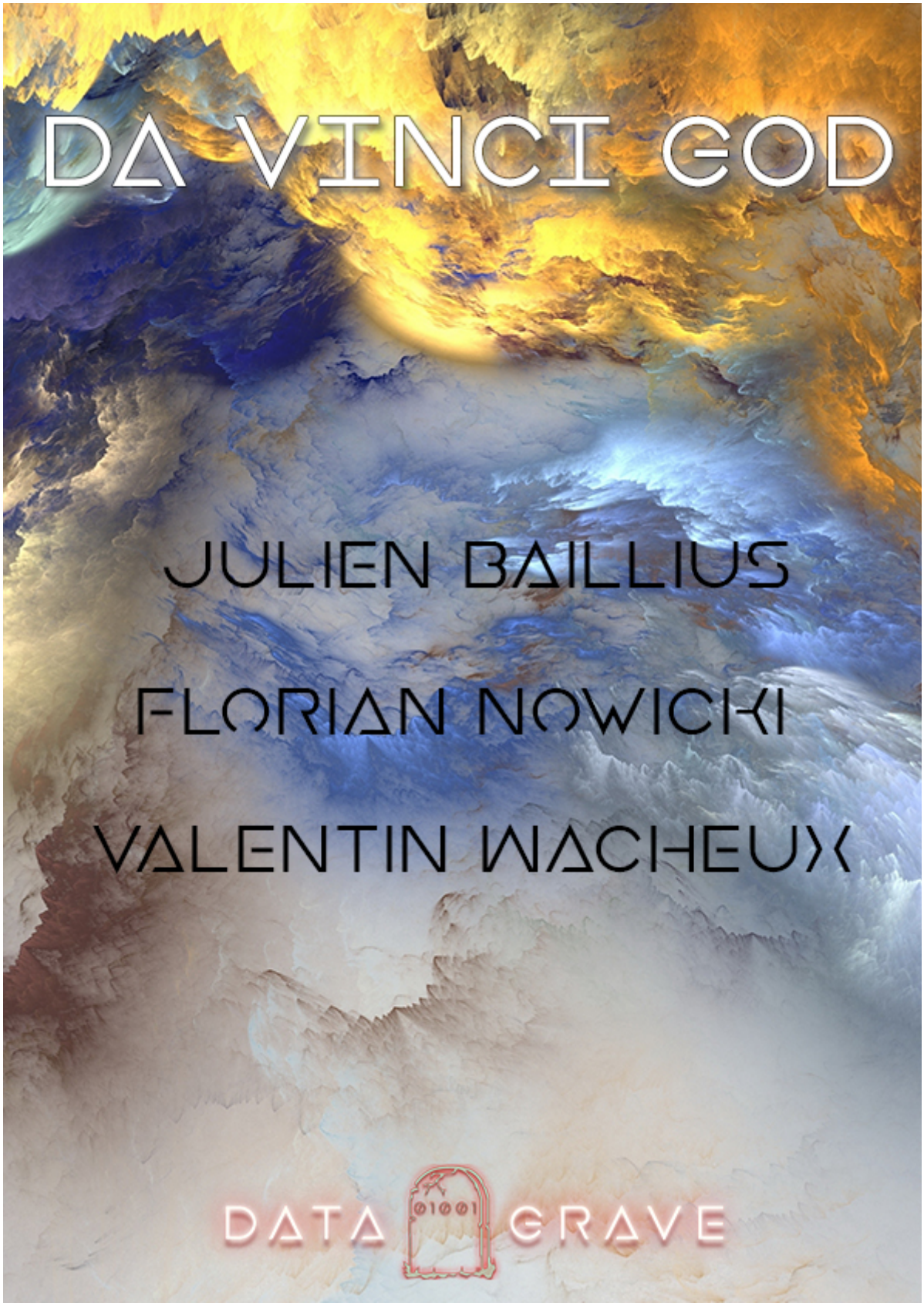

DA VINCI GOD
JULIEN BAILLIUS
FLORIAN NOWICKI
VALENTIN WACHEUX
DATA GRAVE

Présentation

Dés les premiers cours d'informatique et sciences du numérique, nous avons eu droit à diverses présentations de projets antérieurs, mais aucun ne nous a réellement inspiré ou même séduit. L'idée de faire une application n'était pas dans nos envies, c'est pour cela que notre choix s'est porté sur un jeu. Nous avons alors réfléchi au type de jeu que nous voulions réaliser. Il était hors de question de faire un jeu trop basique, un simple *point and clic,* ou alors à l'inverse, trop complexe. Nous avons longtemps débattu sur la question, et nous nous sommes mis d'accord sur le principe d'un jeu de gestion qui est très à la mode depuis quelques années sur les plate-formes mobiles et PC. Mais pour sortir du lot nous avons décidé de créer un jeu qui sort du model *Clash of Clan like*. Nous avons choisi le thème de l'espace, avec pour lignes inspiratrices *Stellaris* et *Civilisation* puis nous avons décidé que le joueur créerai lui-même son environnement de jeu (forme de vie, planètes, etc...). Voila donc les prémisses du projet *DaVinciGod.*

Pour nous encourager dans cette création, nous nous sommes donné un nom de groupe, comme une petite entreprise. Ainsi est né *DATA GRAVE.* Dès lors, nous voulions construire notre projet non pas seulement comme un travail à rendre en fin d'année, mais comme une fin en soi.

Positionnement du projet avec des solutions existantes

Nous savions déjà qu'il existais une multitude de jeux mélangeant les univers du jeu de gestion dans l'espace. Certains d'entre eux sont même connu du grand publique malgré qu'ils visent une audience réduite, comme *Stellaris, SandBox Univers* et bien d'autres. Nous savions qu'avec les moyens à notre disposition, nous ne pourrions pas atteindre un tel niveau de confection d'application.

Le jeu vidéo de gestion met l'accent sur la gestion de ressources limitées dans le but de prospérer. Parce qu'ils tendent à simuler des phénomènes réels, la plupart des jeux de gestion sont assimilés aux jeux de simulation. Le jeu de gestion concerne le processus, la stratégie du changement, il fait surtout appel à la planification et oblige à concilier les exigences contradictoires de court terme et de long terme. La progression peut parfois être découpée en missions mais le *gameplay* tend souvent à être non-linéaire et sans fin. La plupart se déroule en temps réel. Il existe une grande variété de titre et l'expérience de jeu est très dépendantes des domaines d'activités simulés ainsi que la profondeur recherchée par les développeurs : les jeux peuvent être complexes, simplistes, sérieux ou humoristiques.

Nous nous sommes donc renseigné sur internet pour trouver des solutions viables en corrélation avec le langage enseigné en ISN, à savoir le Python (3.5). Mais parmi le peu de choses trouvés, seul les moins utilisables étaient en *Open Source*. Nous avions donc choisis de tout faire « à notre sauce ». Notre projet *DaVinciGod* est donc un jeu hybride qui croise le jeu de gestion et le jeu de simulation de vie.

Analyse du besoin – Recherche d'idées

Le premier reflex qui nous avions eu, était de faire un *Brain Storming* sur un *notepad online* (Etherpad). Après une heure, nous avions une multitude d'idées, qui au fil du temps, se sont affinées pour devenir des objectifs.

Nous avions alors un but, et deux parties fondamentales au sein de notre projet. Le but visé était de proposer aux joueurs un jeux avec une histoire complète et un *gameplay* qui offre de multiples fonctions. Nos deux objectifs principaux divisent en quelques sortes le projet en deux parties : D'un côté la création de planètes et de l'autre des espèces qui les domineront, en plus de la dimension de gestion de ces deux précédents aspects.
Les objectifs à réaliser étaient les suivants :

- Pouvoir créer les planètes de son système solaire
- Faire face à des éventements aléatoires
- Prendre des décisions qui influent sur le jeu
- Simuler une gestion d'environnement et de population
- Avoir accès à deux mini-jeux de hasard pour changer la situation
- Avancer dans le jeux à travers deux arbres de compétences

Pour l'environnement de travail nous avons tous travaillés sous *Windows 10,* sauf sur les dernières semaines où j'ai commencé à m'intéresser à Linux avec Ubuntu.

Comme IDE, j'ai personnellement une grande préférence pour *Sublime Text* qui m'a accompagné tout au long du projet, même si j'ai du parfois passer par *Pyzo* et *PyCharm* pour corriger quelques erreurs de compatibilités. Car en effet, nous n'avons fais l'usage que d'un seul et unique langage, et il s'agit du Python, langage étudié en cour comme dis précédemment. Mais nous avons quand même fais l'utilisation d'une bibliothèque extérieur à *Miniconda* et Python.

Pour réaliser le projet *DaVinciGod* nous n'avons pas utilisé la bibliothèque *Tkinter* étudié en cours, car pas assez performante et optimisée pour nos besoins. Nous avons porté notre intérêt sur la bibliothèque *Cocos2d* mais nous avons vite abandonné l'optique de l'utiliser, en effet, bien que parfaite dans la conception de jeu en 2D, elle était beaucoup trop complexe en termes d'utilisation et de compréhension, son apprentissage nous aurait pris trop de temps et nous aurait empêché de réaliser l'ensemble du projet. C'est pour cela que nous avons choisi la bibliothèque *Pygame*, très simple d'utilisation, d'installation et de compréhension. Nos principales sources d'informations ont été *OpenClasseroom, StackOverFlow, Documentation Pygame.*

Afin de pouvoir réaliser notre première partie, nous avons cherché dans un premier temps comment l'on pouvait mettre en orbite un objet *dans un plan*. Pour cela nous avons réalisé des recherches sur le site pygame.org afin de voir s'il existait déjà un programme réalisant l'objectif recherché.
Nous avons alors trouvé un projet du nom de « *Orbit Duel* », grâce à ce dernier nous avons pu avoir un début d'approche sur le sujet.

A la suite de cette avancé sur cette partie du projet nous nous sommes interrogés sur la rotation des planètes, en effet, le problème était de savoir si l'on voulait que les planètes aient une rotation à partir d'un point fixe ou si elle devait suivre les lois physiques qui s'exercent dans la réalité. Pour évité de perdre du temps nous avons rapidement décidé qu'elles auraient une rotation qui varierai en fonction des paramètres que nous laisserons entre les mains du joueur. Le point de référence pour la rotation a donc était défini par le soleil au centre de l'écran de jeu. « *Orbit Duel* » n'étais donc plus utile pour nous, les idées sur les quelles nous nous basions étaient seulement les nôtres.

Nous nous sommes également questionnés sur les dimensions et l'échelle à laquelle on souhaitait voir évoluer le jeu. On voulait au départ réaliser un jeu en dimension réel convertie en pixel mais après une discution à propos du sujet avec notre professeur d'ISN, il nous a montré que réalisé un jeu avec de tels dimension serait impossible. C'est pour cela que nous avons restreint le nombre de planètes à 5 maximum (pour la fluidité du jeu également) afin d'éviter les problèmes d'échelle.

La deuxième partie du projet est centré autour de la gestion de forme vie. Pour cela nous avons créé un système de point de compétence et d'arbre de compétence afin de permettre au joueur de créer et améliorer ses formes de vies. Formes de vies qui peuvent être gérer dans un menu de statique pour chaque création, donnant des informations très précises pour chaque planète. Ce travail à étais plus rapide à mettre en place grâce au squelette de la partie précédente ainsi qu'à l'habitude de la bibliothèque *Pygame*.

Répartition des taches et démarche collaborative

(voir code couleur en annexes)

X	Valentin	Florian	Julien
Séance 1	Proposition de type de jeu, discutions générale	Travail de recherche sur notre future application	Réflexion et travail papier sur les objectifs du jeu
Séance 2	Imagination d'un début de scénario	Recherche des objectifs et contenus du je	Réflexion et travail papier sur les objectifs du jeu
Séance 3	Travail sur les interfaces graphiques du jeu	Réflexion sur l'organisation de la gestion des planètes en s'appuyant sur des formules scientifiques	Travail de recherche sur les planètes du système solaire et des corps célestes
Séance 4	Apprentissage du code orienté objet	Travail sur l'organisation de la gestion des planètes	Travail papier sur l'arbre de compétence Nº1 A et B
Séance 5	Développement des arbres de compétences	Apprentissage du code orienté objet et du multi-threading	Définition des événements aléatoires de la 1ère partie du jeu
Séance 6	conception du système de création des planètes et de leurs mises en place	Apprentissage de la bibliothèque pygame	Travail de pixel art sur le fond du jeu et design du soleil
Séance 7	événement aléatoire et leurs influences	Écriture d'un programme faisant animer une suite d'images afin de créer des animations	Création des premières planètes en pixel art

Séance 8	récoltes et utilisations des données à travers toutes les fonctions	Résolution de problèmes liés au multi-threading	Création de planètes en pixel art
Séance 9	mise en place d'interfaces graphique pour les menus de créations	Réflexion sur l'organisation de la gestion de la vie	Réflexion papier d'amélioration en termes de gameplay pour le jeu
Séance 10	arborescence du jeu et fusions des scripts	Définition des variables liées à la gestion des planètes	Travail papier sur l'arbre de compétence Nº2 A et B
Séance 11	Mise en place finale du système de création de planètes	Programmation de la gestion des planètes(composition...)	Programmation schématique du jeu des chapeaux
Séance 12	Mise en place du scénario sous aspect graphique	Programmation de la gestion des formes de vies	Programmation schématique du jeu de la roulette
Séance 13	Mise en place graphique et fonctionnelle des mini-jeux	Recherche d'image pour l'arbre de compétence B	Recherche d'image pour l'arbre de compétence Nº2 A
Séance 14	Création d'interface graphiques	Programmation de la gestion des formes de vies	Travail sur le scénario du jeu (Introduction, Milieu du jeu et fin du jeu)
Séance 15	Assembling , debugging et correction	Assemblage des différents codes et correction de bugs	Création des dernières planètes en pixel art

Nous avons communiqué via Etherpad, en effet, nous avons créé un pad au nom de notre projet. Ce site nous a vraiment était utile car il nous a permis d'échanger des morceaux de code, des liens vers des images utilisés plus tard pour l'aspect graphique du jeu ou encore des réflexions sur différents problèmes rencontré au cours du projet.

Durant une grande partie partie du projet Julien venait régulièrement chez moi pour que l'on puisse avancer plus efficacement. Il m'apportais toujours le travail que je lui demandais pour gagner du temps (écriture de scénario, pixel art, morceaux de code…).
Avec Florian, c'est surtout vers la fin du projet que nous avons fait preuve de travail d'équipe solide, dans le sens où nous avons passé des nuits à coder en communiquant via messages pour s'entraider et assemblé nos codes, interfaces, et s'apporter des éléments de correction.

Réalisation

Pour la réalisation du code, nous avons du faire face à une multitudes de petit problèmes qui à nous à pousser à utiliser des stratagèmes pour contourner ces derniers. Ici, nous verrons les exemples que je trouve les plus intéressants, et que je peux expliquer le plus efficacement. Nous verrons donc ces morceaux de code dans l'ordre où ils apparaissent dans le script principal. Ne tardons plus pour commencer et voyons le premier problème sur lequel j'ai buté. Il s'agit des Hitboxes. En effet, pygame ne nous fournit pas directement des zones cliquables à l'aide d'une simple fonction. J'avais quand même trouvé grâce à leurs documentations, qu'il existe deux fonctions qu'on pourrait mettre en commun pour faire une zone cliquable. Il s'agit de :

- pygame.Rect((posx posy), (x, y))
- pygame,Rect.collidepoint(event.pos)

Je savais que nous devrions utiliser ces commandes des centaines de fois avec des paramètres différents et qu'il s'agirait d'une grande perte de temps. J'ai alors décidé d'apprendre les bases de la POO, pour en faire une *class* :

```
class Hitboxes:
    def __init__(self,taille_x,taille_y,place_x,place_y,opacite,point):
        self.rect = pygame.Rect((place_x, place_y), (taille_x, taille_y))
        self.surf = pygame.Surface(self.rect.size)
        self.surf.fill((255,0,255))
        self.surf.set_alpha(opacite)
        self.point = point
```

En effet, 88 objets sont issus de cette *class*, et ils sont appelés plus de 375 dans le code. Grâce à ce tout petit morceau de code, j'ai pu utiliser des *class* sans aucuns problème dans le reste du code. Ce qui m'avais fait penser à faire pareil pour les planètes, car effectivement, il s'agit toujours du même objet avec les mêmes fonctions, mais seulement des paramètres différents.

C'est là qu'un problème était survenu et m'avait fait perdre tout espoir. Le problème était le suivant :
« Je peux créer des planètes dans le script en appelant la bonne *class*, mais je ne peux pas faire générer des objets qui ne sont pas déjà là. »

Je ne savais pas comment faire pour donner la possibilité au joueur de créer ses planètes une à une. Et j'ai trouver une solution improbable, j'ai crée une « banque de planète ». En effet, j'ai fais en sorte de créer tous les objets dés le début grâce à une fonction « *définir* » :

```
plan_log1 = definir(text_nom,text_taille[balise_taille],text_masse,text_type[balise_type],look_check[i_look],0)
plan_log2 = definir(text_nom,text_taille[balise_taille],text_masse,text_type[balise_type],look_check[i_look],0)
plan_log3 = definir(text_nom,text_taille[balise_taille],text_masse,text_type[balise_type],look_check[i_look],0)
plan_log4 = definir(text_nom,text_taille[balise_taille],text_masse,text_type[balise_type],look_check[i_look],0)
plan_log5 = definir(text_nom,text_taille[balise_taille],text_masse,text_type[balise_type],look_check[i_look],0)
```

fonction qui retourne un objet de la *class* voulu, à savoir celle pour les planètes. Mais alors où sont ces planètes dans le jeux alors qu'elles sont toutes crées pendant le lancement ? Elle ne sont tout simplement pas affichées ! Dans le menu de création le joueur va modifier les variables que la fonction « *définir* » prend en compte, puis les affecter aux planètes tour à tour. La variable « *creation* » va s'agrémenter de 1 à chaque fois qu'une planète serra demandée, affichant la première puis la seconde et ainsi de suite. Comme une image parle pour mille mots :

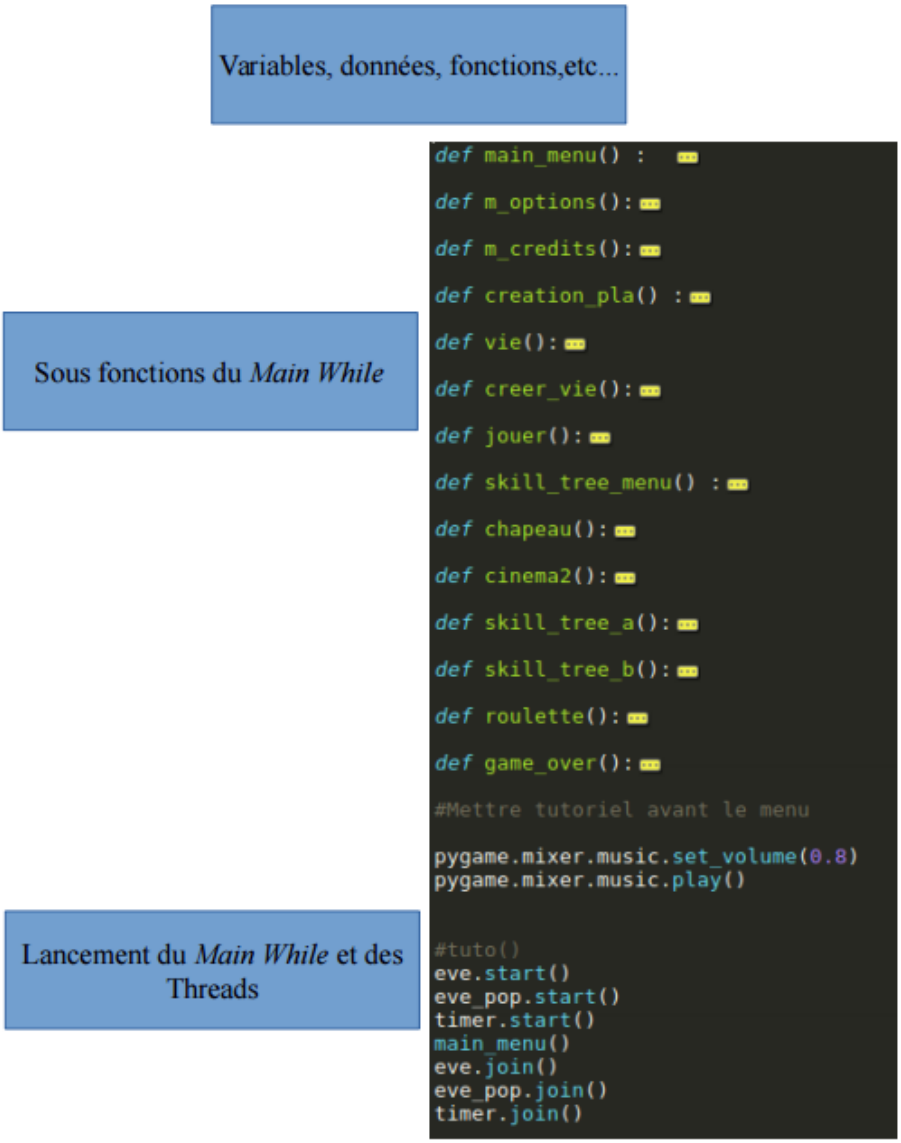
```
if creation == 0 :
  log1_1, log1_2, log1_3, log1_4, log1_5 = text_nom, text_taille[balise_taille],
  plan_log1 = definir(log1_1, log1_2, log1_3, log1_4, log1_5,random.randint(0,2))

  alerte_check = True
  creation += 1

elif creation == 1 :
  log2_1, log2_2, log2_3, log2_4, log2_5 = text_nom, text_taille[balise_taille],
  plan_log2 = definir(log2_1, log2_2, log2_3, log2_4, log2_5,random.randint(0,2))

  if plan_log1.nom == plan_log2.nom  :
    alerte_same_name = True

  else :
    alerte_check = True
    creation += 1
```

```
if creation >= 1 :
  plan_log1.dessiner()
if creation >= 2 :
  plan_log2.dessiner()
if creation >= 3 :
  plan_log3.dessiner()
if creation >= 4 :
  plan_log4.dessiner()
if creation >= 5:
  plan_log5.dessiner()
```

Il y a eu pleins de petites choses qui semblent simple et qui enfaîte ne le sont pas quand on doit le rendre sous forme code. J'ai très vite compris par la force des choses que *Pygame* avait ses avantages comme ses inconvénients. La première chose qui m'a demander de me concentrer sur les documentions de *pygame,* c'est quand j'ai dû coder une manière de recevoir les *inputs* du clavier du joueur pour sélectionner le nom d'une planète par exemple. Et étrangement, il n'y avait rien pour m'aider sur internet. Mais en quelques heures, le progrès est présent.

Il y a eu bien d'autres problématiques majeures de ce genre à résoudre. Nous devrions présenter nos fonctions ainsi que leurs rendus, mais leurs nombres étant trop important nous nous pencherons sur l'aspect structurel du code. En effet, due à l'utilisation de *pygame*, nous avons dû utiliser une technique plutôt particulière. Ce dernier nécessite une faire tourner le code dans une boucle *while* pour que la fonction « *pygame.display.update()* » puisse s'exécuter à une certaine fréquence (ici 60 fois par seconde) pour permettre des animations ou tout autre modifications, d'apparaître sur l'écran.

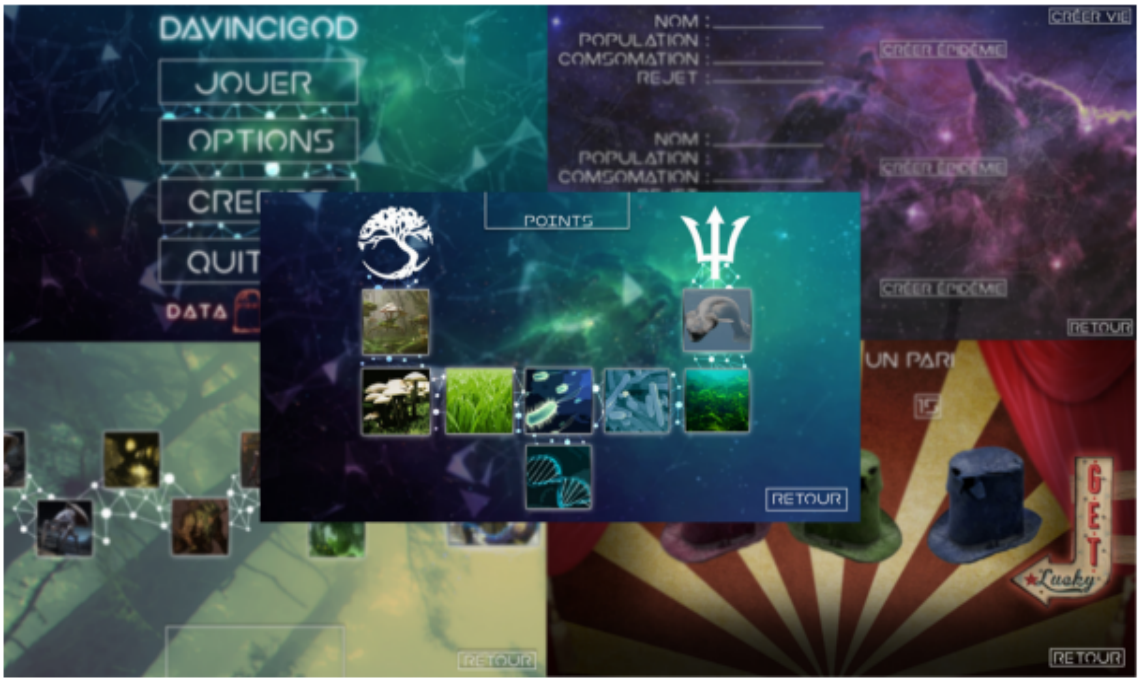
Nous avons donc un *main while* qui lancera les autres boucles correspondantes aux menus demandées. Nous avons donc au final une fonction pour chaque menus du jeu qui tournera dans une autre boucle *while.* Cette dernière prendra fin quand sa condition ne serra plus valide, pour laisser place à une autre boucle pour un autre menu. Ce qui donne un aspect assez particulier à notre script principal.

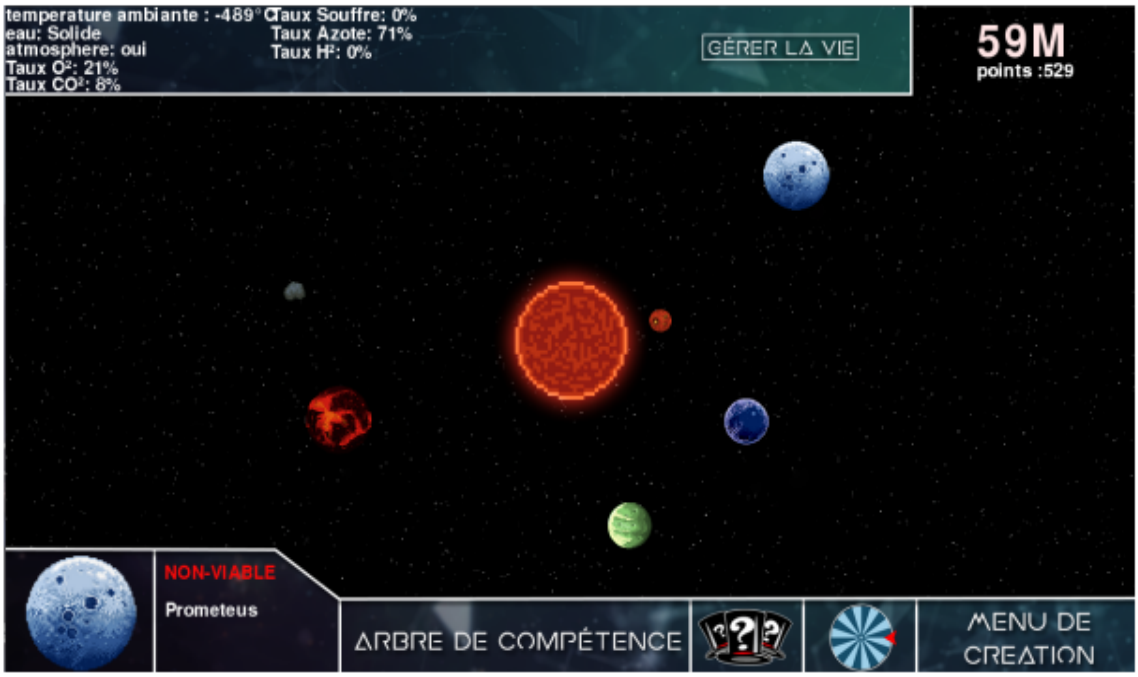


Intégration et validation

M'étant occupé totalement de l'aspect graphique des interfaces du jeu, j'ai dû m'occuper par la même occasion d'une grande partie du code, notamment tout ce que nous avons vu dans la partie au dessus, à savoir la mise en animation et place des corps célestes, la mise en commun et création des variable des arbres de compétences ainsi que leurs fonctionnement, et surtout la structure du code dans l'enchaînement des fonctions dans tout les menus. J'ai en parallèle mis en place les deux mini-jeux disponible au joueur depuis l'écran de contrôle.

Après assemblage de tout nos travaux, le jeu correspond bien aux normes que nous nous étions fixées.





Bilan et perspectives

Pour développer le projet, la première idée qui me vient à l'esprit est tout simplement de tout refaire, dans le sens où il faudrait opter pour un langage bien plus performant et optimisé dans la POO. Je pense notamment au C#.
Dans les conditions actuelles, pour améliorer l'expérience de jeu, il aurait était bien d'avoir accès à un arbre de compétence pour chacune des planètes. Prolonger la longévité du jeu est aussi une bonne idée.

Ce projet ma permit d'apprendre davantage les subtilités de la programmation, notamment avec la POO et le multi threading. Cela m'a aussi donné l'envie d'apprendre d'autres langages pour pouvoir toucher à tout, comme le JAVA, HTML/CSS, C#.

ANNEXES

Code couleur du tableau commun : Travail en groupe entier, Travail en scindé en équipe deux, Travail personnel.

Langage utilisé : Python 3.5

Lien vers le code : https://github.com/DankLordOfTheMemes/DaVinciGod

Sites utiles :
- https://github.com
- https://stackoverflow.com/
- https://www.pygame.org/