I62 - Projet

Sébastien Richard, Adrien Sablayrolles, Adam Lewis

Dernière mise à jour : 1er décembre 2023

Compte-rendu de projet de la matière I62 "Génie Logiciel", sous la direction de M. Francisci Dominique, socio-professionnel. Toutes les informations, documentations et archives du projet sont disponibles sur https://i62.richardsebastien.fr/.

Table des matières

1 Introduction **2**

2 Exigences **2**
2.1 Création de l'application . . . 2
2.2 Création du monde . . . 2
2.3 Gestion de la simulation . . . 3
2.4 Gestion des fourmis . . . 3
2.5 Gestion des ressources . . . 4

3 Axe fonctionnel **4**
3.1 Diagramme statique . . . 4
3.2 Diagramme de cas d'usage . . . 5

4 Scénario nominal **8**
4.1 Cas alternatifs . . . 9
4.2 Cas d'erreur . . . 9
4.3 Diagramme de séquence . . . 9

5 Axe statique **10**
5.1 Diagramme de classe . . . 10
5.1.1 Diagramme de classe avec packages . . . 15
5.2 Diagramme d'objets . . . 20

6 Manuel d'installation **21**
6.1 Installation de l'application . . . 21

7 Manuel utilisateur **21**
7.1 Exécuter l'application . . . 21
7.2 Application . . . 21
7.3 Gestion du monde . . . 23
7.3.1 Créer un nouveau monde . . . 23
7.3.2 Importer un monde . . . 23
7.4 Ajouter des nids et des ressources . . . 25
7.5 Quitter l'application . . . 26
7.6 Statistiques . . . 27

1 Introduction

Ce projet est un logiciel de simulation et de visualisation d'auto-organisation chez les insectes sociaux, dans notre cas les fourmis, on pourra au travers de cette application simuler la gestion des ressources, des fourmis, du monde dans lequel les fourmis vivent. Le but de ce logiciel est de pouvoir simuler et donc étudier les comportements et l'évolution d'un ensemble de fourmilières dans la recherche de ressource et la lutte entre espèces.

2 Exigences

2.1 Création de l'application

— Le système doit permettre de visualiser les règles à n'importe quel moment de la simulation.
— Le système doit permettre de mettre en pause et de reprendre la simulation.
— Le système doit permettre de choisir la vitesse de simulation.
— Le système doit permettre d'avancer pas à pas dans la simulation.

2.2 Création du monde

— Le système doit permettre le chargement d'un monde prédéfini au démarrage de celui-ci.
— Le système doit permettre le placement d'une ressource à l'aide d'un clique de souris.
— Le système doit permettre le placement d'un nid de fourmis à l'aide d'un clique de souris.
— Le système doit permettre de lire un fichier d'extension ".map" pour générer le monde.
— Tant qu'un nid ou une ressouce n'est pas encore placée, le système doit permettre la modification du monde.
— Le système vérifie s'il y a bien au moins 1 nid et 1 ressource posés dans le monde.

2.3 Gestion de la simulation

— Le système doit permettre la gestion de la vitesse d'une simulation.
— Le système doit permettre de revenir pas à pas en arrière dans la simulation.
— Le système doit permettre la gestion du cycle du jour.
— Lors d'une simulation le système doit permettre l'affichage des statistiques des différents nids et ressources.
— Le système doit permettre de désigner un vainqueur losrque toutes les ressources ont était récolté ou après un temps donné par l'utilisateur.

2.4 Gestion des fourmis

— Le système doit permettre d'utiliser les ressources pour la création de tout type de fourmi.
— Le système doit permettre de changer les paramètres des espèces.
— Le système doit permettre de créer une fourmi ayant une taille de X pixels, qui se déplace de X pixels dans toutes les directions.
— Le système doit permettre la modification de la portée de la vision des fourmis.
— Le système doit empêcher le déplacement d'une fourmi dans un mur.
— Le système doit permettre de gérer la population d'un nid déjà placé.
— Le système doit permettre que lorsque la fourmi sort du nid, celle-ci dépose une faible quantité de phéromones sur son chemin pour retrouver son nid.
— Le système doit permettre l'évaporation, à un certain taux, des phéromones à chaque itération.
— Le système doit permettre que lorsqu'une ressource est découverte, la fourmi laisse une trace plus importante de phéromones sur la trace déjà présente (renforcement).
— Lorsqu'il fait nuit, le système doit permettre à la fourmi de retrouver son chemin grâce à des traces de phéromones.
— Lorsqu'il fait nuit, le système doit obstruer la vision des fourmis.
— Le système doit permettre à la fourmi de se déplacer d'un nombre X de cases avant de devoir rentrer au nid.
— Le système doit permettre la gestion d'un conflit entre les fourmis issues de nids différents qui récoltent la même ressource.
— Le système doit permettre la différenciation visuelle des fourmis qui ne proviennent pas du même nid à l'aide de couleurs.
— Le système doit permettre de changer la fréquence d'aléas dans les mouvements des fourmis.
— Le système doit permettre la sortie de fourmi "combattante" lorsqu'il y a un conflit avec un autre nid, ce qui coûte des ressources.

— Le système doit permettre l'émision de phéromone de "bloquage" lorsqu'une fourmi reste bloqué au même endroit sans avoir recolter de ressources après un certaint temps.(La même chose si la fourmi à récolté mais n'arrive pas à retourner à son nid.)

— Le système doit permettre la détection d'une zone avec des phéromone de "blocage" alors les fourmi doivent contourner la zone.

— Le système doit permettre de laisser des traces de phéromone "Danger/Combat/..." lorsque il y a un contact avec un autre groupe de fourmi sur une ressource.

2.5 Gestion des ressources

— Lorsque qu'une ressource est placée, le système doit permettre la gestion de la richesse de cette ressource.

— Le système doit permettre que si une ressource est épuisée, les traces de phéromones s'évaporent, de moins en moins de fourmis emprunteront donc ce chemin et il finira donc par disparaître.

TABLE 1 – Index des codes des catégories des exigences

Code catégorie	N° section	Nom catégorie
CREER_APPLICATION_010	2.1	Création de l'application
CREER_MONDE_020	2.2	Création du monde
GESTION_MONDE_030	2.3	Gestion du monde
GESTION_FOURMIS_040	2.4	Gestion des fourmis
GESTION_RESSOURCES_050	2.5	Gestion des ressources

3 Axe fonctionnel

3.1 Diagramme statique

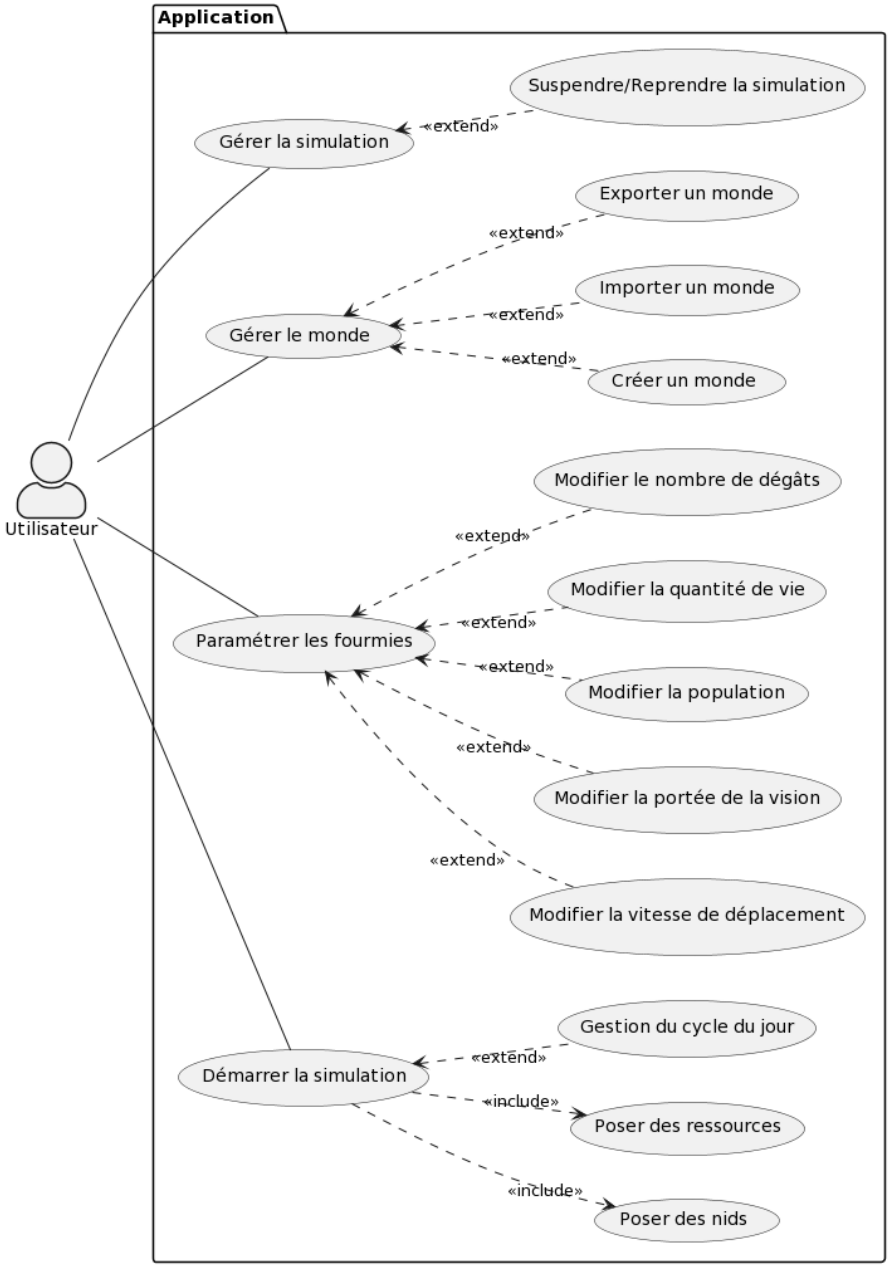
```
1 @startuml
2 left to right direction
3 :Utilisateur:--[Application]
4 @enduml
```

3.2 Diagramme de cas d'usage

Premier diagramme de cas d'usage qui ne nous a pas convaincu.
Code plantUML :

```
@startuml
skinparam actorStyle awesome
left to right direction


package Application{
usecase "Gérer la simulation" as UC1
usecase "Suspendre/Reprendre la simulation" as UC11

usecase "Gérer le monde" as UC3
usecase "Créer un monde" as UC32
usecase "Importer un monde" as UC31
usecase "Exporter un monde" as UC312

usecase "Paramétrer les fourmies" as UC4
usecase "Modifier la vitesse de déplacement" as UC41
usecase "Modifier la portée de la vision" as UC42
usecase "Modifier la population" as UC43
usecase "Modifier la quantité de vie" as UC44
usecase "Modifier le nombre de dégâts" as UC45

usecase "Démarrer la simulation" as UC5
usecase "Poser des nids" as UC51
usecase "Poser des ressources" as UC52
usecase "Gestion du cycle du jour" as UC53


}

:Utilisateur: -- UC1
:Utilisateur: -- UC3
:Utilisateur: -- UC5

UC1 <.. "<<extend>>" UC11
UC3 <.. "<<extend>>"UC32
UC3 <.. "<<extend>>"UC31
UC3 <.. "<<extend>>"UC312

UC4 <.. "<<extend>>"UC41
UC4 <.. "<<extend>>"UC42
UC4 <.. "<<extend>>"UC43
UC4 <.. "<<extend>>"UC44
UC4 <.. "<<extend>>"UC45

UC5 ..> "<<include>>"UC51
UC5 ..> "<<include>>"UC52
UC5 <.. "<<extend>>"UC53

:Utilisateur: -- UC4
@enduml
```

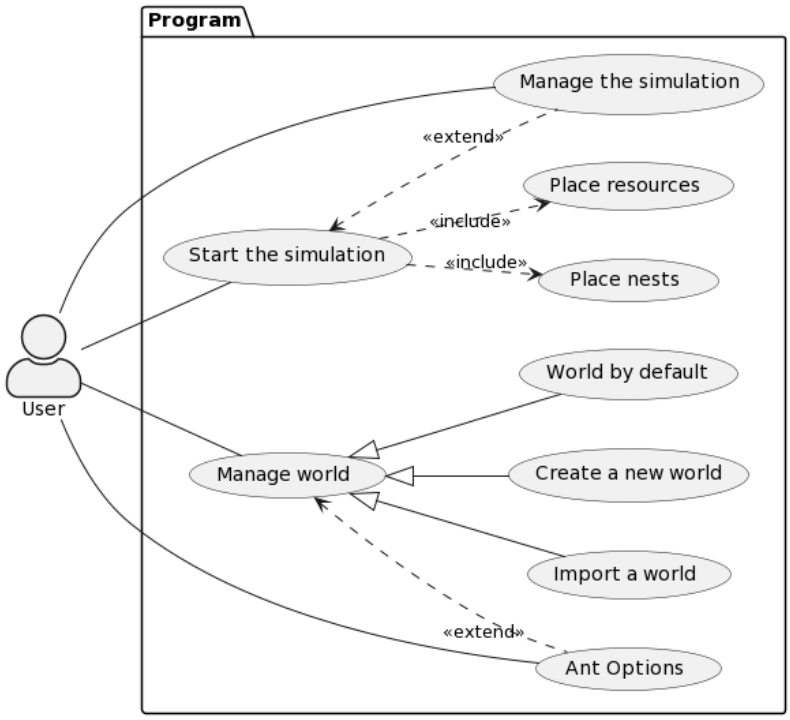
Diagramme :



Une version du diagramme de cas d'usage modifié, qui décrit mieux notre solution.
Code plantUML :

```
@startuml
skinparam actorStyle awesome
left to right direction


package Program{
usecase "Manage the simulation" as UC1

usecase "Manage world" as UC3
usecase "World by default" as UC31
usecase "Import a world" as UC32
usecase "Create a new world" as UC33


usecase "Ant Options" as UC4


usecase "Start the simulation" as UC5
usecase "Place nests" as UC51
usecase "Place resources" as UC52



}

:User: -- UC1
:User: -- UC3
:User: -- UC4
:User: -- UC5

UC3 <|-- UC31
UC3 <|-- UC32
UC3 <|-- UC33

UC3 <.. "<<extend>>"UC4
UC5 <.. "<<extend>>"UC1

UC5 ..> "<<include>>"UC51
UC5 ..> "<<include>>"UC52



@enduml
```

Diagramme :



4 Scénario nominal

Lancement de la simulation :

1. L'utilisateur lance l'application.
2. Le système demande si l'utilisateur veut un monde par défaut, importer son propre fichier ou créer un nouveau monde.
3. Si l'utilisateur choisi par défaut, l'application charge le fichier "default.map".
4. Si l'utilisateur choisi de charger un autre monde, l'application demande le fichier et le charge.
5. Si L'utilisateur choisi de créer un monde, l'application ouvre la fenêtre de création de monde, le sauvegarde une fois la création terminée puis le charge.
6. Si l'utilisateur choisi "paramétrer les fourmies"
7. Le système affiche les différentes options à modifier
8. L'utilisateur fait le choix de lancer la simulation
9. Le système vérifie si il y a bien au moins 1 nid et 1 ressource de posé dans le monde.
10. L'application lance la simulation.

4.1 Cas alternatifs

A1 : Le fichier ".map" n'est pas sous le bon format. L'enchaînement A1 commence au point 4 du scénario nominal.

5. l'application ne peux pas charger le monde
6. message d'erreur

Le scénario reprend à l'étape 2

A2 : Il manque un nid ou une resource au lancement de la simulation. L'enchaînement A2 commence au point 9 du scénario nominal.

10. l'application demande de placer aumoin un nid et une resource
11. l'utilisateur les place

Le scénario reprend à l'étape 10

A3 : L'utilisateur veux recomencer. L'enchaînement A3 commence a n'importe quel point du scénario nominal.

X. l'utilisateur réinitialise le monde

Le scénario reprend à l'étape 2.

4.2 Cas d'erreur

E1 : L'application est fermé. L'enchaînement E1 commence a n'importe quel point du scénario nominal.

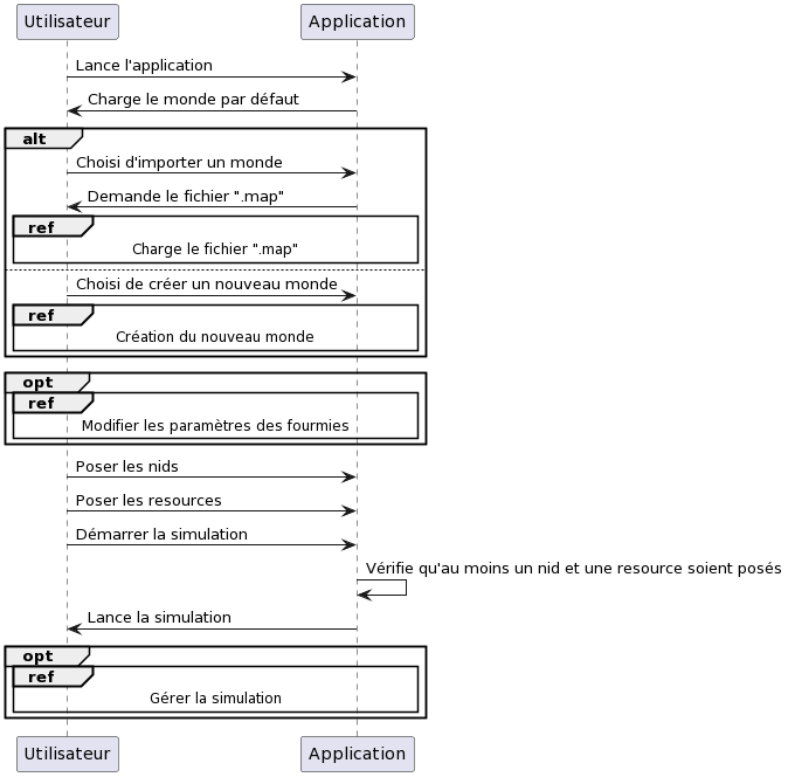
X. l'utilisateur ferme l'application ou l'application plante

L'UC se termine en échec.

4.3 Diagramme de séquence

```
@startuml
Utilisateur -> Application : Lance l'application
Utilisateur <- Application : Charge le monde par défaut

alt
Utilisateur -> Application : Choisi d'importer un monde
Utilisateur <- Application : Demande le fichier ".map"
ref over Utilisateur,Application : Charge le fichier ".map"
else
Utilisateur -> Application : Choisi de créer un nouveau monde
ref over Utilisateur,Application : Création du nouveau monde
end

Opt
ref over Utilisateur,Application : Modifier les paramètres des
    fourmies
end
Utilisateur -> Application : Poser les nids
Utilisateur -> Application : Poser les resources
Utilisateur -> Application : Démarrer la simulation
```

```
20  Application -> Application : Vérifie qu'au moins un nid et une
        resource soient posés
21  Utilisateur <- Application : Lance la simulation
22
23  Opt
24  ref over Utilisateur,Application : Gérer la simulation
25  end
26  @enduml
```



5 Axe statique

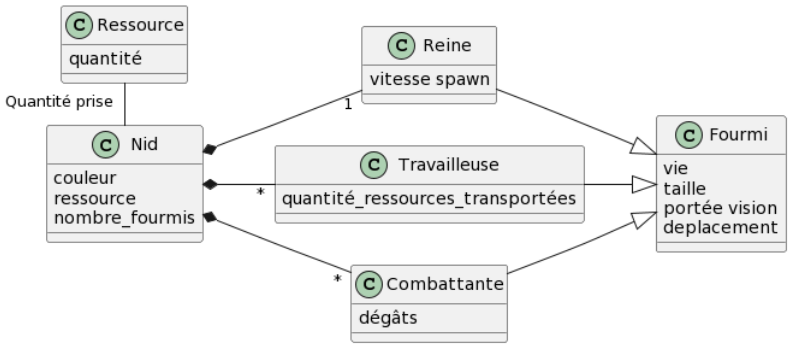
5.1 Diagramme de classe

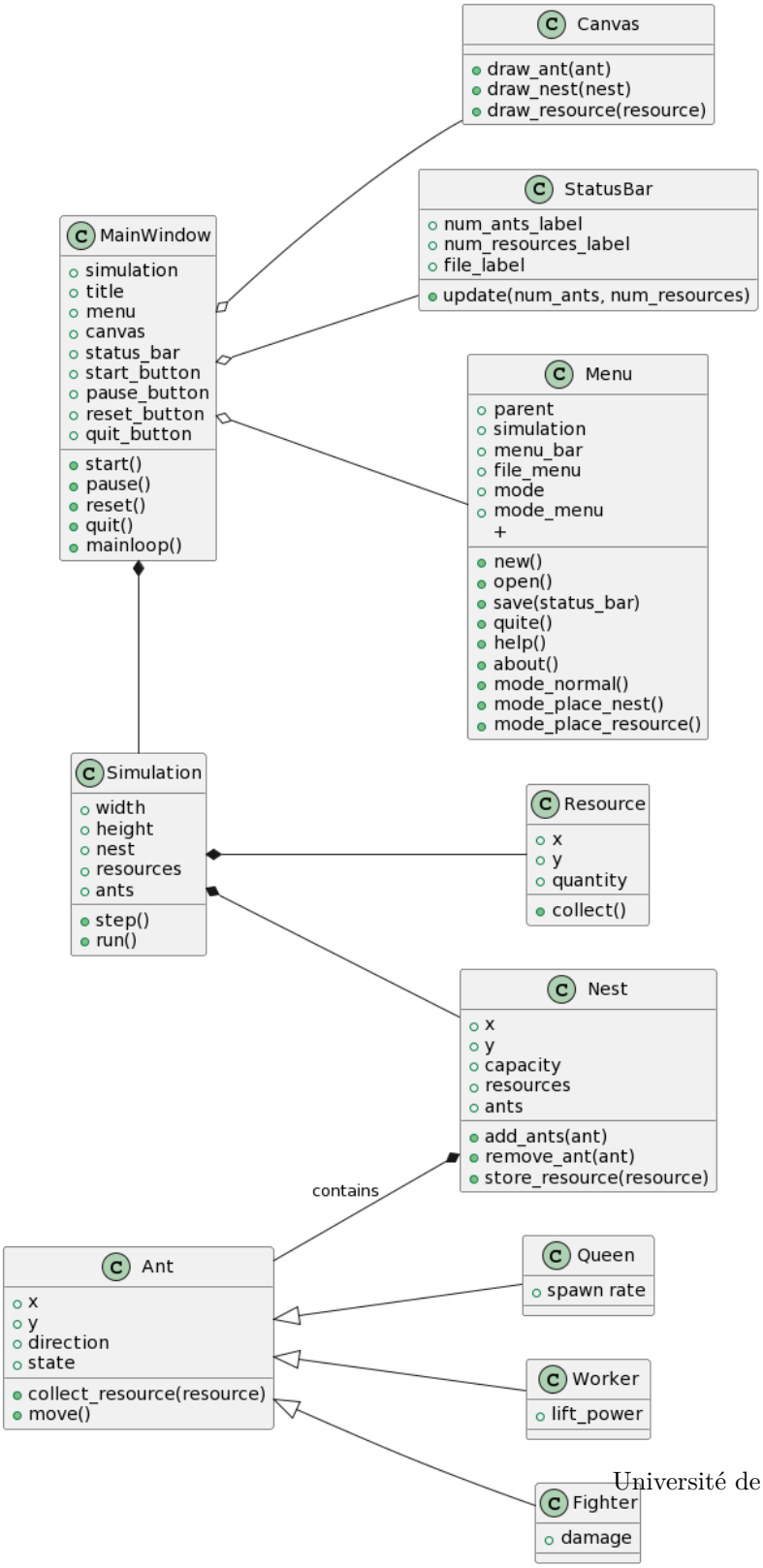
```
1  @startuml
2
3  left to right direction
4
5  class Nid{
6  couleur
7  ressource
8  nombre_fourmis
```

```
}

class Ressource{
quantité
}

class Fourmi{
vie
taille
portée vision
deplacement
}

class Reine{
vitesse spawn
}

class Travailleuse{
quantité_ressources_transportées
}

class Combattante {
dégâts
}

Ressource"Quantité prise" - Nid

Nid*-- "1"Reine
Nid*-- "*"Travailleuse
Nid*-- "*"Combattante
Reine --|> Fourmi
Travailleuse --|> Fourmi
Combattante --|> Fourmi
@enduml

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
@startuml
left to right direction
class Ant {
+ x
+ y
+direction
+state
+collect_resource(resource)
+move()
}

class MainWindow {
+simulation
+title
+menu
+canvas
+status_bar
+start_button
+pause_button
+reset_button
```

```
+quit_button

+start()
+pause()
+reset()
+quit()
+mainloop()
}
class Simulation {
+width
+height
+nest
+resources
+ants

+step()
+run()

}

class Resource {
+ x
+ y
+ quantity
+collect()
}

class Nest{
+ x
+ y
+capacity
+resources
+ants
+add_ants(ant)
+remove_ant(ant)
+store_resource(resource)
}

class Queen {
+spawn rate
}

class Worker {
+lift_power
}

class Fighter {
+damage
}

class Canvas {
+draw_ant(ant)
+draw_nest(nest)
+draw_resource(resource)
}

```

```
class StatusBar {
+num_ants_label
+num_resources_label
+file_label
+update(num_ants, num_resources)
}

class Menu {
+parent
+simulation
+menu_bar
+file_menu
+mode
+mode_menu
+new()
+open()
+save(status_bar)
+quite()
+help()
+about()
+mode_normal()
+mode_place_nest()
+mode_place_resource()
+
}


Ant --* Nest : contains
Simulation *-- Resource
Simulation *-- Nest
MainWindow *- Simulation

Ant <|-- Queen
Ant <|-- Fighter
Ant <|-- Worker

MainWindow o-- Canvas
MainWindow o-- StatusBar
MainWindow o-- Menu
@enduml
```




Canvas
draw_ant(ant)
draw_nest(nest)
draw_resource(resource)
StatusBar
num_ants_label
num_resources_label
file_label
update(num_ants, num_resources)
MainWindow
simulation
title
menu
canvas
status_bar
start_button
pause_button
reset_button
quit_button
start()
pause()
reset()
quit()
mainloop()
Menu
parent
simulation
menu_bar
file_menu
mode
mode_menu
+
new()
open()
save(status_bar)
quite()
help()
about()
mode_normal()
mode_place_nest()
mode_place_resource()
Simulation
width
height
nest
resources
ants
step()
run()
Resource
x
y
quantity
collect()
Nest
x
y
capacity
resources
ants
add_ants(ant)
remove_ant(ant)
store_resource(resource)
contains
Ant
x
y
direction
state
collect_resource(resource)
move()
Queen
spawn rate
Worker
lift_power
Fighter
damage

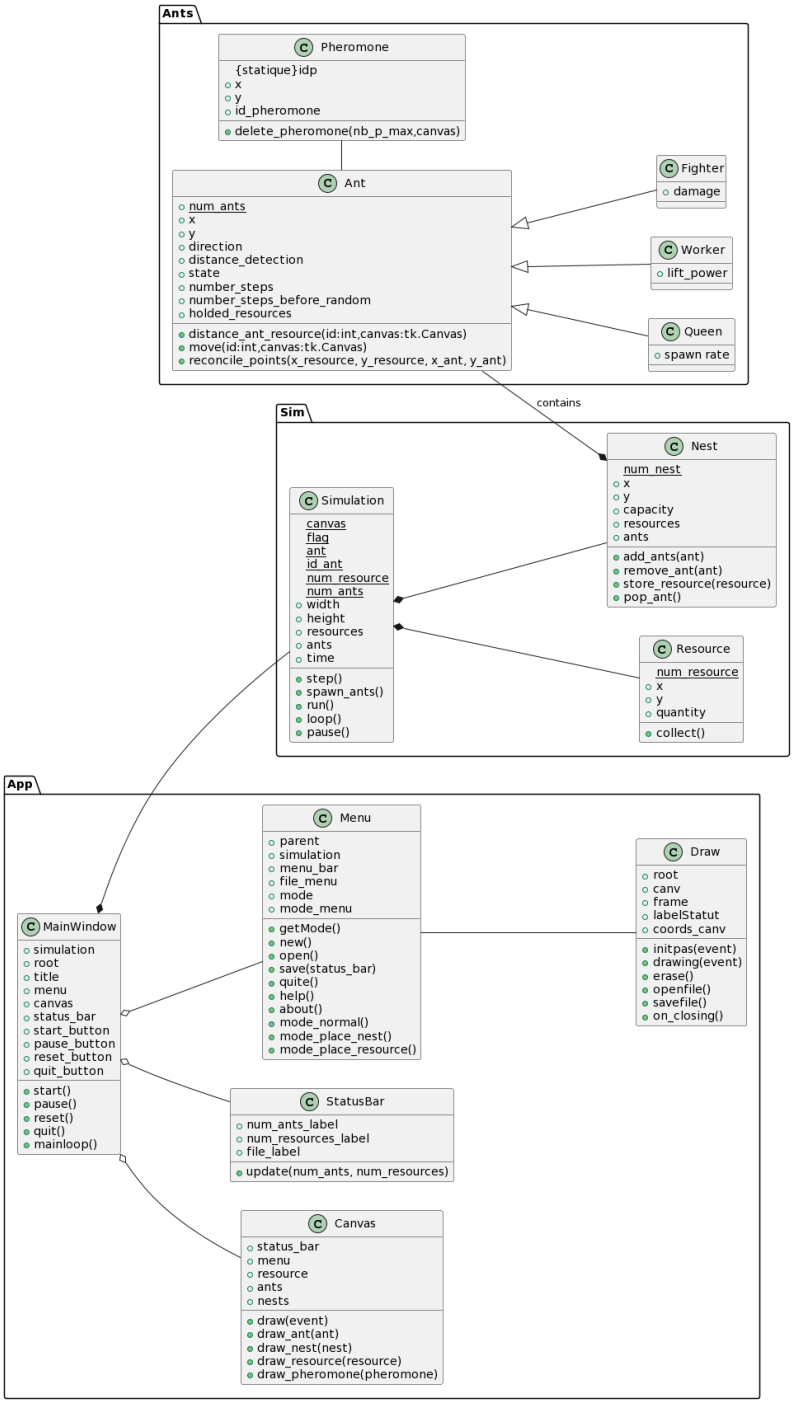
5.1.1 Diagramme de classe avec packages

```
@startuml
left to right direction


package Sim {
class Simulation {
{static} canvas
{static} flag
{static} ant
{static} id_ant
{static} num_resource
{static} num_ants
+width
+height
+resources
+ants
+time

+step()
+spawn_ants()
+run()
+loop()
+pause()

}

class Nest{
{static} num_nest
+ x
+ y
+capacity
+resources
+ants
+add_ants(ant)
+remove_ant(ant)
+store_resource(resource)
+pop_ant()
}
class Resource {
{static} num_resource
+ x
+ y
+ quantity
+collect()
}
Simulation *-- Resource
Simulation *-- Nest
}


package Ants{
class Ant {
+ {static}num_ants
+ x
```

```
+ y
+ direction
+ distance_detection
+ state
+ number_steps
+ number_steps_before_random
+ holded_resources
+ distance_ant_resource(id:int,canvas:tk.Canvas)
+ move(id:int,canvas:tk.Canvas)
+ reconcile_points(x_resource, y_resource, x_ant, y_ant)
}

class Pheromone{
{statique}idp
+ x
+ y
+id_pheromone
+delete_pheromone(nb_p_max,canvas)
}

class Queen {
+spawn rate
}

class Worker {
+lift_power
}

class Fighter {
+damage
}
Ant - Pheromone
Ant <|-- Queen
Ant <|-- Fighter
Ant <|-- Worker
}

package App{
class MainWindow {
+simulation
+root
+title
+menu
+canvas
+status_bar
+start_button
+pause_button
+reset_button
+quit_button

+start()
+pause()
+reset()
+quit()
+mainloop()
}
```

```
111
112 class Canvas {
113 +status_bar
114 +menu
115 +resource
116 +ants
117 +nests
118 +draw(event)
119 +draw_ant(ant)
120 +draw_nest(nest)
121 +draw_resource(resource)
122 +draw_pheromone(pheromone)
123 }
124
125 class StatusBar {
126 +num_ants_label
127 +num_resources_label
128 +file_label
129 +update(num_ants, num_resources)
130 }
131
132 class Menu {
133 +parent
134 +simulation
135 +menu_bar
136 +file_menu
137 +mode
138 +mode_menu
139 +getMode()
140 +new()
141 +open()
142 +save(status_bar)
143 +quite()
144 +help()
145 +about()
146 +mode_normal()
147 +mode_place_nest()
148 +mode_place_resource()
149 }
150
151 class Draw{
152 +root
153 +canv
154 +frame
155 +labelStatut
156 +coords_canv
157 +initpas(event)
158 +drawing(event)
159 +erase()
160 +openfile()
161 +savefile()
162 +on_closing()
163 }
164
165 MainWindow o-- Canvas
166 MainWindow o-- StatusBar
```

```
167  MainWindow o-- Menu
168  Menu -- Draw
169  }
170
171  MainWindow *-- Simulation
172  Ant --* Nest : contains
173
174  @enduml
```

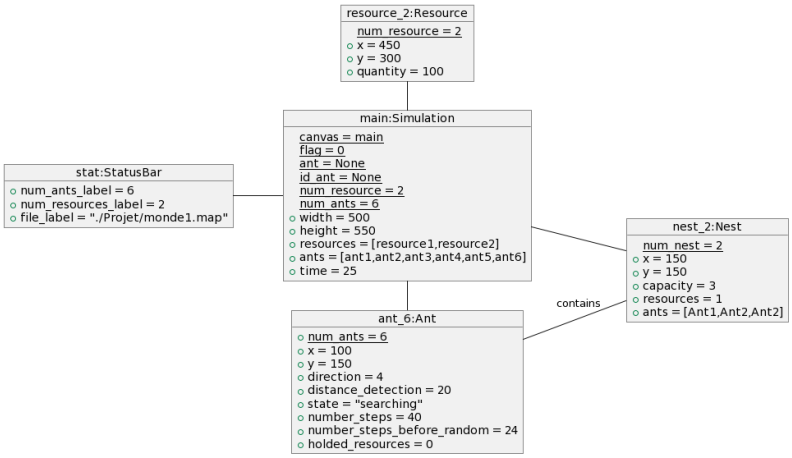

Ants
Pheromone
{statique}idp
x
y
id_pheromone
delete_pheromone(nb_p_max,canvas)
Ant
num_ants
x
y
direction
distance_detection
state
number_steps
number_steps_before_random
holded_resources
distance_ant_resource(id:int,canvas:tk.Canvas)
move(id:int,canvas:tk.Canvas)
reconcile_points(x_resource, y_resource, x_ant, y_ant)
Fighter
damage
Worker
lift_power
Queen
spawn rate
contains
Sim
Nest
num_nest
x
y
capacity
resources
ants
add_ants(ant)
remove_ant(ant)
store_resource(resource)
pop_ant()
Simulation
canvas
flag
ant
id_ant
num_resource
num_ants
width
height
resources
ants
time
step()
spawn_ants()
run()
loop()
pause()
Resource
num_resource
x
y
quantity
collect()
App
Menu
parent
simulation
menu_bar
file_menu
mode
mode_menu
getMode()
new()
open()
save(status_bar)
quite()
help()
about()
mode_normal()
mode_place_nest()
mode_place_resource()
Draw
root
canv
frame
labelStatut
coords_canv
initpas(event)
drawing(event)
erase()
openfile()
savefile()
on_closing()
MainWindow
simulation
root
title
menu
canvas
status_bar
start_button
pause_button
reset_button
quit_button
start()
pause()
reset()
quit()
mainloop()
StatusBar
num_ants_label
num_resources_label
file_label
update(num_ants, num_resources)
Canvas
status_bar
menu
resource
ants
nests
draw(event)
draw_ant(ant)
draw_nest(nest)
draw_resource(resource)
draw_pheromone(pheromone)

5.2 Diagramme d'objets

Voici un exemple de diagramme d'objets mettant en jeu quelques unes des différentes classes de l'application.

```
@startuml
left to right direction
object "ant_6:Ant" as Ant {
+ {static}num_ants = 6
+ x = 100
+ y = 150
+ direction = 4
+ distance_detection = 20
+ state = "searching"
+ number_steps = 40
+ number_steps_before_random = 24
+ holded_resources = 0
}

object "main:Simulation" as Simulation {
{static} canvas = main
{static} flag = 0
{static} ant = None
{static} id_ant = None
{static} num_resource = 2
{static} num_ants = 6
+width = 500
+height = 550
+resources = [resource1,resource2]
+ants = [ant1,ant2,ant3,ant4,ant5,ant6]
+time = 25
}

object "resource_2:Resource" as Resource {
{static} num_resource = 2
+ x = 450
+ y = 300
+ quantity = 100
}

object "nest_2:Nest" as Nest {
{static} num_nest = 2
+ x = 150
+ y = 150
+capacity = 3
+resources = 1
+ants = [Ant1,Ant2,Ant2]
}


object "stat:StatusBar" as StatusBar {
+num_ants_label = 6
+num_resources_label = 2
+file_label = "./Projet/monde1.map"
}

```

```
52
53  Ant -- Nest : contains
54
55  Resource - Simulation
56  Simulation -- Nest
57  Simulation - Ant
58  StatusBar -- Simulation
59
60
61  @enduml
```



6 Manuel d'installation

6.1 Installation de l'application

L'application ne requiert pas d'installation spécifique, autre que Python version 3 minimum. Vous trouverez ensuite le projet sur moodle sous le nom "I62_Sim_Fourmis.zip".

7 Manuel utilisateur

7.1 Exécuter l'application

Pour exécuter l'application il vous faudra donc **Python3**. Pour lancer le programme, utilisez la commande bash : "python3 main.py"

7.2 Application

Lors du lancement de l'application, celle-ci ouvrira une nouvelle fenêtre nommée "Simulateur de fourmis" comme sur la figure n°1.

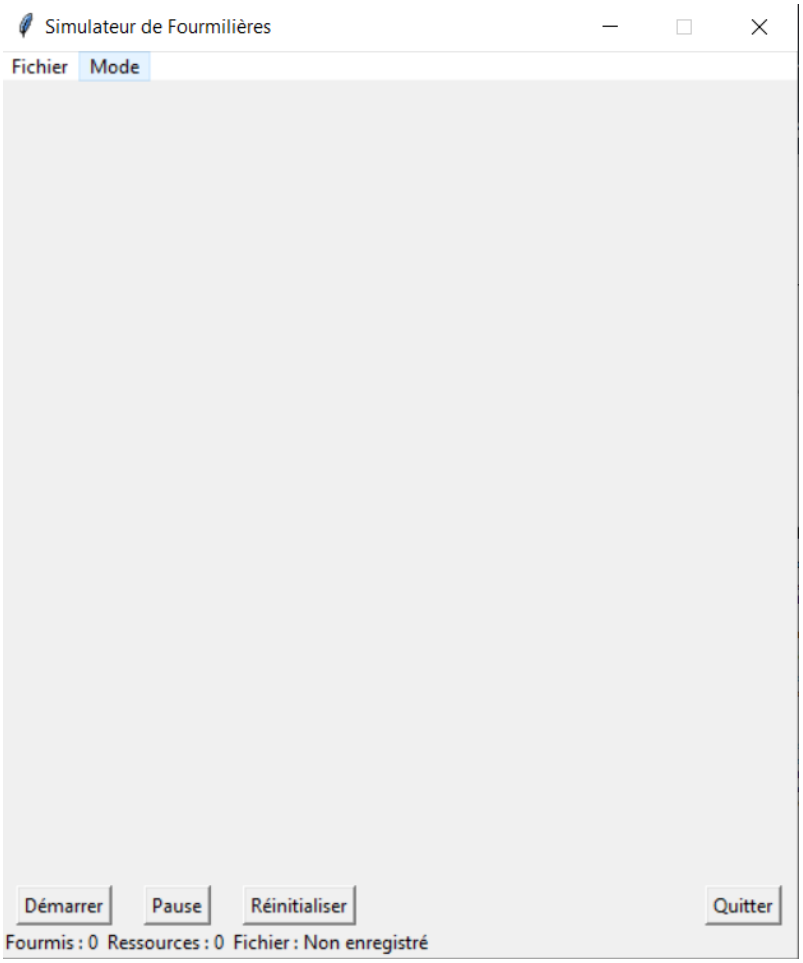


FIGURE 1 – Simulation

7.3 Gestion du monde

7.3.1 Créer un nouveau monde

Pour créer un nouveau monde, il suffit de se rendre dans le menu de l'applciation, section "Fichier", puis de cliquer sur "Nouveau monde" (voir figure n°3). Une nouvelle fenêtre apparaîtra donc comme sur la figure n°2

7.3.2 Importer un monde

Au niveau du coin supérieur gauche de la fenêtre, il y a un bouton "Fichier" qui peut-être utilisé afin d'importer un nouveau monde (figure n°3). Pour charger un monde, il faut importer un fichier se finissant par «.map», créé préalablement via l'outil de création de monde (voir section 7.3.1).

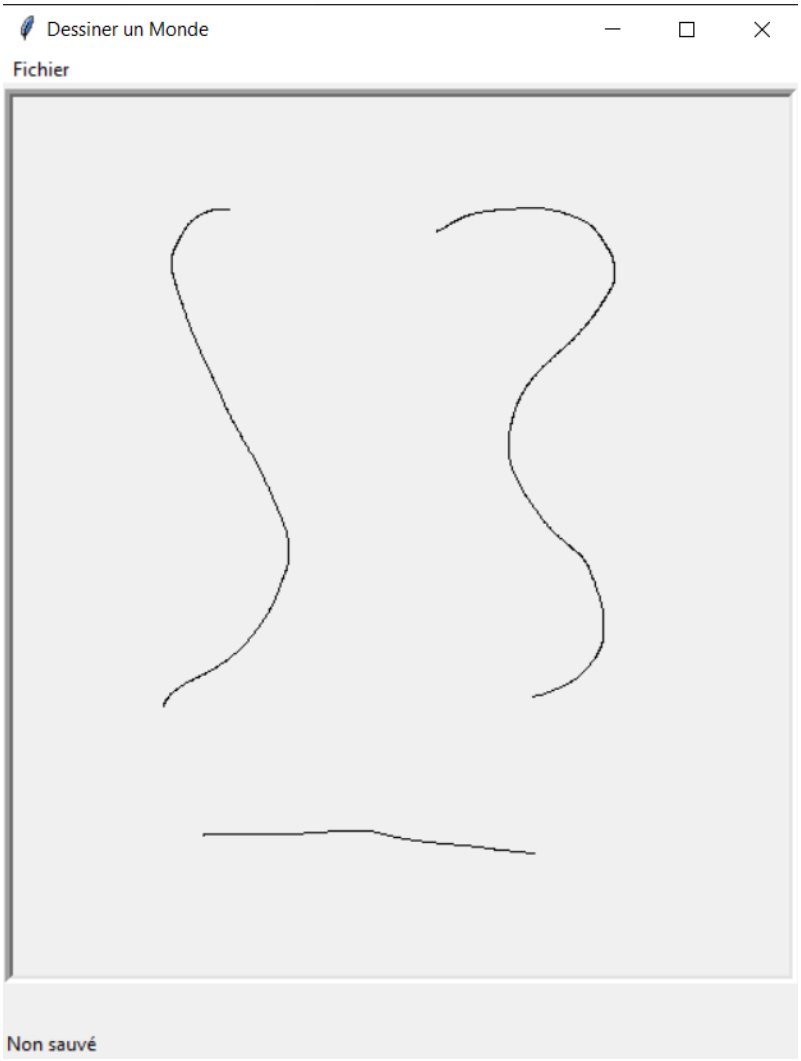


FIGURE 2 – Monde

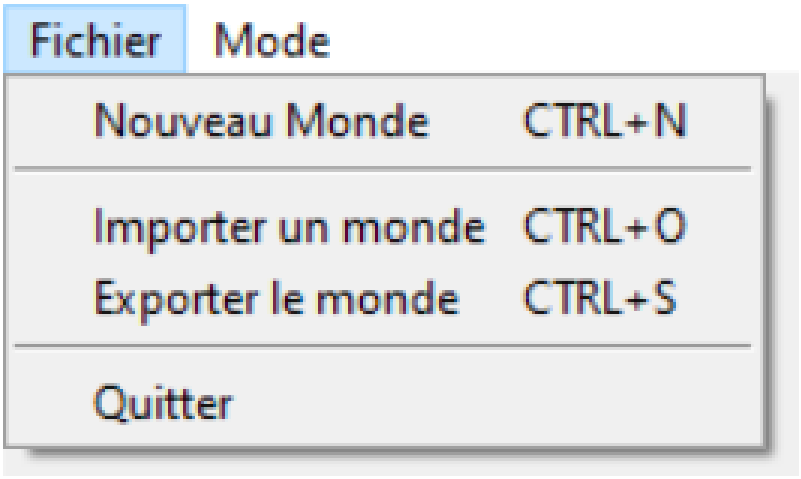


FIGURE 3 – Menu Monde

7.4 Ajouter des nids et des ressources

Sur la droite du bouton "Fichier", se trouve un second bouton "Mode" (figure n°4) qui permet de placer des nids et des ressources dans le monde actuel, il vous faudra placer au moins un nid et une ressource pour démarrer la simulation via le bouton "Démarrer" qui se situe en bas à gauche de la fenêtre.

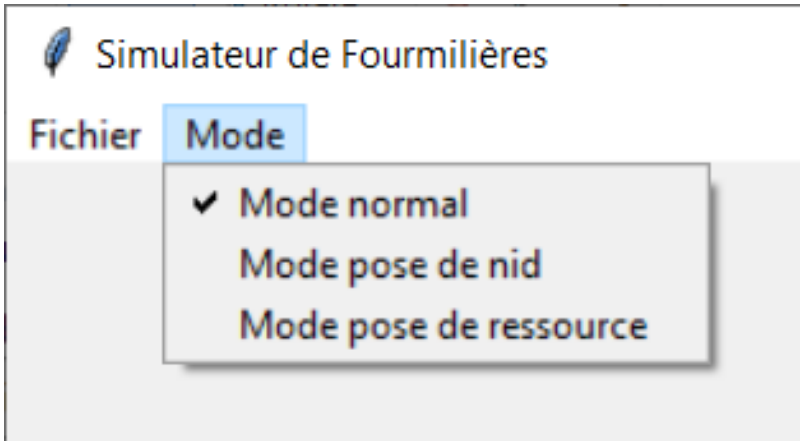


FIGURE 4 – Mode

7.5 Quitter l'application

Pour arrêter l'application, un bouton "Quitter" se trouve en bas à droite de la fenêtre ou alors vous pouvez également utiliser la croix rouge en haut à droite. Une fenêtre de vérification s'ouvrira donc (voir figure n°5) si vous voulez annuler la fermeture il faudra donc appuyer sur le bouton «Cancel», au contraire si vous souhaitez continuer la fermeture du programme, il faudra alors appuyer sur le bouton « Ok ».

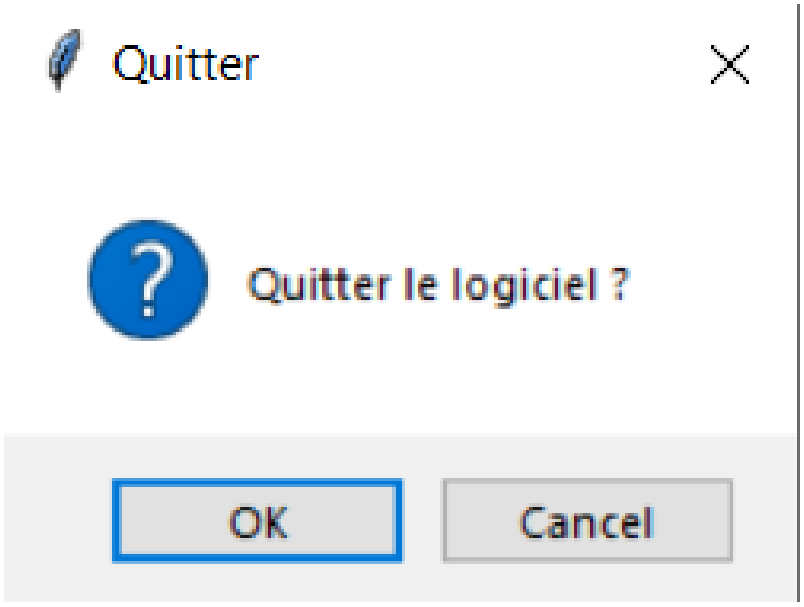


FIGURE 5 – Fenêtre de vérification

7.6 Statistiques

Vous pouvez voir les statistique de la partie en bas de la fenêtre (figure 6) qui montre le nombre de fourmis, les ressources, le nom du fichier. On voulait ajouter les ressources de chaque nid et les ressources restantes mais on a pas eu le temps de l'implanter.

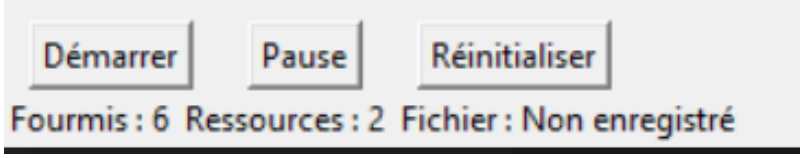


FIGURE 6 – Statistiques

Post-scriptum

En ce qui concerne la gestion des collisions des murs et obstacles, nous n'avons pas eu le temps de l'implanter, l'implantation de "l'intelligence artificielle" des fourmis nous ayant pris beaucoup de temps à développer.

De plus, l'internationalisation du programme a été développée (voir figure n°7) mais celle-ci provoquait différents bugs non résolus sur la version finale du programme, nous avons alors pris la décision de ne pas l'intégrer dans la version finale.

```
1       import importlib
2       import gettext
3       importlib.reload(gettext)
4       # Configuration de gettext pour utiliser les fichiers de
        messages appropriés
5       lang = 'fr_FR'
6       locale_path = './locales'
7       _ = gettext.gettext
8
9       lang = gettext.translation('messages', localedir=locale_path,
        languages=['fr_FR'])
10      lang.install()
11
```

FIGURE 7 – Code Python de l'internationalisation du programme

On utilisera donc dans la suite du code de notre programme les noms des "messages" de getText afin d'avoir la traduction dans la langue spécifiée dans "languages" (voir figure n°8). Les "messages" sous la forme *_("message")* sont donc automatiquement traduits ici en Français.

```
1       self.start_button = tk.Button(self.root, text=_("Start"),
        command=self.start)
2       self.start_button.pack(side=tk.LEFT, padx=10)
3       self.pause_button = tk.Button(self.root, text=_("Pause"),
        command=self.pause)
4       self.pause_button.pack(side=tk.LEFT, padx=10)
5       self.reset_button = tk.Button(self.root, text=_("Reset"),
        command=self.reset)
6       self.reset_button.pack(side=tk.LEFT, padx=10)
7       self.quit_button = tk.Button(self.root, text=_("Quit"), command=
        self.quit)
8       self.quit_button.pack(side=tk.RIGHT, padx=10)
9
```

FIGURE 8 – Exemples d'utilisation de getText dans notre programme