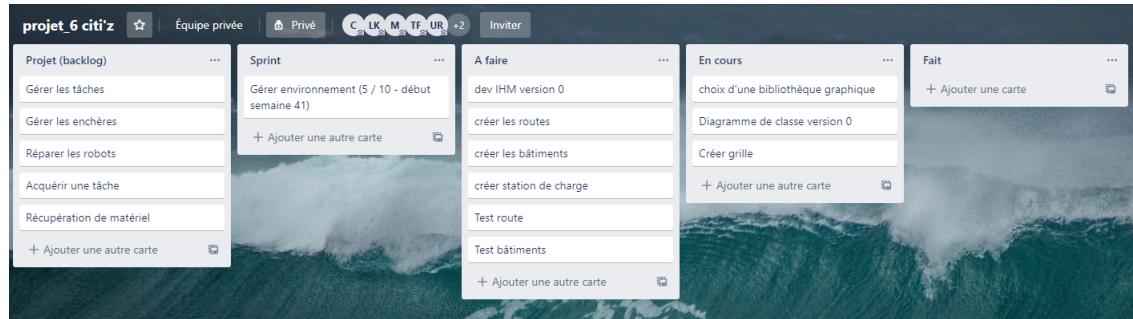


# [M3301]Livrable 1: smart city et robots

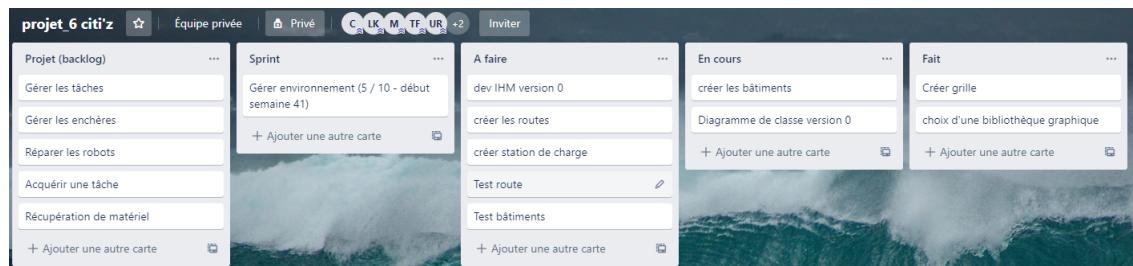
Groupe 6 : CAMUS Benjamin / FONTENIT Théo  
LELOUCHE Matthias/ KUCHEIDA Lucas  
AMATHIEU Fléric/ GUYOT Augustin

## 1 Sprint 1

### 1.1 Premier jet de tâches (22/09)



### 1.2 Bibliotheque graphique choisie (29/09 - 9h45)



Ayant alors choisi Tkinter comme biliothèque graphique nous avons alors débuté le code python

### 1.2.1 Aperçu de la classe Building

```
class Building:  
    def __init__(self,x,y,size,bType):  
        self.x = x  
        self.y = y  
        self.size = size  
        self.bType = bType
```

La classe Building est définie par la position x, y sur la carte et par une taille et un type de construction. Elle permet de définir l'emplacement, la taille des construction ainsi que leur type (décor / bâtiments de recharge).

### 1.2.2 Aperçu de la classe Case

```
class Case :  
    def __init__(self,x,y,status="route"):  
        self.x = x  
        self.y = y  
        self.edges = {'n' : True, 'e' : True, 's' : True, 'o' : True}  
        self.status = status  
  
    def setEdges(self,orient,presence):  
        self.edges[orient] = presence  
  
    def __str__(self):  
        return "Case : " + str(self.x) + " horizontal, " + str(self.y) + " vertical et son status est "+ str(self.status)
```

La classe Case est définie par une position x, y et par un status qui permet de savoir si la construction est une route ou un bâtiment

### 1.2.3 Aperçu de la classe City

```

class City:

    def __init__(self, sizeX, sizeY):
        self.sizeX = sizeX
        self.sizeY = sizeY
        self.matrix = []
        for i in range(sizeX):
            matrixLine = []
            for j in range(sizeY):
                matrixLine += [Case(i,j,False)]
            self.matrix += [matrixLine]
        self.buildings = []

    def getCell(self,x,y):
        return self.matrix[x][y]

    def createBuilding(self,bat):
        #gestion des status des cases
        for i in range(bat.x,bat.x+bat.size):
            for j in range(bat.y,bat.y+bat.size):
                self.getCell(i,j).status = "building"
        #gestion des murs des cases
        for i in range(bat.x,bat.x+bat.size):
            for j in range(bat.y,bat.y+bat.size):
                if self.getCell(i-1,j).status == "building":
                    self.getCell(i-1,j).edges['e'] = False
                if self.getCell(i+1,j).status == "building":
                    self.getCell(i+1,j).edges['o'] = False
                if self.getCell(i,j-1).status == "building":
                    self.getCell(i,j-1).edges['s'] = False
                if self.getCell(i,j+1).status == "building":
                    self.getCell(i,j+1).edges['n'] = False
        #ajout du bâtiment à la liste des bâtiments de la ville
        self.buildings += [bat]

    def RandomGenerate(self):
        pointerX = 1
        pointerY = 1
        verifyY = True
        while verifyY == True:
            if self.getCell(pointerX,pointerY).status != "route":
                pointerY += 2
            randSize = random.randint(1,3)
            if randSize+pointerY <= self.sizeY :
                bat = Building(pointerX,pointerY,random.randint(1,3),"decor")
                self.createBuilding(bat)
            else :
                pointerY = 1
                verifyY = False
        #imbriquer un deuxième while pour faire la génération horizontale

```

La classe City est définie par une taille x et y elle est constituée de plusieurs cases côtes à côtes.

getCell permet d'obtenir la cellule.

createBuilding permet de modifier les cases de la ville pour les faire passer de l'état route à l'état bâtiment.

randomGenerate sert à générer de manière aléatoire des bâtiments dans la ville, cependant elle n'est pas terminée.

## 1.3 Crédit et test des bâtiments/routes (29/09 - 11h00)



### 1.3.1 Aperçu du fichier du test

```
from Classes.City import City
from Classes.Case import Case
from Classes.Building import Building
from tkinter import *

fenetre = Tk()
testCity = City(10,10)

canvas = Canvas(fenetre, width=testCity.sizeX*50+50, height=testCity.sizeY*50+50, background='white')
city = canvas.create_rectangle(50, 50, testCity.sizeX*50, testCity.sizeY*50, fill='grey')

#print(testCity.matrix)
testCity.getCell(0,3).setEdges('n',False)
build = Building(2,4,3,"decon")
testCity.createBuilding(build)

#création graphique des bâtiments
buildings = []
for i in range(len(testCity.buildings)):
    buildings += [canvas.create_rectangle(testCity.buildings[i].x*50,
                                         testCity.buildings[i].y*50, testCity.buildings[i].size*50,
                                         testCity.buildings[i].size*50,
                                         fill='blue')]

#print(testCity.getCell(0,3).edges)
#print(testCity.getCell(2,4))
#print(testCity.getCell(2,4).edges)
#print(testCity.getCell(2,5))
#print(testCity.getCell(2,5).edges)
#print(testCity.getCell(3,4))
#print(testCity.getCell(3,4).edges)
#print(testCity.getCell(3,5))
#print(testCity.getCell(3,5).edges)

canvas.pack()
fenetre.mainloop()
```

Tout d'abord on défini la fenêtre tkinter.

Ensuite on défini une ville de taille 10x10.

On défini un canvas blanc dans lequel la ville va se dessiner.

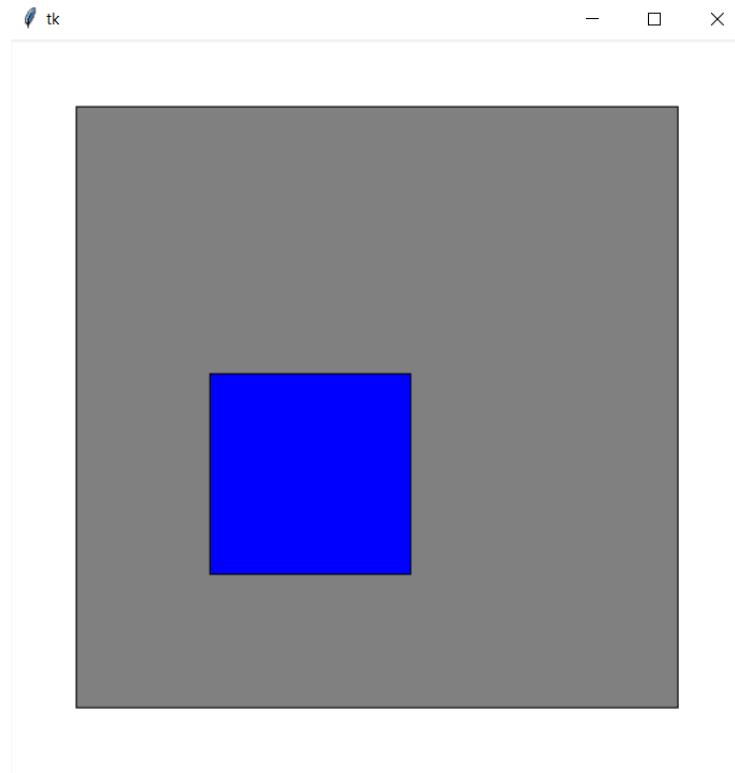
On dessine la ville (ici un rectangle gris) sur le canvas.

Ensuite on commence à intégrer des construction (ici nous avons essayé d'ajouter un building).

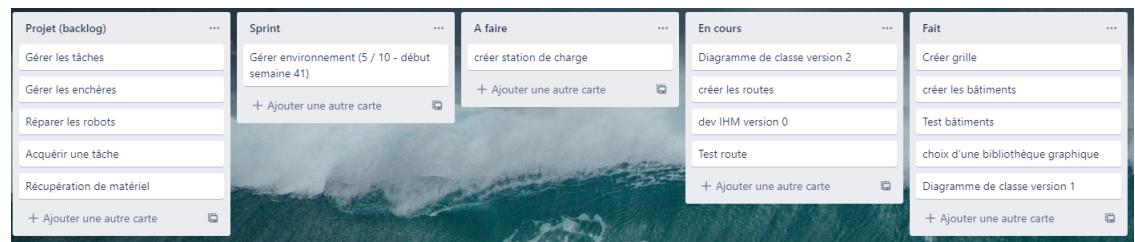
Finalement on "compile" les éléments.

Puis on affiche ceux ci avec une boucle pour ne faire qu'une image fixe.

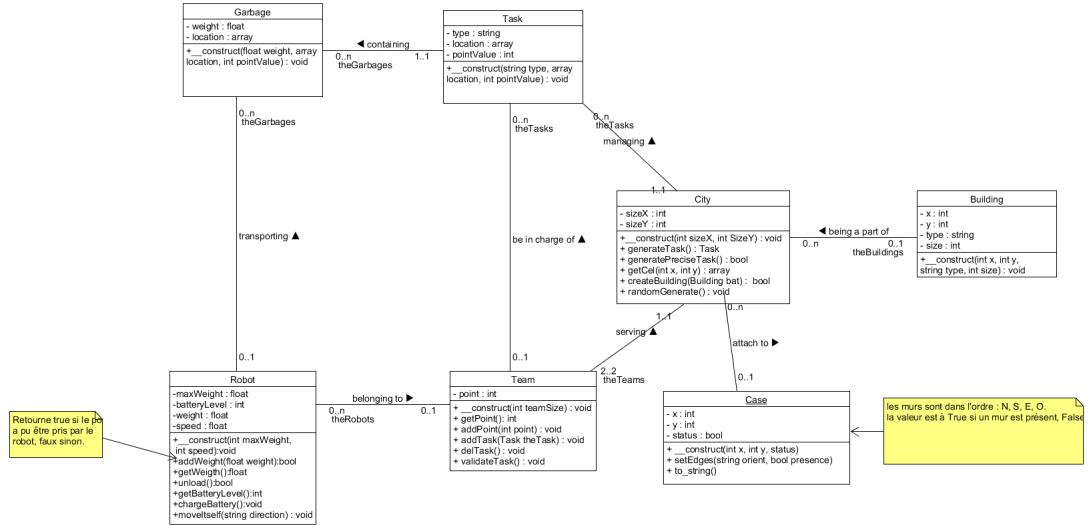
### 1.3.2 Aperçu du test



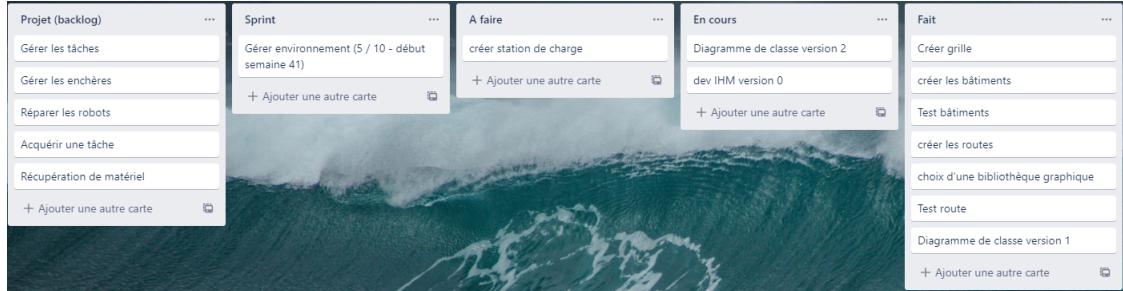
### 1.4 Mise à jour du diagramme de classe (03/10)



### 1.4.1 Diagramme de classe

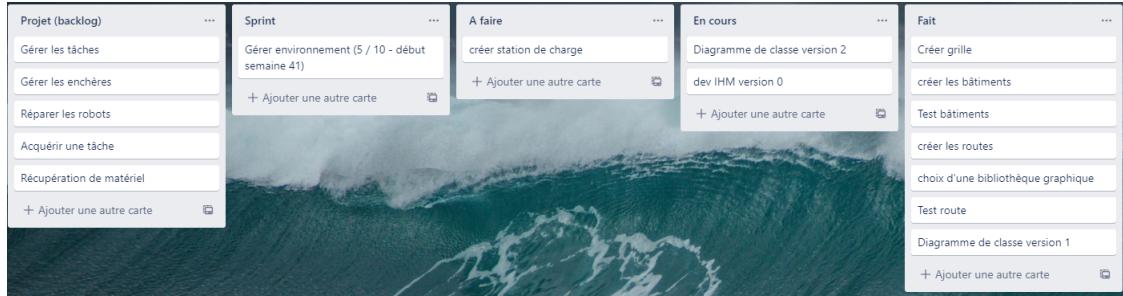


## 1.5 Nouvelles tâches en cours en fin du sprint 1 (03/10)



## 2 Sprint 2: Gestion des tâches et déplacement aléatoire du robot

### 2.1 Dernier jet de tâches sprint 1(03/10)



### 2.2 Création des classes: en cours (10/10)

Actuellement nous sommes en train de créer les classes du robot et des tâches (non terminées).

#### 2.2.1 Aperçu de la classe Robot

```
class Robot:
    def __init__(self,x,y,maxWeight=0,batteryLevel=0,weight=0,speed=0):
        self.maxWeight = maxWeight
        self.batteryLevel = batteryLevel
        self.weight = weight
        self.speed = speed
        self.x = x
        self.y = y

    def addWeight(self,weight):
        self.weight = weight

    def unload(self):
        retour = False
        if self.weight == 0:
            retour = True
        return retour
    ...

    def chargeBattery(self):
        if self.batteryLevel < 20:
            ...

    def moveItself(self,direction):
        dx = 0
        dy = 0
        if direction == "N":
            self.y -= 1
            dy = -20
        elif direction == "S":
            self.y += 1
            dy = 20
        elif direction == "W":
            self.x -= 1
            dx = -20
        elif direction == "E":
            self.x += 1
            dx = 20
        return [dx,dy]
```

La classe Robot est définie par la position x, y sur la carte, un poids de transport maximum, un niveau de batterie, un poids de transport actuel, et une vitesse. Elle permet de définir son emplacement, son poids max transportable, son niveau de batterie, sa charge actuelle et sa vitesse

### 2.2.2 Aperçu de la classe Task

```
class Task:  
    def __init__(self,x,y,pointValue,typeT="base"):  
        self.x = x  
        self.y = y  
        self.pointValue = pointValue  
        self.typeT = typeT
```

La classe Task est définie par une position x, y, le nombre de point que la tâche rapporte, un type de tâche.

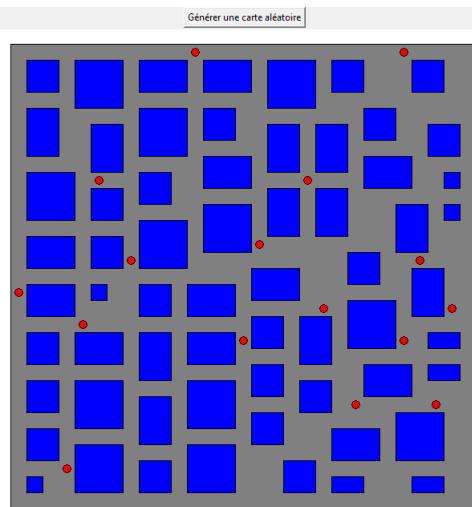
## 2.3 Implémentation des tâches dans la ville (10/10)

Nous avons implémenté les tâches dans la ville par une méthode d'ajout de tâche et une méthode de génération aléatoire des tâches.

### 2.3.1 Aperçu des méthodes

```
def addTask(self,task):  
    if self.getCell(task.x,task.y).status == "route":  
        self.tasks.append(task)  
  
def addRandomTask(self):  
    k = 0  
    for i in range(self.sizeX-2):  
        for j in range(self.sizeY-2):  
            if k > 65:  
                k = 0  
                taille = 20  
                task = Task(i,j,taille,taille)  
                self.addTask(task)  
            k += random.randint(1, 6)
```

### 2.3.2 Aperçu du test



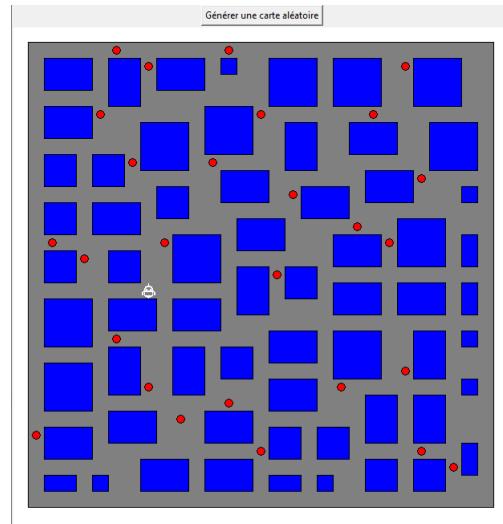
## 2.4 Ajout d'un robot et de déplacement aléatoire de celui ci (12/10)

```
robotDep = canvas.create_image(30,30,image=img)
"""
def deplacement():
    global dx, dy
    robot = Robot(0,0)
    depl = robot.moveItSelf("E")
    dx = depl[0]
    dy = depl[1]
    print(dx,dy)
    print("pos robot : ", robot.x,robot.y)
    canvas.move(robotDep,dx,dy)
    canvas.pack()
    fenetre.after(100,deplacement)
    print("test")
canvas.pack()
deplacement()
"""

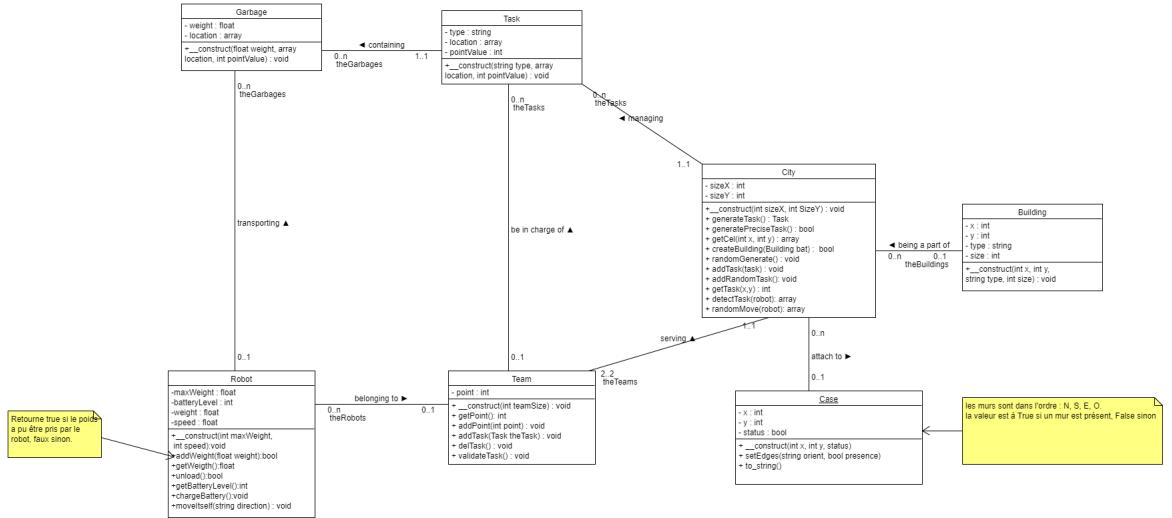
robot = Robot(0,0)
def deplacementAlea():
    global dx, dy
    depl = testcity.randomMove(robot)
    dx = depl[0]
    dy = depl[1]
    #print(dx,dy)
    #print("pos robot : ", robot.x,robot.y)
    canvas.move(robotDep,dx,dy)
    canvas.pack()
    fenetre.after(10,deplacementAlea)
    #print("test")
canvas.pack()
deplacementAlea()

dx = 0
dy = 20
```

### 2.4.1 Aperçu du test



## 2.5 Mise à jour du diagramme de classe (15/10)



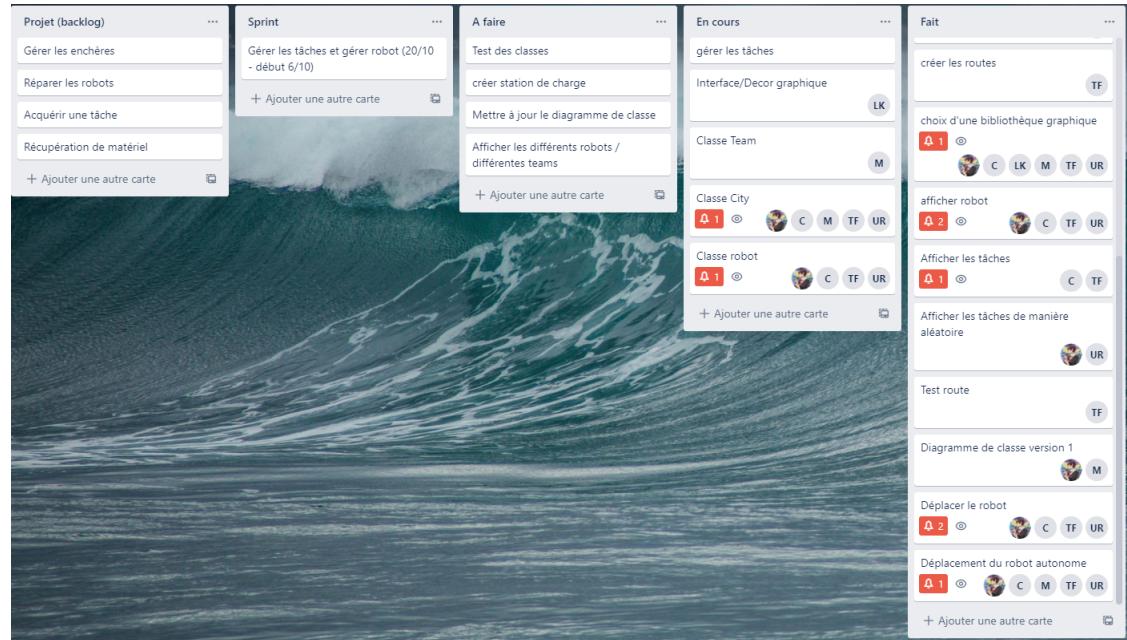
## 2.6 Aperçu des tâches à la fin du sprint 2 (19/10)

The screenshot shows a project management interface with the following sections:

- Projet (backlog)**:
  - Gérer les enchères
  - Réparer les robots
  - Acquérir une tâche
  - Récupération de matériel
  - + Ajouter une autre carte
- Sprint**:
  - Gérer les tâches et gérer robot (20/10) - début 6/10
  - + Ajouter une autre carte
- A faire**:
  - Test des classes
  - créer station de charge
  - Mettre à jour le diagramme de classe
  - Afficher les différents robots / différentes teams
  - + Ajouter une autre carte
- En cours**:
  - gérer les tâches
  - Interface/Decor graphique LK
  - Classe Team M
  - Classe City LK
  - Classe robot M
  - + Ajouter une autre carte
- Fait**:
  - créer les routes TF
  - choix d'une bibliothèque graphique UR
  - afficher robot UR
  - Afficher les tâches C TF
  - Afficher les tâches de manière aléatoire UR
  - Test route TF
  - Diagramme de classe version 1 M
  - Déplacer le robot UR
  - Déplacement du robot autonome UR
  - + Ajouter une autre carte

### 3 Sprint 3: Bouton de génération de robot + algorithme A\* (IA)

#### 3.1 Dernier jet de tâches sprint 2(19/10)



#### 3.2 Implémentation du bouton de génération de robot (02/11)

Nous avons donc implémenté un nouveau bouton pour pouvoir générer autant de robot que l'on souhaite.

##### 3.2.1 Aperçu de la génération de Robot

```
def addRobot():
    testCity.robots += [Robot(0,0,"a*")]
    #testCity.imgRobots += [canvas.create_image(30,30,image=img)]
    testCity.imgRobots += [canvas.create_oval(25,25,35,35)]
    deplacement()
ba = Button(fenetre, text="YA UN ROBOT", command=addRobot)
ba.pack()
```

On crée un robot à la position 0,0 qui utilise l'algorithme A\*. On ajoute au robot l'apparence d'oval dans le canvas. On lance la méthode de déplacement. Puis on définit le bouton qui prends en paramètre la méthode d'ajout de robot.

### 3.3 Implémentation de l'algorithme A\* - En cours - (01/11)

Nous avons implémenté et optimisé l'algorithme A\* pour notre code afin que le robot puisse chercher un chemin optimisé pour d'atteindre une tâche.

#### 3.3.1 Aperçu des fonctions

```

def deplacement():
    #print(testCity.tasks)
    for i in range(len(testCity.robots)):
        #print('coucou')
        if testCity.robots[i].moveType == "alea":
            dep1 = testCity.randomMove(testCity.robots[i])
        elif testCity.robots[i].moveType == "a*":
            postask = testCity.findTaskFromXY()
            chemin = testCity.aToileMove(testCity.robots[i],postask)
            print("chemin")
            print("-----^chemin-----")
            dep1 = testCity.robots[i].moveItSelfFromCoord([chemin[0][0],chemin[0][1]] + [chemin[0][2],chemin[0][3]])
            print(depl)
            print("-----^dep1-----")
            dx = dep[0]
            dy = dep[1]
            print(dx,dy)
            print("-----^dx dy-----")
            if dep[2] != -1:
                print("avant delete : ", ourTasks)
                canvas.delete(ourTasks[dep[2],dep[3]])
                testCity.removeTask([dep[2],dep[3]])
                print("après delete : ", ourTasks)
                print("//////////////SWINGZ/////////////")
            canvas.move(testCity.imgRobots[i],dx,dy)
    canvas.pack()
    fenetre.after(100,deplacement)

def aToileMove(self, robot, postask): #ajout d'une tâche dans la paume pour ne pas avoir à la fin dans la fonction et ne renvoyer que le dernier déplacement ou ne récupérer que le dernier postask = self.findTaskFromXY()
#print("//////////////// la fonction ///////////////////////////////")
#print(postask)
#print("robot x / y : ", robot.x, " / ", robot.y)
open_nodes = []
closed_nodes = []
open_nodes += [self.determineNodesInfos([robot.x,robot.y],[robot.x,robot.y],postask)+[-1,-1]]
compt = 0
#verif = False
#PARCOURS DE LA GRILLE AVEC L'ALGO
while compt < 500:    #AAAAAAA.....AAAAAAA.....AAAAAAA.....AAAAAAA
#while verif == False :
    compt+=1
    #print(compt)
    #print("open : ", open_nodes)
    #print("closed : ", closed_nodes)
    minIF = open_nodes[0][1]
    minIH = open_nodes[0][2]
    current = open_nodes[0]
    index = 0
    for i in range(len(open_nodes)):
        if open_nodes[i][1] < minIF: #à modif pour les égalités
            minIF = open_nodes[i][1]
            minIH = open_nodes[i][2]
            current = open_nodes[i]
            index = i
    open_nodes.pop(index)
    closed_nodes += [current]
    #print("Le courant : ",current)

    #CREATION DE LA LISTE RETOUR
    if current[0] == postask:
        verif = True
        robot_move_node = current
        retour = []
        while verif == False :
            #print("Le robot move node : ", robot_move_node)
            if robot_move_node[4] == [-1,-1]:
                verif = True
            for i in range(len(closed_nodes)):
                if closed_nodes[i][0] == robot_move_node[4]:
                    robot_move_node = closed_nodes[i]
        
```

```

        if closed_nodes[1][0] == robot_move_node[4]:
            robot_move_node = closed_nodes[1]
        if robot_move_node[0][0] != robot.x or robot_move_node[0][1] != robot.y : #différent de la pos du robot -> on ajoute
            #print("le x du robot move node ",robot_move_node[0][0])
            retour = [robot_move_node[0]+self.detectTask(robot)] + retour
        retour += [[postTask[0],postTask[1],postTask[0],postTask[1]]]
        #print("-----ON A FINI-----")
        #print("/////////// la fin de fonction ///////////")
    return retour

#AJOUT DES VOISINS DANS L'OPEN_NODES
#print("les voisins : ", self.getNeighbours((current[0][0],current[0][1]),(robot.x,robot.y),postTask))
for i in self.getNeighbours((current[0][0],current[0][1]),(robot.x,robot.y),postTask):
    #print("devrait être ajouté si pas déjà dedans ",i)
    if self.getCel([i[0][0],i[0][1]].status != "building" and self.isNodeInTheList(i,closed_nodes) == False and self.isNodeInTheList(i,open_nodes) == False : #peut être erreur
        #print("dans cours gros ")
        open_nodes+=[[i[0][0],current[0][1]]]
    ....
else:
    #print("bah non mon reuf")
.....

def findTaskFromXY(self): #à refaire car pas très bon
    for i in range(self.sizeX):
        for j in range(self.sizeY):
            if self.getCel(j,i).status == "task":
                return [j,i]
    return [-1,-1]

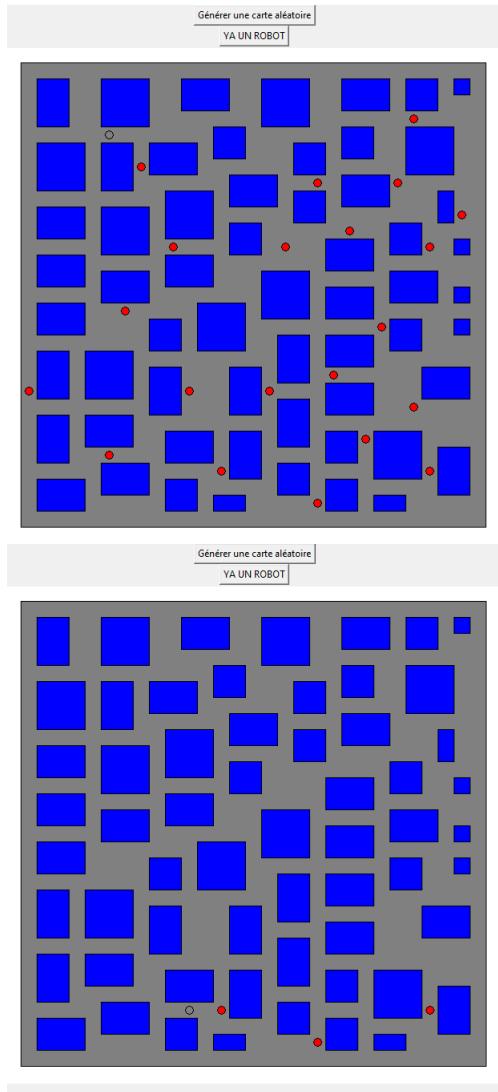
def determineNodesInfos(self,pos,start,goal):
    # [0] = position du noeud, [1] = G cost (distance depuis le début), [2] = H cost (distance depuis l'arrivée), [3] F cost = G + H, [4] = pos parent
    return [[pos[0],pos[1]], abs(pos[0]-start[0])+abs(pos[1]-start[1]), abs(pos[0]-goal[0])+abs(pos[1]-goal[1]), abs(pos[0]-start[0])+abs(pos[1]-start[1])+abs(pos[0]-goal[0])+abs(pos[1]-goal[1])]

def getNeighbours(self,pos,start,goal) : #surement pb sortie de grille
    retour = []
    if pos[0] > 0:
        retour += [self.determineNodesInfos((pos[0]-1,pos[1]),start,goal)]
    if pos[0] < self.sizeX:
        retour += [self.determineNodesInfos((pos[0]+1,pos[1]),start,goal)]
    if pos[1] > 0:
        retour += [self.determineNodesInfos((pos[0],pos[1]-1),start,goal)]
    if pos[1] < self.sizeY:
        retour += [self.determineNodesInfos((pos[0],pos[1]+1),start,goal)]

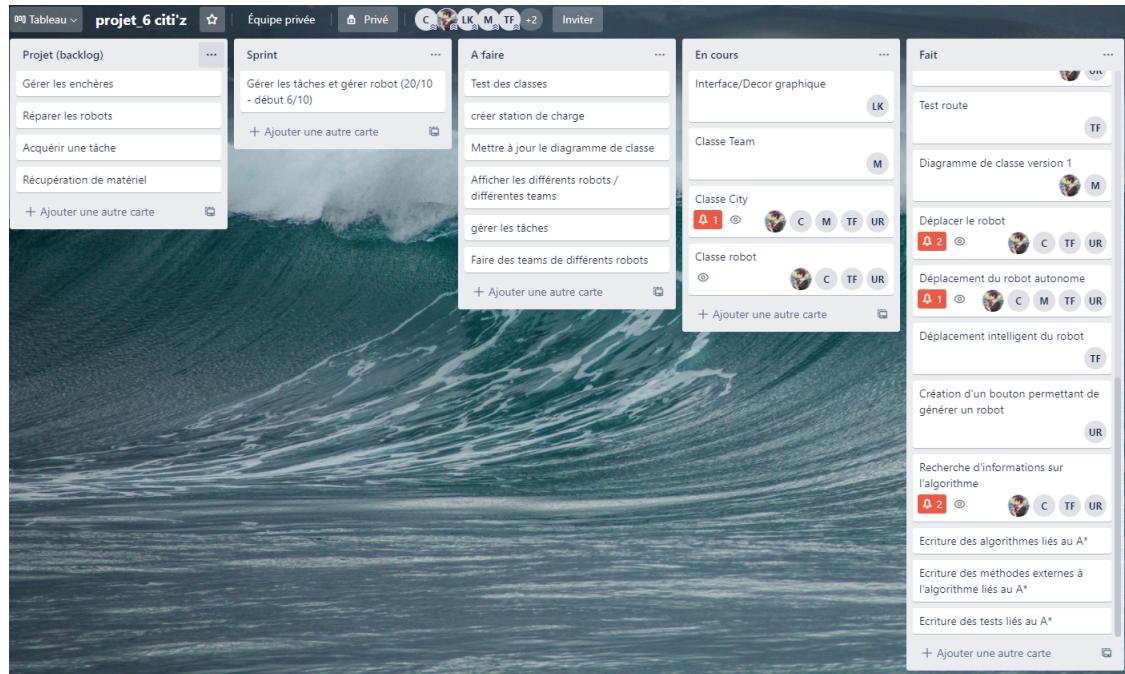
def isNodeInTheList(self,node,lister):
    for i in lister:
        if node[0] == i[0]:
            return True
    return False

```

### 3.3.2 Aperçu du test

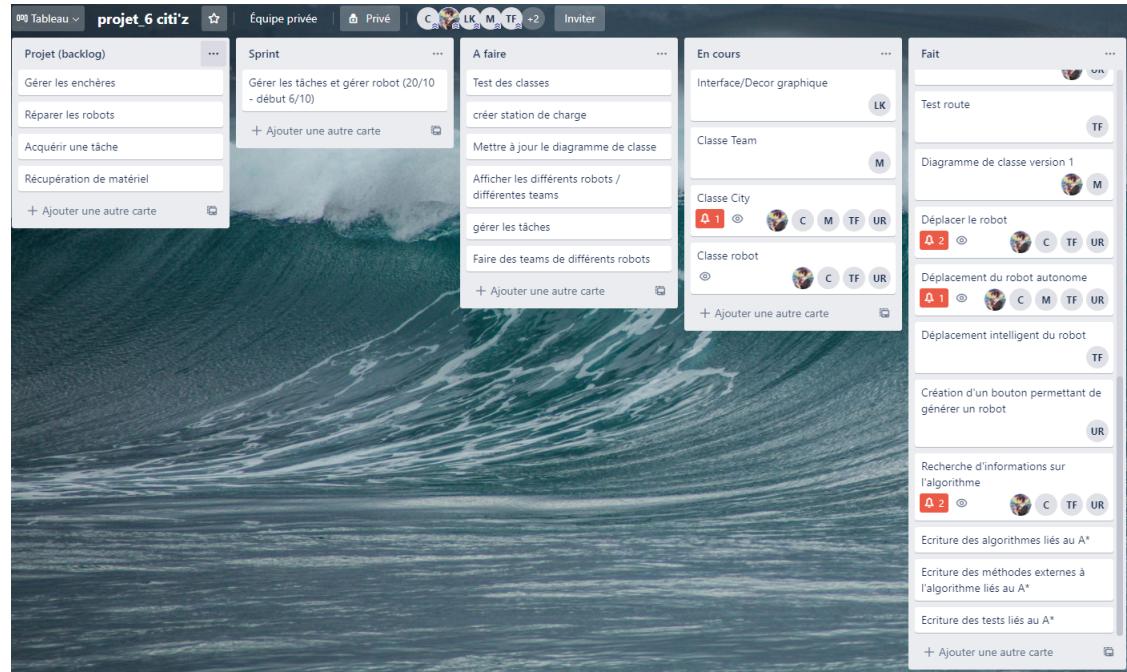


### 3.4 Aperçu des tâches à la fin du sprint 3 (04/11)



## 4 Sprint 4: Implementation fonctionnelle des bâtiments de recharge et de la mécanique relative au robot

### 4.1 Dernier jet de tâches sprint 3(04/11)



### 4.2 HORS SPRINT - Déplacement des méthodes - Uniformisation des classes (06/11)

Nous avons donc déplacé toutes les méthodes dédiées aux robots dans la classe robot, tout en exploitant les informations venant de la classe City. Nous avons donc déplacé randomMove, detectTask, aEtoileMove, determineNodesInfos, getNeighbours, isNodeInTheList dans la classe robot.

### 4.3 HORS SPRINT Ajout d'une documentation complète (06/11)

Suite au bon fonctionnement de nos méthodes, nous avons donc implémenté la documentation très complète de celles ci. Cela permettra à quiconque lira le code, de comprendre son fonctionnement.

#### 4.3.1 Aperçu de la documentation (exemple sur 2 méthodes)

```

def __init__(self,x,y,moveType,maxWeight=0,batteryLevel=0,speed=0):
    """
        Constructeur de la classe Robot
        ***Param***
        int x - position x d'apparition du robot \n
        int y - position y d'apparition du robot \n
        str moveType - type de déplacement du robot (Aléatoire ou A*) \n
        int maxWeight - charge maximale que peut porter le robot \n
        int batteryLevel - niveau de charge du robot \n
        int speed - vitesse du robot
    """

def aEtoileMove(self, city, postTask)->list: #ajout d'une tâche dans la param pour ne pas avoir
    """
        Algorithme de déplacement A*.
        Permet au robot mis en paramètre de rejoindre un tâche en suivant l'algorithme A*.
        ***Param***
        robot (Robot::class) - robot auquel on applique l'algorithme \n
        postTask tuple - comporte les coordonnées de la tâche à atteindre
        ***Return***
        list de listes des différentes cases par lesquelles passer pour rejoindre la tâche. \n
        Liste sous le format suivant : [[case1],[case2],...,[caseTâche]] \n
        [caseI] = [Position x de la prochaine cellule, \n
                  Position y de la prochaine cellule, \n
                  Position x de la tâche sur laquelle le robot se situe (-1 si aucune case), \n
                  Position y de la tâche sur laquelle le robot se situe (-1 si aucune case)]
    """

```

## 4.4 Mise en place des bâtiments de recharge (14/11)

#### 4.4.1 Aperçu du code d'implémentation des bâtiments de recharge

```

stock = []
for i in range(3):
    alea = random.randint(0, len(self.buildings)-1)
    while (alea in stock) and (len(stock) < len(self.buildings)):
        alea = random.randint(0, len(self.buildings)-1)
    stock += [alea]
    self.buildings[alea].bType = "station"

for i in range(len(self.buildings)):
    if self.buildings[i].bType == "decor":
        buildings = canvas.create_rectangle(self.buildings[i].x*20+20,
                                              self.buildings[i].y*20+20,
                                              self.buildings[i].x*20+20+self.buildings[i].sizeX*20,
                                              self.buildings[i].y*20+20+self.buildings[i].sizeY*20,
                                              fill='blue')
    elif self.buildings[i].btype == "station":
        buildings = canvas.create_rectangle(self.buildings[i].x*20+20,
                                              self.buildings[i].y*20+20,
                                              self.buildings[i].x*20+20+self.buildings[i].sizeX*20,
                                              self.buildings[i].y*20+20+self.buildings[i].sizeY*20,
                                              fill='green')
        for j in range(self.buildings[i].sizeY):
            self.getCel(self.buildings[i].x-1, self.buildings[i].y+j).charge = True
            self.getCel(self.buildings[i].x+self.buildings[i].sizeX, self.buildings[i].y+j).charge = True
        for j in range(self.buildings[i].sizeX):
            self.getCel(self.buildings[i].x+j, self.buildings[i].y-1).charge = True
            self.getCel(self.buildings[i].x+j, self.buildings[i].y+self.buildings[i].sizeY).charge = True

```

Ce code permet la génération des stations de recharge en vérifiant qu'il y ait place autour de celles-ci pour que le robot puisse se recharger.

## 4.5 Mise en place de nouveaux mécanismes pour le robot (16/11)

### 4.5.1 Aperçu du code de vérification du niveau de batterie

```
def needLoading(self) -> bool :  
    """  
        Permet de savoir si un robot a besoin de passer à une station de rechargement ou non.  
        ***Return***  
        bool True si le robot a besoin de se recharger, False sinon.  
    """  
    if self.batteryLevel <= 30 :  
        return True  
    return False
```

Si il detecte qu'il est inférieur ou égale à 30 pourcent il va chercher à se recharger près de la station la plus proche.

### 4.5.2 Aperçu du code de recherche de station

```
def findLoadingFromRobotPos(self,city)->list:  
    """  
        Trouve la case de recharge en balayant la ville autour du robot.  
        ***Return***  
        list comportant les coordonnées de la tâche.  
    """  
    initialPos = (self.x,self.y)  
    #print("initial pos : ", initialPos)  
    digit = [-1,0,1]  
    currentDigit = 1  
    while len(city.tasks) > 0:  
        permut = itertools.product(digit,repeat=2)  
        #print("-----début de la Liste-----")  
        for i in permut:  
            #print(i)  
            if initialPos[0]+i[0]>=0 and initialPos[0]+i[0]<city.sizeX and initialPos[1]+i[1]>=0 and initialPos[1]+i[1]<city.sizeY and city.getCel(initialPos[0]+i[0],initialPos[1]+i[1]).charge == True:  
                #print("-----fin de la liste-----")  
                #print("cible : ",initialPos[0]+i[0],initialPos[1]+i[1])  
                return [initialPos[0]+i[0],initialPos[1]+i[1]]  
        currentDigit+=1  
        digit = [-currentDigit] + digit  
        digit += [currentDigit]  
    return [-1,-1]
```

Le robot va chercher la station la plus proche de lui.

### 4.5.3 Aperçu du code de vérification de position pour charger

```
def detectLoading(self, city)->list:  
    """  
        Permet de savoir si un robot est présent sur une case de rechargement ou non.  
        ***Param***  
        city (City::class) - ville dans laquelle la tâche est détectée.  
        ***Return***  
        list contenant les coordonnées de la case de recharge si le robot est dessus. \n  
        [-1,-1] si le robot n'est sur aucune case de rechargement.  
    """  
    if city.getCel(self.x,self.y).charge == True:  
        return [self.x,self.y]  
    return [-1,-1]
```

On vérifie si le robot est positionné sur la case rechargement pour commencer à le charger.

#### 4.5.4 Aperçu du code de déplacement dépendant de la charge du robot

```

def deplacement(self, canvas, fenetre):
    for i in range(len(self.robots)):
        """
        GARDER LE DEPLACEMENT ALÉATOIRE ?
        if self.robots[i].moveType == "alea":
            dep1 = self.robots[i].randomMove(self)
        .....
        if self.robots[i].isLoading == False :
            if self.robots[i].moveType == "a*":
                if self.robots[i].needLoading() == False :
                    target = self.robots[i].findTaskFromRobotPos(self)
                else :
                    target = self.robots[i].findLoadingFromRobotPos(self)
                chemin = self.robots[i].AetoileMove(self,target)
                dep1 = self.robots[i].moveItSelfFromCoord([chemin[0][0],chemin[0][1]] + [chemin[0][2],chemin[0][3]])
            dx = dep1[0]
            dy = dep1[1]
            if dep1[2] != -1:
                if self.robots[i].needLoading() == False and self.getCel(dep1[2],dep1[3]).status == "task":
                    canvas.delete(self.ourTasks[dep1[2],dep1[3]])
                    self.removeTask([dep1[2],dep1[3]])
                else :
                    self.robots[i].isLoading = True
                canvas.move(self.imgRobots[i],dx,dy)
            if self.robots[i].isLoading :
                if self.robots[i].batteryLevel + 5 <= self.robots[i].maxBattery :
                    self.robots[i].batteryLevel += 5
                else :
                    self.robots[i].batteryLevel = self.robots[i].maxBattery
                    self.robots[i].isLoading = False
        canvas.pack()
    fenetre.after(100, self.deplacement, canvas, fenetre)

```

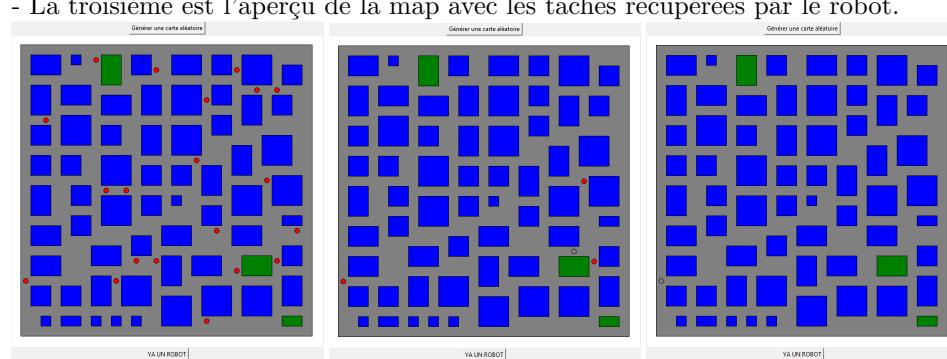
Le robot effectue l'algorithme A\* pour trouver des tâches jusqu'à ce que sa batterie passe en dessous de 30 pourcent.

Si celle ci passe en dessous, il va effectuer l'algorithme A\* afin de trouver la station de recharge la plus proche.

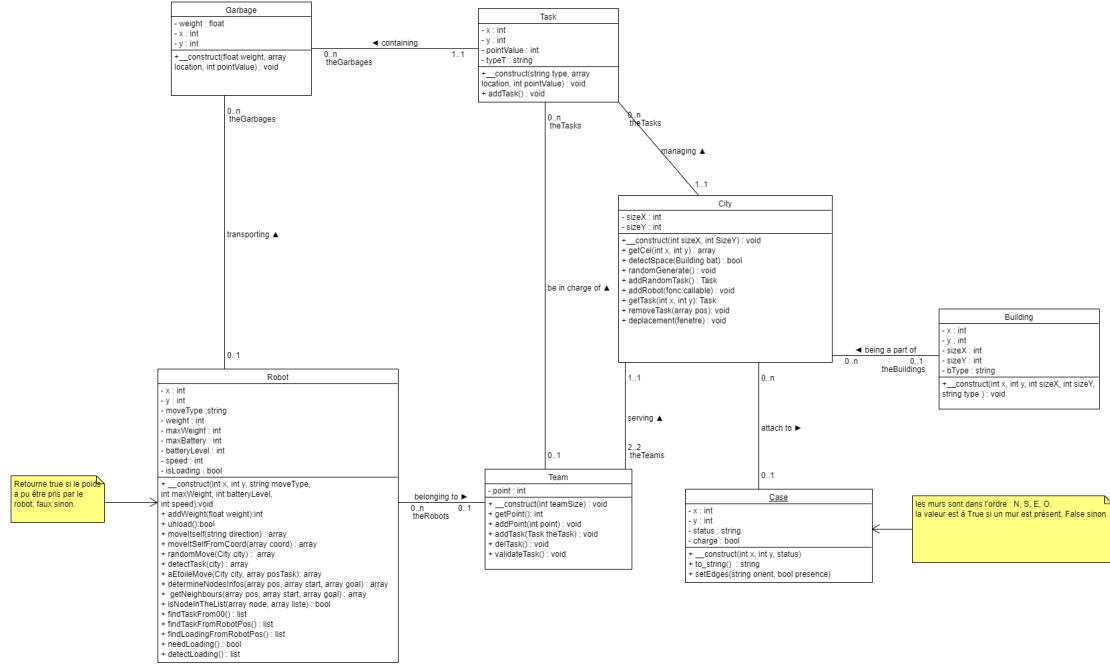
#### 4.5.5 Aperçu du test

Ci dessous 3 images du test:

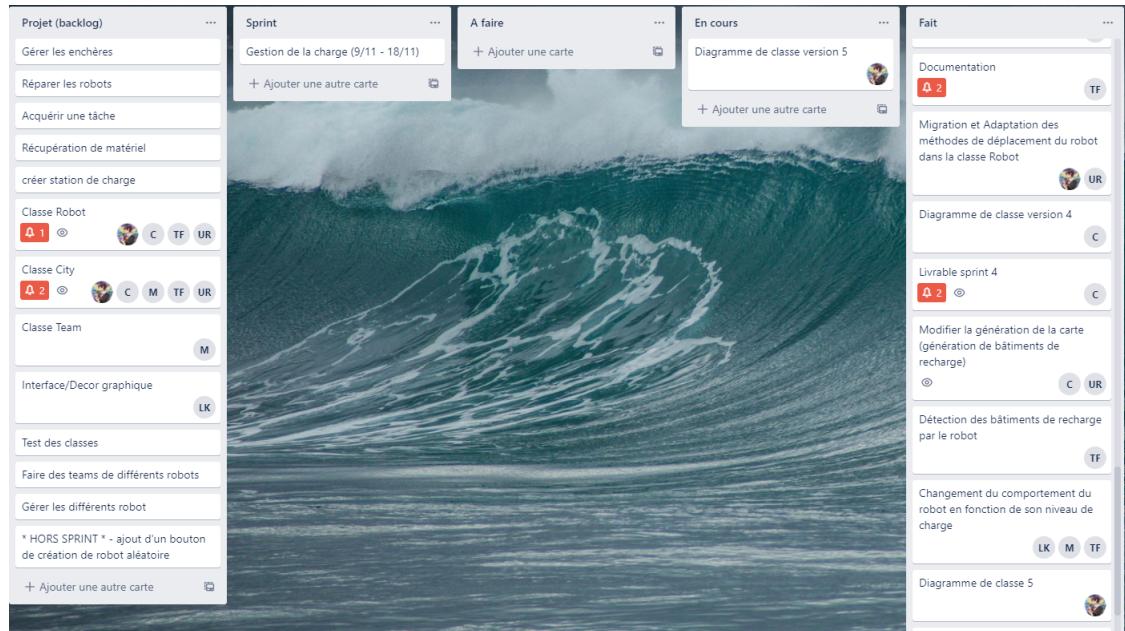
- La première est l'aperçu de la génération de la map avec les bâtiments.
- La deuxième est l'aperçu du déplacement du robot jusqu'à ce qu'il soit déchargé et qu'il décide d'abandonner sa tâche pour aller se recharger en se collant à la station de recharge.
- La troisième est l'aperçu de la map avec les tâches récupérées par le robot.



## 4.6 Mise à jour du Diagramme de classe (17/11)

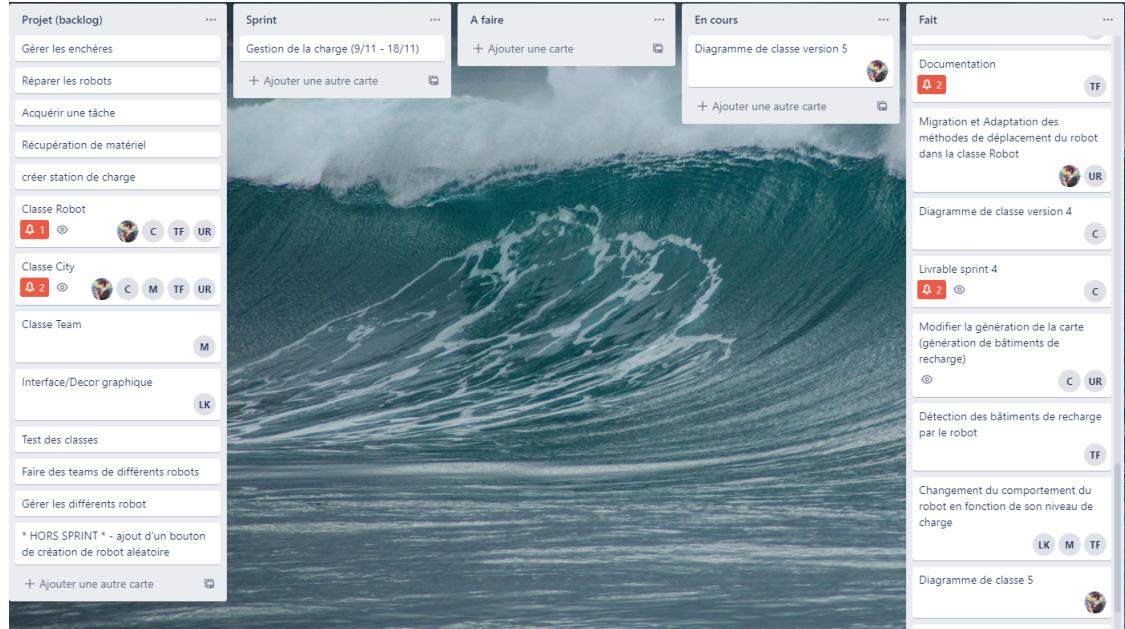


## 4.7 Aperçu des tâches à la fin du sprint 4 (17/11)



## 5 Sprint 5: Mise en place des Teams Amélioration de l'interface

### 5.1 Dernier jet de tâches sprint 3(17/11)



### 5.2 Mise en place des teams (25/11)

#### 5.2.1 Aperçu du code d'implémentation des teams

```
def generateTeams(self, canvas, nbTeams = 2, nbRobots = 2):
    color = ["blue", "yellow", "red", "green"]
    for i in range(nbTeams):
        self.teams += [Team(color[i])]
    for j in range(nbRobots):
        if color[i] == "blue":
            battery = random.randint(60,100)
            self.teams[i].addRobot(Robot(0,0,"a*","blue",canvas,self,battery,battery))
            #self.imgRobots += [canvas.create_oval(25,25,35,35,fill="blue")]
        elif color[i] == "yellow":
            self.teams[i].addRobot(Robot(self.sizeX-2,self.sizeY-2,"a*","yellow",canvas,self))
```

Ce code permet la création du nombre de teams souhaité et le nombre de robot par teams. Ici on génère les robots de chaque team avec une batterie maximum aléatoire entre 60 et 100 pourcent

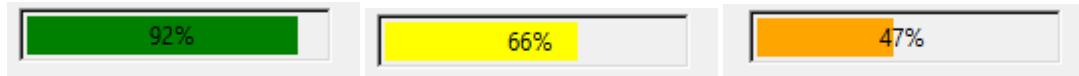
## 5.3 Amélioration de l'interface (28/11)

### 5.3.1 Mise en place de l'affichage du niveau de batterie (pour 1 robot uniquement pour le moment)

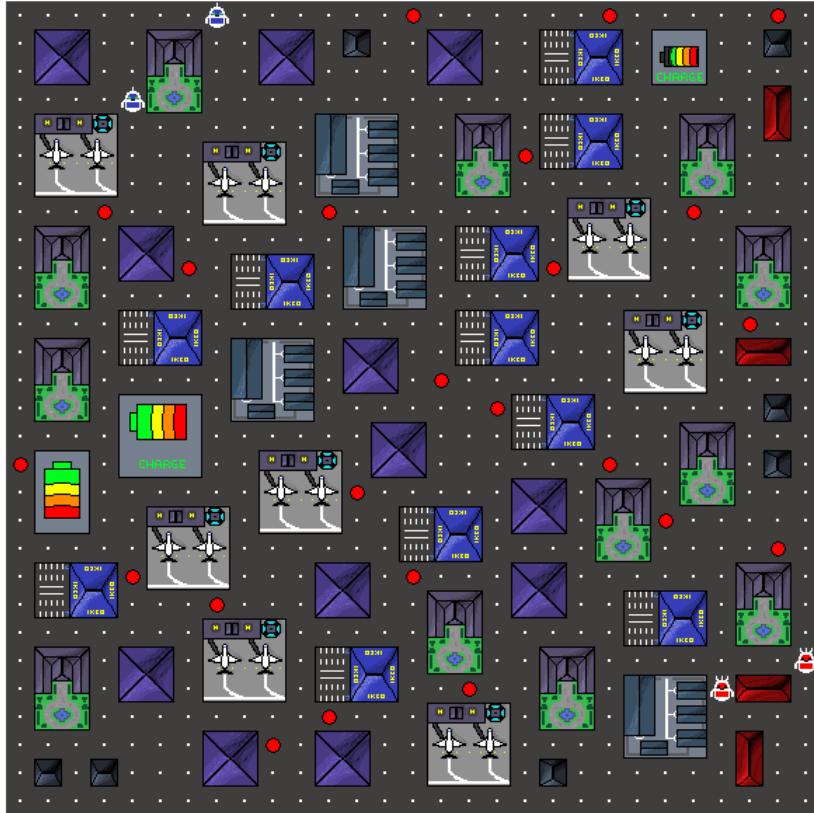
```
def showbattery(self, meter):
    battery = self.batteryLevel
    meter.config(value=battery/100)
    if battery < 70:
        meter.config(fill="yellow")
    if battery < 50:
        meter.config(fill="orange")
    if battery < 30:
        meter.config(fill="red")
    meter.update()
```

Ce code permet l'actualisation de la valeur de la batterie ainsi que la couleur en fonction du niveau de charge.

### 5.3.2 Aperçu du test en fonction de l'évolution du robot dans la map



### 5.3.3 Mise en place de l'interface graphique (images des bâtiments/route)



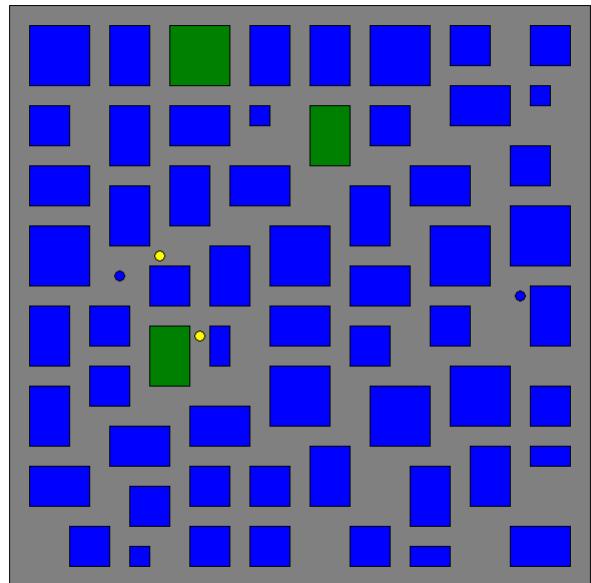
### 5.3.4 Implémentation du zoom + possibilité de se déplacer dans la carte (sans texture) [En cours]

#### 5.3.5 Aperçu du code

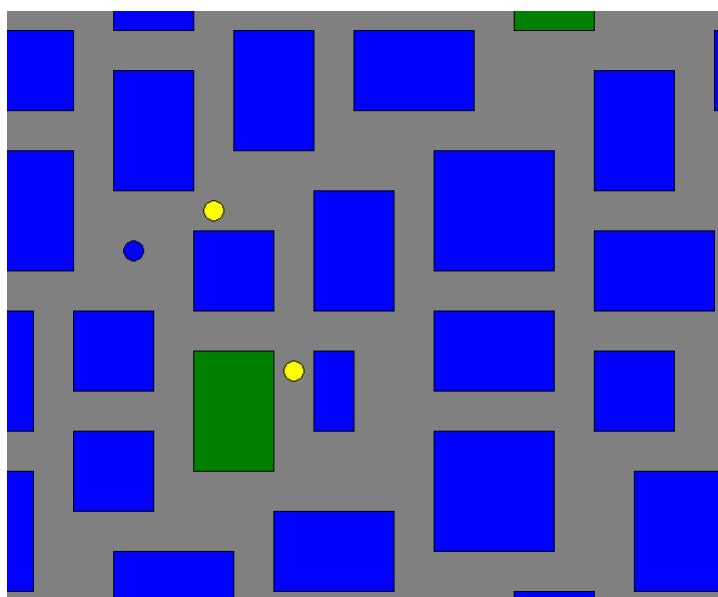
```
#move
def move_start(event):
    canvas.scan_mark(event.x, event.y)
def move_move(event):
    canvas.scan_dragto(event.x, event.y, gain=1)

#windows zoom
def zoomer(event):
    if (event.delta > 0):
        canvas.scale("all", event.x, event.y, 2, 2)
        testCity.mult *= 2
    elif (event.delta < 0):
        canvas.scale("all", event.x, event.y, 0.5, 0.5)
        testCity.mult *= 0.5
    canvas.configure(scrollregion = canvas.bbox("all"))
```

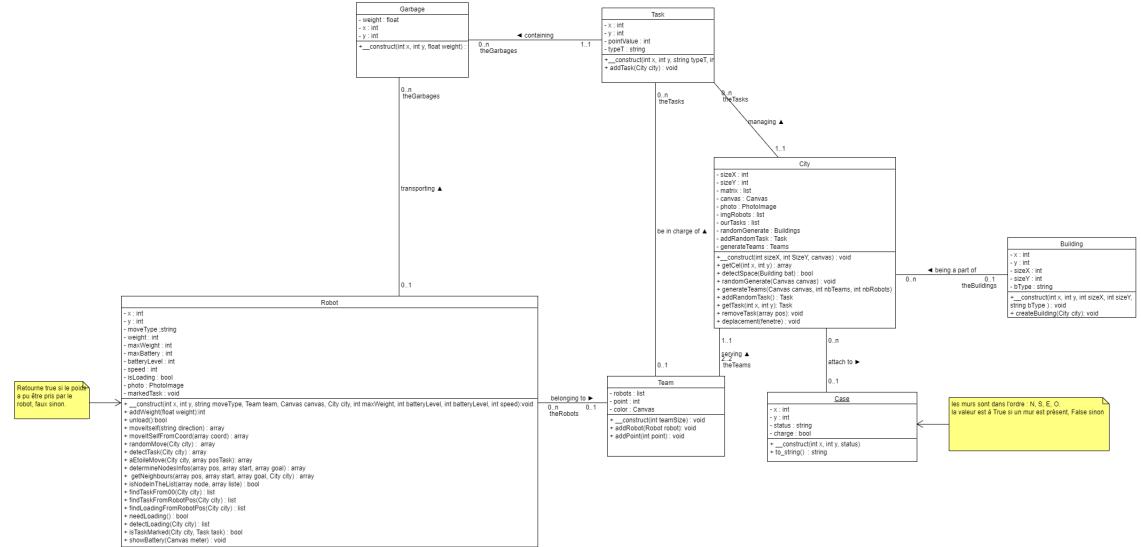
### 5.3.6 Aperçu du test sans zoom



### 5.3.7 Aperçu du test avec zoom



## 5.4 Mise à jour du Diagramme de classe (02/11)

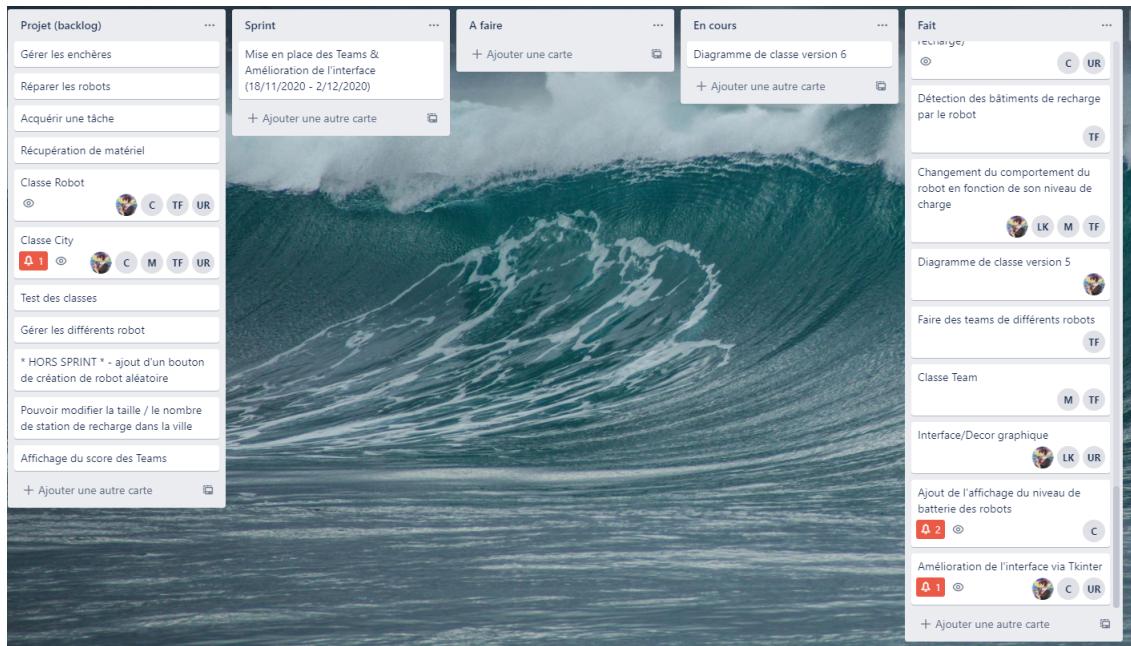


## 5.5 Aperçu des tâches à la fin du sprint 5 (02/11)

Projet (backlog)	Sprint	A faire	En cours	Fait
<ul style="list-style-type: none"> <li>Gérer les enchères</li> <li>Réparer les robots</li> <li>Acquérir une tâche</li> <li>Récupération de matériel</li> <li>Classe Robot</li> <li>Classe City</li> <li>Test des classes</li> <li>Gérer les différents robot</li> <li>* HORS SPRINT * - ajout d'un bouton de création de robot aléatoire</li> <li>Pouvoir modifier la taille / le nombre de station de recharge dans la ville</li> <li>Affichage du score des Teams</li> </ul>	<p>Mise en place des Teams &amp; Amélioration de l'interface (18/11/2020 - 2/12/2020)</p>	<p>+ Ajouter une carte</p>	<p>Diagramme de classe version 6</p> <p>+ Ajouter une autre carte</p>	<p>Détection des bâtiments de recharge par le robot</p> <p>Changement du comportement du robot en fonction de son niveau de charge</p> <p>Diagramme de classe version 5</p> <p>Faire des teams de différents robots</p> <p>Classe Team</p> <p>Interface/Decor graphique</p> <p>Ajout de l'affichage du niveau de batterie des robots</p> <p>Amélioration de l'interface via Tkinter</p>

## 6 Sprint 6: Système d'enchères Amélioration de l'interface Zoom/Déplacement dans la map

### 6.1 Dernier jet de tâches sprint 4(09/12)



## 6.2 Mise en place du système d'enchère (05/12)

### 6.2.1 Aperçu du code d'implémentation des enchères

```
def auctionAgainstOtherTeam(self,task,team):
    """
    Détermine à quelle équipe est remise une tâche sur le principe des enchères
    ***Param***
    task (Task::class) - tâche à départager \n
    team (Team::class) - 2e équipe voulant la tâche
    """
    verif = False
    currentTeam = 0
    bidAgain = 0.1
    while verif == False:
        if random.random() > bidAgain :
            bidAgain += 0.1
            currentTeam += 1
        else :
            verif = True
        if currentTeam%2 == 0 :
            self.point -= bidAgain*100
            task.belongsTo = self.color
        else:
            team.point -= bidAgain*100
            task.belongsTo = team.color
```

Ce code permet de déterminer à quelle équipe est remise la tâche.

## 6.3 Amélioration de l'interface (28/11 - 15/12)

### 6.3.1 Mise en place de l'affichage du niveau de batterie + label des teams et des robots

#### 6.3.2 Aperçu du code

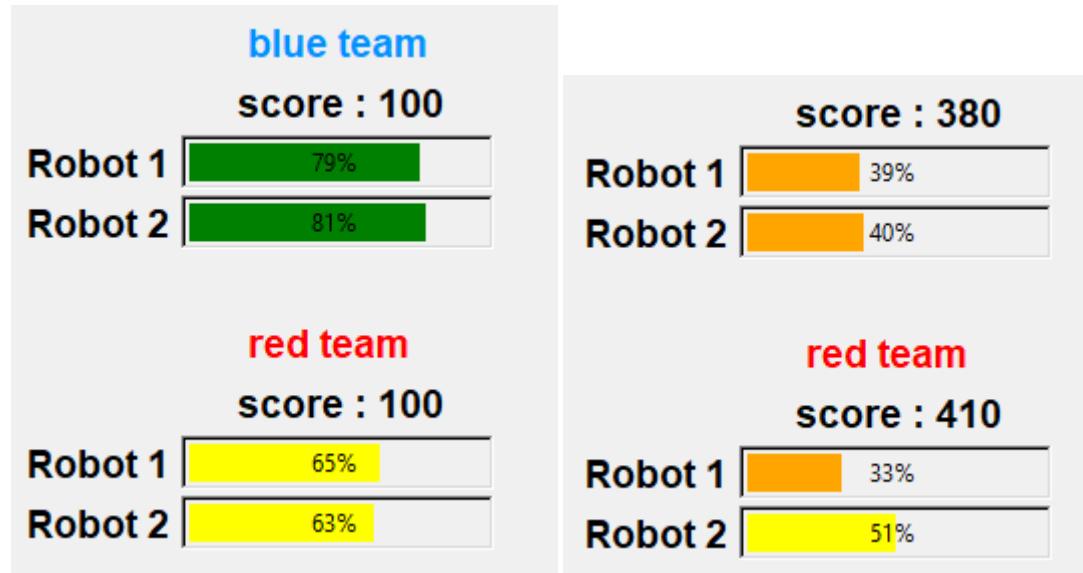
```
k = 0
for i in range(len(self.teams)):
    label = Label(fenetre, text=f"{self.teams[i].color} team", font='Helvetica 14 bold')
    if self.teams[i].color == "blue":
        label.config(fg="#0091ff")
    else:
        label.config(fg="red")
    #print("création label")
    label1 = Label(fenetre, text=f"score : {int(self.teams[i].point)}", font='Helvetica 14 bold')
    label.pack()
    label1.pack()
    label.place(x=130,y=60+30*k)
    label1.place(x=125,y=90+30*k)
    for j in range(len(self.teams[i].robots)):
        label = Label(fenetre, text=f"Robot {j+1}", font='Helvetica 14 bold')
        label.pack()
        label.place(x=20,y=120+30*k)
        self.teams[i].robots[j].meter.pack()
        self.teams[i].robots[j].meter.place(x=100,y=120+30*k)
        self.teams[i].robots[j].showbattery(self.teams[i].robots[j].meter)
        k+=1
    k+=3
fenetre.after(50*len(self.teams)*len(self.teams[i].robots), self.deplacement, canvas, fenetre)
```

Ce code permet l'affichage des batteries ainsi que des labels de teams et de robots.

```
def showbattery(self, meter):
    """
    Affichage de la barre de chargement du robot en fonction de son niveau decharge actuel
    ***param***
    meter - barre de chargement
    """
    battery = self.batteryLevel
    meter.config(value=battery/100)
    if battery > 80:
        meter.config(fill="green")
    if battery < 70:
        meter.config(fill="yellow")
    if battery < 50:
        meter.config(fill="orange")
    if battery < 30:
        meter.config(fill="red")
    meter.update()
```

Ce code permet l'actualisation de la batterie en + du changement de couleur en fonction du niveau de celle-ci.

### 6.3.3 Aperçu du test en fonction de l'évolution des robots dans la map

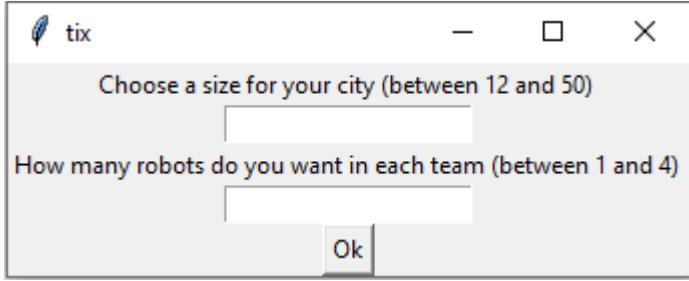


### 6.3.4 Mise en place du choix de taille de la map et du nombre de robot + implémentation de zoom/déplacement dans la map

#### 6.3.5 Aperçu du code

```
def popup(b):
    top=Toplevel(fenetre)
    l=Label(top,text="Choose a size for your city (between 12 and 50) ")
    l.pack()
    e=Entry(top)
    e.pack()
    l=Label(top,text="How many robots do you want in each team (between 1 and 4) ")
    l.pack()
    er=Entry(top)
    er.pack()
    be=Button(top,text='Ok',command=lambda:cleanup(e, er, top))
    be.pack()
    b["state"] = "disabled"
    fenetre.wait_window(top)
    b["state"] = "normal"
```

Ce code permet l'affichage du popup du choix de la taille de la carte ainsi que du nombre de robot par team.



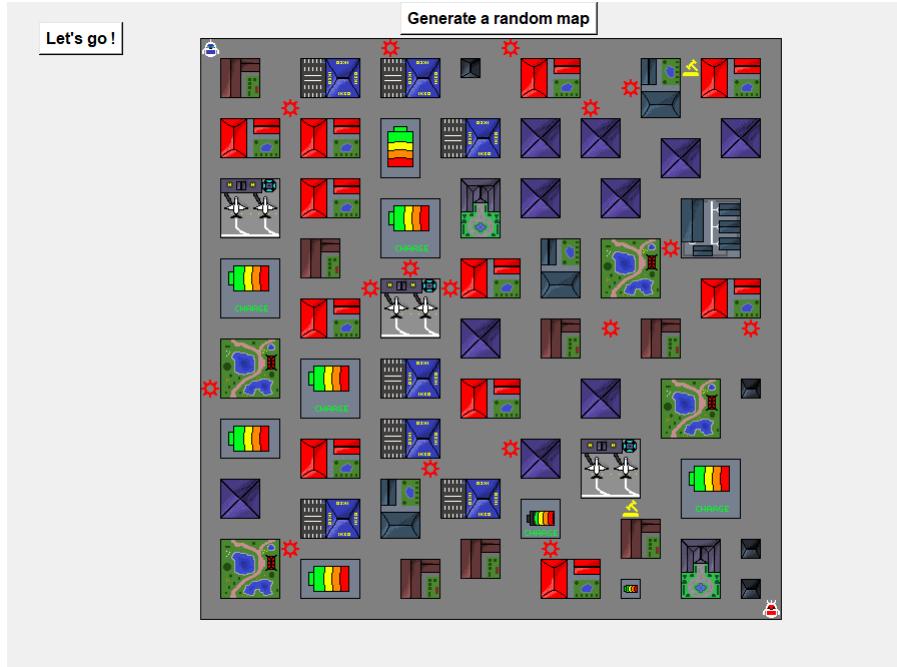
### 6.3.6 Implémentation du zoom + possibilité de se déplacer dans la carte

### 6.3.7 Aperçu du code

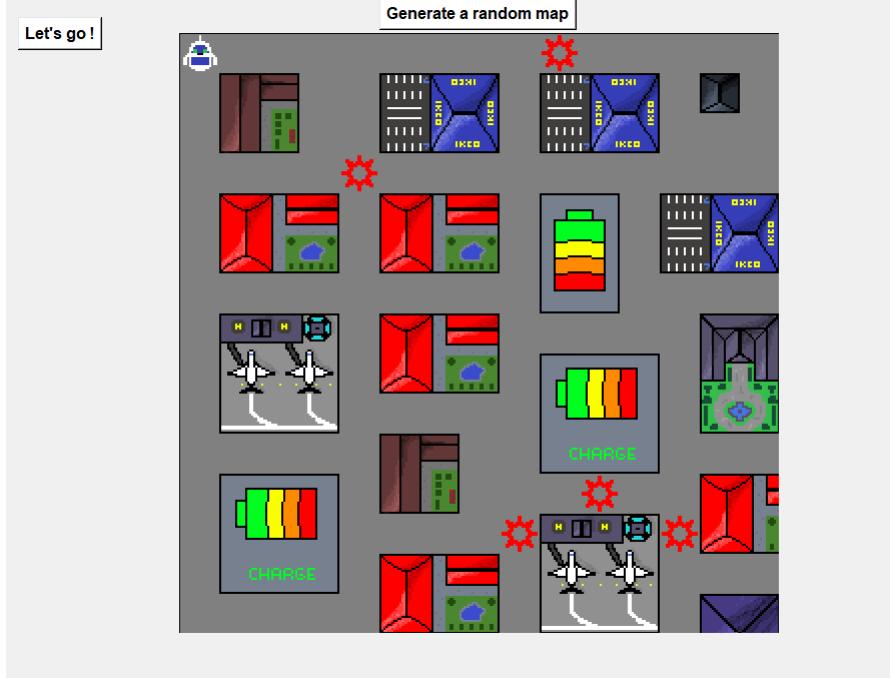
```
#move
def move_start(event):
    canvas.scan_mark(event.x, event.y)
def move_move(event):
    canvas.scan_dragto(event.x, event.y, gain=1)

#windows zoom
def zoomer(event):
    if (event.delta > 0):
        canvas.scale("all", event.x, event.y, 2, 2)
        testCity.mult *= 2
    elif (event.delta < 0):
        canvas.scale("all", event.x, event.y, 0.5, 0.5)
        testCity.mult *= 0.5
    canvas.configure(scrollregion = canvas.bbox("all"))
```

### 6.3.8 Aperçu du test sans zoom



### 6.3.9 Aperçu du test avec zoom



### 6.3.10 Affichage des enchères en cours

#### 6.3.11 Aperçu du code

```

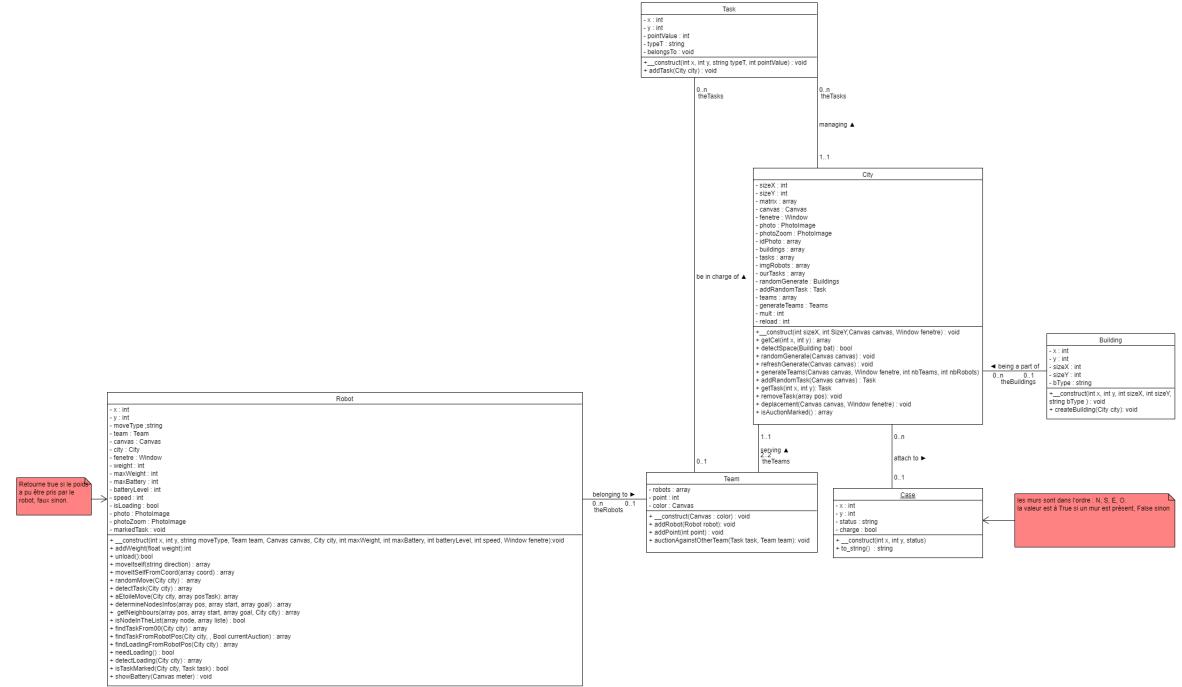
if currentAuction[0] == True:
    if self.labelench == []:
        self.labelench.append(Label(fenetre, text=f"Robot {currentAuction[0]+1} / {self.teams[currentAuction[0]].color} team is participating in an auction", font='Arial 12', fg="white", bg="#036A73"))
        self.labelench[0].place(x=920,y=150)
        self.labelench[0].pack()
        self.labelench[0].place(x=920,y=150)
    if self.teams[0].color == "blue":
        c = 0
    else:
        c = 4
    self.labelench.append(Label(fenetre, image=self.teams[currentAuction[0]].robots[currentAuction[0]].photo[self.teams[currentAuction[0]].robots[currentAuction[0]].numRobot+c], bg="#036A73"))
    self.labelench[1].pack()
    self.labelench[1].place(x=1030,y=126)
else:
    while self.labelench != []:
        self.labelench[0].destroy()
        self.labelench.pop(0)

```

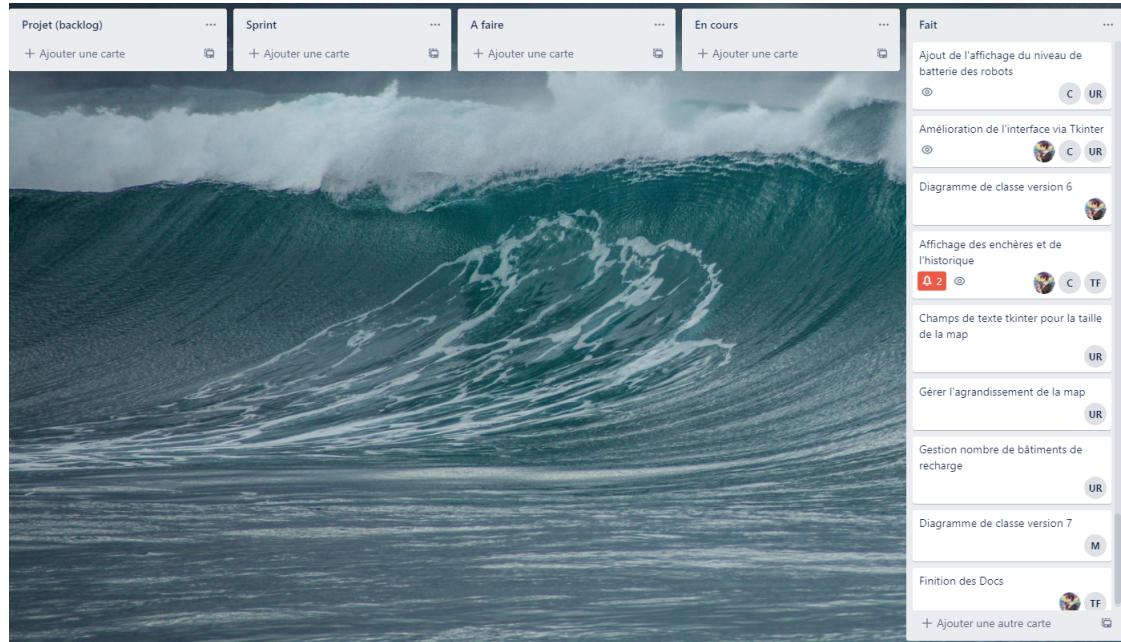
#### 6.3.12 Aperçu du résultat



## 6.4 Mise à jour du Diagramme de classe (09/12)



## 6.5 Aperçu des tâches à la fin du sprint 6 (18/12)



## 6.6 Démo vidéo

Cliquez ici pour vous rendre sur la démo vidéo.