

MEMORIA DESCRIPTIVA

Sistema Interactivo de Algoritmos de Pathfinding

PRAI: Pathfinding Race AI

INFORMACIÓN DEL PROYECTO

Campo	Descripción
Nombre del Proyecto	PRAI: Pathfinding Race AI
Tecnología Principal	Python 3.13 + Pygame
Repositorio	Juego_Carrera_IA
Tipo de Aplicación	Sistema educativo interactivo
Fecha de Desarrollo	Septiembre 2025

1. DESCRIPCIÓN GENERAL

1.1 Propósito del Sistema

Aplicación educativa que permite **visualizar y comparar** algoritmos de búsqueda de caminos (pathfinding) de manera interactiva. Facilita el aprendizaje de conceptos de inteligencia artificial mediante experiencias prácticas.

1.2 Objetivos Principales

- ✓ Visualización paso a paso de algoritmos de pathfinding
- ✓ Comparación directa entre diferentes algoritmos
- ✓ Experiencia de juego humano vs IA
- ✓ Herramienta de análisis con métricas detalladas

1.3 Estructura General del Proyecto

```
Juego_Carrera_IA/  
├── algorithms/           # Implementaciones de algoritmos  
│   ├── pathfinder_base.py # Clase base común  
│   ├── a_star.py         # Algoritmo A*  
│   ├── dijkstra.py       # Algoritmo de Dijkstra  
│   ├── greedy.py        # Búsqueda voraz  
│   └── uniform_cost.py   # Costo uniforme  
├── components/          # Componentes reutilizables  
│   ├── grid.py          # Sistema de cuadrícula  
│   ├── agent.py         # Entidades móviles  
│   └── button.py        # Interfaz de usuario  
└── scenes/              # Modos de interacción  
    ├── menu_scene.py    # Menú principal  
    └── race_scene.py    # Humano vs IA
```

—	ia_vs_ia_scene.py	# IA vs IA
—	testing_scene.py	# Modo debugging
—	editor_scene.py	# Editor de mapas
—	assets/	# Recursos multimedia
—	fonts/	# Tipografías
—	maps/	# Mapas predefinidos
—	utils/	# Utilidades
—	map_manager.py	# Gestión de mapas
—	config.py	# Configuración global
—	main.py	# Punto de entrada

◆ 2. MODOS DE JUEGO

2.1 Modo Carrera (Humano vs IA)

- **Propósito:** Competir directamente contra algoritmos de IA
- **Control:** Teclas direccionales con movimiento continuo
- **Selección de IA:** A*, Dijkstra, Voraz, Costo Uniforme
- **Configuración:** Movimiento diagonal ON/OFF
- **Objetivo:** Llegar primero al destino

Características:

- **Control humano:** Teclas direccionales con movimiento continuo
- **Selección de IA:** Intercambio entre los 4 algoritmos
- **Configuración dinámica:** Toggle de movimiento diagonal
- **Medición temporal:** Cronometraje preciso de finalización

Mecánicas de Juego:

```
# Sistema de movimiento continuo
self.player_move_timer += dt
if self.player_move_timer >= self.player_move_speed:
    # Procesar movimiento si tecla está presionada
    self.process_continuous_movement()
```

Criterios de Victoria:

1. **Primer criterio:** Llegada al destino
2. **Desempate:** Menor tiempo de ejecución
3. **Información adicional:** Nodos explorados por IA

2.2 Modo Comparación (IA vs IA)

- **Propósito:** Comparar dos algoritmos simultáneamente
- **Visualización:** Dos cuadrículas lado a lado
- **Métricas:** Nodos explorados, iteraciones, longitud de camino

- **Criterio de victoria:** Velocidad y eficiencia
- **Desempate:** Menor cantidad de nodos explorados

Criterios de Evaluación:

```
# Sistema de desempate inteligente
if p1_finished and p2_finished:
    if ai1_nodes_expanded < ai2_nodes_expanded:
        winner = "IA1 (MÁS EFICIENTE)"
    elif ai2_nodes_expanded < ai1_nodes_expanded:
        winner = "IA2 (MÁS EFICIENTE)"
    else:
        winner = "EMPATE PERFECTO"
```

2.3 Modo Testing (Análisis Detallado)

- **Propósito:** Visualizar algoritmo paso a paso
- **Controles:** Avanzar/retroceder pasos individualmente
- **Modo automático:** Ejecución continua con velocidad ajustable
- **Información:** Valores g, h, f en cada nodo
- **Historial:** Navegación completa por todos los pasos

Características:

- **Ejecución paso a paso:** Control granular del progreso
- **Historial navegable:** Avanzar/retroceder en la ejecución
- **Modo automático:** Visualización continua con velocidad ajustable
- **Visualización de costos:** Valores g, h, f en cada nodo

Sistema de Historial:

```
# Captura de estados para navegación
def get_current_state_snapshot(self):
    return {
        "open_list": copy.deepcopy(self.pathfinder.open_list),
        "closed_list": copy.deepcopy(self.pathfinder.closed_list),
        "path": copy.deepcopy(self.pathfinder.path),
        "is_finished": self.pathfinder.is_finished
    }
```

2.4 Editor de Mapas

- **Herramientas:** Obstáculos, inicio, destino, borrador
 - **Funcionalidad:** Arrastrar para pintar áreas
 - **Guardado:** Mapas personalizados en formato JSON
 - **Carga:** Selección de mapas predefinidos o creados
-

3. ALGORITMOS IMPLEMENTADOS

3.1 Tipos de Algoritmos

A (A-Star)*

- **Tipo:** Búsqueda informada óptima
- **Características:**
 - o Utiliza función $f(n) = g(n) + h(n)$
 - o Heurística admisible garantiza optimalidad
 - o Eficiente en memoria y tiempo
- **Complejidad:** $O(b^d)$ en el peor caso
- **Uso recomendado:** Cuando se requiere camino óptimo

Dijkstra

- **Tipo:** Búsqueda de camino más corto
- **Características:**
 - o Solo utiliza costo real $g(n)$
 - o Garantiza camino óptimo sin heurística
 - o Explora uniformemente en todas direcciones
- **Complejidad:** $O((V + E) \log V)$
- **Uso recomendado:** Grafos con pesos negativos o sin heurística confiable

Búsqueda Voraz (Greedy)

- **Tipo:** Búsqueda informada no-óptima
- **Características:**
 - o Solo utiliza heurística $h(n)$
 - o Rápido pero no garantiza optimalidad
 - o Puede quedar atrapado en mínimos locales
- **Complejidad:** $O(b^m)$ en el peor caso
- **Uso recomendado:** Cuando velocidad es prioritaria sobre optimalidad

Costo Uniforme

- **Tipo:** Búsqueda ciega por costo
- **Características:**
 - o Expande nodos por costo acumulado
 - o Garantiza optimalidad sin heurística
 - o Similar a Dijkstra pero para árboles

- **Complejidad:** $O(b^{(C^*/\epsilon)})$
- **Uso recomendado:** Espacios de búsqueda con costos variables

3.2 Sistema de Heurísticas

¿Cómo se calculan las heurísticas?

Heurística Manhattan (4-direccional):

```
def manhattan_distance(pos1, pos2):
    dx = abs(pos1[0] - pos2[0])
    dy = abs(pos1[1] - pos2[1])
    return dx + dy
```

- **Propiedades:** Admisible, consistente
- **Uso:** Movimiento ortogonal únicamente

Heurística Diagonal (8-direccional):

```
def diagonal_distance(pos1, pos2):
    dx = abs(pos1[0] - pos2[0])
    dy = abs(pos1[1] - pos2[1])
    return max(dx, dy) + (math.sqrt(2) - 1) * min(dx, dy)
```

- **Propiedades:** Admisible, más precisa para movimiento diagonal
- **Justificación matemática:** Combina movimientos diagonales ($\sqrt{2}$) y ortogonales (1)

4. GESTIÓN DE MAPAS

4.1 Creación de Mapas

- **Editor interactivo:** Dibujar con mouse
- **Herramientas disponibles:** Obstáculos, punto inicio, punto destino, borrador
- **Funcionalidad drag-to-paint:** Arrastrar para pintar áreas grandes
- **Validación automática:** Verificar que existan inicio y destino

4.2 Carga y Guardado

- **Formato:** Archivos JSON en carpeta `assets/maps/`
- **Estructura:**

```
{
  "width": 20,
  "height": 15,
  "start": [1, 1],
  "end": [18, 13],
  "obstacles": [[5, 5], [6, 5], [7, 5]]
}
```




- **Mapas incluidos:** default_map.json, custom_map_1.json, etc.
- **Selección:** Menú desplegable en selector de mapas

5. VISUALIZACIÓN DE ÁRBOLES DE BÚSQUEDA

5.1 ¿Qué muestran los árboles?

- **Nodos explorados:** Cada casilla visitada por el algoritmo
- **Orden de exploración:** Secuencia temporal de la búsqueda
- **Conexiones padre-hijo:** Cómo se construye el camino
- **Valores de función:** g, h, f en cada nodo (modo Testing)

5.2 Interpretación visual

-  **Azul claro:** Nodos en lista abierta (candidatos)
-  **Rojo:** Nodos en lista cerrada (ya procesados)
-  **Amarillo:** Camino final encontrado
- **Números:** Valores de las funciones de evaluación

5.3 Diferencias entre algoritmos

- **A*:** Árbol dirigido hacia el objetivo
- **Dijkstra:** Expansión uniforme en círculos
- **Voraz:** Línea directa hacia el objetivo (puede desviarse)
- **Costo Uniforme:** Similar a Dijkstra pero por costo acumulado

6. INSTALACIÓN Y USO

6.1 Requisitos

- **Python:** 3.8 o superior
- **Biblioteca:** pygame (`pip install pygame`)
- **Sistema:** Windows, macOS, o Linux

6.2 Instalación:

1. **Clonar repositorio:**

```
git clone https://github.com/Nicols7A4/Juego_Carrera_IA.git
cd Juego_Carrera_IA
```

2. **Instalar dependencias:**

```
pip install -r requirements.txt
```

6.3 Ejecución

```
python main.py
```

6.4 Controles básicos

- **ESC:** Regresar al menú principal
- **Teclas direccionales:** Mover jugador (modo Carrera)
- **Click botones:** Configurar algoritmos y opciones

7.3 Configuración

Archivo config.py:

```
# Configuraciones principales
SCREEN_WIDTH = 1400
SCREEN_HEIGHT = 800
CELL_SIZE = 40
FPS = 60

# Estados de celda
STATE_EMPTY = 0
STATE_OBSTACLE = 1
STATE_START = 2
STATE_END = 3

# Colores
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
```

8. MANUAL DE USUARIO

8.1 Controles Principales

Acción	Control	Contexto
Menú Principal	ESC	Cualquier escena
Movimiento Jugador	↑ ↓ ← →	Modo Carrera
Avanzar Paso	Click "Siguiente"	Modo Testing
Retroceder Paso	Click "Atrás"	Modo Testing
Auto/Manual	Click "Auto"	Modo Testing
Cambiar Algoritmo	Click "Algoritmo"	Cualquier modo
Toggle Diagonal	Click "Diagonal"	Configuración

8.2 Flujo de Uso Típico







Sesión de Aprendizaje Sugerida:

1. **Comenzar en Modo Testing** para entender un algoritmo

2. **Usar paso a paso** para ver la lógica interna
3. **Cambiar a modo automático** para observar ejecución completa
4. **Comparar algoritmos** en Modo IA vs IA
5. **Probar habilidades** en Modo Carrera contra IA
6. **Crear mapas personalizados** en Editor

8.3 Interpretación de Visualizaciones

Colores en Testing Mode:

-  **Azul claro:** Nodos en lista abierta (por explorar)
-  **Rojo:** Nodos en lista cerrada (ya explorados)
-  **Amarillo:** Camino final encontrado
-  **Negro:** Obstáculos
-  **Verde:** Posición de inicio
-  **Rojo intenso:** Posición objetivo

Información Mostrada:

- **Valor F:** Función de evaluación total
 - **Valor G:** Costo acumulado desde inicio
 - **Valor H:** Heurística al objetivo
 - **Contadores:** Nodos explorados e iteraciones
-

9. APÉNDICES

9.1 Estructura de Datos Utilizadas

Clase Nodo:

```
class Nodo:
    def __init__(self, padre=None, posicion=None):
        self.padre = padre          # Referencia al nodo padre
        self.posicion = posicion    # Coordenadas (x, y)
        self.g = 0                  # Costo desde inicio
        self.h = 0                  # Heurística al objetivo
        self.f = 0                  # Función de evaluación f = g + h
```

Listas de Trabajo:

- **Lista Abierta:** Priority queue de nodos por explorar
- **Lista Cerrada:** Set de nodos ya procesados
- **Camino:** Lista ordenada de posiciones solución

9.2 Complejidad Algorítmica

Algoritmo	Tiempo	Espacio
A*	$O(b^d)$	$O(b^d)$
Dijkstra	$O((V+E)\log V)$	$O(V)$
Voraz	$O(b^m)$	$O(b^m)$
Costo Uniforme	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^{\lceil C^*/\epsilon \rceil})$

Donde:

- **b**: Factor de ramificación
- **d**: Profundidad de solución
- **V**: Número de vértices
- **E**: Número de aristas
- **m**: Profundidad máxima
- **C.***: Costo de solución óptima
- **ε**: Costo mínimo de acción

✓ 10. CONCLUSIONES

10.1 Funcionalidades Principales Logradas

- ✓ **4 algoritmos de pathfinding** completamente funcionales
- ✓ **3 modos de juego** interactivos y educativos
- ✓ **Sistema de heurísticas** matemáticamente correcto
- ✓ **Visualización paso a paso** con historial navegable
- ✓ **Editor de mapas** con funcionalidad completa
- ✓ **Métricas de comparación** detalladas y precisas

10.2 Valor Educativo

El sistema permite **entender visualmente** cómo funcionan los algoritmos de búsqueda:

- **Diferencias entre enfoques**: Heurístico vs no-heurístico
- **Trade-offs**: Velocidad vs optimalidad
- **Estructuras de datos**: Listas abiertas y cerradas en acción
- **Impacto de heurísticas**: Cómo guían la búsqueda

10.3 Logros Técnicos

- **Heurísticas admisibles** que garantizan optimalidad en A*
- **Código modular y extensible** para futuras mejoras

- **Documentación completa** con comentarios educativos
- **Interface intuitiva** que facilita el aprendizaje

Memoria Descriptiva - Versión 1.0

Fecha: Septiembre 2025

Proyecto: Sistema Interactivo de Algoritmos de Pathfinding