya que las líneas son bidireccionales).

```
    def reverse_line(self, line):
        return (line[1], line[0])
```

`animate_line`: Anima el dibujo de una línea entre dos puntos seleccionados. La línea se dibuja paso a paso con el color del jugador actual.

```
    def animate_line(self, line):
        (x1, y1), (x2, y2) = line
        steps = 10

        for i in range(steps + 1):

            x = x1 + (x2 - x1) * i / steps
            y = y1 + (y2 - y1) * i / steps

            self.canvas.create_line(x1, y1, x, y,
fill=self.player_colors[self.current_player], width=2)
            self.root.update()
            time.sleep(0.02)
```

`computer_turn`: Simula el turno de la computadora. Clasifica las líneas en tres categorías:

- Completar cuadros.
- Seguras.
- Peligrosas.

La computadora elige líneas de manera estratégica y las traza.

```
    def computer_turn(self):
        available_lines = self.get_available_lines()

        if not available_lines:
            print("No hay más líneas disponibles. El juego ha
terminado.")
            return

        lines_to_complete_square = []
        safe_lines = []
        dangerous_lines = []

        for line in available_lines:

            squares_for_line = self.possible_squares(line)
            is_dangerous = False
            completes_square = False

            for square in squares_for_line:

                filled_lines = sum(1 for l in square if l in
self.lines or self.reverse_line(l) in self.lines)

                if filled_lines == 3:
                    completes_square = True
                    break
                elif filled_lines == 2:
                    is_dangerous = True

            if completes_square:

                lines_to_complete_square.append(line)

            elif is_dangerous:

                dangerous_lines.append(line)

            else:
                safe_lines.append(line)

        if lines_to_complete_square:

            line = random.choice(lines_to_complete_square)

        elif safe_lines:

            line = random.choice(safe_lines)

        elif dangerous_lines:

            line = random.choice(dangerous_lines)
        else:
            print("No hay más líneas disponibles para la
computadora.")
            return

        self.lines.add(line)

        self.animate_line(line)

        completed_squares = self.check_square(line)

        if not completed_squares:
            self.current_player = 3 - self.current_player
        else:
            self.update_score()

        if self.current_player != self.player_choice:
            self.root.after(1000, self.computer_turn)
```

`get_available_lines`: Obtiene todas las líneas que aún no han sido trazadas en el tablero.

```
    def get_available_lines(self):
        available_lines = []

        for i in range(self.grid_size):
            for j in range(self.grid_size):
                line_horizontal = (
                    ((i + 0.5) * self.cell_size, (j + 0.5) *
self.cell_size),
                    ((i + 1.5) * self.cell_size, (j + 0.5) *
self.cell_size)
                )
                if line_horizontal not in self.lines and
self.reverse_line(line_horizontal) not in self.lines:
                    available_lines.append(line_horizontal)

                line_vertical = (
                    ((i + 0.5) * self.cell_size, (j + 0.5) *
self.cell_size),
                    ((i + 0.5) * self.cell_size, (j + 1.5) *
self.cell_size)
                )
                if line_vertical not in self.lines and
self.reverse_line(line_vertical) not in self.lines:
                    available_lines.append(line_vertical)

        for i in range(self.grid_size):
            line = (
                ((i + 0.5) * self.cell_size, (self.grid_size + 0.5)
* self.cell_size),
                ((i + 1.5) * self.cell_size, (self.grid_size + 0.5)
* self.cell_size)
            )
            if line not in self.lines and self.reverse_line(line)
not in self.lines:
                available_lines.append(line)

        for j in range(self.grid_size):
            line = (
                ((self.grid_size + 0.5) * self.cell_size, (j + 0.5)
* self.cell_size),
                ((self.grid_size + 0.5) * self.cell_size, (j + 1.5)
* self.cell_size)
            )
            if line not in self.lines and self.reverse_line(line)
not in self.lines:
                available_lines.append(line)

        return available_lines
```

`check_square`: Verifica si una línea recién trazada completa un cuadro. Si lo hace, rellena el cuadro con el color del jugador y actualiza el puntaje.

```
    def check_square(self, line):
        completed_squares = []
        for square in self.possible_squares(line):
            if all(l in self.lines or self.reverse_line(l) in
self.lines for l in square):
                self.squares[square] = self.current_player
                self.fill_square(square)
                completed_squares.append(square)

        return completed_squares
```

`possible_squares`: Obtiene los cuadros posibles que podrían completarse con una línea trazada.

```
    def possible_squares(self, line):

        squares = []
        (x1, y1), (x2, y2) = line

        if x1 == x2 and abs(y1 - y2) == self.cell_size:
            left_square = (
                ((x1 - self.cell_size, y1), (x1, y1)),
                ((x1 - self.cell_size, y2), (x1, y2)),
                ((x1 - self.cell_size, y1), (x1 - self.cell_size,
y2)),
                ((x1, y1), (x1, y2))
            )
            squares.append(left_square)

            right_square = (
                ((x1, y1), (x1 + self.cell_size, y1)),
                ((x1, y2), (x1 + self.cell_size, y2)),
                ((x1, y1), (x1, y2)),
                ((x1 + self.cell_size, y1), (x1 + self.cell_size,
y2))
            )
            squares.append(right_square)

        if y1 == y2 and abs(x1 - x2) == self.cell_size:

            upper_square = (
                ((x1, y1 - self.cell_size), (x1, y1)),
                ((x2, y2 - self.cell_size), (x2, y2)),
                ((x1, y1 - self.cell_size), (x2, y2 -
self.cell_size)),
                ((x1, y1), (x2, y2))
            )
            squares.append(upper_square)

            lower_square = (
                ((x1, y1), (x1, y1 + self.cell_size)),
                ((x2, y2), (x2, y2 + self.cell_size)),
                ((x1, y1 + self.cell_size), (x2, y2 +
self.cell_size)),
                ((x1, y1), (x2, y2))
            )
            squares.append(lower_square)

        return squares
```

`fill_square`: Rellena un cuadro completado con el color del jugador actual.

```
    def fill_square(self, square):

        (p1, p2), (p3, p4), (p5, p6), (p7, p8) = square

        self.canvas.create_rectangle(p1[0], p1[1], p3[0], p3[1],
fill=self.player_colors[self.current_player])
```

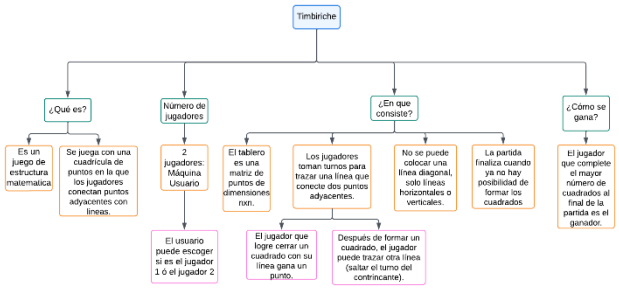
`update_score`: Actualiza el puntaje del jugador tras completar cuadros.

```
    def update_score(self):

        self.score[self.current_player] += 1

        print(f"Jugador {self.current_player} puntaje:
{self.score[self.current_player]}")
```

`check_winner`: Verifica si se han completado todos los cuadros y declara un ganador o un empate si el puntaje es igual.

```
def check_winner(self):
        """Verifica si ya se han completado todos los cuadros para
determinar un ganador"""
        total_squares = self.grid_size * self.grid_size
        if len(self.squares) == total_squares:
            if self.score[1] == self.score[2]:
                print("¡El juego ha terminado en empate!")
            else:
                winner = max(self.score, key=self.score.get)
                print(f"¡El jugador {winner} ha ganado!")
```

MAPA CONCEPTUAL: JUEGO DE LOS PALILLOS (NIM)



RESULTADOS

Los dos juegos resultaron funcionales e interactivos, proporcionando una experiencia fluida para los usuarios. En el **juego de Timbiriche**, los jugadores pudieron trazar líneas y completar cuadrados correctamente, con el sistema de puntaje reflejando adecuadamente el progreso de cada uno.

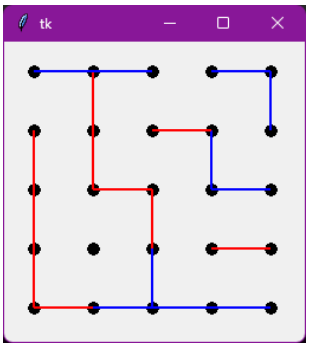


Ilustración 3: Durante la partida

La alternancia de turnos y la capacidad de la computadora para tomar el control cuando es su turno fueron implementadas con éxito. La cuadrícula de puntos se mostró de manera eficiente y los eventos de clic respondieron rápidamente a las acciones del usuario.

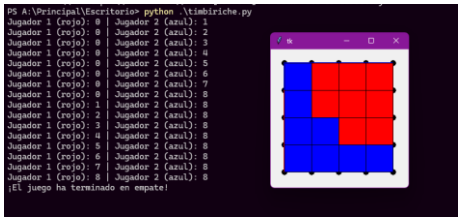


Ilustración 4: Partida finalizada

En el **Juego de los Palillos**, se verificó que la interacción con el usuario para introducir el número de palillos y el máximo permitido por turno se realizara correctamente.

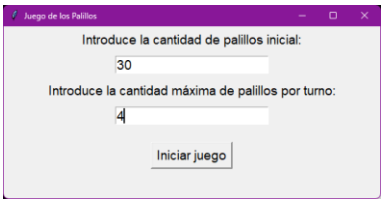


Ilustración 5: Instrucciones iniciales

El ciclo de juego alternó entre el jugador y la computadora sin interrupciones, y las restricciones sobre la cantidad de palillos que podían tomarse en cada turno fueron efectivas.

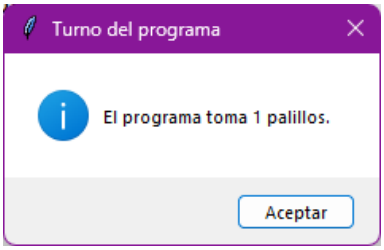


Ilustración 6: Aviso del turno del programa

El juego finaliza adecuadamente cuando un jugador se ve obligado a tomar el último palillo, declarando un ganador claro. En este caso, se logró implementar un nivel básico de inteligencia artificial, haciendo que la computadora tomara decisiones estratégicas basadas en la cantidad de palillos restantes.

Ilustración 7: El programa ha ganado

CONCLUSIONES

El desarrollo de estos dos juegos en Python utilizando Tkinter demuestra la eficacia de esta biblioteca para crear interfaces gráficas interactivas y juegos simples, tanto recreativos como estratégicos. A través de la implementación del **juego de Timbiriche**, se pudo explorar la gestión de eventos gráficos, mientras que en el **Juego de los Palillos**, se hizo evidente la importancia de la lógica y la programación de reglas específicas del juego. Ambos proyectos ilustran cómo se pueden aplicar conceptos de programación orientada a objetos y control de flujo en aplicaciones prácticas, brindando un espacio para el aprendizaje interactivo de la programación.

Además, se destacan las oportunidades de ampliación y mejora de estos juegos, como la posibilidad de incorporar niveles de dificultad o funcionalidades adicionales como partidas en línea. El uso de Python para estos juegos también subraya su flexibilidad como lenguaje de programación tanto para el desarrollo de aplicaciones lúdicas como para el aprendizaje de lógica y control de interfaces. Estos dos proyectos representan ejemplos claros de cómo la informática puede modelar experiencias interactivas mediante un código relativamente sencillo, pero altamente funcional.

AGRADECIMIENTOS

Quiero expresar mi más sincero agradecimiento al profesor Fernando Zacarias Flores, cuya orientación y apoyo han sido fundamentales en el desarrollo de los juegos de Timbiriche y el Juego de los Palillos. Su pasión por la enseñanza y su compromiso con la educación en programación y desarrollo de juegos han sido invaluables para la realización de este proyecto.

El profesor Zacarias Flores ha proporcionado una guía experta en las técnicas de programación necesarias para implementar la lógica y la interacción de los juegos. Su habilidad para desglosar conceptos complejos en componentes manejables ha sido crucial para entender y aplicar efectivamente la lógica detrás de cada juego, facilitando el diseño de interfaces gráficas interactivas y el manejo de eventos en Tkinter.

Además, la retroalimentación constructiva y el interés genuino del profesor por los proyectos han contribuido significativamente a la mejora de ambas implementaciones. Su apoyo ha permitido una comprensión más profunda de los desafíos técnicos, así como la identificación de soluciones efectivas para crear una experiencia de usuario fluida y atractiva.

El profesor Zacarias también ha fomentado un ambiente de creatividad y experimentación, animándonos a explorar nuevas ideas y enfoques en el desarrollo de juegos. Su compromiso con la formación de futuros programadores y su dedicación a la mejora continua han sido una fuente constante de inspiración.
Agradezco profundamente el tiempo que el profesor Zacarias ha invertido en asesorarme y en compartir su vasta experiencia, lo cual

ha sido esencial para el éxito de estos proyectos. Su influencia ha dejado una huella duradera en mi desarrollo académico y profesional, y su guía seguirá siendo valiosa en mis futuros emprendimientos en el campo de la programación y el desarrollo de juegos.

Finalmente, quiero reconocer el esfuerzo y la dedicación de todos los que han colaborado en estos proyectos, así como la comunidad académica que promueve el avance del conocimiento en programación de juegos. La colaboración y el intercambio de ideas han sido esenciales para lograr resultados satisfactorios en este ámbito en constante evolución.

REFERENCIAS

[1] Brownlee, J. (2019). *Deep Learning for Natural Language Processing: Develop Deep Learning Models for your Natural Language Problems*. Machine Learning Mastery

[2] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems* (pp. 8024-8035).

[3] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 38-45).