

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 crée 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)`.

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 (en abscisses avec `anchor_x`) et à droite (en ordonnées avec `anchor_y`) ce menu.

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

Placement du menu en fonction des valeurs de *anchor_x* et de *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.

<i>anchor_x</i> : left <i>anchor_y</i> : top	<i>anchor_x</i> : center <i>anchor_y</i> : top	<i>anchor_x</i> : right <i>anchor_y</i> : top
<i>anchor_x</i> : left <i>anchor_y</i> : center	<i>anchor_x</i> : center <i>anchor_y</i> : center	<i>anchor_x</i> : right <i>anchor_y</i> : center
<i>anchor_x</i> : left <i>anchor_y</i> : bottom	<i>anchor_x</i> : center <i>anchor_y</i> : bottom	<i>anchor_x</i> : right <i>anchor_y</i> : bottom

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

Voici la description de la méthode *setup(self)* qui permet d'afficher le menu interactif avec l'utilisateur et la description 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*).

```
# 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)*.

5/ **Ecrire** l'instruction suivante dans la méthode *draw(self)*.

```
def draw(self) :
    self.game.manager.draw()
```

Exécuter le programme, un bouton **Quit** doit s'afficher en bas à droite et fonctionner correctement.

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

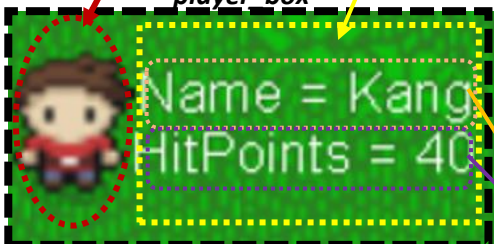
Appeler le professeur (si besoin)

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

player_texture_button

v_box_2

player box



```
# 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)* et **compléter** la méthode *create_clicked_mob_box(self, clicked_mob)* en suivant la même progression.

7/ **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.

Ancrage en haut à gauche du descriptif du joueur

```
# 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()

    # Ajout de la box du joueur dans le manager
    self.game.manager.add(arcade.gui.UIAnchorWidget(anchor_x="left",
                                                    anchor_y="top", child=self.player_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 😊).

Remarque : ne pas hésiter à modifier les **paramètres** affichés du joueur ou même l'interface.

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 attaquant ou pas par exemple) : pour cela, il va falloir notamment relier « *clic de souris* » et « *entité ciblée* ».

Exemple ici :



1/ Dans le fichier `gui.py`

Voici le descriptif de la méthode `add_clicked_mob(self, clicked_mob)` :

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.

Création des caractéristiques de l'entité cliquée (paramètre `clicked_mob`).

Création d'une box horizontale qui accueillera le descriptif du joueur et celle de l'entité cliquée.

Placement de la box précédente dans le manager et à haut / gauche de la fenêtre de jeu.

```
# IHM de base avec un monstre de cliqué
def add_clicked_mob(self, clicked_mob) :
    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()

    # Ajout des caractéristiques du monstre / joueur cliqué
    self.create_clicked_mob_box(clicked_mob)

    # Création de la box recevant les caractéristiques du joueur et
    # monstre (éventuellement) cliqué
    mobs_box = arcade.gui.UIBoxLayout(vertical = False)
    mobs_box.add(self.player_box)

    # Ajout du monstre cliqué
    # A MODIFIER pour ne pas afficher deux fois le joueur
    mobs_box.add(self.clicked_mob_box)

    # Ajout du menu joueur / monstre (si cliqué) dans l'interface
    self.game.manager.add(arcade.gui.UIAnchorWidget(anchor_x="left",
                                                    anchor_y="top", child=mobs_box))
```

1/ **Recopier** le code précédent dans la méthode `add_clicked_mob(self, clicked_mob)`.

2/ 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 recopier et 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).

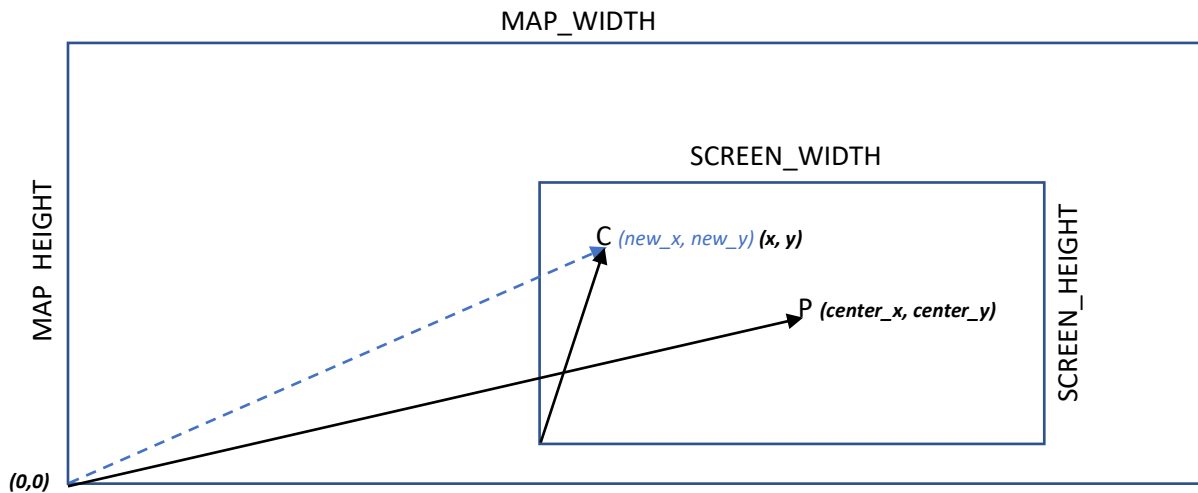
En cas de **clic** sur une entité, **affichage** de ses caractéristiques sinon on utilise la gui de base.

```
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 #####

    # Récupère la liste d'entités cliquées
    clicked_mob = arcade.get_sprites_at_point((new_x, new_y), self.sprites_list)

    # Si clic sur une entité (un seul à priori)
    if len(clicked_mob) > 0 :
        self.gui.add_clicked_mob(clicked_mob[0])
    # Sinon on affiche la GUI basique (menu + joueur)
    else :
        self.gui.setup()
```

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



2/ Sachant que le **joueur est toujours 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`.

3/ **Proposer** une amélioration du programme empêchant les caractéristiques de s'afficher deux fois si l'on clique dessus.

4/ (*) **Proposer** une autre interface.

Appel au professeur pour validation