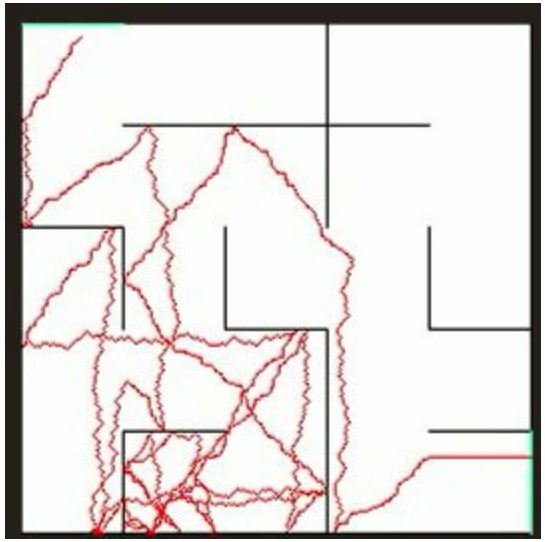
Random Maze Solver

Bellezza Francesco
Bertini Marco

07/09/2023

Esempio



- La particella dovrà trovare la strada per uscire dal labirinto. In verde sono evidenziate le entrate e le uscite del labirinto.
- La particella si muoverà in modo completamente casuale

Il codice sequenziale

```python
class LabyrinthCell:
    def __init__(
        self, wall_north: bool, wall_south: bool, wall_east: bool, wall_west: bool
    ):
        self.wall_north = wall_north
        self.wall_south = wall_south
        self.wall_west = wall_west
        self.wall_east = wall_east

    def set_wall_north(self, wall_north):
        self.wall_north = wall_north

    def set_wall_south(self, wall_south):
        self.wall_south = wall_south

    def set_wall_west(self, wall_west):
        self.wall_west = wall_west

    def set_wall_east(self, wall_east):
        self.wall_east = wall_east

    def has_wall_north(self):
        return self.wall_north

    def has_wall_south(self):
        return self.wall_south

    def has_wall_west(self):
        return self.wall_west

    def has_wall_east(self):
        return self.wall_east
```

```python
class Labyrinth:
    def __init__(
        self, dimension: int
    ):
        self.dimension = dimension
        self.grid = np.empty((dimension, dimension), dtype=LabyrinthCell)
        self.starting_cell = (0,0)
        self.ending_cell = (dimension - 1, dimension - 1)

    def get_grid(self):
        return self.grid

    def get_dimension(self):
        return self.dimension

    def get_ending_cell(self):
        return self.ending_cell
```

- Labirinto visto come una matrice di celle
- Ogni cella può avere o meno un muro a nord, sud, est e ovest. Tale situazione viene rappresentata da un booleano per ciascun punto cardinale.

Il codice sequenziale

```
class LabyrinthDrawSolve:
    def __init__(self, labyrinth: Labyrinth, window_size: int):
        self.labyrinth = labyrinth
        self.window_size = window_size
        self.line_length = math.floor(window_size / labyrinth.get_dimension())
```

- Classe con il ruolo di disegnare il labirinto.
- E' stata utilizzata la libreria opencv per disegnare il labirinto ed il percorso della particella.
- La lunghezza della linea è definita dalla divisione della dimensione della finestra fratto la dimensione del labirinto.
- Nei nostri esempi la dimensione della finestra è fissata a 256x256 pixels.

Il codice sequenziale

```
def find_exit_with_random_particles(self,video, img,color=(0,0,255),lock=None):

    old_x = rand.randrange(self.line_length)
    old_y = 0

    current_grid_position = (0, 0)
    first_point_delimiter = (0, 0)
    second_point_delimiter = (self.line_length, self.line_length)
    path_particle = [(old_x, old_y)]
    path_grid = [(0, 0)]
    x = old_x
    y = old_y
    direction = self.get_random_point_direction()
    allowed_movements = self.get_allowed_movements(direction)
    #DO
    exit_x = (self.labyrinth.dimension*self.line_length)
```

- old_x e old_y contengono le coordinate iniziali della particella. La x viene generata casualmente nell'intervallo da 0 alla lunghezza della linea del singolo riquadro.
- path_particle conterrà tutte le posizioni visitate dalla particella
- path_grid conterrà tutte le celle visitate dalla particella

Il codice sequenziale

```
def find_exit_with_random_particles(self,video, img,color=(0,0,255),lock=None):

    old_x = rand.randrange(self.line_length)
    old_y = 0

    current_grid_position = (0, 0)
    first_point_delimiter = (0, 0)
    second_point_delimiter = (self.line_length, self.line_length)
    path_particle = [(old_x, old_y)]
    path_grid = [(0, 0)]
    x = old_x
    y = old_y
    direction = self.get_random_point_direction()
    allowed_movements = self.get_allowed_movements(direction)
    #DO
    exit_x = (self.labyrinth.dimension*self.line_length)
```

- get_random_point_direction() ritorna un intero compreso tra 0 e 8. Ciascuno di questi numeri rappresenta un diverso punto cardinale.
- get_allowed_movements controlla che la direzione scelta sia effettivamente percorribile.
- exit_x indica la coordinata delle ascisse più grande oltre la quale la particella non può procedere.

Il codice sequenziale

```
while y not in range((self.labyrinth.dimension-1)*self.line_length, self.labyrinth.dimension*self.line_length) or x != exit_x
    if(fnd.FOUND_PATH==True):
        return
    if(rand.randrange(1000)<=1):
        direction = self.get_random_point_direction()
        allowed_movements = self.get_allowed_movements(direction)
    if(current_grid_position==self.labyrinth.ending_cell):
        movement_x, movement_y= (1,0)
    else:
        movement_x, movement_y = rand.choice(allowed_movements)
    x += movement_x
    y += movement_y
    if self.is_particle_movement_allowed(
        current_grid_position,
        (x, y),
        first_point_delimiter,
        second_point_delimiter
    ):
        path_particle.append((x, y))
        (
            match,
            new_cell,
            first_point_delimiter,
            second_point_delimiter,
        ) = self.is_particle_go_to_new_cell(
            current_grid_position,
            (x, y),
            first_point_delimiter,
            second_point_delimiter,
        )
```

Il codice sequenziale

Ciclo while dura fino a quando x non è uguale all'ascissa più esterna e fino a quando la y non si trova nel range dell'ultimo muro

```
while y not in range((self.labyrinth.dimension-1)*self.line_length, self.labyrinth.dimension*self.line_length) or x != exit_x
    if(fnd.FOUND_PATH==True):
        return
    if(rand.randrange(1000)<=1):
        direction = self.get_random_point_direction()
        allowed_movements = self.get_allowed_movements(direction)
    if(current_grid_position==self.labyrinth.ending_cell):
        movement_x, movement_y= (1,0)
    else:
        movement_x, movement_y = rand.choice(allowed_movements)
    x += movement_x
    y += movement_y
    if self.is_particle_movement_allowed(
        current_grid_position,
        (x, y),
        first_point_delimiter,
        second_point_delimiter
    ):
        path_particle.append((x, y))
        (
            match,
            new_cell,
            first_point_delimiter,
            second_point_delimiter,
        ) = self.is_particle_go_to_new_cell(
            current_grid_position,
            (x, y),
            first_point_delimiter,
            second_point_delimiter,
        )
```

Il codice sequenziale

Ciclo while dura fino a quando x non è uguale all'ascissa più esterna e fino a quando la y non si trova nel range dell'ultimo muro

```
while y not in range((self.labyrinth.dimension-1)*self.line_length, self.labyrinth.dimension*self.line_length) or x != exit_x
    if(fnd.FOUND_PATH==True):
        return
    if(rand.randrange(1000)<=1):
        direction = self.get_random_point_direction()
        allowed_movements = self.get_allowed_movements(direction)
    if(current_grid_position==self.labyrinth.ending_cell):
        movement_x, movement_y= (1,0)
    else:
        movement_x, movement_y = rand.choice(allowed_movements)
    x += movement_x
    y += movement_y
    if self.is_particle_movement_allowed(
        current_grid_position,
        (x, y),
        first_point_delimiter,
        second_point_delimiter
    ):
        path_particle.append((x, y))
        (
            match,
            new_cell,
            first_point_delimiter,
            second_point_delimiter,
        ) = self.is_particle_go_to_new_cell(
            current_grid_position,
            (x, y),
            first_point_delimiter,
            second_point_delimiter,
        )
```

Segnala nell'esecuzione del codice parallelo la terminazione di uno dei thread.

Il codice sequenziale

Ciclo while dura fino a quando x non è uguale all'ascissa più esterna e fino a quando la y non si trova nel range dell'ultimo muro

```
while y not in range((self.labyrinth.dimension-1)*self.line_length, self.labyrinth.dimension*self.line_length) or x != exit_x
    if(fnd.FOUND_PATH==True):
        return
    if(rand.randrange(1000)<=1):
        direction = self.get_random_point_direction()
        allowed_movements = self.get_allowed_movements(direction)
    if(current_grid_position==self.labyrinth.ending_cell):
        movement_x, movement_y= (1,0)
    else:
        movement_x, movement_y = rand.choice(allowed_movements)
    x += movement_x
    y += movement_y
    if self.is_particle_movement_allowed(
        current_grid_position,
        (x, y),
        first_point_delimiter,
        second_point_delimiter
    ):
        path_particle.append((x, y))
        (
            match,
            new_cell,
            first_point_delimiter,
            second_point_delimiter,
        ) = self.is_particle_go_to_new_cell(
            current_grid_position,
            (x, y),
            first_point_delimiter,
            second_point_delimiter,
        )
```

Segnala nell'esecuzione del codice parallelo la terminazione di uno dei thread.

Con probabilità bassa si cambia direzione casualmente

Il codice sequenziale

Ciclo while dura fino a quando x non è uguale all'ascissa più esterna e fino a quando la y non si trova nel range dell'ultimo muro

```
while y not in range((self.labyrinth.dimension-1)*self.line_length, self.labyrinth.dimension*self.line_length) or x != exit_x
    if(fnd.FOUND_PATH==True):
        return
    if(rand.randrange(1000)<=1):
        direction = self.get_random_point_direction()
        allowed_movements = self.get_allowed_movements(direction)
    if(current_grid_position==self.labyrinth.ending_cell):
        movement_x, movement_y= (1,0)
    else:
        movement_x, movement_y = rand.choice(allowed_movements)
    x += movement_x
    y += movement_y
    if self.is_particle_movement_allowed(
        current_grid_position,
        (x, y),
        first_point_delimiter,
        second_point_delimiter
    ):
        path_particle.append((x, y))
        (
            match,
            new_cell,
            first_point_delimiter,
            second_point_delimiter,
        ) = self.is_particle_go_to_new_cell(
            current_grid_position,
            (x, y),
            first_point_delimiter,
            second_point_delimiter,
        )
```

Segnala nell'esecuzione del codice parallelo la terminazione di uno dei thread.

Con probabilità bassa si cambia direzione casualmente

La particella prova a spostarsi

Il codice sequenziale

Ciclo while dura fino a quando x non è uguale all'ascissa più esterna e fino a quando la y non si trova nel range dell'ultimo muro

```
while y not in range((self.labyrinth.dimension-1)*self.line_length, self.labyrinth.dimension*self.line_length) or x != exit_x
    if(fnd.FOUND_PATH==True):
        return
    if(rand.randrange(1000)<=1):
        direction = self.get_random_point_direction()
        allowed_movements = self.get_allowed_movements(direction)
    if(current_grid_position==self.labyrinth.ending_cell):
        movement_x, movement_y= (1,0)
    else:
        movement_x, movement_y = rand.choice(allowed_movements)
    x += movement_x
    y += movement_y
    if self.is_particle_movement_allowed(
        current_grid_position,
        (x, y),
        first_point_delimiter,
        second_point_delimiter
    ):
        path_particle.append((x, y))
        (
            match,
            new_cell,
            first_point_delimiter,
            second_point_delimiter,
        ) = self.is_particle_go_to_new_cell(
            current_grid_position,
            (x, y),
            first_point_delimiter,
            second_point_delimiter,
        )
```

Segnala nell'esecuzione del codice parallelo la terminazione di uno dei thread.

Con probabilità bassa si cambia direzione casualmente

La particella prova a spostarsi

Aggiorna il percorso se movimento è consentito

Il codice sequenziale

Ciclo while dura fino a quando x non è uguale all'ascissa più esterna e fino a quando la y non si trova nel range dell'ultimo muro

```
while y not in range((self.labyrinth.dimension-1)*self.line_length, self.labyrinth.dimension*self.line_length) or x != exit_x
    if(fnd.FOUND_PATH==True):
        return
    if(rand.randrange(1000)<=1):
        direction = self.get_random_point_direction()
        allowed_movements = self.get_allowed_movements(direction)
    if(current_grid_position==self.labyrinth.ending_cell):
        movement_x, movement_y= (1,0)
    else:
        movement_x, movement_y = rand.choice(allowed_movements)
    x += movement_x
    y += movement_y
    if self.is_particle_movement_allowed(
        current_grid_position,
        (x, y),
        first_point_delimiter,
        second_point_delimiter
    ):
        path_particle.append((x, y))
        (
            match,
            new_cell,
            first_point_delimiter,
            second_point_delimiter,
        ) = self.is_particle_go_to_new_cell(
            current_grid_position,
            (x, y),
            first_point_delimiter,
            second_point_delimiter,
        )
```

Segnala nell'esecuzione del codice parallelo la terminazione di uno dei thread.

Con probabilità bassa si cambia direzione casualmente

La particella prova a spostarsi

Aggiorna il percorso se movimento è consentito

Controlla se la particella va in una nuova cella ed aggiorna i due delimiter e la posizione della particella nella griglia.

Il codice sequenziale

- I due punti delimiter rappresentano il punto più alto a sinistra e il punto più basso a destra della cella. Servono per comprendere se la particella esce dalla cella corrente.

```
def is_particle_going_north(
    self, tuple_particle_position_new, first_point_delimiter
):
    return tuple_particle_position_new[1] <= first_point_delimiter[1]
```

- Ad esempio la particella va a nord se e solo se la coordinata delle y è inferiore o uguale al delimiter del primo punto.
- Ogni direzione ha la sua condizione specifica.

Il codice sequenziale

```
def is_particle_movement_allowed(
    self,
    current_grid_position,
    tuple_particle_position_new,
    first_point_delimiter,
    second_point_delimiter,
):
    grid = self.labyrinth.get_grid()
    current_cell = grid[current_grid_position[0], current_grid_position[1]]
    if current_cell.has_wall_north() == True and self.is_particle_going_north(
        tuple_particle_position_new, first_point_delimiter
    ):
        return False
```

Il codice sequenziale

```
def is_particle_movement_allowed(
    self,
    current_grid_position,
    tuple_particle_position_new,
    first_point_delimiter,
    second_point_delimiter,
):
    grid = self.labyrinth.get_grid()
    current_cell = grid[current_grid_position[0], current_grid_position[1]]
    if current_cell.has_wall_north() == True and self.is_particle_going_north(
        tuple_particle_position_new, first_point_delimiter
    ):
        return False
```

→ Controllo se è presente un muro a nord e se la particella sta andando a nord… Allora tale movimento non è corretto! (Vale anche per altre direzioni…)

Il codice sequenziale

```
def is_particle_movement_allowed(
    self,
    current_grid_position,
    tuple_particle_position_new,
    first_point_delimiter,
    second_point_delimiter,
):
    grid = self.labyrinth.get_grid()
    current_cell = grid[current_grid_position[0], current_grid_position[1]]
    if current_cell.has_wall_north() == True and self.is_particle_going_north(
        tuple_particle_position_new, first_point_delimiter
    ):
        return False
```

→ Controllo se è presente un muro a nord e se la particella sta andando a nord… Allora tale movimento non è corretto! (Vale anche per altre direzioni…)

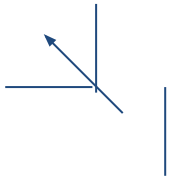
```
return self.diagonal_movement_is_possible(
current_grid_position,
tuple_particle_position_new,
first_point_delimiter,
second_point_delimiter)
```

→ Inoltre devo effettuare un diverso controllo per i muri diagonali (controllare i muri anche dei vicini, non solo della cella corrente)

Il codice sequenziale

```
def diagonal_movement_is_possible(self,current_grid_position,
    tuple_particle_position_new,
    first_point_delimiter,
    second_point_delimiter):
    if self.is_particle_going_north(tuple_particle_position_new,first_point_delimiter) == True and self.is_particle_going_west(tuple_particle_position_new,first_point_delimiter):
        if current_grid_position[0]-1 > 0 and current_grid_position[1]-1 > 0:
            element_grid = self.labyrinth.get_grid()[current_grid_position[0]-1][current_grid_position[1]-1]
            if(element_grid.has_wall_south()==True and element_grid.has_wall_east()==True):
                return False
            else:
                return True
        else:
            return False
```

- Riportiamo l'esempio per una particella che va a nord-ovest.
- Se la posizione della griglia non esce dal labirinto, controlliamo nella eventuale cella di arrivo se sono presenti sia il muro a sud che il muro ad est. Se sì, il movimento non è consentito!



- Senza questo tipo di controllo, tale movimento sarebbe possibile!

Il codice sequenziale

```
if self.is_particle_movement_allowed(
    current_grid_position,
    (x, y),
    first_point_delimiter,
    second_point_delimiter
):
    path_particle.append((x, y))
    (
        match,
        new_cell,
        first_point_delimiter,
        second_point_delimiter,
    ) = self.is_particle_go_to_new_cell(
        current_grid_position,
        (x, y),
        first_point_delimiter,
        second_point_delimiter,
    )
    if match == True:
        path_grid.append(new_cell)
        current_grid_position = new_cell
    allowed_movements = self.get_allowed_movements(direction)
else:
    x -= movement_x
    y -= movement_y
    direction = self.get_random_point_direction()
    allowed_movements = self.get_allowed_movements(direction)
old_x = x
old_y = y
```

Se particella va in nuova posizione, allora aggiorno la posizione nella griglia.

Il codice sequenziale

```
if self.is_particle_movement_allowed(
    current_grid_position,
    (x, y),
    first_point_delimiter,
    second_point_delimiter
):
    path_particle.append((x, y))
    (
        match,
        new_cell,
        first_point_delimiter,
        second_point_delimiter,
    ) = self.is_particle_go_to_new_cell(
        current_grid_position,
        (x, y),
        first_point_delimiter,
        second_point_delimiter,
    )
    if match == True:
        path_grid.append(new_cell)
        current_grid_position = new_cell
    allowed_movements = self.get_allowed_movements(direction)
else:
    x -= movement_x
    y -= movement_y
    direction = self.get_random_point_direction()
    allowed_movements = self.get_allowed_movements(direction)
old_x = x
old_y = y
```

Se particella va in nuova posizione, allora aggiorno la posizione nella griglia.

se movimento non consentito, torno indietro e cambio direzione

Il codice parallelo

- In Python abbiamo deciso di adoperare la libreria JobLib per creare un contesto parallelo. Questo perché Python, a causa del GIL, non consente di parallelizzare, ma solamente di creare un contesto concorrente. Il metodo utilizzato è il solito sia per il codice parallelo, sia per quello sequenziale.

```
if(lock!=None):
    lock.acquire()
    if(fnd.FOUND_PATH==False):
        fnd.FOUND_PATH=True
        self.draw_path_particle_final(path_particle, video, img,color)
    lock.release()
else:
    self.draw_path_particle_final(path_particle, video, img,color)
```

Nel codice sequenziale si passa direttamente all'else clause.

Il codice parallelo

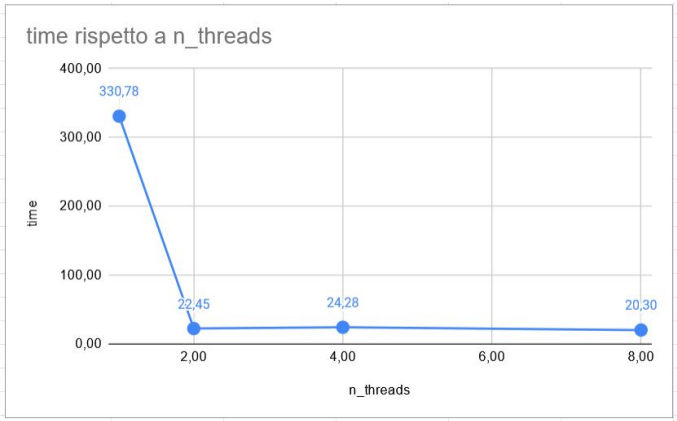
- In Python abbiamo deciso di adoperare la libreria JobLib per creare un contesto parallelo. Questo perché Python, a causa del GIL, non consente di parallelizzare, ma solamente di creare un contesto concorrente. Il metodo utilizzato è il solito sia per il codice parallelo, sia per quello sequenziale.

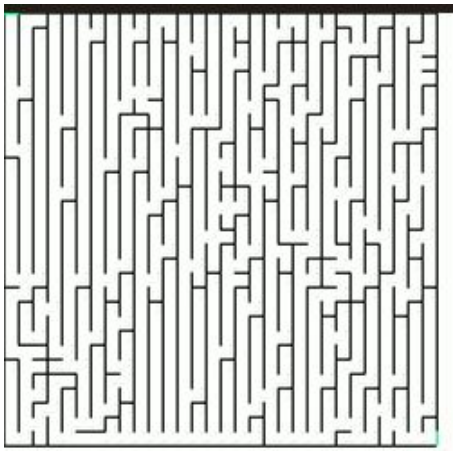
```
if(lock!=None):
    lock.acquire()
    if(fnd.FOUND_PATH==False):
        fnd.FOUND_PATH=True
        self.draw_path_particle_final(path_particle, video, img,color)
    lock.release()
else:
    self.draw_path_particle_final(path_particle, video, img,color)
```

Nel codice sequenziale si passa direttamente all'else clause.

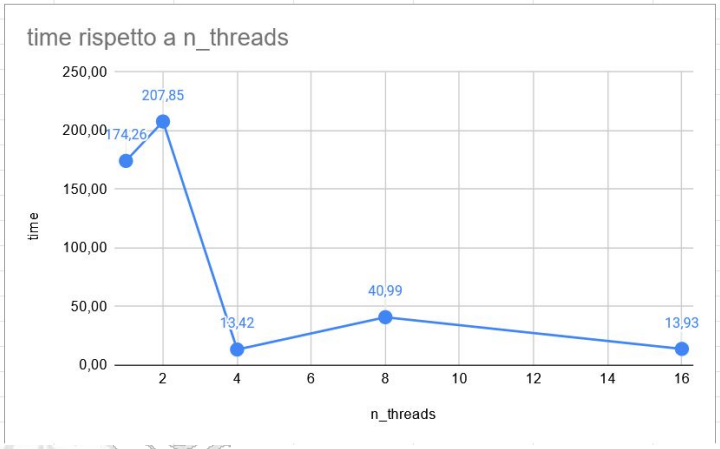
Il thread acquisisce il lock e controlla se il percorso è stato già completato da un altro thread. Se sì, allora non fa niente, altrimenti significa che è lui che ha completato per primo il percorso. Imposta la variabile FOUND_PATH a vero e disegna il percorso inverso (da traguardo a partenza). Infine, rilascia il Lock.

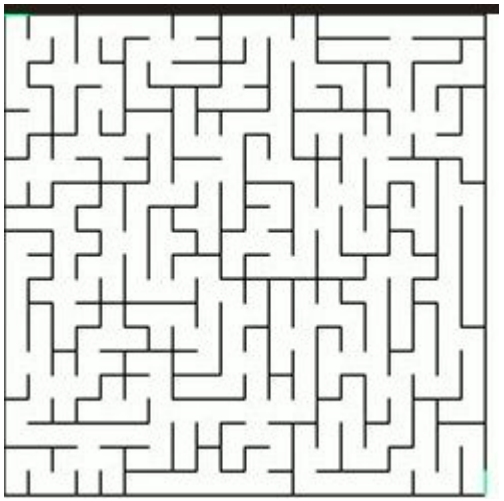
Test effettuati





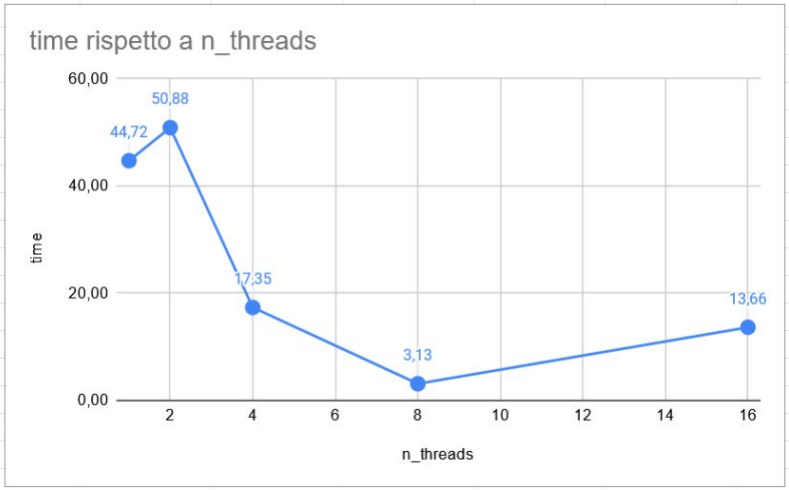
Speedup=14,81

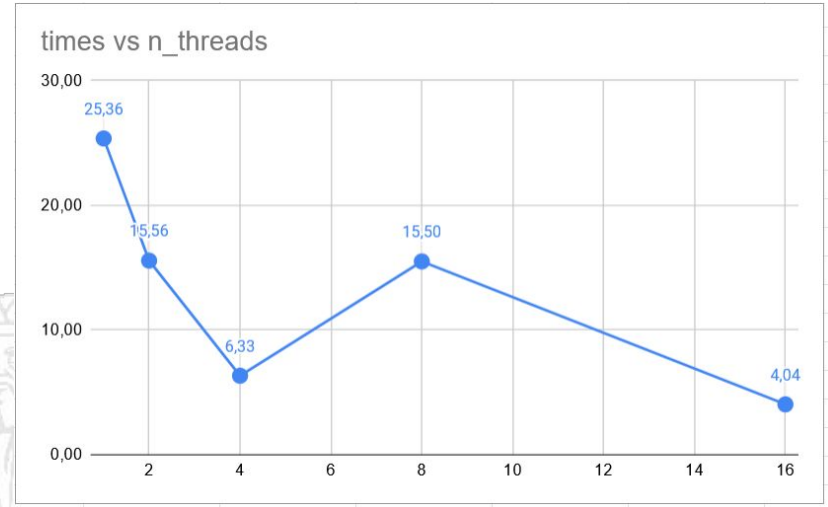


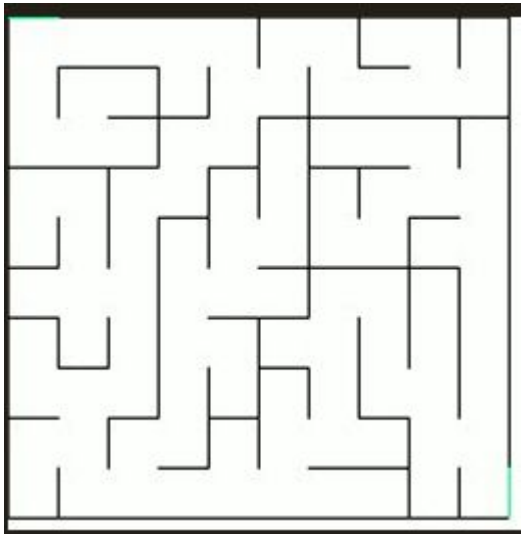


Speedup=2,5

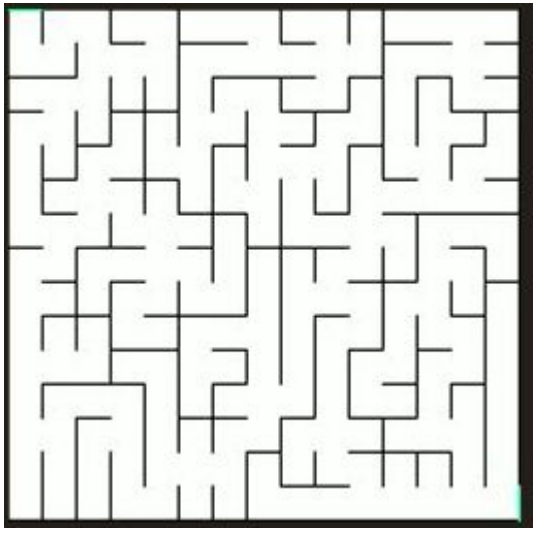
Test effettuati







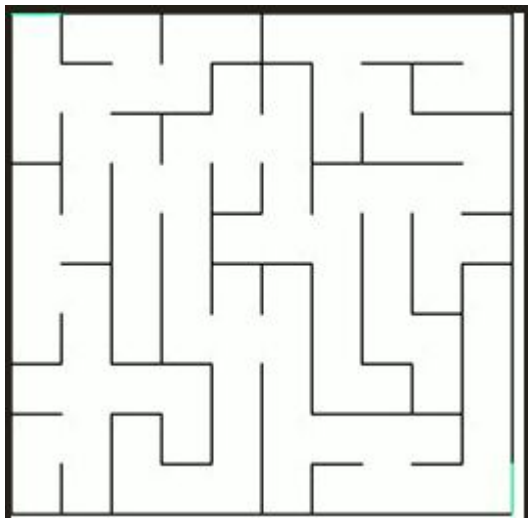
Speedup=2,45

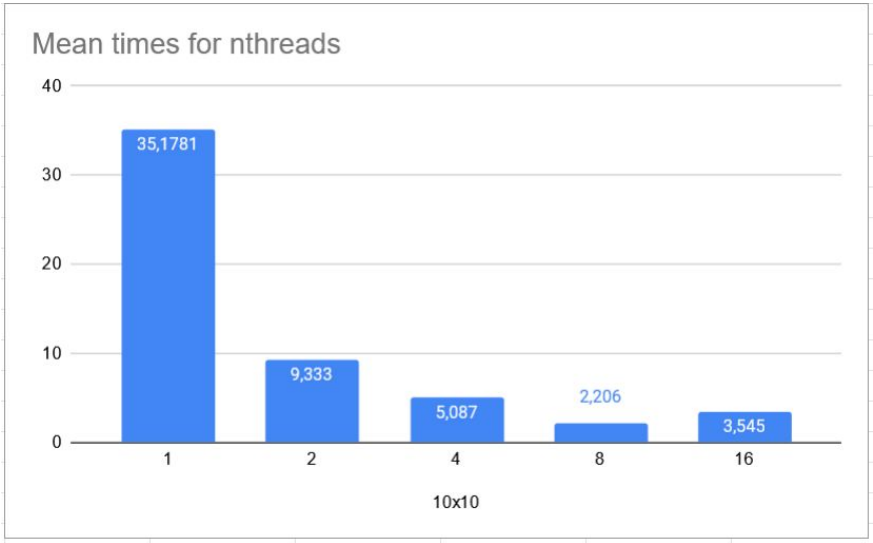


Speedup=2,10

Test effettuati

- Abbiamo provato a fare test ripetuti su un labirinto e fatto una media per ogni tipo di esecuzione (sequenziale, 2 thread…16 thread)

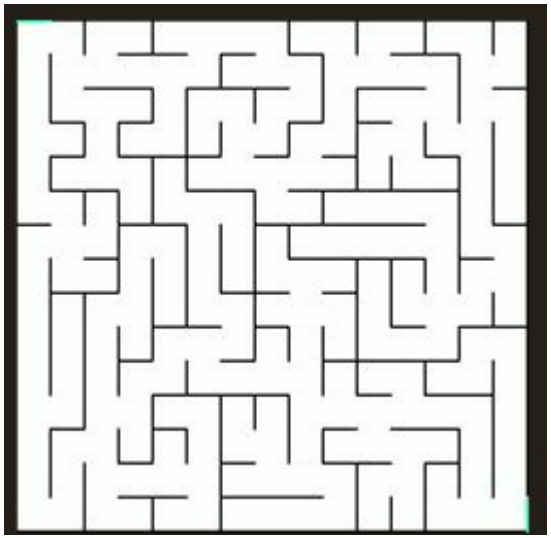


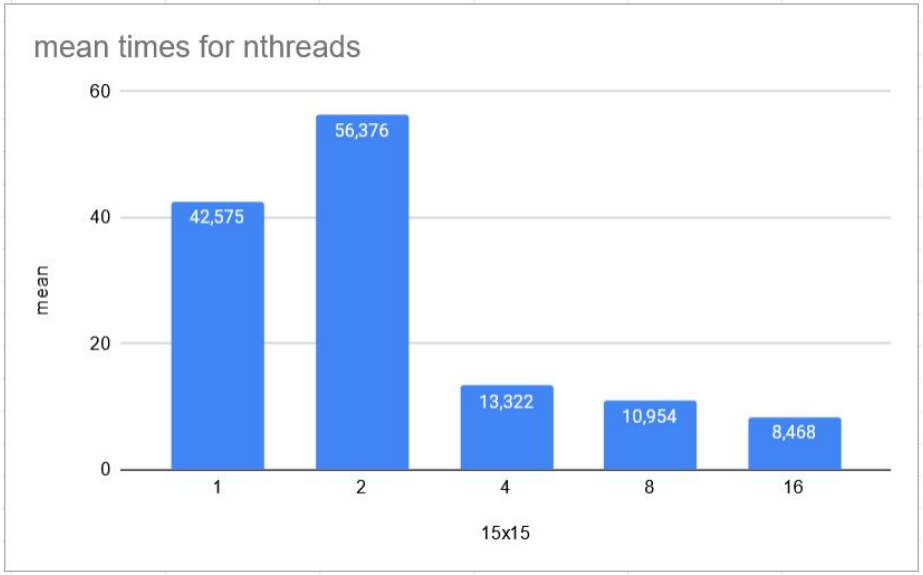


Speedup=6,97

Test effettuati

- Abbiamo provato a fare test ripetuti su un labirinto e fatto una media per ogni tipo di esecuzione (sequenziale, 2 thread…16 thread)

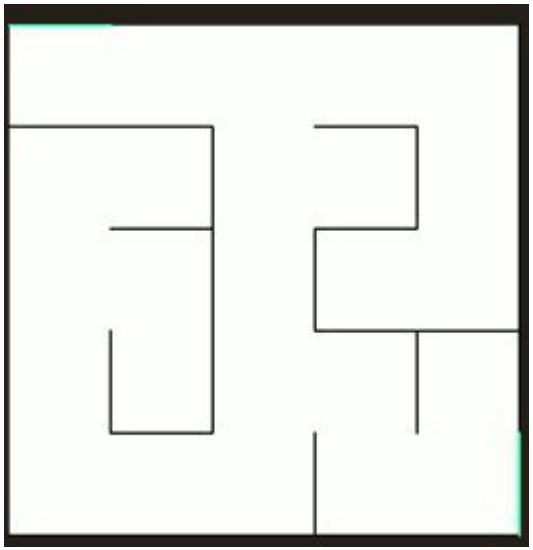


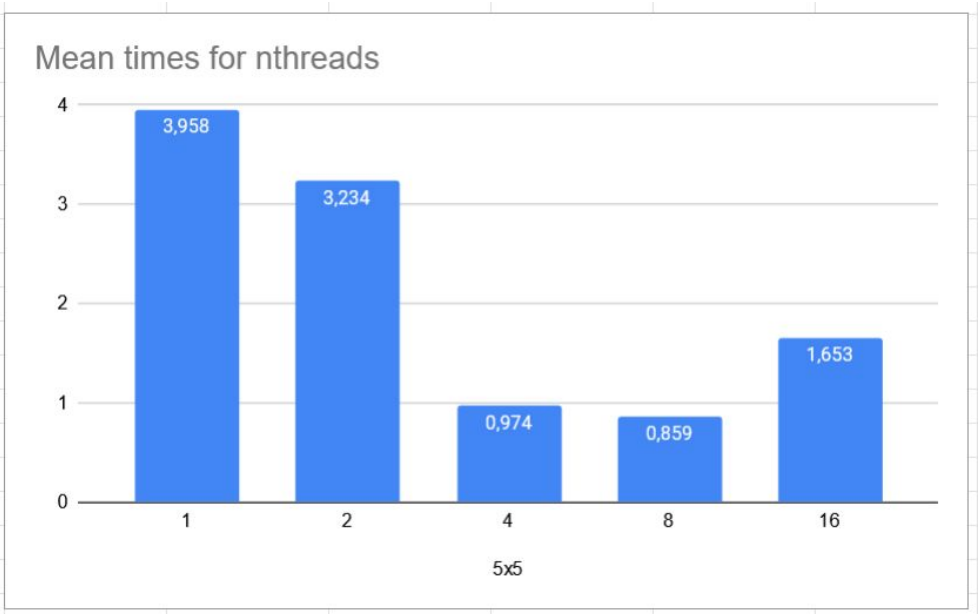


Speedup=1,94

Test effettuati

- Abbiamo provato a fare test ripetuti su un labirinto e fatto una media per ogni tipo di esecuzione (sequenziale, 2 thread…16 thread)





Speedup=2,35

Considerazioni finali

- Lo speedup ottenuto dal processo parallelo non è sempre lo stesso. Questo è dovuto anche al fatto che le varie particelle si muovono in maniera completamente casuale senza tenere conto del percorso che hanno effettuato in precedenza.
- In ogni caso, in linea di massima, nei test che abbiamo svolto, il codice parallelo risulta più veloce rispetto al codice sequenziale.