# 2. Pregunta 2

## 2.1. Actividad 1

Para completar las funciones solicitadas se resolvió de la siguiente manera:

```
def manhattan(x1, y1, x2, y2):
        return abs(x1 - x2) + abs(y1 - y2)

def euclidian(x1, y1, x2, y2):
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

def octile(x1, y1, x2, y2):
    dx = abs(x1 - x2)
    dy = abs(y1 - y2)
    return max(dx, dy) + (math.sqrt(2) - 1) * min(dx, dy)
```

El resultado de las heurísticas lo vemos a continuación con `num_probs = 10`:

| Mapa | Heurística | Expansiones | Nodos Generados | Tiempo de Cómputo (s) | Costo del Camino |
|---|---|---|---|---|---|
| **Barcelona** | Manhattan | 12367 | 12770 | 0.22 | 3361 |
| **Barcelona** | Euclidean | 12807 | 13082 | 0.24 | 3361 |
| **Barcelona** | Octile | 12689 | 13008 | 0.23 | 3361 |
| **Barcelona** | Zero | 14375 | 14436 | 0.29 | 3361 |
| **Maze4** | Manhattan | 35251 | 34753 | 0.48 | 2236 |
| **Maze4** | Euclidean | 38311 | 39045 | 0.46 | 2236 |
| **Maze4** | Octile | 37476 | 38236 | 0.45 | 2236 |
| **Maze4** | Zero | 53704 | 54078 | 0.55 | 2236 |
| **simpleton** | Manhattan | 158 | 289 | 0.08 | 79 |
| **simpleton** | Euclidean | 189 | 313 | 0.08 | 79 |
| **simpleton** | Octile | 176 | 299 | 0.08 | 79 |
| **simpleton** | Zero | 460 | 539 | 0.07 | 79 |
| **starcraft** | Manhattan | 173719 | 167824 | 1.06 | 4456 |
| **starcraft** | Euclidean | 295191 | 299341 | 1.71 | 4456 |
| **starcraft** | Octile | 272988 | 277328 | 1.60 | 4456 |
| **starcraft** | Zero | 927004 | 931163 | 4.57 | 4456 |
| **oficinas_chico** | Manhattan | 3617 | 4248 | 0.17 | 636 |
| **oficinas_chico** | Euclidean | 6628 | 7309 | 0.19 | 636 |
| **oficinas_chico** | Octile | 6056 | 6648 | 0.20 | 636 |
| **oficinas_chico** | Zero | 30594 | 31216 | 0.38 | 636 |

En ese sentido, la mejor heurística es octile ya que es muy eficiente en comparación. Particularmente en mapas más complejos donde el uso de diagonales resulta útil, como en starcraft.

Sin embargo, el rendiemiento de Manhattan es superior para mapas más simple, no obstante el rendimiento de octile en este tipo de mapas sigue manteniendo una buena proporción entre expansiones y tiempo de ejecución, por lo que de manera general octile es la mejor heurística.

## 2.2. Actividad 2

Al realizar la comparación entre A* y Early-A* podemos ver que se obtienen los siguientes resultados. Para esto utilizaremos la `heurística Octile` para todos los casos de evaluación.

| Mapa | Algoritmo | Expansiones | Nodos Generados | Tiempo de Cómputo (s) | Costo del Camino |
|---|---|---|---|---|---|
| **Barcelona** | A* | 12689 | 13008 | 0.25 | 3361 |
| **Barcelona** | Early-A* | 12688 | 13002 | 0.24 | 3342 |
| **Maze4** | A* | 37476 | 38236 | 0.44 | 2236 |
| **Maze4** | Early-A* | 37433 | 38181 | 0.45 | 2194 |
| **simpleton** | A* | 176 | 299 | 0.08 | 79 |
| **simpleton** | Early-A* | 163 | 280 | 0.08 | 54 |
| **starcraft** | A* | 272988 | 277328 | 1.60 | 4456 |
| **starcraft** | Early-A* | 272853 | 277177 | 1.64 | 3941 |
| **oficinas_chico** | A* | 6056 | 6648 | 0.20 | 636 |
| **oficinas_chico** | Early-A* | 6025 | 6634 | 0.20 | 552 |

Podemos ver que el algoritmo `Early-A*` es considerablemente más eficiente que `A*` ya que en la relación de nodos $expansiones/tiempo$ es considerablemente mayor que su competencia `A*`.

A esto se le suma que, a la hora de ver el costo del camino, podemos darnos cuenta que además de tener una mejor proporción entre expansiones y tiempo de ejecución, también llegamos a un costo del camino menor ya que al evaluar previo a la expansión, nos ahorramos posibles desarrollos.

## 2.3. Actividad 3

Ahora realizaremos la comparación entre `Conectividad 4` y `Conectividad 8` tanto con `Early-A*` como con `A*`. Para esto utilizaremos la `heurística Octile` para todos los casos de evaluación.

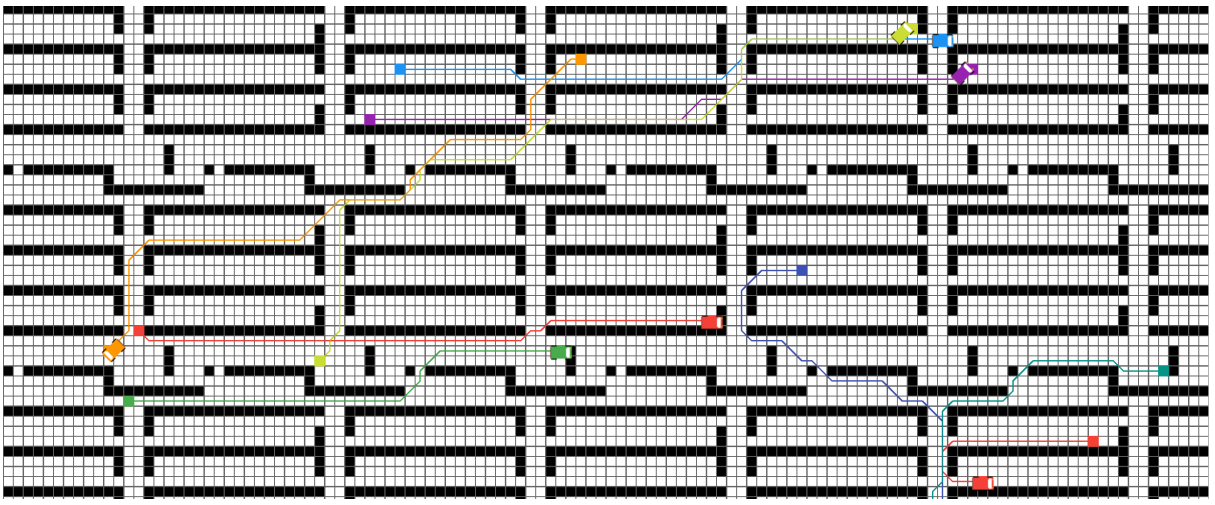
| Mapa | Algoritmo | Conectividad | Expansiones | Tiempo de Cómputo (s) | Costo del Camino |
|---|---|---|---|---|---|
| **Barcelona** | A* | 4 | 12689 | 0.25 | 3361 |
| **Barcelona** | Early-A* | 4 | 12688 | 0.24 | 3342 |
| **Barcelona** | A* | 8 | 15518 | 0.32 | 2791.03 |
| **Barcelona** | Early-A* | 8 | 12370 | 0.27 | 2767.96 |
| **Maze4** | A* | 4 | 37476 | 0.44 | 2236 |
| **Maze4** | Early-A* | 4 | 37433 | 0.45 | 2194 |
| **Maze4** | A* | 8 | 47675 | 0.65 | 1890.97 |
| **Maze4** | Early-A* | 8 | 33805 | 0.55 | 1874.90 |
| **simpleton** | A* | 4 | 176 | 0.08 | 79 |
| **simpleton** | Early-A* | 4 | 163 | 0.08 | 54 |
| **simpleton** | A* | 8 | 79 | 0.08 | 62.01 |
| **simpleton** | Early-A* | 8 | 72 | 0.08 | 52.36 |
| **starcraft** | A* | 4 | 272988 | 1.60 | 4456 |
| **starcraft** | Early-A* | 4 | 272853 | 1.64 | 3941 |
| **starcraft** | A* | 8 | 239953 | 2.15 | 3704.44 |
| **starcraft** | Early-A* | 8 | 165739 | 1.51 | 3590.47 |
| **oficinas_chico** | A* | 4 | 6056 | 0.20 | 636 |
| **oficinas_chico** | Early-A* | 4 | 6025 | 0.20 | 552 |
| **oficinas_chico** | A* | 8 | 5618 | 0.20 | 578.59 |
| **oficinas_chico** | Early-A* | 8 | 4628 | 0.17 | 573.59 |

Vemos que en primer lugar el tiempo de ejecución del algoritmo aumenta en comparación a su homólogo de `Conectividad 4`. Esto en particular se debe a que cada nodo ahora tiene 8 posibles nuevas rutas en lugar de 4, haciendo que el algoritmo tarde más en evaluar las posibilidades.

De todos modos vemos como `Early-A*` colabora en reducir los tiempos, por lo que si usamos `Early-A*` y `Conectividad 8` tenemos una suerte de *trade off* entre los tiempos de ejecución.

Por otra parte, al tener una mayor gama de movimientos, también vemos como una disminución significativa del costo a asociado a la ruta, esto debido a que al tener caminos diagonales con valor $\sqrt{2}$, podemos evitar algo que previamente pudo significar en el mejor de los casos 2 movimientos con un valor total de $1 + 1 = 2$.

En este caso podemos ver como el auto optimiza su recorrido al tomar las diagonales cada vez que puede hacerlo para acortar el costo de su recorrido.



En cuanto a la cantidad de nodos expandidos depende del mapa, ya que si por defecto la ruta óptima fuese una diagonal continua, tendíamos una menor cantidad de expansiones con la nueva conectividad, sin embargo si el caso no se acercara a ese modelo ideal, tenemos que considerar que para cada nodo hijo ahora tenemos el doble de opciones posibles, por lo que es intuitivo pensar que podemos tener una mayor cantidad de expansiones, si fuese el caso en el que el mapa no prioriza un recorrido diagonal.

## 2.4. Actividad 4

Completando el código faltante en el archivo `map.py` vemos que queda de la siguiente manera:

```
elif n == "primitiveStraight":
    '''
    COMPLETAR
    '''
    cost = 1
    initialPose = np.array([0, 1], dtype = int)
    rotations =
    ↪  [(1/4)*np.pi,(2/4)*np.pi,(3/4)*np.pi,(4/4)*np.pi,(5/4)*np.pi,(6/4)*np.pi,(7/4)*np.pi,(8/4)*np.pi]

    numberOldPose = 0 #0 grados
    relativePosition = np.array([[0, 1], [0, 2], [0, 3], [0, 4], [0, 5]])
    initialMove = np.array([0, 5], dtype = int)

    for k in range(len(rotations)):
        neighbor = self.generate_primitive_neighbor(initialPose, numberOldPose, initialMove,
        ↪  relativePosition, rotations, k, cost)
        self.neighbors.append(neighbor)
        self.costs.append(cost)

# Aquí va más código, revisar el archivo para más detalles

elif n == "primitiveDiagonal":
    '''
    COMPLETAR
    '''
    cost = math.sqrt(2)
    initialPose = np.array([0, 1], dtype = int)
    numberOldPose = 0 #0 grados
    rotations =
    ↪  [(1/4)*np.pi,(2/4)*np.pi,(3/4)*np.pi,(4/4)*np.pi,(5/4)*np.pi,(6/4)*np.pi,(7/4)*np.pi,(8/4)*np.pi]

    numberOldPose = 0 #0 grados
    relativePosition = np.array([[0, 1], [1, 2], [2, 3], [3, 4], [4, 5]])
    initialMove = np.array([4, 5], dtype = int)

    for k in range(len(rotations)):
        neighbor = self.generate_primitive_neighbor(initialPose, numberOldPose, initialMove,
        ↪  relativePosition, rotations, k, cost)
        self.neighbors.append(neighbor)
        self.costs.append(cost)
```

De esta manera podemos responder las preguntas planteadas:

¿Cuántos vecinos tiene que revisar successors para cada conjunto?

En particular para las nuevas implementaciones de esta actividad, `primitiveStraight` solo incluye movimientos rectos, es decir que solo puedo invoucrar hasta 4 posibles eventos.

Por otra parte, `primitiveDiagonal` solo involucra movimientos en diagonal, por lo que en este caso nuevamente solo pueden ser 4.

¿Por qué los costos de los movimientos de giro cuestan 2.5π y 4.888?

Como podemos ver en la imagen del enunciado, vemos que la trayectoria de giro involucra un desplazamiento continuo hasta una posición `(abs(5),abs(5))`, en ese sentido tenemos un

valor radio = 5 y un ángulo de $\theta = \frac{\pi}{2}$, es por esto que al calcular $\theta * r = 2{,}5\pi$ obtiendo así el costo asociado.

Por otra parte, el valor de 4.888 se atribuye a que en `primitiveDiagonalAngle` tenemos un ángulo de 45º y en este caso tenemos un radio de distancia hasta involucra un desplazamiento continuo hasta una posición `(abs(2),abs(3))`, es decir $\sqrt{13} = 3{,}6$, esto ahora debemos multipicarlo por el valor anterior de $\frac{\pi}{4} * \sqrt{2}$ lo que nos da un valor aproximado de 4.888.

## 2.5. Actividad 5

Ahora realizaremos la comparación entre `Conectividad 4`, `Conectividad 8` y `State Lattices`. utilizando el algoritmo `Early-A*`. Para esto utilizaremos la `heurística Octile` para todos los casos de evaluación y un `num_probs = 5`

| Mapa | Algoritmo | Conectividad | Expansiones | Tiempo de Cómputo (s) | Costo del Camino |
|---|---|---|---|---|---|
| **Barcelona** | Early-A* | 4 | 12688 | 0.24 | 3342 |
| **Barcelona** | Early-A* | 8 | 12370 | 0.27 | 2767.96 |
| **Barcelona** | Early-A* | Right | 11209 | 0.44 | 325.83 |
| **Barcelona** | Early-A* | Diagonal | 13609 | 0.58 | 229.52 |
| **Barcelona** | Early-A* | Full | 15824 | 0.92 | 229.52 |
| **Maze4** | Early-A* | 4 | 37433 | 0.45 | 2194 |
| **Maze4** | Early-A* | 8 | 33805 | 0.55 | 1874.90 |
| **Maze4** | Early-A* | Right | 37702 | 1.48 | 146 |
| **Maze4** | Early-A* | Diagonal | 74423 | 3.34 | 134.30 |
| **Maze4** | Early-A* | Full | 76864 | 5.02 | 134.30 |
| **simpleton** | Early-A* | 4 | 163 | 0.08 | 54 |
| **simpleton** | Early-A* | 8 | 72 | 0.08 | 52.36 |
| **simpleton** | Early-A* | Right | Error | Error | Error |
| **simpleton** | Early-A* | Diagonal | 290 | 0.07 | 33.84 |
| **simpleton** | Early-A* | Full | 294 | 0.06 | 33.84 |
| **starcraft** | Early-A* | 4 | 272853 | 1.64 | 3941 |
| **starcraft** | Early-A* | 8 | 165739 | 1.51 | 3590.47 |
| **starcraft** | Early-A* | Right | 300915 | 14.70 | 454 |
| **starcraft** | Early-A* | Diagonal | 710913 | 37.79 | 445.14 |
| **starcraft** | Early-A* | Full | 718555 | 60.41 | 445.72 |
| **oficinas_chico** | Early-A* | 4 | 6025 | 0.20 | 552 |
| **oficinas_chico** | Early-A* | 8 | 4628 | 0.17 | 573.59 |
| **oficinas_chico** | Early-A* | Right | Error | Error | Error |
| **oficinas_chico** | Early-A* | Diagonal | 882 | 0.12 | 87.71 |
| **oficinas_chico** | Early-A* | Full | 933 | 0.15 | 88.54 |

Al agregar las conectividades asociadas a `State Lattices` permite que el algoritmo pueda expandirse en más opciones de nodos, aumentando la gama de movimientos. Esto produce evidentemente aumentar en gran cantidad los nodos explorados y permitiendo así obtener soluciones más óptimas pero a un mayor tiempo de ejecución.

## 2.6. Actividad 6

Para evaluar la incidencia de la variable `epsilon` haremos la comparación en un caso controlado el cual será el mapa `Maze4`, con `Early-A*`, `heurística Octile` para todos los casos de evaluación, un `num_probs = 5` y una conectividad `primitiveFull`.

Al probar con el valor de `epsilon = 2` por defecto, vemos que obtenemos los valores de:

- Suma exp: 73653
- Suma gen: 42875
- Suma cost: 133.88
- Runtime total: 4.81 s

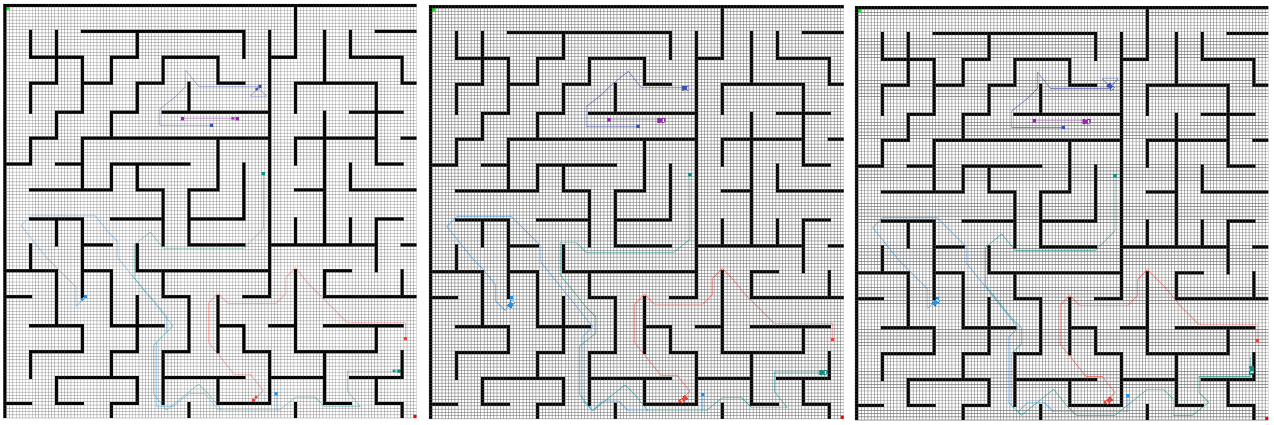
En cambio, probando con `epsilon = 0`, vemos que este otorga el siguiente resultado:

- Suma exp: 82003
- Suma gen: 42247
- Suma cost: 138.65
- Runtime total: 5.50 s

Ahora, probaremos con un valor extremadamente alto de `epsilon = 1000`:

- Suma exp: 69707
- Suma gen: 43007
- Suma cost: 137.30
- Runtime total: 4.63 s

En ese mismo orden, los mapas se ven de la siguiente forma



En cuanto a la eficiencia del código, solo podemos decir que el valor de `epsilon` le da un peso adicional a cierto tiempo de recorridos por lo que ayuda a estrechar las posibilidades y por ende conseguir mejores resultados en cuanto al tiempo de ejecución a mayor valor de `epsilon`.

Visualmente podemos ver que a mayor valor de `epsilon`, menos movimientos en diagonal efectuamos o de cambio de sentido. Esto nos ayuda a determinar que el valor de `epsilon` particularmente castiga los movimientos de cambio de sentido y prioriza mantener un sentido permanente, es por esto que a mayor valor de `epsilon` vemos como el recorrido óptimo es aquel que minimiza los cambios de sentido.