

# RAPPORT DE STAGE TECHNIQUE

Dynamic Object Language Labs, Inc.



114 Waltham Street Lexington, MA 02421

ESQESE (UCLy)



10 place des Archives, Lyon 69002

11 mai 2020 – 31 juillet 2020

Bachelor Sciences du Numérique

Heidi SOUIBKI  
Intern

Paul ROBERTSON  
Manager

# Table des matières

<b>GLOSSAIRE.....</b>	<b>1</b>
<b>INTRODUCTION.....</b>	<b>3</b>
<b>I/ L'ENTREPRISE ET SON SECTEUR D'ACTIVITÉ.....</b>	<b>4</b>
Le secteur : La recherche en Intelligence Artificielle.....	4
L'entreprise par rapport au secteur .....	6
<b>II/ LE CADRE DU STAGE.....</b>	<b>7</b>
Description de l'entreprise .....	7
Déroulé du stage .....	8
<b>III/ LES TRAVAUX EFFECTUÉS .....</b>	<b>9</b>
Portage de GAIT Java sur Linux.....	9
Configuration de Gazebo.....	9
Actions du Turtlebot3.....	13
Quaternions.....	15
GAIT & RabbitMQ.....	16
<b>CONCLUSION.....</b>	<b>21</b>
<b>ANNEXES .....</b>	<b>22</b>
Portage de GAIT Java sur Linux.....	23
Installation de VirtualBox, Ubuntu et ROS .....	43
Mise en place de la simulation Gazebo sous ROS .....	46

# REMERCIEMENTS

Avant tout développement sur cette expérience professionnelle, il semble nécessaire de commencer ce rapport de stage par des remerciements, à ceux qui m'ont beaucoup appris au cours de ce stage et aussi à ceux qui ont eu la gentillesse de faire de ce stage un moment très profitable.

Aussi, je remercie tout particulièrement mon maitre de stage, M. Paul ROBERTSON, directeur de DOLL Inc., qui a eu l'amabilité de m'accueillir comme stagiaire au sein de sa société.

Je tiens également à remercier Jianyong XUE pour son aide précieuse, ses conseils avisés et son temps.

Je remercie M. Olivier GEORGEON pour avoir particulièrement bien supervisé ce stage et m'avoir permis, malgré ces temps difficiles, de travailler dans les meilleures conditions.

Pour finir, je remercie l'équipe de DOLL, Daniel CERYs, Prakash MANGHWANI et Thao PHUONG pour leur bienveillance et leur accueil bien que j'aurais aimé avoir l'occasion de mieux les connaître.

# GLOSSAIRE

Cette section liste les termes techniques spécifiques. **Les mots en gras** sont ceux que j'ai définis comme technique, comme par exemple les noms de fichiers ou encore les commandes systèmes. Les mots soulignés que vous rencontrerez tout au long de votre lecture sont définis ici.

**BitBucket** : Bitbucket est un service web d'hébergement et de gestion de développement logiciel utilisant le logiciel de gestion de versions Git.

**CPS** : Cyber-Physical System. Systèmes informatiques agissant physiquement.

**DARPA** : Defense Advanced Research Projects Agency. Agence américaine à propos de la recherche.

**Gazebo** : Gazebo est un simulateur 3D, cinématique, dynamique et multirobot permettant de simuler des robots articulés dans des environnements complexes, intérieurs ou extérieurs, réalistes et en trois dimensions.

**GAIT** : Generating and Analyzing Interaction Traces. Algorithme d'intelligence artificielle.

**Multithreading** : Le multithreading permet d'exécuter plusieurs séquences d'instructions (threads) parallèlement les unes aux autres dans le cadre d'un processus.

**Python** : Python est un langage de programmation placé sous licence libre, fonctionnant sur la plupart des plates-formes informatiques et donc très populaire.

**Quaternions** : Les quaternions englobent les nombres réels et complexes dans un système de nombres où la multiplication n'est plus une loi commutative. Les quaternions sont ainsi le premier exemple de nombres hypercomplexes. Très utilisés en Physique et Informatique.

**RabbitMQ** : RabbitMQ est un logiciel d'agent de messages open source qui implémente le protocole AMQP, il permet de gérer des files de messages afin que différents clients/logiciels puissent communiquer entre eux.

**ROS** : Robot Operating System, est un ensemble d'outils informatiques open source permettant de développer des logiciels pour la robotique.

**Théorie de l'esprit** : La théorie de l'esprit (ToM) désigne l'aptitude cognitive permettant à un individu d'attribuer des états mentaux inobservables comme l'intention ou le désir à lui-même ou à d'autres individus.

**Thread** : une suite d'instructions déroulées séquentiellement. Un programme/processus contient au moins un thread.

**Turtlebot3** : Le Turtlebot3 est un petit robot abordable et programmable, conçu pour être utilisé sous ROS. Il est principalement utilisé dans l'éducation et la recherche.

**Virtualenv** : virtualenv est un outil pour créer des environnements virtuels Python isolés. virtualenv crée un dossier qui contient tous les exécutables nécessaires pour utiliser les paquets qu'un projet Python pourrait nécessiter.

# INTRODUCTION

Du 11 mai 2020 au 31 juillet 2020, j'ai effectué un stage au sein de l'entreprise DOLL Inc. Située à Boston, dans l'État du Massachusetts, aux États-Unis. Ce stage a été, pour moi, l'occasion d'en apprendre énormément sur la robotique et l'intelligence artificielle.

Plus largement, ce stage a été pour moi l'opportunité de découvrir pour la première fois le développement logiciel et l'administration et la gestion des systèmes qui en découle. J'ai pu remarquer les difficultés que l'on rencontre lorsque l'on doit travailler sur des outils développés par d'autres et la nécessité d'en discuter avec eux afin d'être le plus efficace possible. L'influence que ce stage aura dans mes futures orientations professionnelles n'est pas à négliger.

Le premier objectif de ce rapport est de présenter, à l'écrit et en détails, le déroulement du stage, ce que j'ai pu apprendre et les travaux que j'y ai menés. Le second, est de permettre, à condition d'avoir les connaissances nécessaires, de reproduire et de reprendre mon travail. C'est pourquoi la partie concernant ce travail se présentera sous la forme d'une documentation.

Dans une première partie, je présenterais DOLL Inc. ainsi que son secteur d'activité, dans une seconde partie, je détaillerais le cadre dans lequel le stage a pris forme et, enfin, j'exposerais les travaux que j'ai effectués tout au long de celui-ci.

# I/ L'ENTREPRISE ET SON SECTEUR D'ACTIVITÉ

## 1. Le Secteur : La recherche en Intelligence Artificielle

### A. L'intelligence Artificielle

Historiquement, l'intelligence artificielle émerge dans les années 1950, en même temps que l'informatique moderne, dans les travaux d'Alan Turing. La question est alors de savoir si une machine peut « penser ». Cette problématique donnera par la suite lieu au fameux test de Turing.

Dans les années 1980, l'apprentissage automatique se développe et des algorithmes « apprenants » sont développés. Par exemple, Deep Blue bat le meilleur joueur d'échecs.

L'intelligence artificielle va devenir, peu à peu, un domaine de recherche international, principalement aux États-unis, en Europe et en Chine, un peu au Japon.

Vers les années 2000, le big data et autres technologies récentes permettent d'explorer des masses de données sans précédent. L'apprentissage profond apparaît, l'un de ses inventeurs est le français Yann Le Cun.

Depuis 2015, différents grands défis se sont imposés à l'intelligence artificielle : la perception visuelle, la compréhension du langage naturel (écrit ou parlé), l'analyse automatique ou encore la prise de décision autonome. Pouvoir produire et organiser de grandes quantités de données de qualité ou la capacité de déduire et de généraliser de manière pertinente à partir de peu de données sont d'autres objectifs à plus long terme.

Les investissements dans ce secteur ont été décuplés ces dernières années et ne cessent d'augmenter, atteignant des dizaines de milliards de dollars.

### B. La recherche

Dans le domaine militaire, la Défense nationale ou Sécurité nationale désigne l'ensemble des moyens civils et militaires mis en œuvre par un État pour assurer l'intégrité de son territoire, la protection de sa population ou la sauvegarde de ses intérêts. Le budget

alloué pour sa défense ainsi que le nombre de soldats, son avancée technologique ou encore le type et la quantité de matériel dont il dispose sont les principaux facteurs définissant la puissance militaire d'un État.

Dans le cas des États-Unis, c'est le Département de la Défense (Le Pentagone) qui est en charge de toutes ces questions. Le Département de la défense se divise en différentes institutions dans lesquelles on retrouve les différentes forces armées (US Army, US Navy, US Air Force, etc...) ainsi qu'une multitude d'agences comme la Defense Intelligence Agency (DIA), la National Security Agency (NSA), le Cyber Crime Center (DC3) ou encore la Defense Advanced Research Projects Agency (DARPA).

La DARPA est l'agence chargée de la recherche et du développement des nouvelles technologies destinées à un usage militaire. Elle est à l'origine de nombreuses technologies qui ont eu un impact considérable dans le monde entier comme par exemple ARPANET, ancêtre d'Internet. La mission de la DARPA est de « faire des investissements cruciaux dans les technologies révolutionnaires pour la sécurité nationale des États-Unis ».

Actuellement, la DARPA gère différents projets :

- Boeing X-37, navette spatiale
- Boeing SolarEagle, drone solaire
- XOS, exosquelette
- BigDog, robot pouvant porter des charges lourdes
- Atlas, robot humanoïde
- Plusieurs projets concernant les modifications génétiques
- IND, interface de communication directe entre un cerveau et un dispositif externe



## 2. L'entreprise par rapport au secteur

Comme souvent, les grandes innovations technologiques sont développées dans le cadre militaire puis portées dans les autres domaines. L'intelligence artificielle n'échappe pas à la règle. Aussi, DOLL Inc. est une entreprise subventionnée par la DARPA, elle-même dépendante du Pentagone.

L'entreprise fait de la recherche en intelligence artificielle destinée à un usage militaire. Cependant, aucune mise en application n'est envisagée avant au moins 20 ou 30 ans. DOLL Inc. concentre son activité sur les systèmes cyber-physiques (CPS) et sur leur autonomie. Un système cyber-physique est un système où les éléments informatiques collaborent pour le contrôle d'entités physiques, l'exemple le plus parlant est celui de la voiture autonome.

Les recherches de l'entreprise s'étendent également à la robotique, aux applications possibles de l'intelligence artificielle en matière de cybersécurité, à l'apprentissage machine ou encore la vision par ordinateur.

DOLL travaille en collaboration avec différentes universités. En particulier avec le Massachusetts Institute of Technology (MIT) et la Vanderbilt University (VU) située à Nashville dans le Tennessee.

## II/ LE CADRE DU STAGE

### 1. Description de l'entreprise

Dynamic Object Language Labs, Inc. est une société américaine fondée en juin 1993 par Dr Paul ROBERTSON, qui en deviendra le président, et Dr Robert LADDAGA. Ils ont tous deux travaillé en tant que chercheurs au MIT. L'entreprise n'est pas bien grande puisqu'en plus de ses deux fondateurs, elle n'emploie que trois personnes : Daniel CERYs, Prakash MANGHWANI et Thao PHUONG. Tous travaillent en tant que développeurs logiciels.

L'entreprise se concentre donc complètement sur ses activités de recherches. Pour tout ce qui concerne la partie gestion et administration, le président fait appel aux services adéquats : avocats, comptables, etc...

L'entreprise mène différents projets en parallèle, voici une petite liste non exhaustive de ces projets :

- ADAM : Projet sur l'apprentissage machine. L'objectif est de développer un outil capable de reprendre la main sur un avion hors de contrôle afin d'éviter une catastrophe. On utilise un quadricoptère (drone) et un simulateur. Le champ d'application est très large, il pourrait être utilisé dans de nombreux cas comme, par exemple, dans les systèmes des centrales nucléaires.
- RITA (Robust Intelligence Team Assistant) : Projet se basant sur la théorie de l'esprit. L'objectif est de pouvoir développer, à l'avenir, des robots capables de travailler en équipes. Cela permettrait également de construire des robots possédant des modèles de l'esprit humain, ces robots pourraient alors communiquer et travailler directement avec des soldats.

## 2. Déroulement du stage

L'entreprise étant basé à Boston, il était initialement prévu que j'aille effectuer mon stage sur place, dans les locaux de DOLL Inc., à Lexington. J'ai donc entrepris toutes les démarches nécessaires : signature du DS-7002 puis recours à un sponsor (Parenthèse Paris) pour obtenir mon DS-2019 puis mon Visa à l'ambassade américaine.

Malheureusement, c'est à ce moment que la crise sanitaire liée au COVID-19 a pris toute son ampleur. Les décisions prises au niveau national et international comme le confinement et la fermeture des frontières ont forcé l'annulation de mon voyage.

Le stage dans son intégralité a donc eu lieu en télétravail. J'ai travaillé seul sur le projet Init. Un projet initié par Olivier GEORGEON et Paul ROBERTSON à propos d'un algorithme d'apprentissage machine appelé GAIT et développé par Jianyong XUE.

Sur toute la durée du stage, nous avons planifié une réunion journalière à 13h à Lyon (7h à Boston) sur le modèle des Daily Standup Meetings en Scrum. Il y avait également une réunion hebdomadaire à 21h à Lyon (15h à Boston) avec toute l'équipe DOLL Inc. où un point était fait sur l'avancée des projets en cours grâce à des démonstrations. A chaque fin de journée (ou presque), je poussais mon code sur un dépôt distant BitBucket, pour que M. ROBERTSON puisse avoir accès à mon travail et puisse m'aider à avancer le plus efficacement possible.

# III/ LES TRAVAUX EFFECTUÉS

## 1. Portage de GAIT Java sur Linux

Cette partie explique la procédure à suivre pour l'installation de GAIT Java sur Linux, elle est rédigée en anglais. Elle est devenue quelque peu obsolète car une version de GAIT en Python a été rendue disponible depuis. J'ai donc décidé de la mettre en annexe.

## 2. Configuration de Gazebo

Cette partie a pour objectif de créer un environnement virtuel, dans Gazebo, pour que le Turtlebot3 puisse évoluer dans les conditions requises aux expériences que j'ai menées.

La première chose à faire dans cette partie a été de créer un fichier ayant comme extension **.launch** et un autre fichier ayant comme extension **.world**. Ces deux fichiers sont nécessaires au lancement d'une simulation Gazebo. Pour faire cela, la manière la plus simple est de récupérer les fichiers **turtlebot3\_empty\_world.launch** et **empty.world** et de copier leur contenu dans deux nouveaux fichiers que j'ai nommé : **turtlebot3\_my\_world.launch** et **turtlebot3\_my\_world.world**.

Ces nouveaux fichiers doivent impérativement être dans le même dossier que le fichier de base. Il est très important de noter que ces fichiers doivent être enregistrés au format XML.

Les emplacements sont les suivant :

- `~/catkin_ws/src/turtlebot3_simulations/turtlebot3_gazebo/launch`
- `~/catkin_ws/src/turtlebot3_simulations/turtlebot3_gazebo/worlds`

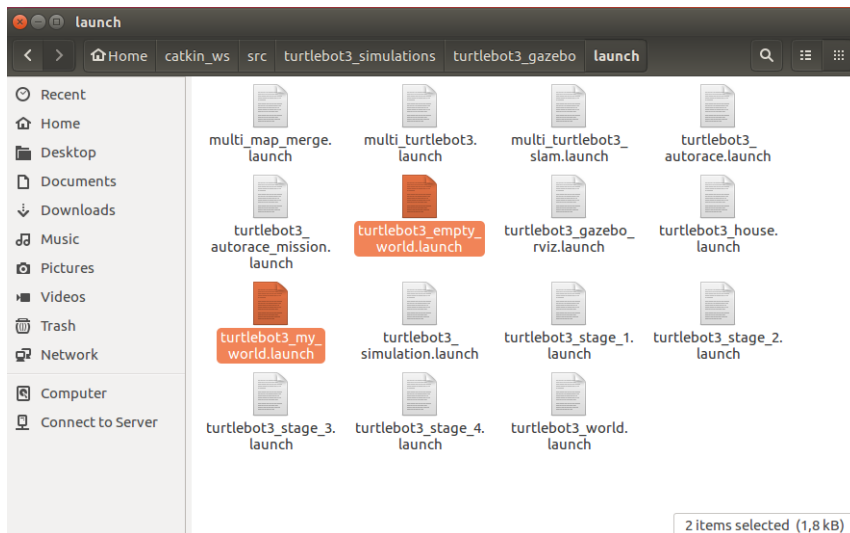


Figure 1 : Emplacement du fichier **turtlebot3\_my\_world.launch**, copie de **turtlebot3\_empty\_world.launch**.

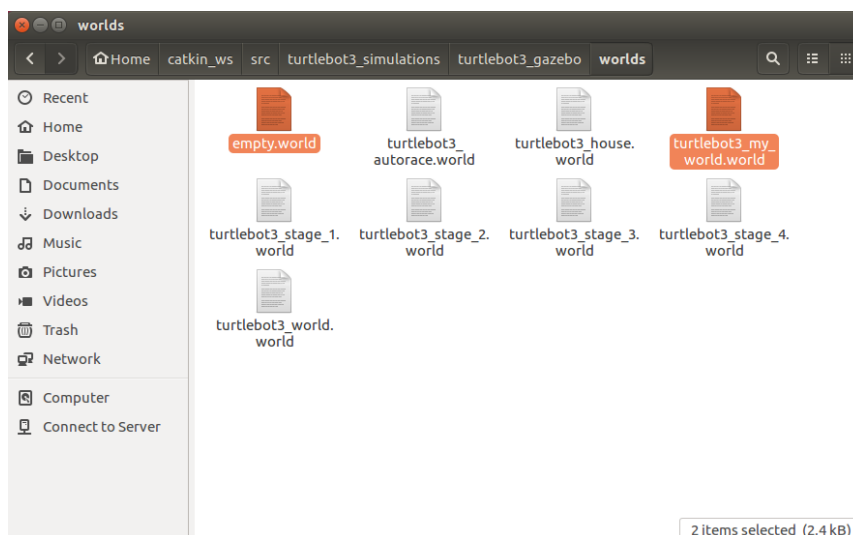


Figure 2 : Emplacement du fichier **turtlebot3\_my\_world.world**, copie de **empty.world**.

En lançant une simulation appelant ces deux fichiers, Gazebo crée un monde vide au centre duquel se trouve le Turtlebot3.

La deuxième étape est de modifier ces fichiers pour qu'ils lancent, non pas un monde vide, mais un monde en adéquation avec nos expériences. Il s'agit de créer le labyrinthe de GAIT. J'en ai également profité pour recréer le labyrinthe de DOLL, le monde contiendra donc deux labyrinthes.

Ne pas oublier de compiler l'espace de travail, rendez-vous donc dans le dossier `~/catkin_ws` puis exécuter la commande suivante : **catkin\_make**. Les nouveaux fichiers sont maintenant prêts à l'utilisation.

Une fois la recompilation effectuée, je peux lancer la simulation du Turtlebot3 dans ce monde vide que j'ai créé grâce à la commande suivante : **roslaunch turtlebot3\_gazebo turtlebot3\_my\_world.launch**.

La création des labyrinthes se fera directement dans Gazebo. Pour passer en mode construction dans Gazebo, dans le menu : Edit > Building Editor.

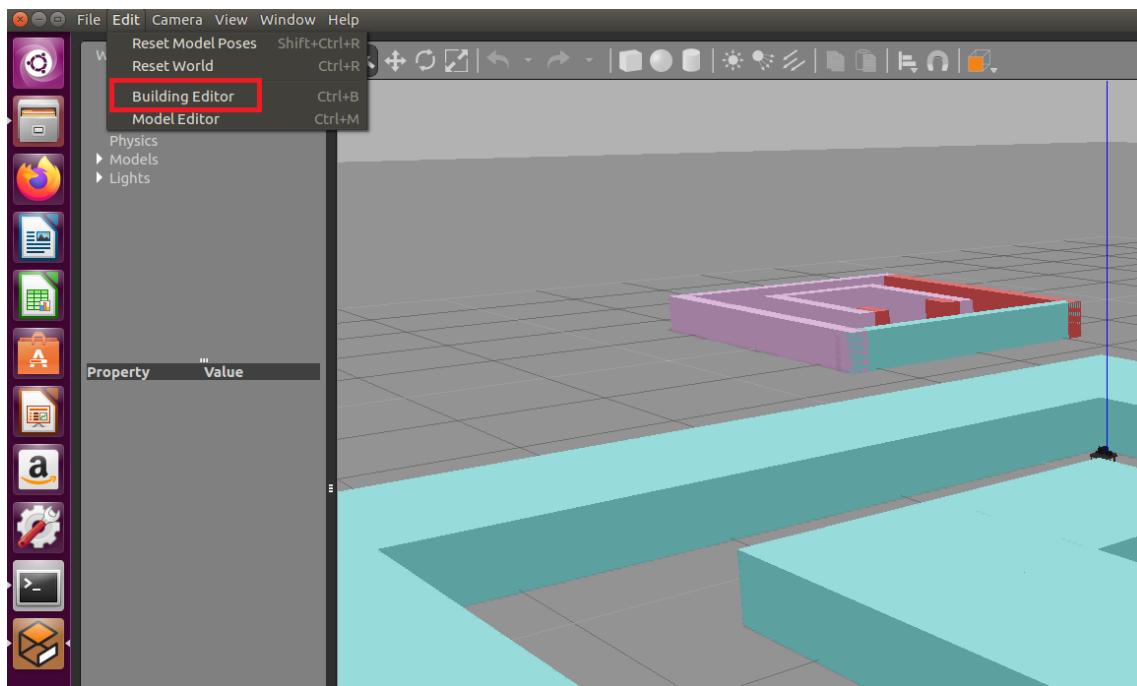


Figure 3 : Ouvrir le menu de construction de Gazebo.

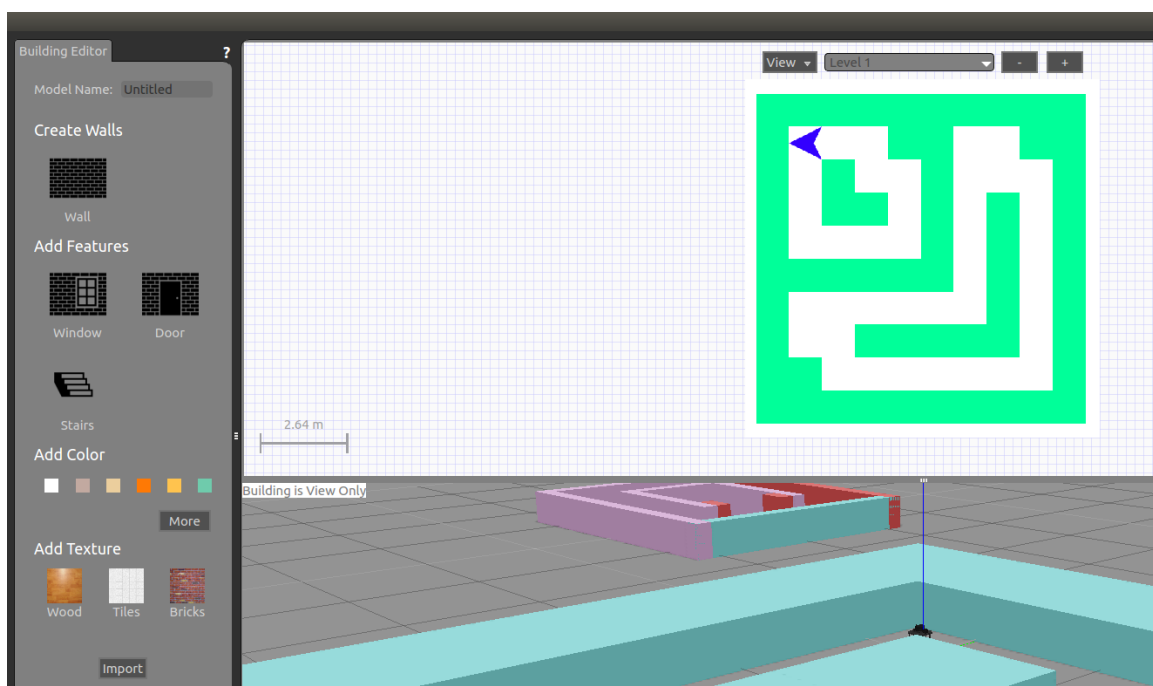


Figure 4 : Image de GAIT utilisée pour créer le labyrinthe.

Une fois ce menu ouvert, il est possible d'importer une image sur laquelle je pourrais me baser pour créer mon labyrinthe. Il faut sélectionner l'échelle qui convient puis disposer les murs par-dessus l'image. Il est possible de changer tout un tas de paramètres comme le point exact de départ et d'arrivée, la hauteur, l'épaisseur, la longueur, la couleur et même la texture. Les murs apparaissent au fur et à mesure dans la vue 3D.

Lorsque tous les murs ont été créés, le labyrinthe peut être enregistré. Il faut aller dans File > Save As. Il faut choisir le nom du labyrinthe, par exemple : **gait\_maze**. Attention, il faut obligatoirement que le fichier soit enregistré à l'emplacement suivant :

**~/gazebo/models/gait\_maze**. Je peux quitter le Building Editor.

Il est maintenant nécessaire d'ajouter le modèle du labyrinthe au fichier **turtlebot3\_my\_world.world** pour qu'il soit appelé systématiquement à chaque lancement. Pour ce faire, il va falloir éditer le fichier et ajouter quelques lignes de code. Il ne faut pas oublier d'enregistrer le fichier au format XML.

```
1 <sdf version="1.4">
2   <world name="default">
3
4     <scene>
5       <ambient>0.4 0.4 0.4 1</ambient>
6       <background>0.7 0.7 0.7 1</background>
7       <shadows>true</shadows>
8     </scene>
9
10    <!-- A global light source -->
11    <include>
12      <uri>model://sun</uri>
13    </include>
14
15    <!-- A ground plane -->
16    <include>
17      <uri>model://ground_plane</uri>
18    </include>
19
20    <!-- Gait2.0's maze -->
21    <include>
22      <uri>model://gait_maze2.0</uri>
23      <pose>5.066191 -3.515671 0 0 0 0</pose>
24    </include>
25
26    <!-- Doll's maze -->
27    <include>
28      <uri>model://doll_maze</uri>
29      <pose>-5 2 0 0 0 0</pose>
30    </include>
31
32    <physics type="ode">
33      <real_time_update_rate>1000.0</real_time_update_rate>
34      <max_step_size>0.001</max_step_size>
35      <real_time_factor>1</real_time_factor>
36      <ode>
37        <solver>
38          <type>quick</type>
39          <iters>150</iters>
40          <precon_iters>0</precon_iters>
41          <sor>1.400000</sor>
42          <use_dynamic_moi_rescaling>1</use_dynamic_moi_rescaling>
43        </solver>
44      </ode>
45    </physics>
46  </world>
47 </sdf>
```

Figure 5 : Contenu du fichier **turtlebot3\_my\_world.world**. Les lignes encadrées en rouge sont celles à ajouter. Elles comprennent le lien vers le labyrinthe ainsi que l'emplacement exact où il doit apparaître dans le monde.

Après avoir recompilé en exécutant la commande **catkin\_make** dans `~/catkin_ws`, le labyrinthe se retrouve attaché au monde et opérationnel à chaque lancement.

### 3. Actions du Turtlebot3

Cette partie traite des actions que le Turtlebot3 pourra exécuter pour interagir avec son environnement. Durant les cours du module de gestion de projet SCRUM du semestre 4, nous avons travaillé sur ces actions. Nous avons développé l'action **move\_forward**, qui permet au robot d'avancer en ligne droite sur une distance donnée, et avons commencé à travailler sur les rotations qui nous avait donné du fil à retordre. Au cours du stage, le développement se fait exclusivement en Python mais il faut préciser que j'ai utilisé deux versions différentes (Python2.7 & Python3.5).

Mon premier objectif était donc de créer deux différentes actions : **left\_rotate**, qui permet au robot de tourner à gauche sur un angle donné, et **right\_rotate**, qui permet au robot de tourner à droite sur un angle donné. J'ai donc commencé par créer deux fichiers distincts, un pour chaque action. Ces deux fichiers sont très similaires, la principale différence réside dans la valeur de la variable **clockwise** qui est négative dans l'un et positive dans l'autre. L'import de **time** permet de donner le temps nécessaire à l'exécution du programme.

Le problème ici était de réussir à faire une rotation d'exactly 90°. Le taux de rafraichissement étant peu élevé, je dois prendre une valeur d'**angle** supérieure à 90° qui soit pertinente sachant que la vitesse de rotation du robot valait 30. J'ai donc choisi 130 comme valeur d'**angle**. Le taux de réussite de cette fonction était d'environ 95%, il pouvait y avoir une rotation supérieure à 90° de temps en temps.



```

1 #!/usr/bin/env python
2
3 import time
4 import rospy
5 from geometry_msgs.msg import Twist
6 PI = 3.1415926535897
7
8 def right_rotate(speed=30, angle=130, clockwise=1):
9     rospy.init_node('rotate_turtlebot')
10
11     velocity_publisher = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
12     rate = rospy.Rate(1)
13     vel_msg = Twist()
14
15     #input
16     #print("Let's rotate")
17     #speed = 30                #degrees/sec
18     #angle = 130              #degrees
19     #clockwise = 1            #True or false
20
21     #from angles to radians
22     angular_speed = speed*2*PI/360
23     relative_angle = angle*2*PI/360
24
25     #we won't use this
26     vel_msg.linear.x=0
27     vel_msg.linear.y=0
28     vel_msg.linear.z=0
29     vel_msg.angular.x = 0
30     vel_msg.angular.y = 0
31
32     #clockwise or counterclockwise
33     if clockwise:
34         vel_msg.angular.z = -abs(angular_speed)
35     else:
36         vel_msg.angular.z = abs(angular_speed)
37
38     current_angle = 0
39
40     while (current_angle < relative_angle):
41         velocity_publisher.publish(vel_msg)
42         current_angle += angular_speed
43         print(current_angle, relative_angle)
44         rate.sleep()
45
46     vel_msg.angular.z = 0
47     velocity_publisher.publish(vel_msg)
48     return 0
49
50 if __name__ == '__main__':
51     try:
52         # time
53         start = time.time()
54
55         right_rotate()
56
57         # time
58         end = time.time()
59         print("time : ", end - start)
60
61     except rospy.ROSInterruptException:
62         pass

```

Librairie Python permettant de programmer sous ROS.

Messages contenant la vitesse linéaire et angulaire du robot, permet de donner au robot l'ordre de se déplacer au robot.

Définition du sens de rotation

Figure 6 : Contenu du fichier `right-rotate.py`.

Lorsque les rotations étaient prêtes, je suis passé à **move\_forward**. Comme je l'ai dit précédemment, cette action avait déjà été programmée par le BSN mais le comportement du programme ne me convenait plus. En reprenant la structure algorithmique utilisée pour les rotations et en l'adaptant au besoin d'avancer en ligne droite, le programme serait bien plus performant que celui que nous avons développé en cours. C'est donc ce que j'ai fait.

Bien plus tard, en commençant à utiliser GAIT, je me suis rendu compte qu'il fallait également que je développe les fonctions de détection du robot : **touch\_in\_front**, **touch\_right** et **touch\_left**. Chacune détectant la présence éventuelle d'un obstacle à l'avant, à droite ou à gauche.

J'ai donc développé six fichiers Python, pour les six actions que GAIT nécessite. Chaque fichier Python contient une action. Chaque action est développée sous la forme d'une fonction. Les fonctions ont des structures très similaires entre elles (voir Figure 4).

## 4. Quaternions

Les quaternions sont des nombres très utilisés en infographie, robotique, traitement du signal, mécanique spatiale, mécanique quantique géométrie différentielle... En dehors de ces domaines, ils ne sont quasi jamais utilisés.

Étant à propos de la robotique, j'ai été confronté à ces nombres durant ce stage. En effet, les actions de rotations à droite et à gauche échouant de temps à autre, l'utilisation des quaternions pour mieux contrôler la rotation semblait tout à fait adapté. N'ayant aucune connaissance sur ces nombres, j'ai dû en apprendre un maximum à l'aide de vidéos explicatives, de tutoriels, etc...

Pour faire simple, dans une simulation informatique, le nombre de dimensions n'est pas de trois mais de quatre dans le but de correspondre parfaitement à la réalité. L'utilisation des quaternions permet de repasser dans un système à trois dimensions, c'est la raison pour laquelle ils sont autant utilisés. Cependant, leur prise en main n'est pas aisée.

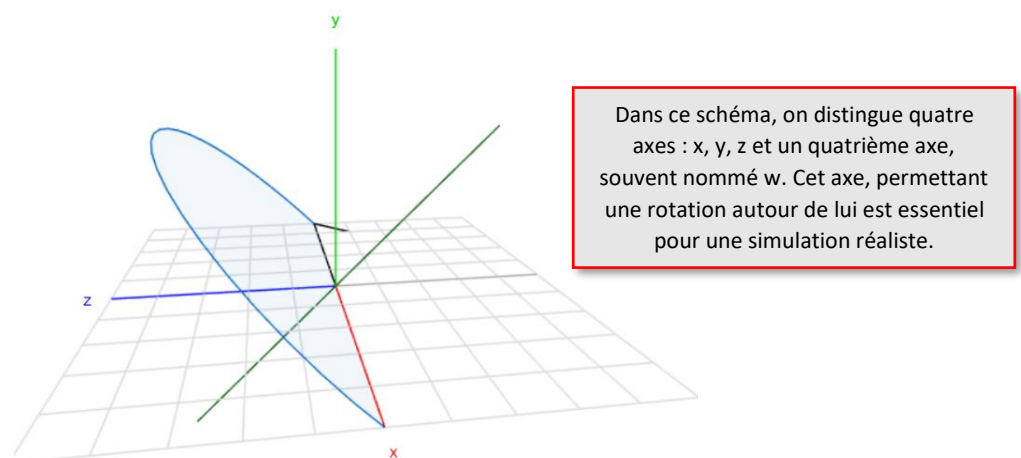


Figure 7 : Schéma explicatif des axes dans une simulation.

Après avoir réuni les informations nécessaires, j'ai commencé par créer différents petits programmes utilisant les quaternions pour m'y familiariser. J'ai ensuite travaillé sur les rotations en utilisant les quaternions pendant plusieurs jours mais l'exercice n'était vraiment pas facile. Il m'aurait fallu bien plus de temps pour réussir à créer des rotations fonctionnelles avec les quaternions. Un temps que nous n'avons pas pris pour pouvoir avancer le plus possible sur le projet.

```
1 #!/usr/bin/env python
2
3 import rospy
4 #from nav_msgs.msg import Odometry
5 from sensor_msgs.msg import Imu
6 from tf.transformations import euler_from_quaternion, quaternion_from_euler
7
8 roll = pitch = yaw = 0.0
9
10 def get_rotation(msg):
11     global roll, pitch, yaw
12     #print(msg.orientation)
13     orientation_q = msg.orientation
14     orientation_list = [orientation_q.x, orientation_q.y, orientation_q.z, orientation_q.w]
15     #print(orientation_list)
16     (roll, pitch, yaw) = euler_from_quaternion(orientation_list)
17     print "roll : ", roll
18     print "pitch: ", pitch
19     print "yaw : ", yaw
20
21 rospy.init_node('quaternion_to_euler')
22
23 sub = rospy.Subscriber('/imu', Imu, get_rotation)
24 rate = rospy.Rate(1)
25
26 while not rospy.is_shutdown():
27     quat = quaternion_from_euler(roll, pitch, yaw)
28     print quat
29     rate.sleep()
```

Figure 8 : Programme permettant de convertir les valeurs des axes en quaternions et vice-versa.

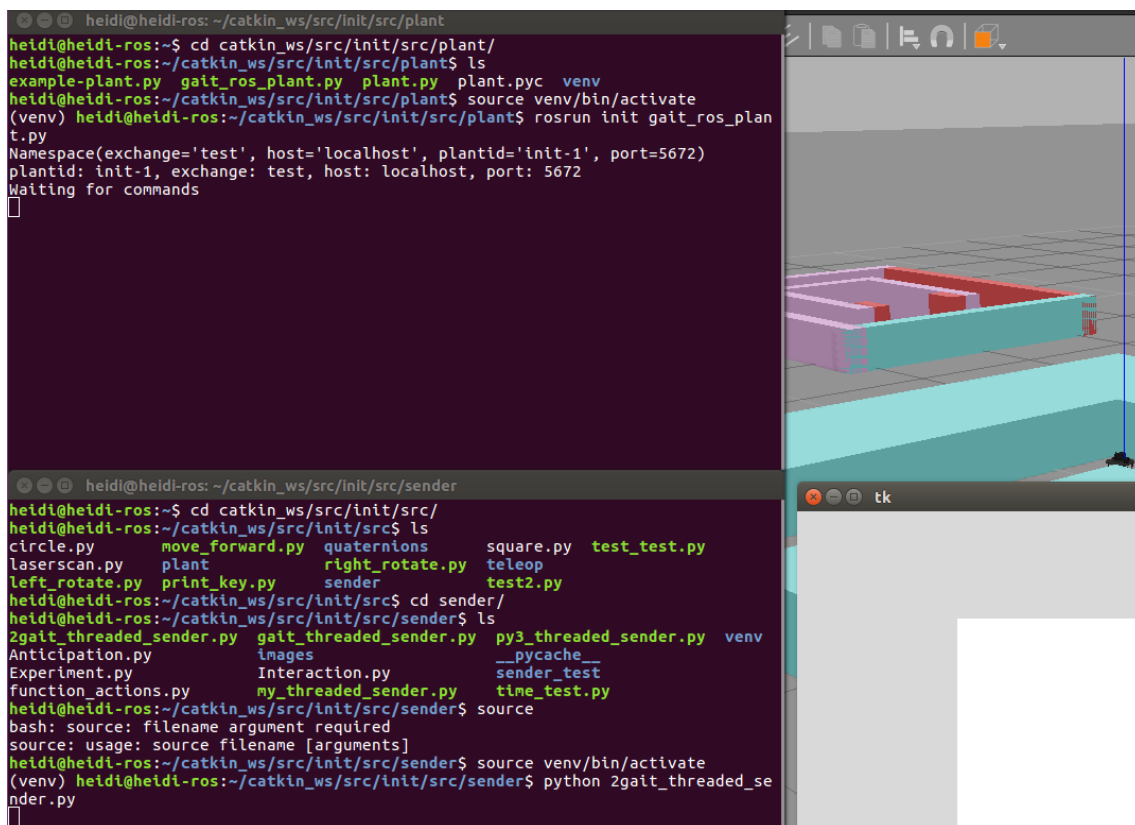
## 5. GAIT & RabbitMQ

Cette partie traite de GAIT et de son fonctionnement avec RabbitMQ. RabbitMQ est utilisé dans tous les projets de DOLL Inc. pour la communication, il est donc normal qu'il soit implémenté dans le projet INIT. Dans ce projet, RabbitMQ a servi à faire le lien entre ROS (toute la partie de simulation du robot) et GAIT (algorithme d'intelligence artificielle utilisé).

J'ai donc dans un premier temps, récupéré et modifié le code de DOLL concernant RabbitMQ. Ce code avait été précédemment utilisé pour un projet utilisant un rover. Dans

un deuxième temps, j'ai ajouté le code de GAIT en le modifiant pour qu'il fonctionne avec RabbitMQ, sans rien modifier concernant l'algorithme en lui-même. Et dans un troisième temps, j'ai implémenté les fonctions que j'avais programmées plus tôt. Le tout a donné deux énormes fichiers de code : **gait\_ros\_plant.py** (400 lignes) & **2gait\_threaded\_sender.py** (550 lignes). Le premier est en Python2.7 et le deuxième en Python3.5. La raison est que ROS est seulement compatible avec Python2 (la dernière version de ROS prend désormais en charge Python3), alors que GAIT, lui, est développé en Python3.

Pour pouvoir lancer ces fichiers, il est donc nécessaire d'avoir les deux versions de Python. Il faut également installer virtualenv pour pouvoir les utiliser en même temps sans trop de difficultés.



```

heidi@heidi-ros: ~/catkin_ws/src/init/src/plant
heidi@heidi-ros:~$ cd catkin_ws/src/init/src/plant/
heidi@heidi-ros:~/catkin_ws/src/init/src/plant$ ls
example-plant.py  gait_ros_plant.py  plant.py  plant.pyc  venv
heidi@heidi-ros:~/catkin_ws/src/init/src/plant$ source venv/bin/activate
(venv) heidi@heidi-ros:~/catkin_ws/src/init/src/plant$ roslaunch init gait_ros_plant.py
Namespace(exchange='test', host='localhost', plantid='init-1', port=5672)
plantid: init-1, exchange: test, host: localhost, port: 5672
Waiting for commands
[]

heidi@heidi-ros: ~/catkin_ws/src/init/src/sender
heidi@heidi-ros:~$ cd catkin_ws/src/init/src/sender/
heidi@heidi-ros:~/catkin_ws/src/init/src/sender$ ls
circle.py  move_forward.py  quaternions  square.py  test_test.py
laserscan.py  plant  right_rotate.py  teleop
left_rotate.py  print_key.py  sender  test2.py
2gait_threaded_sender.py  gait_threaded_sender.py  py3_threaded_sender.py  venv
Anticipation.py  images  __pycache__
Experiment.py  Interaction.py  sender_test
function_actions.py  my_threaded_sender.py  time_test.py
heidi@heidi-ros:~/catkin_ws/src/init/src/sender$ source
bash: source: filename argument required
source: usage: source filename [arguments]
heidi@heidi-ros:~/catkin_ws/src/init/src/sender$ source venv/bin/activate
(venv) heidi@heidi-ros:~/catkin_ws/src/init/src/sender$ python 2gait_threaded_sender.py

```

Figure 9 : Commandes pour le lancement de GAIT.

Je vais expliciter la manière dont ces fichiers fonctionnent. Le fonctionnement est le suivant :

- 1) Dans le programme **2gait\_threaded\_sender.py** se trouve tout le code de GAIT développé par Jianyong XUE que j'ai refactorisé. Au lancement, GAIT décide d'une action à entreprendre. Il a le choix parmi les six actions que j'ai programmé

précédemment. Lorsque le choix est fait, l'ordre est envoyé d'exécuter l'action via RabbitMQ.

Il faut savoir que RabbitMQ fonctionne normalement avec un programme appelé **producer** qui envoie les données et un programme appelé **consumer** qui les reçoit.

Au lieu de créer deux programmes différents, j'ai utilisé deux threads différents. C'est ce qu'on appelle le multithreading. GAIT ordonne donc l'exécution d'une action, **myProducerThread** envoie l'information, **myConsumerThread** la reçoit.

- 2) Lorsque l'information a été reçue, elle est traitée. Cette information est une valeur qui correspond à une action. Un tableau associe à chaque valeur un nom de fonction dans la class **Rmq**. Cette class contient toutes les fonctions nécessaires au traitement des données reçues. Parmi ces fonctions, on retrouve six fonctions correspondant aux différentes actions possibles. Ces actions, que j'ai programmé auparavant, se trouvent toutes dans la class **Init**.

```
417 class myConsumerThread (threading.Thread):
418     def __init__(self):
419         threading.Thread.__init__(self)
420     def run(self):
421         print("Starting myConsumerThread")
422         self.connection = pika.BlockingConnection(pika.ConnectionParameters('localhost', 5672))
423         self.channel = self.connection.channel()
424         self.channel.queue_declare(queue='test')
425         self.channel.exchange_declare(exchange='test', exchange_type='topic')
426         self.result = self.channel.queue_declare(queue='observations', exclusive=False)
427         self.queue_name = self.result.method.queue
428         self.channel.queue_bind(exchange='test', queue=self.queue_name)
429         self.channel.basic_consume(consumer_callback, queue='observations', no_ack=True)
430
431         with lock:
432             print(' [*] Waiting for messages. To exit press CTRL+C')
433
434         self.channel.start_consuming()
435
436 class myProducerThread (threading.Thread):
437     def __init__(self, canvas, count):
438         threading.Thread.__init__(self)
439         self.canvas = canvas
440         self.count = count
441     def run(self):
442         print("Starting myProducerThread")
443         self.connection = pika.BlockingConnection(pika.ConnectionParameters('localhost', 5672))
444         self.channel = self.connection.channel()
445         self.channel.exchange_declare(exchange='test', exchange_type='topic')
446
447         global startState, xValue, yValue, enactedInteraction, CANVAS_HEIGHT, INTERVAL,
isIntendedInteractionComposite
448         intendedInteractionWidth = 0
449         if startState:
```

Figure 10 : Déclaration des threads.

```

271 def move_forward(self, msg):
272     self.plant.started(msg)
273     myobservations = [self.plant.make_observation('move_forward_observation',
init.move_forward())]
274     print('Publishing result observation: ', myobservations)
275     self.plant.observations(msg, myobservations)
276     self.plant.finished(msg)
277
278 def left_rotate(self, msg):
279     self.plant.started(msg)
280     myobservations = [self.plant.make_observation('left_rotate_observation', init.left_rotate
281 ())]
282     print('Publishing result observation: ', myobservations)
283     self.plant.observations(msg, myobservations)
284     self.plant.finished(msg)
285
286 def right_rotate(self, msg):
287     self.plant.started(msg)
288     myobservations = [self.plant.make_observation('right_rotate_observation',
init.right_rotate())]
289     print('Publishing result observation: ', myobservations)
290     self.plant.observations(msg, myobservations)
291     self.plant.finished(msg)
292
293 def touch_in_front(self, msg):
294     self.plant.started(msg)
295     myobservations = [self.plant.make_observation('touch_in_front_observation',
init.touch_in_front())]
296     print('Publishing result observation: ', myobservations)
297     self.plant.observations(msg, myobservations)
298     self.plant.finished(msg)
299
300 def touch_right(self, msg):
301     self.plant.started(msg)
302     myobservations = [self.plant.make_observation('touch_right_observation', init.touch_right
303 ())]
304     print('Publishing result observation: ', myobservations)
305     self.plant.observations(msg, myobservations)
306     self.plant.finished(msg)
307
308 def touch_left(self, msg):
309     self.plant.started(msg)
310     myobservations = [self.plant.make_observation('touch_left_observation', init.touch_left
311 ())]
312     print('Publishing result observation: ', myobservations)
313     self.plant.observations(msg, myobservations)
314     self.plant.finished(msg)

```

Figure 11 : Les six fonctions de la class Rmq renvoyant aux fonctions de la class Init.

```

93 class Init:
94     def __init__(self, to_rmq, from_rmq):
95         self.to_rmq = to_rmq
96         self.from_rmq = from_rmq
97         self.laser_callback = rospy.Subscriber('/scan', LaserScan, self.laser_callback)
98         self.move_forward_action = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
99         self.left_rotate_action = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
100         self.right_rotate_action = rospy.Publisher('/cmd_vel', Twist, queue_size=1)
101
102     def laser_callback(self, msg):
103         global scan_in_front
104         global scan_right
105         global scan_left
106         scan_in_front = msg.ranges[0]
107         scan_right = msg.ranges[-90]
108         scan_left = msg.ranges[90]
109
110     def touch_in_front(self):
111         table = [0]
112         table.append(scan_in_front)
113         if scan_in_front < 0.7:
114             table[0] = 1
115         print table
116         return table
117
118     def touch_right(self):
119         table = [0]
120         table.append(scan_right)
121         if scan_right < 0.7:
122             table[0] = 1
123         print table
124         return table
125
126     def touch_left(self):
127         table = [0]
128         table.append(scan_left)
129         if scan_left < 0.7:
130             table[0] = 1
131         print table
132         return table
133
134     def move_forward(self):
135         table = [0]
136
137         rate = rospy.Rate(1)
138         vel_msg = Twist()

```

Figure 12 : Quelques fonctions de la class Init.

- 3) La class **RosActionClient** permet d'avoir des informations sur le déroulement de l'action de la class **Init**, sur son déclenchement, son succès, ou encore son échec. C'est une class qui était déjà présente dans le code que j'ai reçu, j'ai simplement appris à l'utiliser.

Les fonctions dans la class **Init** sont liées à ROS, elles agissent directement sur le Turtlebot3 dans la simulation Gazebo. Un nœud **roscore** doit évidemment être actif pour que tout fonctionne. Lorsque l'action est appelée par GAIT, elle est exécutée, le robot interagit avec son environnement et un feedback est récupéré. Ce feedback est un tableau de deux valeurs. La première est un booléen indiquant la présence d'un obstacle, la deuxième est la distance à laquelle se trouve cet obstacle. Si aucun obstacle n'est détecté cette deuxième valeur est nulle. Cette valeur n'est d'ailleurs pas utilisée par GAIT.

- 4) Lorsque l'action a été exécutée, GAIT va recevoir le feedback correspondant à cette action. Ce feedback est aussi envoyé depuis ROS à GAIT via RabbitMQ, elles se trouvent dans la variable **myobservations** définie dans la class **Rmq**.

GAIT peut alors considérer ce feedback et décider de la prochaine action à entreprendre en fonction de ce dernier et des précédents. Au fur et à mesure, cela permet de créer des interactions plus ou moins poussées (plusieurs actions exécutées les unes à la suite des autres) et un comportement que l'on peut qualifier d'intelligent.

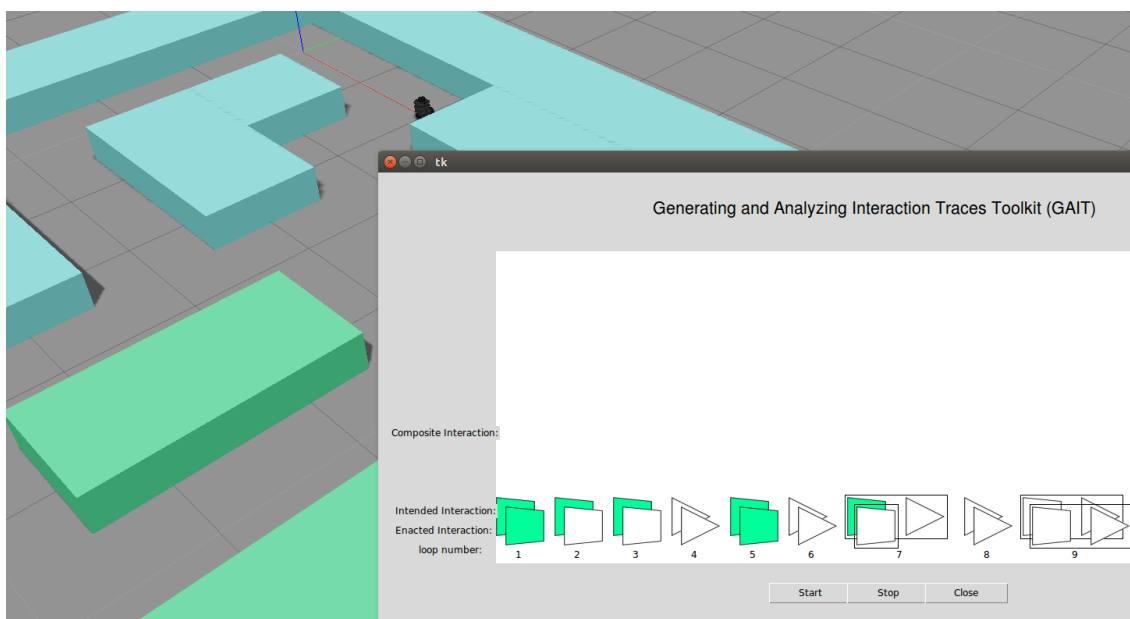


Figure 13 : GAIT en cours de fonctionnement. Les formes sont les traces des interactions effectuées ou en cours. Le robot se déplace à l'arrière-plan.

# CONCLUSION

Pour conclure, j'ai effectué mon stage de fin de deuxième année en tant que stagiaire en développement logiciel au sein de l'entreprise DOLL Inc. Malgré les circonstances particulières dans lequel ce stage s'est déroulé, j'ai pu mettre en pratique les connaissances acquises tout au long de ma formation. J'ai, notamment, développé avec Python mais j'ai également utilisé mes notions en Scrum ainsi que mes compétences avec des outils tels que Git. Dans le monde du travail, je me suis confronté aux difficultés que l'on rencontre quotidiennement lorsque l'on mène un projet de développement dans le secteur informatique/robotique.

Ce stage a été très enrichissant pour moi, il m'a permis de découvrir le domaine de la recherche et le manière dont celle-ci fonctionne aux États-Unis. Il m'a permis de participer concrètement à l'un des projets menés par DOLL Inc. au travers de mes différentes missions de développement et d'administration des systèmes. Ce stage m'a permis de comprendre que la robotique est un domaine qui m'intéresse particulièrement. J'ai également compris que les opportunités à l'international sont très intéressantes car au-delà de l'aspect professionnel, elles permettent très certainement de « grandir » personnellement.

Fort de cette expérience, j'aimerais donc beaucoup travailler dans la robotique à l'échelle internationale. Cependant, je me suis découvert un certain intérêt pour l'intelligence artificielle, et pour la recherche dans ce domaine. Sachant que la recherche fonctionne différemment en fonction de chaque pays, après les États-Unis, je suis curieux de découvrir son fonctionnement en France.

Je tenais à ajouter que j'ai mis en ligne une vidéo expliquant le fonctionnement de GAIT. Cette vidéo est en anglais, vous trouverez le lien ci-dessous.

Lien vers vidéo de présentation : [https://www.youtube.com/watch?v=YHrBSU\\_D8ro](https://www.youtube.com/watch?v=YHrBSU_D8ro)



# ANNEXES

J'ai décidé de ne pas modifier les mises en forme des documents qui vont suivre. Celles-ci ne correspondent donc pas tout à fait à celle que j'utilise depuis le début du rapport.

1. Portage de GAIT Java sur Linux (23-42)
2. Installation de VirtualBox, Ubuntu et ROS (43-45)
3. Mise en place de la simulation Gazebo sous ROS (46-48)

# GAIT on Linux

## 1. Intro








This documentation will help you to install GAIT on Linux. It is designed for Ubuntu 16.04 / 18.04 / 20.04.

This work is in progress.

## 2. Java Installation

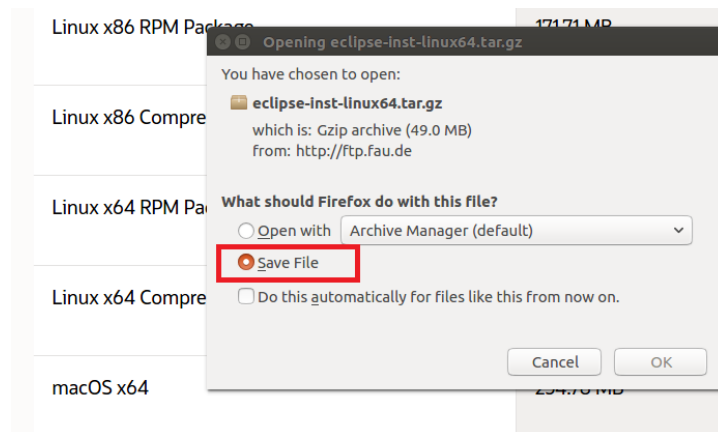
GAIT is written in Java. First of all, we need to install Java Development Kit. The one we have to install is Java SE Development Kit 8u251. You can find it at this address:

<https://www.oracle.com/java/technologies/javase-jdk8-downloads.html>

Java SE Development Kit 8u251		
This software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE		
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.87 MB	 <a href="#">jdk-8u251-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM 64 Hard Float ABI	69.77 MB	 <a href="#">jdk-8u251-linux-arm64-vfp-hflt.tar.gz</a>
Linux x86 RPM Package	171.71 MB	 <a href="#">jdk-8u251-linux-i586.rpm</a>
Linux x86 Compressed Archive	186.6 MB	 <a href="#">jdk-8u251-linux-i586.tar.gz</a>
Linux x64 RPM Package	171.16 MB	 <a href="#">jdk-8u251-linux-x64.rpm</a>
Linux x64 Compressed Archive	186.09 MB	 <a href="#">jdk-8u251-linux-x64.tar.gz</a>
macOS x64	254.78 MB	 <a href="#">jdk-8u251-macosx-x64.dmg</a>

Select the Linux x64 Compressed Archive.

**WARNING!** You need to login with your Oracle account or to create it.



By default, system will select “Open with...”. Please select “Save File”.

When the file is downloaded, open a terminal (CTRL + ALT + T) and go to your downloads folder. You will find your JDK archive. You can extract it.

Commands:

Go to Downloads folder →

```
$ cd Downloads/
```

Check your JDK archive →

```
$ ls
```

Extract it →

```
$ tar -zxvf jdk-8u251-linux-x64.tar.gz
```

Now, you have to put your extracted folder in the right place. For that, please check that a file named “java” is in your extracted JDK folder. If not, (and probably not) you need to create it.

Commands:

Check your extracted JDK folder →

```
$ ls
```

Got to the right place →

```
$ cd /usr/lib
```

Check whether a "java" folder already exists here →

```
$ ls
```

**IF NOT:** Create it →

```
$ sudo mkdir java
```

Return to your Downloads folder →

```
$ cd ~/Downloads
```

Move your extracted folder in the folder you just created →

```
$ sudo mv jdk1.8.0_251 /usr/lib/java/
```

Go into the "bin" folder of your extracted file →

```
$ cd /usr/lib/java/jdk1.8.0_251/bin
```

Generally, you will find a lot of files in here. Please verify that a "java" file and a "javac" file are here.

```
heidi@heidi-ros:/usr/lib/java/jdk1.8.0_251/bin$ ls
appletviewer  javac      javaws     jinfo      jsadbugd    orbd        serialver
ControlPanel  javadoc    jcmd       jjs        jstack      pack200     servertool
extcheck      javafxpackager jconsole   jmap       jstat       policytool  tnameserv
idlj          javah      jcontrol   jmc        jstatd      rmic        unpack200
jar           javap      jdb        jmc.ini    jvisualvm   rmid        wsgen
jarsigner     javapackager jdeps      jps        keytool     rmiregistry wsimport
java          java-rmi.cgi jhat       jrunscript native2ascii schemagen   xjc
heidi@heidi-ros:/usr/lib/java/jdk1.8.0_251/bin$
```

### 3. JDK Setting Up

Now, you will be setting up java for updates and packages management.

Go back in your jdk folder →

```
$ cd ..
```

Write these three commands one by one →

```
$ sudo update-alternatives --install "/usr/bin/java" "java"  
"/usr/lib/java/jdk1.8.0_251/bin/java" 1
```

```
$ sudo update-alternatives --install "/usr/bin/javac" "javac"  
"/usr/lib/java/jdk1.8.0_251/bin/javac" 1
```

```
$ sudo update-alternatives --install "/usr/bin/javaws" "javaws"  
"/usr/lib/java/jdk1.8.0_251/bin/javaws" 1
```

**WARNING!** Don't forget to replace `jdk1.8.0_251` by the version you downloaded.

Now, you need to edit your `.bashrc` file and to add some lines.

Let's return to your home →

```
$ cd ~
```

Run `gedit` to edit your `.bashrc` file →

```
$ gedit .bashrc
```

Add the following lines to the end of your `.bashrc` file:

```
#JAVA HOME directory setup  
export JAVA_HOME=/usr/lib/java/jdk1.8.0_251  
export PATH="$PATH:$JAVA_HOME/bin"
```

**WARNING!** Don't forget to replace `jdk1.8.0_251` by the version you downloaded.

To verify that our installation works, we will test it with the following command:

First, close and reopen your terminal

Write →

```
$ java
```

```

heid@heid-ros:~$ java
Usage: java [-options] class [args...]
           (to execute a class)
   or java [-options] -jar jarfile [args...]
           (to execute a jar file)
where options include:
    -d32          use a 32-bit data model if available
    -d64          use a 64-bit data model if available
    -server       to select the "server" VM
                  The default VM is server,
                  because you are running on a server-class machine.

    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                  A : separated list of directories, JAR archives,
                  and ZIP archives to search for class files.
    -D<name>=<value>
                  set a system property
    -verbose:[class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -version:<value>
                  Warning: this feature is deprecated and will be removed
                  in a future release.
                  require the specified version to run
    -showversion  print product version and continue
    -jre-restrict-search | -no-jre-restrict-search
                  Warning: this feature is deprecated and will be removed
                  in a future release.
                  include/exclude user private JREs in the version search
    -? -help     print this help message
    -X           print help on non-standard options
    -ea[:<packagename>...]:<classname>]
    -enableassertions[:<packagename>...]:<classname>]
                  enable assertions with specified granularity
    -da[:<packagename>...]:<classname>]
    -disableassertions[:<packagename>...]:<classname>]
                  disable assertions with specified granularity
    -esa | -enablesystemassertions
                  enable system assertions
    -dsa | -disablesystemassertions
                  disable system assertions
    -agentlib:<libname>[=<options>]
                  load native agent library <libname>, e.g. -agentlib:hprof
                  see also, -agentlib:jdwp=help and -agentlib:hprof=help
    -agentpath:<pathname>[=<options>]
                  load native agent library by full pathname
    -javaagent:<jarpath>[=<options>]
                  load Java programming language agent, see java.lang.instrument
    -splash:<imagepath>
                  show splash screen with specified image

```

You should have that in your terminal.

Write →

\$ javac

You should have a similar thing above

Write →

\$ java -version

```

heid@heid-ros:~$ java -version
java version "1.8.0_251"
Java(TM) SE Runtime Environment (build 1.8.0_251-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.251-b08, mixed mode)

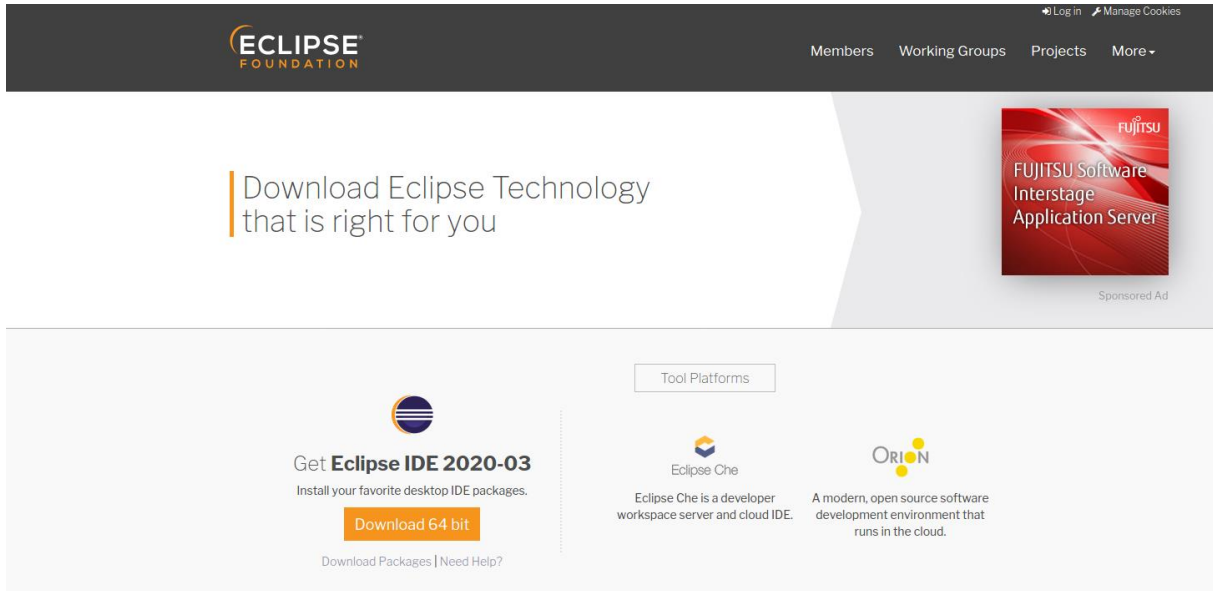
```

You installed Java for Linux!

## 4. Eclipse IDE Installation

First, you need to download the archive file. To do this, go to this address :

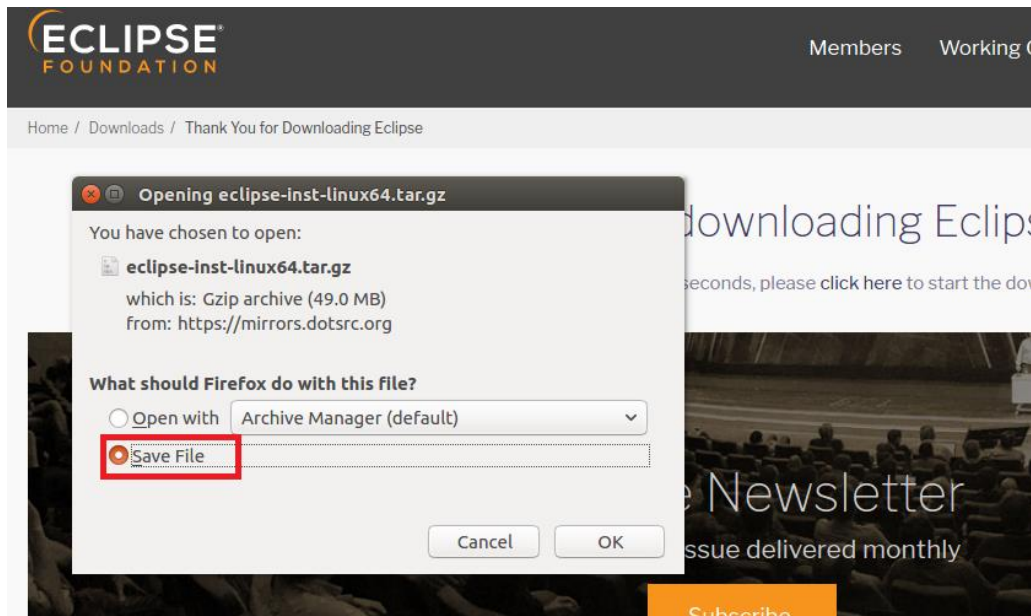
<https://www.eclipse.org/downloads/>



Click on the download button.

This will take you to a new page, click again on the download button.

A pop-up window will appear to ask you what to do with the file.



Be careful to select "Save file" and click on the "OK" button.

From here, we will follow a similar procedure to what we did above for the Java installation.

So, you can open a new terminal (CTRL+ALT+T).

Go to your Downloads folder →

```
$ cd Downloads/
```

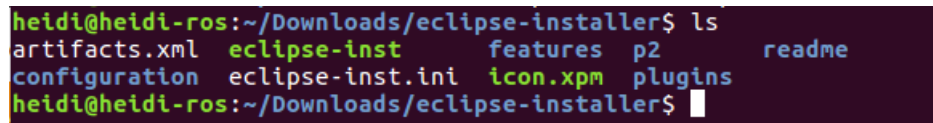
Extract your eclipse archive →

```
$ tar -zxvf eclipse-inst-linux64.tar.gz
```

If you check your folders, you can see that you have now a file named “eclipse-installer”.

Go into this file →

```
$ cd eclipse-installer
```

A terminal window screenshot showing the command 'ls' being executed in the directory ~/Downloads/eclipse-installer. The output lists several files and subdirectories: artifacts.xml, eclipse-inst, features, p2, readme, configuration, eclipse-inst.ini, icon.xpm, and plugins. The prompt is heidi@heidi-ros:~/Downloads/eclipse-installer\$.

```
heidi@heidi-ros:~/Downloads/eclipse-installer$ ls
artifacts.xml  eclipse-inst  features  p2          readme
configuration  eclipse-inst.ini  icon.xpm  plugins
```

You should have these files.

**WARNING!** Before continue, be sure to be done with the JDK Setting Up.

Run the “eclipse-inst” file →

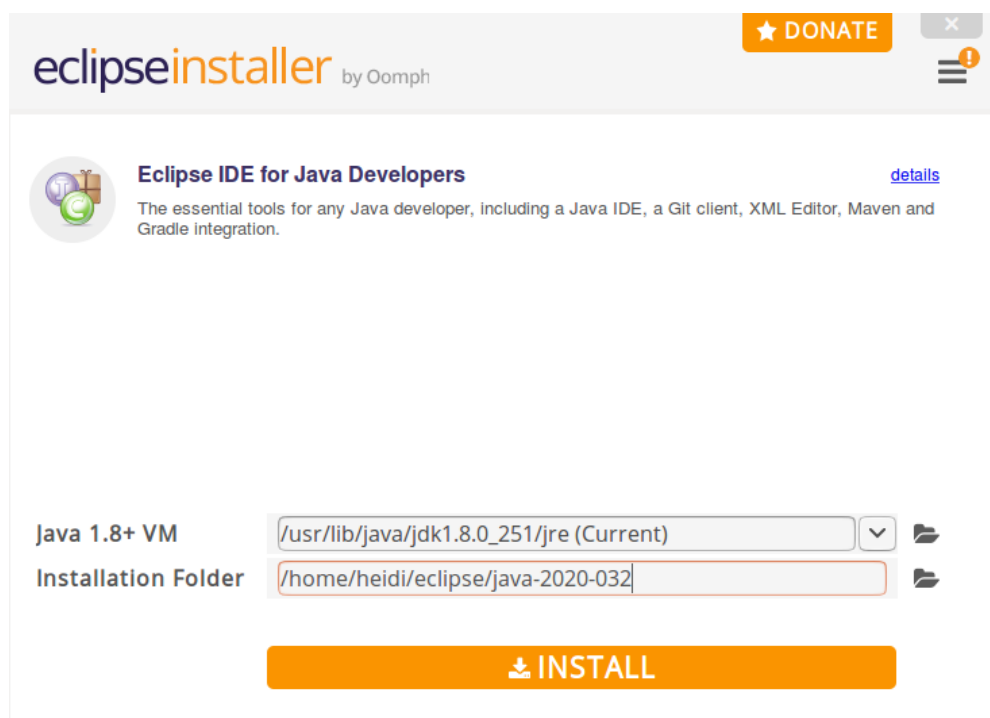
```
$ ./eclipse-inst
```



The following window will appear:



Select the first option.

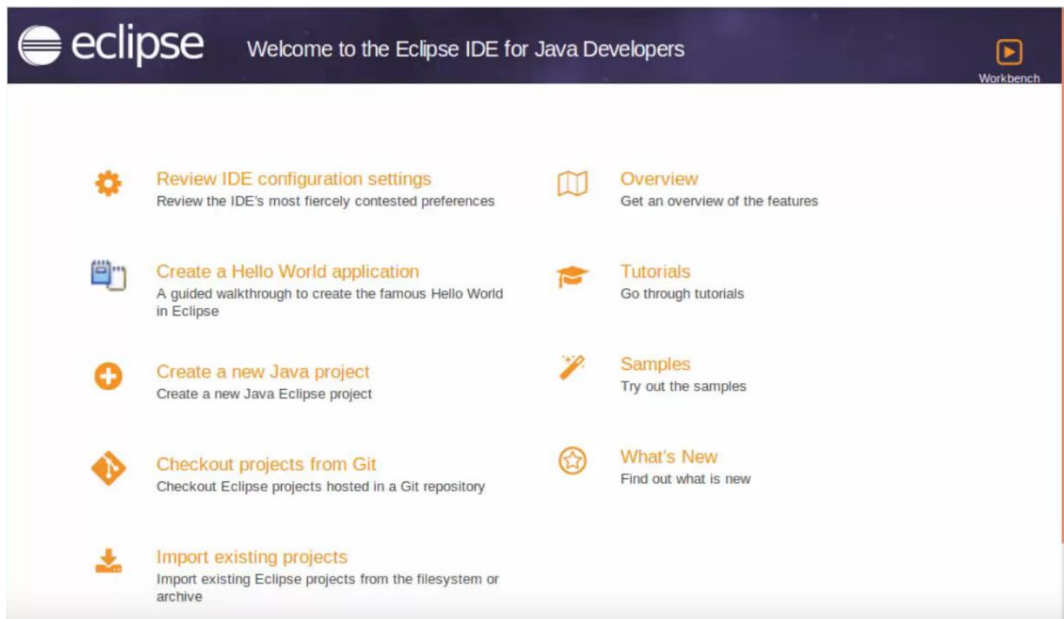


You can leave the installation folders by default. Click on the "INSTALL" button.

Follow the installation instructions:

- accept licenses
- select and accept certificates

You can now click on the “LAUNCH” button and eclipse will start. The first time the eclipse IDE starts, it will ask you to define the default workspace. You can leave it. You can also check the button to always use these settings by default.



You are now on the welcome page of the eclipse IDE. You can close it.

Eclipse is installed.

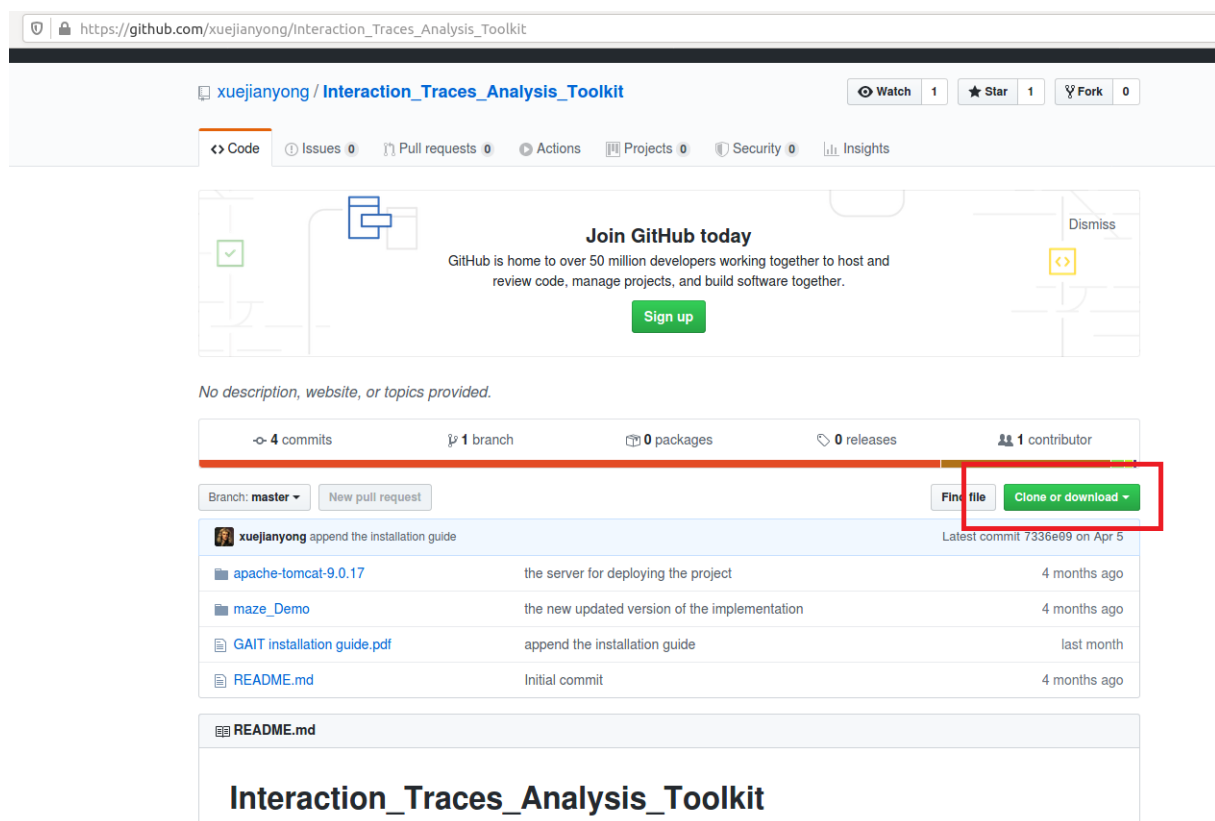
## 5. GAIT Installation

The first step of this part is to import GAIT's repository from Github. To do this, you can download the archive file from this address:

[https://github.com/xuejianyong/Interaction\\_Traces\\_Analysis\\_Toolkit](https://github.com/xuejianyong/Interaction_Traces_Analysis_Toolkit)

Or you can clone the repository in local with git. It's a better way to do it. If you prefer this way, skip to the next page.

Archive solution:



Click the "Clone or download" button and then "Download ZIP".

Open a new terminal (CTRL+ALT+T) and go to your Downloads file →

```
$ cd Downloads/
```

Extract the GAIT file →

```
$ tar -zcvf Interaction_Traces_Analysis_Toolkit-master.zip
```

### Git Solution:

**WARNING!** You need to install git packages to continue.

Open a new terminal (CTRL+ALT+T) and write this command →

```
$ git clone  
https://github.com/xuejianyong/Interaction Traces Analysis Toolkit.git
```

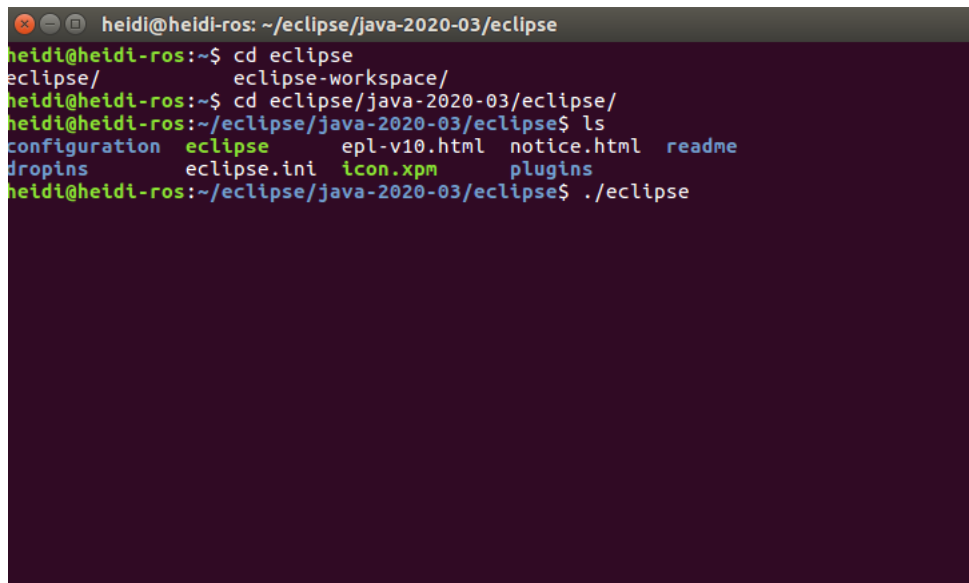
The repository is cloned.

## 6. GAIT Importation in Eclipse

Go into the folder where you installed Eclipse and run eclipse →

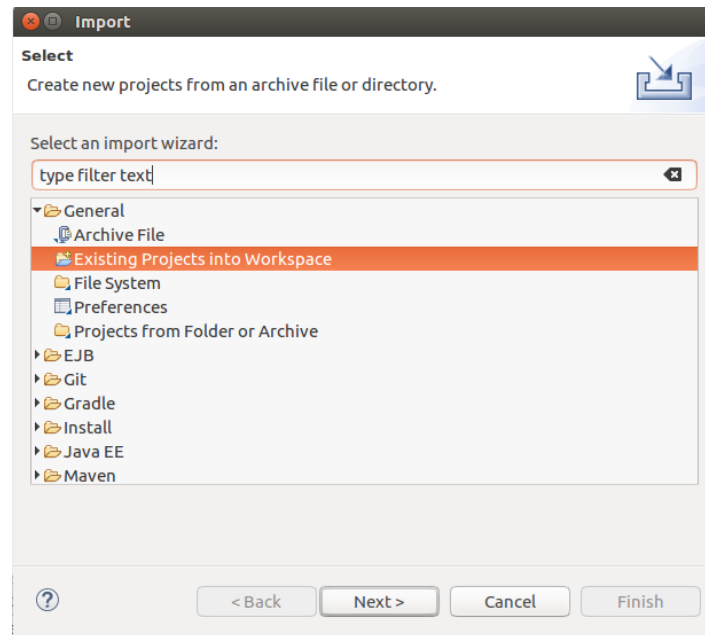
```
$ cd ~/eclipse/java-2020-03/eclipse
```

```
$ ./eclipse
```

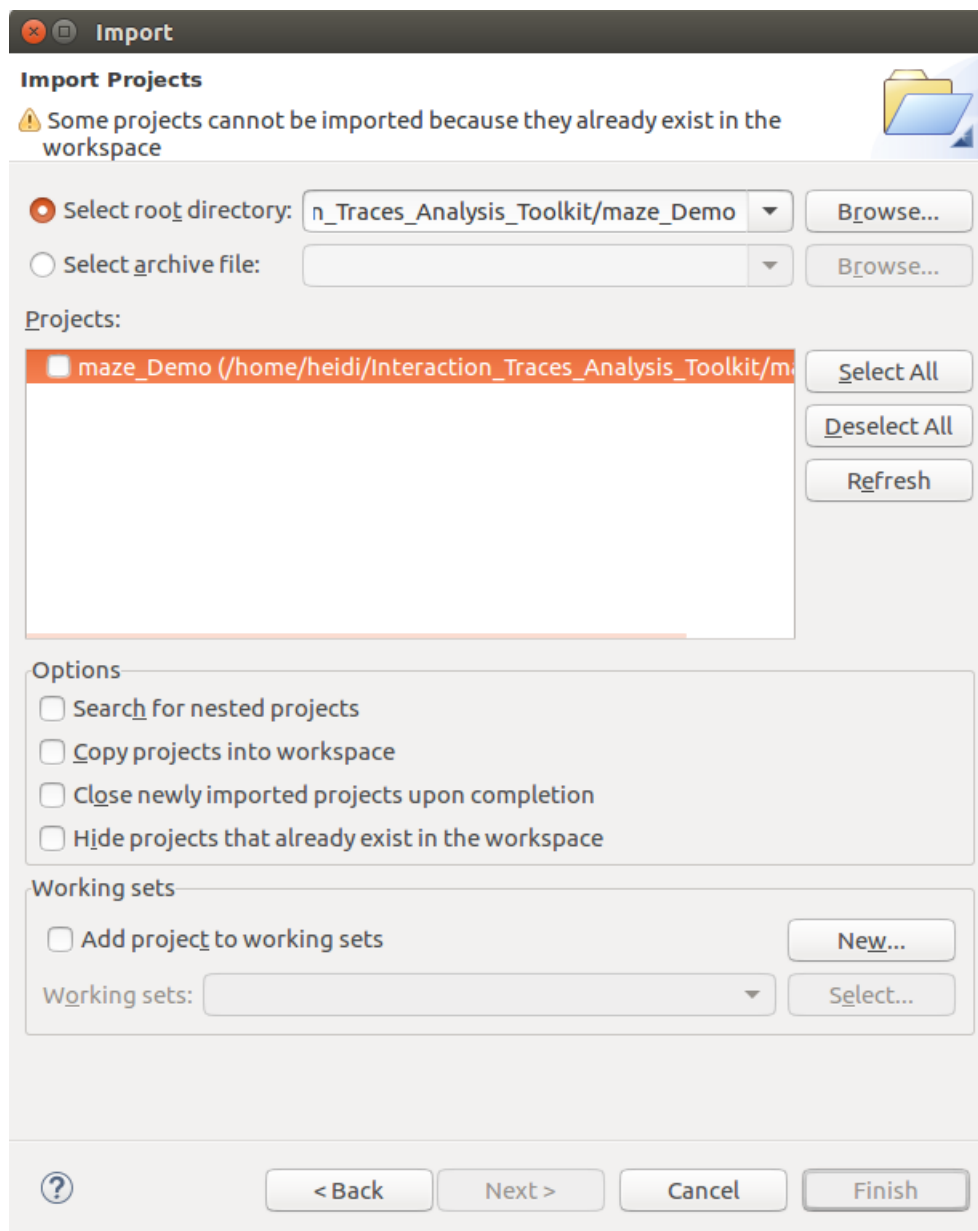
A terminal window with a dark purple background. The title bar shows a window icon, a close button, and the text 'heidi@heidi-ros: ~/eclipse/java-2020-03/eclipse'. The terminal content shows the following commands and output:

```
heidi@heidi-ros:~$ cd eclipse  
eclipse/          eclipse-workspace/  
heidi@heidi-ros:~$ cd eclipse/java-2020-03/eclipse/  
heidi@heidi-ros:~/eclipse/java-2020-03/eclipse$ ls  
configuration  eclipse      epl-v10.html  notice.html  readme  
dropins        eclipse.ini  icon.xpm      plugins  
heidi@heidi-ros:~/eclipse/java-2020-03/eclipse$ ./eclipse
```

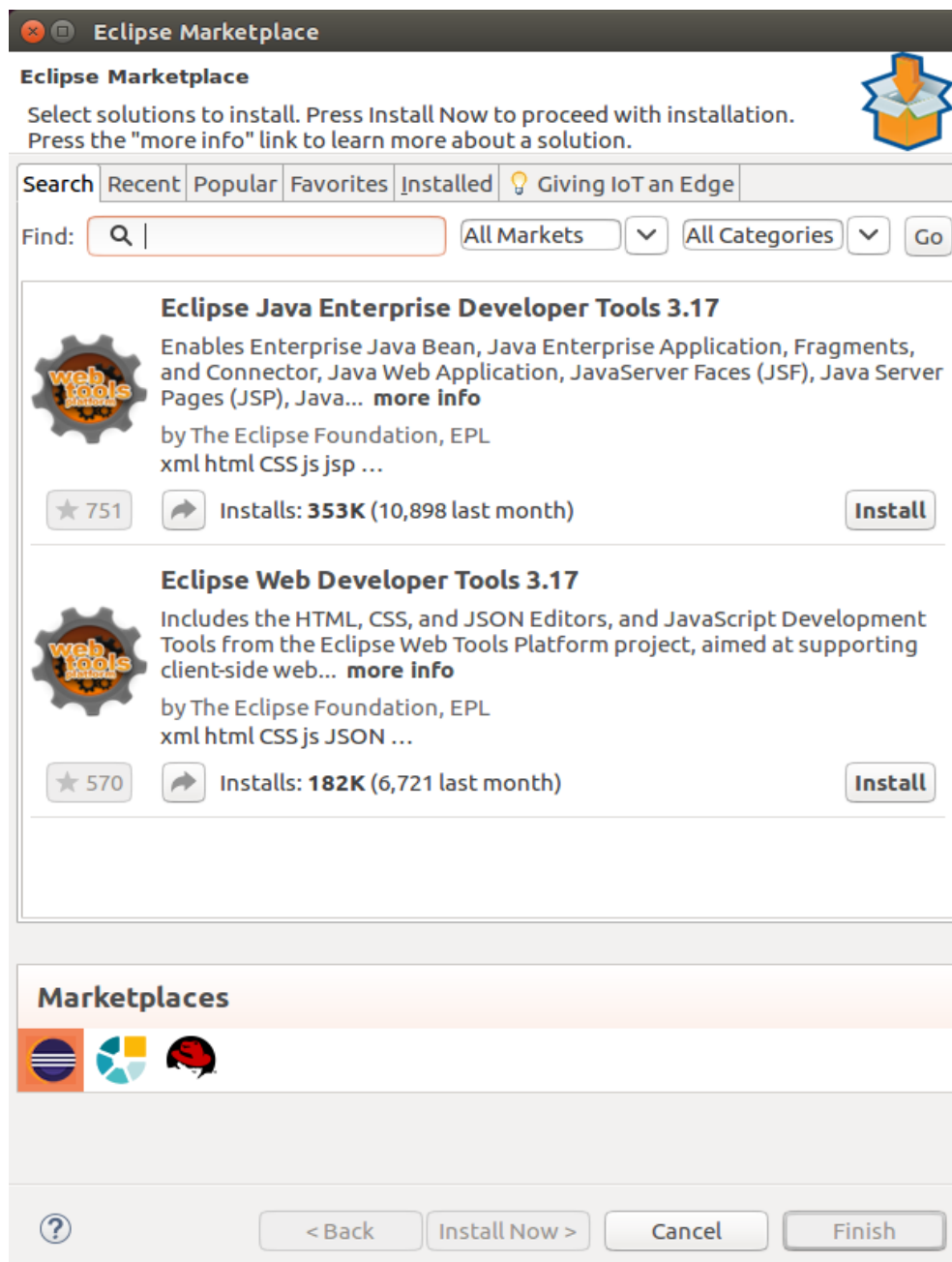
You have to import GAIT in Eclipse. Open “File” and find “Import” and select “Existing Projects into Workspace” from the Selection Wizard, then select “Next”.



Browse to find the location where you download the project named “maze Demo”, make sure the project is checked, then hit “Finish”



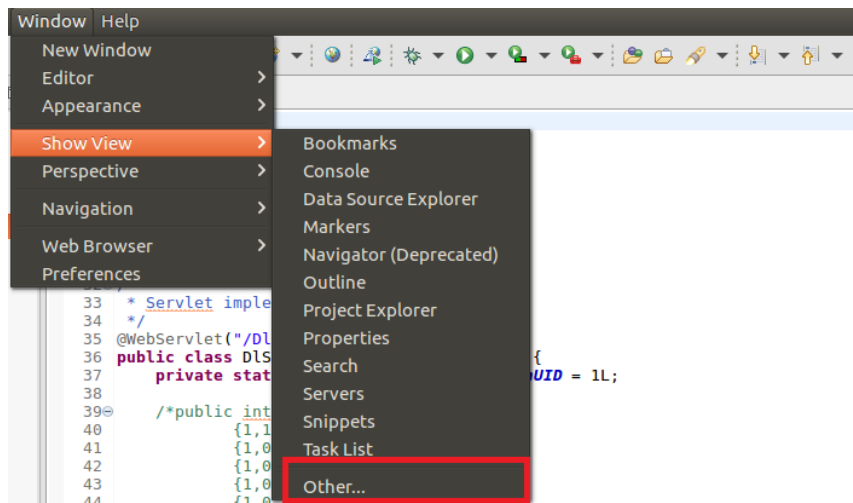
Eclipse may ask you to install Marketplace extensions, install it.



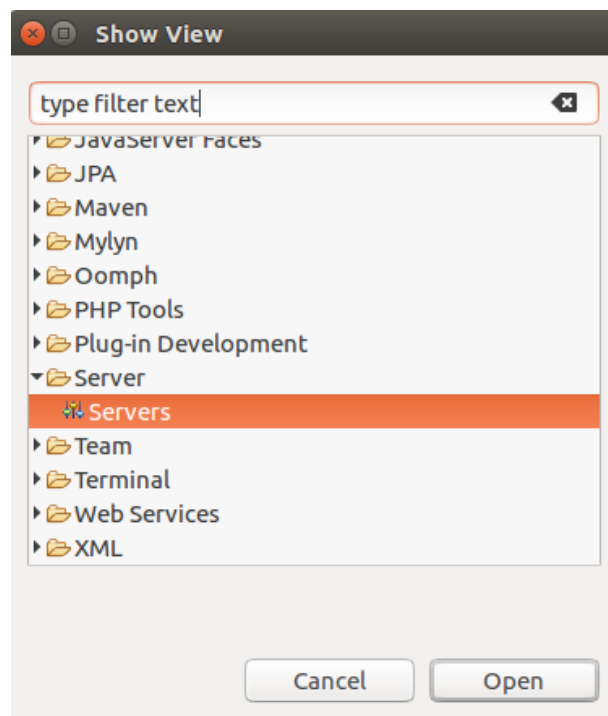
After the project has been recompiled, you will be surprised to find that some Java classes will show some errors in the console window. Please don't worry and remain calm, these errors will not affect the function of the project. The errant code and classes were designed to test the functionality of the code, which will be gradually removed and the project code will be updated in a later release version.

## 7. Tomcat Server Configuration

Firstly, you have to add an extension of Eclipse. Got to “Window”, then “Show view”, then “Other”.

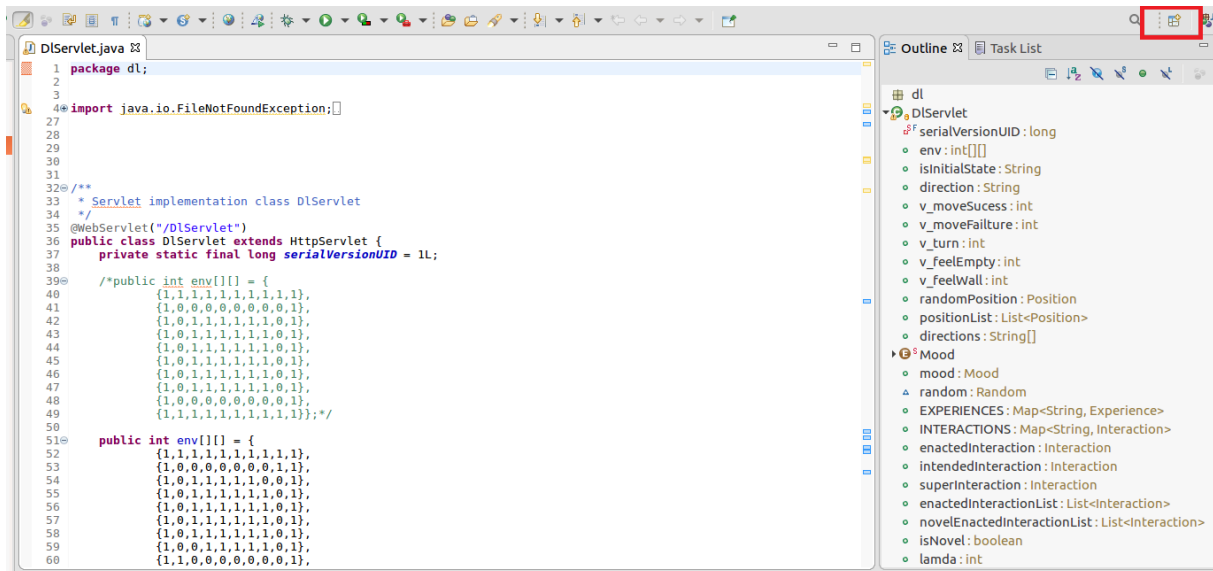


Search “Server” and select “Servers”.

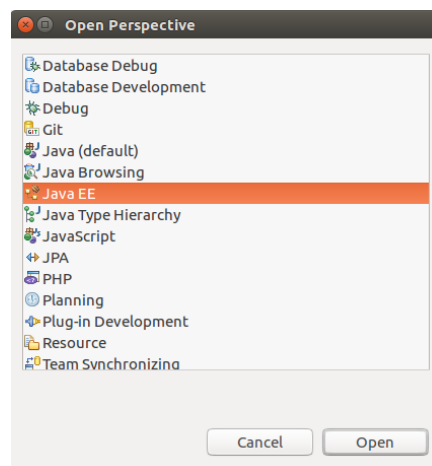




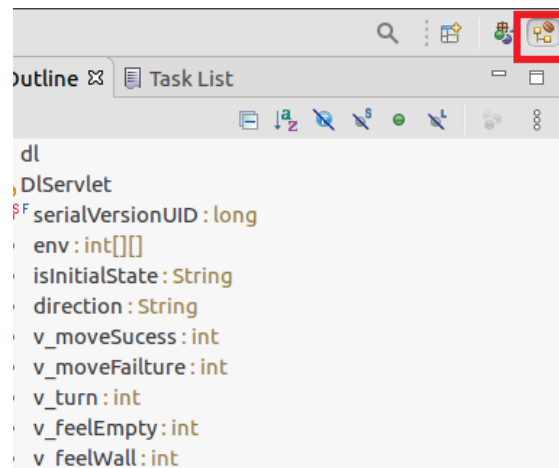
Click the button “Open Perspective”.



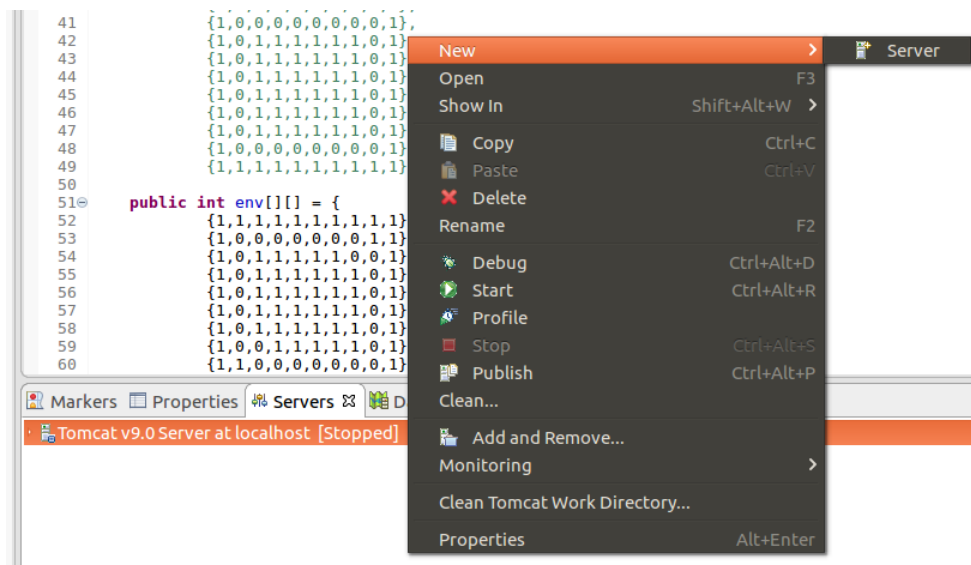
Select “Java EE”.



Click the “Java EE” Button.



A “Servers” Window” appears, right click on it. Select “New”, then “Server.



Select “Tomcat v9.0 Server”, then select “Next” to get the Import Server.

New Server

Define a New Server

Choose the type of server to create

Select the server type:

type filter text

Tomcat v7.0 Server

Tomcat v8.0 Server

Tomcat v8.5 Server

Tomcat v9.0 Server

Basic

Eclipse Libra

IBM

ObjectWeb

Oracle

Publishes and runs J2EE and Java EE Web projects and server configurations to a local Tomcat server.

Server's host name:

localhost

Server name:

Tomcat v9.0 Server at localhost (2)

Server runtime environment:

Apache Tomcat v9.0

Add...

Configure runtime environments...

?

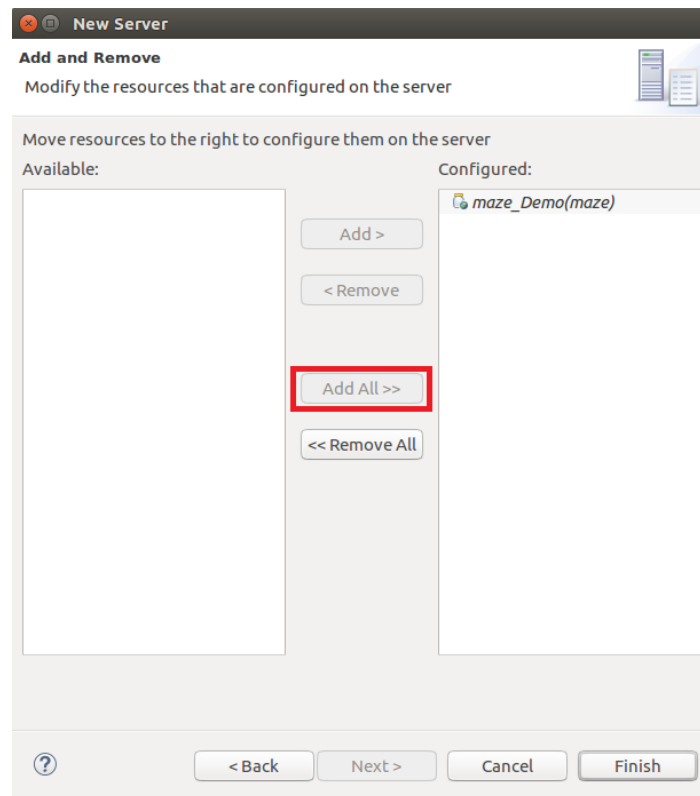
< Back

Next >

Cancel

Finish

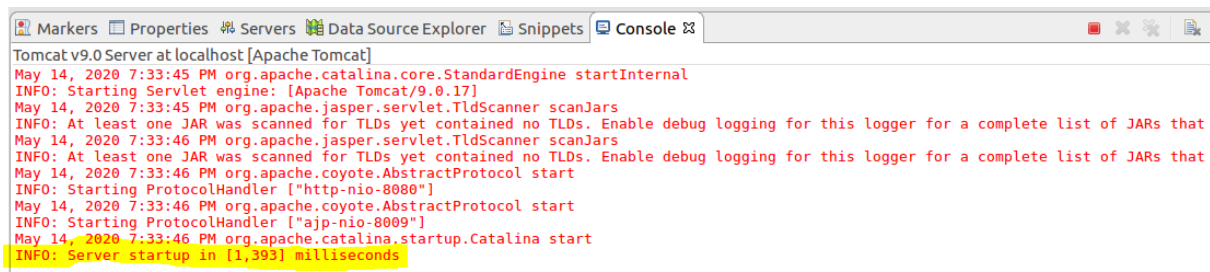
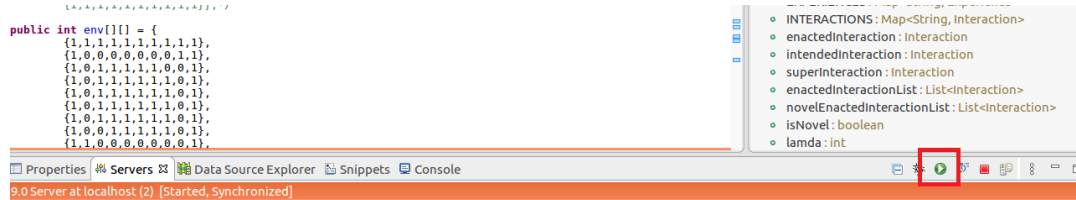
Click on the “Add All >>” to configure the server with maze\_Demo(maze). Then, you can click on “Finish”.



The project is now deployed on your Tomcat server.

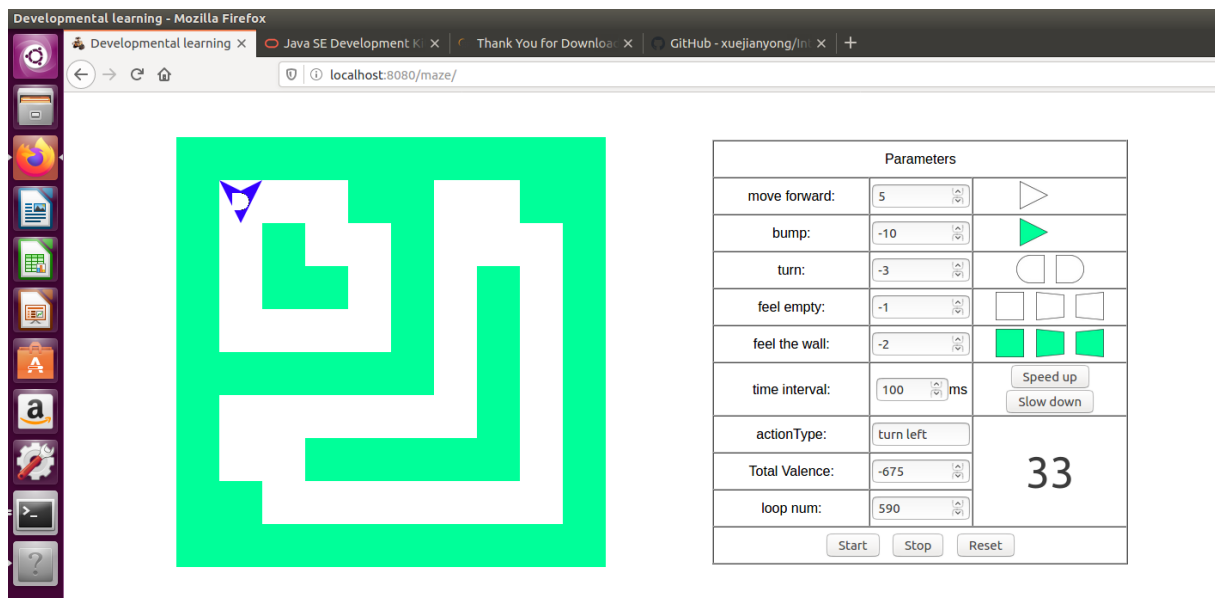
## 8. GAIT Launch

You can now run the project with the “Run” button.



The server started.

There is only one more thing to do: you have to open your navigator and to paste this address: <http://localhost:8080/maze/>



Good Job! GAIT is now ready to use!

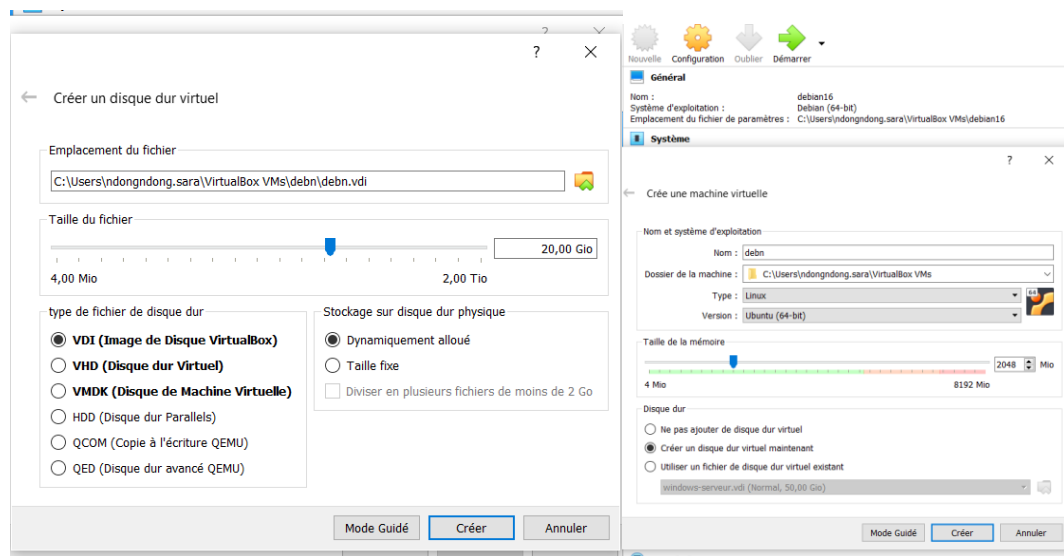
## I. ORACLE VIRTUALBOX ET UBUNTU

### 1. Télécharger et installer oracle VirtualBox

(<https://www.virtualbox.org/wiki/Downloads>)

### 2. Ouvrir VirtualBox et créer une machine virtuelle Ubuntu(64bits) sous linux

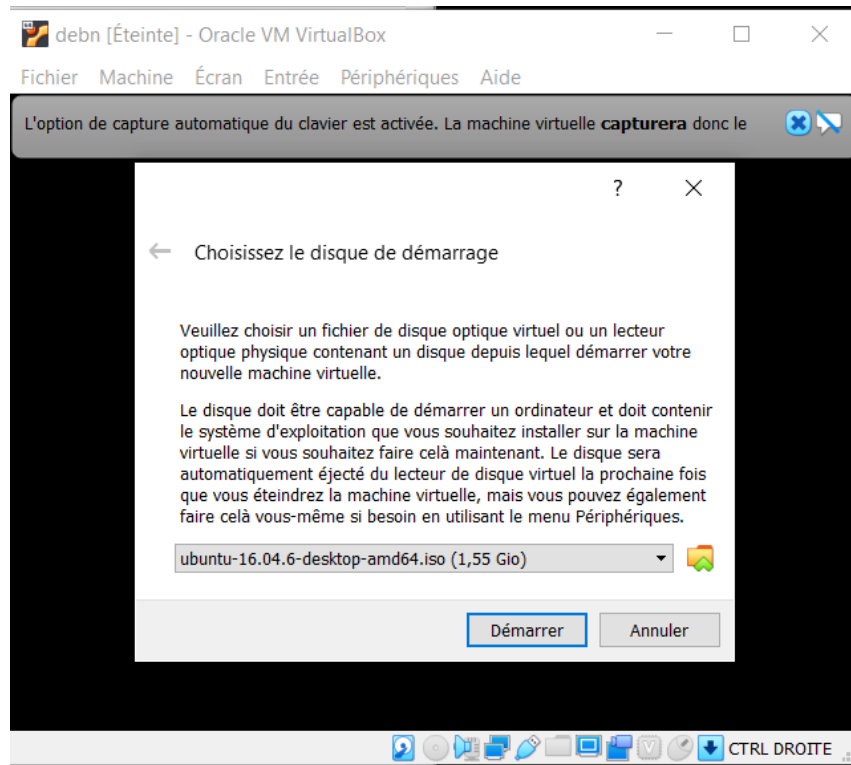
### 3. Augmenter la taille de la mémoire de la machine a 1084mio ou plus et la taille du fichier a 10 gio ou plus



### 4. Télécharger une version Ubuntu(dans notre cas la 16.04)

(<http://releases.ubuntu.com/16.04/>) prendre desktop-amd64.iso

### 5. Démarrer la machine virtuelle et introduire le disque Ubuntu télécharger puis installer Ubuntu (laisser les paramètres par défaut)



## II. INSTALLATION DE ROS-KINETIC

Installer ros-kinetic dans la machine virtuelle

(<http://wiki.ros.org/kinetic/Installation/Ubuntu> lien du tuto)

6. Configurez votre source. List faire :

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.lis'
```

Configurez vos clés

7. faire:

```
$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Installer ros .

8. faire:

```
$ sudo apt-get update
$ sudo apt-get install ros-kinetic-desktop-full
```

C'est pratique si les variables d'environnement ROS sont automatiquement ajoutées à votre session bash chaque fois qu'un nouveau shell est lancé.

9. Faire :

```
$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc  
$ source ~/.bashrc
```

Dépendances pour la construction de packages .

10. Faire :

```
$ sudo apt install python-rosdep python-rosinstall python-rosinstall-generator  
python-wstool build-essential
```

Avant de pouvoir utiliser de nombreux outils ROS, vous devez initialiser rosdep. rosdep vous permet d'installer facilement les dépendances système pour la source que vous souhaitez compiler et est requis pour exécuter certains composants de base dans ROS.

11. Faire:

```
$ sudo apt install python-rosdep  
$ rosdep update
```

### III. CREER UN ESPACE DE TRAVAIL ROS

12. Faire :

```
$ mkdir -p ~/catkin_ws/src  
$ cd ~/catkin_ws/src  
Dans /catkin_ws/src faire : $ catkin_init_workspace  
Dans /catkin_ws/src faire : $ cd ..  
Dans /catkin_ws faire : $ catkin_make
```

Pour les utilisateurs de Python 3 installez les dépendances .

13. Faire :

```
$catkin_make -DPYTHON_EXECUTABLE=/usr/bin/python3
```



## SIMULATION TURTLEBOT3 AVEC ROS

1. Ouvrez un nouveau terminal et entrez :

```
$ cd ~/catkin_ws/src/
```

```
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
```

```
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
```

```
$ cd ~/catkin_ws && catkin_make
```

TurtleBot3 a trois modèles, Burger, Waffle et Waffle Pi, vous devez donc définir le modèle que vous souhaitez utiliser avant de lancer TurtleBot3. Tapez cette commande pour ouvrir le fichier `bashrc` pour ajouter ce paramètre.

2. Faire :

```
$ gedit ~/ .bashrc
```

3. Ajoutez cette ligne en bas du fichier:

```
$ export TURTLEBOT3_MODEL = burger
```

```
# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands. Use like so:
# sleep 10; alert
alias alert='notify-send --urgency=low -i "${?} = 0" && echo terminal'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

source /opt/ros/melodic/setup.bash

source ~/catkin_ws/devel/setup.bash
export TURTLEBOT3_MODEL=burger
```

Enregistrez le fichier et fermez-le.

Rechargez maintenant .bashrc afin de ne pas avoir à vous déconnecter et vous reconnecter .

4. Faire :

```
$ source ~/.bashrc
```

Maintenant, nous devons télécharger les fichiers de simulation TurtleBot3.

5. Faire :

```
$ cd ~/catkin_ws/src/
```

Utilisons maintenant Gazebo pour faire la simulation TurtleBot3 .

6. Faire :

```
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
```

```
ros@ros-VirtualBox:~/catkin_ws$ cd ~/catkin_ws/src/
ros@ros-VirtualBox:~/catkin_ws/src$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
Cloning into 'turtlebot3_simulations'...
remote: Enumerating objects: 148, done.
remote: Counting objects: 100% (148/148), done.
remote: Compressing objects: 100% (92/92), done.
remote: Total 1761 (delta 75), reused 116 (delta 52), pack-reused 1613
Receiving objects: 100% (1761/1761), 13.97 MiB | 4.51 MiB/s, done.
Resolving deltas: 100% (1012/1012), done.
ros@ros-VirtualBox:~/catkin_ws/src$
```

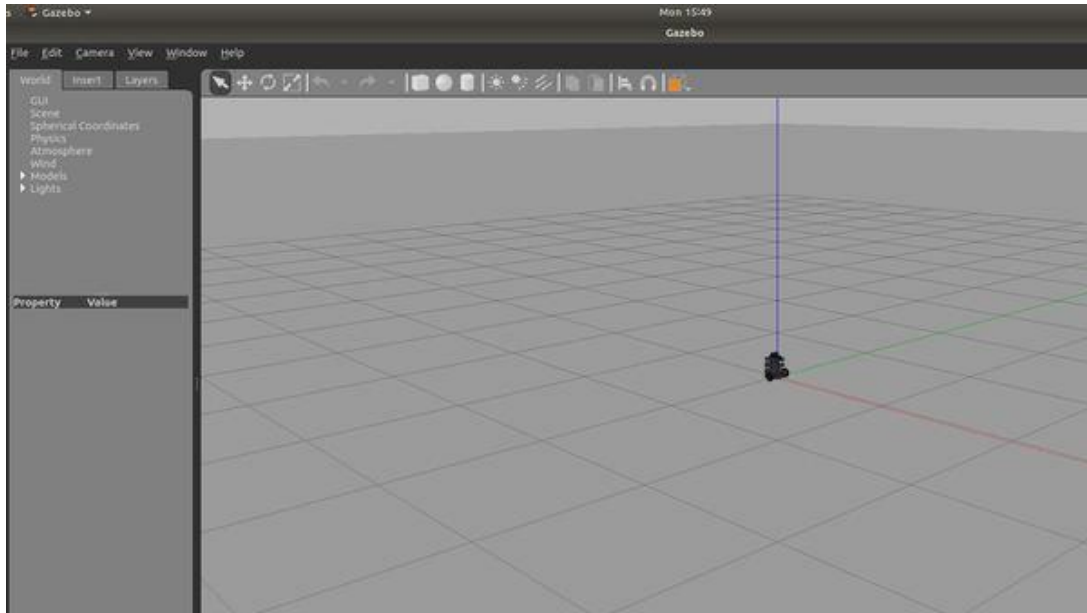
```
$ cd ~/catkin_ws && catkin_make
```

```
[ 30%] Built target turtlebot3_msgs_generate_messages_cpp
[ 34%] Built target turtlebot3_msgs_generate_messages_lisp
[ 46%] Built target turtlebot3_example_generate_messages_eus
[ 59%] Built target turtlebot3_example_generate_messages_py
[ 69%] Built target turtlebot3_example_generate_messages_nodejs
Scanning dependencies of target turtlebot3_fake_node
Scanning dependencies of target turtlebot3_drive
[ 80%] Built target turtlebot3_example_generate_messages_lisp
[ 90%] Built target turtlebot3_example_generate_messages_cpp
[ 90%] Built target turtlebot3_msgs_generate_messages
[ 93%] Built target turtlebot3_diagnostics
[ 93%] Built target turtlebot3_example_generate_messages
[ 95%] Building CXX object turtlebot3_simulations/turtlebot3_gazebo/CMakeFiles/turtlebot3_drive.dir/src/turtlebot3_drive.cpp
[ 96%] Building CXX object turtlebot3_simulations/turtlebot3_fake/CMakeFiles/turtlebot3_fake_node.dir/src/turtlebot3_fake.cpp
[ 98%] Linking CXX executable /home/ros/catkin_ws/devel/lib/turtlebot3_gazebo/turtlebot3_drive
[ 98%] Built target turtlebot3_drive
[100%] Linking CXX executable /home/ros/catkin_ws/devel/lib/turtlebot3_fake/turtlebot3_fake_node
[100%] Built target turtlebot3_fake_node
ros@ros-VirtualBox:~/catkin_ws$
```

Commençons par lancer TurtleBot3 dans un environnement vide.

7. Tapez cette commande:

```
$ roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```



8. Si vous souhaitez déplacer TurtleBot3 sur l'écran, ouvrez une nouvelle fenêtre de terminal et tapez la commande suivante :

```
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Cliquez sur la fenêtre du terminal et utilisez les touches ci-dessous pour contrôler le mouvement de votre TurtleBot (par exemple, appuyez sur la touche W pour avancer, la touche X pour reculer et S pour arrêter).

```
PARAMETERS
* /model: burger
* /roscdistro: melodic
* /rosversion: 1.14.3

NODES
/
  turtlebot3_teleop_keyboard (turtlebot3_teleop/turtlebot3_teleop_key)
ROS_MASTER_URI=http://localhost:11311
process[turtlebot3_teleop_keyboard-1]: started with pid [19540]
Control Your TurtleBot3!
-----
Moving around:
    w    a    s    d    x
    |    |    |    |    |
w/x : increase/decrease linear velocity (Burger : ~ 0.22, Waffle and Waffle Pi : ~ 0.26)
a/d : increase/decrease angular velocity (Burger : ~ 2.84, Waffle and Waffle Pi : ~ 1.82)
space key, s : force stop
CTRL-C to quit
```