Projet Minetest

Spécialité NSI 1ères

Lycée Saint sernin

Introduction

Projet Minetest

Objectif

Minetest est un jeu vidéo de monde ouvert qui s'inspire fortement de Minecraft. Il est open source, gratuit, multi plateformes.

Sommaire

1. Découvrir Minetest
    a) Installation
    b) Prise en main
2. Exécuter du code
    a) Installation
    b) Découverte de la bibliothèque
    c) Manipuler
3. Utilisation de données externes
    a) Données csv : Labyrinthe
    b) Données json : Blocs

Découvrir Minetest

Projet Minetest

Installation

Remarque

La combinaison de touches alt tab permet de passer d'un logiciel à l'autre.

Activité 1 : L'application **minetest** est normalement installée sur la clé usb. Dans le répertoire **bin,** lancer le fichier **minetest.**

Prise en main

- Activité 2 :
  1. Sélectionner le monde `Pycraft` et cliquer sur
     - `Jouer`.
  2. Tester les commandes :
     - **W** avancer
     - **S** reculer
     - **D** aller à droite
     - **A** aller à gauche
     - **Espace** sauter ; double-cliquer pour voler
     - **Shift** descendre
     - **I** inventaire
     - **clic-gauche** miner un bloc
     - **clic-droit** déposer un bloc

Remarque

Si le vol n’est pas fonctionnel.

Activité 3 :

1. Appuyer sur le touche **T** (Tchat).
2. Dans le tchat, entrer le code :

```
/grant singleplayer all
```

3. Double-cliquer sur la barre d’espace pour voler.

Activité 4 :

1. Prendre un lit dans l’inventaire.
2. Le placer sur la carte.
3. Un clic-droit sur le lit permet de passer la nuit.

Exécuter du code

Projet Minetest

Installation

Projet Minetest

Exécuter du code - Installation

Remarque

Cette étape est déjà faite car le proxy n'autorise pas l'appel à pip. Mais c'est la manière de faire
La documentation est ici: https://pypi.org/project/minetest-python-nsi/

Activité 5 :

1. Ouvrir l'EDI **Thonny**.
2. Dans le menu **Outils**, choisir **Gérer les paquets**.
3. Chercher et installer le paquet **`minetest-python-nsi`**.
4. Depuis le menu de présentation du paquet, ouvrir la **page PyPI**.

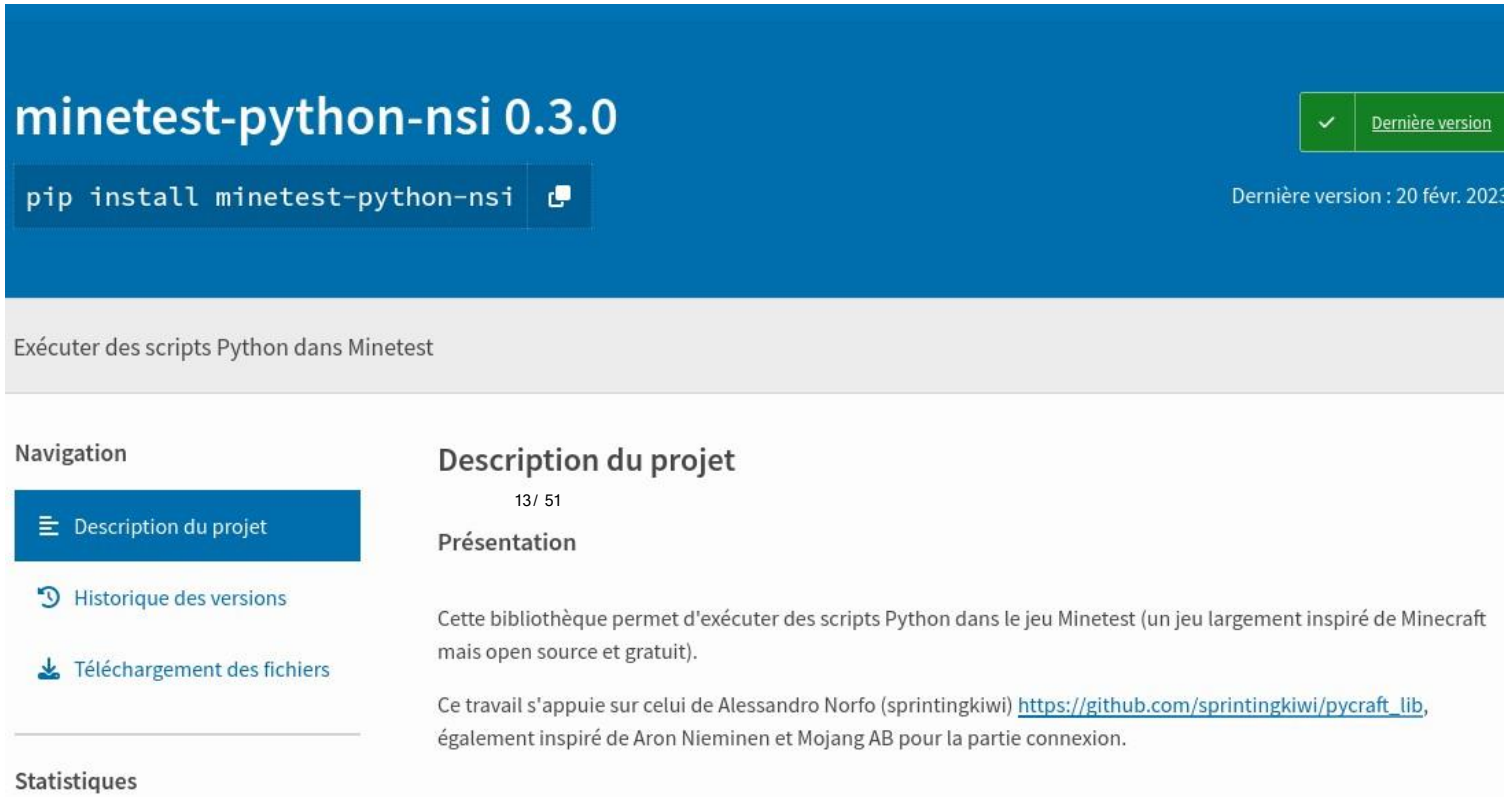


Figure 1 – La description propose également des exemples d'utilisation.

Installation

Projet Minetest

Découverte de la bibliothèque

À retenir

Une bibliothèque est normalement accompagnée d'une documentation. Elle présente le rôle et le mode d'utilisation de chaque fonction proposée.

Activité 6 :

1. Lire la **page PyPI** puis ouvrir la documentation (lien en bas de page). L'adresse est :

   https://cviroulaud.gitlab.io/minetest-doc/

2. Parcourir la documentation.
3. Ouvrir la liste des blocs existants.

Module **minetest_python_nsi**

@Author: Christophe Viroulaud @Time: Dimanche 05 Février 2023 23:16

▸ EXPAND SOURCE CODE

Fonctions

```
def changer_ma_position(coord: tuple) -> None
```

- Repositionnement du joueur sur une autre tuile

Paramètres

- coord (tuple): nouvelle coordonnées

▸ EXPAND SOURCE CODE

```
def changer_position(id: int, coord: tuple) -> None
```

- Repositionnement du joueur 'id' sur une autre tuile

Paramètres

- id (int): identifiant du joueur
- coord (tuple): nouvelle coordonnées

▸ EXPAND SOURCE CODE

Figure 2 – Prendre le temps de comprendre la documentation

Activité 7 :

1. Depuis **Thonny**, créer un fichier **`minetest-python.py`**
2. Importer la bibliothèque.
3. Créer une connexion avec le jeu en mode local.
4. Récupérer les coordonnées de la position du joueur.
5. Récupérer la rotation du joueur.

Aide

```
recuperer_ma_position() -> tuple

    Récupère la position de la tuile sous le
   joueur

    Renvoi

        tuple: coordonnées (x, y, z)
```

Code 1 – La fonction n'accepte aucun paramètre et renvoie un tuple : on peut utiliser ce tuple dans le programme.

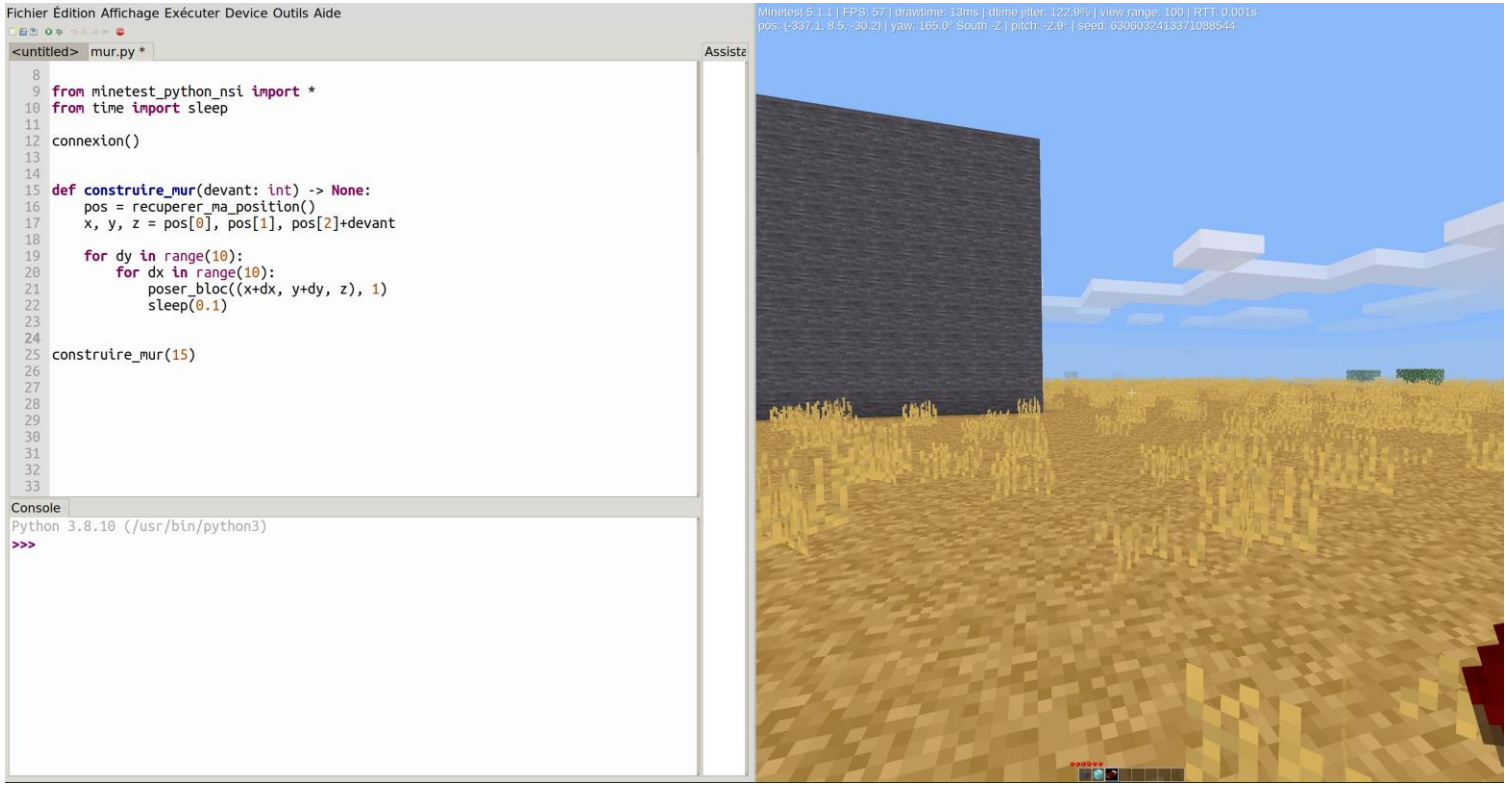


Figure 3 – Agencer son espace de travail

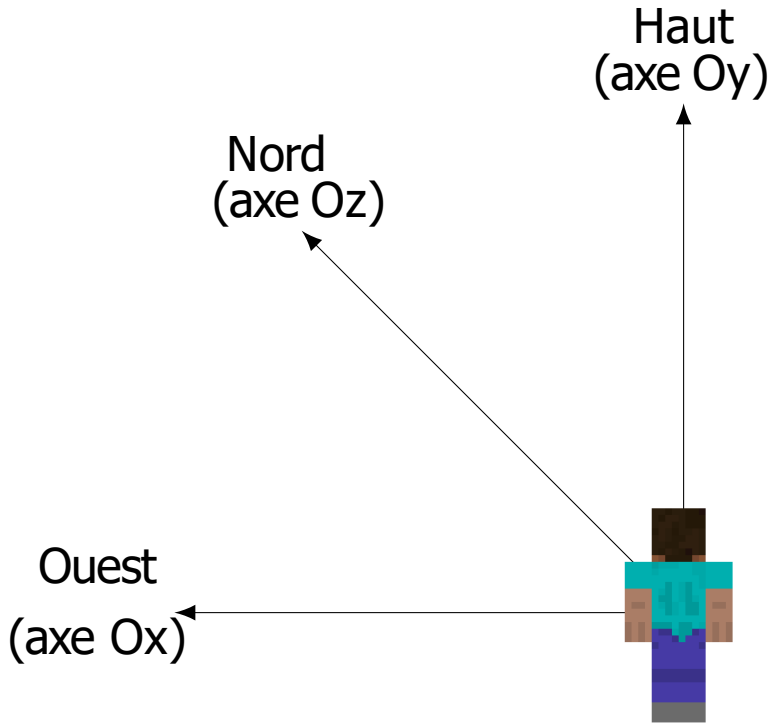


Figure 4 – Système de coordonnées

ATTENTION

Dans le jeu, il est possible de faire apparaître une console de débogage (**F5**).
Le Nord et le Sud sont inversés par rapport aux valeurs de rotation de `recuperer_ma_rotation()`.

Activité 8 :

1. Poser un bloc de type **GRASS** à côté du joueur.
2. Poser à côté un bloc de type **WOOD** en **Chêne.**

```
def poser_bloc(coord: tuple, bloc: int, donnees: int = 0) -> None:
"""
- place un bloc aux coordonnées données
- pour certains blocs, il est possible de donner des informations
supplémentaires (couleur, type de bois, orientation...)

Paramètres:
    - coord (tuple): coordonnées du bloc
    - bloc (int): type de bloc
    - donnees (int, optionnel): informations supplémentaires (couleur,
      type de bois...) """
```

Remarque

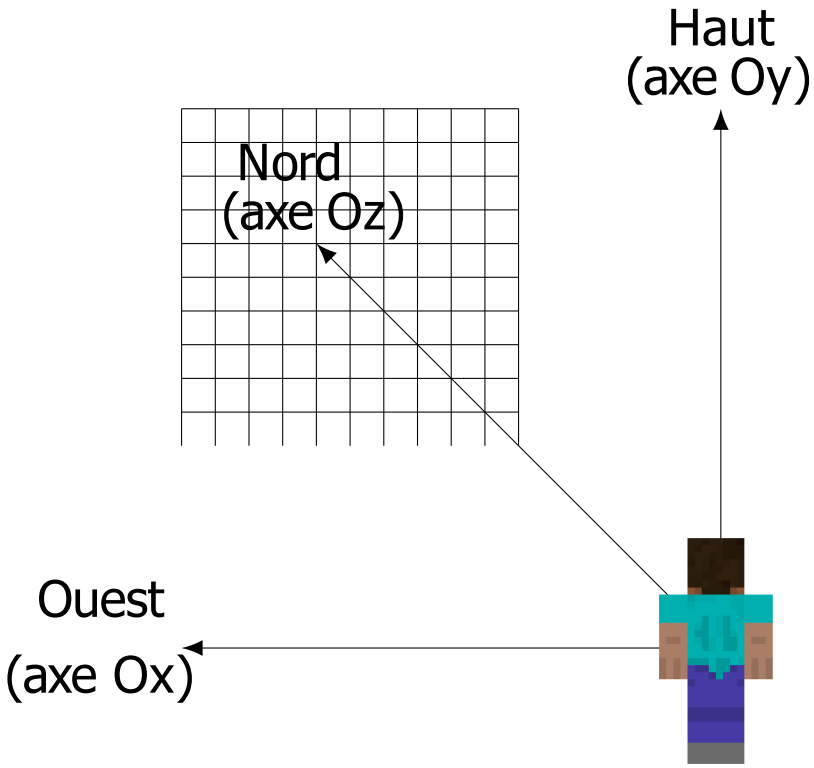
La fonction **poser_bloc** accepte un paramètre optionnel.

Labyrinthe : données en csv

Projet Minetest

Manipuler

Activité 9 : Écrire la fonction **`construire_mur(devant: int)`** → **None** qui construit un mur carré de 10 blocs de **STONE** de côté, dans le plan xOy et à la distance **`devant`** vers le nord.



Utilisation de données externes - Données `csv` : Labyrinthe

Activité 10 :

1. Télécharger et extraire le dossier compressé **`minetest.zip`**
2. Ouvrir le fichier **`labyrinthe.csv`** avec le bloc-notes.

- **0** espace vide
- **1** mur
- **2** danger

Figure 5 – Dans le labyrinthe

Activité 11 :

1. Proposer un algorithme qui construit un labyrinthe à partir des données du fichier.
2. Écrire un programme qui :
    1. charge les données de **`labyrinthe.csv`** et crée un lecteur **`csv`**.
    2. construit le labyrinthe correspondant avec :
        - des murs de glace de hauteur 2,
        - des feux pour danger.

Correction algortihme

Pour chaque ligne :

- initialiser x à sa valeur initiale
- augmenter z de 1
- Pour chaque valeur de la ligne :
  - Si la valeur est 1, construire un mur.
  - Sinon si la valeur est 2, construire un danger
  - augmenter x de 1

```
1  pos = recuperer_ma_position()
2  x, y, z = pos[0], pos[1], pos[2]
3
4  #pour observer la construction du labyrinthe
5  changer_ma_position((pos[0]+10, pos[1]+20, pos[2]))
```

Code 4 – Attention au placement de la ligne

Construction de blocs : données en json

Projet Minetest

Données `json` : Blocs

À retenir

Il existe plusieurs formats de stockage de données :

- **`csv`:** Comma Separated Values ; stockage *linéaire* des données.
- **`json`:** JavaScript Notation Object ; stockage structuré des données sous forme d'objets.

Activité 12 :

1. Ouvrir le fichier **`blocs.json`** avec **un editeur de texte** ou **Firefox.**
2. À quelles structures de données Python peut-on comparer les constructions du fichier ?

Correction

```
{
    "AIR": {
        "id": 0,
        "nom_fr": "Air",
        "donnees": {}
    },
    "STONE": {
        "id": 1,
        "nom_fr": "Roche",
        "donnees": {}
    },
    ...
}
```

Code 5 – Un dictionnaire de dictionnaires

```
"WOOD_PLANKS": {
    "id": 5,
    "nom_fr": "Planche",
    "donnees": {
        "0": {
            "us": "Oak",
            "fr": "chêne"
        },
        "1": {
            "us": "Spruce",
            "fr": "épicéa"
        },
        ...
    }
}
```

Code 6 – La clé **donnees** référence un dictionnaire.

Remarque

Par défaut, les clés des dictionnaires sont des chaînes de caractères.

À retenir

Comme pour les fichiers **csv**, le langage Python propose une bibliothèque pour importer les fichiers **json**.
Cette bibliothèque convertit automatiquement les accolades en dictionnaires et les crochets en tableaux.

```
1 import json
2
3 fichier = open("blocs.json", "r", encoding="utf8")
4 blocs = json.load(fichier)
5 fichier.close()
```

Activité 13 :

1. Dans un nouveau fichier Python, exécuter le code ci-dessus.
2. Afficher les données associées à la clé **NETHER_REACTOR_CORE**

Activité 14 : Écrire un programme qui construit un mur de longueur 10, composé de tous les blocs existants.

- Si un bloc est déclinable en plusieurs variantes, il faudra créer 1 bloc par possibilité.
- De plus, certains blocs ne restent pas solides quand ils sont posés (lave, eau). Ils ne devront pas être utilisés (blocs 8, 9, 10, 11)
- Enfin il faudra poster dans le chat, le nom du bloc construit.

```
1  for nom, bloc in blocs.items():
2      if bloc["id"] not in (8, 9, 10, 11):
3          poster("construction de "+ nom)
4          coord = (pos[0]+dx, pos[1]+dy, pos[2]+dz)
5
6          #avec ou sans données
7          if len(bloc["donnees"]) == 0:
8              poser_bloc(coord, bloc["id"])
9          else:
0              for data in bloc["donnees"]:
1                  # attention: la clé est un str
2                  poser_bloc(coord, bloc["id"], int(data))
```

```
1            #mise à jour des coordonnées
2            dx = dx+1
3            if dx == 9:
4                dx = 0
5                dy = dy+1
```

```
pos = recuperer_ma_position()
dx, dy, dz = 0, 0, 10

for nom, bloc in blocs.items():
    if bloc["id"] not in (8, 9, 10, 11):
        poster("construction de "+ nom)
        coord = (pos[0]+dx, pos[1]+dy, pos[2]+dz)

        if len(bloc["donnees"]) == 0:
            poser_bloc(coord, bloc["id"])
        else:
            for data in bloc["donnees"]:
                poser_bloc(coord, bloc["id"], int(data))

        dx = dx+1
        if dx == 9:
            dx = 0
            dy = dy+1
    sleep(0.5)
```