

```

1  <From value.h>
2
3
4
5
6  #ifndef _VALUES_
7  #define _VALUES_
8
9  #define HEIGHT 100
10 #define WIDTH 100
11 #define INIT 0
12
13 #define VISION_DISTANCE 10
14 #define VISION_FIELD 60
15 #define ROTATION_SPEED 40
16 #define MEAN_PERIOD 2
17
18 #define NB_LEADERS 10
19 #define K_MEAN_ACCURACY 20
20 #define CROWD_DISTANCE 5
21
22 #endif
23
24
-----
25
26
27
28 <From entity/guard.h>
29
30
31
32
33 #ifndef _GUARD_
34 #define _GUARD_
35
36 #include "position.h"
37 #include "../scene/scene.h"
38
39 typedef struct{
40     position there;
41     int dx;
42     int dy;
43     int err;
44     int angle;
45     bool moving;
46     bool rotating;
47 } goal; //Nécessaire pour les mouvements simple
48
49 typedef struct behave_option_t behave_option;
50 typedef struct guard_t guard;
51 typedef struct guard_list_t guard_list;
52
53 struct behave_option_t {
54     int* angle_list;
55     position* position_list;
56     int list_size;
57     int current_task;
58     void (*move_function)(guard_list*,scene_object*,int,int);
59 };//Permet la complexité de comportement
60
61 struct guard_t{
62     position* pos;
63     int angle;
64     int behave;
65     int rotation_speed;
66     goal to_do;
67     behave_option option;
68 };
69
70 struct guard_list_t {
71     int nb;
72     int color;
73     int color2;
74     guard** tab;
75 };
76
77 guard* new_guard(int,int,int,position*);
78 void delete_guard(guard*);
79 guard_list* new_guard_list(int,int,int,int,int);
80 void delete_guard_list(guard_list*);
81
82 void initializer(guard*);
83
84 guard* copy_guard(guard*,int,int);
85 guard_list* copy_guard_list(guard_list*,int,int);
86 void edit_behaves(guard_list*,int);
87
88 void behave(guard_list*,scene_object*,int);
89
90 void guard_rotate_by(guard*,int);
91 void guard_rotate_to(guard*,int);
92 void guard_move_to(guard*,position,scene_object*);
93 void guard_natural_move_to(guard*,position,scene_object*);
94
95 int* find_leaders(guard_list*);
96 void arange_gards(guard_list*);
97
98 void nothing(guard*);
99 void rotation(guard*);
100 void random_line(guard*);
101 void corridor(guard*);
102 void centred_square(guard*);
103 void mousaid(guard*);
104 void snake(guard*);
105
106 void speed_rotation_fonction(guard_list*,scene_object*,int,int);
107 void classic_move_fonction(guard_list*,scene_object*,int,int);
108 void random_move_fonction(guard_list*,scene_object*,int,int);
109 void mousaid_move_fonction(guard_list*,scene_object*,int,int);
110 void snake_move_fonction(guard_list*,scene_object*,int,int);
111
112 #endif
113
114
-----
115
116
117
118 <From entity/player.h>
119
120
121
122 #ifndef _PLAYER_
123 #define _PLAYER_
124
125 #include "position.h"
126 #include "../tools/stack.h"
127 #include "../tools/priority.h"
128 #include "../scene/scene.h"
129
130 typedef struct {
131     position* pos;
132 } player;
133
134 player* new_player(int,int);
135 void delete_player(player*);
136
137 void move_left(position*,scene_object*);
138 void move_right(position*,scene_object*);
139 void move_up(position*,scene_object*);
140 void move_down(position*,scene_object*);
141
142 void naive_path(player*,scene_object**);
143 void A_path(player*,scene_object**,int,int,stack*,priority_list*);
144
145 void
multicore_path(player*,scene_object**,int,int,stack*,priority_list*,int);
146 void path(player*,scene_object**,int,int,stack*,priority_list*);
147 int check_and_move(player*,scene_object*,scene_object*,int,int,
148                     void (*f)(position*, scene_object*));
149
150 #endif
151
152
-----
153
154
155
156 <From entity/position.h>
157
158
159
160 #ifndef _POSITION_
161 #define _POSITION_
162
163 #include <stdbool.h>
164
165 typedef struct {
166     int x;
167     int y;
168 } position;
169
170 typedef struct {
171     int size;
172     position** tab;
173 } list_position;
174
175 position* new_position(int,int);
176 position* copy_position(position*);
177 void delete_position(position*);
178
179 list_position* new_list_position(int);
180 list_position* copy_list_position(list_position*);

```

```

181 void delete_list_position(list_position*);
182
183 void moveto(position*,int,int);
184
185 bool equals(position*,position*);
186
187 float range(position*,position*);
188
189 typedef struct q_e queue_elt;
190 struct q_e {
191     position pos;
192     queue_elt* next;
193     queue_elt* prev;
194 };
195
196 typedef struct {
197     queue_elt* end;
198     queue_elt* start;
199 } queue_position;
200
201 queue_elt* new_queue_elt(position);
202 void delete_queue_elt(queue_elt*);
203
204 queue_position* new_queue_position();
205 void delete_queue_position(queue_position*);
206
207 void enqueue(queue_position*,queue_elt*);
208 queue_elt* dequeue(queue_position*);
209
210 queue_position* copy_queue(queue_position*);
211
212 typedef struct t_n tree_node;
213 struct t_n {
214     tree_node** nexts;
215     int nb_nexts;
216     int nb_deleted;
217     tree_node* prev;
218     int name;
219     tree_node* start;
220     position pos;
221 };
222
223 tree_node* new_tree_node(position,tree_node*,tree_node*);
224 void add_next_tree_node(tree_node*,tree_node*);
225 void delete_tree_node(tree_node*);
226 void delete_tree_node_rac(tree_node*);
227
228 tree_node* new_start_tree_node(tree_node*);
229
230 #endif
231
232

```

```

233
234
235
236 <From old_function/threads.h>
237
238
239
240 #ifndef _THREADS_

```

```

241 #define _THREADS_
242
243 #include "stack.h"
244 #include "priority.h"
245 #include "../scene/scene.h"
246 #include "../entity/player.h"
247 #include <pthread.h>
248
249 typedef struct {
250     scene_object** future;
251     int time;
252     int accuracy;
253     position* end;
254     stack* closedList;
255     priority_list* openList;
256     pthread_mutex_t* m_prio;
257     pthread_mutex_t* m_stack;
258 } A_arg;
259
260 void* multicore_A_star(void*);
261
262 #endif
263
264

```

```

293
294
295
296 <From scene/simulation.h>
297
298
299
300 #ifndef _SIMULATION_

```

```

301 #define _SIMULATION_
302
303 #include <stdbool.h>
304 #include <stdio.h>
305 #include "scene.h"
306 #include "../entity/player.h"
307 #include "../entity/guard.h"
308
309 int simulate(scene_object*,guard_list*,player*,int,int,bool,FILE*);
310
311 #endif
312
313

```

```

314
315
316
317 <From tools/draw.h>
318
319
320
321 #ifndef _DRAW_
322 #define _DRAW_
323
324 #include "priority.h"
325 #include "../entity/position.h"
326 #include "../scene/scene.h"
327 #include <stdlib.h>
328
329 void check_position(position*,scene_object*);
330 void draw_position(position*,scene_object*,int);
331 void draw_cross(position*,scene_object*,int);
332
333 list_position* select_circle(position*,int);
334 list_position* select_arc(position*,int,int,int);
335 void check_selection(list_position*,scene_object*);
336 list_position* delete_double(list_position*);
337
338 void _draw_line(position*,position*,scene_object*,int,int,bool);
339 void draw_line(position*,position*,scene_object*,int,int);
340 void draw_line_in_cross(position*,position*,scene_object*,int,int);
341
342 void draw_cone(position*,int,int,int,scene_object*,int,int);
343 //collorie plusieurs fois la même case mais permet les obstacles
344 void
draw_cone_with_cross(position*,int,int,int,scene_object*,int,int);
345 void quick_draw_cone(position*,int,int,int,scene_object*,int,int);
346 void print_trajectory(scene_object*,priority_list*,int);
347
348 void clean_holl(scene_object*,int,int);
349
350 #endif
351
352

```

```

353
354
355
356 <From tools/graph.h>
357

```

```

358
359
360 #ifndef _GRAPH_
361 #define _GRAPH_
362
363 #include "../scene/scene.h"
364 #include "../entity/position.h"
365 #include "stack.h"
366 #include "priority.h"
367
368 list_position* neighbors2(scene_object*, position*, int, int);
369 list_position* neighbors(scene_object*, position*, int, int);
370 position*
A_star(scene_object**, int, int, position*, stack*, priority_list*);
371
372 #endif
373
374

```

```

375
376
377
378 <From tools/out.h>
379
380
381
382 #ifndef _OUT_
383 #define _OUT_
384
385 #include <stdio.h>
386 #include "../scene/scene.h"
387
388 char* convert_to_string(int);
389 void display_grid(int**, int, int, FILE*);
390 void display_scene(scene_object*, FILE*);
391
392 #endif
393
394

```

```

395
396
397
398 <From tools/priority.h>
399
400
401
402 #ifndef _PRIORITY_
403 #define _PRIORITY_
404
405 #include <stdbool.h>
406 #include "../entity/position.h"
407
408 typedef struct{
409     int prio;
410     int cout;
411     position* node;
412     tree_node* way;
413 }data;
414

```

```

415 typedef struct {
416     int size;
417     data** tas;
418 } priority_list;
419
420
421 data* new_data(int, position*, tree_node*, int);
422 void delete_data(data*);
423
424 priority_list* new_priority_list(int);
425 bool empty_priority_list(priority_list*);
426
427 void swap(int, int, priority_list*);
428 void percolate_up(int, priority_list*);
429 int choose_son(int, priority_list*);
430 void percolate_down(int, priority_list*);
431
432 void insert(data*, priority_list*);
433 data* remove_rac(priority_list*);
434
435 bool is_in_priority(priority_list*, position*, int);
436
437 void delete_priority_list(priority_list*);
438
439 void adjust_priority_list(priority_list*, position*, position);
440
441 #endif
442
443

```

```

444
445
446
447 <From tools/stack.h>
448
449
450
451 #ifndef _stack_
452 #define _stack_
453
454 #include <stdbool.h>
455 #include "../entity/position.h"
456
457 typedef struct s_e stack_elt;
458 struct s_e{
459     position* pos;
460     int time;
461     stack_elt* next;
462 };
463
464 typedef struct{
465     stack_elt* top;
466 }stack;
467
468 stack_elt* new_stack_elt(position*, int);
469 void delete_stack_elt(stack_elt*);
470
471 stack* new_stack();
472 void delete_stack(stack*);
473
474 void enstack(stack_elt*, stack*);
475 stack_elt* destack(stack*);

```

```

476
477 bool empty_stack(stack*);
478 bool is_in_stack(position*, int, stack*);
479
480 void adjust_stack(stack*, int);
481
482 #endif
483
484

```

```

----
485
486
487
488 <From main.c>
489
490
491
492 #include <stdio.h>
493 #include <stdlib.h>
494 #include <stdbool.h>
495 #include <time.h>
496 #include <string.h>
497 #include <pthread.h>
498 #include "values.h"
499 #include "entity/guard.h"
500 #include "entity/player.h"
501 #include "scene/scene.h"
502 #include "scene/simulation.h"
503
504 #define SOURCE_NAME_1 "../out/score/"
505 #define SOURCE_NAME_2 "../out/rotation/"
506 #define EXT ".txt"
507 #define MAX_BEHAVE 4
508
509 char* filled_zero(int k){
510     char* res;
511     if(k>=0){
512         res = malloc((k+1)*sizeof(char));
513         for(int i=0; i<k; i++){
514             res[i] = '0';
515         }
516         res[k] = '\0';
517     }
518     else{
519         res = malloc(sizeof(char));
520         res[0] = '\0';
521     }
522     return res;
523 }
524
525 int nb_zeros(int k){
526     if(k==0){
527         return 1;
528     }
529     int cur = k;
530     int res = 0;
531     while(cur != 0){
532         cur = cur/10;
533         res++;
534     }
535     return res;

```

```

536 }
537
538 char* new_name(int behave, int number, char* source_name){
539     char name[20];
540     char* postname = malloc(30*sizeof(char));
541     char* zero1 = filled_zero(2-nb_zeros(behave));
542     char* zero2 = filled_zero(3-nb_zeros(number));
543     sprintf(name,"%s%d_%s%d",zero1,behave,zero2,number);
544     sprintf(postname,"%s",source_name);
545     char* filename = strcat(name,EXT);
546     char* file = strcat(postname,filename);
547     free(zero1);
548     free(zero2);
549     return file;
550 }
551
552 guard_list** copy_lab(guard_list** storage, int size, int behave, int
r_speed){
553     guard_list** lab = malloc(size*sizeof(guard_list*));
554     for(int i=0; i<size; i++){
555         lab[i] = copy_guard_list(storage[i],behave,r_speed);
556     }
557     return lab;
558 }
559
560 void delete_lab(guard_list** lab, int size){
561     for(int i=0; i<size; i++){
562         delete_guard_list(lab[i]);
563     }
564     free(lab);
565 }
566
567 typedef struct {
568     int k;
569     int number;
570     bool behave;
571     int behave_angle;
572     int accuracy;
573     guard_list** storage;
574 } k_sim_arg;
575
576 void* thread_simulation(void* parameter){
577     k_sim_arg* args = parameter;
578     guard_list** lab;
579     char* file;
580     if(args->behave){
581         lab = copy_lab(args->storage,args->k,args->
>behave_angle,ROTATION_SPEED);
582         file = new_name(args->behave_angle,args->
>number,SOURCE_NAME_1);
583     }
584     else{
585         lab = copy_lab(args->storage,args->k,6,args->
>behave_angle); //Changer le comportement
586         file = new_name(args->behave_angle,args->
>number,SOURCE_NAME_2);
587     }
588     FILE* score = fopen(file,"w");//pour conserver les précédents
589
590     for(int i=0; i<args->k; i++){
591         scene_object* scene = new_scene(WIDTH,HEIGHT,INIT);
592         player* bot = new_player(WIDTH/2,0);

```

```

593         if(args->behave_angle == 0){
594             fprintf(score,"%d\n",simulate(scene,lab[i],bot,2,args->
>accuracy,false,NULL));
595         }
596         else{
597             fprintf(score,"%d\n",simulate(scene,lab[i],bot,5,args->
>accuracy,false,NULL));
598         }
599         delete_scene(scene);
600         delete_player(bot);
601     }
602     delete_lab(lab,args->k);
603     fclose(score);
604     free(file);
605     pthread_exit(NULL);
606 }
607
608 void k_simulation(int k,int number,int behave_start,int behave_end, int
accuracy){
609     printf("number : %d\n",number);
610     guard_list** storage = malloc(k*sizeof(guard_list*));
611     for(int i=0; i<k; i++){
612         storage[i] = new_guard_list(number,0,ROTATION_SPEED,1,4);
613     }
614     pthread_t* core = malloc((behave_end-behave_start+1)*sizeof(pthread_t));
615     k_sim_arg* parametres = malloc((behave_end-
behave_start+1)*sizeof(k_sim_arg));
616
617     for(int behave=behave_start; behave<=behave_end; behave++){
618         k_sim_arg parametre = {k,number,true,behave,accuracy,storage};
619         parametres[behave-behave_start] = parametre;
620         pthread_create(&core[behave-behave_start],NULL
        ,thread_simulation,&parametres[behave-behave_start]);
621     }
622
623     for(int behave=behave_start; behave<=behave_end; behave++){
624         pthread_join(core[behave-behave_start],NULL);
625     }
626
627     free(core);
628     free(parametres);
629     delete_lab(storage,k);
630 }
631
632 void simulate_from_to(int k, int n, int m, int pas, int range, int accuracy){
633     for(int i=n; i<=m; i+= pas){
634         k_simulation(k,i,5,6,accuracy); //Selectionner la Range
635     }
636 }
637
638 void k_simulation_rotation(int k,int number,int angle_start,int angle_end,
int pas, int accuracy){
639     printf("number : %d\n",number);
640     guard_list** storage = malloc(k*sizeof(guard_list*));
641     for(int i=0; i<k; i++){
642         storage[i] = new_guard_list(number,0,ROTATION_SPEED,1,4);
643     }
644     pthread_t* core = malloc((angle_end-angle_start+1)*sizeof(pthread_t));
645     k_sim_arg* parametres = malloc((angle_end-
angle_start+1)*sizeof(k_sim_arg));
646
647     for(int r_s=angle_start; r_s<=angle_end; r_s+=pas){
648         k_sim_arg parametre = {k,number,false,r_s,accuracy,storage};

```

```

649         parametres[r_s-angle_start] = parametre;
650         pthread_create(&core[r_s-
angle_start],NULL,thread_simulation,&parametres[r_s-angle_start]);
651     }
652     for(int r_s=angle_start; r_s<=angle_end; r_s+=pas){
653         pthread_join(core[r_s-angle_start],NULL);
654     }
655
656     free(core);
657     free(parametres);
658     delete_lab(storage,k);
659 }
660
661 void find_angle_rotation(int angle_start, int angle_end, int
angle_pas, int k,
662                         int n, int m, int pas, int accuracy){
663     for(int i=n; i<=m; i+=pas){
664         k_simulation_rotation(k,i,angle_start,angle_end,angle_pas,accuracy);
665     }
666 }
667
668 int main(int argc, char **argv){
669     int seed = time(NULL);
670     // seed = 1716554888;
671     srand(seed);
672     printf("Seed : %d\n",seed);
673
674     if(argc < 6){
675         printf("\nWARNING !!!\n \
        \nNeed 4 parametre (in this order) :\
        \n\t-k : the number of simulation per situation\
        \n\t-n and m : simule from n to m guards\
        \n\t-pas : it means what it means\
        \n\t-accuracy : accuracy of the player\n\n");
681     }
682     else{
683         int k = atoi(argv[1]);
684         int n = atoi(argv[2]);
685         int m = atoi(argv[3]);
686         int pas = atoi(argv[4]);
687         int accuracy = atoi(argv[5]);
688         simulate_from_to(k,n,m,pas,MAX_BEHAVE,accuracy);
689         // find_angle_rotation(40,60,2,k,n,m,pas,accuracy);
690     }
691
692     return 0;
693 }
694
695
-----
----
696
697
698
699 <From entity/guard.c>
700
701
702
703 #include <stdio.h>
704 #include <stdlib.h>
705 #include <math.h>

```



```

706 #include "player.h"
707 #include "guard.h"
708 #include "../tools/draw.h"
709 #include "../values.h"
710
711 guard* new_guard(int angle, int behave, int rotation_speed, position*
pos){
712     guard* res = calloc(1,sizeof(guard));
713     res->angle = angle;
714     res->behave = behave;
715     res->rotation_speed = rotation_speed;
716     res->pos = pos;
717     res->to_do.angle=0;
718     res->to_do.there = *pos;
719     initializer(res);
720     return res;
721 }
722
723 void delete_guard(guard* guard){
724     delete_position(guard->pos);
725     free(guard->option.position_list);
726     free(guard->option.angle_list);
727     free(guard);
728 }
729
730 guard_list* new_guard_list(int nb, int behave,int r_speed ,int color,
int color2){
731     guard_list* res = malloc(sizeof(guard_list));
732     res->nb = nb;
733     res->color = color;
734     res->color2 = color2;
735     res->tab = malloc(nb*sizeof(guard*));
736     for(int i=0; i<nb; i++){
737         position* pos;
738         int x = rand()%100;
739         int y = rand()%100;
740         if(x<=WIDTH/2+VISION_DISTANCE && x>=WIDTH/2-VISION_DISTANCE
741             && y<=VISION_DISTANCE)
742             i--;
743         else{
744
745             pos = new_position(x,y);
746             res->tab[i] = new_guard((360 + rand()
%360)%360,behave,r_speed,pos);
747         }
748     }
749     return res;
750 }
751
752 void delete_guard_list(guard_list* guard_list){
753     for(int i=0; i<guard_list->nb; i++){
754         delete_guard(guard_list->tab[i]);
755     }
756     free(guard_list->tab);
757     free(guard_list);
758 }
759
760 guard* copy_guard(guard* guard, int behave, int r_speed){
761     position* pos = copy_position(guard->pos);
762     return new_guard(guard->angle,behave,r_speed,pos);
763 }
764

```

```

765 guard_list* copy_guard_list(guard_list* guards, int behave, int r_speed){
766     guard_list* res = malloc(sizeof(guard_list));
767     res->nb = guards->nb;
768     res->color = guards->color;
769     res->color2 = guards->color2;
770     res->tab = malloc(res->nb*sizeof(guard*));
771     for(int i=0; i<res->nb; i++){
772         res->tab[i] = copy_guard(guards->tab[i],behave,r_speed);
773     }
774     return res;
775 }
776
777 void edit_behaves(guard_list* guards, int behave){
778     for(int i=0; i<guards->nb; i++){
779         guards->tab[i]->behave = behave;
780         free(guards->tab[i]->option.angle_list);
781         free(guards->tab[i]->option.position_list);
782         initializer(guards->tab[i]);
783     }
784 }
785
786 void behave(guard_list* guards, scene_object* scene, int time){
787     scene_init(scene,scene->init);
788     int* leader = find_leaders(guards);
789     if(time == 0){
790         arange_gards(guards);
791     }
792     for(int i=0;i<guards->nb;i++){
793         guard* guard = guards->tab[i];
794         guard->option.move_function(guards,scene,i,leader[i]);
795         scene->time = time;
796         draw_cone_with_cross(guard->pos,VISION_DISTANCE,
797             guard->angle,VISION_FIELD,scene,guards->color,guards->color2);
798         clean_holl(scene,guards->color,guards->color2);
799     }
800     free(leader);
801 }
802
803 void guard_rotate_by(guard* guard, int deg){
804     goal* target = &guard->to_do;
805     if(target->angle < guard->rotation_speed){
806         target->angle = deg;
807     }
808
809     int signe = target->angle > 0 ? 1 : -1;
810     if(target->angle*signe > guard->rotation_speed){
811         target->angle += (-guard->rotation_speed)*signe;
812         guard->angle += guard->rotation_speed*signe;
813         target->rotating = true;
814     }
815     else{
816         guard->angle += target->angle;
817         target->angle = 0;
818         target->rotating = false;
819     }
820     guard->angle = (360+guard->angle)%360;
821 }
822
823 void guard_rotate_to(guard* guard, int angle){
824     int deg = abs(angle)-guard->angle;
825     int signe = deg>0 ? 1 : -1;
826     if(angle>=0){

```

```

827         guard_rotate_by(guard,deg*signe>180 ? (deg*signe-360)*signe :
deg);
828     }
829     else{
830         guard_rotate_by(guard,deg*signe>180 ? deg : (deg*signe-
360)*signe);
831     }
832 }
833
834 void guard_move_to(guard* guard, position pos, scene_object* scene){
835     goal* target = &guard->to_do;
836     position* p1 = guard->pos;
837     position* p2 = &target->there;
838     int err2 = 2 * target->err;
839     if (err2 + target->dy > 0) {
840         target->err -= target->dy;
841         p1->x += (p1->x - p2->x < 0) ? 1 : -1;
842     }
843     if (0 < target->dx - err2) {
844         target->err += target->dx;
845         p1->y += (p1->y - p2->y < 0) ? 1 : -1;
846     }
847     if(equals(p1,p2)){
848         target->there = pos;
849         check_position(&target->there,scene);
850         target->dx = abs(p2->x - p1->x);
851         target->dy = abs(p2->y - p1->y);
852         target->err = target->dx - target->dy;
853         target->moving = false;
854     }
855 }
856
857 void guard_natural_move_to(guard* guard, position pos, scene_object*
scene){
858     goal* target = &guard->to_do;
859     int x = pos.x-guard->pos->x;
860     int y = pos.y-guard->pos->y;
861     int angle = guard->angle;
862     if(x!=0 || y!=0){
863         angle = ((int)(-atan2(y,x)*(180.0)/(3.14)) + 360)%360;
864     }
865     if(!target->moving){
866         guard_rotate_to(guard,angle);
867     }
868     if(!target->rotating || target->moving){
869         target->moving = true;
870         guard_move_to(guard,pos,scene);
871     }
872 }
873
874 void update_classe(guard_list* guards,int* res, int* classe, bool
last){
875     for(int i=0; i<guards->nb;i++){
876         float min_range = range(guards->tab[i]->pos,guards-
>tab[res[0]]->pos);
877         for(int j=1; j<NB_LEADERS;j++){
878             float r = range(guards->tab[i]->pos,guards->tab[res[j]]-
>pos);
879             if(r<min_range){
880                 min_range = r;
881                 classe[i] = (last?res[j]:j);
882             }

```

```

883     }
884 }
885 }
886
887 void _closest_guards(guard_list* guards, int* res, position* tmp){
888     for(int j=0; j<NB_LEADERS;j++){
889         float min_range = range(guards->tab[0]->pos,&tmp[j]);
890         for(int i=1; i<guards->nb; i++){
891             float r = range(guards->tab[i]->pos,&tmp[j]);
892             if(r<min_range){
893                 min_range = r;
894                 res[j] = i;
895             }
896         }
897     }
898 }
899
900 void _update_res(guard_list* guards, int* res, int* classe){
901     position* tmp = calloc(NB_LEADERS,sizeof(position));
902     int* count = calloc(NB_LEADERS,sizeof(int));
903     for(int i=0;i<guards->nb;i++){
904         tmp[classe[i]].x += guards->tab[i]->pos->x;
905         tmp[classe[i]].y += guards->tab[i]->pos->y;
906         count[classe[i]] ++;
907     }
908     for(int i=0;i<NB_LEADERS;i++){
909         int n = (count[i]>0?count[i]:1);
910         tmp[i].x = tmp[i].x/n;
911         tmp[i].y = tmp[i].y/n;
912     }
913     free(count);
914     _closest_guards(guards,res,tmp);
915     free(tmp);
916 }
917
918 int* find_leaders(guard_list* guards){
919     int* res = malloc(NB_LEADERS*sizeof(int));
920     int* classe = calloc(guards->nb,sizeof(int));
921     for(int i=0;i<NB_LEADERS;i++){
922         res[i] = i%guards->nb;
923     }
924     for(int i=0;i<K_MEAN_ACCURACY;i++){
925         _update_classe(guards,res,classe,false);
926         _update_res(guards,res,classe);
927     }
928     _update_classe(guards,res,classe,true);
929     free(res);
930     return classe;
931 }
932
933 void _guards_colone(guard_list* guards, int* res, position* tmp){
934     for(int i=0; i<guards->nb; i++){
935         float min_range = range(guards->tab[i]->pos,&tmp[0]);
936         for(int j=1; j<guards->nb;j++){
937             float r = range(guards->tab[i]->pos,&tmp[j]);
938             if(r<min_range){
939                 min_range = r;
940                 res[i] = j;
941             }
942         }
943     }
944 }

```

```

945
946 void _swap_guard(guard** tab, int i, int j){
947     guard* tmp = tab[i];
948     tab[i] = tab[j];
949     tab[j] = tmp;
950 }
951
952 void arange_gards(guard_list* guards){
953     int size = guards->nb;
954     int* colone = calloc(size,sizeof(int));
955     position* tmp = malloc(size*sizeof(position));
956     bool* is_use = calloc(size,sizeof(bool));
957     for(int i=0; i<size; i++){
958         tmp[i].x = (int)(i*(double)WIDTH/size);
959         tmp[i].y = HEIGHT/2;
960     }
961     _guards_colone(guards,colone,tmp);
962     for(int i=0; i<size; i++){
963         for(int j=0; j<size; j++){
964             int k = (colone[i]+j)%size;
965             if(!is_use[k]){
966                 is_use[k] = true;
967                 _swap_guard(guards->tab,i,k);
968                 break;
969             }
970         }
971     }
972     free(tmp);
973     free(colone);
974     free(is_use);
975 }
976
977 // _____ BEHAVES _____
978
979 void initializer(guard* res){
980     void (*init)(guard*);
981     switch (res->behave){
982         case 1:
983             init = &rotation;
984             break;
985         case 2:
986             init = &random_line;
987             break;
988         case 3:
989             init = &corridor;
990             break;
991         case 4:
992             init = &centred_square;
993             break;
994         case 5:
995             init = &mousaid;
996             break;
997         case 6:
998             init = &snake;
999             break;
1000         default:
1001             init = &nothing;
1002             break;
1003     }
1004     init(res);
1005 }
1006

```

```

1007 void nothing(guard* guard){
1008     guard->option.angle_list = calloc(1,sizeof(int));
1009     guard->option.position_list = malloc(sizeof(position));
1010     guard->option.position_list[0] = *guard->pos;
1011     guard->option.list_size = 1;
1012     guard->option.move_function = &classic_move_fonction;
1013 }
1014
1015 void rotation(guard* guard){
1016     nothing(guard);
1017     guard->option.angle_list[0] = 360;
1018     guard->option.move_function = &speed_rotation_fonction;
1019 }
1020
1021 void random_line(guard* guard){
1022     nothing(guard);
1023     guard->option.move_function = &random_move_fonction;
1024 }
1025
1026 void corridor(guard* guard){
1027     guard->option.angle_list = calloc(2,sizeof(int));
1028     guard->option.position_list = malloc(2*sizeof(position));
1029     guard->option.position_list[0] = *guard->pos;
1030     guard->option.position_list[1] = *guard->pos;
1031     switch ((guard->angle-45)/90){
1032         case 0:
1033             guard->option.position_list[1].x += 20;
1034             break;
1035         case 1:
1036             guard->option.position_list[1].y -= 20;
1037             break;
1038         case 2:
1039             guard->option.position_list[1].x -= 20;
1040             break;
1041         default:
1042             guard->option.position_list[1].y += 20;
1043             break;
1044     }
1045     scene_object* scene = new_scene(WIDTH,HEIGHT,0);
1046     check_position(&guard->option.position_list[1],scene);
1047     delete_scene(scene);
1048     guard->option.list_size = 2;
1049     guard->option.move_function = &classic_move_fonction;
1050 }
1051
1052 int _min_dist(guard* guard){
1053     int res = 0;
1054     position* list = guard->option.position_list;
1055     int size = guard->option.list_size;
1056     for(int i=1;i<size;i++){
1057         if(range(guard->pos,&list[i]) < range(guard->pos,&list[res])){
1058             res = i;
1059         }
1060     }
1061     if((guard->pos->x == list[res].x && guard->pos->x ==
list[(res+1)%size].x)
1062        || (guard->pos->y == list[res].y && guard->pos->y ==
list[(res+1)%size].y)){
1063         for(int i=0; i<size/2; i++){
1064             position tmp = list[i];
1065             list[i] = list[size-1-i];
1066             list[size-1-i] = tmp;

```

```

1067     }
1068     res = size-1-res;
1069 }
1070 return res;
1071 }
1072
1073 void centred_square(guard* guard){
1074     int mid_x = WIDTH/2;
1075     int mid_y = HEIGHT/2;
1076     int r_x = abs(guard->pos->x - mid_x);
1077     int r_y = abs(guard->pos->y - mid_y);
1078     if(r_x<VISION_DISTANCE && r_y<VISION_DISTANCE){
1079         rotation(guard);
1080     }
1081     else{
1082         guard->option.list_size = 4;
1083         guard->option.angle_list = calloc(4,sizeof(int));
1084         guard->option.position_list = malloc(4*sizeof(position));
1085         int r = fmax(r_x,r_y);
1086         for(int i=0;i<4;i++){
1087             int s[2] = {1,-1};
1088             position p = {mid_x + s[i/2]*r, mid_y + s[(i+1)?1:0)}
1089 %2]*r});
1089             guard->option.position_list[i]=p;
1090         }
1091         guard->option.current_task = _min_dist(guard);
1092         guard->option.move_function = &classic_move_fonction;
1093     }
1094 }
1095
1096 void mousaid(guard* guard){
1097     guard->option.angle_list = calloc(1,sizeof(int));
1098     guard->option.position_list = malloc(1*sizeof(position));
1099     guard->option.list_size = 1;
1100     guard->option.position_list[0] = *guard->pos;
1101     guard->option.current_task = 0;
1102     guard->option.move_function = &mousaid_move_fonction;
1103 }
1104
1105 void snake(guard* guard){
1106     guard->option.angle_list = calloc(5,sizeof(int));
1107     guard->option.position_list = malloc(5*sizeof(position));
1108     guard->option.list_size = 5;
1109     for(int i=0; i<5; i++){
1110         guard->option.position_list[i] = *guard->pos;
1111     }
1112     guard->option.current_task = 0;
1113     guard->option.move_function = &snake_move_fonction;
1114 }
1115
1116 void speed_rotation_fonction(guard_list* guards, scene_object* scene,
int i, int leader){
1117     guard* guard = guards->tab[i];
1118     guard->angle = (guard->angle + guard->rotation_speed)%360;
1119 }
1120
1121 void classic_move_fonction(guard_list* guards, scene_object* scene,
int i, int leader){
1122     guard* guard = guards->tab[i];
1123     behave_option option = guard->option;
1124     int current = option.current_task;
1125     position next_pos = option.position_list[current];

```

```

1126     if(option.angle_list[current] != 0){
1127         guard_rotate_by(guard,option.angle_list[current]);
1128     }
1129     else{
1130         guard_natural_move_to(guard,next_pos,scene);
1131     }
1132     if(option.angle_list[current] == 0 && equals(guard->pos,&next_pos)){
1133         guard->option.current_task = (current+1)%option.list_size;
1134     }
1135 }
1136
1137 void random_move_fonction(guard_list* guards, scene_object* scene, int i, int
leader){
1138     guard* guard = guards->tab[i];
1139     behave_option option = guard->option;
1140     position next_pos = option.position_list[0];
1141     guard_natural_move_to(guard,next_pos,scene);
1142     if(equals(guard->pos,&next_pos)){
1143         guard->option.position_list[0].x = rand()%scene->width;
1144         guard->option.position_list[0].y = rand()%scene->height;
1145     }
1146 }
1147
1148 void _bounce_position(position* pos, scene_object* scene){
1149     int b1 = pos->x/scene->width > 0;
1150     int b2 = pos->y/scene->height > 0;
1151     if(pos->x < 0 || b1){
1152         pos->x = -pos->x + 2*b1*scene->width;
1153     }
1154     if(pos->y < 0 || b2){
1155         pos->y = -pos->y + 2*b2*scene->height;
1156     }
1157     check_position(pos,scene);
1158 }
1159
1160 void _micro_decision(guard* leader,scene_object* scene, guard* guard){
1161     int D = 8;
1162     int x_min=guard->pos->x-D;
1163     int x_max=guard->pos->x+D;
1164     int y_min=guard->pos->y-D;
1165     int y_max=guard->pos->y+D;
1166     position* p1 = guard->pos;
1167     position* p = &guard->option.position_list[0];
1168     for(int x=x_min;x<=x_max ;x++){
1169         for(int y=y_min;y<=y_max ;y++){
1170             position q = {x,y};
1171             check_position(&q,scene);
1172             if(scene->grid[q.y][q.x] != scene->init){
1173                 p->y += (p1->y-q.y>0?1:-1);
1174                 p->x += (p1->x-q.x>0?1:-1);
1175             }
1176         }
1177     }
1178 }
1179
1180 void mousaid_move_fonction(guard_list* guards, scene_object* scene, int i,
int j){
1181     guard* leader = guards->tab[j];
1182     guard* guard = guards->tab[i];
1183     behave_option option = guard->option;
1184     position next_pos = option.position_list[0];
1185     int D = 3;

```

```

1186     if(j == i){
1187         guard_natural_move_to(guard,next_pos,scene);
1188         if(equals(guard->pos,&next_pos)){
1189             position new_pos = *guard->pos;
1190             int angle = guard->angle*(3.14/180);
1191             new_pos.x += (int)(cos(angle)*D);
1192             new_pos.y -= (int)(sin(angle)*D);
1193             _bounce_position(&new_pos,scene);
1194             option.position_list[0] = new_pos;
1195         }
1196     }
1197     else{
1198         guard_natural_move_to(guard,next_pos,scene);
1199         position* p1 = guard->pos;
1200         position* p2 = leader->pos;
1201         position p = leader->option.position_list[0];
1202         if(range(p1,p2)<VISION_DISTANCE){
1203             if((((leader->angle-45)/90)%2){
1204                 p.y += (p1->y-p2->y>0?1:-1)*VISION_DISTANCE;
1205             }
1206             else{
1207                 p.x += (p1->x-p2->x>0?1:-1)*VISION_DISTANCE;
1208             }
1209         }
1210         guard->option.position_list[0]=p;
1211         _micro_decision(leader,scene,guard);
1212         _bounce_position(&option.position_list[0],scene);
1213     }
1214 }
1215 }
1216
1217 void snake_move_fonction(guard_list* guards, scene_object* scene, int
i, int leader){
1218     guard* guard = guards->tab[i];
1219     behave_option option = guard->option;
1220     int current = option.current_task;
1221     position next_pos = option.position_list[current];
1222     guard_natural_move_to(guard,next_pos,scene);
1223     if(option.angle_list[current] == 0 && equals(guard-
>pos,&next_pos)){
1224         guard->option.current_task = (current+1)%option.list_size;
1225         if(guard->option.current_task == 0){
1226             int x1 = (int)((double)scene->width/guards->nb);
1227             int x2 = (int)((i+1)*(double)scene->width/guards->nb);
1228             if(x2>=scene->width){
1229                 x2 = scene->width-1;
1230             }
1231             int dy = sin(VISION_FIELD*3.14/360)*VISION_DISTANCE;
1232             guard->option.current_task++;
1233             int x[2] = {x2,x1};
1234             int h = scene->height;
1235             for(int j=1;j<5;j++){
1236                 guard->option.position_list[j].y = (guard->pos->y-
(j/2)*dy+h)%h;
1237                 guard->option.position_list[j].x = x[(j-1)/2];
1238             }
1239         }
1240     }
1241 }
1242

```

```

1243 -----
1244 -----
1245
1246 <From entity/player.c>
1247
1248
1249
1250
1251 #include <stdio.h>
1252 #include <stdlib.h>
1253 #include <string.h>
1254 #include "player.h"
1255 #include "../tools/graph.h"
1256
1257 player* new_player(int x, int y){
1258     player* bot = malloc(sizeof(player));
1259     position* pos = new_position(x, y);
1260     bot->pos = pos;
1261     return bot;
1262 }
1263
1264 void delete_player(player* bot){
1265     delete_position(bot->pos);
1266     free(bot);
1267 }
1268
1269 void move_left(position* pos,scene_object* scene){
1270     if(pos->x > 0){
1271         pos->x--;
1272     }
1273 }
1274
1275 void move_right(position* pos,scene_object* scene){
1276     if(pos->x < scene->width-1){
1277         pos->x++;
1278     }
1279 }
1280
1281 void move_down(position* pos,scene_object* scene){
1282     if(pos->y > 0){
1283         pos->y--;
1284     }
1285 }
1286
1287 void move_up(position* pos,scene_object* scene){
1288     if(pos->y < scene->height-1){
1289         pos->y++;
1290     }
1291 }
1292
1293 void naive_path(player* bot, scene_object** future) {
1294     scene_object* today = (future[0]->time > future[1]->time) ?
future[1] : future[0];
1295     scene_object* tomorrow = (future[0]->time > future[1]->time) ?
future[0] : future[1];
1296
1297     if (bot->pos->x == 0) {
1298         if (bot->pos->y == 0) {
1299             if (check_and_move(bot, today, tomorrow, bot->pos->y + 1,
bot->pos->x, move_up) ||

```

```

1300         check_and_move(bot, today, tomorrow, bot->pos->y, bot->pos->x
+ 1, move_right)) {
1301             return;
1302         }
1303     } else {
1304         if (check_and_move(bot, today, tomorrow, bot->pos->y + 1, bot-
>pos->x, move_up) ||
1305         check_and_move(bot, today, tomorrow, bot->pos->y, bot->pos->x
+ 1, move_right) ||
1306         check_and_move(bot, today, tomorrow, bot->pos->y - 1, bot-
>pos->x, move_down)) {
1307             return;
1308         }
1309     }
1310     } else if (bot->pos->x == today->width - 1) {
1311         if (bot->pos->y == 0) {
1312             if (check_and_move(bot, today, tomorrow, bot->pos->y + 1, bot-
>pos->x, move_up) ||
1313             check_and_move(bot, today, tomorrow, bot->pos->y, bot->pos->x
- 1, move_left)) {
1314                 return;
1315             }
1316         } else {
1317             if (check_and_move(bot, today, tomorrow, bot->pos->y + 1, bot-
>pos->x, move_up) ||
1318             check_and_move(bot, today, tomorrow, bot->pos->y, bot->pos->x
- 1, move_left) ||
1319             check_and_move(bot, today, tomorrow, bot->pos->y - 1, bot-
>pos->x, move_down)) {
1320                 return;
1321             }
1322         }
1323     } else {
1324         if (bot->pos->y == 0) {
1325             if (check_and_move(bot, today, tomorrow, bot->pos->y + 1, bot-
>pos->x, move_up) ||
1326             check_and_move(bot, today, tomorrow, bot->pos->y, bot->pos->x
+ 1, move_right) ||
1327             check_and_move(bot, today, tomorrow, bot->pos->y, bot->pos->x
- 1, move_left)) {
1328                 return;
1329             }
1330         } else {
1331             if (check_and_move(bot, today, tomorrow, bot->pos->y + 1, bot-
>pos->x, move_up) ||
1332             check_and_move(bot, today, tomorrow, bot->pos->y, bot->pos->x
+ 1, move_right) ||
1333             check_and_move(bot, today, tomorrow, bot->pos->y, bot->pos->x
- 1, move_left) ||
1334             check_and_move(bot, today, tomorrow, bot->pos->y - 1, bot-
>pos->x, move_down)) {
1335                 return;
1336             }
1337         }
1338     }
1339 }
1340
1341 int check_and_move(player* bot, scene_object* today, scene_object* tomorrow,
1342 int y, int x, void (*move_function)(position*,
scene_object**)){
1343     if ((today->grid[y][x] == today->init) &&
1344         (tomorrow->grid[y][x] == tomorrow->init)) {

```

```

1345         move_function(bot->pos, today);
1346         return 1;
1347     }
1348     return 0;
1349 }
1350
1351
1352 void A_path(player* bot, scene_object** future, int time, int
accuracy,stack* closedList,
1353 priority_list* openList){
1354     position* end = new_position(bot->pos->x,future[0]->height-1);
1355     position* next =
A_star(future,time,accuracy,end,closedList,openList);
1356     if(next!=NULL){
1357         moveto(bot->pos,next->x,next->y);
1358     }
1359     delete_position(next);
1360     delete_position(end);
1361 }
1362
1363 void path(player* bot, scene_object** future, int time,int accuracy,
1364 stack* closedList, priority_list* openList){
1365     if(accuracy == 2){
1366         naive_path(bot,future);
1367     }
1368     else{
1369         A_path(bot,future,time,accuracy,closedList,openList);
1370     }
1371 }
1372
1373 -----
1374 -----
1375
1376 <From entity/position.c>
1377
1378
1379
1380
1381 #include <stdio.h>
1382 #include <stdlib.h>
1383 #include <math.h>
1384 #include "position.h"
1385 #include "../scene/scene.h"
1386
1387 position* new_position(int x, int y){
1388     position* pos = malloc(sizeof(position));
1389     pos->x = x;
1390     pos->y = y;
1391     return pos;
1392 }
1393
1394 position* copy_position(position* pos){
1395     return new_position(pos->x, pos->y);
1396 }
1397
1398 void delete_position(position* pos){
1399     free(pos);
1400 }
1401
1402 list_position* new_list_position(int nb){

```



```

1403     list_position* list = malloc(sizeof(list_position));
1404     position** tab = calloc(nb, sizeof(position*));
1405     list->size = nb;
1406     list->tab = tab;
1407     return list;
1408 }
1409
1410 void delete_list_position(list_position* list_pos){
1411     for(int i=0; i<list_pos->size; i++){
1412         delete_position(list_pos->tab[i]);
1413     }
1414     free(list_pos->tab);
1415     free(list_pos);
1416 }
1417
1418 void moveto(position* pos, int x, int y){
1419     pos->x = x;
1420     pos->y = y;
1421 }
1422
1423 bool equals(position* pos1, position* pos2){
1424     if(pos1->y == pos2->y && pos1->x == pos2->x){
1425         return true;
1426     }
1427     else{
1428         return false;
1429     }
1430 }
1431
1432 float range(position* pos1, position* pos2){
1433     int dx = (pos1->x - pos2->x);
1434     int dy = (pos1->y - pos2->y);
1435     return sqrt(dx*dx+dy*dy);
1436 }
1437
1438 queue_elt* new_queue_elt(position pos){
1439     queue_elt* res = malloc(sizeof(queue_elt));
1440     res->next = NULL;
1441     res->prev = NULL;
1442     res->pos = pos;
1443     return res;
1444 }
1445
1446 void delete_queue_elt(queue_elt* elt){
1447     free(elt);
1448 }
1449
1450 queue_position* new_queue_position(){
1451     queue_position* res = malloc(sizeof(queue_position));
1452     res->end = NULL;
1453     res->start = NULL;
1454     return res;
1455 }
1456
1457 void delete_queue_position(queue_position* q){
1458     while (q->end != NULL){
1459         delete_queue_elt(dequeue(q));
1460     }
1461     free(q);
1462 }
1463
1464 void enqueue(queue_position* q, queue_elt* elt){

```

```

1465     if(q->start != NULL){
1466         elt->next = q->start;
1467         q->start->prev = elt;
1468         q->start = elt;
1469     }
1470     else{
1471         q->end = elt;
1472         q->start = elt;
1473     }
1474 }
1475
1476 queue_elt* dequeue(queue_position* q){
1477     queue_elt* res;
1478     if(q->end == q->start){
1479         res = q->start;
1480         q->start = NULL;
1481         q->end = NULL;
1482     }
1483     else{
1484         res = q->end;
1485         res->prev->next = NULL;
1486         q->end = res->prev;
1487     }
1488     res->prev = NULL;
1489     res->next = NULL;
1490     return res;
1491 }
1492
1493 queue_position* copy_queue(queue_position* q){
1494     queue_position* res = new_queue_position();
1495     if(q != NULL && q->end != NULL){
1496         queue_elt* elt = q->end;
1497         while (elt != q->start){
1498             enqueue(res, new_queue_elt(elt->pos));
1499             elt = elt->prev;
1500         }
1501         enqueue(res, new_queue_elt(elt->pos));
1502     }
1503     return res;
1504 }
1505
1506 tree_node* new_tree_node(position pos, tree_node* start, tree_node* prev){
1507     tree_node* res = malloc(sizeof(tree_node));
1508     res->nexts = calloc(5, sizeof(tree_node*));
1509     res->nb_nexts = 0;
1510     res->nb_deleted = 0;
1511     res->prev = prev;
1512     res->start = start;
1513     res->pos = pos;
1514     return res;
1515 }
1516
1517 void add_next_tree_node(tree_node* node, tree_node* next){
1518     if(node->nb_nexts < 5){
1519         node->nexts[node->nb_nexts] = next;
1520         next->name = node->nb_nexts;
1521         node->nb_nexts ++;
1522     }
1523 }
1524
1525 void swap_t_n(tree_node* node, int i, int j){
1526     node->nexts[i]->name = j;

```

```

1527     node->nexts[j]->name = i;
1528     tree_node* tmp = node->nexts[i];
1529     node->nexts[i] = node->nexts[j];
1530     node->nexts[j] = tmp;
1531 }
1532
1533 void delete_tree_node(tree_node* node){
1534     for(int i=0; i<node->nb_nexts; i++){
1535         node->nexts[i]->prev = NULL;
1536         delete_tree_node_rac(node->nexts[i]);
1537     }
1538     if(node->prev != NULL){
1539         tree_node* prev = node->prev;
1540         swap_t_n(prev, node->name, prev->nb_nexts-1);
1541         prev->nb_deleted ++;
1542         prev->nb_nexts --;
1543         if(prev->nb_deleted >= 5){
1544             delete_tree_node(prev);
1545         }
1546         free(node->nexts);
1547         free(node);
1548     }
1549 }
1550
1551 void delete_tree_node_rac(tree_node* rac){
1552     delete_tree_node(rac);
1553     free(rac->nexts);
1554     free(rac);
1555 }
1556
1557 tree_node* new_start_tree_node(tree_node* node){
1558     tree_node* current = node;
1559     tree_node* prev = current->prev;
1560     if(prev == NULL || prev->prev == NULL){
1561         return NULL;
1562     }
1563     while(prev != NULL && prev->prev != NULL){
1564         current = prev;
1565         prev = current->prev;
1566     }
1567     return current;
1568 }
1569
1570 -----
1571 ----
1572
1573 <From old_function/clean_holl.c>
1574
1575
1576
1577
1578 #include "../entity/position.h"
1579 #include "../scene/scene.h"
1580
1581 void clean_holl1(scene_object* scene, int color){
1582     for(int i=0; i<scene->height; i++){
1583         for(int j=0; j<scene->width; j++){
1584             if(scene->grid[i][j]!=color){
1585                 if(i==0){
1586                     if(j==0){

```

```

1587         if(scene->grid[i][j+1]==color && scene-
>grid[i+1][j]==color)
1588             scene->grid[i][j] = color;
1589     }
1590     else if(j==scene->width-1){
1591         if(scene->grid[i][j-1]==color && scene-
>grid[i+1][j]==color)
1592             scene->grid[i][j] = color;
1593     }
1594     else{
1595         if(scene->grid[i][j-1]==color && scene-
>grid[i][j+1]==color
1596             && scene->grid[i+1][j]==color) scene-
>grid[i][j] = color;
1597     }
1598 }
1599 else if(i==scene->height-1){
1600     if(j==0){
1601         if(scene->grid[i-1][j]==color && scene-
>grid[i][j+1]==color)
1602             scene->grid[i][j] = color;
1603     }
1604     else if(j==scene->width-1){
1605         if(scene->grid[i-1][j]==color && scene-
>grid[i][j-1]==color)
1606             scene->grid[i][j] = color;
1607     }
1608     else{
1609         if(scene->grid[i-1][j]==color && scene-
>grid[i][j-1]==color
1610             && scene->grid[i][j+1]==color) scene-
>grid[i][j] = color;
1611     }
1612 }
1613 else{
1614     if(j==0){
1615         if(scene->grid[i-1][j]==color && scene-
>grid[i][j+1]==color
1616             && scene->grid[i+1][j]==color) scene-
>grid[i][j] = color;
1617     }
1618     else if(j==scene->width-1){
1619         if(scene->grid[i-1][j]==color && scene-
>grid[i][j-1]==color
1620             && scene->grid[i+1][j]==color) scene-
>grid[i][j] = color;
1621     }
1622     else{
1623         if(scene->grid[i-1][j]==color && scene-
>grid[i][j-1]==color
1624             && scene->grid[i][j+1]==color && scene-
>grid[i+1][j]==color)
1625             scene->grid[i][j] = color;
1626     }
1627 }
1628 }
1629 }
1630 }
1631 }
1632

```

```

1633 -----
1634
1635
1636 <From old_function/draw_line.c>
1637
1638
1639
1640 #include "../entity/position.h"
1641 #include "../scene/scene.h"
1642 #include "../tools/draw.h"
1643
1644 void draw_line(position* p,position* p2,scene_object* scene,int color){
1645
1646     position* p1 = copy_position(p);
1647
1648     int dx = p2->x - p1->x;
1649     int dy = p2->y - p1->y;
1650
1651     if (dx!=0){//Non vertical
1652
1653         if(dx>0){//Partie droite
1654
1655             if(dy!=0){//Non horizontal
1656
1657                 if(dy>0){//Partie haute, donc de 0 à pi/2
1658
1659                     if(dx >= dy){//Donc 0 à pi/4, octant numéro 1
1660
1661                         int e = dx;
1662                         dx = e*2;
1663                         dy = dy*2;
1664
1665                         while(p1->x != p2->x){
1666                             draw_position(p1,scene, color);
1667                             e = e - dy;
1668                             if(e<0){
1669                                 p1->y++;
1670                                 e = e + dx;
1671                             }
1672                             p1->x++;
1673                         }
1674                     }
1675                 }
1676             }
1677         }
1678         else {//donc de pi/4 à pi/2, octant numéro 2 (symétrique
par rapport à x=y)
1679
1680             int e = dy;
1681             dx = dx*2;
1682             dy = e*2;
1683
1684             while(p1->y != p2->y + 1){
1685                 draw_position(p1,scene, color);
1686                 e = e - dx;
1687                 if(e<0){
1688                     p1->x++;
1689                     e = e + dy;
1690                 }
1691                 p1->y++;
1692             }
1693

```

```

1693     }
1694 }
1695 }
1696 else{//Donc Partie basse (toujours gauche)
1697
1698     if(dx>=-dy){//Octant numéro 8
1699
1700         int e = dx;
1701         dx = e*2;
1702         dy = dy*2;
1703
1704         while(p1->x != p2->x + 1){
1705             draw_position(p1,scene, color);
1706             e = e + dy;
1707             if(e<0){
1708                 p1->y--;
1709                 e = e + dx;
1710             }
1711             p1->x++;
1712         }
1713     }
1714 }
1715 else{//Octant numéro 7
1716
1717     int e = dy;
1718     dx = dx*2;
1719     dy = e*2;
1720
1721     while(p1->y != p2->y - 1){
1722         draw_position(p1,scene, color);
1723         e = e + dx;
1724         if(e>0){
1725             p1->x++;
1726             e = e + dy;
1727         }
1728         p1->y--;
1729     }
1730 }
1731 }
1732 }
1733 }
1734 }
1735 else{//Horizontal à droite
1736
1737     while(p1->x != p2->x + 1){
1738         draw_position(p1,scene, color);
1739         p1->x++;
1740     }
1741 }
1742 }
1743 }
1744 }
1745 else{//Partie Guauche
1746
1747     if(dy!=0){//Non Horizontal
1748
1749         if(dy>0){//Partie Haute-Guauche
1750
1751             if(-dx >= dy){//Octant numéro 4
1752
1753                 int e = dx;
1754                 dx = e*2;
1755

```

```

1755         dy = dy*2;
1756
1757         while(p1->x != p2->x - 1){
1758             draw_position(p1,scene, color);
1759             e = e + dy;
1760             if(e>=0){
1761                 p1->y++;
1762                 e = e + dx;
1763             }
1764             p1->x--;
1765         }
1766     }
1767 }
1768 else{//Octane numéro 3
1769
1770     int e = dy;
1771     dx = dx*2;
1772     dy = e*2;
1773
1774     while(p1->y != p2->y + 1){
1775         draw_position(p1,scene, color);
1776         e = e + dx;
1777         if(e<=0){
1778             p1->x--;
1779             e = e + dy;
1780         }
1781         p1->y++;
1782     }
1783
1784 }
1785
1786 }
1787 else{
1788
1789     if(dx<=dy){//Octant numéro 5
1790
1791         int e = dx;
1792         dx = e*2;
1793         dy = dy*2;
1794
1795         while(p1->x != p2->x - 1){
1796             draw_position(p1,scene, color);
1797             e = e - dy;
1798             if(e>=0){
1799                 p1->y--;
1800                 e = e + dx;
1801             }
1802             p1->x--;
1803         }
1804     }
1805 }
1806 else{//Octant numéro 6
1807
1808     int e = dy;
1809     dx = dx*2;
1810     dy = e*2;
1811
1812     while(p1->y != p2->y - 1){
1813         draw_position(p1,scene, color);
1814         e = e - dx;
1815         if(e>=0){
1816             p1->x--;
1817
1818             e = e + dy;
1819         }
1820         p1->y--;
1821     }
1822 }
1823
1824 }
1825
1826 }
1827 else{//Horizontale Gauche
1828
1829     while (p1->x != p2->x - 1){
1830         draw_position(p1,scene,color);
1831         p1->x--;
1832     }
1833 }
1834
1835 }
1836
1837 }
1838 }
1839 else{//Vertical
1840
1841     if(dy!=0){
1842
1843         if(dy>0){
1844
1845             while (p1->y != p2->y + 1){
1846                 draw_position(p1,scene,color);
1847                 p1->y++;
1848             }
1849
1850         }
1851         else{
1852
1853             while (p1->y != p2->y - 1){
1854                 draw_position(p1,scene,color);
1855                 p1->y--;
1856             }
1857
1858         }
1859
1860     }
1861     else{//p1 = p2
1862
1863         draw_position(p1,scene,color);
1864
1865     }
1866 }
1867 }
1868
1869 delete_position(p1);
1870 }
1871 }
1872
1873 -----
1874
1875
1876
1877 <From old_function/k_simulation.c>
1878
1879
1880
1881 void k_simulation(int k,int number,int behave_start,int behave_end,
1882 int accuracy){
1883     guard_list** storage = malloc(k*sizeof(guard_list*));
1884     for(int i=0; i<k; i++){
1885         storage[i] = new_guard_list(number,0,ROTATION_SPEED,1,4);
1886     }
1887
1888     for(int behave=behave_start; behave<=behave_end; behave++){
1889         guard_list** lab = copy_lab(storage,k,behave);
1890         char* file = new_name(behave,number);
1891         FILE* score = fopen(file,"w");
1892
1893         for(int i=0; i<k; i++){
1894             scene_object* scene = new_scene(WIDTH,HEIGHT,INIT);
1895             player* bot = new_player(WIDTH/2,0);
1896             if(behave == 0){
1897                 fprintf(score,"%d\\
1898 n",simulate(scene,lab[i],bot,3,accuracy,false,NULL));
1899             }
1900             else{
1901                 fprintf(score,"%d\\
1902 n",simulate(scene,lab[i],bot,10,accuracy,false,NULL));
1903             }
1904             //multithreads ici
1905
1906             delete_scene(scene);
1907             delete_player(bot);
1908         }
1909         delete_lab(lab,k);
1910         fclose(score);
1911         free(file);
1912     }
1913     delete_lab(storage,k);
1914 }
1915
1916 -----
1917
1918 <From old_function/multicore_path.c>
1919
1920
1921 void multicore_path(player* bot, scene_object** future, int time, int
1922 accuracy,
1923 stack* closedList, priority_list* openList, int
1924 nb_threads){
1925     pthread_t* core = malloc(nb_threads*sizeof(pthread_t));
1926     pthread_mutex_t m_prio = PTHREAD_MUTEX_INITIALIZER;
1927     pthread_mutex_t m_stack = PTHREAD_MUTEX_INITIALIZER;
1928
1929     position* end = new_position(bot->pos->x,future[0]->height-1);
1930     position* next = NULL;
1931     A_arg arg =
1932 {future,time,accuracy,end,closedList,openList,&m_prio,&m_stack};
1933
1934     for(int i = 0; i < nb_threads; i++){

```



```

2078             (tomorrow->grid[bot->pos->y][bot->pos->x-
1] == tomorrow->init)){
2079
2080             int left_range = abs(today->width/2 - bot->pos->x-1);
2081             int right_range = abs(today->width/2 - bot->pos-
>x+1);
2082             if(left_range < right_range){
2083                 move_left(bot->pos,today);
2084             }
2085             else{
2086                 move_right(bot->pos,today);
2087             }
2088         }
2089         else if((today->grid[bot->pos->y][bot->pos->x+1] ==
today->init) &&
2090             (tomorrow->grid[bot->pos->y][bot->pos->
>x+1] == tomorrow->init)){
2091             move_right(bot->pos,today);
2092         }
2093         else if((today->grid[bot->pos->y][bot->pos->x-1] ==
today->init) &&
2094             (tomorrow->grid[bot->pos->y][bot->pos->x-
1] == tomorrow->init)){
2095             move_left(bot->pos,today);
2096         }
2097         else{
2098             move_down(bot->pos,today);
2099         }
2100     }
2101 }
2102 }
2103
2104 -----
-----
2105
2106
2107
2108 <From old_function/neighbors.c>
2109
2110
2111
2112 #include "../entity/player.h"
2113
2114 list_position* neighbors(scene_object* scene, position* pos, int
height, int width){
2115     list_position* list = new_list_position(5);
2116     int cpt=0;
2117
2118     if(pos->x == 0){
2119         if(pos->y == 0){
2120             if(scene->grid[pos->y][pos->x] == scene->init){
2121                 list->tab[cpt] = new_position(pos->x,pos->y);
2122                 cpt++;
2123             }
2124             if(scene->grid[pos->y+1][pos->x] == scene->init){
2125                 list->tab[cpt] = new_position(pos->x,pos->y+1);
2126                 cpt++;
2127             }
2128             if(scene->grid[pos->y][pos->x+1] == scene->init){
2129                 list->tab[cpt] = new_position(pos->x+1,pos->y);
2130                 cpt++;

```

```

2131         }
2132     }
2133     else if(pos->y == height-1){
2134         if(scene->grid[pos->y][pos->x] == scene->init){
2135             list->tab[cpt] = new_position(pos->x,pos->y);
2136             cpt++;
2137         }
2138         if(scene->grid[pos->y-1][pos->x] == scene->init){
2139             list->tab[cpt] = new_position(pos->x,pos->y-1);
2140             cpt++;
2141         }
2142         if(scene->grid[pos->y][pos->x+1] == scene->init){
2143             list->tab[cpt] = new_position(pos->x+1,pos->y);
2144             cpt++;
2145         }
2146     }
2147     else{
2148         if(scene->grid[pos->y][pos->x] == scene->init){
2149             list->tab[cpt] = new_position(pos->x,pos->y);
2150             cpt++;
2151         }
2152         if(scene->grid[pos->y+1][pos->x] == scene->init){
2153             list->tab[cpt] = new_position(pos->x,pos->y+1);
2154             cpt++;
2155         }
2156         if(scene->grid[pos->y-1][pos->x] == scene->init){
2157             list->tab[cpt] = new_position(pos->x,pos->y-1);
2158             cpt++;
2159         }
2160         if(scene->grid[pos->y][pos->x+1] == scene->init){
2161             list->tab[cpt] = new_position(pos->x+1,pos->y);
2162             cpt++;
2163         }
2164     }
2165 }
2166 else if(pos->x == width-1){
2167     if(pos->y == 0){
2168         if(scene->grid[pos->y][pos->x] == scene->init){
2169             list->tab[cpt] = new_position(pos->x,pos->y);
2170             cpt++;
2171         }
2172         if(scene->grid[pos->y+1][pos->x] == scene->init){
2173             list->tab[cpt] = new_position(pos->x,pos->y+1);
2174             cpt++;
2175         }
2176         if(scene->grid[pos->y][pos->x-1] == scene->init){
2177             list->tab[cpt] = new_position(pos->x-1,pos->y);
2178             cpt++;
2179         }
2180     }
2181     else if(pos->y == height-1){
2182         if(scene->grid[pos->y][pos->x] == scene->init){
2183             list->tab[cpt] = new_position(pos->x,pos->y);
2184             cpt++;
2185         }
2186         if(scene->grid[pos->y+1][pos->x] == scene->init){
2187             list->tab[cpt] = new_position(pos->x,pos->y+1);
2188             cpt++;
2189         }
2190         if(scene->grid[pos->y][pos->x-1] == scene->init){
2191             list->tab[cpt] = new_position(pos->x-1,pos->y);
2192             cpt++;

```

```

2193         }
2194     }
2195     else{
2196         if(scene->grid[pos->y][pos->x] == scene->init){
2197             list->tab[cpt] = new_position(pos->x,pos->y);
2198             cpt++;
2199         }
2200         if(scene->grid[pos->y+1][pos->x] == scene->init){
2201             list->tab[cpt] = new_position(pos->x,pos->y+1);
2202             cpt++;
2203         }
2204         if(scene->grid[pos->y-1][pos->x] == scene->init){
2205             list->tab[cpt] = new_position(pos->x,pos->y-1);
2206             cpt++;
2207         }
2208         if(scene->grid[pos->y][pos->x-1] == scene->init){
2209             list->tab[cpt] = new_position(pos->x-1,pos->y);
2210             cpt++;
2211         }
2212     }
2213 }
2214 else{
2215     if(pos->y == 0){
2216         if(scene->grid[pos->y][pos->x] == scene->init){
2217             list->tab[cpt] = new_position(pos->x,pos->y);
2218             cpt++;
2219         }
2220         if(scene->grid[pos->y+1][pos->x] == scene->init){
2221             list->tab[cpt] = new_position(pos->x,pos->y+1);
2222             cpt++;
2223         }
2224         if(scene->grid[pos->y][pos->x+1] == scene->init){
2225             list->tab[cpt] = new_position(pos->x+1,pos->y);
2226             cpt++;
2227         }
2228         if(scene->grid[pos->y][pos->x-1] == scene->init){
2229             list->tab[cpt] = new_position(pos->x-1,pos->y);
2230             cpt++;
2231         }
2232     }
2233 }
2234 else if(pos->y == height-1){
2235     if(scene->grid[pos->y][pos->x] == scene->init){
2236         list->tab[cpt] = new_position(pos->x,pos->y);
2237         cpt++;
2238     }
2239     if(scene->grid[pos->y-1][pos->x] == scene->init){
2240         list->tab[cpt] = new_position(pos->x,pos->y-1);
2241         cpt++;
2242     }
2243     if(scene->grid[pos->y][pos->x+1] == scene->init){
2244         list->tab[cpt] = new_position(pos->x+1,pos->y);
2245         cpt++;
2246     }
2247     if(scene->grid[pos->y][pos->x-1] == scene->init){
2248         list->tab[cpt] = new_position(pos->x-1,pos->y);
2249         cpt++;
2250     }
2251 }
2252 else{
2253     if(scene->grid[pos->y][pos->x] == scene->init){
2254         list->tab[cpt] = new_position(pos->x,pos->y);

```

```

2255         cpt++;
2256     }
2257     if(scene->grid[pos->y+1][pos->x] == scene->init){
2258         list->tab[cpt] = new_position(pos->x,pos->y+1);
2259         cpt++;
2260     }
2261     if(scene->grid[pos->y-1][pos->x] == scene->init){
2262         list->tab[cpt] = new_position(pos->x,pos->y-1);
2263         cpt++;
2264     }
2265     if(scene->grid[pos->y][pos->x+1] == scene->init){
2266         list->tab[cpt] = new_position(pos->x+1,pos->y);
2267         cpt++;
2268     }
2269     if(scene->grid[pos->y][pos->x-1] == scene->init){
2270         list->tab[cpt] = new_position(pos->x-1,pos->y);
2271         cpt++;
2272     }
2273 }
2274 }
2275 list->size = cpt;
2276 return list;
2277 }
2278
2279
-----
2280
2281
2282
2283 <From old_function/threads.c>
2284
2285
2286
2287 #include <stdio.h>
2288 #include "threads.h"
2289 #include "graph.h"
2290
2291 void* multicore_A_star(void* args){
2292     scene_object** scene_list = ((A_arg*)args)->future;
2293     int time = ((A_arg*)args)->time;
2294     int accuracy = ((A_arg*)args)->accuracy;
2295     position* default_end = ((A_arg*)args)->end;
2296     stack* closedList = ((A_arg*)args)->closedList;
2297     priority_list* openList = ((A_arg*)args)->openList;
2298     pthread_mutex_t* m_prio = ((A_arg*)args)->m_prio;
2299     pthread_mutex_t* m_stack = ((A_arg*)args)->m_stack;
2300
2301     position* res = NULL;
2302
2303     int height = scene_list[0]->height;
2304     int width = scene_list[0]->width;
2305     position end = *default_end;
2306
2307     while (true){
2308         pthread_mutex_lock(m_prio);
2309         data* u;
2310         bool is_empty = empty_priority_list(openList);
2311         if(!is_empty){
2312             u = openList->tas[1];
2313             if(u->way != NULL && u->way->start != NULL){
2314                 end.x = u->way->start->pos.x;

```

```

2315     }
2316     else{
2317         end.x = default_end->x;
2318     }
2319
2320     if((u->node->x == end.x && u->node->y == end.y) || u->cout >=
accuracy-1){
2321         position* res = copy_position(&(u->way->end->pos));
2322         pthread_mutex_unlock(m_prio);
2323
2324         pthread_exit((void*)res);
2325     }
2326
2327     u = remove_rac(openList);
2328 }
2329 pthread_mutex_unlock(m_prio);
2330
2331     if(!is_empty){
2332         list_position* voisins
2333             = neighbors(scene_list[(time+u->cout+1)%accuracy],u-
>node,height,width);
2334         for(int i = 0; i<voisins->size;i++){
2335             position* elt = voisins->tab[i];
2336
2337             pthread_mutex_lock(m_prio);
2338             bool condition = !(is_in_stack(elt,u->cout+1,closedList) ||
is_in_priority(openList,elt,u->cout+1));
2339             pthread_mutex_unlock(m_prio);
2340
2341
2342             if(condition){
2343                 elt = copy_position(elt);
2344                 int d = range(&end,elt);
2345                 queue_position* new_way = copy_queue(u->way);
2346                 enqueue(new_way,new_queue_elt(*elt));
2347                 data* v = new_data(u->cout+1+d,elt,new_way,u->cout+1);
2348
2349                 pthread_mutex_lock(m_prio);
2350                 insert(v,openList);
2351                 pthread_mutex_unlock(m_prio);
2352             }
2353         }
2354
2355         delete_list_position(voisins);
2356
2357         pthread_mutex_lock(m_stack);
2358         enstack(new_stack_elt(copy_position(u->node),u-
>cout),closedList);
2359         pthread_mutex_unlock(m_stack);
2360
2361         delete_data(u);
2362     }
2363     else{
2364         break;
2365     }
2366 }
2367
2368     pthread_exit((void*)res);
2369 }
2370
2371
-----
2372

```

```

2373
2374
2375 <From scene/scene.c>
2376
2377
2378
2379 #include <stdio.h>
2380 #include <stdlib.h>
2381 #include "scene.h"
2382 #include "../tools/out.h"
2383
2384 scene_object* new_scene(int width, int height, int init){
2385     scene_object* scene = malloc(sizeof(scene_object));
2386     int** grid = malloc(height*sizeof(int*));
2387     for(int i=0;i<height;i++){
2388         int* row = calloc(width,sizeof(int));
2389         grid[i] = row;
2390     }
2391     scene->height = height;
2392     scene->width = width;
2393     scene->grid = grid;
2394     scene->time = 0;
2395     scene->init = init;
2396     return scene;
2397 }
2398
2399 void scene_init(scene_object* scene,int value){
2400     for(int i=0;i<scene->height;i++){
2401         for(int j=0;j<scene->width;j++){
2402             scene->grid[i][j] = value;
2403         }
2404     }
2405 }
2406
2407 void delete_scene(scene_object* scene){
2408     for(int i=0; i< scene->height;i++){
2409         free(scene->grid[i]);
2410     }
2411     free(scene->grid);
2412     free(scene);
2413 }
2414
2415 void wait(scene_object* scene){
2416     scene->time++;
2417 }
2418
2419 scene_object* copy_scene(scene_object* scene){
2420     scene_object* res = new_scene(scene->width,scene->height,scene-
>init);
2421     for(int i=0; i<res->height;i++){
2422         for(int j=0; j<res->width;j++){
2423             res->grid[i][j] = scene->grid[i][j];
2424         }
2425     }
2426     res->init=scene->init;
2427     return res;
2428 }
2429
2430
-----
2431

```

```

2432
2433
2434 <From scene/simulation.c>
2435
2436
2437
2438 #include <stdio.h>
2439 #include <stdlib.h>
2440 #include <string.h>
2441 #include "../tools/draw.h"
2442 #include "../tools/out.h"
2443 #include "simulation.h"
2444 #include "../tools/graph.h"
2445
2446 int simulate(scene_object* scene, guard_list* guard_list, player*
bot, int speed,
2447             int accuracy, bool export, FILE* output){
2448
2449     int limit_time = scene->height*speed;
2450     int time = 0;
2451
2452     scene_object** future = calloc(accuracy, sizeof(scene_object*));
2453     for(time=0; time<accuracy; time++){
2454         future[time] = copy_scene(scene);
2455         future[time]->time = time;
2456         behave(guard_list,future[time],time);
2457     }
2458
2459     stack* closedList = new_stack();
2460     priority_list* openList = new_priority_list(1);
2461     position* start = copy_position(bot->pos);
2462     tree_node* rac = new_tree_node(*start,NULL,NULL);
2463     data* data_start = new_data(scene->height,start,rac,0);
2464     insert(data_start,openList);
2465
2466     for(time = 0;time<limit_time;time++){
2467         draw_position(bot->pos,scene,2);
2468         // draw_position(guard_list->tab[0]->pos,scene,2);
2469         // printf("Avancement : %d %c ; time = %d/%d\n"
2470             //      ,100*bot->pos->y/scene-
>height,'% ',time,limit_time);
2471
2472         int curent_value = future[time%accuracy]->grid[bot->pos->y]
[bot->pos->x];
2473         if(curent_value == guard_list->color
|| curent_value == guard_list->color2){
2474             if(export){
2475                 draw_position(bot->pos,future[time%accuracy],2);
2476                 display_scene(future[time%accuracy],output);
2477             }
2478             time = -time;
2479             break;
2480         }
2481     }
2482     if(bot->pos->y >= scene->height-1){
2483         break;
2484     }
2485
2486     if(export){
2487         draw_position(bot->pos,future[time%accuracy],2);
2488         display_scene(future[time%accuracy],output);
2489     }
2490

```

```

2491         behave(guard_list,future[(time+accuracy-1)%accuracy],time+accuracy-
1);
2492
2493         path(bot,future,time,accuracy,closedList,openList);
2494         if(export){
2495             print_trajectory(future[(time+1)%accuracy],openList,5);
2496         }
2497         position end = {bot->pos->x,scene->height-1};
2498         adjust_priority_list(openList,bot->pos,end);
2499         adjust_stack(closedList,time);
2500     }
2501
2502     for(int i=0; i<accuracy; i++){
2503         delete_scene(future[i]);
2504     }
2505
2506     free(future);
2507     if(export){
2508         display_scene(scene,output);
2509     }
2510     delete_tree_node_rac(rac);
2511     delete_priority_list(openList);
2512     delete_stack(closedList);
2513     if(time == limit_time){
2514         time = -time;
2515     }
2516     return time;
2517 }
2518
-----
2519
2520
2521 <From tests/tests.c>
2522
2523
2524
2525
2526 #include <stdio.h>
2527 #include <stdlib.h>
2528 #include <math.h>
2529 #include <time.h>
2530 #include "../values.h"
2531 #include "../tools/draw.h"
2532 #include "../scene/scene.h"
2533 #include "../entity/player.h"
2534 #include "../entity/guard.h"
2535 #include "../scene/simulation.h"
2536 #include "../tools/out.h"
2537 #include "../entity/position.h"
2538
2539 void test_scene(){
2540     scene_object* scene = new_scene(10,5,0);
2541     scene_init(scene,scene->init);
2542     display_scene(scene,stdout);
2543     delete_scene(scene);
2544 }
2545
2546 void test_player(){
2547     player* bot = new_player(0,0);
2548     delete_player(bot);
2549 }
2550

```

```

2551 void test_guard(){
2552     guard_list* guards = new_guard_list(10,0,ROTATION_SPEED,1,4);
2553     delete_guard_list(guards);
2554 }
2555
2556 void test_postion(){
2557     scene_object* scene = new_scene(10,5,0);
2558     position* pos = new_position(0,0);
2559     scene_init(scene,scene->init);
2560
2561     draw_position(pos,scene,1);
2562     move_right(pos,scene);
2563     printf("position : (%d,%d)\n",pos->x,pos->y);
2564     draw_position(pos,scene,2);
2565     move_up(pos,scene);
2566     printf("position : (%d,%d)\n",pos->x,pos->y);
2567     move_down(pos,scene);
2568     printf("position : (%d,%d)\n",pos->x,pos->y);
2569     move_left(pos,scene);
2570     printf("position : (%d,%d)\n",pos->x,pos->y);
2571     draw_position(pos,scene,3);
2572     move_down(pos,scene);
2573     printf("position : (%d,%d)\n",pos->x,pos->y);
2574     move_left(pos,scene);
2575     printf("position : (%d,%d)\n",pos->x,pos->y);
2576     draw_position(pos,scene,4);
2577     display_scene(scene,stdout);
2578
2579     moveto(pos,9,4);
2580     printf("position : (%d,%d)\n",pos->x,pos->y);
2581     draw_position(pos,scene,5);
2582     move_right(pos,scene);
2583     printf("position : (%d,%d)\n",pos->x,pos->y);
2584     draw_position(pos,scene,6);
2585     display_scene(scene,stdout);
2586
2587     move_down(pos,scene);
2588     printf("position : (%d,%d)\n",pos->x,pos->y);
2589     draw_position(pos,scene,5);
2590     display_scene(scene,stdout);
2591
2592     delete_position(pos);
2593     delete_scene(scene);
2594 }
2595
2596 void test_select_circle(){//pas bon à partir de la
2597     scene_object* scene = new_scene(101,101,0);
2598     scene_init(scene,scene->init);
2599     position* pos = new_position(50,50);
2600     list_position* circle = select_circle(pos,45);
2601
2602     for(int i = 0; i < circle->size;i++){
2603         draw_position(circle->tab[i],scene,1);
2604         wait(scene);
2605         display_scene(scene,stdout);
2606     }
2607
2608     printf("Il y a %d points !!\n\n",circle->size - 1);
2609
2610     delete_position(pos);
2611     delete_list_position(circle);
2612     delete_scene(scene);

```

```

2613 }
2614
2615 void test_select_arc(){
2616     scene_object* scene = new_scene(101,101,0);
2617     scene_init(scene,scene->init);
2618     position* pos = new_position(20,20);
2619
2620     for(int j=0;j < 37;j++){
2621         list_position* arc = select_arc(pos,45,10*j,60);
2622         //modifier 360 pour voir la différence suivant les angels
2623         check_selection(arc,scene);
2624         arc = delete_double(arc);
2625         draw_position(pos,scene,0);
2626         for(int i=0; i<arc->size;i++){
2627             draw_position(arc->tab[i],scene,1);
2628         }
2629         draw_line(pos,arc->tab[arc->size/2],scene,1,4);
2630         wait(scene);
2631         display_scene(scene,stdout);
2632         scene_init(scene,scene->init);
2633         printf("Il y a %d points !!\n\n",arc->size - 1);
2634         delete_list_position(arc);
2635     }
2636
2637     delete_position(pos);
2638     delete_scene(scene);
2639 }
2640
2641 void test_draw_line(){
2642     scene_object* scene = new_scene(101,101,0);
2643     scene_init(scene,scene->init);
2644     position* pos = new_position(50,50);
2645     list_position* circle = delete_double(select_circle(pos,45));
2646
2647     for(int i = 0; i< circle->size;i++){
2648         draw_line(pos,circle->tab[i],scene,1,4);
2649         display_scene(scene,stdout);
2650         scene_init(scene,scene->init);
2651         wait(scene);
2652     }
2653
2654     delete_position(pos);
2655     delete_list_position(circle);
2656     delete_scene(scene);
2657 }
2658
2659 void test_draw_cone(){
2660     scene_object* scene = new_scene(201,201,0);
2661     scene_init(scene,scene->init);
2662     position* pos = new_position(99,50);
2663
2664     for(int i = 0; i< 37;i++){
2665         /*quick_*/draw_cone_with_cross(pos,100,10*i,60,scene,1,4);
2666         // clean_holl(scene,1,1);
2667         display_scene(scene,stdout);
2668         scene_init(scene,scene->init);
2669         wait(scene);
2670     }
2671
2672     delete_position(pos);
2673     delete_scene(scene);
2674

```

```

2675 }
2676
2677 void test_copy_scene(){
2678     scene_object* scene1 = new_scene(10,10,0);
2679     scene_object* scene2 = copy_scene(scene1);
2680
2681     scene_init(scene2, scene2->init);
2682     display_scene(scene1,stdout);
2683     display_scene(scene2,stdout);
2684
2685     delete_scene(scene1);
2686     delete_scene(scene2);
2687 }
2688
2689 void test_simulate_path(int n){
2690     int h = HEIGHT, w = WIDTH;
2691     scene_object* scene = new_scene(w,h,0);
2692     scene->init=0;
2693     scene_init(scene,scene->init);
2694     int number = 50;
2695     guard_list* guards = new_guard_list(number,6,ROTATION_SPEED,1,4);
2696     player* bot = new_player(w/2,0);
2697
2698     // printf("init angle : %d\n",guards->tab[0]->angle);
2699     // position pos = {50,50};
2700     // moveto(guards->tab[0]->pos,50,50);
2701
2702     FILE* f = fopen("./out/test.txt","w");
2703     fprintf(f,"%d\n%d\n",scene->width,scene->height);
2704     printf("Score : %d\n",simulate(scene,guards,bot,3,n,true,f));
2705     fclose(f);
2706
2707     delete_guard_list(guards);
2708     delete_player(bot);
2709     delete_scene(scene);
2710 }
2711
2712 int main(void){
2713     int seed = time(NULL);
2714     // seed = 1716130203;
2715     srand(seed);
2716     printf("seed : %d\n",seed);
2717     // test_scene();
2718     // test_player();
2719     // test_guard();
2720     // test_postion();
2721     // test_select_circle();
2722     // test_select_arc();
2723     //test_draw_line();
2724     // test_draw_cone();
2725     // test_copy_scene();
2726     test_simulate_path(1);
2727
2728     return 0;
2729 }
2730
2731 -----
2732
2733
2734
2735 <From tools/draw.c>

```

```

2736
2737
2738
2739 #include <stdio.h>
2740 #include <math.h>
2741 #include "draw.h"
2742
2743 void check_position(position* pos, scene_object* scene){
2744     int b1 = pos->x/scene->width > 0;
2745     int b2 = pos->y/scene->height > 0;
2746     if(pos->x < 0 || b1){
2747         pos->x = b1*(scene->width-1);
2748     }
2749     if(pos->y < 0 || b2){
2750         pos->y = b2*(scene->height-1);
2751     }
2752 }
2753
2754 void draw_position(position* pos, scene_object* scene, int color){
2755     check_position(pos,scene);
2756     scene->grid[pos->y][pos->x] = color;
2757 }
2758
2759 void draw_cross(position* pos, scene_object* scene, int color){
2760     int x = pos->x;
2761     int y = pos->y;
2762     int all_x[5] = {x, x+1, x-1, x, x};
2763     int all_y[5] = {y, y, y, y+1, y-1};
2764     for(int i=0; i<5; i++){
2765         position* new_pos = new_position(all_x[i], all_y[i]);
2766         draw_position(new_pos,scene,color);
2767         delete_position(new_pos);
2768     }
2769 }
2770
2771 list_position* select_circle(position* pos,int radius){
2772     int x = 0,y = radius, m = 5 - 4*radius;
2773     int cpt = 0;
2774     while(x<=y){
2775         cpt++;
2776         if(m>0){
2777             y = y - 1;
2778             m = m - 8*y;
2779         }
2780         x = x + 1;
2781         m = m + 8*x + 4;
2782     }//count
2783
2784     int size = cpt*8;
2785     list_position* res = new_list_position(size);
2786     x = 0,y = radius, m = 5 - 4*radius;
2787
2788     for(int i = 0;i<cpt;i++){
2789         // res->tab[i] = new_position(y+pos->x, x+pos->y);// 0 to
pi/4
2790         // res->tab[2*cpt - i - 1] = new_position(x+pos->x, y+pos-
>y);// pi/2 to pi/4
2791
2792         // res->tab[i + 2*cpt] = new_position(-x+pos->x, y+pos-
>y);// pi/2 to 3pi/4
2793         // res->tab[4*cpt - i - 1] = new_position(-y+pos->x, x+pos-
>y);// pi to 3pi/4

```



```

2794
2795 // res->tab[i + 4*cpt] = new_position(-y+pos->x, -x+pos->y); // pi to -3pi/4
2796 // res->tab[6*cpt - i - 1] = new_position(-x+pos->x, -y+pos->y); // -pi/2 to -3pi/4
2797
2798 // res->tab[i + 6*cpt] = new_position(x+pos->x, -y+pos->y); // -pi/2 to -pi/4
2799 // res->tab[8*cpt - i - 1] = new_position(y+pos->x, -x+pos->y); // 0 to -pi/4
2800
2801 for(int j =0; j<8;j++){
2802     int var1 = (j&1) ? (j+1)*cpt - i - 1 : j*cpt + i ;
2803     int var2_ = (!(j>>1)&1)^(j&1))*y + (((j>>1)&1)^(j&1))*x;
2804     int var2 = (j>1 && j<6) ? -var2_ : var2_ ;
2805     int var3_ = (!(j>>1)&1)^(j&1))*x + (((j>>1)&1)^(j&1))*y;
2806     int var3 = (j<4) ? var3_ : -var3_ ;
2807     res->tab[var1] = new_position(var2 + pos->x,var3 + pos->y);
2808 }
2809
2810 if(m>0){
2811     y = y - 1;
2812     m = m - 8*y;
2813 }
2814 x = x + 1;
2815 m = m + 8*x + 4;
2816 }
2817
2818 res->size = size;
2819
2820 return res;
2821 }
2822
2823 list_position* select_arc(position* pos, int radius, int direction, int angle){
2824     list_position* circle = select_circle(pos, radius);
2825     direction = (360+direction%360)%360;
2826     int octant = circle->size/8;
2827
2828     int start=0;
2829     float pourcentage = angle/720.0;
2830     int nb_points = circle->size * pourcentage;
2831
2832     if(direction !=0){
2833         int dire_45 = (direction-1)/45;
2834         start = (7 - dire_45)*octant;
2835         int condition1 = 45<direction && dire_45 == 1;
2836         int condition2 = 225<direction && dire_45 == 5;
2837         if(condition1 || condition2){
2838             start++;
2839         }
2840     }
2841
2842     // if(325 < direction) start = 0; //switch
2843     // else if(270 < direction) start = octant;
2844     // else if(direction == 270) start = 2*octant;
2845     // else if(225 < direction) start = 2*octant+1;
2846     // else if(180 < direction) start = 3*octant;
2847     // else if(135 < direction) start = 4*octant;
2848     // else if(90 < direction) start = 5*octant;
2849     // else if(direction == 90) start = 6*octant;

```

```

2850 // else if(45 < direction) start = 6*octant+1;
2851 // else if(0 < direction) start = 7*octant;
2852
2853 position* mid = circle->tab[start];
2854 int num = start;
2855
2856 if(direction != 90 && direction != 270){
2857     double tan_direction = tan(direction*3.14/180);
2858     double tan_mid = (double)(pos->y - mid->y)/(pos->x - mid->x);
2859     for(int i = 0; i < octant;i++){
2860         double tan_point = (double)(pos->y - circle->tab[start+i]->y)/
2861             (circle->tab[start+i]->x - pos->x);
2862         if(fabs(tan_mid - tan_direction) > fabs(tan_point -
tan_direction)){
2863             tan_mid = tan_point;
2864             num = start + i;
2865             mid = circle->tab[num];
2866         }
2867     }
2868 }
2869
2870 list_position* res = new_list_position(2*nb_points+1);
2871 res->tab[nb_points] = copy_position(mid);
2872
2873 for(int i = 0; i<nb_points;i++){
2874     position* pos1 = circle->tab[(circle->size + num - i)%circle->size];
2875     position* pos2 = circle->tab[(i+num)%circle->size];
2876     res->tab[nb_points - i - 1] = copy_position(pos1);
2877     res->tab[1 + i + nb_points] = copy_position(pos2);
2878 }
2879
2880 delete_list_position(circle);
2881
2882 return res;
2883 }
2884
2885 void check_selection(list_position* arc, scene_object* scene) {
2886     for(int i = 0;i<arc->size;i++) {
2887         check_position(arc->tab[i],scene);
2888     }
2889 }
2890 }
2891
2892 list_position* delete_double(list_position* arc){
2893     int nb_doubles = 0;
2894     for(int i = 1; i <arc->size;i++){
2895         if (equals(arc->tab[i-1],arc->tab[i])){
2896             nb_doubles++;
2897         }
2898     }
2899
2900     list_position* res = new_list_position(arc->size - nb_doubles);
2901     res->tab[0] = copy_position(arc->tab[0]);
2902     int nb_deleted = 0;
2903
2904     for(int i = 1; i <arc->size;i++){
2905         if (!equals(arc->tab[i-1],arc->tab[i])){
2906             res->tab[i-nb_deleted] = copy_position(arc->tab[i]);
2907         }
2908         else{
2909             nb_deleted++;
2910         }

```

```

2911     }
2912
2913     delete_list_position(arc);
2914     return res;
2915 }
2916
2917 void _draw_line(position* p1, position* p2, scene_object* scene,
2918     int color,int color2, bool cross){
2919
2920     int dx = abs(p2->x - p1->x);
2921     int dy = abs(p2->y - p1->y);
2922     int incX = (p1->x < p2->x) ? 1 : -1;
2923     int incY = (p1->y < p2->y) ? 1 : -1;
2924     int err = dx - dy;
2925     position* p = new_position(p1->x,p1->y);
2926     draw_position(p, scene, color);
2927     do{
2928         if(!cross){
2929             draw_position(p, scene, color);
2930         }
2931         int err2 = 2 * err;
2932         if (err2 > -dy) {
2933             err -= dy;
2934             if(cross && err2 < dx){
2935                 p->y += incY;
2936                 draw_position(p, scene, color);
2937                 p->y -= incY;
2938             }
2939             p->x += incX;
2940             if(cross){
2941                 draw_position(p, scene, color);
2942             }
2943         }
2944         if (err2 < dx) {
2945             err += dx;
2946             p->y += incY;
2947             if(cross){
2948                 draw_position(p, scene, color);
2949             }
2950         }
2951     } while (!equals(p,p2));
2952     draw_position(p, scene, color2);
2953     delete_position(p);
2954 }
2955
2956 void draw_line(position* p1, position* p2, scene_object* scene,
2957     int color,int color2){
2958     _draw_line(p1,p2,scene,color,color2,false);
2959 }
2960
2961 void draw_line_in_cross(position* p1, position* p2, scene_object*
scene,
2962     int color,int color2){
2963     _draw_line(p1,p2,scene,color,color2,true);
2964 }
2965
2966 void draw_cone(position* pos, int radius, int direction, int angle,
scene_object* scene, int color, int color2){
2967
2968     list_position* arc = select_arc(pos,radius,direction,angle);
2969     check_selection(arc,scene);
2970     arc = delete_double(arc);

```

```

2972     draw_position(pos,scene,color);
2973
2974     for(int i=0;i<arc->size;i++){
2975         draw_line(pos,arc->tab[i],scene,color,color2);
2976     }
2977
2978     delete_list_position(arc);
2979 }
2980
2981 void draw_cone_with_cross(position* pos, int radius, int direction,
int angle,
2982     scene_object* scene, int color, int color2){
2983     list_position* arc = select_arc(pos,radius,direction,angle);
2984     check_selection(arc,scene);
2985     arc = delete_double(arc);
2986     draw_position(pos,scene,color);
2987
2988     draw_line(pos,arc->tab[0],scene,color,color2);
2989     draw_line(pos,arc->tab[arc->size-1],scene,color,color2);
2990     for(int i=1;i<arc->size-1;i++){
2991         draw_line_in_cross(pos,arc->tab[i],scene,color,color2);
2992     }
2993
2994     delete_list_position(arc);
2995 }
2996
2997 void quick_draw_cone(position* pos,int radius,int direction,int
angle,
2998     scene_object* scene,int color,int color2){
2999     list_position* arc = select_arc(pos,radius,direction,angle);
3000     check_selection(arc,scene);
3001     arc = delete_double(arc);
3002
3003     draw_line(pos,arc->tab[0],scene,color,color2);
3004     draw_line(pos,arc->tab[arc->size-1],scene,color,color2);
3005
3006     for(int i=0;i<arc->size;i++){
3007         draw_position(arc->tab[i],scene,color);
3008     }
3009
3010     delete_list_position(arc);
3011 }
3012
3013 void print_trajectory(scene_object* scene, priority_list* list, int
color){
3014     if(!empty_priority_list(list)){
3015         tree_node* elt = list->tas[1]->way;
3016         tree_node* prev = elt->prev;
3017         if(prev != NULL){
3018             while (prev->prev != NULL){
3019                 position pos = {elt->pos.x,elt->pos.y};
3020                 draw_position(&pos,scene,color);
3021                 elt = prev;
3022                 prev = elt->prev;
3023             }
3024         }
3025     }
3026 }
3027
3028 void clean_holl(scene_object* scene, int color, int color2){
3029     for (int i = 0; i < scene->height; i++) {
3030         for (int j = 0; j < scene->width; j++) {

```

```

3031         if (scene->grid[i][j] == scene->init) {
3032             int neighbors = 0;
3033             if (i > 0 &&
3034                 (scene->grid[i - 1][j] == color
3035                  || scene->grid[i - 1][j] == color2)) neighbors++;
3036             if (i < scene->height - 1 &&
3037                 (scene->grid[i + 1][j] == color
3038                  || scene->grid[i + 1][j] == color2)) neighbors++;
3039             if (j > 0 &&
3040                 (scene->grid[i][j - 1] == color
3041                  || scene->grid[i][j-1] == color2)) neighbors++;
3042             if (j < scene->width - 1 &&
3043                 (scene->grid[i][j + 1] == color
3044                  || scene->grid[i][j + 1] == color2)) neighbors++;
3045
3046             if (neighbors >= 3) scene->grid[i][j] = color;
3047         }
3048     }
3049 }
3050 }
3051
3052 -----
3053
3054
3055 <From tools/graph.c>
3056
3057
3058
3059 #include <stdio.h>
3060 #include <string.h>
3061 #include <stdlib.h>
3062 #include "graph.h"
3063 #include "out.h"
3064
3065 list_position* neighbors2(scene_object* scene, position* pos, int height, int
width) {
3066     list_position* list = new_list_position(5);
3067     int cpt = 0;
3068
3069     for (int dx = -1; dx <= 1; dx++) {
3070         for (int dy = -1; dy <= 1; dy++) {
3071             if (abs(dx * dy) != 1) {
3072                 int new_x = pos->x + dx;
3073                 int new_y = pos->y + dy;
3074
3075                 if (new_x >= 0 && new_x < width && new_y >= 0
&& new_y < height && scene->grid[new_y][new_x] == scene-
3076                     >init) {
3077                     list->tab[cpt++] = new_position(new_x, new_y);
3078                 }
3079             }
3080         }
3081     }
3082
3083     list->size = cpt;
3084     return list;
3085 }
3086
3087 list_position* neighbors(scene_object* scene, position* pos, int height, int
width){

```

```

3089     list_position* list = new_list_position(5);
3090     int cpt=0;
3091
3092     int positionsx[5] = {pos->x,pos->x,pos->x,pos->x+1,pos->x-1};
3093     int positionsy[5] = {pos->y-1,pos->y,pos->y+1,pos->y,pos->y};
3094
3095     for(int i = 0;i<5;i++){
3096         int new_x = positionsx[i];
3097         int new_y = positionsy[i];
3098         int condition = new_x >= 0 && new_x < width && new_y >= 0 &&
new_y < height;
3099         if (condition && scene->grid[new_y][new_x] == scene->init) {
3100             list->tab[cpt] =
new_position(positionsx[i],positionsy[i]);
3101             cpt++;
3102         }
3103     }
3104
3105     list->size = cpt;
3106     return list;
3107 }
3108
3109 position* A_star(scene_object** scene_list, int time,int accuracy,
position* default_end,
3110     stack* closedList, priority_list* openList){
3111     int height = scene_list[0]->height;
3112     int width = scene_list[0]->width;
3113     position end = *default_end;
3114
3115     while (!empty_priority_list(openList)){
3116         data* u = openList->tas[1];
3117         end.x = u->way->pos.x;
3118
3119         if((u->node->x == end.x && u->node->y == end.y) || u->cout >=
accuracy-1){
3120             position* res = copy_position(u->node);
3121             if(u->way->start != NULL){
3122                 delete_position(res);
3123                 res = copy_position(&(u->way->start->pos));
3124             }
3125             return res;
3126         }
3127
3128         u = remove_rac(openList);
3129         list_position* voisins = neighbors(scene_list[(time+u-
>cout+1)%accuracy],u->node,height,width);
3130         bool have_neighbors = false;
3131
3132         for(int i = 0; i<voisins->size;i++){
3133             position* elt = voisins->tab[i];
3134             if(!(is_in_stack(elt,u->cout+1,closedList) ||
is_in_priority(openList,elt,u->cout+1))){
3135                 have_neighbors = true;
3136                 elt = copy_position(elt);
3137                 int d = range(&end,elt);
3138                 tree_node* start = u->way->start;
3139                 if(start == NULL && u->way->prev != NULL){
3140                     start = u->way;
3141                 }
3142                 tree_node* new_way = new_tree_node(*elt,start,u->way);
3143                 add_next_tree_node(u->way,new_way);
3144                 data* v = new_data(u->cout+1+d,elt,new_way,u->cout+1);
3145

```

```

3146         insert(v,openList);
3147     }
3148 }
3149 if(!have_neighbors){
3150     delete_tree_node(u->way);//soulage la mémoire
3151 }
3152 delete_list_position(voisins);
3153 enstack(new_stack_elt(copy_position(u->node),u-
>cout),closedList);
3154 delete_data(u);
3155 }
3156 return NULL;
3157 }
3158
3159
-----
3160
3161
3162
3163 <From tools/out.c>
3164
3165
3166
3167 #include "out.h"
3168 #include <stdio.h>
3169 #include <stdlib.h>
3170
3171 char* convert_to_string(int value){
3172     switch (value)
3173     {
3174     case 0 :
3175         return " ";
3176         break;
3177     case 1 :
3178         return "1";
3179         break;
3180     case 2 :
3181         return "x";
3182         break;
3183     case 3 :
3184         return "3";
3185         break;
3186     case 4 :
3187         return "4";
3188         break;
3189     case 5 :
3190         return "5";
3191         break;
3192     case 6 :
3193         return "6";
3194         break;
3195     default:
3196         return "";
3197         break;
3198     }
3199 }
3200
3201 void display_grid(int** grid, int height, int width, FILE* file){
3202     for(int i = 0;i<height;i++){
3203         fprintf(file,"| ");
3204         for(int j=0;j<width;j++){

```

```

3205         fprintf(file,"%s ",convert_to_string(grid[i][j]));
3206     }
3207     fprintf(file,"|\n");
3208 }
3209 fprintf(file,"\n");
3210 }
3211
3212 void display_data(int** grid, int height, int width,FILE* file){
3213     fprintf(file,"0\n");
3214     for(int i=0;i<height;i++){
3215         for(int j=0;j<width;j++){
3216             if(grid[i][j]!=0){
3217                 fprintf(file,"%d;%d;%d\n",i,j,grid[i][j]);
3218             }
3219         }
3220     }
3221     fprintf(file,"C\n");
3222 }
3223
3224 void display_scene(scene_object* scene,FILE* file){
3225     if(file == stdout){
3226         fprintf(file,"Scène en : %d par %d ; numéro de tour : %d\n\n",
3227             scene->width,scene->height,scene->time);
3228         display_grid(scene->grid,scene->height,scene->width,file);
3229     }
3230     else{
3231         display_data(scene->grid,scene->height,scene->width,file);
3232     }
3233 }
3234 }
3235
3236
-----
3237
3238
3239
3240 <From tools/priority.c>
3241
3242
3243
3244 #include <stdio.h>
3245 #include <stdlib.h>
3246 #include "priority.h"
3247
3248 data* new_data(int prio, position* pos, tree_node* old, int cout){
3249     data* res = malloc(sizeof(data));
3250     res->prio = prio;
3251     res->node = pos;
3252     res->way = old;
3253     res->cout = cout;
3254     return res;
3255 }
3256
3257 void delete_data(data* e){
3258     delete_position(e->node);
3259     free(e);
3260 }
3261
3262 priority_list* new_priority_list(int n){
3263     priority_list* list = malloc(sizeof(priority_list));
3264     data** tab = calloc(n+1, sizeof(data*));
3265     list->size = n;

```

```

3266     list->tas = tab;
3267     list->tas[0] = new_data(0,0,0,0);
3268     return list;
3269 }
3270
3271 bool empty_priority_list(priority_list* list){
3272     return list->tas[0]->cout == 0;
3273 }
3274
3275 void delete_priority_list(priority_list* list){
3276     while (!empty_priority_list(list)){
3277         delete_data(remove_rac(list));
3278     }
3279     free(list->tas[0]);
3280     free(list->tas);
3281     free(list);
3282 }
3283
3284 void swap(int i,int j,priority_list* list){
3285     data* tmp = list->tas[i];
3286     list->tas[i] = list->tas[j];
3287     list->tas[j] = tmp;
3288 }
3289
3290 void percolate_up(int k,priority_list* list){
3291     if(k>1){
3292         int m = k/2;
3293         if(list->tas[m]->prio > list->tas[k]->prio){
3294             swap(m,k,list);
3295             percolate_up(m,list);
3296         }
3297     }
3298 }
3299
3300 int choose_son(int k, priority_list* list){
3301     if(k*2<list->tas[0]->cout){
3302         if(list->tas[2*k]->prio < list->tas[2*k+1]->prio){
3303             return 2*k;
3304         }
3305         else{
3306             return 2*k+1;
3307         }
3308     }
3309     else{
3310         if(2*k == list->tas[0]->cout){
3311             return 2*k;
3312         }
3313         else{
3314             return -1;
3315         }
3316     }
3317 }
3318
3319 void percolate_down(int k, priority_list* list){
3320     int m = choose_son(k,list);
3321     if(m>-1 && list->tas[m]->prio < list->tas[k]->prio){
3322         swap(m,k,list);
3323         percolate_down(m,list);
3324     }
3325 }
3326
3327 void insert(data* e,priority_list* list){

```

```

3328     if(list->tas[0]->cout < list->size){
3329         list->tas[0]->cout++;
3330         list->tas[list->tas[0]->cout] = e;
3331         percolate_up(list->tas[0]->cout,list);
3332     }
3333     else{
3334         list->tas = realloc(list->tas, (2*list-
3335 >size+1)*sizeof(data*));
3336         list->size = list->size*2;
3337         insert(e,list);
3338     }
3339
3340 data* remove_rac(priority_list* list){
3341     data* res = NULL;
3342     if(list->tas[0]->cout > 0){
3343         res = list->tas[1];
3344         list->tas[1] = list->tas[list->tas[0]->cout];
3345         list->tas[0]->cout--;
3346         percolate_down(1,list);
3347     }
3348     return res;
3349 }
3350
3351 bool is_in_priority(priority_list* list, position* pos, int cout) {
3352     for(int i = 1; i <= list->tas[0]->cout; i++){
3353         if(equals(list->tas[i]->node,pos) && list->tas[i]->cout <=
3354 cout) return true;
3355     }
3356     return false;
3357 }
3358
3359 void adjust_priority_list(priority_list* list, position* pos,
3360 position end){
3361     int n = list->tas[0]->cout;
3362     priority_list* tmp = new_priority_list(list->size);
3363
3364     for(int i = 1; i<=n; i++){
3365         if(list->tas[i]->way->start != NULL
3366 && equals(&(list->tas[i]->way->start->pos),pos)){
3367             data* e = list->tas[i];
3368             e->cout--;
3369             e->prio = e->cout+range(e->node,&end);
3370
3371             e->way->start->prev = NULL;
3372             e->way->start = new_start_tree_node(e->way);
3373
3374             insert(e,tmp);
3375         }
3376         else{
3377             delete_tree_node(list->tas[i]->way);
3378             delete_data(list->tas[i]);
3379         }
3380     }
3381     free(list->tas[0]);
3382     free(list->tas);
3383     list->tas = tmp->tas;
3384     free(tmp);
3385 }

```

```

3386
-----
3387
3388
3389
3390 <From tools/stack.c>
3391
3392
3393
3394 #include <stdio.h>
3395 #include <stdlib.h>
3396 #include "stack.h"
3397
3398 stack_elt* new_stack_elt(position* pos,int time){
3399     stack_elt* elt = malloc(sizeof(stack_elt));
3400     elt->pos = pos;
3401     elt->time = time;
3402     elt->next = NULL;
3403     return elt;
3404 }
3405
3406 void delete_stack_elt(stack_elt* elt){
3407     delete_position(elt->pos);
3408     free(elt);
3409 }
3410
3411 stack* new_stack(){
3412     stack* s = malloc(sizeof(stack));
3413     s->top = NULL;
3414     return s;
3415 }
3416
3417 void delete_stack(stack* s){
3418     while(!empty_stack(s)){
3419         delete_stack_elt(destack(s));
3420     }
3421     free(s);
3422 }
3423
3424 void enstack(stack_elt* elt,stack* s){
3425     elt->next = s->top;
3426     s->top = elt;
3427 }
3428
3429 stack_elt* destack(stack* s){
3430     if(!empty_stack(s)){
3431         stack_elt* elt = s->top;
3432         s->top = elt->next;
3433         return elt;
3434     }
3435     return NULL;
3436 }
3437
3438 bool empty_stack(stack* s){
3439     return s->top == NULL;
3440 }
3441
3442 bool is_in_stack(position* pos, int time, stack* s){
3443     stack_elt* elt = s->top;
3444     while (elt != NULL){
3445         if(equals(pos,elt->pos) && time == elt->time){
3446             return true;

```

```

3447     }
3448     elt = elt->next;
3449 }
3450 return false;
3451 }
3452
3453 void adjust_stack(stack* s, int time){
3454     stack_elt* elt = s->top;
3455     while(elt != NULL){
3456         stack_elt* next = elt->next;
3457         if(next != NULL && next->time < time){
3458             elt->next = next->next;
3459             delete_stack_elt(next);
3460         }
3461         else{
3462             elt = next;
3463         }
3464     }
3465     if(s->top != NULL && s->top->time < time){
3466         stack_elt* elt = s->top;
3467         s->top = s->top->next;
3468         delete_stack_elt(elt);
3469     }
3470 }
3471
3472
-----
3473
3474
3475
3476 <From out/sublim.py>
3477
3478
3479
3480
3481 from PIL import Image
3482 import numpy as np
3483
3484 from const import *
3485
3486 def color(n):
3487     match n:
3488         case 1:
3489             return [0,0,255]
3490         case 2:
3491             return [255,0,0]
3492         case 3:
3493             return [0,255,0]
3494         case 4:
3495             return [0,0,100]
3496         case 5:
3497             return [255,0,255]
3498         case _:
3499             return [255,255,255]
3500
3501 def minmax(t,cpt):
3502     min,max = cpt,0
3503     for l in t:
3504         for v in l:
3505             if v[2]<min:
3506                 min = v[2]

```



```

3507         elif v[2]>max:
3508             max = v[2]
3509     return min,max
3510
3511 def grids(filename,k):
3512     file = open(rac+filename,"r")
3513     width = int(file.readline())
3514     height = int (file.readline())
3515     cpt = 0
3516     img = None
3517     array = None
3518     line = file.readline()
3519     while line:
3520         if line[0] == "0":
3521             array = np.zeros([width*k,height*k,3],dtype=np.uint8)
3522             array[:,:] = [255,255,255]
3523         elif line[0] == "C":
3524             img = Image.fromarray(array)
3525             img.save(rac+"jpg/"+str(cpt)+".bmp", "bmp")
3526             cpt += 1
3527         else:
3528             values = line.split(";")
3529             v1,v2,v3=int(values[0])*k,int(values[1])*k,int(values[2])
3530             array[v1:v1+k,v2:v2+k] = color(v3)
3531         line = file.readline()
3532     file.close()
3533     return cpt
3534
3535 def mean_grid(filename,cpt,k):
3536     file = open(rac+filename,"r")
3537     width = int(file.readline())
3538     height = int (file.readline())
3539     mid = np.zeros([width,height,3],dtype=np.uint64)
3540     b = np.zeros([width,height])
3541     line = file.readline()
3542     while line:
3543         if line[0] == "0":
3544             b[:,:] = True
3545         elif line[0] != "C":
3546             values = line.split(";")
3547             v1,v2=int(values[0]),int(values[1])
3548             if b[v1,v2]:
3549                 mid[v1,v2,2] += 1
3550                 b[v1,v2] = False
3551         line = file.readline()
3552     min,max = minmax(mid,cpt)
3553     new_mid = np.zeros([width*k,height*k,3],dtype=np.uint8)
3554     for i in range(width):
3555         for j in range(height):
3556             new_mid[i*k:(i+1)*k,j*k:(j+1)*k] = [255,255,255]
3557             new_mid[i*k:(i+1)*k,j*k:(j+1)*k,1:3] = 255 -
3558             int((mid[i,j,2]-min)*(255/(max-min+1)))
3559     img = Image.fromarray(new_mid)
3560     img.save(rac+"mid.bmp")
3561     file.close()
3562
-----
-----
3563
3564
3565

```

```

3566 <From out/analyses.py>
3567
3568
3569
3570
3571 from matplotlib import pyplot as plt
3572 from os import listdir
3573
3574 from const import *
3575
3576 # def color(k):
3577 #     match k:
3578 #         case 0:
3579 #             return "r"
3580 #         case 1:
3581 #             return "g"
3582 #         case 2:
3583 #             return "y"
3584 #         case 3:
3585 #             return "m"
3586 #         case 4:
3587 #             return "b"
3588 #         case 5:
3589 #             return "c"
3590 #         case _:
3591 #             return "k"
3592
3593 def color(k):
3594     match k:
3595         case 1:
3596             return "g"
3597         case 5:
3598             return "r"
3599         case _:
3600             return "b"
3601
3602 def noms(k):
3603     match k:
3604         case 0:
3605             return "immobile"
3606         case 1:
3607             return "rotation"
3608         case 2:
3609             return "aléatoire"
3610         case 3:
3611             return "aller-retour"
3612         case 4:
3613             return "patrouille carré"
3614         case 5:
3615             return "en essaim"
3616         case 6:
3617             return "en S"
3618
3619 def fill_dico(dico):
3620     files = sorted(listdir(rac+folder))
3621     for f in files:
3622         data = f.split("_")
3623         behave = int(data[0])
3624         nb = int(data[1].split(".")[0])
3625         file = open(rac+folder+f,"r")
3626         line = file.readline()
3627         if not (behave in dico):

```

```

3628         dico[behave] = {}
3629         dico[behave][nb] = []
3630         while line:
3631             try:
3632                 l = int(line)
3633                 if l == -200 and behave == 0:
3634                     dico[behave][nb].append(-500)
3635                 else:
3636                     dico[behave][nb].append(l)
3637             except:
3638                 pass
3639             line = file.readline()
3640
3641
3642 def sum(l):
3643     sum = 0
3644     for e in l:
3645         sum += e
3646     return sum
3647
3648 def mean(l):
3649     res = sum(l)
3650     if res == 0:
3651         return 0
3652     else:
3653         return res/len(l)
3654
3655 def smooth(x,y) :
3656     k = 5
3657     new_x = []
3658     new_y = []
3659     for xi in x :
3660         new_x.append(xi)
3661     for i in range(len(y)):
3662         new_y.append(mean(y[max(i-k,0):i+k+1]))
3663     return new_x,new_y
3664
3665 def plot_mean_dico(dico,legend):
3666     plt.figure("Mean")
3667     for num_fil,d in dico.items():
3668         x = []
3669         y = []
3670         for nb,l in d.items():
3671             x.append(nb)
3672             y.append(mean(l))
3673         x,y = smooth(x,y)
3674         plt.plot(x,y,color=color(num_fil),label=legend+noms(num_fil))
3675     plt.legend()
3676     plt.xlabel("nombre de gardes")
3677     plt.ylabel("temps moyen mis par le joueur")
3678     plt.show()
3679
3680 def scatter_all_dico(dico,legend):
3681     for num_fil,d in dico.items():
3682         cpt = 0
3683         plt.figure(legend+" : "+str(num_fil))
3684         for nb,l in d.items():
3685             cpt += len(l)
3686             y = [e for e in l if e<301 and e>-301]
3687             x = [nb for i in range(len(y))]
3688             plt.scatter(x,y,color=color(num_fil))
3689         plt.xlabel("nombre de gardes")

```

```

3690     plt.ylabel("temps mis par le joueur")
3691     plt.show()
3692     print(cpt)
3693
3694 def plot_achive_rate(dico,legend):
3695     plt.figure("Rate")
3696     for num_fil,d in dico.items():
3697         x = []
3698         y = []
3699         for nb,l in d.items():
3700             x.append(nb)
3701             y.append(sum([1/len(l) for e in l if e > 0]))
3702         x,y = smooth(x,y)
3703         plt.plot(x,y,color=color(num_fil),label=legend+noms(num_fil))
3704     plt.legend()
3705     plt.xlabel("nombre de gardes")
3706     plt.ylabel("taux de réussite du joueur")
3707     plt.show()
3708
3709

```

```

-----
3710
3711
3712
3713 <From out/const.py>
3714
3715
3716
3717
3718 rac = "out/"
3719 folder = "view/"
3720
3721

```

```

-----
3722
3723
3724
3725 <From out/main.py>
3726
3727
3728
3729
3730 import glob
3731 import shutil
3732 import os
3733
3734 from sublim import *
3735 from analyses import *
3736
3737 if os.path.exists(rac+"jpg"):
3738     shutil.rmtree(rac+"jpg")
3739 os.mkdir(rac+"jpg")
3740
3741 if os.path.exists(rac+"test.txt"):
3742     cpt = grids("test.txt",3)
3743     mean_grid("test.txt",cpt,3)
3744
3745     frames = [Image.open(image) for image in
sorted(glob.glob(rac+"jpg/*.bmp"), key=lambda x:
int(os.path.basename(os.path.splitext(x)[0])))]

```

```

3746     frame_one = frames[0]
3747     frame_one.save(rac+"test.gif", format="GIF",
append_images=frames,save_all=True, duration=80, loop=1)
3748
3749
3750 if os.path.exists(rac+folder) and listdir(rac+folder) != []:
3751     dico = {}
3752     fill_dico(dico)
3753     legend = "comportement : "
3754     plot_achive_rate(dico,legend)
3755     plot_mean_dico(dico,legend)
3756     # scatter_all_dico(dico,legend)
3757
3758 # shutil.rmtree(rac+"jpg")

```