Création du GUI

I/ Dans le fichier *main.py*

1/ Dans la méthode *`setup(self)`* de la classe *`My_Game`*, **écrire** à l'endroit indiqué le code à droite qui créé une instance de la classe *`Gui`*.

```
# Création de l'IHM
self.gui = Gui(self)
self.gui.setup()
```

2/ Dans la méthode *`on_draw(self)`* de la classe *`My_Game`*, **écrire** à l'endroit indiqué l'instruction permettant à l'attribut *gui* d'appeler la méthode *`draw(self)`*.

```
# Affichage de l'IHM (GUI)
##### A COMPLETER ####
```

II/ Dans le fichier *gui.py*

Voici la description du constructeur de la classe *`Gui`* :

La **création** de l'attribut *game* permet d'accéder à tous les **attributs** de la classe *`My_Game`*, par exemple, l'instruction *self.game.player* permet d'accéder aux attributs du joueur 😊.

Menu général du jeu : bouton ***Quit*** ici.
Affiche à l'écran les caractéristiques du joueur.
Affiche celles du monstre cliqué (s'il y en a un), sinon rien ne doit s'afficher.

```
class Gui() :
    def __init__(self,game) :
        # Pointeur sur l'instance de la classe `My_Game`
        self.game = game
        # Permet de générer un GUI dans la fenêtre de jeu
        self.game.manager = arcade.gui.UIManager()
        self.game.manager.enable()

        # Menu du jeu
        self.menu_bar = None
        # Affiche les caractéristiques principales du joueur
        self.player_box = None
        # Affiche les caractéristiques principales du mob cliqué
        self.clicked_mob_box = None
```

Voici la description de la méthode *create_menu_bar(self)* qui affiche le menu du jeu. Pour l'instant, il n'y a que le bouton ***Quit*** qui sera inséré (mais on peut imaginer d'autres options) :

Création de la box qui contiendra tous les éléments du menu (sous forme de boutons ici) : l'instruction *vertical=False* permet de l'orienter à l'horizontale.
Création d'un bouton *Quit*, instance de la classe *`QuitButton`*.

Ajout du bouton précédent dans le menu
Ajout de ce menu à l'interface : il sera en bas (*bottom*) à droite (*right*) de l'écran.

```
# Création du menu du jeu
def create_game_menu(self) :
    self.menu_bar = arcade.gui.UIBoxLayout(vertical = False)

    # Quitter le jeu
    quit_button = QuitButton(text="Quit", width=50)
    # Associe ce bouton à la fenêtre à fermer
    quit_button.set_window(self.game)

    self.menu_bar.add(quit_button)
    self.game.manager.add(arcade.gui.UIAnchorWidget(anchor_x="right",
            anchor_y="bottom", child=self.menu_bar))
```

Zoom sur la dernière instruction :

```
self.game.manager.add(arcade.gui.UIAnchorWidget(anchor_x="right",
        anchor_y="bottom", child=self.menu_bar))
```

Elle permet deux choses :

- **d'ajouter** le menu du jeu via l'argument *self.menu_bar*. On remarquera que le nom du paramètre *child* est bien choisi puisqu'une GUI se représente sous forme d'un arbre,
- **d'ancrer** ce menu en bas (<u>en abscisses</u> avec *anchor_x*) et à droite (<u>en ordonnées</u> avec *anchor_y*) ce menu.

Placement des menus en fonction des valeurs de *anchor_x* et *anchor_y* :

Les menus sont placés de manière **relatives** (grâce à des données directionnelles) et non **absolues**, ce qui permet de s'adapter à la taille de la fenêtre du jeu.

<u>*Placement du menu en fonction des valeurs*</u> de ***anchor_x*** et de ***anchor_y***

anchor_x : *left* ***anchor_y*** : *top*	***anchor_x*** : *center* ***anchor_y*** : *top*	***anchor_x*** : *right* ***anchor_y*** : *top*
anchor_x : *left* ***anchor_y*** : *center*	***anchor_x*** : *center* ***anchor_y*** : *center*	***anchor_x*** : *right* ***anchor_y*** : *center*
anchor_x : *left* ***anchor_y*** : *bottom*	***anchor_x*** : *center* ***anchor_y*** : *bottom*	***anchor_x*** : *right* ***anchor_y*** : *bottom*

3/ **Recopier** la méthode précédente *create_game_menu(self)*.

```python
# Création du menu du jeu (réduit ici au bouton Quit)
def create_game_menu(self) :
    # Création d'une box horizontale
    ########## A COMPLETER #########

    # Quitter le jeu
     ########## A COMPLETER #########

    # Associe ce bouton au jeu à quitter (game)
     ########## A COMPLETER #########

    # Ajut du bouton dans la box et placement en bas/ droite de l'écran
     ########## A COMPLETER #########
```

Voici la description de la méthode *setup(self)* qui permet d'afficher le <u>menu interactif</u> avec l'utilisateur et la <u>description</u> du joueur :

Création de l'interface qui sera rattachée à la fenêtre de jeu.

Appel à la méthode précédente créant le menu du jeu (ici composé d'un bouton *Quit*).

```python
# IHM de base
def setup(self) :
    self.game.manager = arcade.gui.UIManager()
    self.game.manager.enable()

    # Ajout du menu du jeu
    self.create_game_menu()
```

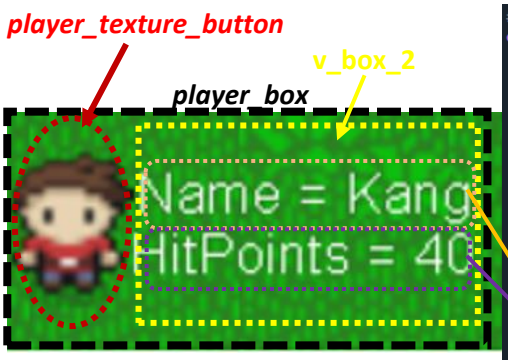
4/ **Recopier** la méthode précédente *setup(self)*.

<u>POUR SPYDER UNIQUEMENT</u> : ne pas oublier de relancer le kernel avant d'exécuter le programme.
Exécuter le programme, un bouton ***Quit*** doit s'afficher en bas à droite et fonctionner correctement.

<u>Remarque</u> : ne pas hésiter à modifier le bouton *Quit* (taille, texte …) et son placement sur la carte.

<u>Appeler le professeur pour vérification</u>

Voici la description imagée de la méthode *create_player_box(self)* :




```python
# Caractéristiques principales du joueur
def create_player_box(self) :
    # Création de la box principale (horizontale)
    self.player_box = arcade.gui.UIBoxLayout(vertical = False)

    # Création d'un bouton texturé pour accueillir une image du joueur
    player_texture_button = arcade.gui.UITextureButton(texture = self.game.player.textures[0][0],
            width = 40, height = 68)

    self.player_box.add(player_texture_button) # Ajout dans player_box du bouton

    # Création d'une box verticale
    v_box_2 = arcade.gui.UIBoxLayout()

    # Création de deux labels pour le nom et le nombre d'HP du joueur
    hp_label_1 = arcade.gui.UILabel(text = "Name = " + str(self.game.player.attributes["Name"]))
    hp_label_2 = arcade.gui.UILabel(text = "HitPoints = " + str(self.game.player.attributes["HitPoints"]))

    # Ajout des labels dans v_box_2
    v_box_2.add(hp_label_1)
    v_box_2.add(hp_label_2)

    self.player_box.add(v_box_2) # Ajout de v_box_2 dans player_box
```

6/ **Recopier** la méthode *create_player_box(self)* ci-dessus.

7/ **compléter** la méthode *create_clicked_mob_box(self, clicked_mob)* en suivant la même progression.

```python
# Caractéristiques du monstre cliqué
def create_clicked_mob_box(self, clicked_mob) :
    # Création de la box principale (horizontale)
    self.clicked_mob_box = arcade.gui.UIBoxLayout(vertical = False)

    # Création d'un bouton texturé pour acueillir l'image du monstre clické
    ##### A COMPLETER #####

    ##### A COMPLETER #####                                    # Ajout dans la box

    # Création d'une box verticale
     ##### A COMPLETER #####

    # Création de deux labels pour le nom et le nombre d'HP du monstre clické
     ##### A COMPLETER #####

    # Ajout des labels dans v_box_2
     ##### A COMPLETER #####

    # Ajout de v_box_2 dans la box principale
     ##### A COMPLETER #####
```

8/ **Compléter** désormais la méthode *setup(self)* pour qu'elle prenne en compte le descriptif du joueur :

Création de l'interface qui sera rattachée à la fenêtre de jeu.

Création du menu du jeu (ici composé d'un simple bouton *Quit*).

Création du descriptif du joueur.

Si un monstre a été sélectionné, on va afficher ses **caractéristiques**.

Ancrage en haut à gauche du descriptif du joueur.

```python
# IHM de base
def setup(self) :
    self.game.manager = arcade.gui.UIManager()
    self.game.manager.enable()

    # Ajout du menu du jeu
    self.create_game_menu()

    # Ajout des caractéristiques du joueur
    self.create_player_box()

    mobs_box = arcade.gui.UIBoxLayout(vertical = False)
    mobs_box.add(self.player_box)

    if self.game.player.clicked_mob is not None :
        self.create_clicked_mob_box(self.game.player.clicked_mob)
        mobs_box.add(self.clicked_mob_box)

    self.game.manager.add(arcade.gui.UIAnchorWidget(anchor_x="left",
            anchor_y="top", child=mobs_box))
```

Exécuter le programme, le descriptif du joueur doit s'afficher en bas à droite en plus du bouton *Quit* (qui doit toujours fonctionner correctement 😊).

Appeler le professeur pour validation

III/ Adaptation entre la GUI et les interactions du joueur avec le RPG

Jusqu'à présent, la GUI affichée était statique et indépendante des actions du joueur. Pour être efficace, une GUI doit bien sûr s'adapter aux choix de l'utilisateur tout en respectant également un souci de design et de cohérence.
Le jeu RPG proposé permet d'afficher les **caractéristiques d'un monstre ciblé** (pour vérifier s'il est attaquable ou pas par exemple) : pour cela, il va falloir notamment relier « *clic de souris* » et « *entité ciblée*».

Exemple ici :

1/ Dans le fichier *main.py*

La classe *`My_Game`* propose des **méthodes** permettant au **joueur d'interagir** avec l'écran que ce soit par le clavier ou la souris.

La méthode *on_mouse_press(self, button, x, y, modifier)* permet de gérer les **clics** de souris.
Voici le descriptif de ses paramètres :

- *button* indique le **type** de bouton de la souris cliqué (*right*, *left*, *center*).
- *x, y* sont les **coordonnées relatives** du clic effectif de la souris.
- *modifier* n'a pas d'intérêt à ce stade.

Voici le code qui sera à compléter dans le programme :

Coordonnées absolues du clic. **Il faudra les déterminer** *(voir plus bas)*.
Récupère les **entités cliquées** de l'attribut *self.sprites_list* (principe de la boundingbox).

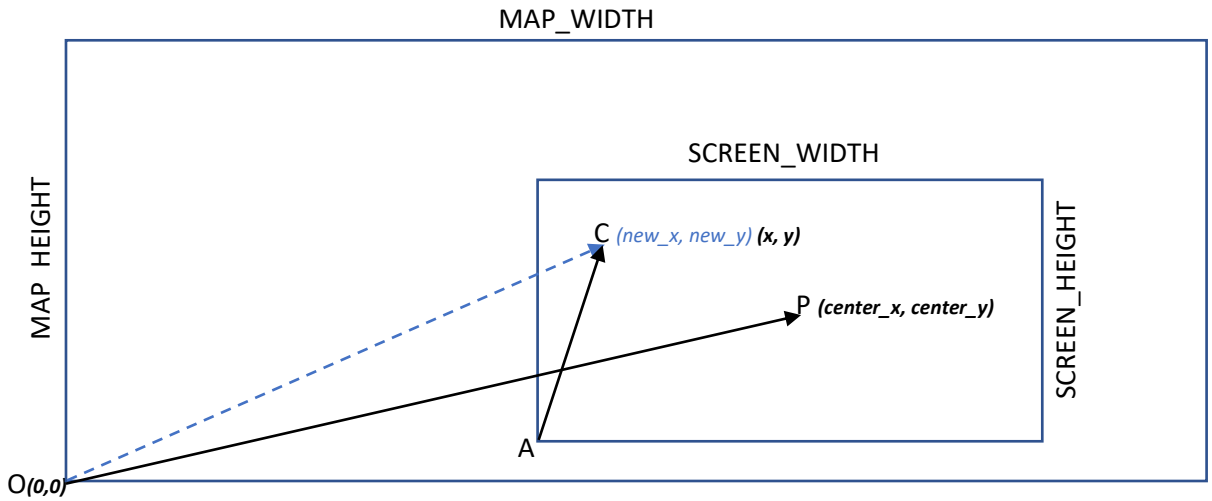
```
def on_mouse_press(self, x, y, button, modifiers) :
    # Attention à transformer les coordonnées relatives en coordonnées absolues
    new_x = # A compléter
    new_y = # A compléter

    # clicked_mob est une liste, attention !
    clicked_mob = arcade.get_sprites_at_point((new_x, new_y), self.sprites_list)

    # Si un monstre a bien été choisi et qui n'est pas le joueur (un seul en même temps)
    if len(clicked_mob) > 0 :
        self.player.clicked_mob = clicked_mob[0] # clicked_mob[0] est ciblé par le joueur

    # Mise à jour de l'IHM
    self.gui.setup()
```

Détermination des coordonnées absolues *new_x* et *new_y*.



2/ (*) Sachant que le **joueur est <u>toujours</u> centré** sur l'écran, **déterminer** les coordonnées *(new_x, new_y)* représentées par la flèche bleu en pointillés en fonction de *x, y, player.center_x, player.center_y, SCREEN_WIDTH* et *SCREEN_HEIGHT.*

<u>Aide</u> : trouver les coordonnées relatives du point P (c'est-à-dire celles du vecteur $\overrightarrow{AP}$). Chercher ensuite une relation (Chasles) entre le vecteur $\overrightarrow{OC}$ et les vecteurs $\overrightarrow{AC}$, $\overrightarrow{OP}$et $\overrightarrow{AP}$.
En déduire ensuite les coordonnées new_x et new_y demandées.

3/ **Vérifier** que le programme affiche les caractéristiques du monstre si on clique dessus.
Que se passe-t-il si on clique sur soi-même ? **Proposer** une **condition supplémentaire** pour empêcher cela (toujours dans la méthode *on_mouse_press()*).

<u>Appel au professeur pour validation</u>

<u>Remarque</u> : la touche 'escape' permet ici de décibler un monstre. Il suffit pour cela :

- de mettre à **None** l'attribut *`clicked_mob'* de l'objet *`self.player`*,
- de **mettre à jour** l'IHM

```
def on_key_press(self, key, modifiers):
    # Mouvements du joueur
    if key == arcade.key.LEFT : # Vers la gauche
        self.player.change_x = -self.player.attributes['Speed']
    ##### A COMPLETER #####
    elif key == arcade.key.LEFT : # Déciblage d'un monstre
        self.gui.update()
```

4/ (*) **Proposer** une autre interface (ajout d'autres caractéristiques à afficher, organisation des données etc.)

<u>Appel au professeur pour validation</u>