

Modélisation microscopique d'une foule & application à l' évacuation d'un lycée.

Comment modéliser l'
évacuation d'une foule ?

Tom GILGENKRANTZ
MP - Option Informatique
Numéro SCEI : **47269**

Table des matières

I) Modélisation

II) 1^{ère} tentative échouée et difficultés de la modélisation

III) 2nd tentative : mon modèle d'automate cellulaire

IV) Application à l'évacuation de mon lycée

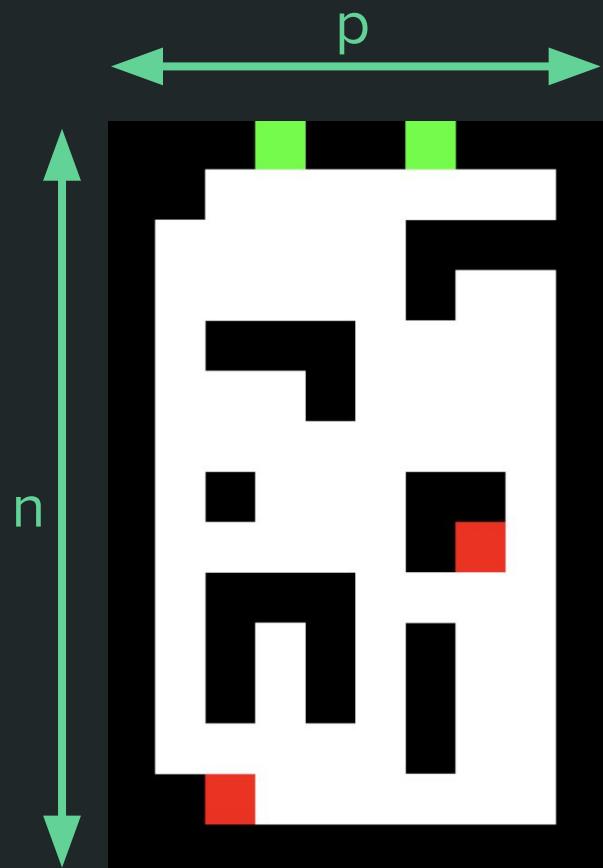
V) Comparaison avec un modèle existant

I) Modélisation

Image au format **PNG**

- Dimensions $(n,p) \in \mathbb{N}^2$
- Obstacles
- Sorties (vert)
- Escaliers (rouge)

Salle obligatoirement **connexe**



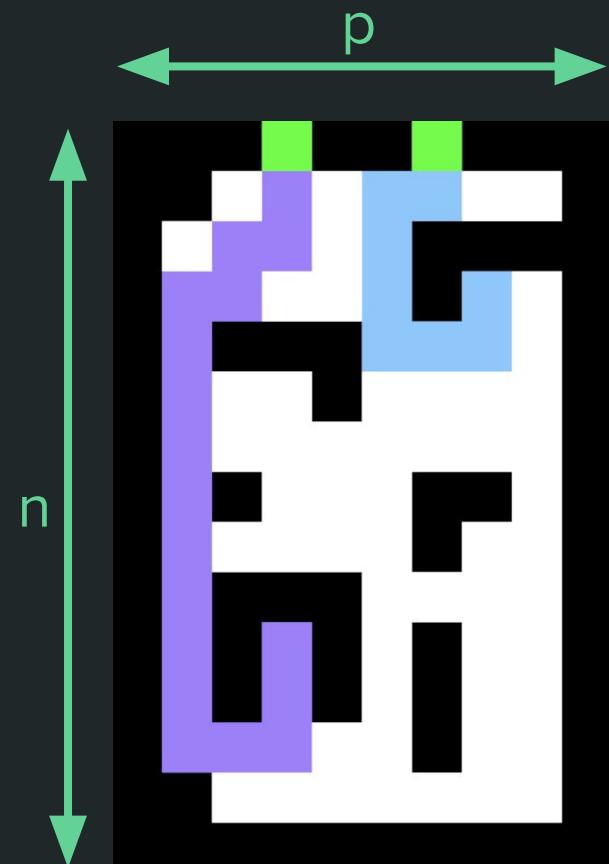
I) Modélisation

Format chemin :

- Chemin sous la forme d'une liste
 - Longueur $m \in \mathbb{N}$
 - Eléments $(x_i, y_i)_{i \in \llbracket 1, m \rrbracket} \in \llbracket 1, n \rrbracket \times \llbracket 1, p \rrbracket$

Recherche du plus court chemin :

⇒ Dijkstra

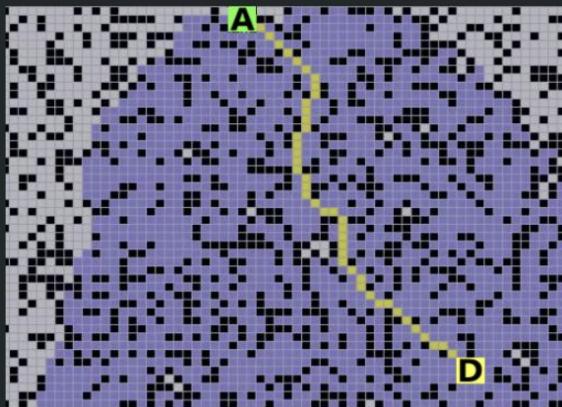


I) Modélisation



Dijkstra trop peu efficace

⇒ A-étoile avec fonction heuristique (distance euclidienne)



Dijkstra A-étoile

Pour n sommets & p arrêtes :

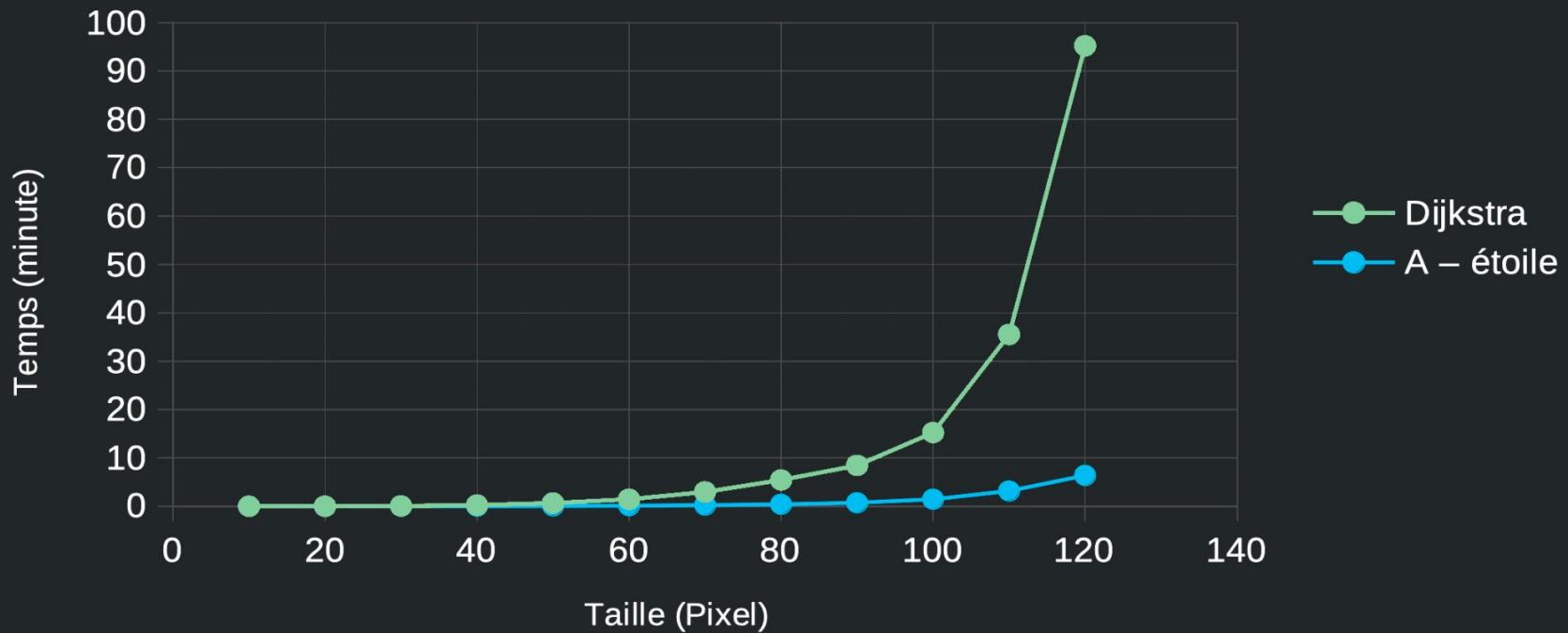
$O(p + \log(n))$

$O(n)$

I) Modélisation

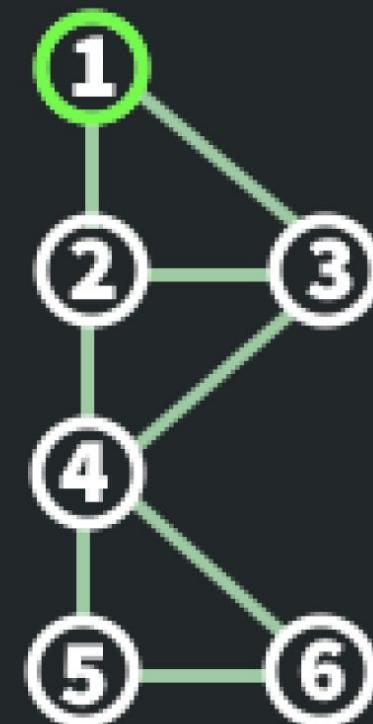
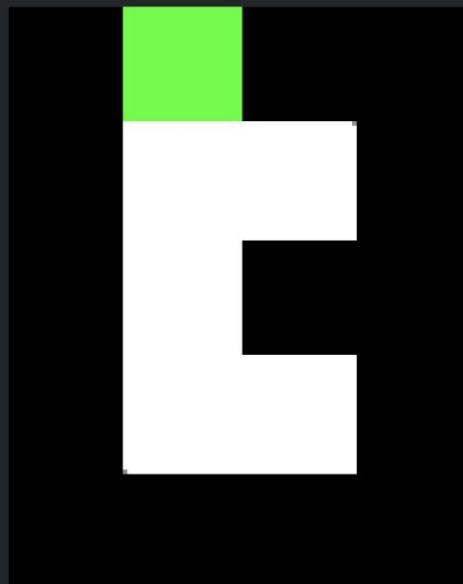
Comparaison Dijkstra / a-étoile

(Temps d'exécution en fonction du côté d'une salle carrée aléatoire)



I) Modélisation

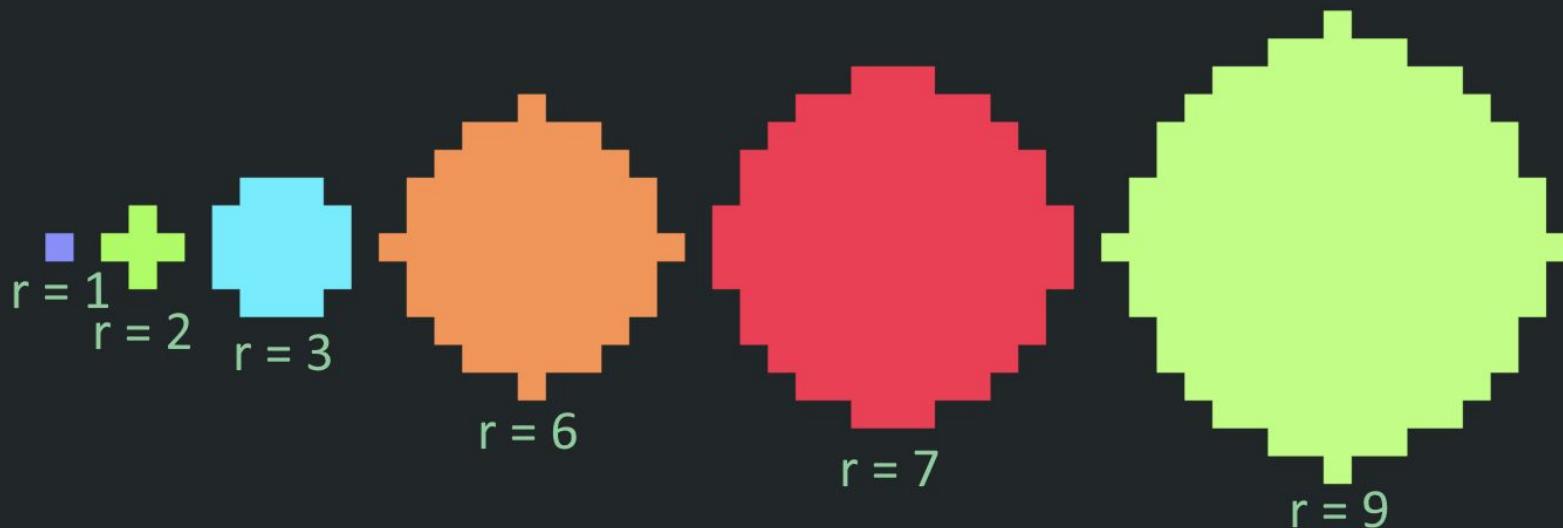
Conversion image vers graphe



II) 1^{ère} tentative

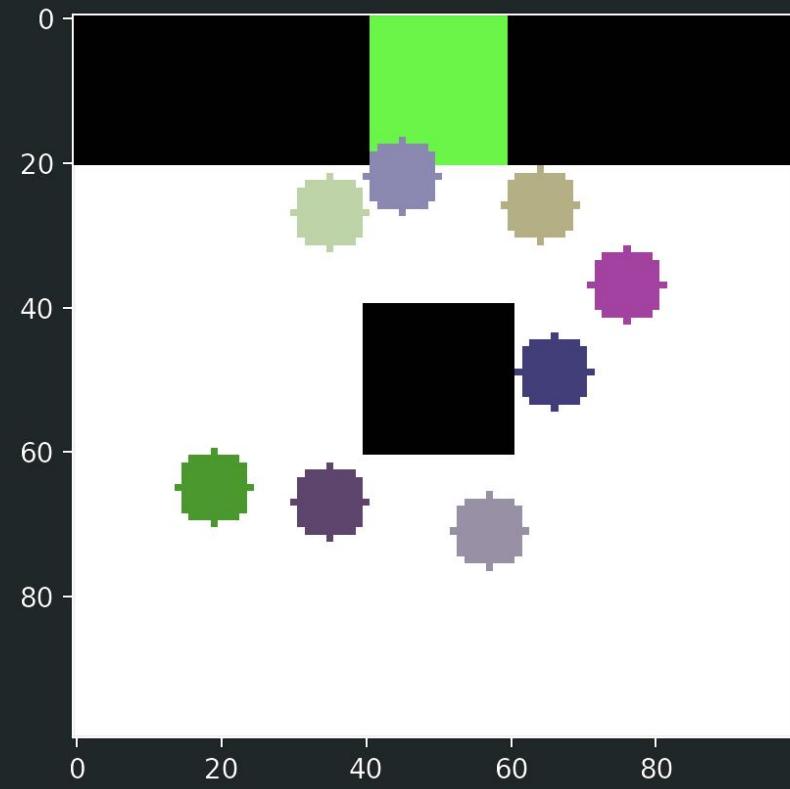
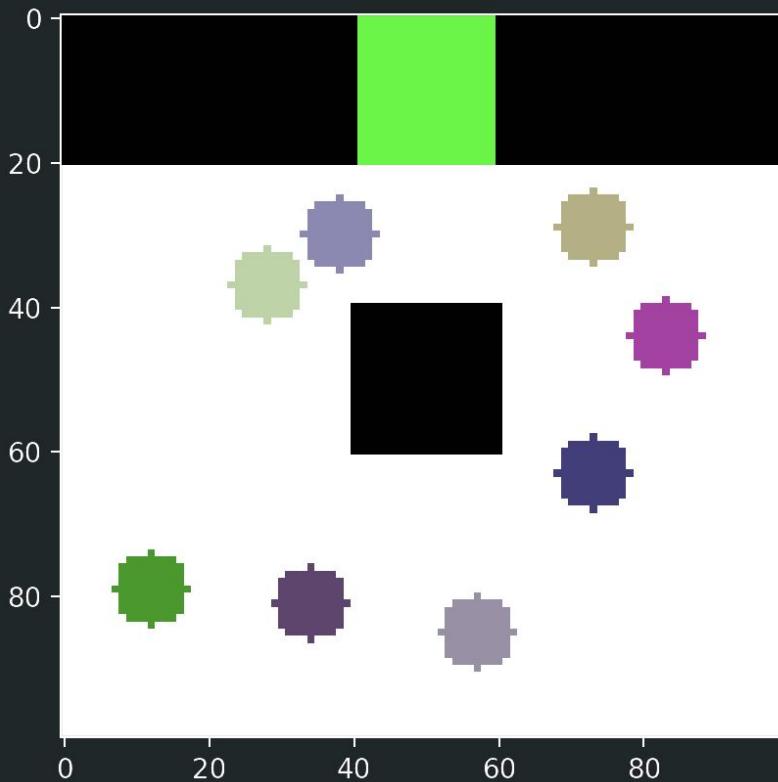
Modélisation des individus “sphériques”

- Rayon : $r \in \mathbb{N}$
- Centre : $(x,y) \in [[r,n-r]] \times [[r,p-r]]$
- Couleur RGB : $(a,b,c) \in [[0,255]]^3$



II) 1^{ère} tentative

⚠ Problème de **complexité**

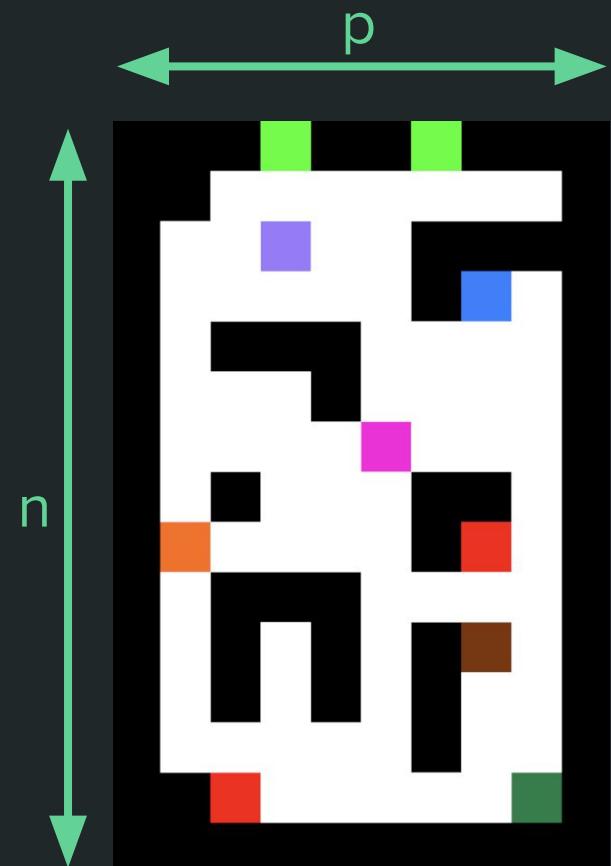


III) Mon modèle d'automate cellulaire

Modélisation des individus :

- Position : $(x,y) \in [[1,n]] \times [[1,p]]$
- Distance à la sortie la plus proche : $d \in \mathbb{R}$
- Couleur RGB : $(a,b,c) \in [[0,255]]^3$

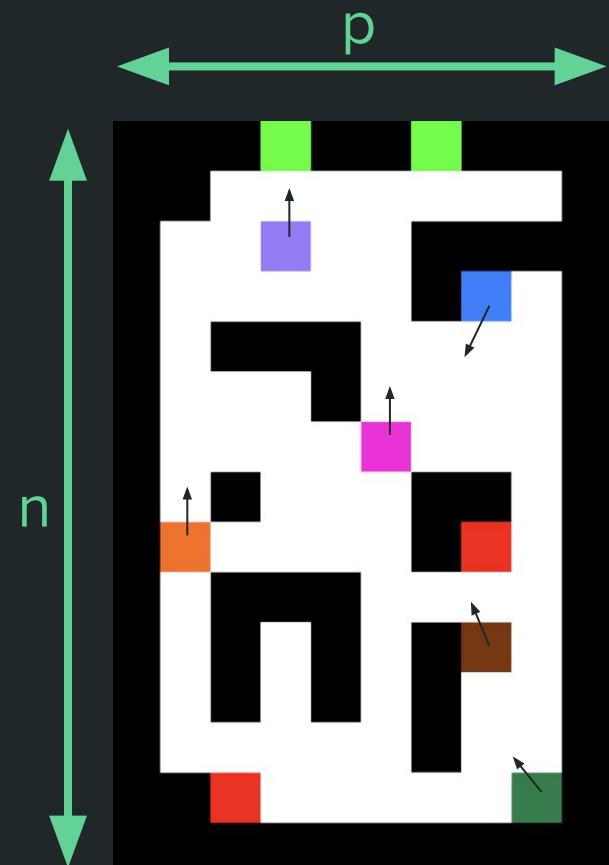
Liste d'individus **triée par distance à la sortie** croissante



III) Mon modèle d'automate cellulaire

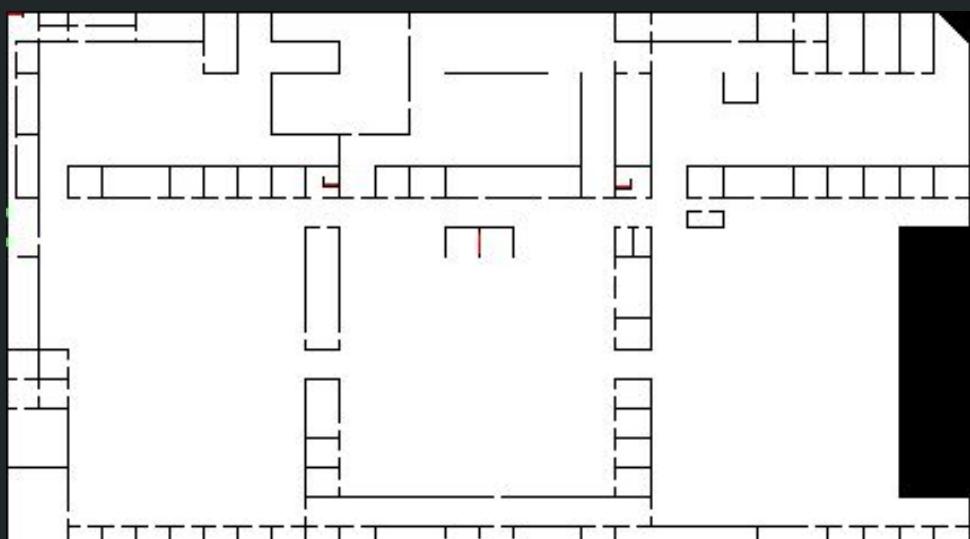
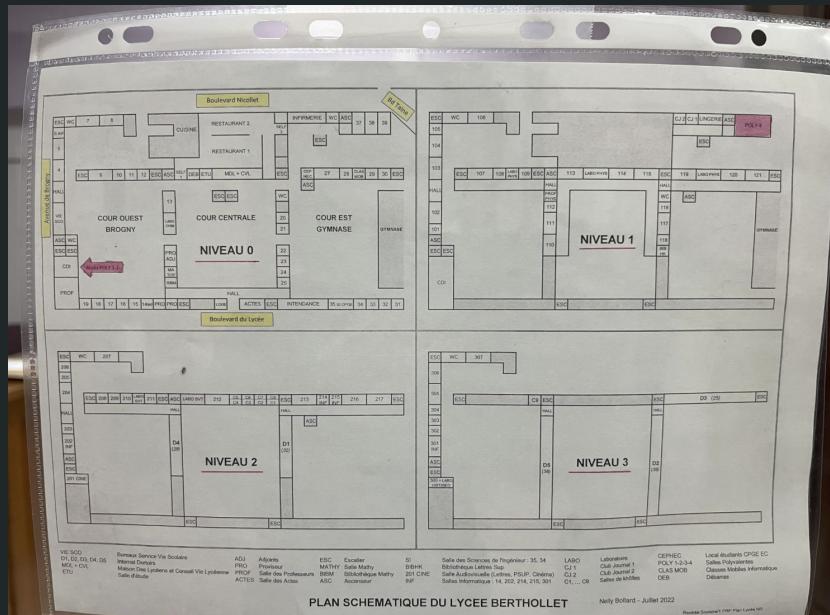
Comportement des individus :

- Prise de décision individuelle
- Recherche du chemin le plus court
- Déplacement pixel par pixel (diagonales possibles)
- Chevauchement d'individus impossible
- Pénétration d'obstacle interdite



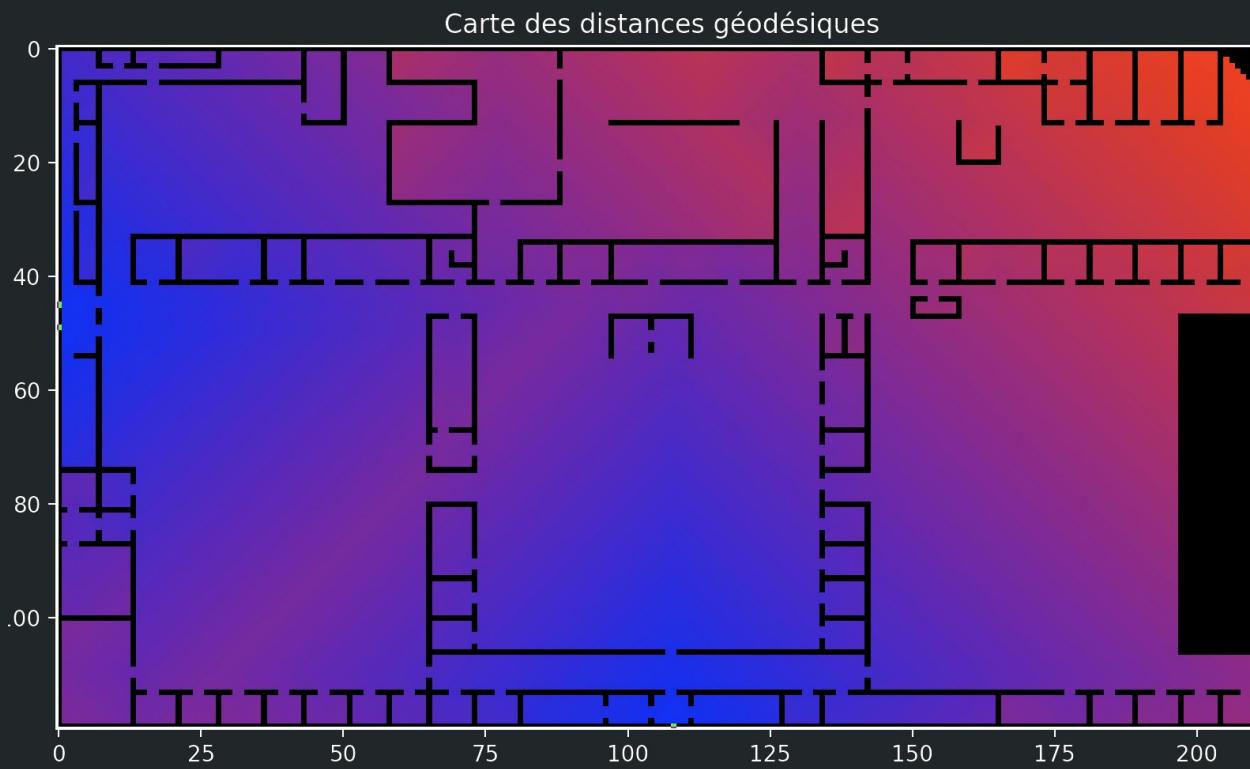
IV) Évacuation de mon lycée

Modélisation de l'environnement



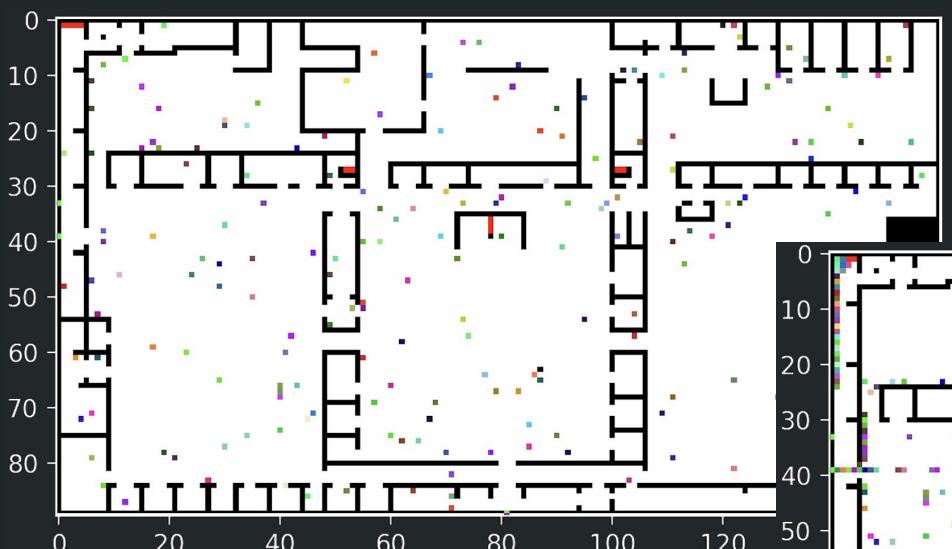
IV) Évacuation de mon lycée

Tracé de la carte des distances géodésiques ou
“distances à la sortie” :

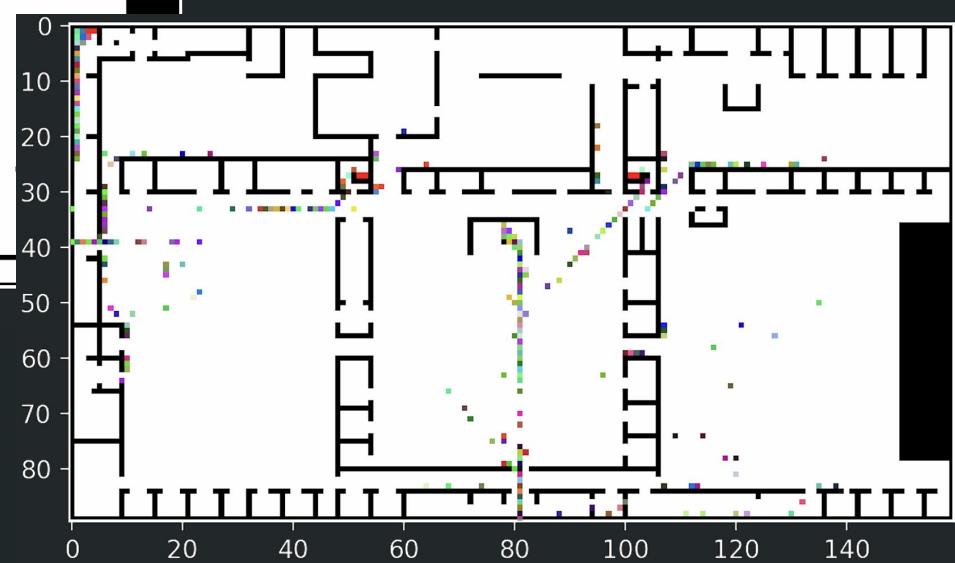


IV) Évacuation de mon lycée

Résultats $t = 0$

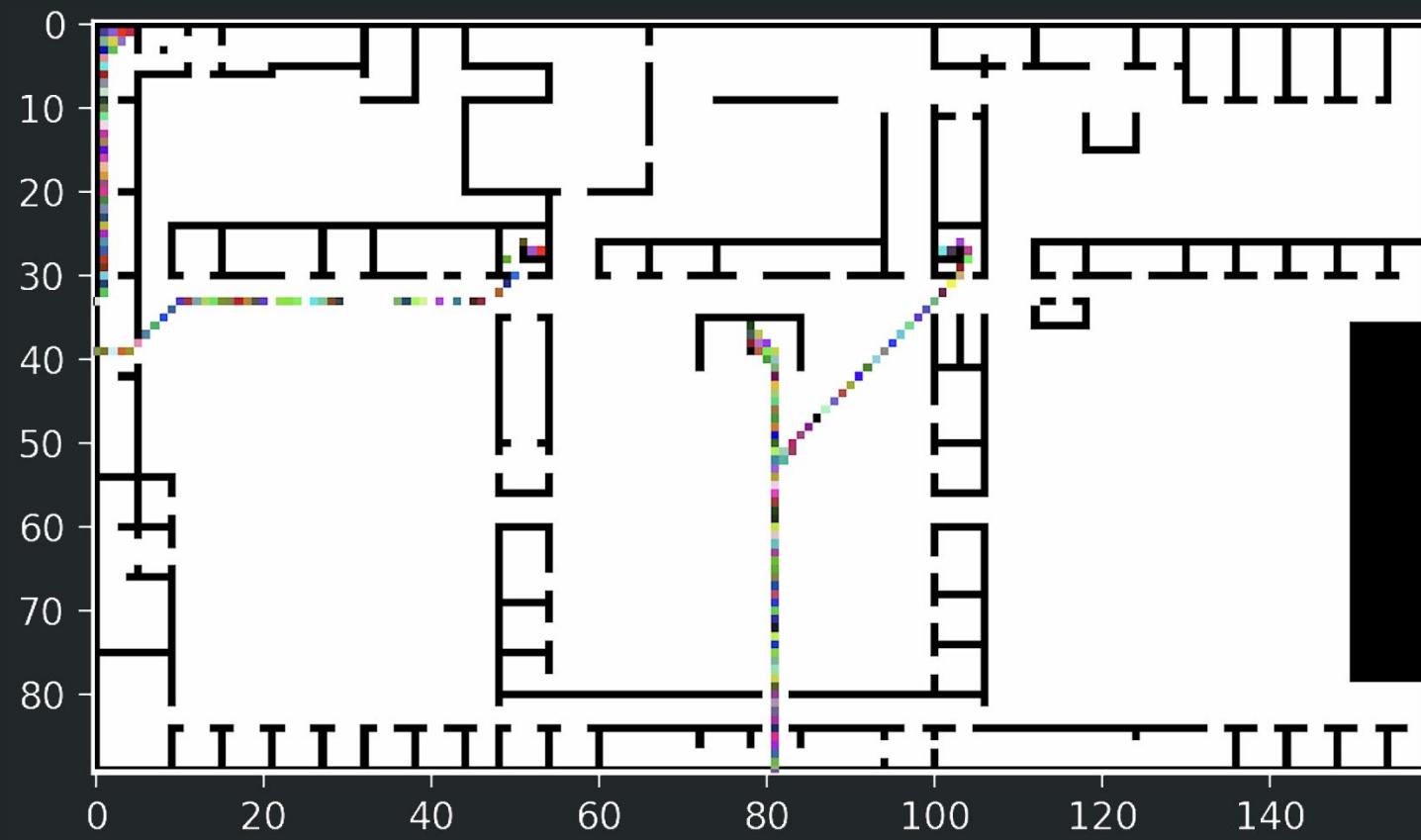


$t = 200$



IV) Évacuation de mon lycée

Conclusions et limites $t = 400$



IV) Évacuation de mon lycée

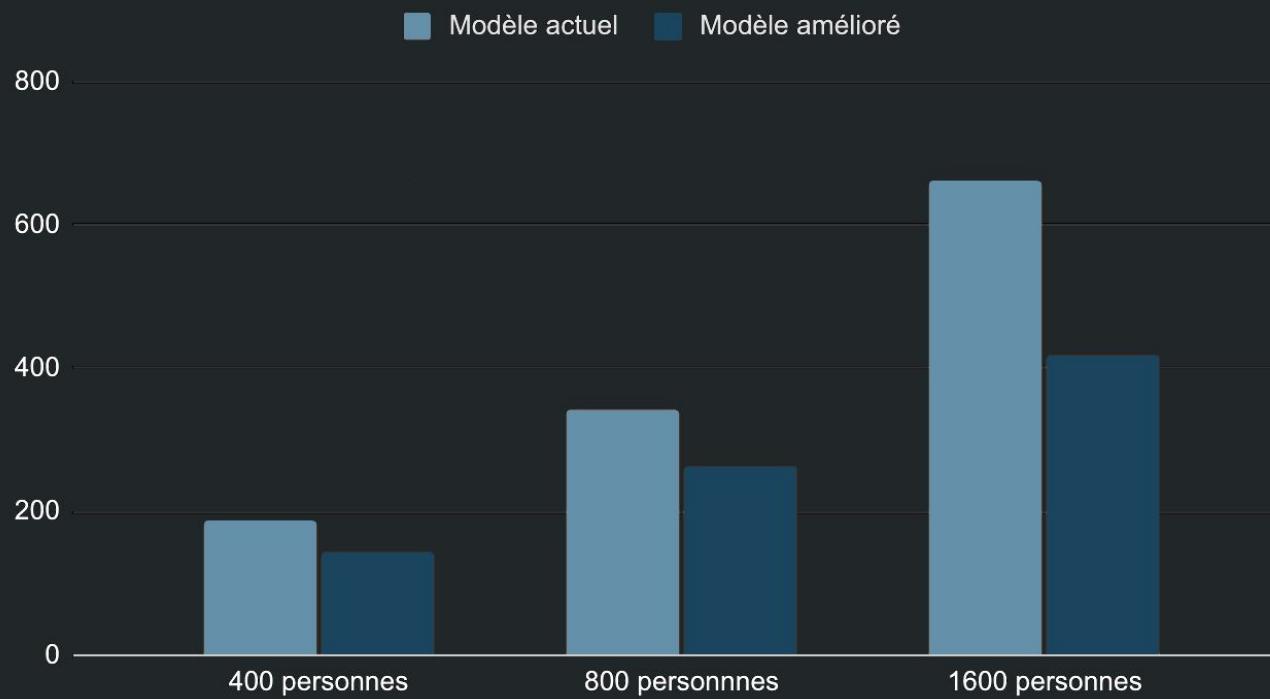
Solution proposée



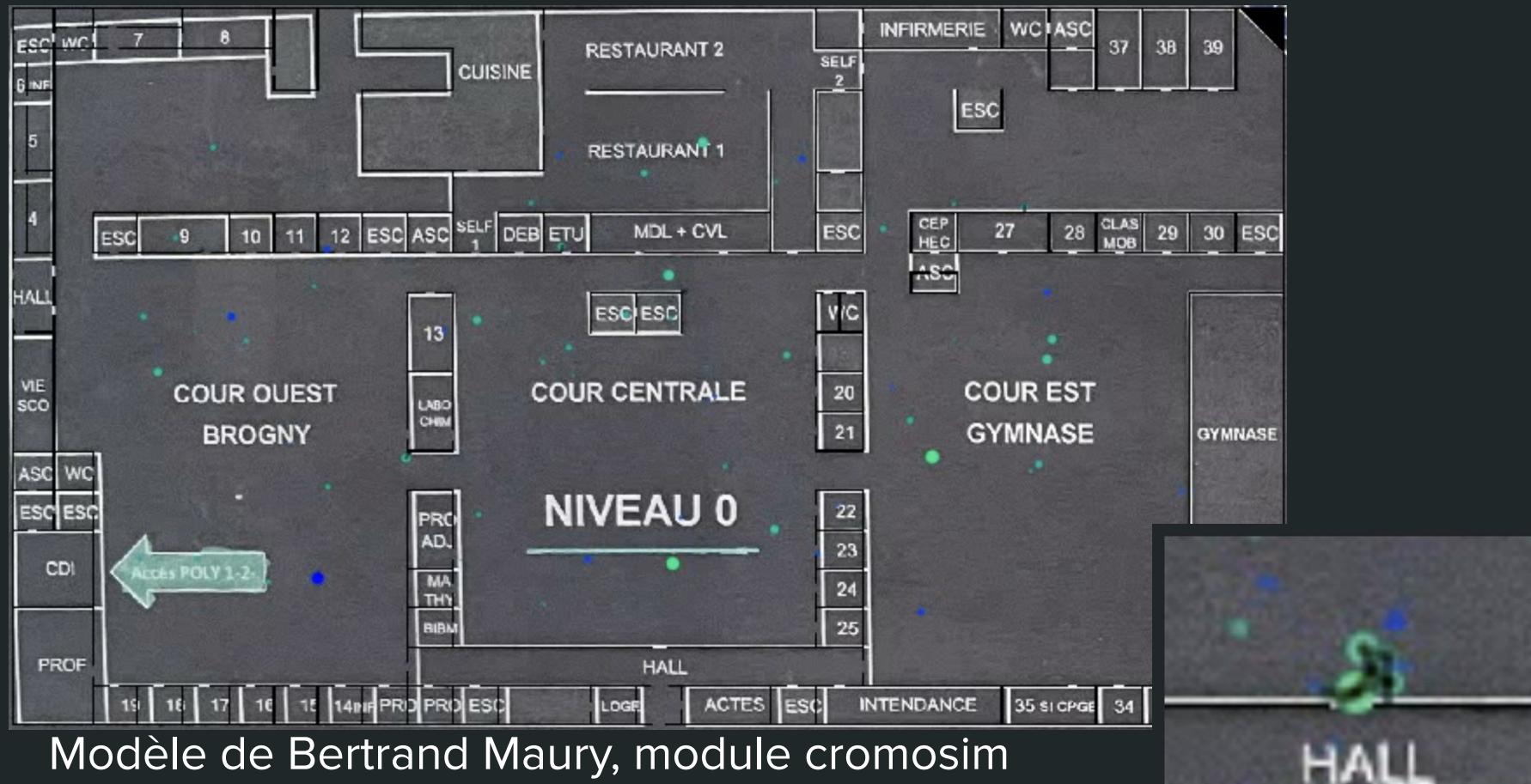
IV) Évacuation de mon lycée

Test de cette solution

Temps d'évacuation en fonction du nombre de personnes



V) Comparaison avec un modèle existant



Modèle de Bertrand Maury, module cromosim

IV) Questions

Septembre 2022 : recherches bibliographiques sur les modélisations actuelles de la foule, découverte du module Opensource "chromosim".

Octobre - Novembre 2022 : modélisation de l'environnement, travail sur dijkstra puis sur A - étoile.

Décembre 2022 : essai raté de modélisation qui m'a incité à trouver des programmes beaucoup plus efficaces.

Janvier 2022 : développement de mon modèle d'automate cellulaire avec succès.

Février 2022 : tests et utilisations de mon modèle d'automate cellulaire.

Mars 2022 : utilisation du module chromosim puis comparaisons avec mon modèle.

IV) Conclusion

Merci pour votre attention

IV) Annexe

```
19 # Implémentation de A* :
20 def A_étoile(salle, départ, arrivées):
21     # Introduction des variables
22     distances = [[float('inf') for j in range(len(salle[0]))] for i in range(len(salle))]
23     distances[départ[0]][départ[1]] = 0
24     f_score = [[float('inf') for j in range(len(salle[0]))] for i in range(len(salle))]
25     f_score[départ[0]][départ[1]] = distance_heuristique(départ, arrivées[0])
26     came_from = [[None for j in range(len(salle[0]))] for i in range(len(salle))]
27     visités = [[False for j in range(len(salle[0]))] for i in range(len(salle))]
28     queue, chemins = [], {}
29     heapq.heappush(queue, (f_score[départ[0]][départ[1]], départ))
30     for arrivée in arrivées:
31         chemins[arrivée] = None
32     # Algorithme
33     while queue and not all(value is not None for value in chemins.values()):
34         f_actuel, actuel = heapq.heappop(queue)
35         visités[actuel[0]][actuel[1]] = True
36         if actuel in arrivées:
37             chemin = [actuel]
38             act = actuel
39             while act != départ:
40                 act = came_from[act[0]][act[1]]
41                 chemin.append(act)
42             chemin.reverse()
43             chemins[actuel] = chemin
44             if all(value is not None for value in chemins.values()):
45                 break
46         for voisin in voisins(salle, actuel):
47             if not visités[voisin[0]][voisin[1]]:
48                 dist_voisin = distances[actuel[0]][actuel[1]] + 1
49                 if dist_voisin < distances[voisin[0]][voisin[1]]:
50                     distances[voisin[0]][voisin[1]] = dist_voisin
51                     f_score[voisin[0]][voisin[1]] = min([dist_voisin + distance_heuristique(voisin, arrivée) for arrivée in arrivées])
52                     came_from[voisin[0]][voisin[1]] = actuel
53                     heapq.heappush(queue, (f_score[voisin[0]][voisin[1]], voisin))
54     return chemins
55
56 def distance_heuristique(noeud, arrivee):
57     return abs(noeud[0] - arrivee[0]) + abs(noeud[1] - arrivee[1])
```

IV) Annexe

```
59 def dijkstra(salle, start, sorties):
60     # Introduction des variables
61     distances = { (i, j): float('inf') for i in range(len(salle)) for j in range(len(salle[0])) }
62     distances[start] = 0
63     chemins = {sortie: [] for sortie in sorties}
64     file_priorite = [(0, start)]
65     # Algorithme
66     while file_priorite:
67         distance_actuelle, noeud_actuel = heapq.heappop(file_priorite)
68         # Si le nœud actuel est une sortie, ajouter le chemin à la liste des chemins
69         if noeud_actuel in sorties:
70             chemin = [noeud_actuel]
71             while chemin[-1] != start:
72                 for voisin in voisins(salle,chemin[-1]):
73                     if distances[voisin] == distances[chemin[-1]] - 1:
74                         chemin.append(voisin)
75                         break
76             chemins[noeud_actuel] = chemin[::-1] # Explorer les voisins du nœud actuel
77         for voisin in voisins(salle,noeud_actuel):
78             nouvelle_distance = distance_actuelle + 1
79             if nouvelle_distance < distances[voisin]:
80                 distances[voisin] = nouvelle_distance
81                 heapq.heappush(file_priorite, (nouvelle_distance, voisin))
82     return chemins
83
84 def voisins(salle,point):
85     voisins = [(point[0]-1,point[1]),(point[0]+1,point[1]),(point[0],point[1]-1),(point[0],point[1]+1),(point[0]-1,point[1]-1),
86                 (point[0]+1,point[1]-1),(point[0]-1,point[1]+1),(point[0]+1,point[1]+1)]
87     accessibles = []
88     for voisin in voisins:
89         if 0 <= voisin[0] < len(salle) and 0 <= voisin[1] < len(salle[0]) and salle[voisin[0]][voisin[1]] == 0:
90             accessibles.append(voisin)
91     return accessibles
92
```

IV) Annexe

```
93 def créer_dictionnaire_chemin(mode,salle,sorties,escaliers,fichier,algo) : # 0:pas d'affichage ; 1:affichage rapide
94     points , dictionnaire = point_accessible(salle) , {}
95     for i in tqdm(range(len(points)),mininterval=16) :
96         if algo == "A" :
97             dictionnaire_chemin = A_étoile(salle, points[i], sorties)
98         elif algo == "D" :
99             dictionnaire_chemin = dijkstra(salle, points[i], sorties)
100         mini = next(iter(dictionnaire_chemin))
101         for chemin in dictionnaire_chemin :
102             if len(dictionnaire_chemin[chemin]) < len(dictionnaire_chemin[mini]) :
103                 mini = chemin
104             dictionnaire[points[i]] = [mini,dictionnaire_chemin[mini]]
105             afficher_salle_chemins(salle,sorties,escaliers,dictionnaire_chemin,mode)
106             sauvegarder(dictionnaire,fichier) # Dictionnaire au format {point : [sortie la plus proche,[chemin]],...}
107             print("Dictionnaire créé avec succès.")
108         return dictionnaire
109 #####
110
111
112
113 #####
114 # Fonctions d'affichage :
115 def afficher_salle(salle, sorties,escaliers,mode):
116     if mode == 0 :
117         return
118     img = np.zeros((len(salle),len(salle[0]), 3))
119     for i in range(len(salle)):
120         for j in range(len(salle[0])):
121             if salle[i][j] == 0:
122                 img[i][j] = [1, 1, 1] # zone accessible en blanc
123             else:
124                 img[i][j] = [0, 0, 0] # obstacle en noir
125     for i, j in sorties:
126         img[i][j] = [0, 1, 0] # sorties en vert
127     for i, j in escaliers:
128         img[i][j] = [1, 0, 0] # escaliers en rouge
129     plt.imshow(img)
130     plt.show(block=False)
131     if mode == 1 :
132         plt.pause(1)
133         plt.close()
134     return
```

IV) Annexe

```
136 def carte_distances_géodésiques(salle,sorties,dictionnaire) :
137     maxi , image = len(dictionnaire[max_dictionnaire(dictionnaire)][1]) , np.zeros((len(salle),len(salle[0]), 3))
138     for i in range(len(salle)):
139         for j in range(len(salle[0])):
140             if (i,j) in dictionnaire :
141                 couleur = 1 - len(dictionnaire[(i,j)][1]) / maxi
142                 image[i][j] = [1-couleur,0,couleur]
143     for sortie in sorties :
144         image[sortie[0]][sortie[1]] = [0,1,0]
145     plt.imshow(image)
146     plt.title("Carte des distances géodésiques")
147     plt.show()
148     return image
149
150 def afficher_salle_chemins(salle, sorties, escaliers, chemins , mode):
151     if mode == 0 :
152         return
153     img = np.zeros((len(salle),len(salle[0]), 3))
154     for i in range(len(salle)):
155         for j in range(len(salle[0])):
156             if salle[i][j] == 0:
157                 img[i][j] = [1, 1, 1] # zone accessible en blanc
158             else:
159                 img[i][j] = [0, 0, 0] # obstacle en noir
160     for i, j in sorties:
161         img[i][j] = [0, 1, 0] # sorties en vert
162     for i, j in escaliers:
163         img[i][j] = [1, 0, 0] # escaliers en rouge
164     colors = plt.cm.rainbow(np.linspace(0, 1, len(chemins)))
165     for i, ((x, y), chemin) in enumerate(chemins.items()):
166         for xi, xj in chemin:
167             img[xi][xj] = colors[i][:3]
168     plt.imshow(img)
169     plt.show(block=False)
170     if mode == 1 :
171         plt.pause(1)
172         plt.close()
173     return
```

IV) Annexe

```
175 def afficher_salle_personnes(salle, sorties, escaliers, personnes, mode):
176     if mode == 0 :
177         return
178     img = np.zeros((len(salle),len(salle[0]), 3))
179     for i in range(len(salle)):
180         for j in range(len(salle[0])):
181             if salle[i][j] == 0:
182                 img[i][j] = [1, 1, 1] # zone accessible en blanc
183             else:
184                 img[i][j] = [0, 0, 0] # obstacle en noir
185     for i, j in sorties :
186         img[i][j] = [0, 1, 0] # sorties en vert
187     for i, j in escaliers:
188         img[i][j] = [1, 0, 0] # escaliers en rouge
189     for personne in personnes :
190         img[personne[0][0]][personne[0][1]] = personne[2] # personnes
191     if mode == 4 :
192         return img
193     plt.imshow(img)
194     plt.show(block=False)
195     if mode == 1 :
196         plt.pause(1)
197         plt.close()
198     return
199
200 def créer_vidéo(dictionnaire,salle,sorties,escaliers,personnes,nb_étages,nb_personnes,FPS,nom):
201     liste_images = []
202     liste_images.append(afficher_salle_personnes(salle,sorties,escaliers,personnes,4))
203     while len(personnes) != 0 :
204         nb_personnes, nouvelles_personnes = gestion_étages(personnes,escaliers,nb_personnes,dictionnaire)
205         ajouter_trier_personnes(personnes,nouvelles_personnes)
206         bouger_personnes(personnes,dictionnaire)
207         liste_images.append(afficher_salle_personnes(salle,sorties,escaliers,personnes,4))
208     fig, ax = plt.subplots()
209     im = ax.imshow(liste_images[0])
210     def update_image(frame):
211         im.set_array(liste_images[frame])
212         return im,
213     anim = animation.FuncAnimation(fig, update_image, frames=len(liste_images), interval=FPS)
214     anim.save(nom, writer='pillow')
215     plt.show()
216     return
```

IV) Annexe

```
218 def afficher_liste(liste) :
219     chaine_caractère = ""
220     for element in liste :
221         chaine_caractère = chaine_caractère + "\n" + str(element)
222     return chaine_caractère
223 #####
224
225
226
227 #####
228 # Création et gestion des personnes :
229 def placer_personnes(nombre,salle,dictionnaire) : # renvoie une liste de personnes triée par distance à la sortie
230     personnes, compteur = [] , 0
231     while compteur != nombre :
232         point = point_aleatoire(salle)
233         if point not in personnes :
234             personnes.append([point,len(dictionnaire[point][1]),[random(),random(),random()]])
235             compteur += 1
236     return personnes
237
238 def ajouter_trier_personnes(liste_personnes,nouvelles_personnes) :
239     for personne in nouvelles_personnes :
240         liste_personnes.append(personne)
241     for i in range(len(liste_personnes)) :
242         for j in range(len(liste_personnes)-1) :
243             if liste_personnes[j][1] > liste_personnes[j+1][1] :
244                 liste_personnes[j] , liste_personnes[j+1] = liste_personnes[j+1] , liste_personnes[j]
245
246 def detecter_collision_escaliers(point,personnes) :
247     for personne in personnes :
248         if point == personne[0] :
249             return True
250     return False
251
252 def detecter_collision_sortie(point,personnes,à_pop) :
253     for k in range(len(personnes)) :
254         if k not in à_pop :
255             if point == personnes[k][0] :
256                 return True
257     return False
258
```

IV) Annexe :

```
259 def gestion_étages(personnes,escaliers,nb_personnes,dictionnaire) :
260     nouvelles_personnes = []
261     for i in range(len(escaliers)) :
262         if randint(0,1) == 0 and nb_personnes > 0 and not detecter_collision_escaliers(escaliers[i],personnes) : # Escaliers remplis à 50%
263             nouvelles_personnes.append([escaliers[i],len(dictionnaire[escaliers[i]][1]),[random(),random(),random()]])
264             nb_personnes = nb_personnes - 1
265     return nb_personnes , nouvelles_personnes
266
267 def bouger_personnes(personnes,dictionnaire) : # La liste personnes doit être triée par distance à une sortie croissante
268     if personnes == [] :
269         return
270     à_pop = []
271     for k in range(len(personnes)) :
272         if personnes[k][1] < 2 :
273             à_pop.append(k)
274         elif not detecter_collision_sortie(dictionnaire[personnes[k][0]][1][1],personnes,à_pop) :
275             personnes[k][0] , personnes[k][1] = dictionnaire[personnes[k][0]][1][1] , personnes[k][1]-1
276         else :
277             pass
278     if à_pop != [] :
279         à_pop = tri_bulle_inverse(à_pop)
280         for indice in à_pop :
281             personnes.pop(indice)
282     return personnes
283 #####
284
285
286
287 #####
288 # Fonctions utilitaires :
289 def max_dictionnaire(dictionnaire) :
290     maxi = next(iter(dictionnaire))
291     for clé in dictionnaire :
292         if len(dictionnaire[clé][1]) > len(dictionnaire[maxi][1]) :
293             maxi = clé
294     return(maxi)
295
296 def point_aleatoire(salle):
297     x,y = randint(0,len(salle)-1),randint(0,len(salle[0])-1)
298     while salle[x][y] == 1:
299         x,y = randint(0,len(salle)-1),randint(0,len(salle[0])-1)
300     return (x,y)
```

IV) Annexe

```
302 def tri_bulle_inverse(liste) :
303     for i in range(len(liste)) :
304         for i in range(len(liste)-1) :
305             if liste[i] < liste[i+1] :
306                 liste[i] , liste[i+1] = liste[i+1] , liste[i]
307     return liste
308
309 def choisir_sorties_aleatoires(salle, nb_sorties):
310     sorties = []
311     n = len(salle)
312     while len(sorties) < nb_sorties:
313         i = random.randint(0, n-1)
314         j = random.randint(0, n-1)
315         if salle[i][j] == 0:
316             sorties.append((i, j))
317     return sorties
318
319 def creer_salle_depuis_image(image_path):
320     image = Image.open(image_path).convert("RGB")
321     pixels , salle , sorties , escaliers = image.load() , [[0 for j in range(image.size[0])] for i in range(image.size[1])] , [] , []
322     for i in range(image.size[1]):
323         for j in range(image.size[0]):
324             if pixels[j, i] == (0,0,0) :
325                 salle[i][j] = 1
326             elif pixels[j, i] == (0,255,0) :
327                 sorties.append((i, j))
328             elif pixels[j, i] == (255,0,0) :
329                 escaliers.append((i, j))
330     return salle, sorties , escaliers
331
```

IV) Annexe

```
332 def créer_salle_aleatoire(longueur,hauteur,obstacle,sorties) :# obstacle est la probabilité d'un obstacle sur chaque case
333     def créer_salle(longueur,hauteur,obstacle,sorties) :
334         salle , sorties_liste = [] , []
335         for i in range(hauteur) :
336             liste = []
337             for j in range(longueur) :
338                 if random() < obstacle or i == 0 or j == 0 or i == hauteur-1 or j == longueur-1:
339                     liste.append(1)
340                 else :
341                     liste.append(0)
342             salle.append(liste)
343         while len(sorties_liste) < sorties :
344             test = [[0,hauteur-1][randint(0,1)],randint(0,longueur-1),(randint(0,hauteur-1),[0,longueur-1][randint(0,1)])][randint(0,1)]
345             if test not in sorties_liste and test not in [(0,0),(0,longueur-1),(hauteur-1,0),(hauteur-1,longueur-1)]:
346                 sorties_liste.append(test)
347                 salle[test[0]][test[1]] = 0
348         return(salle,sorties_liste)
349     salle,sorties_liste = créer_salle(longueur,hauteur,obstacle,sorties)
350     while not est_connexe(salle) :
351         salle,sorties_liste = créer_salle(longueur,hauteur,obstacle,sorties)
352     return(salle,sorties_liste)
353
354 def point_accessiblees(salle) :
355     liste = []
356     for i in range(len(salle)) :
357         for j in range(len(salle[i])) :
358             if salle[i][j] == 0 :
359                 liste.append((i,j))
360     return(liste)
361
362 def sauvegarder(objet,fichier) :
363     with open(fichier, "w") as fichier:
364         fichier.write(str(objet))
365     return
366
367 def importer_dictionnaire(fichier) :
368     print("Importation en cours...")
369     with open(fichier, "r") as fichier:
370         dictionnaire = ast.literal_eval(fichier.read())
371     print("Dictionnaire importé avec succès.")
372     return dictionnaire
```

IV) Annexe

```
374 def est_connexe(salle): # parcours en profondeur
375     visite , debut_trouve , debut= [[False] * len(salle[0]) for _ in range(len(salle))], False , None
376     deplacements = [(-1, 0), (0, 1), (1, 0), (0, -1)] # diagonales interdites ici
377     for i in range(len(salle)):
378         for j in range(len(salle[0])):
379             if salle[i][j] == 0:
380                 debut_trouve = True
381                 debut = (i, j)
382                 break
383             if debut_trouve:
384                 break
385     if debut is None:
386         return False
387     pile = [debut]
388     while pile:
389         i, j = pile.pop()
390         visite[i][j] = True
391         for di, dj in deplacements:
392             ni, nj = i + di, j + dj
393             if 0 <= ni < len(salle) and 0 <= nj < len(salle[0]) and not visite[ni][nj] and salle[ni][nj] == 0:
394                 pile.append((ni, nj))
395     for i in range(len(salle)):
396         for j in range(len(salle[0])):
397             if salle[i][j] == 0 and not visite[i][j]:
398                 return False
399     return True
400
```

IV) Annexe

```
401 def comparaison_dijkstra_Aétoile(taillemin,pas) :
402     abscisses,ordonées1 , ordonées2 = [],[],[]
403     while bool(input("Continuer (True/False) ? ")) == True :
404         salle ,sorties = créer_salle_aleatoire(taillemin,taillemin,0,1)
405         abscisses.append(taillemin)
406         debut = time.time()
407         créer_dictionnaire_chemin(0,salle,sorties,[],"poubelle","D")
408         ordonées1.append(time.time()-debut)
409         debut = time.time()
410         créer_dictionnaire_chemin(0,salle,sorties,[],"poubelle","A")
411         ordonées2.append(time.time()-debut)
412         print(abscisses[-1],ordonées1[-1] , ordonées2[-1])
413         taillemin += pas
414         plt.plot(abscisses,ordonées1, label="Dijkstra")
415         plt.plot(abscisses,ordonées2, label="A*")
416         plt.xlabel("Côté de la salle carrée")
417         plt.ylabel("Temps d'exécution de l'algorithme")
418         plt.title("Comparaison Dijkstra / A-étoile")
419         plt.legend()
420         plt.show()
421     return abscisses
422 #####
423
424
425
426 #####
427 # Test :
428 def évacuation(fichier,personnes,modedico,modeaffichage,algo,nb_étages,FPS,nom) :
429     salle , sorties ,escaliers = créer_salle_depuis_image(fichier+".png")
430     print("Dimension de la salle",len(salle),"x",len(salle[0]))
431     #afficher_salle(salle, sorties,escaliers,2)
432     print("Les sorties sont :",afficher_liste(sorties))
433     print("Les escaliers sont :",afficher_liste(escaliers))
434     debut = time.time()
435     if modedico == "I" : # Import du dictionnaire
436         dictionnaire = importer_dictionnaire(fichier+".txt")
437     elif modedico == "C" :# Création d'un dictionnaire
438         dictionnaire = créer_dictionnaire_chemin(modeaffichage,salle,sorties,escaliers,fichier+".txt",algo)
439     print("Durée d'execution de",time.time()-debut,"secondes .")
440     liste_personnes = placer_personnes(personnes//nb_étages,salle,dictionnaire)
441     créer_vidéo(dictionnaire,salle,sorties,escaliers,liste_personnes,nb_étages,personnes-personnes//nb_étages,FPS,nom)
442     return
443
```

IV) Annexe

```
425 #####
426 # Test :
427 def évacuation(fichier,personnes,modedico,modeaffichage,algo,nb_étages,FPS,nom) :
428     salle , sorties ,escaliers = creer_salle_depuis_image(fichier+".png")
429     print("Dimension de la salle",len(salle),"x",len(salle[0]))
430     #afficher_salle(salle, sorties,escaliers,2)
431     print("Les sorties sont : ",afficher_liste(sorties))
432     print("Les escaliers sont : ",afficher_liste(escaliers))
433     debut = time.time()
434     if modedico == "I" : # Import du dictionnaire
435         dictionnaire = importer_dictionnaire(fichier+".txt")
436     elif modedico == "C" :# Création d'un dictionnaire
437         dictionnaire = créer_dictionnaire_chemin(modeaffichage,salle,sorties,escaliers,fichier+".txt",algo)
438     print("Durée d'exécution de ",time.time()-debut,"secondes.")
439     liste_personnes = placer_personnes(personnes//nb_étages,salle,dictionnaire)
440     créer_vidéo(dictionnaire,salle,sorties,escaliers,liste_personnes,nb_étages,personnes-personnes//nb_étages,FPS,nom)
441     return
442
443 def carte(fichier) :
444     salle , sorties ,escaliers = creer_salle_depuis_image(fichier+".png")
445     dictionnaire = importer_dictionnaire(fichier+".txt")
446     carte_distances_géodésiques(salle,sorties,dictionnaire)
447     return
448 #####
449 #évacuation('test',30,"C",0,"A",3,500)
450 #évacuation("Lycée_avec_portes (213x120)",600,"I",0,"A",3,400,"1.gif") # Création du dico en 3355 secondes
451 #évacuation("Lycée_avec_portes (160x90)",600,"I",0,"A",3,400,"2.gif") # Création du dico en   secondes
452 #évacuation("Lycée_avec_portes (107x60)2",1600,"I",0,"A",3,400,"3.gif") # Création du dico en 356   secondes
453 #évacuation(' Lycée_avec_portes (107x60)',1600,"I",0,"A",3,500,"4.gif") # Création du dico en 102 secondes
454
455 comparaison_dijkstra_Aétoile(10,10)
456 carte = carte("Lycée_avec_portes (213x120)")
```