

Programmation Orientée Objet
Minecraft

Christophe Viroulaud

Terminale - NSI

Lang 01

Programmation Orientée Objet Minecraft

Christophe Viroulaud

Terminale - NSI

Lang 01

Principes du jeu

Programmation
orientée objet

Définition

Modélisation

Instances d'un objet

Implémentation

Créer un objet

Initialiser les attributs

Définir les méthodes

Instancier une classe

└─ Minecraft

Minecraft

Minecraft est un jeu mélangeant construction et aventure, créé en 2009 par Markus « Notch » Persson. Il permet à ses joueurs de manipuler un monde en trois dimensions, composé entièrement de blocs à détruire, placer et transformer.



Minecraft

Minecraft est un jeu mélangeant construction et aventure, créé en 2009 par Markus « Notch » Persson. Il permet à ses joueurs de manipuler un monde en trois dimensions, composé entièrement de blocs à détruire, placer et transformer.



Quel paradigme mettre en place pour programmer Minecraft ?

Quel paradigme mettre en place pour programmer Minecraft ?

1. Principes du jeu
2. Programmation orientée objet
3. Implémentation

Sommaire

1. Principes du jeu

2. Programmation orientée objet

3. Implémentation

faire une démo

Principes du jeu

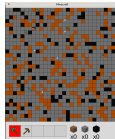


FIGURE 1 – Une version simplifiée

Principes du jeu

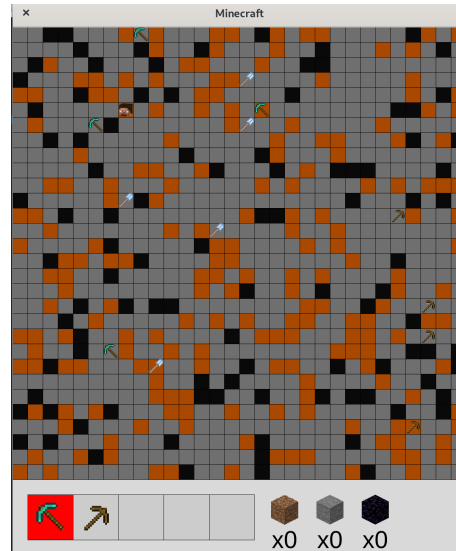


FIGURE 1 – Une version simplifiée

- R : ramasser un outil,
- W X C V B : choisir un outil de l'inventaire,
- Espace : miner,
- Flèches : se déplacer.

Le moteur du jeu se chargera de l'affichage graphique des concepts présentés ci-après.

Règles

- R : ramasser un outil,
- W X C V B : choisir un outil de l'inventaire,
- Espace : miner,
- Flèches : se déplacer.

Le moteur du jeu se chargera de l'affichage graphique des concepts présentés ci-après.

3 types de blocs



FIGURE 2 – Des caractéristiques différentes

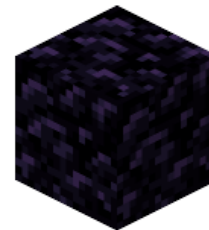
3 types de blocs



terre



roche



obsidienne

FIGURE 2 – Des caractéristiques différentes



3 types d'outils



FIGURE 3 – Des actions différentes



- » récupérer des outils,
- » stocker des blocs,
- » miner des blocs,
- » labourer des terres.

Un héros avec des capacités



- récupérer des outils,
- stocker des blocs,
- miner des blocs,
- labourer des terres.

Sommaire

1. Principes du jeu

2. Programmation orientée objet

2.1 Définition

2.2 Modélisation

2.3 Instances d'un objet

3. Implémentation

Le **paradigme objet** consiste à construire des objets et les faire interagir entre eux. Un objet représente une entité physique, un concept...

Définition

Le **paradigme objet** consiste à construire des *objets* et les faire interagir entre eux. Un objet représente une entité physique, un concept...

Un objet possède :
▸ des caractéristiques : **les attributs**,
▸ des capacités : **les méthodes**.

Un objet possède :

- des caractéristiques : **les attributs**,
- des capacités : **les méthodes**.

Une voiture :
▸ attributs : rouge, électrique...
▸ méthodes : rouler, freiner...

Exemple

Une voiture :

- attributs : rouge, électrique...
- méthodes : rouler, freiner...

Sommaire

1. Principes du jeu

2. Programmation orientée objet

2.1 Définition

2.2 Modélisation

2.3 Instances d'un objet

3. Implémentation

Modélisation

Il est nécessaire de modéliser le problème en représentant les objets et leurs interactions.



Les blocs et leurs attributs

Terre
nom
dureté
couleur
est labourée

Roche
nom
dureté
couleur
est gravat

Obsidienne
nom
dureté
couleur

Les outils et leurs méthodes

Terre	Roche	Obsidienne
nom	nom	nom
dureté	dureté	dureté
couleur	couleur	couleur
est labourée	est gravat	

Pelle	Pioche
nom	nom
résistance	résistance
impact	impact
labourer	piocher

Les outils et leurs méthodes

Terre
nom
dureté
couleur
est labourée

Roche
nom
dureté
couleur
est gravat

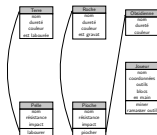
Obsidienne
nom
dureté
couleur

Pelle
nom
résistance
impact
labourer

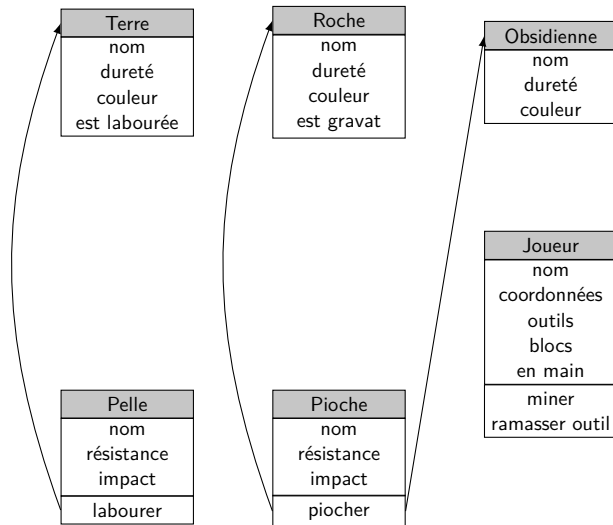
Pioche
nom
résistance
impact
piocher

1. faire relations avec joueur à la main
2. ramasser outil vers les outils
3. miner vers les blocs

Le joueur et les relations



Le joueur et les relations



Sommaire

1. Principes du jeu

2. Programmation orientée objet

2.1 Définition

2.2 Modélisation

2.3 Instances d'un objet

3. Implémentation



Instances d'un objet

- Chaque objet est un modèle qui peut être vu comme un squelette.

Terre
nom
dureté
couleur
est labourée

- Chaque objet est un modèle qui peut être vu comme un squelette.

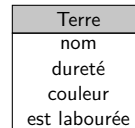


- On crée une **instance** de l'objet. C'est cette instance qui interagit dans le programme.

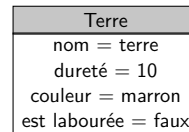
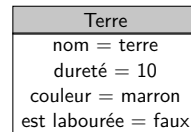
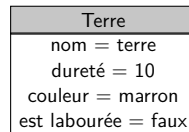


Instances d'un objet

- Chaque objet est un modèle qui peut être vu comme un squelette.



- On crée une **instance** de l'objet. C'est cette instance qui interagit dans le programme.





- On crée une **instance** de l'objet. C'est cette instance qui interagit dans le programme.

Terre
nom = terre
dureté = 10
couleur = marron
est labourée = faux



Terre
nom = terre
dureté = 10
couleur = marron
est labourée = faux



Terre
nom = terre
dureté = 10
couleur = marron
est labourée = faux



- Chaque instance est indépendante des autres.

Terre
nom = terre
dureté = 10
couleur = marron
est labourée = faux



Terre
nom = terre
dureté = 10
couleur = brun
est labourée = vrai



Terre
nom = terre
dureté = 10
couleur = marron
est labourée = faux



Sommaire

1. Principes du jeu

2. Programmation orientée objet

3. Implémentation

3.1 Créer un objet

3.2 Initialiser les attributs

3.3 Définir les méthodes

3.4 Instancier une classe

```
1 class Terre
```

Code 1 – Le mot-clé class

Créer un objet

```
1 class Terre
```

Code 1 – Le mot-clé `class`

Sommaire

1. Principes du jeu

2. Programmation orientée objet

3. Implémentation

3.1 Créer un objet

3.2 Initialiser les attributs

3.3 Définir les méthodes

3.4 Instancier une classe

```

1 class Terre:
2     def __init__(self):
3         self.nom = "terre"
4         self.durete = 10
5         self.couleur = "#a94800"
6         self.est_labouree = False

```

Code 2 – Les attributs sont initialisés dans le constructeur.

Initialiser les attributs

La *méthode* `__init__` est appelée automatiquement quand nous instancions un objet.

```

1 class Terre:
2     def __init__(self):
3         self.nom = "terre"
4         self.durete = 10
5         self.couleur = "#a94800"
6         self.est_labouree = False

```

Code 2 – Les attributs sont initialisés dans le *constructeur*.

Le mot-clé `self` identifie l'instance de l'objet.

Le mot-clé `self` identifie l'instance de l'objet.

Activité 1 :

1. Télécharger et extraire le dossier compressé **minecraft.zip** sur le site <https://cviroulaud.github.io>
2. Dans le fichier **blocs.py** construire les objets :

Roche

- nom : stone,
- dureté : 100,
- couleur : #6f6f6f,
- est_gravat : False

Obsidienne

- nom : obsidian,
- dureté : 1000,
- couleur : #090909

Activité 1 :

1. Télécharger et extraire le dossier compressé **minecraft.zip** sur le site <https://cviroulaud.github.io>
2. Dans le fichier **blocs.py** construire les objets :

Roche

- nom : stone,
- dureté : 100,
- couleur : #6f6f6f,
- est_gravat : False

Obsidienne

- nom : obsidian,
- dureté : 1000,
- couleur : #090909

```

1 class Roche:
2     def __init__(self):
3         self.nom = "stone"
4         self.durete = 100
5         self.couleur = "#6f6f6f"
6         self.est_gravat = False
7
8
9 class Obsidienne:
10    def __init__(self):
11        self.nom = "obsidian"
12        self.durete = 1000
13        self.couleur = "#090909"

```

Correction

```

1 class Roche:
2     def __init__(self):
3         self.nom = "stone"
4         self.durete = 100
5         self.couleur = "#6f6f6f"
6         self.est_gravat = False
7
8
9 class Obsidienne:
10    def __init__(self):
11        self.nom = "obsidian"
12        self.durete = 1000
13        self.couleur = "#090909"

```

```

1 class Pioche:
2     def __init__(self, nom: str):
3         self.nom = nom
4         if nom == "wood_pickaxe":
5             self.resistance = 30
6             self.impact = 5
7         elif nom == "diamond_pickaxe":
8             self.resistance = 100
9             self.impact = 100

```

Code 3 – Une autre classe : il est possible d'ajouter des paramètres au constructeur.

une autre approche : un même objet pour toutes les pioches.

```

1 class Pioche:
2     def __init__(self, nom: str):
3         self.nom = nom
4         if nom == "wood_pickaxe":
5             self.resistance = 30
6             self.impact = 5
7         elif nom == "diamond_pickaxe":
8             self.resistance = 100
9             self.impact = 100

```

Code 3 – Une autre classe : il est possible d'ajouter des paramètres au constructeur.

Sommaire

1. Principes du jeu

2. Programmation orientée objet

3. Implémentation

3.1 Créer un objet

3.2 Initialiser les attributs

3.3 Définir les méthodes

3.4 Instancier une classe

en javascript c'est **this**

On appelle **méthode** une fonction interne à la classe de l'objet.
En Python, le premier paramètre est **toujours self**. C'est un attribut **interne** à la classe.

Définir les méthodes

On appelle **méthode** une fonction interne à la classe de l'objet.

En Python, le premier paramètre est **toujours self**. C'est un attribut **interne** à la classe.


```

1 def piocher(self, bloc: object) -> bool:
2     """
3     donne un coup sur le bloc
4
5     Args:
6         bloc (object): le bloc miné
7
8     Returns:
9         bool: False si l'outil est complètement usé
10    """

```

Code 4 – Signature de la méthode de la classe Pioche

```

1 def piocher(self, bloc: object) -> bool:
2     """
3     donne un coup sur le bloc
4
5     Args:
6         bloc (object): le bloc miné
7
8     Returns:
9         bool: False si l'outil est complètement usé
10    """

```

Code 4 – Signature de la méthode de la classe Pioche

```

1 def piocher(self, bloc: object) -> bool:
2     """
3     donne un coup sur le bloc
4
5     Args:
6         bloc (object): le bloc miné
7
8     Returns:
9         bool: False si l'outil est complètement usé
10
11         bloc.durete -= self.impact

```

Code 5 – On accède à un attribut **par la structure à point**.

```

1 def piocher(self, bloc: object) -> bool:
2     """
3     donne un coup sur le bloc
4
5     Args:
6         bloc (object): le bloc miné
7
8     Returns:
9         bool: False si l'outil est complètement usé
10
11         bloc.durete -= self.impact

```

Code 5 – On accède à un attribut **par la structure à point**.

► En Python, les attributs et les méthodes sont publics.
Ils sont accessibles de n'importe quel endroit du programme.

```
1 bloc.durete
```

- En Python, les attributs et les méthodes sont publics.
Ils sont accessibles de n'importe quel endroit du programme.

```
1 bloc.durete
```

► En Python, les attributs et les méthodes sont publics. Ils sont accessibles de n'importe quel endroit du programme.

```
1 bloc.durete
```

► Les attributs et méthodes *internes* à la classe sont accessibles avec le mot-clé `self`.

```
1 self.impact
```

- En Python, les attributs et les méthodes sont publics. Ils sont accessibles de n'importe quel endroit du programme.

```
1 bloc.durete
```

- Les attributs et méthodes *internes* à la classe sont accessibles avec le mot-clé `self`.

```
1 self.impact
```

```

1 def piocher(self, bloc: object) -> bool:
2     """
3     donne un coup sur le bloc
4
5     Args:
6         bloc (object): le bloc miné
7
8     Returns:
9         bool: False si l'outil est complètement usé
10    """
11    bloc.durete -= self.impact
12    self.resistance -= USURE
13    if self.resistance <= 0 :
14        return False
15    return True

```

Code 6 – Méthode de la classe Pioche

```

1 def piocher(self, bloc: object) -> bool:
2     """
3     donne un coup sur le bloc
4
5     Args:
6         bloc (object): le bloc miné
7
8     Returns:
9         bool: False si l'outil est complètement usé
10    """
11    bloc.durete -= self.impact
12    self.resistance -= USURE
13    if self.resistance <= 0 :
14        return False
15    return True

```

Code 6 – Méthode de la classe Pioche

Activité 2 : Dans le fichier `outils.py`, construire la méthode `labourer` de la classe `Pelle`.

- ▶ S'aider de la *docstring*.
- ▶ La couleur labourée est `#712712`.
- ▶ La pelle s'use.

Activité 2 : Dans le fichier **`outils.py`**, construire la méthode **`labourer`** de la classe **`Pelle`**.

- ▶ S'aider de la *docstring*.
- ▶ La couleur labourée est `#712712`.
- ▶ La pelle s'use.

```

1 def labourer(self, bloc: object) -> bool:
2     """
3     laboure un bloc terre (non déjà labourée),
4     ne fait rien sinon
5
6     Args:
7         bloc (object): le bloc en cours
8
9     Returns:
10        bool: False si l'outil est complètement usé
11        """
12
13     if bloc.nom == "dirt" and not bloc.est_labouree:
14         bloc.est_labouree = True
15         bloc.couleur = "#712712"
16         self.resistance -= USURE
17         if self.resistance <= 0:
18             return False
19         return True

```

Correction

```

1 def labourer(self, bloc: object) -> bool:
2     """
3     laboure un bloc terre (non déjà labourée),
4     ne fait rien sinon
5
6     Args:
7         bloc (object): le bloc en cours
8
9     Returns:
10        bool: False si l'outil est complètement usé
11        """
12
13     if bloc.nom == "dirt" and not bloc.est_labouree:
14         bloc.est_labouree = True
15         bloc.couleur = "#712712"
16         self.resistance -= USURE
17         if self.resistance <= 0:
18             return False
19         return True

```

inventaires : une liste et un dict

Étude de la classe Joueur

```

1 class Joueur:
2     def __init__(self, n: str):
3         self.nom = n
4         self.x = 0
5         self.y = 0
6         self.outils = [] # 5 maxi
7         self.blocs = {"dirt": 0, "stone": 0, "obsidian": 0}
8         self.en_main = 0 # outil en main

```

Code 7 – Constructeur

Étude de la classe Joueur

```

1 class Joueur:
2     def __init__(self, n: str):
3         self.nom = n
4         self.x = 0
5         self.y = 0
6         self.outils = [] # 5 maxi
7         self.blocs = {"dirt": 0, "stone": 0, "obsidian": 0}
8         self.en_main = 0 # outil en main

```

Code 7 – Constructeur


```

1 def miner(self, bloc: object) -> bool:
2     """
3     donne un coup sur le bloc avec l'outil en cours
4     ou la main
5
6     Args:
7         bloc (object): le bloc miné
8
9     Returns:
10        bool: True si le bloc est complètement miné
11        """

```

Code 8 – miner

Des méthodes

```

1 def miner(self, bloc: object) -> bool:
2     """
3     donne un coup sur le bloc avec l'outil en cours
4     ou la main
5
6     Args:
7         bloc (object): le bloc miné
8
9     Returns:
10        bool: True si le bloc est complètement miné
11        """

```

Code 8 – miner

```

1 def ramasser_outil(self, outil: object) -> bool:
2     """
3     place l'outil dans l'inventaire s'il y a de la place
4
5     Args:
6         outil (object): l'outil ramassé
7
8     Returns:
9         bool: True si l'outil a été ramassé
10    """

```

Code 9 – ramasser

```

1 def ramasser_outil(self, outil: object) -> bool:
2     """
3     place l'outil dans l'inventaire s'il y a de la place
4
5     Args:
6         outil (object): l'outil ramassé
7
8     Returns:
9         bool: True si l'outil a été ramassé
10    """

```

Code 9 – ramasser

Activité 3 : Compléter les méthodes de la classe
Joueur dans le fichier `joueur.py`

Activité 3 : Compléter les méthodes de la classe Joueur dans le fichier `joueur.py`

```

1 # récupère l'impact de l'objet en cours
2 impact = self.ouils[self.en_main].impact
3 # l'outil s'use
4 self.ouils[self.en_main].resistance -= USURE

```

Code 10 – Méthode `asoz`

- `impact` est une variable locale à la méthode.
- `USURE` est une variable globale au programme.
- `self.ouils` est un attribut de la classe.
- `self.ouils[self.en_main]` fait référence à un objet (un outil).

Correction

```

1 # récupère l'impact de l'objet en cours
2 impact = self.ouils[self.en_main].impact
3 # l'outil s'use
4 self.ouils[self.en_main].resistance -= USURE

```

Code 10 – Méthode `miner`

- `impact` est une variable *locale* à la méthode.
- `USURE` est une variable globale au programme.
- `self.ouils` est un attribut de la classe.
- `self.ouils[self.en_main]` fait référence à un objet (un outil).

```

1 if outil is not None:
2     if len(self.outils) < NB_OUTILS_INVENTAIRE:
3         # l'outil est ajouté à l'inventaire
4         self.outils.append(outil)
5         return True
6     return False

```

Code 11 – Méthode ramasser_outil

- outil est un paramètre de la méthode.
- self.outils est une méthode de la classe.

Correction

```

1 if outil is not None:
2     if len(self.outils) < NB_OUTILS_INVENTAIRE:
3         # l'outil est ajouté à l'inventaire
4         self.outils.append(outil)
5         return True
6     return False

```

Code 11 – Méthode ramasser_outil

- outil est un paramètre de la méthode.
- self.outils est une méthode de la classe.

Sommaire

1. Principes du jeu

2. Programmation orientée objet

3. Implémentation

3.1 Créer un objet

3.2 Initialiser les attributs

3.3 Définir les méthodes

3.4 Instancier une classe

on trouve aussi *instancier un objet*

Les classes vont permettre de créer les objets manipulables dans le programme.

```
1 un_bloc_terre = Terre()  
2 une_pioche_bois = Pioche("wood_pickaxe")
```

Code 12 – Instanciations

Instancier une classe

Les classes vont permettre de créer les objets manipulables dans le programme.

```
1 un_bloc_terre = Terre()  
2 une_pioche_bois = Pioche("wood_pickaxe")
```

Code 12 – Instanciations

Le jeu crée 900 blocs (**LARGEUR*HAUTEUR**) :

- ▶ 100 blocs d'obsidienne,
- ▶ 200 blocs de terre,
- ▶ 600 blocs de roche.

Il place ensuite au hasard 15 outils :

- ▶ 5 pelles,
- ▶ 5 pioches en bois,
- ▶ 5 pioches en diamant.

La classe **Moteur** fournit la méthode **donnees_coordonnees()** qui renvoie un tuple (x,y) de coordonnées non encore utilisé.

Activité 4 : Compléter le fichier **minecraft.py** en s'aidant des informations ci-dessus.

Le jeu crée 900 blocs (**LARGEUR*HAUTEUR**) :

- ▶ 100 blocs d'obsidienne,
- ▶ 200 blocs de terre,
- ▶ 600 blocs de roche.

Il place ensuite au hasard 15 outils :

- ▶ 5 pelles,
- ▶ 5 pioches en bois,
- ▶ 5 pioches en diamant.

La classe **Moteur** fournit la méthode **donnees_coordonnees()** qui renvoie un tuple (x,y) de coordonnées non encore utilisé.

Activité 4 : Compléter le fichier **minecraft.py** en s'aidant des informations ci-dessus.


```
1 for i in range(0, 100):  
2     grille[i] = Obsidienne()  
3 for i in range(100, 300):  
4     grille[i] = Terre()  
5 for i in range(300, 900):  
6     grille[i] = Roche()
```

Code 13 – Créer les blocs

Correction

```
1 for i in range(0, 100):  
2     grille[i] = Obsidienne()  
3 for i in range(100, 300):  
4     grille[i] = Terre()  
5 for i in range(300, 900):  
6     grille[i] = Roche()
```

Code 13 – Créer les blocs

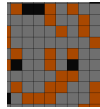


FIGURE 4 – Le moteur assurera la représentation graphique des objets.

Affichage

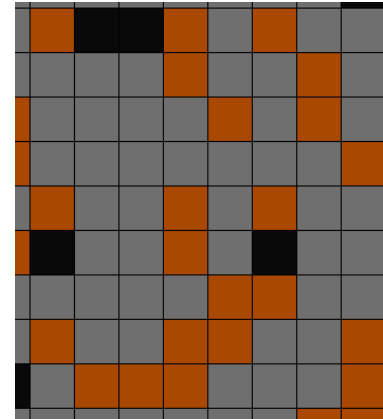


FIGURE 4 – Le moteur assurera la représentation graphique des objets.

```
1 for i in range(5):
2     outils_poses[moteur.donner_coordonnees()] = Pioche
3     ("wood_pickaxe")
4     outils_poses[moteur.donner_coordonnees()] = Pioche
5     ("diamond_pickaxe")
6     outils_poses[moteur.donner_coordonnees()] = Pelle()
```

Code 14 – Placer les outils

Correction

```
1 for i in range(5):
2     outils_poses[moteur.donner_coordonnees()] = Pioche
3     ("wood_pickaxe")
4     outils_poses[moteur.donner_coordonnees()] = Pioche
5     ("diamond_pickaxe")
6     outils_poses[moteur.donner_coordonnees()] = Pelle()
```

Code 14 – Placer les outils

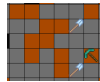


FIGURE 5 – Le moteur s'occupera d'afficher les outils.

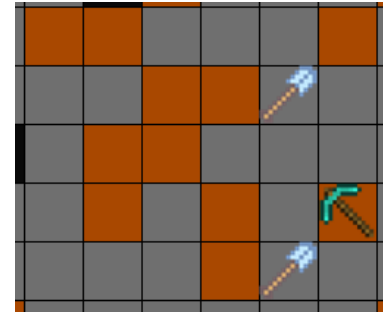


FIGURE 5 – Le moteur s'occupera d'afficher les outils.

Code complet

Le code complet est accessible [ici](#).