**Trace QGIS**

Documentation

ligne horizontale

# 

[**Présentation Général 4**](#_8wtz18gd890d)

[**Notice d’installation 5**](#_dgla4jzfw47k)

[Prérequis 5](#_l05acg6l2vm)

[Qgis 5](#_qjogd1u53sfx)

[Dépendance 5](#_72mfivb459bl)

[Python 5](#_o8dvcoydut68)

[Installation du QGIS 5](#_356lon8e2lki)

[Installation du plugin 5](#_hlmxurs04m3w)

[Téléchargement 5](#_ewcw04ewe3gz)

[Installation des dépendance 6](#_s0q16io2efvn)

[Installation via QGIS 6](#_f7hhfko4yg6g)

[Vérification du fonctionnement 7](#_1madpic2gxy3)

[**Documentation Utilisateur 9**](#_9z8nqb5h9sfa)

[Interface 9](#_1ifd6jrwn950)

[Interface - Paramétrage 9](#_so4khnwmt54e)

[Information 10](#_q2k93ggij1j6)

[Temporalité 11](#_dz6khc1k2zrb)

[Focus 11](#_7s6j542lsdss)

[**Documentation Technique 12**](#_7l092hoy89r6)

[Architecture 12](#_xdnn3hzfu0hm)

[asset 13](#_5brakipeizhe)

[custom/action 13](#_eugvvqa1j89k)

[custom/business 14](#_xazzf5vg6l29)

[custom/constants 15](#_q75240fqw58u)

[custom/enums 15](#_uf4mpkor2id6)

[custom/manager 15](#_ghxwiqw2xvor)

[custom/utils 15](#_j01apkmioymr)

[interface 16](#_9t8dq4udlssl)

[schema 16](#_gezffz69zpq6)

[test 16](#_6ne2fh90nq82)

[Autre 16](#_vmahz2wu3p41)

[LayerTraceQGIS 17](#_q0e3vhp73at9)

[Timer : 17](#_p60cz1yepph3)

[Refresh : 17](#_hou7w27afmc2)

[MapEntity 17](#_39q07l75pkzp)

[generate\_category 18](#_gek9t4wie6p7)

[update\_label\_position 18](#_s3d9y3qghz60)

[generate\_description\_label 18](#_cytufsh0ergl)

[Signal 18](#_7gbyrocvvogn)

[Action 18](#_oyw3mje9xb1)

[Actions définies: 18](#_9nlb2ujql0l6)

[Note 20](#_nvizve7lus1h)

[Utilisation 20](#_ufuhu4tdn698)

# 

# 

# Présentation Général

Trace QGIS est un plugin pour le logiciel QGIS qui permet de visualiser et animer des entités géographiques en fonction d’actions définies dans des plans. Il permet de représenter les mouvements d’entités sur une carte, de générer des traces et divers d'autres actions afin de visualiser des scénarios complexes.

Il embarque des fonctionnalités qui permettent d'interpréter les fichiers problème et domaine du PDDL (Planning Domain Definition Language) afin de générer un fichier qui permet visualisation le plan issus de la résolution.

# 

# Notice d’installation

## Prérequis

### Qgis

Afin d’utiliser le plugin il faut un environnement QGIS valide

### Dépendance

Pour utiliser ce plugin QGIS vous allez devoir installer des librairies python externe les voici:

| Nom | Utilité |
| --- | --- |
| jsonshema | Valider des fichiers JSON à l’aide d’un schéma |
| pyyaml | Permet de lire des fichier YAML |
| pddlpy | Permet de lire les fichier issus de PDDL |

### Python

Avoir l'environnement par défaut python de QGIS

## Installation du QGIS

Pour installer QGIS nous vous conseillons d’utiliser la procédure officiel disponible directement sur le site : <https://qgis.org/>

Voilà les versions valider actuellement:

| Version | LTR | Valide |
| --- | --- | --- |
| QGIS 3.40 | Oui | Oui |

## Installation du plugin

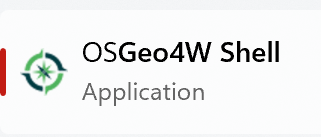
### Téléchargement

Télécharger le plugin via la dernière releases du dépôt officiel : <https://github.com/yo-teck/TraceQgis/releases>

### Installation des dépendance

Afin d’installer les dépendances il vous faut un environnement QGIS.

Il vous suffit ensuite d’ouvrir "OSGeo4W Shell" fournis par QGIS



Une fois le terminal ouvert, il suffit d’installer les dépendances suivante via pip:

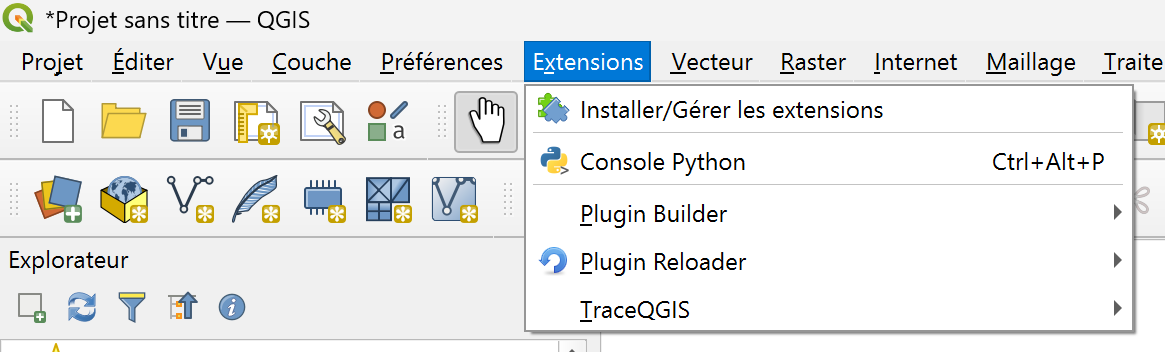
| Nom | Utilité |
| --- | --- |
| jsonshema | Valider des fichiers JSON à l’aide d’un schéma |
| pyyaml | Permet de lire des fichier YAML |
| pddlpy | Permet de lire les fichier issus de PDDL |

Exemple:

| pip install jsonschema pyyaml pddlpy |
| --- |

### Installation via QGIS

Il suffit d’utiliser l’installation classique de QGIS, aller dans Menu Extensions → Installer/Gérer les extensions



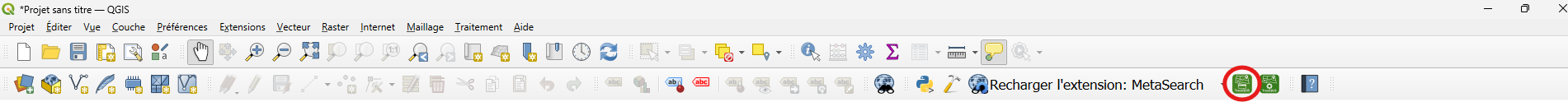
Puis sélectionner le zip issues de la release

# 

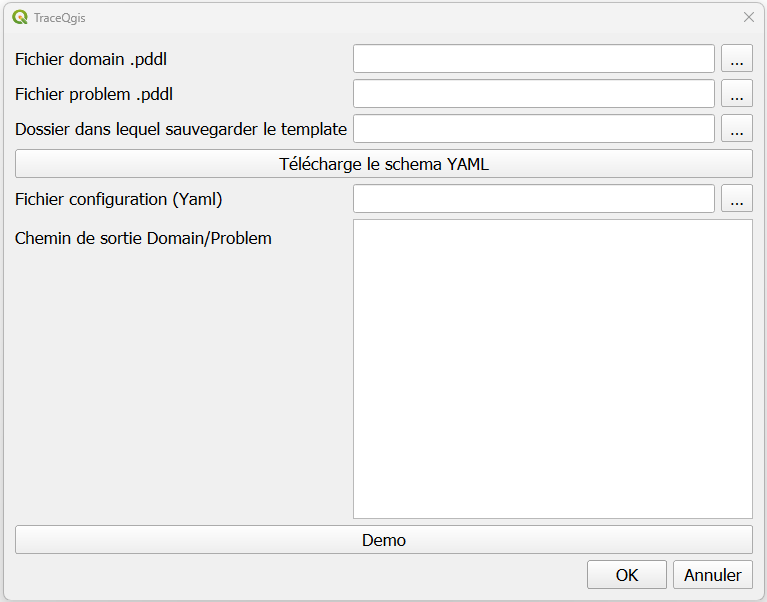
Puis cliquer sur installer l'extension.

### Vérification du fonctionnement

Afin de vérifier le bon fonctionnement vous pouvez utiliser le mode démo en appuyant le premier bouton :



Puis cliquer le sur le bouton “Demo”



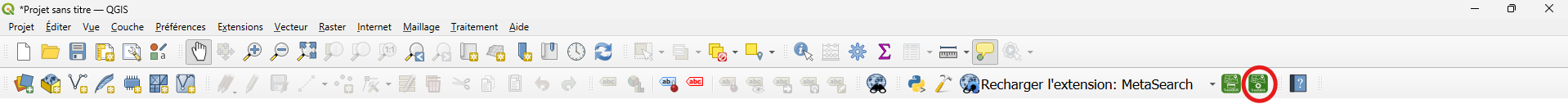
Zoomer sur la couche “Trace qgis” et vérifier que cela s'exécute sans erreur.

# Documentation Utilisateur

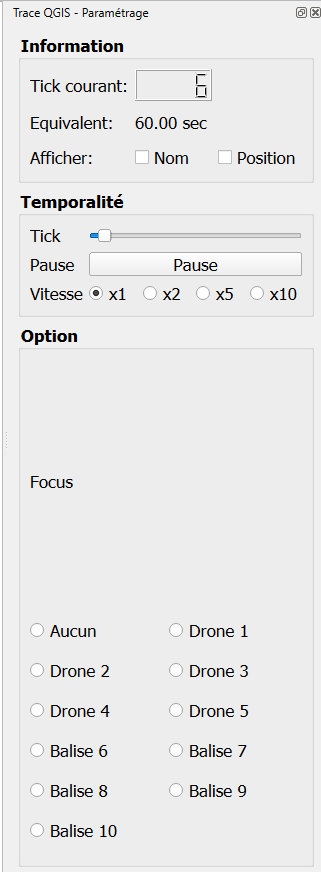
## Interface

### Interface - Paramétrage

Un fois le lancement effectué, vous pouvez ouvrir l’onglet paramétrage du plugin via le deuxième bouton:

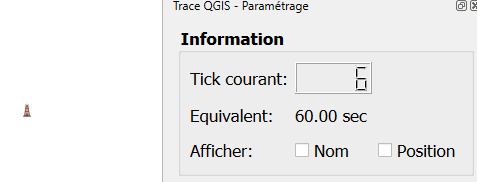


Vous devriez voire apparaître une interface “dock” à droite de l’écran:

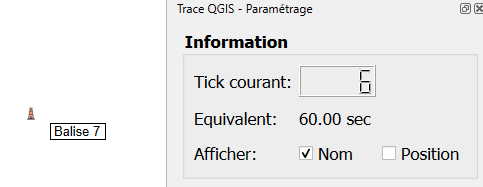


#### Information

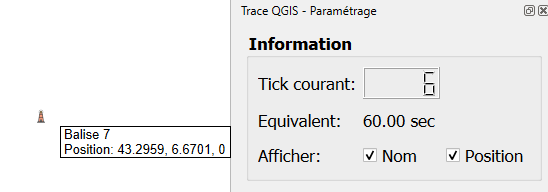
L’onglet information permet de visualiser le tick courant c'est-à-dire le déroulement de l’animation et son équivalent 1 tick = 10 secondes ici.  
  
Il est également possible d'afficher le nom et la position des entitées sur la carte



Avec “Nom” coché:



Avec “Nom” et “Position” coché:



#### Temporalité

L’onglet temporalité permet de mettre en pause le temps, de se déplacer dans la “timeline” à l’aide d’un slider et d'accélérer la vitesse.  
L'accélération de la vitesse diminue le délai de rafraîchissement. Par défaut il y a un rafraichissement par secondes en le passant en x2 on passe 2 par seconde et ainsi de suite. Cette fonctionnalité peut-être gourmande en performance.

#### Focus

L’onglet focus permet de focus une entités en cliquant dessus c’est à dire que le centre de la vue de la carte va être constamment sur l’entité sélectionnée

# 

# Documentation Technique

## Architecture

Cette partie va présenter l’architecture du plugin par dossier et fichier important.

| .  ├── assets  ├── custom  │ ├── actions  │ │ ├── action.py  │ │ ├── action\_add\_text.py  │ │ ├── action\_around.py  │ │ ├── action\_arrow.py  │ │ ├── action\_background.py  │ │ ├── action\_change\_icon.py  │ │ ├── action\_change\_size.py  │ │ ├── action\_factory.py  │ │ ├── action\_highlight.py  │ │ ├── action\_load.py  │ │ ├── action\_move.py  │ │ ├── action\_move\_to.py  │ │ ├── action\_opacity.py  │ │ ├── action\_rotate.py  │ │ └── action\_unload.py  │ ├── business  │ │ ├── dm\_carto\_configuration.py  │ │ ├── domain\_problem\_model.py  │ │ ├── layer\_trace\_qgis.py  │ │ └── map\_entity.py  │ ├── constants  │ │ └── pddl\_yaml.py  │ ├── enums  │ │ ├── action\_mapping.py  │ │ └── predicat\_mapping.py  │ ├── manager  │ │ └── domain\_problem\_manager.py  │ └── utils  │ ├── adapter\_helper.py  │ ├── utils.py  │ └── yaml\_helper.py  ├── help  ├── i18n  ├── icon.png  ├── interface  │ ├── trace\_qgis\_dialog\_file.py  │ ├── trace\_qgis\_dialog\_file.ui  │ ├── trace\_qgis\_dock\_widget\_setting.py  │ └── trace\_qgis\_dock\_widget\_setting.ui  ├── metadata.txt  ├── schema  │ └── base\_yaml\_validator.json  ├── scripts  ├── test  └── trace\_qgis.py |
| --- |

### asset

Dossier prévu pour stocker des fichiers statiques (images, icônes supplémentaires, styles, etc.)

### custom/action

Dossier contenant toutes les classes Action qui modélisent des actions exécutées sur des entités. Ces actions sont interprétées depuis un plan pour animer les entités sur la carte.

* **action.py** Classe de base commune à toutes les actions.
* **action\_add\_text.py** Ajoute un texte à une entité pendant l’animation.
* **action\_around.py** Fait tourner une entité autour d’un point central (orbitation).
* **action\_arrow.py** Ajoute un symbole de fleche sur une entité.
* **action\_background.py** Change l'image de fond associée à une entité.
* **action\_change\_icon.py** Change l’icône affichée pour une entité.
* **action\_change\_size.py** Modifie dynamiquement la taille d'une entité.
* **action\_factory.py** Fabrique des actions à partir de dictionnaires. C’est le point central pour créer les actions dynamiquement.
* **action\_highlight.py** Applique un effet de surbrillance sur une entité (avec couleur).
* **action\_load.py** Charge une entité sur la carte (la rend visible).
* **action\_move.py** Déplace une entité progressivement (avec interpolation des coordonnées).
* **action\_move\_to.py** Déplace une entité vers une autre entité cible.
* **action\_opacity.py** Modifie l’opacité d’une entité.
* **action\_rotate.py** Fait pivoter une entité sur elle-même.
* **action\_unload.py** Supprime une entité de la carte (la rend invisible).

### custom/business

Dossier contenant la logique principale et l’intégration avec QGIS.

* **dm\_carto\_configuration.py** Chargement/gestion des configurations cartographiques spécifiques à ton domaine.
* **domain\_problem\_model.py** Structure les modèles de domaine/problème (ex : PDDL) utilisés pour générer les actions et les entités.
* **layer\_trace\_qgis.py** Cœur du plugin : gère les couches (layers), les entités et l'animation des actions dans QGIS.
* **map\_entity.py** Représente une entité (point sur la carte) : icône, position, label, etc.

### custom/constants

Dossier contenant les constantes nécessaires aux classes.

* **pddl\_yaml.py**Constantes pour interpréter les fichiers PDDL/YAML décrivant les plans.

### custom/enums

Dossier contenant les enums lié au module

* **action\_mapping.py** Correspondance entre types d’action (ex : du YAML) et classes Python.
* **predicat\_mapping.py** Correspondance entre les prédicats PDDL et le modèle Python.

### custom/manager

* **domain\_problem\_manager.py**Gère les fichiers de domaine/problème PDDL et leur parsing.

### custom/utils

Dossier contenant les différents méthodes utiles

* **adapter\_helper.py** Aide pour transformer des structures (adapter des données).
* **utils.py** Fonctions mathématiques et diverses (azimut, point à distance, tri des actions…).
* **yaml\_helper.py** Chargement et validation des fichiers YAML.

### interface

Dossier contenant les interfaces

* **trace\_qgis\_dialog\_file.py / .ui** Fenêtre pour charger un fichier de plan (PDDL/YAML).
* **trace\_qgis\_dock\_widget\_setting.py / .ui** Panneau latéral de configuration pendant l’animation.

### schema

* **base\_yaml\_validator.json**Schéma JSON pour valider les fichiers YAML (structure attendue).

### test

Dossier contenant tous les tests unitaires

### Autre

* metadata.txt  
  Fichier descriptif du plugin pour QGIS (nom, version, description, etc).
* trace\_qgis.py  
  Point d’entrée du plugin (le fichier que QGIS charge).

## LayerTraceQGIS

LayerTraceQGIS est la classe au cœur du fonctionnement des animations, il repose sur plusieurs mécanismes pour mettre à jour et assurer la cohérence des entités et des couches associées dans l'application.  
  
C’est le porteur de la couche TraceQgis. Voilà quelques particularités pour la comprendre

### Timer :

Elle permet de gérer un timer avec un système de tickrate pour afficher et gérer l’animation.

### Refresh :

Le cœur du système de mise à jour. Elle coordonne les différentes étapes nécessaires pour actualiser les entités, couches, ou tout autre élément lié.

reset\_before\_refresh: Permet de faire des actions avant le reset comme supprimer tous les textes.

refresh\_load: Permet de faire ne pas faire apparaître les entités chargé

refresh\_action: Exécute toutes les actions au tick indiquer

refresh\_line: Gère l’affichage des flèches entre les entités

refresh\_focus : Permet gérer le focus sur une entité

Ce système est couplé au système de timer qui permet de l’appeler tous les secondes par défaut.

## MapEntity

La classe MapEntity représente une entité géographique dynamique affichée sur la carte QGIS.

Elle regroupe les données de position, d'apparence, de label, et propose des méthodes pour interagir avec l'entité. Voila les fonction plus important:

### generate\_category

Cette méthode permet de générer la catégorie pour la couche il permet de gérer l'icône, la taille, l’image de fond, la rotation, etc

### update\_label\_position

Cette méthode permet de déplacer l’annotation en fonction de l’écran et de l’interface QGIS.

### generate\_description\_label

Cette méthode permet de gérer la description de l’annotation.

## Signal

Dans le projet nous utilisons pyqtSignal il permet définir des signaux personnalisés. Cela nous permet de créer des signaux ce qui nous permet ensuite d'abonnés des fonctions au signal afin d'exécuter des actions. Ces signaux nous permettent une communication entre différents composants de l’interface et les autres parties du plugin

## Action

Le plugin repose sur une liste d’action qui permet d’animer les entités. Voila celle que nous avons implémenté :

### Actions définies:

| **Type d’action** | **Paramètres** | **Description** |
| --- | --- | --- |
| move | start\_at, end\_at, entity\_id, lat\_from, lon\_from, alti\_from, lat\_to, lon\_to, alti\_to, text | Déplace une entité d’une position géographique de départ (lat\_from, lon\_from, alti\_from) vers une position cible (lat\_to, lon\_to, alti\_to) pendant un intervalle de ticks. |
| move\_to | start\_at, end\_at, entity\_id, entity\_id2, distance, text | Déplace une entité vers une autre entité à une certaine distance. |
| text | start\_at, end\_at, entity\_id, text | Ajoute un texte flottant ou une annotation à l’entité pendant l’action. |
| arrow | start\_at, end\_at, entity\_id, entity\_id2, text | Trace une flèche entre deux entités, souvent pour représenter une relation. |
| around | start\_at, end\_at, entity\_id (entité principal), entity\_id2 (entité vers laquelle on tourne), distance, angle, text | Fait tourner une entité autour d’une autre à une distance donnée et avec un angle initial ou final. |
| image | start\_at, end\_at, entity\_id, image, text | Change l’icône de l’entité en remplaçant son image par celle fournie. |
| background | start\_at, end\_at, entity\_id, image, text | Ajouter une image de fond derrière l’entité (ex : halo, symbole spécial). |
| size | start\_at, end\_at, entity\_id, size, text | Modifie la taille de l’entité (agrandissement ou réduction). |
| opacity | start\_at, end\_at, entity\_id, opacity, text | Change l’opacité de l’entité pour la rendre plus ou moins visible. |
| rotate | start\_at, end\_at, entity\_id, angle, text | Fait pivoter l’entité selon un angle donné. |
| highlight | start\_at, end\_at, entity\_id, color, text | Met en surbrillance l’entité avec une couleur spécifique. |
| load | start\_at, end\_at, entity\_id(entité a charger), entity\_id2(entité dans laquel charger), text | Charge une entité dans une autre. |
| unload | start\_at, end\_at, entity\_id(entité a décharger), entity\_id2(entité de laquel décharger), text | Décharge une entité d’une autre. |

### Note

Pour chaque action il est possible d’ajouter un texte ce qui permet d’éviter de faire 2 actions.

### Utilisation

Pour l’utiliser il suffit de créer un action sous forme d'objet.

Exemple:

| action = {  "type": "move\_to",  "start\_at": 110,  "end\_at": 120,  "entity\_id": 1,  "entity\_id2": 8,  "distance": 0,  "text": "Rapprochement de la balise" } |
| --- |