

# **Magic Wand, Minecrabracadabra**

Auteurs: **Albert and Ludovic**

Affiliation: **Gymnase du Bugnon**

Année: **2017-2018**

Classe: **OC-Informatique**

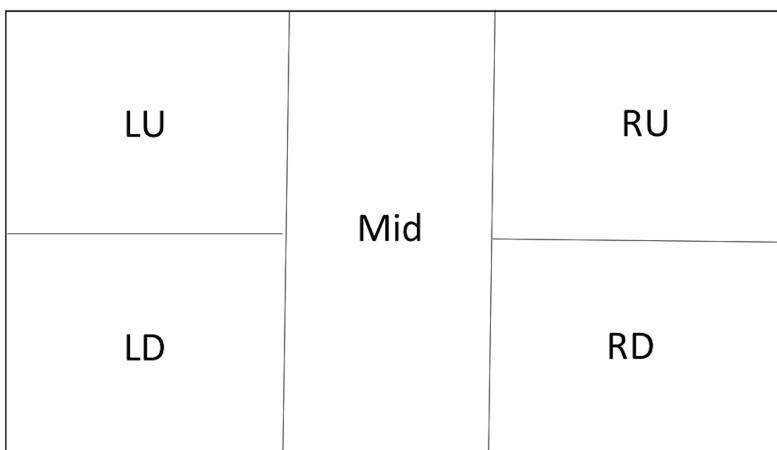
## **Fonctionnement**

**Principe:** Afin d'interagir avec le Raspberry Pi pour pouvoir utiliser les fonctions Minecraft incluses dans python, nous utilisons une baguette sur laquelle nous avons fixés deux cercles un rouge et un bleu. Grace à une caméra, nous détectons si un objet de couleur rouge ou bleu se situe dans un des cadans de l'image perçue par la caméra.

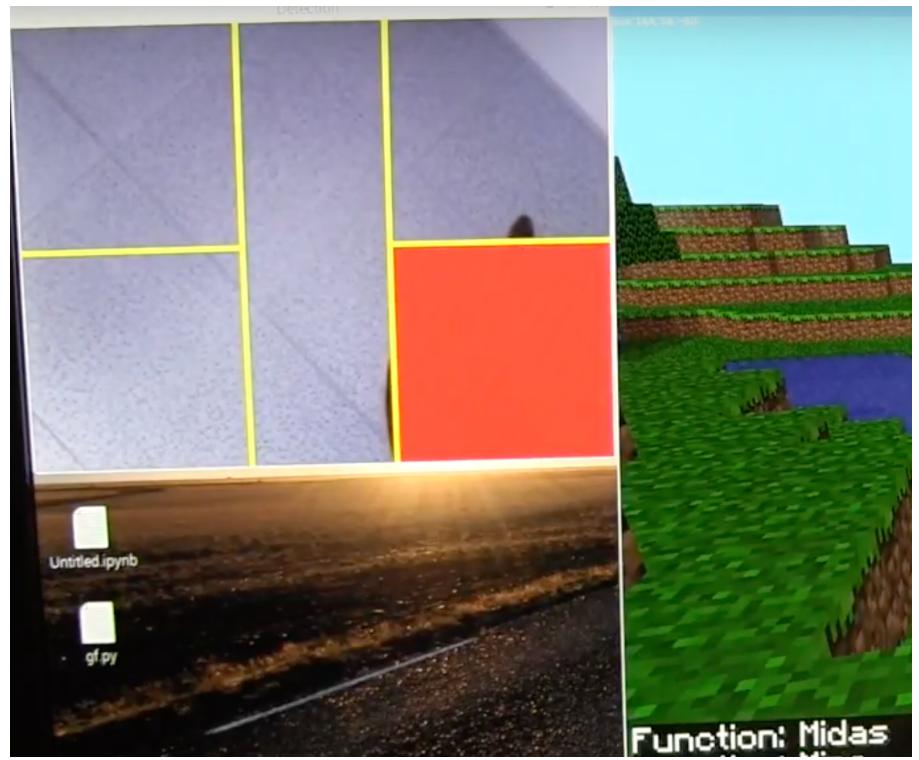
Selon le cadran et la couleur nous définissons la fonction utilisée, parmi 4, Mine, Midas, House et Bridge. Il est aussi possible de choisir 2 matériaux différents ainsi que 2 tailles prédéfinies, petit et grand. Une fois tous les paramètres sélectionnés, si le Raspberry Pi détecte un objet bleu au centre de l'image, il lance la fonction. Si c'est un objet rouge il arrête la caméra et le programme.

Illustrations: La vidéo de présentation pour le concours bugnplay, est une bonne illustration du fonctionnement de Magic Wand: <https://www.youtube.com/watch?v=HtxRfrnMXhs>  
<https://www.youtube.com/watch?v=HtxRfrnMXhs>

La détection se fait dans 5 cadans (LU, LD, Mid, RU,RD):



Lorsque qu'un cadran détecte qu'il y a la baguette en lui (rouge ou bleu), il s'encadre de cette couleur. Et, si la couleur est détectée assez longtemps le paramètre correspondant s'execute et il se colorie:



## Historique

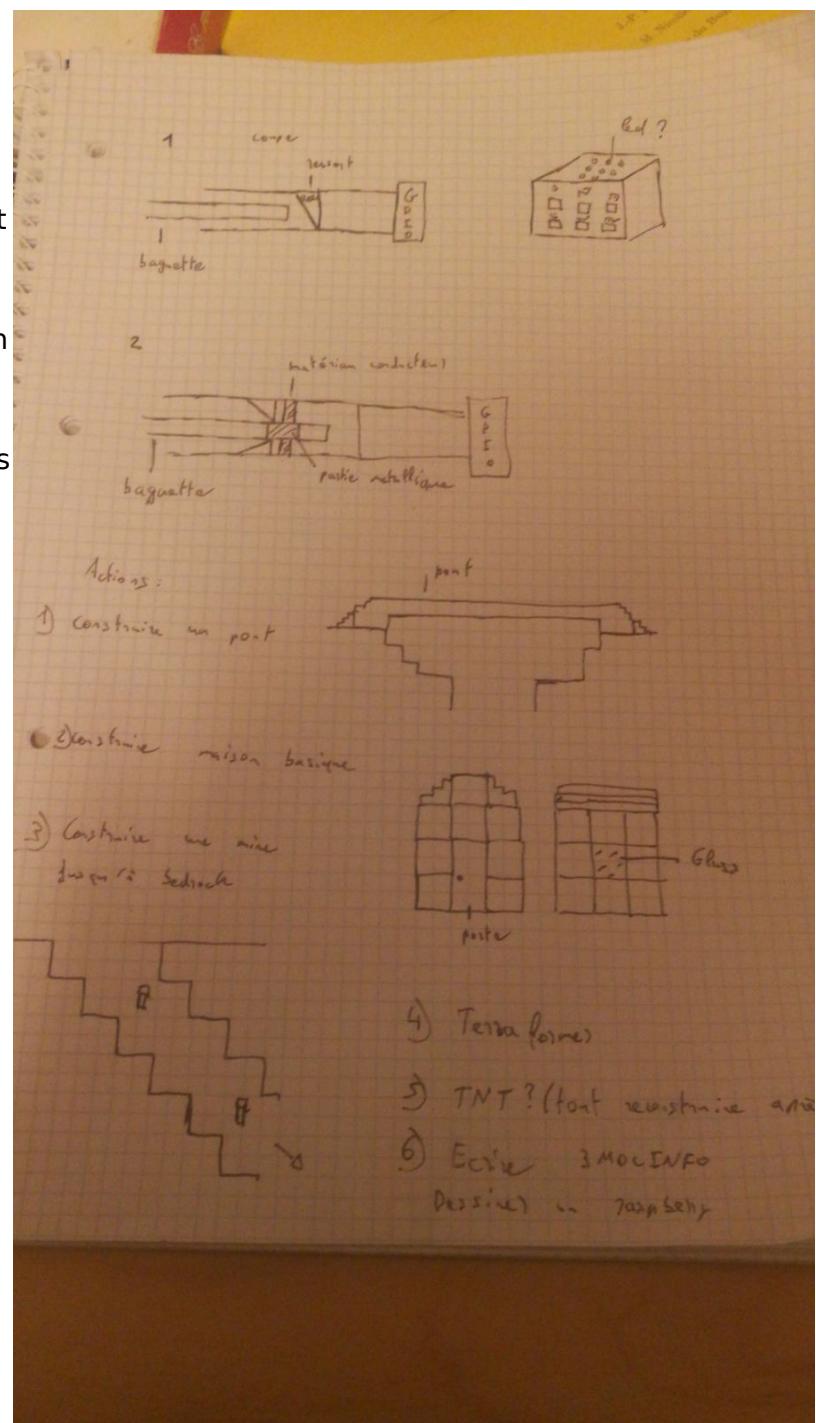
Ce projet a débuté pendant le mois de décembre 2017. Il s'agissait alors du premier des deux projets que nous devions créer pendant le second semestre de l'année scolaire. Le but premier du projet était de lier Minecraft.pi au Raspberry.pi en utilisant le langage python et un support physique. Monsieur Holzer a alors suggéré une baguette magique par exemple et la base du projet a germé de cette proposition. Le projet Minecrabracadabra était né.

La première idée fut donc d'utiliser une baguette et une boîte afin d'utiliser des fonctions que nous allions créer avec python. Parmi ces premières idées de fonctions, la mine la maison et le pont furent conservées bien que les fonctions aient bien changées depuis leur première création. Ci-contre le premier croquis de la boîte et les idées de fonctions à utiliser:

Parmi les 2 idées de baguette possibles:

- La première consistait en une simple baguette qui poussait un bout de métal accroché à un ressort afin de laisser le courant électrique circuler et faire office de bouton avec le GPIO.
- La seconde idée, celle qui fut conservée était une baguette avec un embout métallique qui permet au courant électrique de circuler. Cette idée fut retenue car elle était bien plus simple à mettre en place.

Voici donc le premier prototype de baguette créé, elle est plus que rudimentaire et bien trop fragile pour être utilisée mais elle a servi à illustrer la base de notre projet:



Rapidement un deuxième prototype fut créé (à gauche), il devait aussi inclure des boutons sur la baguette elle-même, afin de sélectionner des paramètres et lancer la fonction avec le courant électrique.

Les croix rouges représentent l'emplacement des futurs boutons.

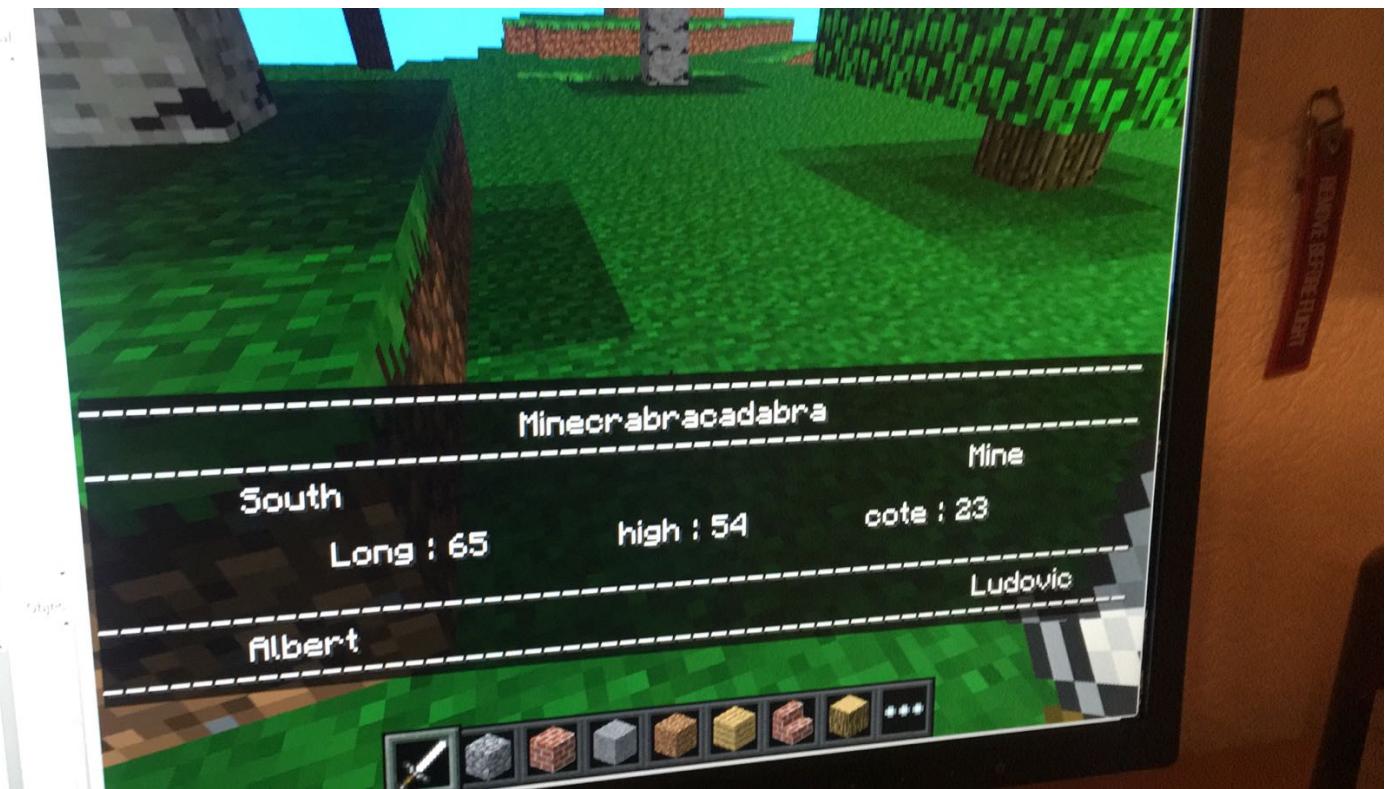
Puis cette même baguette solidifiée et avec un embout en métal: (à droite)



De plus la boite prenait forme elle aussi et bien que nous n'avions pas encore choisi quelle possibilités utiliser pour faire passer un courant la boite était construite:



Pendant ce temps nous travaillions aussi sur l'interface dans Minecraft.pi que nous allions utiliser pour le projet. Bien qu'elle ne fut pas conservée il est intéressant de voir une première ébauche.



Afin de relier les boutons au GPIO du Raspberry.pi nous avions pensé utiliser de long fils électriques détachables afin de ne pas trop nous encombrer mais Monsieur Holzer nous a demandé d'intégrer les connaissances que nous avions acquises sur OpenCV dans notre projet. Ce fut une bonne idée. Nous devions donc utiliser une caméra dans le projet et le prototype de baguette métallique fut abandonné.

Nous avons donc décidé de détecter un objet de couleur accroché à une baguette afin de conserver l'idée de base du projet tout en utilisant une caméra et OpenCV. Voici la baguette en question dont l'une des face est rouge et l'autre bleu:



Plus besoin de percer la boite afin d'installer quelque chose permettant un courant électrique nous avons donc pu entamer le début de la décoration de la boite et nous avons réfléchi à comment installer le Raspberry.pi et la caméra dans cette boite. Finalement nous avons utilisé des bande velcro pour accrocher le Raspberry.pi et la caméra.



Et voici une première idée de fixation du Raspberry.pi. Malheureusement le cable HDMI posait problème lors de la fermeture de la boite:

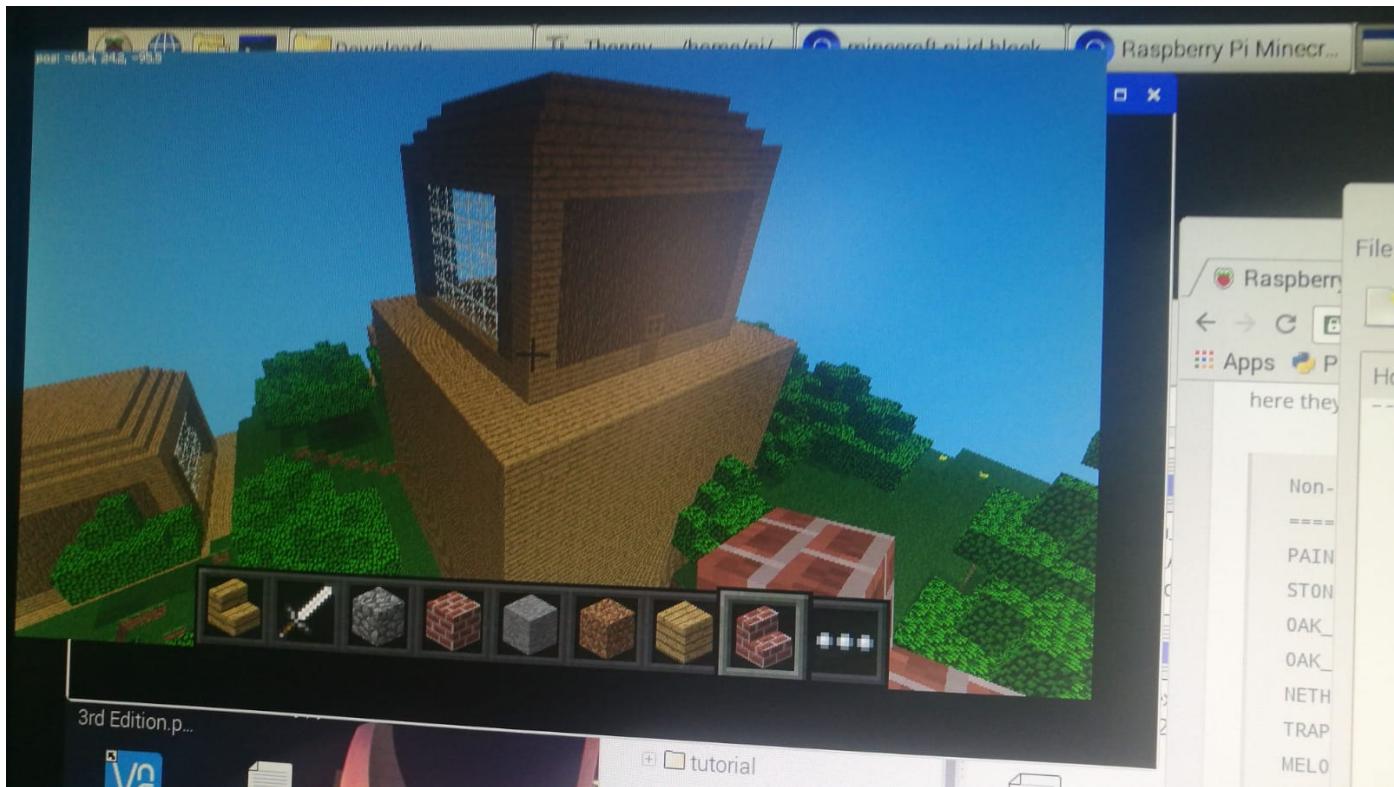


Nous avons donc percé la boîte afin de permettre à tous les câbles de rentrer sans problème et nous avons fixé avec de la bande adhésive la caméra sur la partie supérieure de la boîte. Le RaspberryPi est lui aussi scotché à la boîte. Les vélcros sont maintenant que preuve d'évolution.



Quant aux fonctions elles ont toutes évolué petit à petit sans de grand changement à l'exception de la maison qui est passée par de nombreux essais.





## Les fonctions de construction

### Bridge.py

bridge(x, y, z, blocktype) est une fonction qui crée un pont dans la direction nord du joueur. Ce pont peut être en bois ou en pierre selon l'argument choisi.

La longueur de ce pont n'est pas réglable car la longueur du pont est calculée selon le nombre de blocs vides au nord du joueur.

De plus la longueur minimum du pont est de 5 blocs.

La longueur du pont est définie par la fonction lengthbridge(x, y, z) qui appelle la fonction isvoid(x, y, z) afin de calculer le nombre de blocs vides au Nord du joueur.

La fonction bridge(x, y, z, blocktype) utilise quant à elle les fonctions first\_stage(x, y, z, l, stairs, block) et second\_stage(x, y, z, l, stairs, block) afin de rendre le code plus lisible.

Le matériau choisi implique un id de blocs pour les escaliers et les blocs:

In [1]:

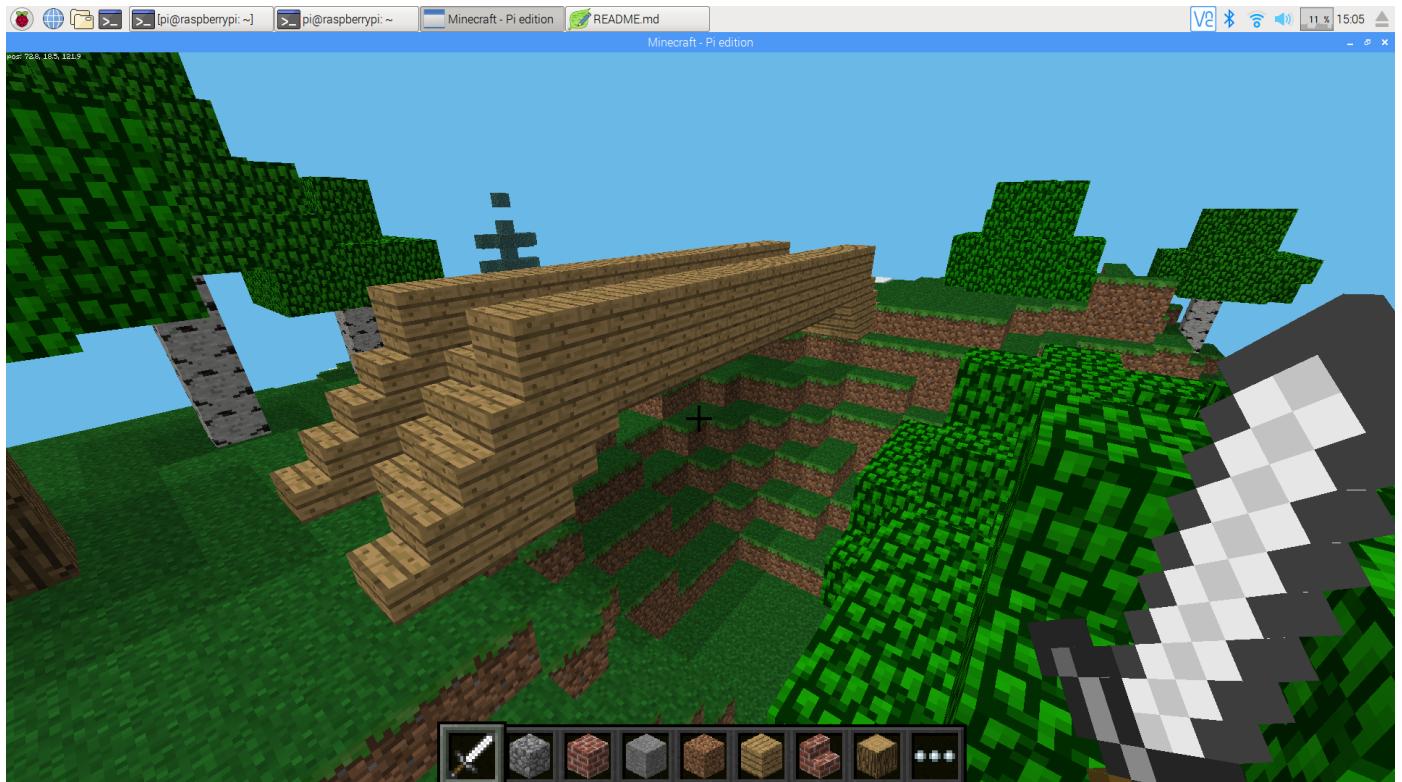
```
if blocktype is "COBBLESTONE":  
    stairs = 67  
    block = 4  
if blocktype is "WOOD":  
    stairs = 53  
    block = 5
```

```
---  
NameError Traceback (most recent call last)  
st)  
<ipython-input-1-b5039a8893cd> in <module>()  
----> 1 if blocktype is "COBBLESTONE":  
      2     stairs = 67  
      3     block = 4  
      4 if blocktype is "WOOD":  
      5     stairs = 53  
  
NameError: name 'blocktype' is not defined
```

Le point de départ du pont est trouvé par la partie ci-dessous de la fonction `bridge(x, y, z, blocktype)` :

In [ ]:

```
while True:  
    if isvoid(x, y-1, z-i) is True or isvoid(x+1, y-1, z-i) is True:  
        xs, ys, zs = x, y-1, z-i  
        break  
    else:  
        i += 1
```



## House.py

house(x, y, z, material, n) est une fonction qui crée une maison autour du joueur avec une porte orientée vers le Nord.

Deux types de matériaux sont disponibles, la pierre et le bois.

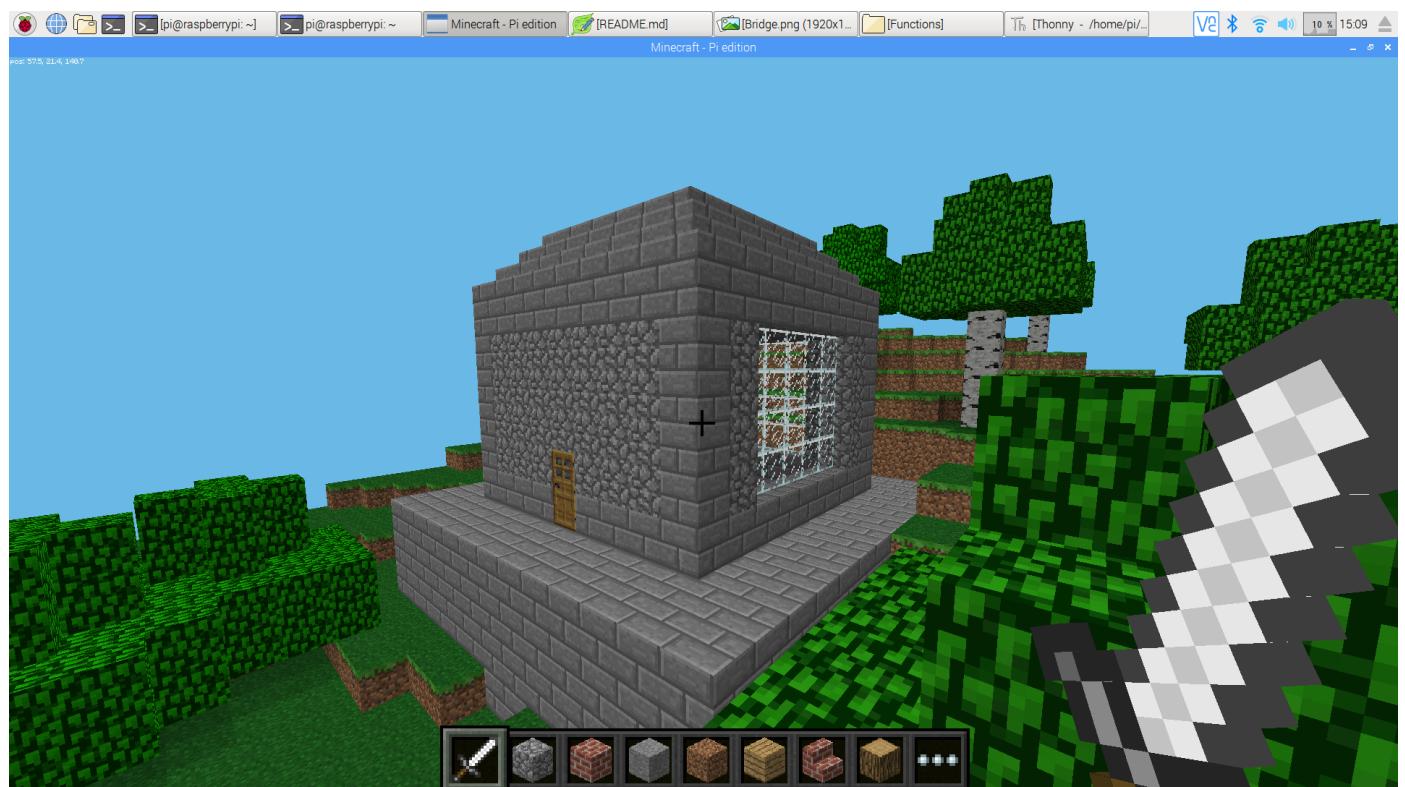
De plus un espace vide de deux blocs est créé autour de la maison et un bloc de fondation se trouve sous la maison.

Tout comme Bridge.py le choix de matériaux implique des id de blocs différents:

In [ ]:

```
if material == "WOOD":  
    M1 = 5  
    M2 = 17  
    stairs = 53  
if material == "COBBLESTONE":  
    M1 = 98  
    M2 = 4  
    stairs = 109
```

afin de créer un bloc vide avant de construire la maison house(x, y, z, material, n) utilise clear\_space(x, y, z, range)



## Midas.py

Midascube(x, y, z, size, blockid) est une fonction qui transforme les blocs non-vide autour du joueur.

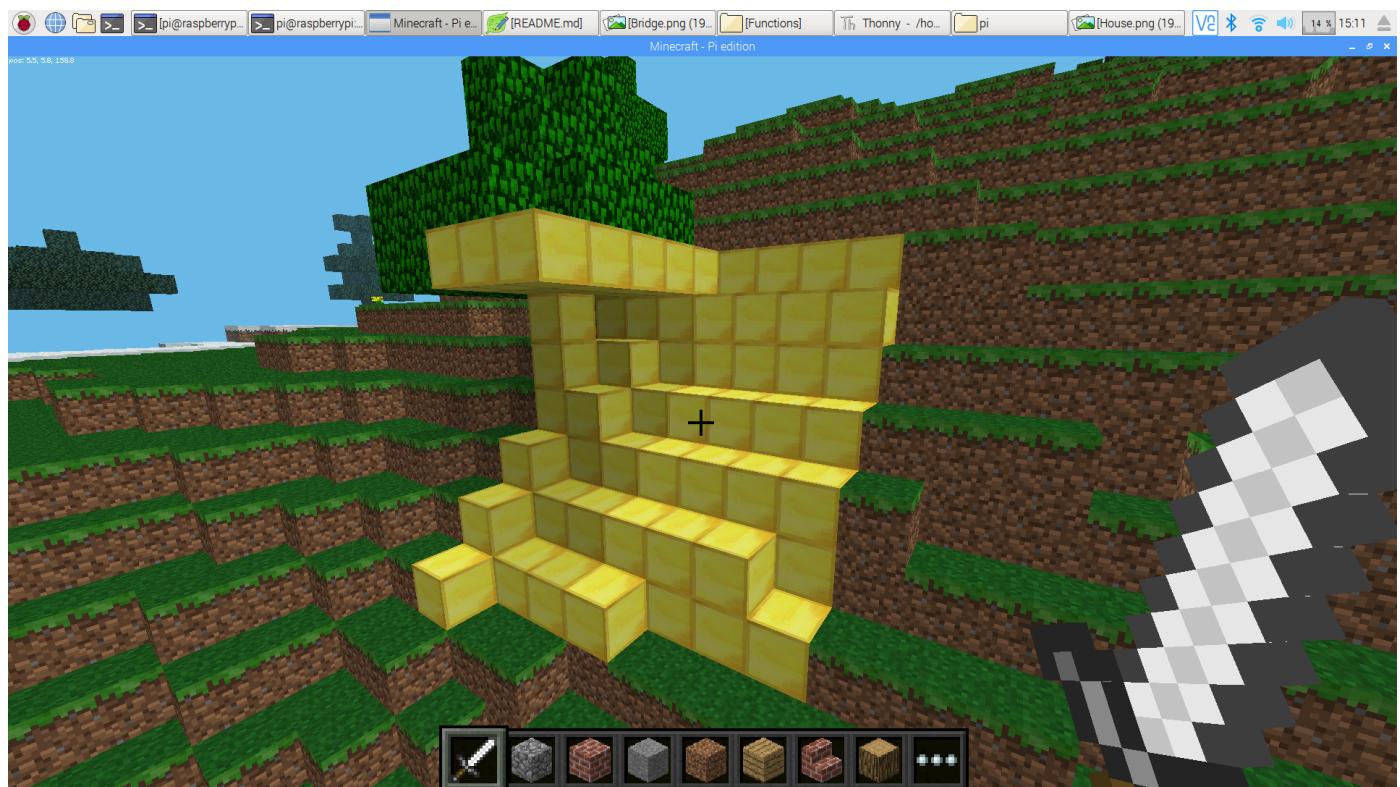
Les blocs transformés peuvent être de n'importe quel matériau mais la taille du bloc autour du joueur

doit etre impaire.

La transformation se fait colonne apres colonne donc la fonction prend relativement beaucoup de temps.

La fonction `Midascube(x, y, z, size, blockid)` appelle `Midassquare(x, y, z, size, blockid)` qui appelle

`Midasline(x, y, z, size, blockid)` qui appelle `isAir(x, y, z)` afin de transformer selon les blocs non-vides

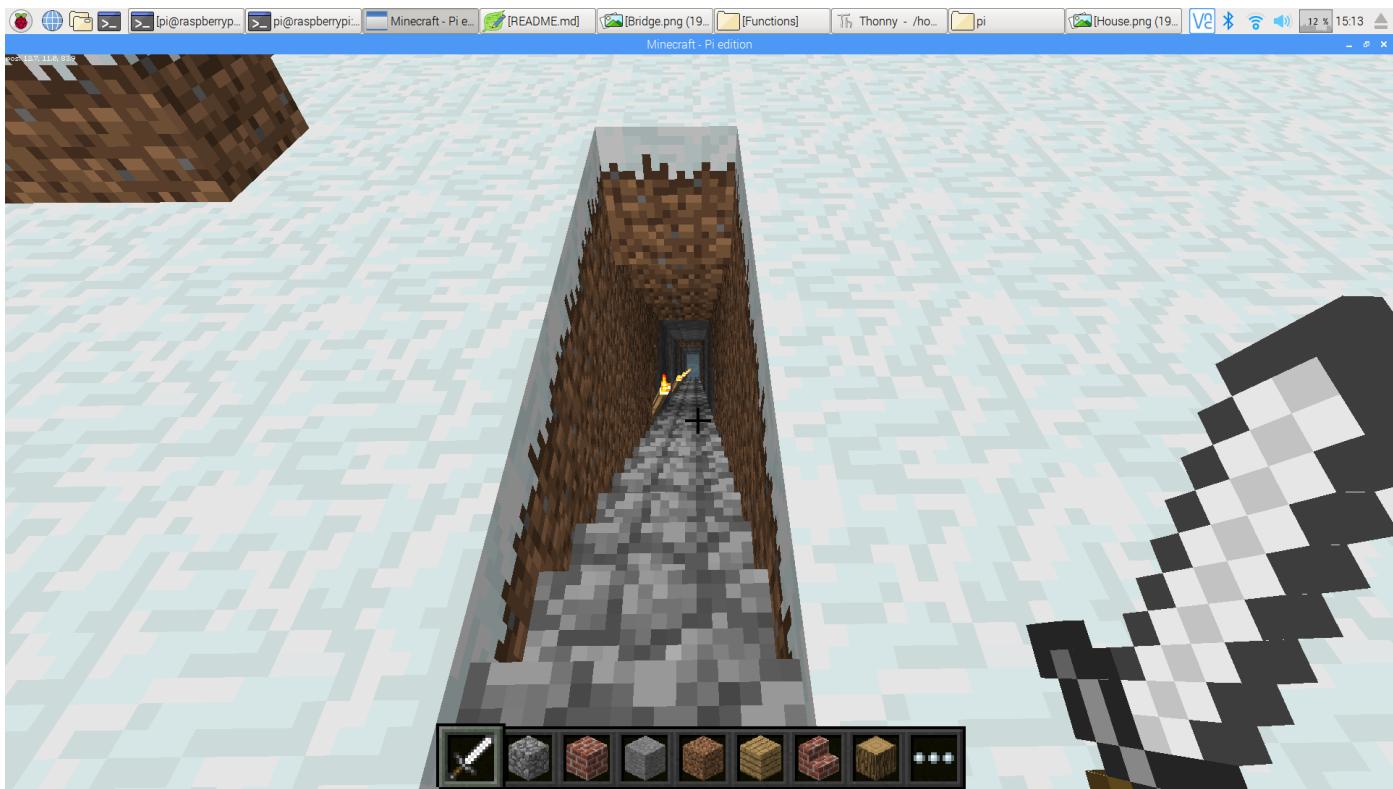


## Mine.py

`Mine(x, y, z, direction, size)` est une fonction qui cree une mine en escalier rectiligne vers le Nord ou le Sud.

Des torches sont posées dans la mine à intervalle de 5 blocs.

La fonction `time.sleep()` est appellée par `Mine(x, y, z, direction, size)` afin d'imposer un délai et laisser d'éventuels blocs de sable ou de gravier tomber dans Minecraft.pi avant que la fonction ne vérifie si ces blocs sont tombés en appelant `isAirz(z, y)`.



## Les fonctions du programme Magic\_Wand.py

Ceci est une description rapide des fonctions. Pour avoir tous les détails, consulter le code qui est totalement annoté. ([https://github.com/Bugnon/minecraft/blob/master/Magic\\_Wand/Magic\\_wand.py](https://github.com/Bugnon/minecraft/blob/master/Magic_Wand/Magic_wand.py)) ([https://github.com/Bugnon/minecraft/blob/master/Magic\\_Wand/Magic\\_wand.py](https://github.com/Bugnon/minecraft/blob/master/Magic_Wand/Magic_wand.py))

Vu l'historique de notre programme nous avons gardé le terme de boutons pour désigner chaque case de chaque couleur.

In [ ]:

```

def button_fct_pressed(n):
    """ Test si le bouton doit etre
    - 0    le bouton n'est pas active
    - ' ' le bouton est en attente d'execution
    - 1    le bouton est enclanche
    - 'e'  le bouton vient d'etre enclanche / attente de fin de coloriage + temps
           d'attente avant reexecusion

Colorie les cases en rouge ou bleu : pleinement si : 1 ou 'e'
encadre si ' '

n: int entre 0 et 9
"""
# Enregistrement en string le bouton sur lequel on "travail" pour pouvoir
# par la suite enregistrer des valeurs dans les listes.
nb = 'button' + str(n+1)

# On commence par travailler par les boutons "bleus" (n < 5)
if n < 5:

    # Si le boutons est enclanche ou entrain d'enclanche et que le temps
    # d'attente de fin d'execution n'est pas fini, la case se fait colorier
    # en bleu et si le bouton valait 1 il est transforme en 'e'
    if ((flags[nb] == 1 or flags[nb] == 'e')
        and time_flag[nb] > time.time()):
        liste[n] = Colorize(liste[n], BLUE, "FULL")
        flags[nb] = 'e'
    return

    # Si le bouton est en attente de fin de coloriage, et que le temps est
    # depasse la case est coloriee une derniere fois et le bouton et a
    # nouveau desactive
    if flags[nb] == 'e' and time_flag[nb] < time.time():
        liste[n] = Colorize(liste[n], BLUE, "FULL")
        flags[nb] = 0

    # Si le bouton n'est pas active et que la baguette est detectee,
    # un cadre se met aux bords de la case et le bouton passe en attente
    # d'execution. Le but est d'avoir un temps durant lequel la baguette
    # doit etre detectee afin que l'on aie le temps de bien choisir ce
    # qu'on veut faire
    if regions_blue[n] == True:
        liste[n] = Colorize(liste[n], BLUE)
        if flags[nb] == 0 and time_flag[nb] < time.time():
            flags[nb] = ' '
            time_flag[nb] = waiting_time + time.time()
        return

    # Si la baguette est detectee depuis assez longtemps, le bouton
    # passe en mode actif est la case est coloriee
    if flags[nb] == ' ' and time_flag[nb] < time.time():
        flags[nb] = 1
        time_flag[nb] = waiting_time + time.time()
        liste[n] = Colorize(liste[n], BLUE, "FULL")
    return

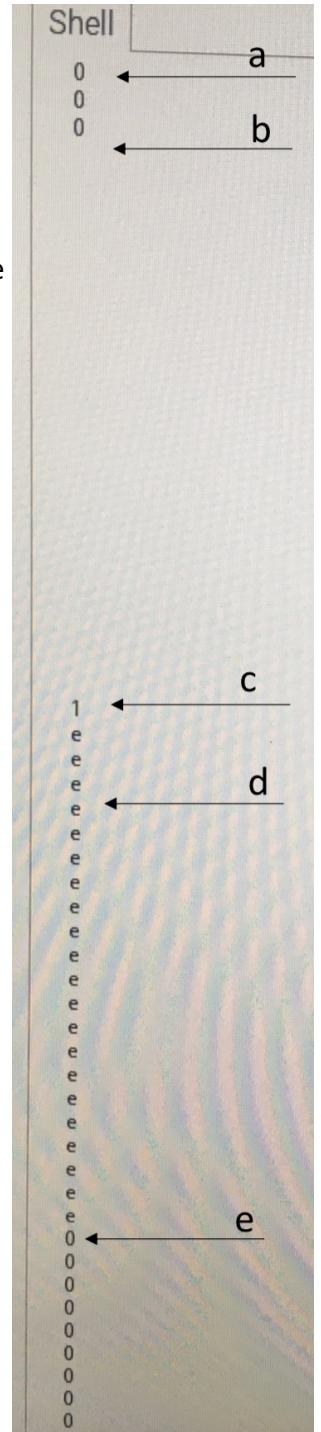
    # Si la baguette n'est pas detectee, le bouton passe en mode inactif
    if regions_blue[n] == False:
        flags[nb] = 0
    return

```

Comme promis dans le code voici une illustration de cette fonction. On voit le "statut" d'un des boutons au fil et à mesure que le temps s'écoule.

On peut séparer la détection en plusieurs étapes:

- en a, la baguette n'est pas détectée, il ne se passe donc rien.
- en b, la baguette est détectée, c'est la que la case a un cadre colorié, il ne reste plus qu'à attendre le temps de détection avant exécution choisi au préalable
- en c, la baguette est toujours détecté et le temps d'attente a été atteint. le "bouton" est comme appuyé, les fonctions sont executées
- en d, maintenant que les fonctions on été exécutée, le "bouton" passe en mode attente après execution (pour pas que les fonctions s'exécutent trop rapidement à la suite).
- en e, la baguette n'est plus là et le temps d'attente après execution est révolu



Maintenant revenons à l'étape c, quand la valeur vaut 1 voilà ce qui s'exécute:

In [ ]:

```
def exefct():
    """Pour chaque bouton, effectue des changements s'il est actif (=1)
    en changement les parametres 1 et 2 (materiaux et taille)
    Peut aussi changer start (pour arreter le programme)
    De plus il execute les fonctions qui execute les fonctions (fctMyfonction)
    De plus, c'est cette fonction qui mets les messages dans minecraft:
        - You need to choose a function!
        - Working, wait...
        - Work done!
    """
    global mc_funct, param1, param2, start

    if flags['button1'] == True: # 1 = True = beeing pressed
        param1 = True
        mc.postToChat("Size: Big")
```

Maintenant, voici la fonction qui s'occupe de la détection des couleurs: ( redmin et bluemim sont définis au préalable)

In [ ]:

```
def detect_colour(img, colour):
    """ Detection des couleurs qui sont entre lower et upper dans chaque case
    qui est entre lower et upper (in the range [lower, upper]).

    enregistre en True/False dans regions_red ou regions_blue les cases ou se
    trouve la couleur voulue

    img: image
    couleur: "RED"/"BLUE"
    """
    if colour == "RED":
        lower = lowerR
        upper = upperR
    if colour == "BLUE":
        lower = lowerB
        upper = upperB

    lower = np.array(lower, dtype="uint8")
    upper = np.array(upper, dtype="uint8")

    # cree un mask qui enleve tout ce qui n'est pas entre lower et upper
    mask = cv2.inRange(img, lower, upper)

    # Enlever les '#' pour avoir un apercu des operations
    # output = cv2.bitwise_and(img, img, mask=mask)
    # cv2.imshow("Mask", mask)
    # cv2.imshow("Detect color", np.hstack([img, output]))

    # Separation de l'image en partie (dim= LU, LD, Mid, RU, RD) => liste
    LUm = mask[0:120, 0:120]
    LDm = mask[120:240, 0:120]
    Midm = mask[0:240, 120:200]
    RUm = mask[0:120, 240:]
    RDm = mask[120:240, 240:]
    regions = [LUm, LDm, Midm, RUm, RDm]

    # enregistre (True/False) dans les listes predifinies en 2.4 afin qu'on
    # puisse utiliser pour executer les fonction dans 'button_fct_pressed(n)'
    if colour == "RED":
        for i in range(5):
            regions_red[i] = np.average(regions[i]) > redmin
            # print(regions_red) # si on veut voir la liste
    if colour == "BLUE":
        for i in range(5):
            regions_blue[i] = np.average(regions[i]) > bluemmin
            # print(regions_blue) # si on veut voir la liste
```

Et celle permettant de colorier les cases:

In [ ]:

```
def Colorize(img, colour, FULL=5):
    """Colorie le cadre de l'image (img) avec la couleur (colour).
    Taille du cadre = FULL (en px).
    Si FULL est égal à -1 ou "FULL" l'image est complètement coloriée.

    img: image
    colour: tuple
    Full: int ou "FULL"
    """
    a = img.shape[1]
    b = img.shape[0]
    if FULL == "FULL":
        FULL = -1
    return cv2.rectangle(img, (0, 0), (a, b), colour, FULL)
```

Pour arrêter le programme, on a deux possibilité, soit on appuie sur la touche `q` du clavier, soit on mets la baguette côté rouge sur Mid:

In [ ]:

```
if cv2.waitKey(1) & 0xFF == ord("q"):
    break

if start == -1:
    break
```

## Conclusion

Ce projet est une bonne conclusion de l'année, Nous avons pu utiliser tous les domaines que nous avons étudié durant l'année (Markdown, python, Jupyter notebook, openCV, etc.) Ça a été un plaisir de bricoler et bidouiller afin d'arriver à nos fins, nous sommes très satisfait et heureux du résultat; nous espérons que vous aussi. Nous tenons à remercier Monsieur Holzer pour son cours durant l'année et son temps d'investissement consacré pour nous.

Autres informations sur le projet: Le code des fonctions de constructions:

[https://github.com/Bugnon/minecraft/tree/master/Magic\\_Wand/Functions](https://github.com/Bugnon/minecraft/tree/master/Magic_Wand/Functions)

([https://github.com/Bugnon/minecraft/tree/master/Magic\\_Wand/Functions](https://github.com/Bugnon/minecraft/tree/master/Magic_Wand/Functions)) Le code qui gère tout:

[https://github.com/Bugnon/minecraft/blob/master/Magic\\_Wand/Magic\\_wand.py](https://github.com/Bugnon/minecraft/blob/master/Magic_Wand/Magic_wand.py)

([https://github.com/Bugnon/minecraft/blob/master/Magic\\_Wand/Magic\\_wand.py](https://github.com/Bugnon/minecraft/blob/master/Magic_Wand/Magic_wand.py)) La participation au

concours Bugnplay:<http://bugnplay.ch/pms/fr/minisite/5297/>

(<http://bugnplay.ch/pms/fr/minisite/5297/>)