Universidad del Valle

ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

PRIMER PROYECTO

INTEGRANTES:

- Cristhian Andrés García - 1765571
- David Alejandro Moreno - 1765509
- Paola Andrea Roa - 1765530

El problema del "viaje de Chihiro" se desarrolló en el lenguaje de programación Python.

ESPECIFICACIONES

COSTOS	
CASILLA VACÍA	1 DE ENERGÍA
MONEDA	2 DE ENERGIA
SIN ROSTRO	2 DE ENERGÍA
PASA POR MONEDA -> SE TOPA UN SIN ROSTRO	DEVUELVE 5 DE ENERGIA, SE QUEDA SIN MONEDAS

Para este laberinto se implementó que los ambientes fueran configurables por medio de un texto plano (.txt). A continuación, le explicare la simbología del ambiente.

SIMBOLO	DESCRIPCIÓN
0	MURO
1	ESPACIO VACIO
2	MONEDA
3	SIN ROSTRO
4	CHIHIRO
5	HAKU

Al momento en el que se vaya a configurar el ambiente deseado, se debe hacer desde la primera línea del archivo. Cada columna se separa con una coma y cada fila se separa con un salto de línea. Ejemplo:

1,0,1,2,3,1,1,1,1,1,1

4,0,0,1,0,0,0,3,0,0,5

1,1,1,1,0,1,1,2,0,1,1

Universidad del Valle

ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

PRIMER PROYECTO

PLANTEAMIENTO DE LA SOLUCIÓN

Para ayudar a Chihiro a encontrar a su amado Haku, se planteó 3 algoritmos de búsqueda, A*, costo uniforme y por amplitud.

Se realizo una clase para el árbol, la cual tiene dos constructores. Uno de esos constructores es para el nodo raíz que carga el archivo .txt y el otro es para los nodos hijos.

A continuación, se tendrá el orden de: nombre archivo, descripción y después imágenes (algunas imágenes con funciones específicas tendrán su descripción debajo de ellas).

FILE TREE.py

```
4   class Tree:
5       BLOQUE = 0
6       ESPACIO_VACIO = 1
7       MONEDA = 2
8       SIN_ROSTRO = 3
9       CHIHIRO = 4
10      HAKU = 5
11
12      def __init__(self, *args):
13          if len(args) == 1:
14              self.movement = "START"
15              self.father = None
16              self.mapa = list()
17              self.x = int()
18              self.y = int()
19              self.depth = 0
20              self.cost = 0
21              self.end = False
22              self.meta_x = -1
23              self.meta_y = -1
24              self.load_file(args[0])
25              self.acumulated_coins = 0
```

Universidad del Valle

ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

PRIMER PROYECTO

```
    else:
        self.movement = args[0]
        self.father = args[1]
        self.mapa = args[2]
        self.x = args[3]
        self.y = args[4]
        self.depth = args[5]
        self.cost = args[6]
        self.end = args[7]
        self.acumulated_coins = args[8]
        self.meta_x = self.father.meta_x
        self.meta_y = self.father.meta_y

    self.children = list()
```

Figura 1. constructor

Descripción: Definición de las variables de los elementos involucrados.
El constructor, ahí se encuentran definidos todos los atributos del objeto.

```
def load_children(self):
    if self.end:
        return
    if self.x > 0 and self.mapa[self.y][self.x - 1] != Tree.BLOQUE:
        self.children.append(self.__load_child("LEFT", -1, 0))

    if self.x < len(self.mapa[0]) - 1 and self.mapa[self.y][self.x + 1] != Tree.BLOQUE:
        self.children.append(self.__load_child("RIGHT", 1, 0))

    if self.y > 0 and self.mapa[self.y - 1][self.x] != Tree.BLOQUE:
        self.children.append(self.__load_child("UP", 0, -1))

    if self.y < len(self.mapa) - 1 and self.mapa[self.y + 1][self.x] != Tree.BLOQUE:
        self.children.append(self.__load_child("DOWN", 0, 1))
```

Figura 2. función load children

Universidad del Valle

ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

PRIMER PROYECTO

Descripción: en esta función, de acuerdo con el mapa que se le haya preestablecido, calcula los posibles movimientos que puede hacer el agente.

```
def __load_child(self, movement, dx, dy):

    x = self.x + dx
    y = self.y + dy
    cost = 0
    end = False
    acumulated_coins = 0
    encontrado = False
    mapa = deepcopy(self.mapa)
    if self.mapa[y][x] == Tree.ESPACIO_VACIO:
        cost += 1
    elif self.mapa[y][x] == Tree.MONEDA:
        cost += 2
        acumulated_coins = 1
        mapa[y][x] = Tree.ESPACIO_VACIO
    elif self.mapa[y][x] == Tree.SIN_ROSTRO:
        cost += 2
        cost -= 5 * self.acumulated_coins
        encontrado = True
        # self.acumulated_coins = 0

        acumulated_coins = 0
    elif self.mapa[y][x] == Tree.HAKU:
        cost += 1
        end = True
```

```
    elif self.mapa[y][x] == Tree.HAKU:
        cost += 1
        end = True

    mapa[self.y][self.x] = Tree.ESPACIO_VACIO
    mapa[y][x] = Tree.CHIHIRO
    depth = self.depth + 1
    cost += self.cost
    if not encontrado:
        acumulated_coins += self.acumulated_coins
    return Tree(movement, self, mapa, x, y, depth, cost, end, acumulated_coins)
```

Figura 3. función load child

Descripción: Esta función crea un hijo dado unos parámetros, y además verifica si es un movimiento posible.

```
def g(self):
    #return abs(self.x - self.meta_x) + abs(self.y - self.meta_y)
    return self.cost
```

Figura 4. función costo acumulado

Descripción: Devuelve el costo acumulado.

```
def h(self):
    # return math.sqrt(math.pow(self.x - self.meta_x, 2) + math.pow(self.y - self.meta_y, 2))
    return abs(self.x - self.meta_x) + abs(self.y - self.meta_y)
```

Figura 5. - función heurística

Descripción: La heurística planteada es la Distancia Manhattan, ya que para laberintos es la mejor opción. Esta función heurística se calcula con la posición actual menos la posición final, tanto en el eje x como en el eje y.

```
def f(self):
    return self.g() + self.h()
```

Figura 6. función f(n)

Definición: Devuelve el total de la suma entre el costo acumulado y la heurística.

```
def calculate_rute(self):
    node = self
    rute = ""
    while node is not None:
        rute = " -> " + node.movement + rute
        node = node.father
    return rute
```

Figura 7. función calcular ruta

Definición: Recibe un nodo y concatena los movimientos realizados.

```
def calculate_nodo(self):
    node = self
    rute = list()
    while node is not None:
        rute = [node] + rute
        node = node.father
    return rute
```

Figura 8. función calcular nodo

Definición: Se concatenan los nodos del árbol en una lista.

```
def load_file(self, ruta: str):
    file = open(ruta)
    content = file.readlines()

    board = str()
    mapa = list()
    heigth = 0
    for line in content:
        if line[0] == '#':
            continue

        board += line
        line = line.replace(",", "")
        temporal_line = list()
        width = 0
```

```
            for char in line:
                if char != "\n":
                    if char == '4':
                        self.x = width
                        self.y = heigth
                    elif char == '5':
                        self.meta_x = width
                        self.meta_y = heigth
                    temporal_line.append(int(char))
                width += 1
            mapa.append(temporal_line)
            heigth += 1
        self.mapa = mapa
```

Figura 9. función load file

Definición: Cargar el archivo .txt donde se encuentra el mapa y lo vuelve una lista.

FILE SEARCH.py

```
def bfs():
    root = Tree("map/map1.txt")
    qeue = list()
    qeue.append(root)

    while qeue:
        node = qeue[0]
        if node.end:
            print("\n\n", node)
            return node
        node.load_children()
        qeue.extend(node.children)

        qeue.remove(node)

    return "No se ha encontrado una solución"
```

Figura 10. Función bfs (por amplitud)

Definición: Esta función es el algoritmo de búsqueda por amplitud, para ese se implemento una cola en la que se van a guardar los nodos del árbol y mientras ese nodo no sea el de llegada, sigue expandiendo sus hijos, los guarda al final de la cola y elimina ese nodo explorado.

```
def a():
    root = Tree("map/map1.txt")
    qeue = PriorityQueue()
    qeue.put(PrioritizedItem(root.f(), root))

    while qeue:
        node = qeue.get().item
        if node.end:
            print("\n\n", node)
            return node

        node.load_children()
        for child in node.children:
            qeue.put(PrioritizedItem(child.f(), child))
```

Figura 11. Función a*

Definición: En esta función se implementa una cola de prioridad donde la prioridad está dada por la variable f (F(n)), siempre se va a expandir el nodo con menor prioridad.

```
def costo_uniforme():
    root = Tree("map/map1.txt")
    queue = PriorityQueue()
    queue.put(PrioritizedItem(root.g(), root))

    while queue:
        node = queue.get().item
        if node.end:
            print("\n\n", node)
            return node

        node.load_children()
        for child in node.children:
            queue.put(PrioritizedItem(child.g(), child))
```

Figura 12. Función costo uniforme

Definición: En esta función se implementa una cola de prioridad donde la prioridad está dada por la variable g (g(n)), siempre se va a expandir el nodo con menor prioridad.

FILE INTERFAZGRAFICA.py

Descripción: Tanto las clases pared, haku, chihiro, no face y coin heredan la clase Sprite del módulo pygame para poder pintar imágenes, en nuestro caso las caras de nuestros personajes, las paredes y las monedas.

```python
class Pared(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.image.load("imagenes/muro - copia.jpg")
        self.rect = self.image.get_rect()

    def dibujar(self, superficie):
        superficie.blit(self.image, self.rect)


class Haku(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.image.load("imagenes/hakuu.png")
        self.rect = self.image.get_rect()

    def dibujar(self, superficie):
        superficie.blit(self.image, self.rect)
```

Figura 13. Clases Pared y Haku

```python
class No_face(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.image.load("imagenes/sincara.png")
        self.rect = self.image.get_rect()

    def dibujar(self, superficie):
        superficie.blit(self.image, self.rect)


class Coin(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.image.load("imagenes/coin.png")
        self.rect = self.image.get_rect()

    def dibujar_moneda(self, superficie):
        superficie.blit(self.image, self.rect)
```

Figura 14. Clases No fase y coin

```python
class Chihiro(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.image.load("imagenes/chihiro.png")
        self.rect = self.image.get_rect()

    def dibujar(self, superficie):
        superficie.blit(self.image, self.rect)
```

Figura 15. Clase Chihiro

Universidad del Valle

ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

PRIMER PROYECTO

```
def viajeChihiro(nodos, nombre):
    nodo = nodos[0]
    mapa = nodo.mapa
    iterador = iter(nodos)
    pared = Pared()
    chihiro = Chihiro()
    noFace = No_face()
    coin = Coin()
    haku = Haku()
    ANCHO = len(mapa[0]) * 80 + 300
    ALTO = len(mapa) * 80
    ventana = pygame.display.set_mode((ANCHO, ALTO))
    pygame.display.set_caption('Viaje de Chihiro: ' + nombre)
    reloj = pygame.time.Clock()
    game_over = False

    pygame.font.init()  # you have to call this at the start,
    # if you want to use this module.
    myfont = pygame.font.SysFont('Comic Sans MS', 30)
```

```
while not game_over:
    reloj.tick(60)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            game_over = True

        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                try:
                    nodo = next(iterador)
                    mapa = nodo.mapa
                except StopIteration:
                    print("Fin")
    # Fondo
    ventana.fill("skyblue")

    draw_string('Costo: ' + str(nodo.g()), len(mapa[0]) * 80, 20, ventana, myfont)
    draw_string('Monedas: ' + str(nodo.acumulated_coins), len(mapa[0]) * 80, 50, ventana, myfont)

    coord_x = 0
    coord_y = 0
```

Universidad del Valle

ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

PRIMER PROYECTO

```
# recorre el mapa y pinta una imagen u otra, dependiendo de que numero obtenga
for fila in mapa:
    for columna in fila:
        if columna == 0:
            pared.rect.x = coord_x
            pared.rect.y = coord_y
            pared.dibujar(ventana)
        elif columna == 4:
            chihiro.rect.x = coord_x
            chihiro.rect.y = coord_y
            chihiro.dibujar(ventana)
        elif columna == 5:
            haku.rect.x = coord_x + 10
            haku.rect.y = coord_y
            haku.dibujar(ventana)
        elif columna == 3:
            noFace.rect.x = coord_x + 20
            noFace.rect.y = coord_y + 10
            noFace.dibujar(ventana)
        elif columna == 2:
            coin.rect.x = coord_x + 10
            coin.rect.y = coord_y + 10
            coin.dibujar_moneda(ventana)
```

```
                elif columna == 2:
                    coin.rect.x = coord_x + 10
                    coin.rect.y = coord_y + 10
                    coin.dibujar_moneda(ventana)

                coord_x += 80
            coord_x = 0
            coord_y += 80
        pygame.display.flip()
    pygame.quit()


def draw_string(string: str, x, y, ventana, font):
    textsurface = font.render(string, False, (255, 255, 255))
    ventana.blit(textsurface, (x, y))
```

Figura 16. Función viaje chihiro

Descripción: Como se puede observar, al principio se instancian las clases anteriores que serán los sprites que se dibujen en la ventana, y se declaran las variables que se van a utilizar para la creación de la interfaz.
En el ciclo while se tienen todas las condiciones necesarias para pintar el mapa en la ventana, recorriendo todo el mapa y pintando el Sprite que corresponda a cada número, también se tienen condiciones para salir de la ventana y para avanzar en el mapa, esto capturando eventos del teclado.
La función draw_string dibuja cadenas de texto en la pantalla.

FILE UTILITIES.py

Descripción: En este archivo se implementa los módulos de threading y time para simular un estado de carga.

```python
import threading
import time


def animated_loading(process):
    t0 = time.process_time()
    while process.is_alive():
        chars = [".", "..", "...", "....", "....."]
        for char in chars:
            if not process.is_alive():
                break
            print('\r' + 'Processing' + char, end="")
            time.sleep(0.01)
    print("\nTime elapsed:", (time.process_time() - t0))


def init(function):
    loading_process = threading.Thread(target=function)
    loading_process.start()
    animated_loading(loading_process)
    loading_process.join()
```

Figura 17. Animated loading

FILE MAIN.py

Descripción: Este es el main, aquí se hará el llamado a los algoritmos de búsqueda y a la parte de la interfaz gráfica.

Universidad del Valle

ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

PRIMER PROYECTO

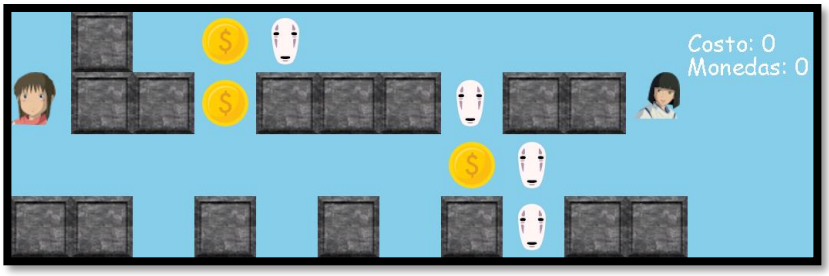
```
import interfazGrafica
import search
import utilities

print("----------------Busqueda por amplitud---------------")
utilities.init(search.bfs)
print("----------------Costo uniforme------------")
utilities.init(search.costo_uniforme)
print("----------------A*----------------------")
utilities.init(search.a)

InterfazGrafica.viajeChihiro(search.bfs().calculate_nodo(), "Busqueda por amplitud")
InterfazGrafica.viajeChihiro(search.costo_uniforme().calculate_nodo(), "Costo uniforme")
InterfazGrafica.viajeChihiro(search.a().calculate_nodo(), "A*")
```

Figura 18. Main

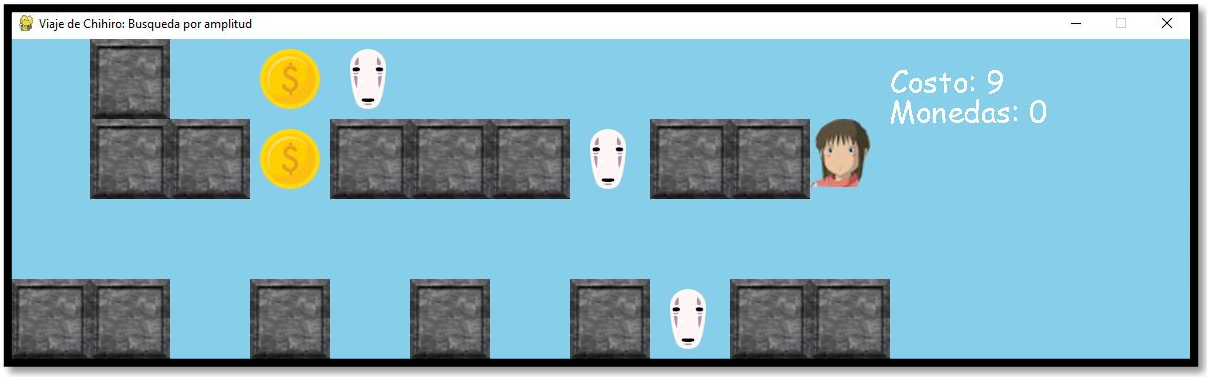
PRUEBAS

Mapa establecido 1

1,0,1,2,3,1,1,1,1,1,1
4,0,0,2,0,0,0,3,0,0,5
1,1,1,1,1,1,1,2,3,1,1
0,0,1,0,1,0,1,0,3,0,0



Búsqueda por amplitud



Universidad del Valle

ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

PRIMER PROYECTO

UP

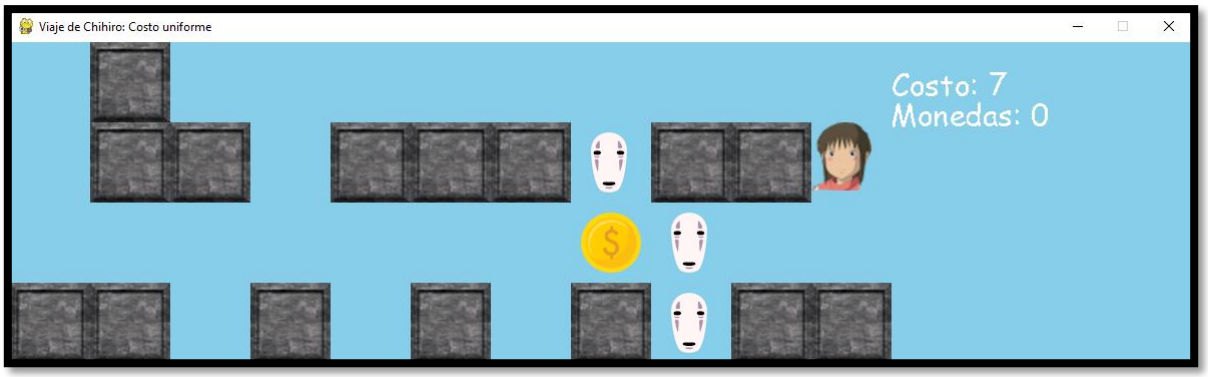
POSITION: 10 1 4

COST: 9

DEPTH: 12

RUTE: -> START -> DOWN -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> UP

Búsqueda por Costo Uniforme



DOWN

POSITION: 10 1 4

COST: 7

DEPTH: 14

RUTE: -> START -> DOWN -> RIGHT -> RIGHT -> RIGHT -> UP -> UP -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> DOWN

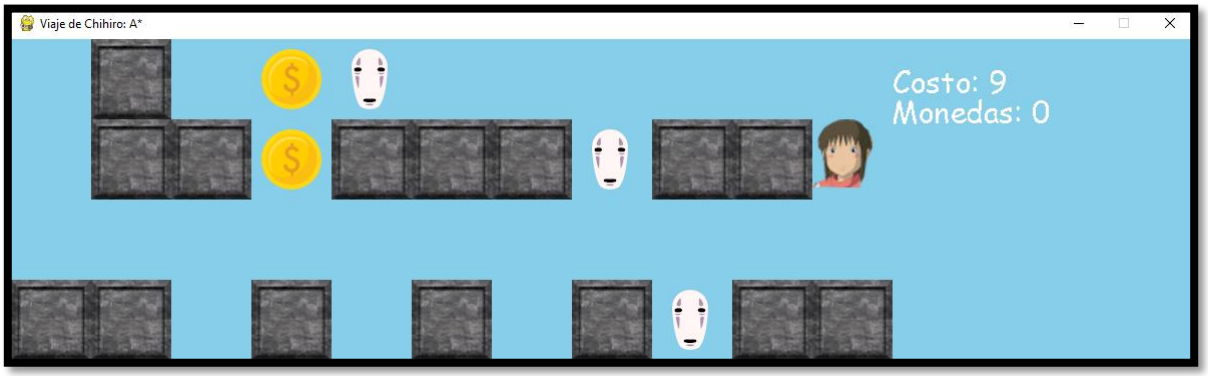
Universidad del Valle

ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

PRIMER PROYECTO

Búsqueda por A*



UP

POSITION: 10 1 4

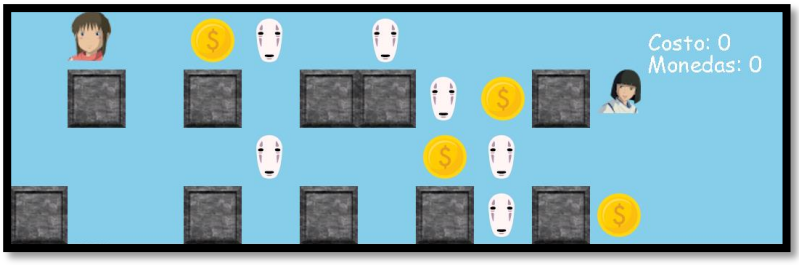
COST: 9

DEPTH: 12

RUTE: -> START -> DOWN -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> UP

Mapa establecido 2

1,4,1,2,3,1,3,1,1,1,1
1,0,1,0,1,0,0,3,2,0,5
1,1,1,1,3,1,1,2,3,1,1
0,1,1,0,1,0,1,0,3,0,2



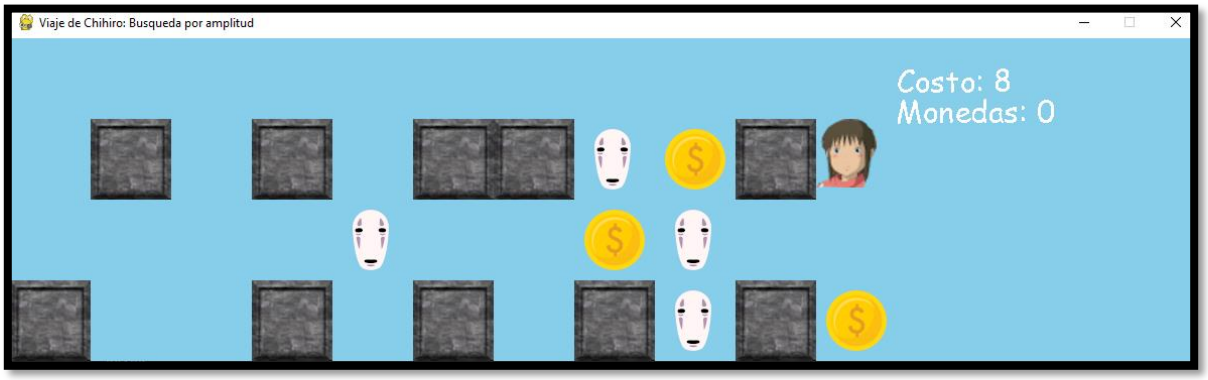
Universidad del Valle

ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

PRIMER PROYECTO

Búsqueda por amplitud



DOWN

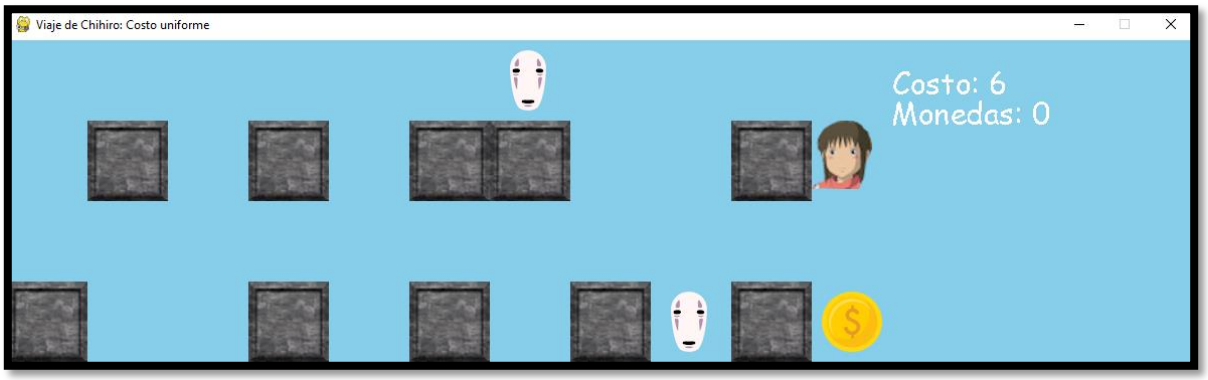
POSITION: 10 1 4

COST: 8

DEPTH: 10

RUTE: -> START -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> DOWN

Búsqueda por Costo uniforme



UP

POSITION: 10 1 4

COST: 6

Universidad del Valle

ESCUELA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

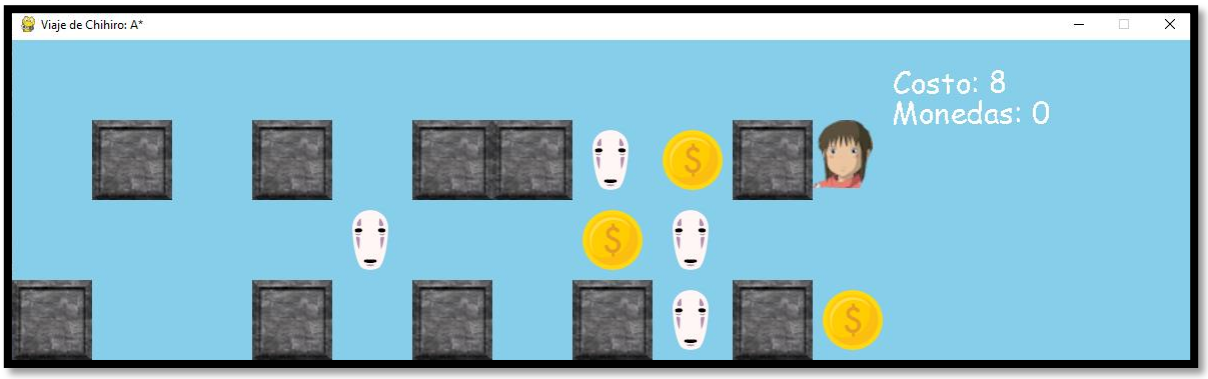
INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

PRIMER PROYECTO

DEPTH: 14

RUTE: -> START -> RIGHT -> RIGHT -> RIGHT -> DOWN -> DOWN -> RIGHT -> RIGHT -> RIGHT -> UP -> RIGHT -> DOWN -> RIGHT -> RIGHT -> UP

Búsqueda por A*



DOWN

POSITION: 10 1 4

COST: 8

DEPTH: 10

RUTE: -> START -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> RIGHT -> DOWN