Travail Pratique 2

par

Nicolas PATENAUDE

DEVOIR PRÉSENTÉ À Loïc CYR

LOG725-01

MONTRÉAL, LE 4 DÉCEMBRE 2025

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

UNIVERSITÉ DU QUÉBEC

Présentation

Pour ce travail, j'ai fait un réusinage du projet "Myriapod" que j'ai trouvé dans un des Repo GitHub recommendé dans l'énoncé du travail: https://github.com/Wireframe-Magazine/Code-the-Classics Le travail est accessible sur mon répertoire git (publique): https://github.com/taste3/LOG725-tp2. Dans ce jeu, on contrôle un petit robot et on tire sur des roches et sur un myriapode (mille-pattes) robotisé.

Les instructions pour lancer le jeu sont disponible dans le fichier README du projet et sont lisibles sur la page d'accueil de mon GitHub.

Modifications

Tout séparé de une seule grosse classe monolithe en différentes classes qui représentent des entitées (conteneurs). Si le jeu était complètement refactoré avec le patron ECS, c'est entitées devraient être encore plus séparés Retiré le système de son du jeu de l'instance du jeu, la fonction qui permet de faire jouer un son utilise l'instance du jeu, mais n'est pas contenue dans cette instance.

1. Patron Singleton

Patron Singleton pour l'instance de jeu globale

Relation avec le contexte du jeu

Le jeu était tout en un seul énorme fichier monolithe de plus de 900 lignes de code. Il y a bien sur plusieurs avantages à avoir tout en seul fichier, mais pour ce TP, il est impossible d'implémenter le patron ECS sans séparer le jeu en différent modules. Après avoir fait la séparation des différents modules du jeu en classes et fichiers séparés, je me suis rapidement rendu compte que l'utilisation de variables globales n'allaient pas fonctionner dans un jeu séparé en plusieurs fichiers. La solution à laquelle j'ai immédiatement pensé est l'utilisation du patron **Singleton**. L'utilisation de ce patron est commune à plusieurs engins de jeux comme unity. Ce patron permet de facilement restreindre la création d'une nouvelle partie à seulement

Avantages et inconvénients

L'avantage principal de l'utilisation de ce patron est qu'il permet de rendre l'accès à la variable "game" dans tout les nouveaux fichiers après la séparation des classes du fichier monolithe en plusieurs fichier. Sans ce patron, il faudrait passer la variable "game" à tout les composants qui l'utilise et cela causerait beaucoup de problème de couplage.

Sans le patron Singleton

```
# Create a new Game object, without a Player object
game = Game()
```

Figure 1 : Description

![Sans le patron Singleton 2](images/without_singleton_2.png)

Figure 2 : Description

Un inconvénient de l'utilisation de ce patron dans le contexte est qu'il y a tellement un fort couplage entre la classe Game et les autres classes que je ne peut pas mettre l'instance de la classe Game dans la classe elle même car cela causerais des références circulaires. Pour résoudre ces références circulaires, il faudrait implémenter ailleurs le patron Singleton et revoir l'architecture du jeu dans son entièreté. Idéalement, l'implémentation de ce patron ressemblerais à ceci:

Implémentation idéale du patron Singleton

```python
class Game:
    instance : Game = None

    def __init__(self, screen, player=None):
        instance = self
        self.screen = screen
        self.player = player
```

Figure 3 : Implémentation idéale du patron Singleton

Pour éviter de refactoriser le jeu en entier, j'ai implémenté la patron Singleton d'une manière différente. J'ai créé une classe à part nommée GameState qui contient l'instance:

Avec le patron Singleton

myriapod-master > systems > gamestate.py > ...

```python
# Singleton qui contient mon instance de Game
class GameState:

    # Instance de Game qui peut être accédée de partout dans le jeu
    game = None

    # Méthode de classe (statique) qui permet la création d'une nouvelle partie
    @classmethod
    def create_game(cls, new_game):
        cls.game = new_game

```

Figure 4 : Description

![Avec le patron Singleton 2](images/with_singleton_2.png)

Figure 5 : Description

2. Patron Observateur

Relation avec le contexte du jeu

Avantages et inconvénients

Sans le patron Observer

```
# Is the space bar currently being pressed down?
space_down = False

# Has the space bar just been pressed? i.e. gone from not being pressed, to being pressed
def space_pressed():
    global space_down
    if keyboard.space:
        if space_down:
            # Space was down previous frame, and is still down
            return False
        else:
            # Space wasn't down previous frame, but now is
            space_down = True
            return True
    else:
        space_down = False
        return False
```

Figure X : Description

Avec le patron Observer

Figure X : Description

3. Patron Entity-Component-System

Relation avec le contexte du jeu

Avantages et inconvénients

Sans le patron ECS

Figure X : Description

Avec le patron ECS

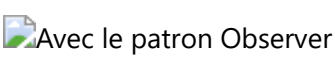

Figure X : Description