Annex - Explicació Codi

Índex

En aquest annex es troba tot el codi utilitzat en el treball, comentat línia per línia per tal de comprendre exactament com funciona. A més dels comentaris línia per línia, també hi ha comentaris que expliquen cada funció. Tota la informació necessària per entendre el funcionament del codi es troba en els comentaris esmentats.

1 Model Discret

1.1 Objectes conceptuals del model

1.1.1 Classe passadis

```
import random
import utils_passadis as up
class Passadis:
    # Inicialitzacio de la classe Passadís
    def __init__(self, id, m, n, amplada_entrada = 1, entrada_unica=False, entrades_laterals=False,
    obstacles=False):
        # Identificador unic del passadís per diferenciar-lo
        self.id = id

        # Nombre de files
        self.m = m

        # Nombre de carrils (columnes)
        self.n = n

        # Nombre que determina quantes cel·les ocupa cada acces (entrada/sortida)
        self.amplada_entrada = amplada_entrada

        # True => totes les cel·les de la primera i ultima fila son entrada
        self.entrada_unica = entrada_unica

        # True => hi han entrades tambe a les parets laterals
        self.entrades_laterals = entrades_laterals

        # True => hi ha obstacles al passadís
        self.obstacles = obstacles

        # Si entrada_unica es False aquesta variable determina quants accesos poden crearse
        self.num_entrades = random.randint(m//amplada_entrada, ((2*(n+m))-4)//amplada_entrada)

        # Creacio de la matriu i obtenció de les posicions de entrades, parets i obstacles
        self.passadis, self.entrades, self.parets, self.obstacles = up.crear_passadis(m, n,
        self.amplada_entrada, self.num_entrades, entrada_unica, entrades_laterals, obstacles)

        # Llista dels individus que es troben al passadís
        self.ind_in_passadis = []

        # Diccionari que conté la posicio de cada individu
        self.diccionari_posicio = {}

    # Retorna l'identificador del passadís
    def get_id(self):
        return self.id
```

```
    # Retorna el nombre de files
    def get_m(self):
        return self.m

    # Retorna el nombre de carrils
    def get_n(self):
        return self.n

    # Retorna les entrades
    def get_entrades(self):
        return self.entrades

    # Retorna l'amplada de l'entrada
    def get_amplada_entrada(self):
        return self.amplada_entrada

    # Retorna el nombre d'entrades
    def get_num_entrades(self):
        return self.num_entrades

    # Retorna si l'entrada es unica o no
    def get_entrada_unica(self):
        return self.entrada_unica

    # Retorna les parets
    def get_parets(self):
        return self.parets

    # Retorna els obstacles
    def get_obstacles(self):
        return self.obstacles

    # Retorna la matriu que conte el passadís
    def get_passadis(self):
        return self.passadis

    # Retorna els individus que es troben al passadís
    def get_ind_in_passadis(self):
        return self.ind_in_passadis

    # Retorna el diccionari de posicions
    def get_diccionari_posicio(self):
        return self.diccionari_posicio
```

1.1.2 Classe individu

```
import classe_trajecte as ct

class Individu:
    # Inicialització de la classe Individu
    def __init__(self, id, posicio, sortida, objectiu, velocitat, m, n, camp_visio, ponderacions):
        # Identificador únic per individu
        self.id = id

        # Dupla (x, y) que dona la posicio inicial (entrada) en el passadís
        self.posicio = posicio
```

```
        self.sortida = sortida

        # Dupla(x, y) posició de l'objectiu de l'individu
        self.objectiu  = objectiu

        # Files passadís
        self.m = m

        # Carrils passadís
        self.n = n

        # Velocitat
        self.velocitat = velocitat

        # Direcció respecte l'últim moviment realitzat
        self.direccio = (0,0)

        # Nombre que al elevar el quadrat et dona les cel·les que podrà observar l'individu
        self.camp_visio = camp_visio

        # Conté recorregut, temps recorregut, nombre de col·lisions, temps agrupat
        # i els coeficients d'importància
        self.trajecte = ct.Trajecte([posicio], 0, 0, 0, ponderacions)

        # Puntuació utilitzada en l'algorisme genètic per avaluar el rendiment de l'individu al passadís
        self.aptitud = 0

    # Retorna l'identificador de l'individu
    def get_id(self):
        return self.id

    # Retorna la posició de l'individu
    def get_posicio(self):
        return self.posicio

    # Retorna la sortida de l'individu
    def get_sortida(self):
        return self.sortida

    # Retorna l'objectiu de l'individu
    def get_objectiu(self):
        return self.objectiu

    # Retorna la velocitat de l'individu
    def get_velocitat(self):
         return self.velocitat_maxima

    # Retorna la direcció de l'individu
    def get_direccio(self):
        return self.direccio

    # Retorna el trajecte de l'individu
    def get_trajecte(self):
        return self.trajecte

    # Retorna el camp de visió de l'individu
    def get_camp_visio(self):
```

```
        return self.camp_visio

    # Retorna l'aptitud de l'individu
    def get_aptitud(self):
        return self.aptitud

    # Estableix una nova posició per a l'individu i l'afegeix al seu recorregut
    def set_posicio(self, nova_posicio):
        self.posicio = nova_posicio
        self.trajecte.recorregut.append(nova_posicio)

    # Estableix un nou objectiu per a l'individu
    def set_objectiu(self, nou_objectiu):
        self.objectiu = nou_objectiu

    # Estableix una nova direcció per a l'individu
    def set_direccio(self, nova_direccio):
        self.direccio = nova_direccio

    # Estableix una nova aptitud per a l'individu
    def set_aptitud(self, nova_aptitud):
        self.aptitud = nova_aptitud
```

1.1.3 Classe trajecte

```
class Trajecte:
    def __init__(self, recorregut, n_colisions, t_recorregut, n_agrupat, ponderacions):
        # Llista de posicions fins al moment de l'individu
        self.recorregut = recorregut

        # Quantitat de col·lisions que ha tingut amb altres individus al llarg del recorregut
        self.n_colisions = n_colisions

        # Temps (segons) que ha trigat en fer el seu recorregut (equivalent a la quantitat de moviments)
        self.t_recorregut = t_recorregut

        # Quantitat de moviments que l'individu ha realitzat agrupat amb altres individus
        # amb una direcció similar al llarg del recorregut
        self.n_agrupat = n_agrupat

        # Quantitat de canvis de direcció realitzats durant el recorregut
        self.canvis_direccio = 0

        # Coeficient d'importància per a l'agrupament amb altres individus
        self.pes_agrupat = ponderacions[0]

        # Coeficient d'importància per a les col·lisions
        self.pes_colisions = ponderacions[1]

        # Coeficient d'importància per a la distància recorreguda
        self.pes_distancia = ponderacions[2]

        # Coeficient d'importància per als canvis de direcció
        self.pes_canvis = ponderacions[3]
```

```
# Retorna el recorregut realitzat fins al moment per l'individu
def get_recorregut(self):
    return self.recorregut

# Retorna la quantitat de col·lisions que ha tingut amb altres individus al llarg del recorregut
def get_n_colisions(self):
    return self.n_colisions

# Suma 1 nova col·lisió al sumatori de col·lisions
def add_colisio(self):
    self.n_colisions += 1

# Retorna quants segons (equival a quants moviments) ha trigat en fer el seu recorregut
def get_t_recorregut(self):
    return self.t_recorregut

# Suma 1 al temps trigat en fer el recorregut
def add_t_recorregut(self):
    self.t_recorregut += 1

# Retorna la quantitat de moviments que ha realitzat agrupat de altres individus
def get_n_agrupat(self):
    return self.n_agrupat

# Suma 1 als moviments que l'individu ha fet agrupat amb altres individus durant el recorregut
def add_agrupat(self):
    self.n_agrupat += 1

def get_canvis_direccio(self):
    return self.canvis_direccio

# Suma les distancies en cada canvi de direcció realitzat durant el recorregut
def add_canvi_direccio(self, nova_distancia):
    self.canvis_direccio += nova_distancia

# Retorna els coeficients d'importància
def get_ponderacions(self):
    return self.pes_agrupat, self.pes_colisions, self.pes_distancia, self.pes_canvis
```

1.2 Creació passadís

```python
import numpy as np

def crear_passadis(m, n, a_entrada, num_entrades, entrada_unica, entrades_laterals, obstacles):
    """
    Funció per a crear un passadís.

    Paràmetres:
    m: nombre de files del passadís.
    n: nombre de columnes del passadís.
    a_entrada: amplada de l'entrada.
    num_entrades: nombre d'entrades al passadís.
    entrada_unica: si és True, tota la part superior i inferior del passadís serà entrada.
    entrades_laterals: si és True, hi hauran entrades a les parets laterals del passadís.
    obstacles: si és True, es crearan obstacles al passadís.
    """

    # Creem una matriu de zeros de m files i n columnes
    passadis = np.zeros((m, n))

    # Creem una llista buida per a guardar les posicions de les parets
    parets = []

    # Recorrem cada cel·la de la matriu
    for i in range(m):
        for j in range(n):
            # Si la cel·la està en el perímetre de la matriu
            if i == 0 or i == (m - 1) or j == 0 or j == (n - 1):
                # Si la cel·la està en una cantonada de la matriu, la tractem com a obstacle
                if (i, j) in [(0, 0), (0, n-1), (m-1, 0), (m-1, n-1)]:
                    passadis[i, j] = 4
                else:
                    # Si la cel·la està en el perímetre però no en una cantonada, la tractem com a paret
                    passadis[i, j] = 2
                    parets.append((i, j))

    # Creem les entrades del passadís
    entrades, passadis = crear_entrades(m, n, passadis, parets, a_entrada, num_entrades,
    entrada_unica, entrades_laterals)

    # Si volem obstacles, creem un quadrat en mig del passadís que actuarà com a obstacle
    obs = []
    if obstacles == True:
        obs = [(int(m/2), int(n/2)), (int(m/2-1), int(n/2)), (int(m/2), int(n/2+1)),
        (int(m/2-1), int(n/2+1)), (int(m/2), int(n/2-1)), (int(m/2-1), int(n/2-1))]
        for ob in obs:
            passadis[ob] = 4

    # Retornem el passadís, les entrades, les parets i els obstacles
    return passadis, entrades, parets, obs
```

```python
def crear_entrades(m, n, passadis, parets_original, a_entrada, num_entrades, entrada_unica,
entrades_laterals):
    """
    Funció per a crear les entrades del passadís.

    Paràmetres:
    m: nombre de files del passadís.
    n: nombre de columnes del passadís.
    passadis: matriu que representa el passadís.
    parets_original: llista de tuples que representen les posicions de les parets.
    a_entrada: amplada de l'entrada.
    num_entrades: nombre d'entrades al passadís.
    entrada_unica: si és True, tota la part superior i inferior del passadís serà entrada.
    entrades_laterals: si és True, hi hauran entrades a les parets laterals del passadís.
    """

    # Creem una llista buida per a guardar les entrades
    entrades = []

    # Copiem la llista de parets
    parets = parets_original.copy()

    # Si no volem entrades laterals, eliminem les parets laterals de la llista de parets
    if entrades_laterals == False:
        parets = [p for p in parets if p[0] == 0 or p[0] == (m-1)]

    # Si volem una entrada única, tota la part superior i inferior del passadís serà entrada
    if entrada_unica == True:
        entrades.append([p for p in parets if p[0] == 0])
        entrades.append([p for p in parets if p[0] == (m-1)])
    else:
        # Si no volem una entrada única, creem diverses entrades
        entrada1 = []
        entrada2 = []
        entrada3 = []
        entrada4 = []
        if entrades_laterals == True:
            for i in range(a_entrada):
                entrada1.append((0, int(n/2) + i - 1))
                entrada2.append((m-1, int(n/2) + i - 1))
                entrada3.append((int(m/2) + i - 1, 0))
                entrada4.append((int(m/2) + i - 1, n-1))
        else:
            for i in range(a_entrada):
                entrada1.append((m - 1, int((n-1)/3) + i - 1))
                entrada2.append((m - 1, int((n-1)/3) * 2 + i - 1))
                entrada3.append((0, int((n-1)/3) + i - 1))
                entrada4.append((0, int((n-1)/3) * 2 + i - 1))

        # Afegim les entrades a la llista d'entrades
        entrades.append(entrada1)
        entrades.append(entrada2)
        entrades.append(entrada3)
        entrades.append(entrada4)

    # Retornem les entrades i el passadís
    return entrades, passadis
```

1.3 Funcions auxiliars dels individus

1.3.1 Sortida i objectiu

```python
def calcul_sortida(entrada, passadis):
    """
    Calcula la sortida, que és un conjunt de posicions, a la que anirà l'individu.

    Paràmetres:
    entrada (dupla): Coordenades de l'entrada des de la qual parteix l'individu.
    passadis (Passadis): L'objecte passadís que conté la configuració del passadís.

    Retorna:
    tuple: Coordenades de la sortida cap a la qual s'ha de moure l'individu.
    """
    entrades = passadis.get_entrades()
    if entrada[0] == 0 or entrada[0] == (passadis.get_m()-1):
        # Excloure les sortides que estan a la mateixa fila que l'entrada
        posibles_sortides = [sortida for sortida in entrades if sortida[0][0] != entrada[0]]
    else:
        #  Excloure les sortides que estan a la mateixa columna que l'entrada
        posibles_sortides = [sortida for sortida in entrades if sortida[0][1] != entrada[1]]
    return random.choice(posibles_sortides)


def calcul_objectiu(posicio, sortida):
    """
    Calcula l'objectiu a partir de les posicions possibles de la sortida.

    Paràmetres:
    posicio (dupla): Coordenades de la posició actual de l'individu.
    sortida (llista): Llista de tuples que contenen les posicions possibles de la sortida.

    Retorna:
    dupla: Coordenades de la posició exacta de la sortida cap al qual s'ha de moure l'individu (objectiu)
    """
    distancies = {}
    for pos in sortida:
        # Distància de Manhatan entre la posició actual i cada posició possible de la sortida
        distancies[np.sum(np.abs(np.array(pos) - np.array(posicio)))] = pos

    # Obtenció de la posició amb la menor distància a la posició actual (objectiu)
    objectiu = distancies[min([key for key in distancies.keys()])]
    return objectiu
```

1.3.2 Direcció i canvi de direcció

```python
def calcul_direccio(seguent_pos, pos):
    """
    Calcula la direcció en la que es mourà l'individu.

    Paràmetres:
    seguent_pos (dupla): Coordenades de la posició següent a la que s'ha de moure l'individu
    pos (dupla): Coordenades de la posició actual de l'individu.

    Retorna:
    dupla: Coordenades de la direcció en la que es mourà l'individu.
    """

    # Calcula la diferencia en el eje x entre la posición siguiente y la posición actual
    direccio_x = seguent_pos[0] - pos[0]

    # Calacula la diferència en el eix y entre la posició següent i l'actual
    direccio_y = seguent_pos[1] - pos[1]
    return (direccio_x, direccio_y)


def calcul_distancia_direccio(individu, pos):
    """
    Calcula la distància entre les posicions a les que es mouria l'individu a partir de dos
    direccions diferents.

    Paràmetres:
    individu: L'objecte individu que conté la informació de l'individu
    pos (dupla): Coordenades de la posició a la que es vol calcular la distancia.

    Retorna:
    La distància entre les posicions considerant dos direccions diferents.
    """

    # Coordenades de la posició actual de l'individu
    x, y = individu.get_posicio()

    # Components de la direcció de l'individu
    dx, dy = individu.get_direccio()

    # Calcula la distància de Manhattan entre les posicions considerant dues direccions diferents
    distancia = np.sum(np.abs(np.array((x+dx, y+dy)) - np.array(pos)))
    return distancia
```

1.3.3 Camp de visió

```
def calcul_camp_visio(individu, direccio):
    """
    Calcula el camp de visió de l'individu en funció de la direcció donada.

    Paràmetres:
    individu (objecte): L'objecte individu que conté la informació de l'individu.
    direccio (dupla): La direcció en què l'individu es mourà.

    Retorna:
    llista: Una llista de posicions que formen el camp de visió de l'individu.
    """
    # Obtenim les coordenades x i y de la posició actual de l'individu
    x, y = individu.get_posicio()

    # Obtenim les coordenades dx i dy de la direcció
    dx, dy = direccio

    # Calculem el signe de dx i de dy
    if dx != 0: signe_dx = int(dx / abs(dx))
    else: signe_dx = 0

    if dy != 0: signe_dy = int(dy / abs(dy))
    else: signe_dy = 0

    # Obtenim la mida del camp de visió de l'individu
    n = individu.get_camp_visio()
    # Creem una llista buida per emmagatzemar les posicions del camp de visió
    posicions_visio = []

    # Comprovem si la direcció no és zero en x i en y
    if dx != 0 and dy != 0:
        # Bucle per generar les posicions del camp de visió en funció de n
        for i in range(n):
            for j in range(n):
                # Afegim la posició (x + signe_dx * i + dx, y + signe_dy * j + dy) a la llista de posicio
                posicions_visio.append((x + dx + signe_dx * i, y + dy + signe_dy * j))

    # Comprovem si la direcció no és zero en x
    elif dx != 0:
        # Bucle per generar les posicions del camp de visió en funció de n
        for i in range(1, n + 1):
            for j in range(1 - i, i):
                # Afegim la posició (x + signe_dx * i, y + j) a la llista de posicions del camp de visió
                posicions_visio.append((x + signe_dx * i, y + j))

    # Comprovem si la direcció no és zero en y
    elif dy != 0:
        # Bucle per generar les posicions del camp de visió en funció de n
        for i in range(1, n + 1):
            for j in range(1 - i, i):
                # Afegim la posició (x + j, y + signe_dy * i) a la llista de posicions del camp de visió
                posicions_visio.append((x + j, y + signe_dy * i))

    # Retornem la llista de posicions que formen el camp de visió de l'individu
    return posicions_visio
```

1.3.4 Anàlisi d'interaccions entre individus

```
def consultar_interaccio(ind_a_objectiu, ind_a_direccio, pos, passadis):
    """
    Consulta l'interacció entre dos individus en una determinada posició.

    Paràmetres:
    ind_a_objectiu (dupla): Coordenades de l'objectiu de l'individu A.
    ind_a_direccio (dupla): Direcció de l'individu A.
    pos (dupla): Coordenades de la posició a consultar l'interacció (posició de l'individu B)
    passadis: L'objecte passadís que conté la configuració del passadís.

    Retorna:
    int: 1 si l'interacció implica un moviment en grup, 0 si és una col·lisió, -1 si no hi ha interacció.
    """
    # Obtenim l'individu que hi ha en aquella posició
    ind_b = passadis.diccionario_posicion.get(pos, None)

    if ind_b is not None:
        ind_b_direccio = ind_b.get_direccio()
        ind_b_objectiu = ind_b.get_objectiu()

        if ind_b_direccio[0] != 0 and ind_a_direccio[0] != 0:

            # Si els individus van en la mateixa direcció
            if ind_b_direccio[0] - ind_a_direccio[0] == 0 and ((ind_a_objectiu[0] == ind_b_objectiu[0])
            or (ind_a_objectiu[1] == ind_b_objectiu[1])):
                return 1
            else:
                # Si els individus no van en la mateixa direcció
                return 0

        elif ind_b_direccio[0] == 0 or ind_a_direccio[0] == 0:
            # Si els individus van en la mateixa direcció
            if ind_b_direccio[1] - ind_a_direccio[1] == 0 and ((ind_a_objectiu[0] == ind_b_objectiu[0])
            or (ind_a_objectiu[1] == ind_b_objectiu[1])):
                return 1
            else:
                # Si els individus no van en la mateixa direcció
                return 0

    else:
        # Si no hi ha interacció
        return -1
```

1.4 Moviment dels individus

```
def moure_individu(individu, passadis):
    """
    Mou un individu en el passadís segons una lògica de moviment.

    Paràmetres:
    individu: L'objecte individu que es vol moure.
    passadis: L'objecte passadís que conté la configuració del passadís.

    Retorna: None
    """

    # Acualitzem el temps que porta fent el recorregut l'individu
    #individu.trajecte.add_t_recorregut()
    individu.trajecte.add_t_recorregut()

    # Fem tants moviments com indiqui la velocitat (en aquest model la velocitat només pot ser entera)
    for i in range(individu.get_velocitat()):
        # Obtenim les coordenades x i y de la posició actual de l'individu
        x, y = individu.get_posicio()

        # Calculem l'objectiu de l'individu
        objectiu = calcul_objectiu((x,y), individu.get_sortida())

        # Establim l'objectiu de l'individu
        individu.set_objectiu(objectiu)

        # Obtenim la matriu del passadís
        matriu = passadis.get_passadis()

        # Eliminem la posició anterior de l'individu al diccionari de posicions
        passadis.diccionario_posicion.pop((x, y), None)

        # Comprovem si l'individu ha arribat a l'objectiu
        if (x, y) == objectiu:
            # Establim la posició de l'individu com a None
            individu.set_posicio(None)

            # Eliminem l'individu de la llista d'individus en el passadís
            passadis.ind_in_passadis.remove(individu)
            return

        posicions = []

        # Obtenim les posicions vàlides adjacents a la posició actual en funció de l'objectiu
        if objectiu[0] != 0:
            posicions = [(x, y), (x+1, y), (x, y-1), (x, y+1), (x+1, y+1), (x+1, y-1),
            (x-1, y-1), (x-1, y+1)]
        else:
            posicions = [(x, y), (x-1, y), (x, y-1), (x, y+1), (x-1, y+1), (x-1, y-1),
            (x+1, y+1), (x+1, y-1)]

        # Analitzem les posicions vàlides
        puntuacions = {}
        for pos in posicions:
            # Les posicions vàlides son aquelles on no hi han obstacles o parets
```

```
        if 0 <= pos[0] < passadis.get_m() and 0 <= pos[1] < passadis.get_n() and matriu[pos] != 2
        and matriu[pos] != 4:
            # Si en una de les posicions adjacents hi ha un individu comprovem quina interacció
            # s'esta tenint
            if matriu[pos] == 1:
                # Si aquest individu va en una direcció similar a la del nostre individu
                # considerem que és un moviment agrupat (en carril)
                if consultar_interaccio(objectiu, individu.get_direccio(), pos, passadis) == 1:
                    individu.trajecte.add_agrupat()

                # Si aquest individu va en una direcció contraria a la del nostre individu
                # considerem que és una col·lisió
                elif consultar_interaccio(objectiu, individu.get_direccio(), pos, passadis) == 0:
                    individu.trajecte.add_colisio()

            else:
                # Calculem la distancia a l'objectiu
                distancia_objectiu = np.sum(np.abs(np.array(pos) - np.array(objectiu)))

                # Calculem la quantitat de canvi que fem en la direccio
                canvi_direccio = calcul_distancia_direccio(individu, pos)
                individu.trajecte.add_canvi_direccio(canvi_direccio)

                # Direcció respecte la nova possible posició
                nova_direccio = (pos[0] - x, pos[1] - y)

                # Camp de visió en el sentit de la nova direcció
                camp_visio = calcul_camp_visio(individu, nova_direccio)

                # Analitzem si fer aquest moviment implica agruparnos o aproparnos a col·lisions
                colisions = 0
                inds_agrupats = 0
                for cv_pos in camp_visio:
                    if 0 <= cv_pos[0] < passadis.get_m() and 0 <= cv_pos[1] < passadis.get_n()
                    and matriu[cv_pos] == 1:
                        if consultar_interaccio(objectiu, nova_direccio, cv_pos, passadis) == 1:
                            inds_agrupats += 1
                        else:
                            colisions += 1

                # Obtenim els coeficients d'importància relatius a cada variable
                p_agrupats, p_colisions, p_distancia, p_canvi = individu.trajecte.get_ponderacions()

                # Calculem la puntuació corresponent a aquesta posició
                puntuacio = inds_agrupats * p_agrupats - colisions * p_colisions
                - distancia_objectiu * p_distancia - canvi_direccio * p_canvi

                puntuacions[puntuacio] = pos

    # Si no hi ha posicions vàlides disponibles no movem a l'individu i finalitzem la funció
    if not puntuacions: return

    # Seleccionem la posició amb la puntuació més alta
    posicio_escollida = puntuacions[max([key for key in puntuacions.keys()])]

    # Calculem la direcció de l'individu (és equivalent a calcular quin és el moviment que ha fet)
    individu.set_direccio(calcul_direccio(posicio_escollida, (x,y)))
```

```
# Si la posició escollida coincideix amb l'objectiu, l'individu finalitza el trajecte
# i eliminem la seva posició
if posicio_escollida in passadis.get_entrades() and individu.get_objectiu() == posicio_escollida:
    individu.set_posicio(posicio_escollida)
    individu.set_posicio(None)
    matriu[x, y] = 0

# Si encara no ha arribat a l'objectiu simplement actualitzem la posició
else:
    # Deixem buida la posició on estava abans l'individu
    matriu[x, y] = 0

    # Ocupem la nova posició
    matriu[posicio_escollida] = 1

    # Actualitzem la posició en l'objecte individu
    individu.set_posicio(posicio_escollida)

    # Actualitzem el diccionari de posicions del passadís
    passadis.diccionario_posicion[individu.posicio] = individu
```

1.5 Representació gràfica

```
def dibuixar_passadis(passadis, screen, cell_size, clock, fps):
    """
    Dibuixa el passadís i els individus a la pantalla en un entorn de simulació pygame.

    Paràmentres:
    passadis (Passadis): L'objecte passadís que conté la configuració de la simulació.
    screen: La superfície de la pantalla de pygame en la qual es dibuixarà el passadís.
    cell_size (tuple): La mida de les cel·les del passadís en píxels.
    clock: L'objecte rellotge de pygame per controlar el temps de la simulació.
    fps (int): El nombre de fotogrames per segon desitjat per la simulació.
    """

    m = passadis.get_m()
    n = passadis.get_n()

    # Definim colors
    black = (0, 0, 0)  # Color negre
    white = (255, 255, 255)  # Color blanc
    gray = (128, 128, 128)  # Color gris
    # Llista de colors
    colors = [(0, 0, 255), (255, 0, 0), (0, 255, 0), (255, 165, 0), (128, 0, 128), (165, 42, 42),
    (255, 255, 0)]

    # Per parar la simulació quan vulguem
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                pygame.quit()
                sys.exit()

    # "Netejar" pantalla
    screen.fill(white)

    # Obté la llista de parets del passadís
    parets = passadis.get_parets()

    # Afegeix els extrems del passadís com a parets
    parets.extend([(0, 0), (0, n - 1), (m - 1, 0), (m - 1, n - 1)])

    # Obté els vectors de les entrades del passadís
    vector_entrades = passadis.get_entrades()

    # Obté les posicions dels vectors de les entrades del passadís
    entrades = [e for entrada in vector_entrades for e in entrada]

    # Obté els obstacles del passadís
    obstacles = passadis.get_obstacles()

    # Dibuixa els obstacles
    for x, y in obstacles:
        pygame.draw.rect(screen, black, (y * cell_size[0], x * cell_size[1], cell_size[0], cell_size[1]))
```

```
for x, y in parets:
    # Comprova si és una paret o una cantonada
    if (x, y) in passadis.get_parets():
        color = gray
    else:
        color = black

    # Dibuixa les parets
    pygame.draw.rect(screen, color, (y * cell_size[0], x * cell_size[1], cell_size[0], cell_size[1]))

    # Comprova si és una entrada
    if (x, y) in entrades:
        for entrada in vector_entrades:
            if (x, y) in entrada:
                # Assigna un color a l'entrada
                color = colors[vector_entrades.index(entrada) % len(colors)]

        # Dibuixa un cercle a l'entrada
        pygame.draw.circle(screen, color, (y * cell_size[0] + cell_size[0] // 2,
        x * cell_size[1] + cell_size[1] // 2), cell_size[0] // 4)

# Dibuixar individus
for ind in passadis.get_ind_in_passadis():
    objectiu = ind.get_objectiu()  # Obté l'objectiu de l'individu
    x, y = ind.get_posicio()  # Obté la posició de l'individu
    dx, dy = ind.get_direccio()  # Obté la direcció de l'individu

    for entrada in vector_entrades:
        if objectiu in entrada:
            # Assigna un color a l'objectiu
            color = colors[vector_entrades.index(entrada) % len(colors)]

            # Dibuixa un cercle a l'objectiu
            pygame.draw.circle(screen, color, (y * cell_size[0] + cell_size[0] // 2,
            x * cell_size[1] + cell_size[1] // 2), cell_size[0]//8)

            # Punt d'inici de la fletxa
            arrow_inici = (y * cell_size[0] + cell_size[0] // 2,
            x * cell_size[1] + cell_size[1] // 2)

            # Punt final de la fletxa
            arrow_final = (arrow_inici[0] + dy * cell_size[0] // 2,
            arrow_inici[1] + dx * cell_size[1] // 2)

            # Dibuixa una fletxa que indica la direcció de l'individu
            draw_arrow(screen, color, arrow_inici, arrow_final, 12)

# Actualitzar pantalla
pygame.display.flip()

# Control de velocitat de la simulació (FPS)
clock.tick(fps)
```

```python
# Dibuixa la fletxa que indica la direcció de cada individu
def draw_arrow(screen, color, inici, final, arrow_head_size):
    """
    Dibuixa una fletxa que indica la direcció de l'individu a la pantalla en un entorn de
    simulació pygame.

    Paràmentres:
    screen: La superfície de la pantalla de pygame en la qual es dibuixarà la fletxa.
    color (tuple): El color de la fletxa en format RGB.
    inici (tuple): Les coordenades del punt d'inici de la fletxa (x, y).
    final (tuple): Les coordenades del punt final de la fletxa (x, y).
    arrow_head_size (int): La mida de la capçalera de la fletxa en píxels.
    """

    dx = final[0] - inici[0]  # Diferència en l'eix x
    dy = final[1] - inici[1]  # Diferència en l'eix y

    # Longitud del vector de direcció
    longitud = math.sqrt(dx * dx + dy * dy)
    if longitud == 0:
        return

    udx = dx / longitud  # Component normalitzada en l'eix x
    udy = dy / longitud  # Component normalitzada en l'eix y

    # Angle de l'extrem de la fletxa
    arrow_head_angle = math.pi / 6

    # Punt de l'extrem de la fletxa
    arrow_tail = (final[0] - udx * arrow_head_size, final[1] - udy * arrow_head_size)


    arrow_left = (
        # Coordenada x de l'extrem esquerre de la fletxa
        arrow_tail[0] - udy * arrow_head_size * math.tan(arrow_head_angle / 2),
        # Coordenada y de l'extrem esquerre de la fletxa
        arrow_tail[1] + udx * arrow_head_size * math.tan(arrow_head_angle / 2),
    )  # Càlcul de les coordenades de l'extrem esquerre de la fletxa

    arrow_right = (
        # Coordenada x de l'extrem dret de la fletxa
        arrow_tail[0] + udy * arrow_head_size * math.tan(arrow_head_angle / 2),
        # Coordenada y de l'extrem dret de la fletxa
        arrow_tail[1] - udx * arrow_head_size * math.tan(arrow_head_angle / 2),
    )  # Càlcul de les coordenades de l'extrem dret de la fletxa

    # Dibuixa la línia des de l'inici fins a l'extrem de la fletxa
    pygame.draw.line(screen, color, inici, arrow_tail, 1)
    # Dibuixa el triangle de la fletxa
    pygame.draw.polygon(screen, color, [final, arrow_left, arrow_right])
```

1.6 Simulació

```
import random
import classe_individus as ci
import utils_individus as ui
import dibuixar_pygame as dpygame
import pygame

def simulacio(passadis, num_iteracions, aforament, fps):
    """
    Realitza una simulació en 2D del passadís amb individus en moviment.

    Paràmentres:
    passadis (Passadis): L'objecte passadís que conté la configuració del passadís.
    num_iteracions (int): El nombre total d'iteracions de la simulació.
    aforament (int): El límit d'aforament del passadís.
    fps (int): El nombre de fotogrames per segon per a la simulació.
    """

    # Matriu sobre la que es farà la simulació
    matriu = passadis.get_passadis()

    # Entrades/sortides del passadís
    entrades = passadis.get_entrades()

    # Creem la variable que distingirà als diferents individus
    id_individu = 0

    # Determinem la velocitat i el camp de visio per a tots els individus
    velocitat = 1
    camp_visio = 7

    # Creem una llista per guardar individus
    individus = []

    # Preparem les variables per la simulació
    pygame.init()
    clock = pygame.time.Clock()

    # Obté el tamany de la pantalla del dispositiu on s'executi el programa
    screen_info = pygame.display.Info()
    screen_width = screen_info.current_w * 0.95
    screen_height = screen_info.current_h * 0.95

    # Calcula el tamany de les cel·les en funció del tamany de la pantalla
    cell_width = screen_width // passadis.get_n()
    cell_height = screen_height // passadis.get_m()

    # Tria el tamany més petit per mantindre les cel·les quadrades
    cell_size = min(cell_width, cell_height)

    # Ajusta el tamany de la pantalla per mantenir les cel·les quadrades
    screen_width = cell_size * passadis.get_n()
    screen_height = cell_size * passadis.get_m()

    # Calcula el tamany de cada cel·la
    cell_size = (cell_size, cell_size)
```

```
# Crea una finestra de pantalla completa
screen = pygame.display.set_mode((screen_width, screen_height), pygame.NOFRAME)

# Bucle per a la creacio i moviment dels individus i representacio gráfica de la simulacio
for t in range(num_iteracions):

    # Creem un nombre aleatori d'individus que anirá de 0 al maxim que ens permeti l'aforament
    # tenint en compte els individus que ja hi han
    nous_individus = random.randint(0, (aforament-len(passadis.ind_in_passadis))//10)
    print(f"Quantitat d'invididus en t = {t} = {len(passadis.ind_in_passadis)}\n")

    # Afegim els individus nous que apareixen a les entrades i els afegim a la llista
    for j in range(nous_individus):
        id_individu += 1

        # Calcula la entrada de l'individu
        vector_entrada = random.choice(entrades)
        entrada = random.choice(vector_entrada)

        # Calcula la sortida i l'objectiu de l'individu
        sortida = ui.calcul_sortida(entrada, passadis)
        objectiu = ui.calcul_objectiu(entrada, sortida)

        # Coeficients de importancia per al calcul de les puntuacions de les posicions
        ponderacions = [0.2, 0.4, 0.3, 0.1]


        # Crea l'individu
        individu = ci.Individu(id_individu, entrada, sortida, objectiu, velocitat, passadis.get_m(),
        passadis.get_n(), camp_visio, ponderacions)

        # S'afegeix l'individu al total d'individus
        individus.append(individu)

        # S'afegeix l'individu a la llista d'individus que actualment estan al passadis
        passadis.ind_in_passadis.append(individu)

        # S'actualitza la posicio de la matriu on ara esta l'individu
        matriu[individus[-1].get_posicio()] = 1

        # S'actualitza el diccionari de posicions amb un nou individu
        passadis.diccionario_posicion[individu.posicio] = individu

    # Movem als individus
    for ind in passadis.get_ind_in_passadis():
        ui.moure_individu(ind, passadis)

    # Quan un individu passa per una entrada marca com a 1 la posició en la matriu, ho corregim
    for entrada in entrades:
        for e in entrada:
            matriu[e] = 3

    # Mostrem quants individus hi han al passadís en el instant 't' de temps
    print(f"Total invididus en t = {t} = {len(individus)}\n")

    # Dibuixem el passadís en cada unitat de temps
    dpygame.dibuixar_passadis(passadis, screen, cell_size, clock, fps)
```

```
        # Imprimim l'estat de la matriu en cada unitat de temps 't'
        print(f"Passadís en t = {t}:\n{matriu}\n")

    # Imprimim informació rellevant de la simulació un cop aquesta ha acabat
    total_colisions = total_agrupacions = 0
    for ind in individus:
        total_colisions += ind.trajecte.get_n_colisions()
        total_agrupacions += ind.trajecte.get_n_agrupat()

        print(f"L'individu {ind.get_id()}, amb objectiu {ind.get_objectiu()}"+
              f" ha tingut {ind.trajecte.get_n_colisions()} col·lisions i {ind.trajecte.get_n_agrupat()}
              moviments agrupat.\n")
        print(f"La puntuació d'aptitud de l'individu {ind.get_id()} = {ind.get_aptitud()}\n")
        print(f"Ha fet en {ind.trajecte.get_t_recorregut()} segons el recorregut:\n
        {ind.trajecte.get_recorregut()}\n\n")

    print(f"En aquesta simulació hi ha hagut un total de {total_colisions} col·lisions
    i {total_agrupacions} agrupacions entre els {(individus[-1].get_id()+1)} individus\n")
    print(f"Per tant, de mitja, hi han hagut {total_colisions/(individus[-1].get_id()+1):.2f}
    col·lisions i {total_agrupacions/(individus[-1].get_id()+1):.2f} agrupacions\n")

# Passadís amb 40 files, 30 columnes, i entrada unica
cas1 = cp.Passadis(0, 40, 30, 6, True, False, False)

# Simulació amb 1000 iteracions, 400 individus i 8 FPS
simulacio(cas1, 1000, 400, 8)
```

2 Algorisme Genètic

2.1 L'algorisme

```
def algorisme_genetic(passadis, n_generacions, n_iteracions, aforament, fps):
    """
    Executa un algorisme genètic per optimitzar la busqueda dels coeficients d'importància per a la
    presa de decisions en el moviment dels individus al llarg del passadís.

    Args:
        passadis (Passadis): Objecte que representa els passadissos disponibles.
        n_generacions (int): Nombre de generacions per a executar l'algorisme genètic.
        n_iteracions (int): Nombre d'iteracions per cada generació.
        aforament (int): Capacitat màxima d'individus en el passadís.
        fps (int): Nombre de fotogrames per segon per a la representació gràfica de la simulació.
    """

    # Obtenim la matriu sobre la qual es realitzarà la simulació
    matriu = passadis.get_passadis()

    # Obtenim les entrades i sortides del passadís
    entrades = passadis.get_entrades()

    # Inicialitzem Pygame i creem un rellotge per controlar el temps
    pygame.init()
    clock = pygame.time.Clock()

    # Obtenim la resolució de la pantalla del dispositiu on s'executa el programa
    screen_info = pygame.display.Info()
    screen_width = screen_info.current_w * 0.95
    screen_height = screen_info.current_h * 0.95

    # Calculem les dimensions de les cel·les en funció de la resolució de la pantalla
    cell_width = screen_width // passadis.get_n()
    cell_height = screen_height // passadis.get_m()

    # Seleccionem la mida més petita per mantenir les cel·les quadrades
    cell_size = min(cell_width, cell_height)

    # Ajustem les dimensions de la pantalla per mantenir les cel·les quadrades
    screen_width = cell_size * passadis.get_n()
    screen_height = cell_size * passadis.get_m()

    # Calculem la mida de cada cel·la
    cell_size = (cell_size, cell_size)

    # Creem una finestra a pantalla completa
    screen = pygame.display.set_mode((screen_width, screen_height), pygame.NOFRAME)

    # Inicialitzem algunes llistes i variables que farem servir més endavant
    millors_individus = []
    individus = []
    passadis.ind_in_passadis = []
    nova_generacio = []
    id_individu = 0
    mitjanes_apt = []
    totes_ponderacions = []
```

```
# Creem una font que farem servir per mostrar text a la pantalla
font = pygame.font.Font(None, 24)

# Bucle per a la creació i moviment dels individus i representació gráfica de la simulacio
for gen in range(n_generacions):
    # Inicialitzem llistes d'individus per a cada nova generació
    individus = []
    passadis.ind_in_passadis = []

    if gen > 0:
        # Variable auxiliar per a iterar sobre la llista nova_generacio
        j = 0

    # Comencem el bucle per a cada unitat de temps en la generació (251 iteracions)
    for t in range(n_iteracions):
        # Creem un nombre aleatori d'individus que aniran de 0 al màxim que ens permeti l'aforament
        nous_individus = random.randint(0, (aforament-len(passadis.ind_in_passadis))//10)
        print(f"Nous individus = {nous_individus}\n")

        # Si estem a la primera generació (gen == 0)
        if gen == 0:
            # Creem els individus de la primera generació
            primers_individus = primera_generacio(passadis, nous_individus, id_individu)
            # Afegeix cada nou individu a la llista d'individus i al passadís
            for ind in primers_individus:
                individus.append(ind)
                passadis.ind_in_passadis.append(ind)

                # S'actualitza la posicio de la matriu on ara esta l'individu
                matriu[individus[-1].get_posicio()] = 1

                # S'actualitza el diccionari de posicions amb un nou individu
                passadis.diccionario_posicion[ind.posicio] = ind

            # Incrementem l'ID de l'individu per als futurs individus
            id_individu += len(primers_individus)

        # Si no estem a la primera generació
        if gen > 0:
            # Creem nous individus fins que el nombre d'individus actuals més els nous individus
            # superi la longitud de la nova generació
            while (len(individus) + nous_individus) > len(nova_generacio):
                # Seleccionem dos dels millors individus a l'atzar per ser pares
                pare1, pare2 = random.sample(millors_individus, 2)

                # Creem un nou individu a partir dels dos pares
                fill = crear_fill(pare1, pare2, passadis, id_individu)
                id_individu += 1

                # Afegim el nou individu a la nova generació
                nova_generacio.append(fill)

            # Afegim els nous individus a la llista d'individus i al passadís, i actualitzem
            # la matriu i el diccionari de posicions
            for _ in range(nous_individus):
                individus.append(nova_generacio[j])
                passadis.ind_in_passadis.append(nova_generacio[j])
```

```
                matriu[individus[-1].get_posicio()] = 1
                passadis.diccionario_posicion[nova_generacio[j].posicio] = nova_generacio[j]
                j += 1

            # Imprimim la longitud de la llista d'individus i el valor de j
            print(f"len(individus) = {len(individus)}\n")
            print(f"j = {j}\n")

        # Movem cada individu al passadís
        for ind in passadis.get_ind_in_passadis():
            ui.moure_individu(ind, passadis)

        # Si un individu ha passat per una entrada, corregim la matriu
        for entrada in entrades:
            for e in entrada:
                matriu[e] = 3

        # Imprimim la quantitat d'individus en el passadís i en total en aquesta unitat de temps
        print(f"Quantitat d'invididus en t = {t} = {len(passadis.ind_in_passadis)}\n")
        print(f"Total invididus en t = {t} = {len(individus)}\n")

        # Imprimim l'estat de la matriu en aquesta unitat de temps
        print(f"Passadís en t = {t}:\n{matriu}\n")

        # Dibuixem el passadís en cada unitat de temps
        dpygame.dibuixar_passadis(passadis, screen, cell_size, clock, fps)

        # Creem un text amb el número de generació actual i el dibuixem a la pantalla
        gen_surface = font.render(f"Generación: {gen}", True, (0, 0, 0))
        screen.blit(gen_surface, (25, 25))

        # Actualitzem la pantalla
        pygame.display.flip()

    # Buidem el passadís
    matriu[1:-1, 1:-1] = 0

    # Inicialitzem diverses variables per recollir dades d'interès
    aptituds = {}  # Diccionari per emmagatzemar les puntuacions d'aptitud
    mitjana_aptitud = 0  # Mitjana d'aptituds
    total_colisions = total_agrupacions = 0  # Total de col·lisions i agrupacions
    ponderacions_totals = [0, 0, 0, 0]  # Llista per emmagatzemar les ponderacions totals

    # Iterem per cada individu
    for ind in individus:
        # Calculem l'aptitud de l'individu i l'emmagatzemem
        aptitud = f_aptitud(ind)
        ind.set_aptitud(aptitud)
        aptituds[ind] = aptitud
        # Incrementem la mitjana d'aptitud amb l'aptitud de l'individu actual
        mitjana_aptitud += aptitud

        # Recopilem dades addicionals sobre col·lisions i agrupacions, i actualitzem
        # les ponderacions totals
        total_colisions += ind.trajecte.get_n_colisions()
        total_agrupacions += ind.trajecte.get_n_agrupat()
        ponderacions_totals[0] += ind.trajecte.get_ponderacions()[0]
```

```
        ponderacions_totals[1] += ind.trajecte.get_ponderacions()[1]
        ponderacions_totals[2] += ind.trajecte.get_ponderacions()[2]
        ponderacions_totals[3] += ind.trajecte.get_ponderacions()[3]

    # Calculem les ponderacions mitjanes i l'aptitud mitjana
    ponderacions_totals[0] = ponderacions_totals[0] / len(individus)
    ponderacions_totals[1] = ponderacions_totals[1] / len(individus)
    ponderacions_totals[2] = ponderacions_totals[2] / len(individus)
    ponderacions_totals[3] = ponderacions_totals[3] / len(individus)
    mitjana_aptitud = mitjana_aptitud / len(individus)
    mitjanes_apt.append(mitjana_aptitud)

    # Imprimim dades sobre cada individu
    for ind in individus:
        if ind.get_posicio() == None:
            print(f"L'individu {ind.get_id()}, amb objectiu {ind.get_objectiu()}"+
                f" ha tingut {ind.trajecte.get_n_colisions()} col·lisions,
                {ind.trajecte.get_n_agrupat()} moviments agrupat"+
                f"i ha fet el recorregut en {ind.trajecte.get_t_recorregut()} moviments\n")
            print(f"La puntuació d'aptitud de l'individu {ind.get_id()} = {ind.get_aptitud()}\n")

    # Calculem el nombre total d'individus i imprimim dades globals de la simulació
    len_individus = individus[-1].get_id() + 1
    print(f"En aquesta simulació hi ha hagut un total de {total_colisions} col·lisions i
    {total_agrupacions} agrupacions entre els {len_individus} individus\n")
    print(f"Per tant, de mitja, hi han hagut {total_colisions/len_individus:.2f} col·lisions i
    {total_agrupacions/len_individus:.2f} agrupacions\n")
    print(f"La puntuació d'aptitud mitjana = {mitjana_aptitud}\n")
    print(f"P = {ponderacions_totals}\n")
    totes_ponderacions.append(ponderacions_totals)

    # Aquí comença un nou cicle de l'algoritme genètic
    millors_individus = []  # Llista dels millors individus
    nova_generacio = []  # Llista per la nova generació d'individus

    # Filtrar els individus que han completat el recorregut
    aptituds = {ind: apt for ind, apt in aptituds.items()
    if ind.trajecte.get_recorregut()[-1] is None}

    # Ordenem els individus per aptitud, en ordre descendent
    aptituds_ordenades = dict(sorted(aptituds.items(), key=lambda item: item[1], reverse=True))

    # Calculem el nombre d'individus en el 50% superior
    n_millors = int(len(aptituds_ordenades) / 2)

    # Obtenim el 50% dels individus amb la millor puntuació d'aptitud
    millors_individus = list(aptituds_ordenades.keys())[:n_millors]

    # Si tenim menys de 20 millors individus, aturem el bucle
    if len(millors_individus) < 20:
        break

    # Inicialitzem un contador i una llista per emmagatzemar les ponderacions dels millors individus
    id_individu = 0
    millors_ponderacions = [0, 0, 0, 0]
```

```
    # Iterem pels millors individus
    for ind in mejores_individuos:
        # Actualitzem les ponderacions dels millors individus
        millors_ponderacions[0] += ind.trajecte.get_ponderacions()[0]
        millors_ponderacions[1] += ind.trajecte.get_ponderacions()[1]
        millors_ponderacions[2] += ind.trajecte.get_ponderacions()[2]
        millors_ponderacions[3] += ind.trajecte.get_ponderacions()[3]

        # Copiem el millor individu i l'afegim a la nova generació
        nou_ind = copiar_individu(ind, passadis, id_individu)
        nova_generacio.append(nou_ind)
        id_individu += 1  # Incrementem el comptador d'individus

    # Calculem les ponderacions mitjanes dels millors individus
    millors_ponderacions[0] = millors_ponderacions[0] / len(millors_individus)
    millors_ponderacions[1] = millors_ponderacions[1] / len(millors_individus)
    millors_ponderacions[2] = millors_ponderacions[2] / len(millors_individus)
    millors_ponderacions[3] = millors_ponderacions[3] / len(millors_individus)
    print(f"\nM = {millors_ponderacions}\n")

    # Començar bucle fins que la nova generació sigui de la mateixa mida que l'anterior
    while len(individus) + 1 > len(nova_generacio):
        # Escollir aleatòriament dos individus dels millors per ser pares
        pare1, pare2 = random.sample(millors_individus, 2)

        # Crear un nou individu (fill) a partir de la combinació de les ponderacions dels pares
        fill = crear_fill(pare1, pare2, passadis, id_individu)
        id_individu += 1  # Incrementar l'identificador per al pròxim individu

        # Afegir el nou individu a la nova generació
        nova_generacio.append(hijo)

    # Una vegada s'ha creat la nova generació, es realitza una mutació
    nova_generacio = mutar(nova_generacio)

    # Mostrar la longitud de la nova generació
    print(f"len(nova_generacio) = {len(nova_generacio)}\n")

    # Pausar durant un segon
    time.sleep(1)

# Iterar a través de totes les ponderacions per a cada generació i imprimir-les
for i in range(len(totes_ponderacions)):
    print(f"Ponderacions en generació {i} = {totes_ponderacions[i]}\n")

# Crear un gràfic per mostrar la evolució de l'aptitud mitjana
plt.plot(mitjanes_apt)  # Generar el gràfic amb les mitjanes d'aptitud
plt.title("Evolución de la aptitud media")  # Afegir títol al gràfic
plt.xlabel("Generaciones")  # Afegir etiqueta al eix X
plt.ylabel("Aptitud media")  # Afegir etiqueta al eix Y
plt.show()  # Mostrar el gràfic
```

2.2 Primera generació

```
# Funció per a generar la primera generació de individus
def primera_generacio(passadis, n_individus, id_individu):
    """
    Genera la primera generació d'individus.

    Args:
        passadis (Passadis): Objecte que representa els passadissos disponibles.
        n_individus (int): Nombre d'individus a generar.
        id_individu (int): Identificador únic per als individus.

    Returns:
        list: Llista d'individus generats.
    """

    # Obtenir les entrades del passadís
    entrades = passadis.get_entrades()
    # Crear una llista buida per als individus
    individus = []

    # Per a cada individu en la generació
    for j in range(n_individus):
        # Incrementar l'identificador del individu
        id_individu += 1

        # Escollir una entrada aleatòria per a l'individu
        vector_entrada = random.choice(entrades)
        entrada = random.choice(vector_entrada)


        # Calcular la sortida i l'objectiu de l'individu
        sortida = ui.calcul_sortida(entrada, passadis)
        objectiu = ui.calcul_objectiu(entrada, sortida)

        # Generar ponderacions aleatòries per a l'individu
        ponderacions = generar_ponderacions(4)

        # Escollir un camp de visió aleatori dins d'un rang definit
        camp_visio = 4

        # Crear un nou individu amb els paràmetres generats
        individu = ci.Individu(id_individu, entrada, sortida, objectiu, 1, passadis.get_m(),
        passadis.get_n(), camp_visio, ponderacions)

        # Afegir l'individu a la llista d'individus
        individus.append(individu)

    # Tornar la llista d'individus
    return individus
```

```python
def generar_ponderacions(n):
    """
    Genera un vector de ponderacions aleatòries.

    Args:
        n (int): Nombre de ponderacions a generar.

    Returns:
        list: Vector de ponderacions generades.
    """

    # Generar n-1 números aleatoris entre 0.1 i 0.7
    ponderacions = [round(random.uniform(0.1, 0.7), 2) for _ in range(n-1)]

    # Calcular la suma total de les ponderacions generades
    total = sum(ponderacions)

    # Normalitzar les ponderacions perquè sumin 0.9 (es deixa un espai per a la quarta ponderació)
    ponderacions = [round(i / total * 0.9, 2) for i in ponderacions]

    # Corregir possibles errors d'arrodoniment
    diferencia = 0.9 - sum(ponderacions)
    if diferencia != 0:
        # Si la primera ponderació més la diferència és major que 0.7, afegim la diferència a la
        # ponderació més petita
        if ponderacions[0] + diferencia > 0.7:
            min_index = ponderacions.index(min(ponderacions))
            ponderacions[min_index] += diferencia
        else:
            # Si no, s'afegeix la diferència a la primera ponderació
            ponderacions[0] += diferencia

    # S'afegeix la quarta ponderació amb un valor de 0.1
    ponderacions.append(0.1)

    # Retorna les ponderacions
    return ponderacions
```

2.3 Funció d'aptitud

```
def f_aptitud(ind):
    """
    Calcula l'aptitud d'un individu.

    Args:
        ind (Individu): Individu per al qual es calcula l'aptitud.

    Returns:
        int: Aptitud de l'individu.
    """

    # Obtenir el nombre de col·lisions de l'individu
    colisions = ind.trajecte.get_n_colisions()

    # Obtenir el nombre d'agrupacions de l'individu
    agrupacions = ind.trajecte.get_n_agrupat()

    # Obtenir el temps total que l'individu ha recorregut
    temps_total = ind.trajecte.get_t_recorregut()

    # Calcular la aptitud com el nombre d'agrupacions menys el nombre de col·lisions i el temps total
    aptitud = agrupacions - colisions - temps_total

    # Retornar la aptitud
    return aptitud
```

2.4 Següents generacions

```
def copiar_individu(ind, passadis, id_individu):
    """
    Copia un individu existent.

    Args:
        ind (Individu): Individu a copiar.
        passadis (Passadis): Objecte que representa els passadissos disponibles.
        id_individu (int): Identificador únic per al nou individu copiat.

    Returns:
        Individu: Nou individu copiat.
    """

    # Obtenir les entrades del passadís
    entrades = passadis.get_entrades()

    # Obtenir les ponderacions de l'individu
    ponderacions = ind.trajecte.get_ponderacions()

    # Establir el camp de visió
    camp_visio = 4

    # Escollir una entrada aleatòria per a l'individu
    vector_entrada = random.choice(entrades)
    entrada = random.choice(vector_entrada)
```

```
    # Calcular la sortida i l'objectiu de l'individu
    sortida = ui.calcul_sortida(entrada, passadis)
    objectiu = ui.calcul_objectiu(entrada, sortida)

    # Crear un nou individu que és una còpia de l'original, però amb un nou identificador
    nou_ind = ci.Individu(id_individu, entrada, sortida, objectiu, 1, ind.m, ind.n,
    camp_visio, ponderacions)
    return nou_ind

# Definim una funció que creï un nou individu a partir de dos pares
def crear_fill(pare1, pare2, passadis, id):
    """
    Crea un nou individu a partir de dos pares.

    Args:
        pare1 (Individu): Primer pare.
        pare2 (Individu): Segon pare.
        passadis (Passadis): Objecte que representa els passadissos disponibles.
        id (int): Identificador únic per al nou individu.

    Returns:
        Individu: El nou individu creat.
    """
    # Obtenir les entrades possibles del passadís
    entrades = passadis.get_entrades()

    # Obtenir les ponderacions de cada pare
    ponderacions_padre1 = pare1.trajecte.get_ponderacions()
    ponderacions_padre2 = pare2.trajecte.get_ponderacions()

    # Donat que el camp de visió és constant obtenim el mateix que un dels pares
    camp_visio = pare1.get_camp_visio()

    # Calcular el punt mitjà de les ponderacions
    meitat = len(ponderacions_padre1) // 2

    # Crear les ponderacions del fill combinant les ponderacions dels pares
    ponderacions_fill = ponderacions_padre1[:meitat] + ponderacions_padre2[meitat:]

    # Normalitzar les ponderacions perquè sumin 1
    suma_total = sum(ponderacions_fill)
    ponderacions_fill = [p/suma_total for p in ponderacions_fill]

    # Calcula l'entrada de l'individu escollint una entrada aleatòria
    vector_entrada = random.choice(entrades)
    entrada = random.choice(vector_entrada)

    # Calcular la sortida i l'objectiu de l'individu
    sortida = ui.calcul_sortida(entrada, passadis)
    objectiu = ui.calcul_objectiu(entrada, sortida)

    # Crear un nou individu amb les ponderacions del fill
    fill = ci.Individu(id, entrada, sortida, objectiu, 1, pare1.m, pare2.n,
    camp_visio, ponderacions_fill)

    # Retornar el nou individu
    return fill
```

2.5 Mutació

```python
def mutar(poblacio, percentatge=0.2):
    """
    Mutar una certa percentatge de la població.

    Args:
        poblacio (list): Llista d'individus de la població.
        percentatge (float, opcional): Percentatge d'individus a mutar. El valor per defecte és 0.2.

    Returns:
        list: La població amb les mutacions realitzades.
    """

    # Calcular la quantitat d'individus a mutar
    num_mutacions = int(len(poblacio) * percentatge)

    # Seleccionar aleatòriament els individus a mutar
    individus_a_mutar = random.sample(poblacio, num_mutacions)

    # Per a cada individu a mutar
    for individu in individus_a_mutar:
        # Seleccionar una ponderació a l'atzar, excluint la quarta ponderació
        index_ponderacio = random.randint(0, len(individu.trajecte.get_ponderacions()) - 2)  # '-2' per a
        # Calcular una nova ponderació que no faci que la suma de ponderacions superi el 0.9
        nova_ponderacio = random.uniform(0.1, 0.9 - sum([p for i, p
        in enumerate(individu.trajecte.get_ponderacions()) if i != index_ponderacio and i != 3]))

        # Actualitzar la ponderació de l'individu
        individu.trajecte.get_ponderacions()[index_ponderacio] = nova_ponderacio

    # Retornar la població amb les mutacions realitzades
    return poblacio
```

3 Model Continu

3.1 Objectes conceptuals del model

3.1.1 Classe passadis

```
class Passadis:
    def __init__(self, id, m, n):
        # Identificador únic del passadís per diferenciar-lo
        self.id = id

        # Nombre de files
        self.m = m

        # Nombre de carrils (columnes)
        self.n= n

        # Llista dels individus que es troben al passadís
        self.ind_in_passadis = []

        # Diccionari que conté la posició de cada individu
        self.ind_posicions = {}

        # Obtenció dels segments de rectes que representen les entrades i parets
        self.entrades, self.parets = up.crear_passadis(m, n)

    # Retorna l'identificador del passadís
    def get_id(self):
        return self.id

    # Retorna el nombre de files
    def get_m(self):
        return self.m

    # Retorna el nombre de carrils
    def get_n(self):
        return self.n

    # Retorna les entrades
    def get_entrades(self):
        return self.entrades

    # Retorna les parets
    def get_parets(self):
        return self.parets

    # Retorna els individus que es troben al passadís
    def get_ind_in_passadis(self):
        return self.ind_in_passadis

    # Retorna el diccionari de posicions
    def get_ind_posicions(self):
        return self.ind_posicions
```

3.1.2 Classe individu

```
class Individu:
    def __init__(self, id, posicio, sortida, objectiu, grup, v_min, v_max, velocitat, m, n,
    radi, temps_horitzo):
        # Identificador únic per individu
        self.id = id

        # Paràmentres posicionals
        # Dupla (x, y) de la posicio actual de l'individu
        self.posicio = posicio

        # dupla (x, y) de la posicio inicial de l'individu
        self.entrada = posicio

        # Llista de duples (x,y) amb les possibles posicions on pot estar l'objectiu
        self.sortida = sortida

        # Dupla(x, y) posició de l'objectiu de l'individu
        self.objectiu  = objectiu

        # Classificació segons la sortida de l'individu (pot valdre 0 o 1)
        self.grup = grup

        # Recorregut realitzat fins al moment per l'individu (una llista ordenada de les
        # posicions de l'individu)
        self.recorregut = [posicio]

        # Files passadís
        self.m = m

        # Carrils passadís
        self.n = n

        # Paràmentres velocitat
        # Velocitat actual individu
        self.velocitat = velocitat

        # Nombres reals que determinen la velocitat mínima i màxima
        self.v_min = v_min
        self.v_max = v_max

        # Paràmentres control entorn
        # Radi de col·lisió de l'individu
        self.radi = radi

        # Temps futur al que mira l'individu per predir el comportament de l'entorn
        self.temps_horitzo = temps_horitzo

        # Radi que determina l'àrea on es poden produir els possibles moviments de l'individu
        self.radi_moviment = v_max + radi

        # Nombre de col·lisions que es tenen al llarg del recorregut
        self.colisions = 0
```

```
# Retorna l'identificador de l'individu
def get_id(self):
    return self.id

# Retorna el valor de m
def get_m(self):
    return self.m

# Retorna el valor de n
def get_n(self):
    return self.n

# Retorna la posició actual
def get_posicio(self):
    return self.posicio

# Retorna la posició per la que el individu ha entrat al passadís
def get_entrada(self):
    return self.entrada

# Retorna una llista que conté un conjunt de posicions entre les que estan els possibles objectius
def get_sortida(self):
    return self.sortida

# Retorna la posició de l'objectiu de l'individu
def get_objectiu(self):
    return self.objectiu

# Retorna el grup al que pertany l'individu (la classificació es fa segons la component y
# de l'objectiu)
def get_grup(self):
    return self.grup

# Retorna el recorregut realitzat fins al moment per l'individu (una llista ordenada de
# les posicions de l'individu)
def get_recorregut(self):
    return self.recorregut

# Retorna el radi de col·lisió de l'individu
def get_radi(self):
    return self.radi

# Retorna el radi de moviment de l'individu
def get_radi_moviment(self):
    return self.radi_moviment

# Retorna el temps d'horitzó de l'individu
def get_temps_horitzo(self):
    return self.temps_horitzo

# Retorna el valor mínim que pot prendre la velocitat
def get_v_min(self):
    return self.v_min

# Retorna el valor màxim que pot prendre la velocitat
def get_v_max(self):
    return self.v_max
```

```
    # Retorna la dupla (x,y) amb la velocitat actual
    def get_velocitat(self):
         return self.velocitat


    # Retorna la quantitat de col·lisions que s'han donat fins a aquest moment
    def get_colisions(self):
        return self.colisions

    # Estableix una nova posició per a l'individu i l'afegeix al seu recorregut
    def set_posicio(self, nova_posicio):
        self.posicio = nova_posicio
        self.recorregut.append(nova_posicio)

    # Estableix un nou objectiu per a l'individu
    def set_objectiu(self, nou_objectiu):
        self.objectiu = nou_objectiu

    # Estableix una nova velocitat per a l'individu
    def set_velocitat(self, nova_velocitat):
        self.velocitat = nova_velocitat

    # Afegeix una nova col·lisió al recompte total de col·lisions
    def add_colisio(self):
        self.colisions += 1
```

3.2 Creació passadís

```
def crear_passadis(m, n):
    """
    Crea un passadís amb les entrades i parets corresponents.

    Paràmetres:
    m (int): Dimensió del passadís en l'eix de l'x.
    n (int): Dimensió del passadís en l'eix de l'y.

    Retorna: Les llistes d'entrades i parets.
    """
    # Creem parets
    parets = crear_parets(m, n)

    # Distància de les entrades respecte la paret en el eix de la y
    delta = n/20

    # Creem entrades
    entrades = crear_entrades(m, n, delta)

    return entrades, parets
```

```python
def crear_parets(m, n):
    """
    Crea les parets del passadís.

    Paràmetres:
    m (int): Dimensió del passadís en l'eix de l'x.
    n (int): Dimensió del passadís en l'eix de l'y.

    Retorna: Una llista amb les coordenades dels limits de les parets [min_x, max_x, min_y, max_y].
    """
    min_x = 0
    max_x = m
    min_y = 0
    max_y = n

    # Torna una llista amb els límits del passadís
    return [min_x, max_x, min_y, max_y]


def crear_entrades(m, n, delta):
    """
    Crea les entrades del passadís.

    Paràmetres:
    m (int): Dimensió del passadís en l'eix de l'x.
    n (int): Dimensió del passadís en l'eix de l'y.
    delta (float): Distància de les entrades respecte a la paret en l'eix de l'y.

    Retorna: Una llista amb les coordenades dels segments d'entrada i sortida.
    """

    # Crear els segments d'entrada i sortida
    entrada1 = [(0, delta), (m, delta)]
    entrada2 = [(0, n - delta), (m, n - delta)]

    # Torna una llista amb tots els segments
    return [entrada1, entrada2]


def esta_dins_segment(punt, segment):
    """
    Comprova si un punt està dins d'un segment.

    Paràmetres:
    punt (dupla): Coordenades del punt a comprovar.
    segment (llista): Coordenades dels extrems del segment [punt1, punt2].

    Retorna:
    bool: True si el punt està dins del segment, False si no ho està.
    """

    # Calcula la distancia entre el punt i els dos extrems del segment
    dist_a = np.linalg.norm(np.array(punt) - np.array(segment[0]))
    dist_b = np.linalg.norm(np.array(punt) - np.array(segment[1]))

    # Calcula la llargada del segment
    llargada_segment = np.linalg.norm(np.array(segment[0]) - np.array(segment[1]))
```

```
    # Comprova si la suma de les dos distancies és igual a la llargada del segment (amb un cert
    # marge d'error per tindre en compte la precisió de punt flotant)
    if np.isclose(dist_a + dist_b, llargada_segment, rtol=1e-05):
        return True
    else:
        return False
```

3.3 Funcions auxiliars dels individus

3.3.1 Entrada, sortida i objectiu

```
def calcul_posicions_entrada(m, n, radi, delta):
    """
    Calcula les posicions en les quals poden aparèixer els individus al passadís.

    Paràmetres:
    m (int): Dimensió del passadís en l'eix de l'x.
    n (int): Dimensió del passadís en l'eix de l'y.
    radi (float): Radi de l'individu.
    delta (float): Distància de l'entrada respecte a la paret en l'eix de l'y.

    Retorna: Una llista de duples que conté les possibles posicions d'entrada al passadís.
    """

    # Llista on es guarden les possibles posicions per entrar
    posiciones = []

    # L'interval de possibles punts comença en x = radi i acaba en x = m
    # No agafem tota la recta sino que agafem punts amb una distància de 2*radi entre cada punt
    for x in np.arange(radi, m, 2*radi):
        # Posicions de la entrada/sortida de la part inferior del passadís
        posiciones.append((x, delta))

        # Posicions de la entrada/sortida de la part superior del passadís
        posiciones.append((x, n - delta))

    # Torna una llista amb les possibles posicions en les que es pot aparèixer al passadís
    return posiciones


# Calcula la sortida (segment de punts on es troba l'objectiu)
def calcul_sortida(ind_entrada, entrades):
    """
    Selecciona una de les entrades(accessos) com a sortida

    Paràmetres:
    ind_entrada (dupla): Coordenades de l'entrada de l'individu.
    entrades (llista): Llista amb els segments de les entrades disponibles.

    Retorna: Una llista amb les coordenades del segment de la sortida de l'individu.
    """

    # D'entre els accessos disponibles descarta el accès que és la entrada de l'individu
    for entrada in entrades:
        if up.esta_dins_segment(ind_entrada, entrada): continue
        else: return entrada
```

```python
# Aquest càlcul de l'objectiu només es dona al principip del trajecte.
# Després el calcul de l'objectiu es fa a través de trobar el punt més proper a l'individu
def calcul_objectiu(sortida):
    """
    Calcula, a l'inici del trajecte, l'objectiu de l'individu a partir del segment de la sortida.

    Paràmetres:
    sortida (llista): Coordenades del segment de la sortida.

    Retorna:
    dupla: Coordenades de l'objectiu de l'individu.
    """

    x = round(np.random.uniform(sortida[0][0], sortida[1][0]),2)
    y = round(np.random.uniform(sortida[0][1], sortida[1][1]),2)

    return (x, y)
```

3.3.2 Velocitat inicial

```python
def calcul_velocitat_inicial(v_min, v_max):
    """
    Calcula la velocitat inicial d'un individu.

    Paràmetres:
    v_min (float): Valor mínim de la magnitud de la velocitat.
    v_max (float): Valor màxim de la magnitud de la velocitat.

    Retorna:
    numpy.ndarray: Un vector numpy que representa la velocitat inicial de l'individu.
    """

    # Genera una magnitud de velocitat aleatoria en el rang [v_min, v_max]
    magnitud = np.random.uniform(v_min, v_max)

    # Genera una direcció de velocitat aleatoria en el rang [0, 2]
    direccio = np.random.uniform(0, 2 * np.pi)

    # Calcula las componentes x e y de la velocidad
    vx = magnitud * np.cos(direccio)
    vy = magnitud * np.sin(direccio)

    return np.array((vx, vy))
```

3.4 Moviment dels individus

3.4.1 Càlcul nova velocitat

```
import numpy as np

def calcul_nova_velocitat(ind, total_individus):
    """
    Calcula la nova velocitat per a un individu en funció de la seva posició, objectiu i
    els individus propers.

    Paràmetres:
    ind (objecte Individu): L'individu per al qual es calcularà la nova velocitat.
    total_individus (llista d'objectes Individu): Tots els individus del passadís.

    Actualitza la velocitat de l'individu indicant la direcció i magnitud en que moure's
    """
    # Velocitat i posició actual de l'individu
    vel = np.array(ind.get_velocitat())
    pos = np.array(ind.get_posicio())

    # Objectiu de l'individu
    objectiu = np.array(ind.get_objectiu())

    # Grup de l'individu
    grup = ind.get_grup()

    # Radi de l'individu
    radi = ind.get_radi()

    # Velocitat mínima i màxima de l'individu
    v_min = ind.get_v_min()
    v_max = ind.get_v_max()

    # Llista de posicions per als individus agrupats
    inds_agrupats = []

    # Llista de posicions per als individus que van en direcció contraria
    inds_colisions = []

    # Llista de distàncies per als individus agrupats
    dists_agrupats = []

    # Llista de distàncies per als individus que van en direcció contraria
    dists_colisions = []

    # Calcular les posicions ajustades segons el radi i les distàncies als individus propers
    for individu in total_individus:
        if individu == ind:
            continue

        # Velocitat i posició relativa respecte a un altre individu
        v_rel = vel - np.array(individu.get_velocitat())
        p_rel = pos - np.array(individu.get_posicio())


        # Distància euclidiana respecte a un altre individu
        dist = np.linalg.norm(p_rel)
```

```
        # Actualitzar el comptador de col·lisions si la distància és menor a dos radis
        if dist < 2*radi:
            ind.add_colisio()

        # Calcular el temps estimat de col·lisió amb un altre individu
        if np.linalg.norm(v_rel) != 0:
            t_col = (dist - radi - individu.get_radi()) / np.linalg.norm(v_rel)
        else:
            t_col = (dist - radi - individu.get_radi()) / 0.001

        # Si el temps de col·lisió és major o igual que el temps d'hortizó de l'individu aleshores
        # no tenim en compte a l'altre individu
        if t_col >= ind.get_temps_horitzo():
            continue

        # Ajustem la posició de l'altre individu segons el radi i la direcció de l'altre individu
        # i l'afegim a la llista de posicions agrupades
        # Calculem també la distància fins a aquesta posició i l'afegim a la llista de distàncies
        if grup == individu.get_grup():
            pos_ajustada = np.array(individu.get_posicio()) + (2 * radi * np.sign(p_rel))
            inds_agrupats.append(pos_ajustada)
            dists_agrupats.append(dist)

        # Ajustem la posició de l'altre individu segons el radi i la direcció de l'altre individu
        # i l'afegim a la llista de posicions en col·lisió
        # Calculem també la distància fins a aquesta posició i l'afegim a la llista de distàncies
        else:
            pos_ajustada = np.array(individu.get_posicio()) - (2 * radi * np.sign(p_rel))
            inds_colisions.append(pos_ajustada)
            dists_colisions.append(dist)

    if inds_agrupats:
        # Calculem els inversos de les distàncies als individus agrupats
        pesos_agr = np.reciprocal(dists_agrupats)
        pesos_agr = np.where(pesos_agr == np.inf, 1e10, pesos_agr)

        # Mitjana ponderada utilitzant els inversos de les distàncies i les posicions
        # ajustades dels individus agrupats
        mitja_agrupats = np.average(np.array(inds_agrupats), axis=0, weights=pesos_agr)
    else:
        mitja_agrupats = pos

    if inds_colisions:
        # Calculem els inversos de les distàncies als individus amb potencial col·lisió
        pesos_col = np.reciprocal(dists_colisions)
        pesos_col = np.where(pesos_col == np.inf, 1e10, pesos_col)

        # Mitjana ponderada utilitzant els inversos de les distàncies i les posicions
        # ajustades dels individus amb potencial col·lisió
        mitja_colisions = np.average(np.array(inds_colisions), axis=0, weights=pesos_col)
    else:
        mitja_colisions = pos
```

```
    """
    Tenint en compte la mitjana dels individus agrupats, la mitjana dels individus en direcció
    contraria i la posició de l'objectiu tenim un triangle en el que ens podem moure per
    tots els punts de l'interior depenent dels valors dels coeficients d'importància
    que multiplicaran amb un valor entre 0 i 1 els tres punts del triangle esmentats.
    """

    # Control de decisió del moviment
    # Si l'individu està aprop de l'objectiu prioritza la posició de l'objectiu en el càlcul
    # de la nova direcció
    if abs(pos[1] - objectiu[1]) <= 0.2 * abs(ind.get_entrada()[1] - objectiu[1]):
        nova_direccio = 0.5 * (objectiu - pos) + 0.2 * (mitja_agrupats - pos)
        - 0.3 * (mitja_colisions - pos)

    # Si dins del temps d'horitzó hi han més individus agrupats que potencials col·lisions
    # es prioritza la mitjana dels individus agrupats
    elif len(inds_agrupats) > len(inds_colisions):
        nova_direccio = 0.05 * (objectiu - pos) + 0.9 * (mitja_agrupats - pos)
        - 0.05 * (mitja_colisions - pos)

    # Si dins del temps d'horitzó hi han més individus col·lisionadors que agrupats s'equilibra
    # la prioritat entre els dos grups d'individus
    elif len(inds_agrupats) <= len(inds_colisions):
        nova_direccio = 0.05 * (objectiu - pos) + 0.5 * (mitja_agrupats - pos)
        - 0.4 * (mitja_colisions - pos)

    # Normalitza la nova direcció dividint el vector 'nova_direccio' per la seva norma.
    # Això garantitza que la magnitud del vector sigui com a màxim a 1 però mantenint la direcció que
    # haviem calculat prèviament
    nova_direccio_norm = nova_direccio / np.linalg.norm(nova_direccio)

    # Fem un primer càlcul de la velocitat a partir del valor màxim que poden tenir les components
    # del vector i de la direcció calculada
    nova_velocitat = v_max * nova_direccio_norm

    # Aquesta funció garanteix que la velocitat estigui dins dels límits establerts per 'v_min'
    # i 'v_max'
    # També ajusta la velocitat segons la direcció de moviment representada per 'nova_direccio_norm'.
    nova_velocitat = limits_velocitat(nova_velocitat, v_min, v_max, nova_direccio_norm)

    # Actualització de la nova velocitat de l'individu
    ind.set_velocitat(nova_velocitat)


def limits_velocitat(vel, v_min, v_max, direccio):
    """
    Limita la velocitat d'un vector a un rang específic ajustant-lo als límits establerts.

    Paràmetres:
    vel (np.array): Vector de velocitat original.
    v_min (float): Valor mínim de velocitat permès.
    v_max (float): Valor màxim de velocitat permès.
    direccio (np.array): Vector de direcció del moviment.

    Retorna:
    np.array: Vector de velocitat modificat, ajustat als límits especificats.
    """
```

```
    nova_velocitat = vel

    # Calculem la magnitud de la velocitat
    nova_velocitat_magnitude = np.linalg.norm(vel)

    # Comprovem si la magnitud de la velocitat està per sota del valor mínim
    if nova_velocitat_magnitude < v_min:
        # Si és inferior, ajustem la velocitat a la direcció multiplicada pel valor mínim
        nova_velocitat = direccio * v_min

    # Comprovem si la magnitud de la velocitat està per sobre del valor màxim
    elif nova_velocitat_magnitude > v_max:
        # Si és superior, ajustem la velocitat a la direcció multiplicada pel valor màxim
        nova_velocitat = direccio * v_max

    return nova_velocitat
```

3.4.2 Actualitzar posició individu

```
def actualitzar_posicio(ind, passadis):
    """
    Actualitza la posició de l'individu en el passadís.

    Paràmetres:
    ind (objecte individu): L'individu a actualitzar la posició.
    passadis: L'objecte passadís que conté la configuració del passadís.

    """

    # Obtenir la velocitat actual de l'individu
    velocitat = ind.get_velocitat()

    # Obtenir la posició actual i el radi de l'individu
    posicio = ind.get_posicio()
    radi = ind.get_radi()

    # Calcular la nova posició
    nova_posicio = np.array(posicio) + np.array(velocitat)

    # Actualitzar la posició de l'individu
    ind.set_posicio(nova_posicio)
    passadis.ind_posicions[ind] = nova_posicio

    # Obtenim variables rellevants
    objectiu = ind.get_objectiu()
    ind.set_objectiu((nova_posicio[0], objectiu[1]))
    parets = passadis.get_parets()
    radi = ind.get_radi()

    # Verificar si l'individu ha arribat al seu objectiu
    # Calculem la distància delta com una fracció de la dimensió de la matriu
    delta = passadis.get_n() / 20

    # Verifiquem si l'objectiu de l'individu està a una distància delta a la primera fila i si
    # la nova posició en l'eix y és menor o igual a l'objectiu
```

```
if objectiu[1] == delta and nova_posicio[1] <= objectiu[1]:
    # Actualitzem la posició de l'individu
    ind.set_posicio(tuple(round(num, 2) for num in nova_posicio))

    # Eliminem la posició de l'individu
    ind.set_posicio(None)

    # Eliminem l'individu de la llista d'individus del passadís
    passadis.ind_in_passadis.remove(ind)

    # Eliminem l'individu del diccionari de posicions del passadís
    del passadis.ind_posicions[ind]

# Verifiquem si l'objectiu de l'individu està a una distància delta a l'última fila i si
# la nova posició en l'eix y és major o igual a l'objectiu
elif objectiu[1] == passadis.get_n() - delta and nova_posicio[1] >= objectiu[1]:
    # Actualitzem la posició de l'individu
    ind.set_posicio(tuple(round(num, 2) for num in nova_posicio))

    # Eliminem la posició de l'individu
    ind.set_posicio(None)

    # Eliminem l'individu de la llista d'individus del passadís
    passadis.ind_in_passadis.remove(ind)

    # Eliminem l'individu del diccionari de posicions del passadís
    del passadis.ind_posicions[ind]

# Verifiquem si la distància entre l'objectiu i la nova posició és menor o igual al
# radi de l'individu
elif np.linalg.norm(np.array(objectiu) - np.array(nova_posicio)) <= radi:
    # Actualitzem la posició de l'individu
    ind.set_posicio(tuple(round(num, 2) for num in nova_posicio))

    # Eliminem la posició de l'individu
    ind.set_posicio(None)

    # Eliminem l'individu de la llista d'individus del passadís
    passadis.ind_in_passadis.remove(ind)

    # Eliminem l'individu del diccionari de posicions del passadís
    del passadis.ind_posicions[ind]

# Si cap de les condicions anteriors és verificada
else:
    # Obtenim els valors de les coordenades de les parets
    min_x, max_x, min_y, max_y = parets

    # Corregim la posició per evitar les parets
    posicio_corregida = (max(min_x + radi, min(nova_posicio[0], max_x - radi)),
    max(min_y + radi, min(nova_posicio[1], max_y)))

    # Actualitzem la posició de l'individu
    ind.set_posicio(tuple(round(num, 2) for num in posicio_corregida))
    passadis.ind_posicions[ind] = ind.get_posicio()
```

3.5 Representació gràfica

```
def dibuixar_passadis(passadis, scale_x, scale_y, screen, clock, fps):
    """
    Dibuixa un passadís i els individus que hi ha dins en una pantalla de pygame.

    Paràmetres:
    passadis: Objecte Passadis que conté la informació del passadís i els individus dins.
    scale_x, scale_y: Factors d'escala per a les coordenades x i y, respectivament.
    Aquests s'utilitzen per ajustar les dimensions del passadís a la pantalla de pygame.

    screen: La superfície de la pantalla de pygame on es dibuixarà el passadís.
    clock: Objecte Clock de pygame que s'utilitza per controlar la velocitat de la simulació.
    fps: Enter que indica el nombre de frames per segon que es volen en la simulació.

    Aquesta funció dibuixa el passadís, les entrades i els individus a la pantalla de pygame.
    Cada individu es dibuixa com un cercle amb un color que correspon a la seva entrada.
    També es dibuixa una fletxa que indica la direcció de cada individu. La funció també gestiona els
    esdeveniments de sortida, com ara tancar la finestra o prémer la tecla ESC.
    """

    # Dimensions del passadís
    m = passadis.get_m()
    n = passadis.get_n()

    # Definim colors
    black = (0, 0, 0)
    white = (255, 255, 255)
    gray = (128, 128, 128)
    color_entrada1 = (0, 0, 255)  # Blau
    color_entrada2 = (255, 0, 0)  # Vermell

    # Asociar cada entrada amb un dels colors
    entrades = passadis.get_entrades()
    entrades_colors = {tuple(entrades[0]): color_entrada1, tuple(entrades[1]): color_entrada2}

    # Per parar la simulació quan vulguem
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                pygame.quit()
                sys.exit()

    # "Netejar" pantalla
    screen.fill(white)

    # De igual manera, al dibujar las lineas debes aplicar el factor de escala a las coordenadas
    pygame.draw.line(screen, color_entrada1, tuple(np.array(entrades[0][0]) * [scale_x, scale_y])
    ,tuple(np.array(entrades[0][1]) * [scale_x, scale_y]), 3)
    pygame.draw.line(screen, color_entrada2, tuple(np.array(entrades[1][0]) * [scale_x, scale_y])
    ,tuple(np.array(entrades[1][1]) * [scale_x, scale_y]), 3)
```

```
for individu in passadis.get_ind_in_passadis():
    pos = individu.get_posicio()

    # El individuo ha salido del mapa
    if pos is None:  continue
    # Obtener el color de la salida de este individuo
    color_individu = entrades_colors[tuple(individu.get_sortida())]
    # Obtener la posición del individuo y aplicar el factor de escala

    pos = (int(pos[0]*scale_x), int(pos[1]*scale_y))

    # Dibuja el punt de la posició en la que està l'individu
    pygame.draw.circle(screen, color_individu, pos, 1)

    # Dibuja la circunferencia
    #radius = individu.get_radi() * min(scale_x, scale_y)
    #pygame.draw.circle(screen, color_individu, pos, radius, 1)

    # Obtenim la velocitat per poder dibuixar la direcció que porta l'individu
    velocitat = individu.get_velocitat()

    if np.linalg.norm(velocitat) != 0:
        # Normalitzar la velocitat per obtindre la direcció
        direccion = velocitat / np.linalg.norm(velocitat)

        # Escalar la direcció per a que la fletxa tingui un tamany constant
        direccion *= 15

        # Calcular els punts per al triangle de la fletxa
        punta = tuple(pos + direccion)

        # Dibuixa una fletxa que indica la direcció de l'individu
        draw_arrow(screen, color_individu, pos, punta, 10)

# Actualitzar pantalla
pygame.display.flip()

# Control de velocitat de la simulació (FPS)
clock.tick(fps)
```

```python
# Dibuixa la fletxa que indica la direcció de cada individu
def draw_arrow(screen, color, start, end, arrow_head_size):
    """
    Dibuixa una fletxa que indica la direcció de l'individu a la pantalla en un entorn
    de simulació pygame.

    Paràmentres:
    screen: La superfície de la pantalla de pygame en la qual es dibuixarà la fletxa.
    color (tupla): El color de la fletxa en format RGB.
    start (dupla): Les coordenades del punt d'inici de la fletxa (x, y).
    end (dupla): Les coordenades del punt final de la fletxa (x, y).
    arrow_head_size (int): La mida de la capçalera de la fletxa en píxels.
    """

    # Diferència en l'eix x
    dx = int(end[0] - start[0])

    # Diferència en l'eix y
    dy = int(end[1] - start[1])

    # Longitud del vector de direcció
    length = math.sqrt(dx * dx + dy * dy)
    if length == 0:
        return

    # Component normalitzada en l'eix x
    udx = dx / length
    # Component normalitzada en l'eix y
    udy = dy / length

    # Angle de l'extrem de la fletxa
    arrow_head_angle = math.pi / 6

    # Punt de l'extrem de la fletxa
    arrow_tail = (end[0] - udx * arrow_head_size, end[1] - udy * arrow_head_size)

    # Càlcul de les coordenades de l'extrem esquerre de la fletxa
    arrow_left = (
        # Coordenada x de l'extrem esquerre de la fletxa
        arrow_tail[0] - udy * arrow_head_size * math.tan(arrow_head_angle / 2),
        # Coordenada y de l'extrem esquerre de la fletxa
        arrow_tail[1] + udx * arrow_head_size * math.tan(arrow_head_angle / 2),
    )
    # Càlcul de les coordenades de l'extrem dret de la fletxa
    arrow_right = (
        # Coordenada x de l'extrem dret de la fletxa
        arrow_tail[0] + udy * arrow_head_size * math.tan(arrow_head_angle / 2),
        # Coordenada y de l'extrem dret de la fletxa
        arrow_tail[1] - udx * arrow_head_size * math.tan(arrow_head_angle / 2),
    )
    # Dibuixa la línia des de l'inici fins a l'extrem de la fletxa
    pygame.draw.line(screen, color, start, arrow_tail, 1)

    # Dibuixa el triangle de la fletxa
    pygame.draw.polygon(screen, color, [end, arrow_left, arrow_right])
```

3.6 Simulació

```
def simulacio(passadis, num_iteracions, aforament, escalat_pixel, fps):
    """
    Simula el moviment d'individus en un passadís 2D per un nombre determinat d'iteracions.

    Paràmetres:
    passadis: Objecte Passadis que conté la informació del passadís i els individus dins.
    num_iteracions: Nombre d'iteracions de la simulació.
    aforament: Nombre màxim d'individus que pot haver-hi al passadís.
    escalat_pixel: Unitat de tamany base dels píxels de la simulació
    fps: Enter que indica el nombre de frames per segon que es volen en la simulació.

    La funció gestiona tota la simulació: crea els individus, els mou i els dibuixa.
    """

    # Obtenim les dimensions i les entrades del passadís
    m = passadis.get_m()
    n = passadis.get_n()
    entrades = passadis.get_entrades()

    # Iniciem Pygame
    pygame.init()

    # Obtenim les dimensions de la pantalla del dispositiu
    screen_info = pygame.display.Info()
    screen_width = screen_info.current_w
    screen_height = screen_info.current_h

    # Calculem les dimensions de la finestra de la simulació
    screen_width = m * escalat_pixel
    screen_height = n * escalat_pixel

    # Creem la finestra de la simulació
    screen = pygame.display.set_mode((screen_width, screen_height))

    # Calculem els factors d'escala per a les dimensions x i y
    scale_x = screen_width / m
    scale_y = screen_height / n

    # Creem un rellotge per controlar la velocitat d'actualització de la pantalla
    clock = pygame.time.Clock()

    # Definim els paràmetres dels individus
    id_individu = 0
    v_min = 0.02
    v_max = 0.2
    radi = 0.3
    temps_horitzo = 3
    delta = n/20

    # Inicialitzem les llistes per als individus i les seves velocitats
    individus = []
    velocitats = {}

    # Iniciem la simulació
    for t in range(num_iteracions):
```

```
if t % 2 == 0:
    # Calculem el nombre d'individus nous a afegir
    nous_individus = random.randint(0, (aforament-len(passadis.ind_in_passadis))//10)
    if nous_individus > int(m/(2*radi)) - 1: nous_individus = int(m/(2 * radi)) - 1

    # Calculem les possibles posicions d'entrada
    posicions_entrades = ui.calcul_posicions_entrada(m, n, radi, delta)

    # Creem els nous individus i els afegim a les llistes corresponents
    for j in range(1, nous_individus + 1):
        # Calculem l'id del nou individu
        id_individu += 1

        # Calculem l'entrada de l'individu
        index_entrada = random.randrange(len(posicions_entrades))
        entrada = posicions_entrades[index_entrada]

        # Eliminem aquesta entrada de les possibles entrades per a no posar dos
        # individus en la mateixa entrada al mateix temps
        posicions_entrades.pop(index_entrada)

        # Calculem la sortida i l'objectiu de l'individu
        sortida = ui.calcul_sortida(entrada, entrades)
        objectiu = ui.calcul_objectiu(sortida)

        # Classifiquem a l'individu segons el seu objectiu
        if objectiu[1] == delta: grup = 0
        else: grup = 1

        # Calculem la velocitat preferida de l'individu
        velocitat = ui.calcul_velocitat_inicial(v_min, v_max)

        # Creem el nou individu
        individu = ci.Individu(id_individu, entrada, sortida, objectiu, grup,
        v_min, v_max, velocitat, m, n, radi, temps_horitzo)

        # Afegim l'individu a les llistes corresponents
        individus.append(individu)
        passadis.ind_in_passadis.append(individu)
        passadis.ind_posicions[individu] = entrada
        velocitats[individu] = []

# Aquest bucle gestiona el moviment dels individus
# Actualitzem la velocitat i la posició de cada individu en el passadís
for ind in passadis.get_ind_in_passadis():
    mov.calcul_nova_velocitat(ind, passadis.get_ind_in_passadis())
    mov.actualitzar_posicio(ind, passadis)
    velocitats[ind].append(ind.get_velocitat())

# Mostrem el nombre d'individus en el passadís i en total en cada iteració
print(f"Quantitat invididus en t = {t} = {len(passadis.ind_in_passadis)}\n")
print(f"Total invididus en t = {t} = {len(individus)}\n")

# Dibuixem el passadís en cada iteració
dp.dibuixar_passadis(passadis, scale_x, scale_y, screen, clock, fps)
```

4 Gestió dels paràmetres inicials i execució del codi

En aquest apartat s'expliquen els paràmentres que inicien el programa en el model discret, model continu i en l'algorisme genètic de forma que l'usuari pugui entendre quin tipus de valors s'esperen i quina utilitat tenen. Així podrà fer les proves que cregui convenients i realitzar simulacions en els tres casos. A més, s'explica com executar fitxers de python en linux, mac i windows.

Els paràmentres per iniciar la simulació en el model discret es troben en el fitxer **'main.py'** dins de la carpeta **'Model Discret'**.

4.1 Model discret

```
# Obté el tamany de la pantalla del dispositiu on s'executi el programa
pygame.init()
screen_info = pygame.display.Info()
screen_width = screen_info.current_w * 0.95
screen_height = screen_info.current_h * 0.95

# Crear una instància de la classe Passadis amb els paràmetres donats
"""
Paràmentres algorisme genètic
- Identificador únic del passadís (si només es crea un passadís pot ficar sempre 0)
- Valor de m (files)
- Valor de n (columnes)
- Amplada de la entrada (quanitat de cel·les que ocupa cada entrada quan entrada_unica és False)
- Entrada única: és una variable booleana que pot valdre True o False
- Entrades laterals: és una variable booleana que pot valdre True o False.
Si entrada única es True no hi hauràn entrades laterals.
- Obstacles: és una variable booleana que pot valdre True o False
"""
passadis = cp.Passadis(0, 40, 25, 1, True, False, False)

# Executar l'algoritme genètic amb el passadís creat i els paràmetres donats
"""
Paràmentres algorisme genètic
- Passadís
- Nombre de iteracions de la simulació
- Aforament
- Fotogrames per segon
- Amplada de la pantalla
- Alçada de la pantalla
"""
sim.simulacio(passadis, 250, 400, 15, screen_width, screen_height)
```

Els paràmentres per iniciar l'algorisme genètic es troben al final del fitxer **'genetic.py'** dins de la carpeta **'Model Discret'**.

4.2 Algorisme genètic

```
# Crear una instància de la classe Passadis amb els paràmetres donats
"""
Paràmentres algorisme genètic
- Identificador únic del passadís (si només es crea un passadís pot ficar sempre 0)
- Valor de m (files)
- Valor de n (columnes)
- Amplada de la entrada (quanitat de cel·les que ocupa cada entrada quan entrada_unica és False)
- Entrada única: és una variable booleana que pot valdre True o False
- Entrades laterals: és una variable booleana que pot valdre True o False
- Obstacles: és una variable booleana que pot valdre True o False
"""
passadis = cp.Passadis(0, 25, 15, 6, True, False, False)

# Executar l'algoritme genètic amb el passadís creat i els paràmetres donats
"""
Paràmentres algorisme genètic
- Passadís
- Quantitat de generacions
- Nombre de iteracions per generació
- Aforament
- Fotogrames per segon
"""
algorisme_genetic(passadis, 10, 250, 50, 40)
```

Els paràmentres per iniciar la simulació en el model continu es troben en el fitxer **'main.py'** dins de la carpeta **'Model Continu'**.

4.3 Model continu

```
"""
Quant més petit és escalat pixel més grans podem fer els passadissos.
Es recomana mantenir l'escalat per sobre de 30 per a una bona representació.

Es recomana mantenir m entre [5, 45]
Es recomana mantenir n entre [10, 35]
"""
 # Unitat de tamany base dels píxels de la simulació
escalat_pixel = 40

# Fotogrames per segon de la simulació
fps = 15

# Durada de la simulació i quantitat màxima d'individus en el passadís
iteracions = 1000
aforament = 100

# Creació del passadís
passadis = cp.Passadis(0, 15, 25)

# Executem la simulació
simulacio(passadis, iteracions, aforament, escalat_pixel, fps)
```

4.4 Execució d'un fitxer python en Linux o Mac

Per executar un fitxer de Python en Linux o Mac, primerament assegureu-vos que Python està instal·lat al vostre sistema. Podeu verificar-ho obrint un terminal i escrivint python3 –version o python –version (depenent de la versió de Python). Això hauria de mostrar la versió de Python instal·lada.

Per executar un fitxer, aneu al directori on es troba el fitxer Python (.py) utilitzant la comanda cd. Per exemple, si el vostre fitxer es diu 'main.py' i es troba al directori 'Documentos', feu: **'cd Documentos'**

Un cop estigueu al directori correcte, podeu executar el fitxer de Python amb la següent comanda: **python main.py** o **python3 main.py**

4.5 Execució d'un fitxer python en Windows

Al igual que en Linux, primerament assegureu-vos que Python està instal·lat al vostre sistema. Podeu verificar-ho obrint un Símbol del sistema i escrivint python –version. Això hauria de mostrar la versió de Python instal·lada.

Per executar un fitxer, primerament haureu de navegar fins al directori on es troba el fitxer Python (.py). Podeu fer-ho utilitzant la comanda cd al Símbol del sistema o PowerShell. Per exemple, si el vostre fitxer es diu 'script.py' i es troba al directori 'Documents', feu: **'cd Documentos'**

Un cop estigueu al directori correcte, podeu executar el fitxer de Python amb la següent comanda: **python main.py**