BORDEAUX INP Ensc

Projet Informatique Individuel

Pokemon Tactics

Bardisbanian Lucas

2021 - 2022

Tuteur : Benjamin Camblor

Encadrants : Edwige Clermont, Pierre Alexandre Favier

1 - Contextualisation

A - Cadre du projet

Ce projet s'inscrit dans le cadre du module "Projet Informatique Individuel" du Semestre 8 de la formation de l'École Nationale Supérieure de Cognitique (ENSC). Le but de ce projet est de mener un projet informatique de la conception à la livraison. Les objectifs sont d'acquérir une expérience d'analyse et de réalisation d'un programme informatique complexe ainsi que de gérer un projet du début à la fin.

Nous avions possibilité de choisir nous même notre sujet, à condition qu'il soit validé par les encadrants et choisi par un tuteur. C'est dans ce contexte que j'ai mis au point le projet "Pokémon Tactics", dont les détails sont évoqués ci-dessous.

B - Inspiration de Pokémon Tactics

Pokémon Tactics est un "Tactical RPG". Un Tactical RPG est un jeu de rôle basé sur des combats reposant sur la tactique et l'utilisation des capacités spéciales de pions. Il peut être soit en temps réel soit en tour par tour et est généralement basé sur une grille. La majorité des jeux de ce genre sont basés sur un univers de fantasy.

J'ai décidé de baser mon jeu sur l'univers de Pokémon car celui-ci est très riche en possibilités, et que celà changerait des jeux Pokémons basiques qui sont du combat tour par tour mais sans aucune notion d'espace ou de combats de groupes.

C - Technologie : Unity

Ce jeu repose sur la technologie Unity et C#. Unity est un moteur de jeu multiplateforme très répandu, notamment dans le domaine du jeu indépendant. Il repose sur la technologie .NET et donc des scripts C#. Il est très populaire notamment pour sa facilité d'utilisation. J'ai choisi cette technologie car je ne la maîtrise pas et ce module était une bonne opportunité pour la prendre en main. De plus, Unity est très répandu, que ça soit dans le monde du jeu vidéo mais aussi celui de la réalité virtuelle ou des interfaces cerveaux machines par exemple. C'est une technologie qu'il est toujours utile de connaître.

D - Ce qui était prévu

L'objectif durant ces 4 mois de développement était de créer un système de combat sur grille en 1 contre 1, Joueur contre Joueur, comprenant des entités avec diverses caractéristiques. Les caractéristiques varient entre points de mouvements, points d'action, puissance, type d'attaques, etc.

Voici les exigences fonctionnelles qui ont été soulevés au début du projet :

Code	Description
EF_1	Grille de jeu avec plusieurs vues (tactique et graphique)

EF_2	Mouvements et actions à la souris (déplacer une entité, lancer son action)
EF_3	Entités jouables avec actions et caractéristiques différentes pour chacunes, interagissant entre elles et avec l'environnement
EF_4	Menu de choix des entités par équipe
EF_5	Écran d'accueil
EF_6	Interface de jeu (avec accès au menu, gestion des entités, gestion de l'entité dont c'est le tour)
EF_7	Déroulement d'un combat de A à Z au tour par tour
EF_8	Animation des sprites lors de diverses actions
EF_9	Musique et effets sonores
EF_10	Elaboration du contenu du jeu (nombre d'entités, terrains, actions, etc)
EF_11	Caméra permettant de naviguer sur la grille
EF_12	Exécutable permettant de joueur au jeu

2 - Objectifs réalisés

A - Exigences fonctionnelles

Au niveau des exigences fonctionnelles, voici l'état final du tableau présenté ci-dessus. La couleur vert correspond à une exigence terminée, la bleu à une exigence en cours, et la rouge à une exigence non réalisée.

Code	Description
EF_1	Grille de jeu avec plusieurs vues (tactique et graphique)
EF_2	Mouvements et actions à la souris (déplacer une entité, lancer son action)
EF_3	Entités jouables avec actions et caractéristiques différentes pour chacunes, interagissant entre elles et avec l'environnement
EF_4	Menu de choix des entités par équipe
EF_5	Écran d'accueil
EF_6	Interface de jeu (avec accès au menu, gestion des entités, gestion de l'entité dont

	c’est le tour)
EF_7	Déroulement d’un combat de A à Z au tour par tour
EF_8	Animation des sprites lors de diverses actions
EF_9	Musique et effets sonores
EF_10	Elaboration du contenu du jeu (nombre d’entités, terrains, actions, etc)
EF_11	Caméra permettant de naviguer sur la grille
EF_12	Exécutable permettant de joueur au jeu

Ainsi, la plupart des exigences ont été réalisées, et celles manquantes reposent sur le Game Design et l’attrait du jeu. En effet, dans l’état actuel des choses, le jeu est jouable mais l’expérience de l’utilisateur est moindre dû aux problèmes d’équilibrage et d’immersion.

B - Présentation du jeu

Au lancement de l’exécutable, on arrive sur un écran d’accueil, permettant soit de lancer une partie, soit de quitter l’application.

Menu de Pokémon Tactics

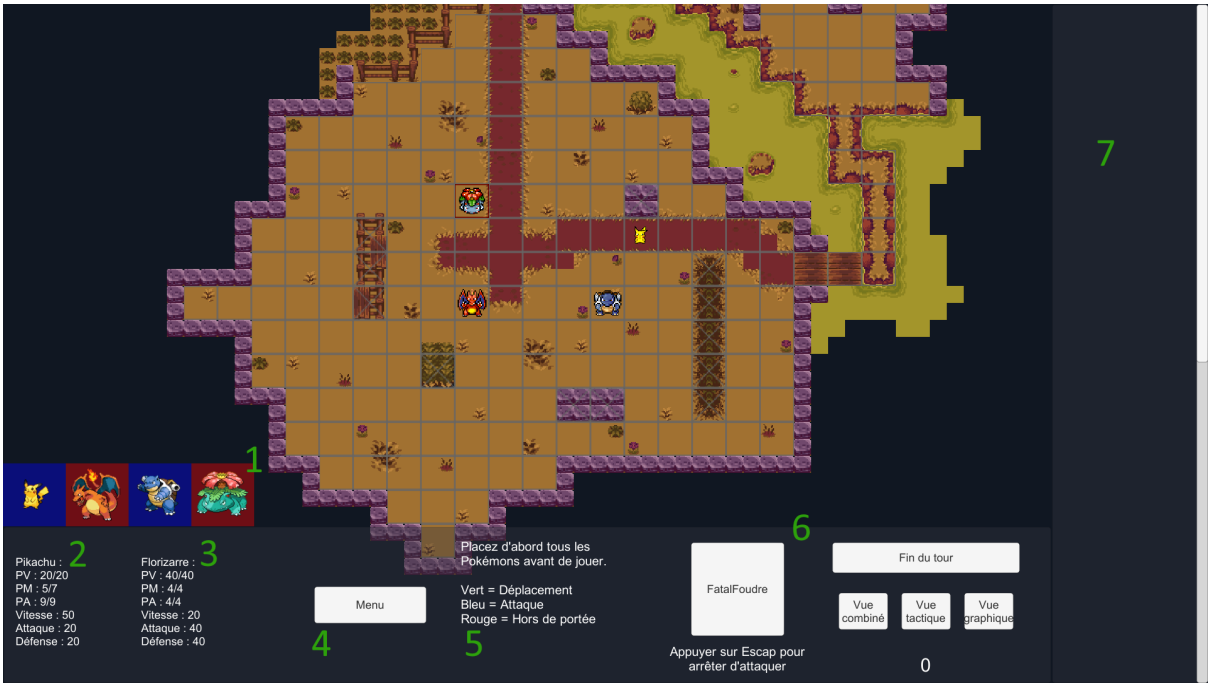
Le bouton “Lancer une Partie” permet d’ouvrir la fenêtre de choix des équipes. Ici, les deux joueurs peuvent composer leurs équipes comme ils le souhaitent, à l’amiable. C'est-à-dire que chaque équipe peut avoir le nombre de Pokémon qu’elle veut mais que chaque Pokémon ne peut être choisi qu’une fois. Pour cela, il faut cliquer sur un Pokémon, puis cliquer sur le

bouton “+” correspondant à l’équipe à laquelle on veut l’ajouter. On peut soit revenir au menu principal, soit lancer la partie avec les réglages choisis.

Fenêtre de choix d’équipe

Après avoir lancé la partie, on arrive sur la fenêtre de jeu ; la grille de combat. Celle-ci possède plusieurs vues : la vue tactique, la vue graphique et la vue combinée. Le joueur peut déplacer la caméra à l’aide des touches fléchées ou des touches du clavier ZQSD. L’interface est présentée de la façon qui suit :

Vue combinée - Vue tactique - Vue graphique

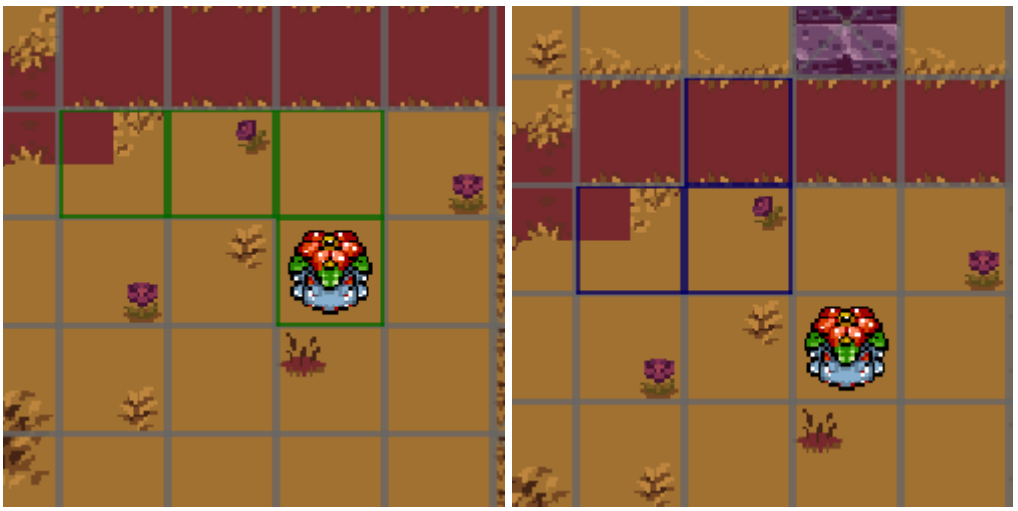


Indication sur l'interface utilisateur

1 : Liste des Pokémons en jeu, le fond indiquant leur équipe (en noir si le Pokémon est KO)
2 : Statistiques du Pokémon en train de jouer
3 : Statistiques du Pokémon survolé par la souris
4 : Retour au menu
5 : Indications de jeu
6 : Bouton d'attaque (ici FatalFoudre par Pikachu), bouton de fin de tour ou de placement, trois boutons de changement de vue, et compteur de tour
7 : Fenêtre de log, indiquant des informations tels que les attaques et dégâts reçues par un Pokémon

Le jeu commence avec une première phase de placement, où tous les Pokémons doivent être placés, du plus rapide au moins rapide (cet ordre restera le même pour toute la partie). Durant cette phase, ils peuvent se déplacer où ils veulent, mais ne peuvent ni attaquer ni interagir avec les autres Pokémons. Une fois cette phase terminée, le combat commence. Les pokémons peuvent réaliser 3 actions :

- Se déplacer, selon un nombre de cases déterminées par leurs points de mouvements.
- Attaquer, selon leurs points d'action restant et selon leur attaque associée.
- Finir son tour.



Florizarre se déplace (en vert), puis Florizarre attaque (en bleu)

Comme on peut le voir ci-dessus, un feedback coloré est donné pour chaque action. La case est coloriée en rouge si elle est hors d'atteinte. Le calcul de chemin a été réalisé grâce à un algorithme de Dijkstra, tout comme la portée des attaques.

Trois types d'attaques ont été réalisées :

- L'attaque Fatal Foudre qui attaque une case à longue portée.
- Les attaques Hydrocanon et Lance Flammes qui attaquent sur 3 cases en ligne droite, soit à gauche, à droite, en bas ou en haut.
- L'attaque Tranche Herbe, qui attaque en ligne de 3 (ou V de 3 si on vise en diagonale), à longue portée également.

Les dégâts sont calculés grâce à la formule suivante :

$$(P \times (A \div D)) \div 2 \times R$$

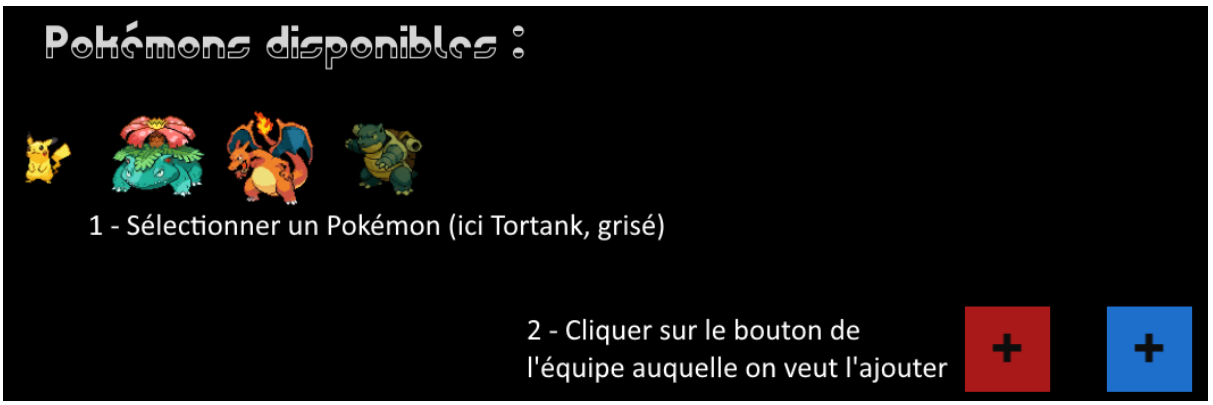
Avec P la puissance de l'attaquant, A l'attaque de l'attaquant, D la défense du défenseur et R la résistance du défenseur au type de l'attaque.
Les caractéristiques des Pokémons et des Attaques sont détaillées dans la partie Architecture. Les attaques touchent toutes les entités présentes sur les cases coloriées, donc alliés compris.

Quand tous les Pokémons d'une équipe sont KO, un message de Victoire s'affiche dans la fenêtre de Log.

C - Comment lancer et jouer au jeu ?

Les fichiers de jeu sont téléchargeables via GitHub. Il existe de dépôt, un premier permettant de télécharger le rapport, l'exécutable pour tester le jeu et quelques fichiers de code importants, accessible ici. Le second permet de télécharger l'intégralité du projet, avec les fichiers Unity, pour pouvoir lancer le projet dans Unity si celui-ci est installé (voir section suivante pour tuto). Ce dernier est accessible ici.

Pour lancer le jeu, il suffit de lancer l'exécutable présent dans le dossier PokemonTactics_Lanceur. Dans le menu, il faut lancer une partie puis sélectionner les Pokémons de chaque équipe. Pour cela, il suffit de cliquer sur un Pokémon puis sur une des croix associée à chaque équipe. Quand les équipes sont composées, il suffit de lancer la partie.



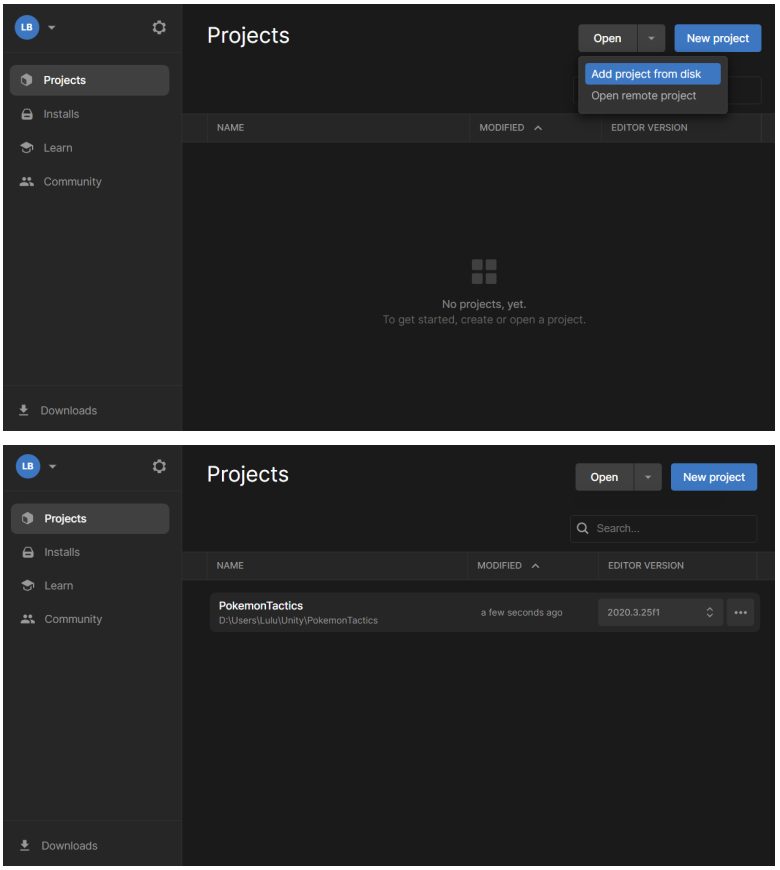
Explication du choix des équipes

La partie commence par une phase de placement où il faut placer chaque Pokémon sur la carte. Le Pokémon actuellement joué a ses caractéristiques affichées en bas à gauche de l'écran. Ensuite, vous pouvez déplacer les Pokémons selon la limite de leurs points de mouvements ou attaquer selon la limite de points d'action et la portée de l'attaque. Attention, les attaques ont des zones de frappe qui peuvent infliger des dégâts collatéraux à vos alliés. On peut bouger la caméra grâce aux touches fléchées ou bien aux touches ZQSD.

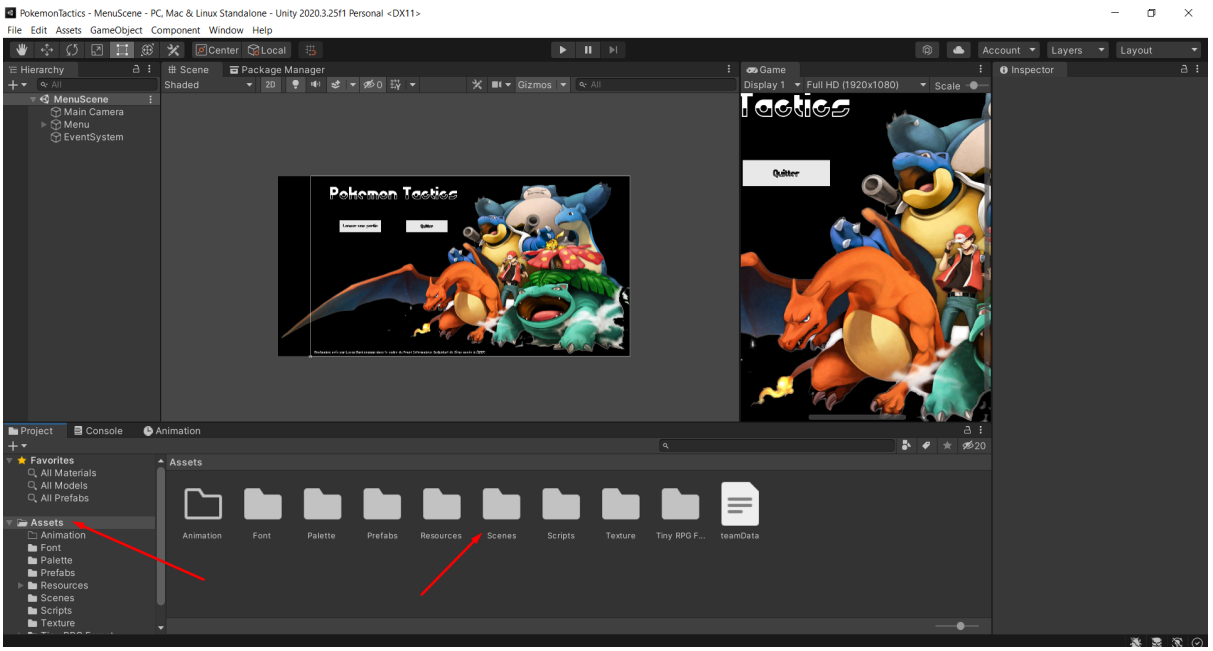
D - Comment explorer les fichiers de jeu ?

Le projet repose sur Unity et plus particulièrement la version *2020.3.25f1*. Pour ouvrir l'interface dans Unity, il faudra donc installer cette version de Unity, disponible dans les archives. Néanmoins, l'installation de Unity est assez lourde, donc afin de ne pas rendre son installation obligatoire pour l'exploration et la correction du projet, de nombreuses captures d'écran seront disponibles au cours du rapport et en annexes.

Une fois la bonne version de Unity installée, il faut importer le projet puis cliquer dessus pour le lancer :



Après chargement, Unity se lance et nous accédons à cette interface :



Le projet est divisé en deux “Scenes”, une pour le menu et une pour la fenêtre de Combat. L’accès aux scènes se fait à partir du dossier Scenes dans Asset, puis en sélectionnant SampleScene pour la scène de combat, ou MenuScene pour la scène de menu. Les grandes bases de l’interface de Unity sont expliquées en Annexe.

Les Scripts et fichiers de jeu sont disponibles dans le dossier Assets. Dans le dossier Scripts, vous trouverez tous les scripts utilisés pour coder le jeu. Dans le dossier Scenes, vous trouverez les différentes scènes du projet. Dans le dossier Resources, vous trouverez les données que j'ai créées pour le projet, c'est-à -dire les fichiers .json des Pokémons et des attaques, ainsi que les textures des sprites des Pokémons et l'image de fond du menu.

3 - Architecture

Unity repose sur des objets appelés "Gameobject" auxquels on peut associer un comportement via des Scripts C# ainsi que des composants déjà existants. Il repose également sur des scènes reliées entre elles. Mon projet contient 2 scènes :

- Une scène MenuScene, gérant le menu d'accueil et la fenêtre de choix des équipes.
- Une scène SampleScene, gérant toute la partie combat.

Pour expliquer au mieux l'architecture de mon projet, j'aborderais une scène puis l'autre.

A - Scène du menu

Dans la scène MenuScene, un GameObject "Menu" comporte deux composants principaux ; un Canvas, composant originaire de Unity permettant de réaliser l'interface utilisateur, et un composant MenuManager, que j'ai codée.

Tous les sous-éléments de Menu sont des textes, des images ou des boutons permettant de naviguer dans l'interface ou de l'afficher.

Le composant MenuManager (accessible dans Assets → Scripts → MenuManager) permet de gérer les fonctions associées aux boutons. Ainsi, diverses fonctions permettent de :

- Afficher la fenêtre de choix des équipes puis lancer une autre fonction qui affiche la liste des Pokémons disponibles
- Quitter l'application
- Sélectionner un Pokémon et changer les fonctions associées aux boutons "+" pour qu'ils aient le Pokémon sélectionné en paramètres
- Ajouter un Pokémon à une équipe et déplacer son image de la liste des Pokémons disponibles jusqu'à son équipe correspondante
- Lancer la partie en chargeant la nouvelle scène après avoir enregistré les données des équipes.

B - Architecture des données

Les données des Pokémons et des Attaques sont enregistrées dans des fichiers JSON situés dans Assets → Resources → Attaques ou Assets → Resources → Sprites → NomDuPokemon.

Un fichier de données d'un Pokémon ressemble à ceci :

```
{
  "_desc": "Florizarre",
  "PointsDeVie": 40,
  "PointsDeMouvement": 4,
  "PointsDAction": 4,
  "Attaque": 40,
  "Defense": 40,
  "Vitesse": 20,
  "ValeursFaiblesses": [
    1, 2, 0.5, 0.25, 0.5, 2, 0.5, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1
  ],
  "Attaques": ["TrancheHerbe"]
}
```


Et celle d'une attaque à ceci :


```
{
  "_desc": "FatalFoudre",
  "Type": "Electrik",
  "Portee": 6,
  "Rayon": 1,
  "Forme": 0,
  "Puissance": 30,
  "Cout": 3
}
```


Ces fichiers JSON permettent de créer des GameObject possédant des composants Entite et Attaque respectivement (accessibles dans Scripts → Entite ou Scripts → Attaque).
La fonction Unity permettant d'initialiser des composants à partir des fichiers JSON ne prenait pas en compte les dictionnaires. C'est pourquoi les faiblesses sont sous forme de tableau de double plutôt qu'un dictionnaire de la forme {CléType}:ValeurFaiblesse.

C - Scène de combat

Cette scène reposait sur trois scripts principaux (accessibles par Assets → Scripts) :

- *DeroulementDuJeu* est le coordinateur. Il commence par initialiser les Pokémons dans chaque équipe selon les données passées par MenuScene et les données JSON montrées ci dessus. Il gère les changements de tour et les attaques entre Pokémon.
- *CommandeGrille* gère l'affichage de la grille. Notamment, il récupère la position de la souris sur la grille et le coloriage des cases selon l'état du jeu. Il repose sur un script secondaire *RechercheDeChemin*, qui met en place l'algorithme de Dijkstra et permet de savoir quelles cases colorier quand un Pokémon se déplace ou si la case visée est à portée. Il repose également sur le script *Attaque*, qui calcule la zone de dégâts (avec collisions) et donc les cases à colorier pour l'attaque. *CommandeGrille* gère aussi les inputs, avec ZQSD ou les flèches pour bouger la caméra, le clic de la souris pour gérer les déplacements et attaques, ou encore la touche Escap/Echap pour quitter le mode attaque.

- *UiManager* permet de gérer l'interface utilisateur et met régulièrement à jour ses différents éléments, tels que l'état des Pokémons, la fenêtre de Log ou encore le tour actuel. Il gère également les fonctions associées aux différents boutons de l'interface utilisateur.

4 - Planning de travail

Au cours de ce projet, j'ai établi un planning qui a évolué 3 fois au cours du temps. Il y a eu un planning initial, un planning intermédiaire et un planning final. Les différences entre le planning initial et le planning intermédiaire sont principalement dûes à une mauvaise estimation du temps accordé à chaque tâche, car la technologie m'était encore inconnue. Au contraire, les différences entre le planning intermédiaire et le planning final reposent plutôt sur une mauvaise gestion de mon temps de travail sur le projet, qui m'a mis en retard. Par conséquent, les parties animation et musique ont été retirées du planning final et les parties reposant sur l'algorithme de recherche de chemin et d'interface utilisateur ont été rallongées.

5 - Pistes d'évolutions

A - Contexte du projet : focus sur le combat

Le projet se concentrait sur la partie combat du jeu Pokemon Tactics, et celle-ci est encore largement imparfaite. Déjà, il faudrait réaliser les exigences techniques non finies ou pouvant être améliorées :

- L'attrait et l'immersion dans un jeu reposent dans une grande mesure sur la musique et l'animation, qui sont inexistants dans ma version du jeu.
- De plus, un jeu intéressant et durable nécessite un contenu développé et surtout équilibré. Ma version du jeu propose seulement 4 Pokémons, avec une attaque chacun, et leurs statistiques ne sont pas équilibrées. Un contenu léger ne permettrait pas un investissement durable des potentiels joueurs sur le jeu.
- L'interface utilisateur n'est pas très intuitive ni très élégante et mérite une refonte, notamment avec une gestion des erreurs et des feedbacks plus implicites. On pourrait également y ajouter des éléments comme un tableau des scores ou un indicateur de puissance de l'équipe.

D'autres fonctionnalités peuvent être ajoutées pour rendre les combats plus complexes et tactiques. Par exemple, la notion de tacle lorsque deux Pokémons sont au corps à corps, évitant qu'ils se séparent sans pénalités. Autre exemple, l'implémentation d'objets utilisables par le joueur afin de renforcer les Pokémons de son équipe ou affaiblir ceux de l'équipe adverse. Enfin, rajouter des caractéristiques liées aux terrains, qui pourraient renforcer ou affaiblir les Pokémons selon leur type. Par exemple, un Pokémon de type Eau ou Plante aurait des malus dans un environnement de type Feu.

Le jeu se joue uniquement en local à l'heure actuelle, sur un même poste. Implémenter une version serveur pour avoir du multijoueur en ligne permettrait de développer le projet.

Également, le jeu se joue uniquement en multijoueur 1 contre 1 actuellement. Le développement d'une IA permettant de jouer en Solo serait une plus-value et permettrait de nombreuses ouvertures sur d'autres mécaniques de jeux, développé dans la partie suivante du rapport.

B - Ouverture

Une fois une IA de combat développée, le jeu pourrait prendre une approche plus proche d'un jeu Pokémon classique et introduire une carte de campagne où le joueur pourrait se déplacer en monde ouvert et ainsi rencontrer et capturer des Pokémons sauvages. A ce moment-là, la notion de niveau d'un Pokémon pourrait être intégrée aux combats, pour encore plus de diversité.

6 - Bilan personnel

A - Difficultés rencontrées

Au niveau de la gestion de projet, la principale difficulté est de juger les avancées selon le temps investi dans le projet. En effet, ce projet se plaçant dans le cadre d'un semestre entier, avec des périodes plus ou moins lourdes en travail personnel à cause d'examens ou d'autres projets, il est difficile d'estimer le temps disponible pour avancer le projet. A cause de cela, mes avancées ont souvent été "saccadées" et parfois mal gérées. Également, certaines fonctionnalités semblaient facile à mettre en place, comme l'interface utilisateur par exemple, mais prenaient bien plus de temps que prévu.

Au niveau de la technologie utilisée, je n'ai pas eu beaucoup de soucis avec la prise en main d'Unity. La documentation est importante et la communauté également, permettant de trouver des réponses à des questions assez facilement. Mon principal souci repose sur la qualité des approches que j'entreprends. En effet, celles-ci sont fonctionnelles, mais je ne sais pas si elles sont qualitatives au niveau efficacité. Il faudrait que je contacte une personne s'y connaissant mieux que moi en Unity, afin de me renseigner et d'optimiser mon code et mes fonctionnalités.

Pour donner quelques exemples, le code de mon jeu repose sur 3 scripts principaux "DeroulementDuJeu", "CommandeGrille" et "UIManager" qui sont liés par "DeroulementDuJeu" et je place toutes mes méthodes dans ces 3 scripts. Je ne sais pas si c'est une bonne façon de faire, mais c'est celle qui m'a paru le plus instinctive et similaire à l'approche POO avec C#. Un autre exemple repose dans l'utilisation de la souris. Au lieu de faire un raycaster qui verraient directement l'objet que pointe la souris, je récupère la position du curseur et je le transforme en coordonnées de grille via des méthodes pré-intégrées à Unity. Cela me semblait plus cohérent et intuitif pour des déplacements sur grilles, mais cela a ses désavantages. Par exemple, pour savoir si une case est occupée je suis obligé de parcourir toutes les entités. Ce problème me semble moindre car il n'y a jamais plus d'une

dizaine de Pokémons en même temps sur la grille. Néanmoins, si mon jeu évolue un jour, ce mode de fonctionnement pourra poser problème.

Ainsi ma difficulté principale repose dans le fait de prévoir ou non si la méthode que j'utilise sera efficace à tout moment dans mon jeu ou est seulement adaptée au jeu dans l'état actuel.

B - Montée en compétences

Ce projet m'a permis gagner des compétences techniques, principalement sur le moteur de jeu Unity, mais également dans une moindre mesure en C# et en Programmation Orientée Objet. Unity est un moteur de jeu agréable à utiliser et assez intuitif et ce projet m'a encouragé à me lancer dans des projets personnels avec cette technologie dans le futur.

Afin de progresser dans cette nouvelle technologie, j'ai préféré reposer sur des recherches internet plutôt que sur des vidéos tutorielles ciblées. En effet, je trouve personnellement que ces vidéos ne sont pas assez pédagogiques et ne permettent pas une large personnalisation et une bonne mémorisation des acquis présentés dans la vidéo. Ainsi, j'ai regardé uniquement une vidéo expliquant le fonctionnement global de Unity, sans rentrer dans les détails. Ensuite, j'essayais d'abord de trouver moi même comment réaliser une tâche puis, si j'étais bloqué, je cherchais de l'aide sur Internet. La plupart de mes recherches portaient principalement sur quelle fonction utiliser et sur la recherche de documentation.

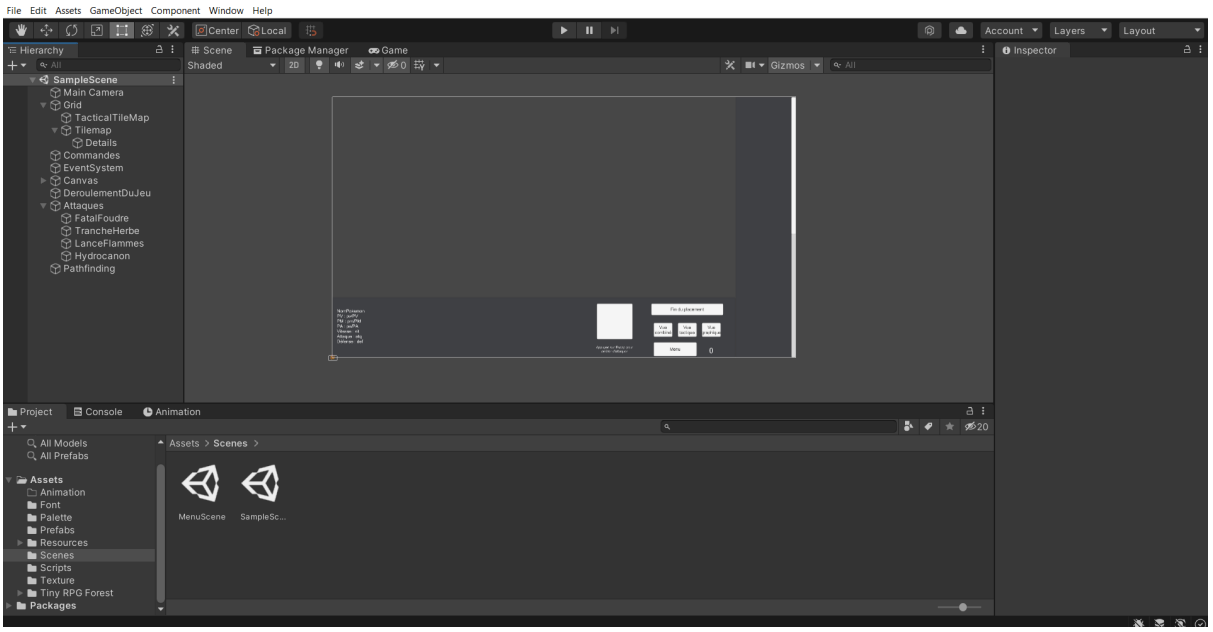
Pour moi, cette méthode m'a permis d'être indépendant dans mon utilisation de Unity et me permettra de réaliser d'autres projets en autonomie. Le principal défaut de cette technique repose sur le fait que je n'ai pas de retour sur les méthodes que j'utilise, et il y a de grandes chances que certaines soient plus efficaces. Pour remédier à ce défaut, j'aurais dû présenter mon projet à un professeur qui s'y connaît dans cette technologie afin qu'il m'aiguille sur des pratiques efficaces. Je ne me suis malheureusement pas donné le temps pour cela.

C - Retour sur le projet

Je suis globalement content de mon travail réalisé dans ce module, malgré la déception de ne pas finir quelques tâches, comme l'animation et la musique. Il est enrichissant d'avoir un œil extérieur sur un projet personnel, par l'intermédiaire du tuteur. Il est possible que je continue ce projet sur mon temps libre à l'avenir.

7 - Annexes

A - Interface Unity

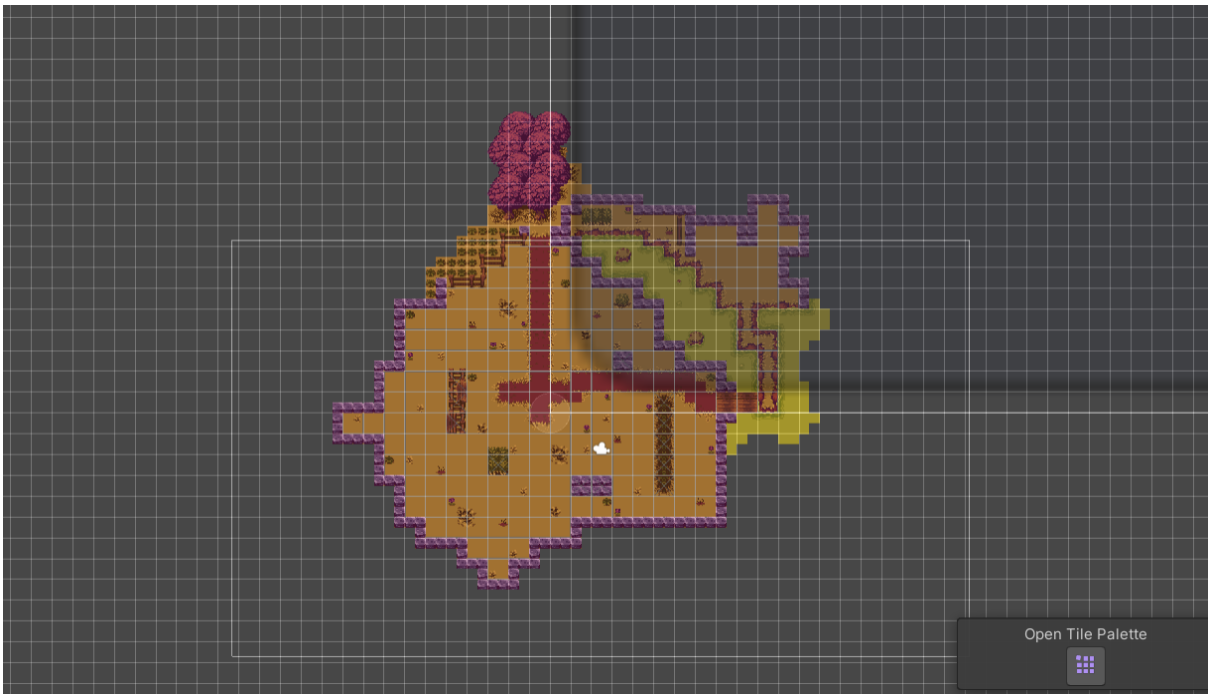


Interface de Unity. En bas, les documents du projet, avec notamment l'accès aux Assets comprenant Scripts, Ressources et Scenes. A gauche, la hiérarchie des GameObject. A droite, l'inspecteur, donnant des informations sur l'élément sélectionné. Au milieu, la scène, avec tous ses éléments. On peut changer d'onglet en cliquant sur Game et voir le rendu final. Pour lancer la simulation, il faut cliquer sur le triangle Play en haut au centre de l'écran.

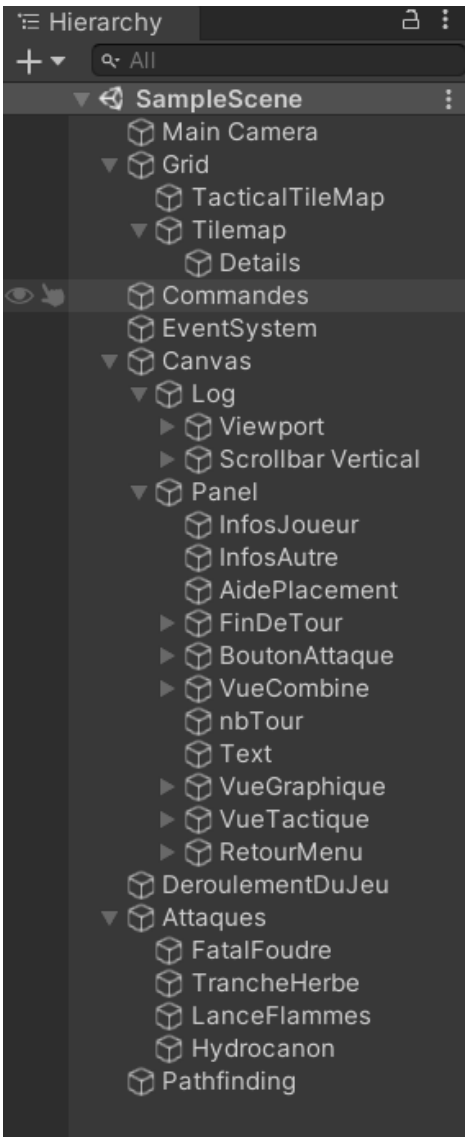
B - Scène de combat

Voici divers screenshots de la scène de combat :

Vue du Canvas

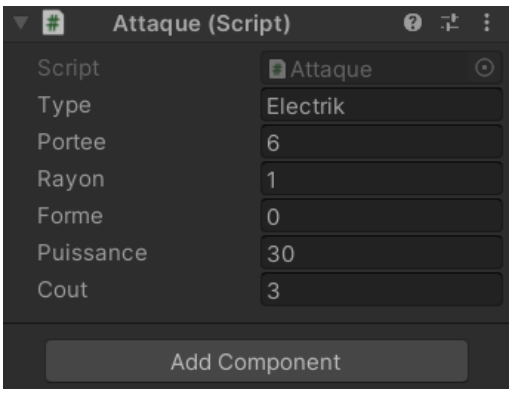


Vue des grilles superposées



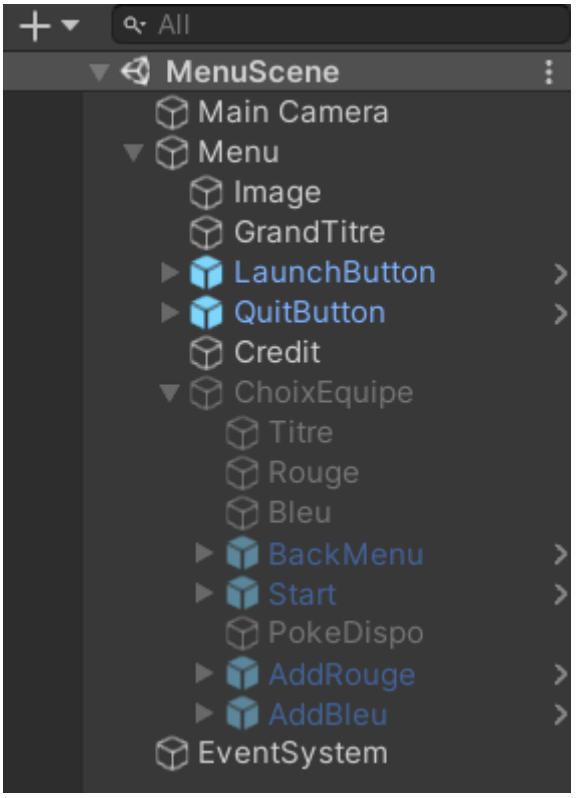
Hiérarchie de la scène

Inspecteur d'un GameObject Entite, ici Pikachu

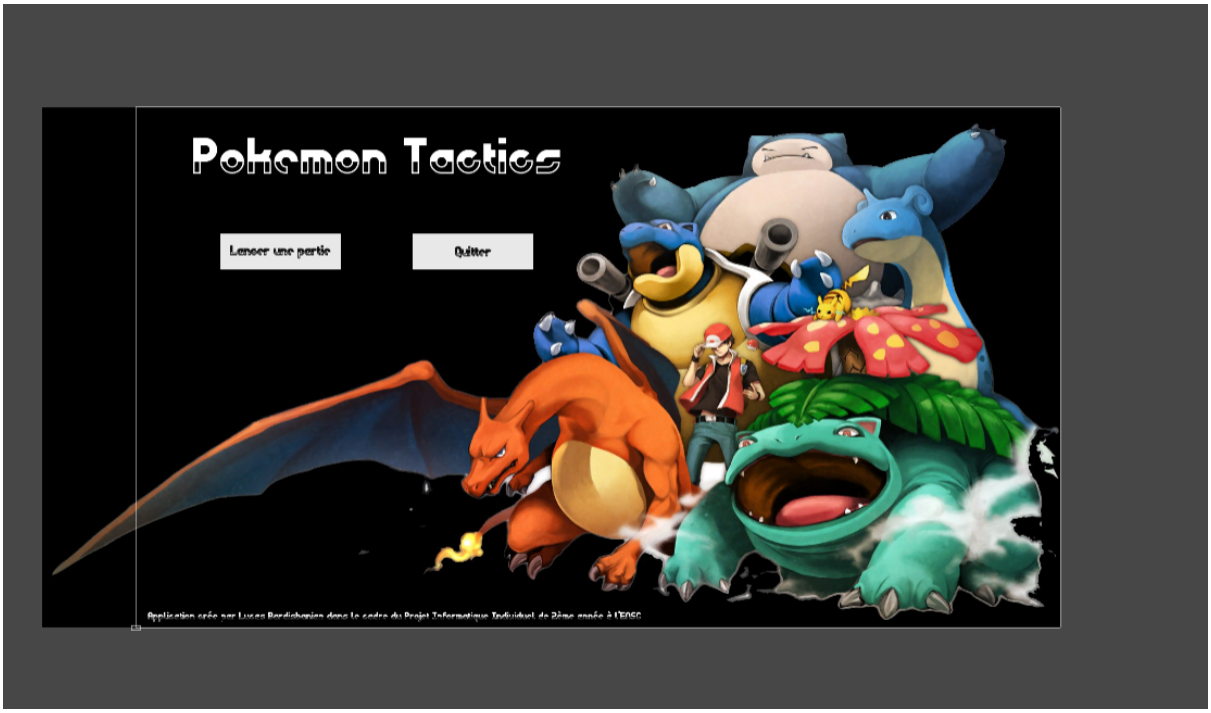


Inspecteur d'une Attaque, ici FatalFoudre

C - Scène de menu



Hiérarchie de la scène de menu (ChoixEquipe est grisé car il est rendu visible par un bouton dans le script)



Vue du Canvas de MenuScene