# *Table des matières*

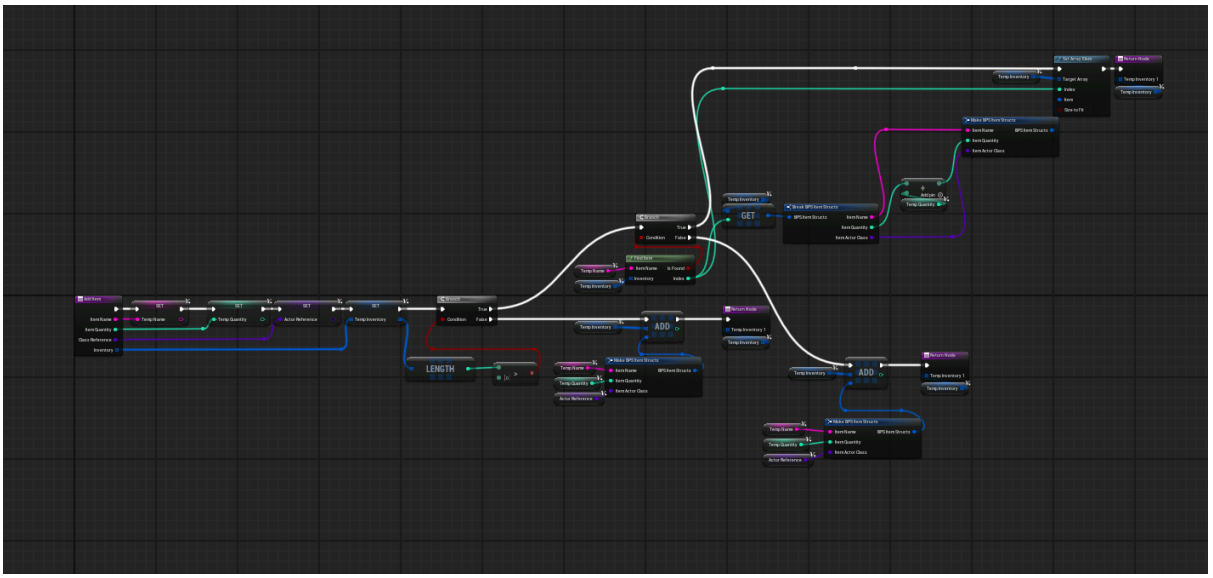
**Structure de données – Niveau 1** **1**
**Structure de données – Niveau 2** **2**
**Structure de données – Niveau 3** **4**
**Conclusion** **5**

## *Structure de données – Niveau 1*

Pour la structure de données de niveau 1, j'ai décidé d'y aller avec une Liste. Dans mon jeu, j'ai mis en place un inventaire qui permet d'ajouter les objets que le joueur ramasse tout au long de sa partie. Quand le joueur utilise un objet, la quantité diminue, et quand elle arrive à zéro, l'objet se retire automatiquement de l'inventaire.

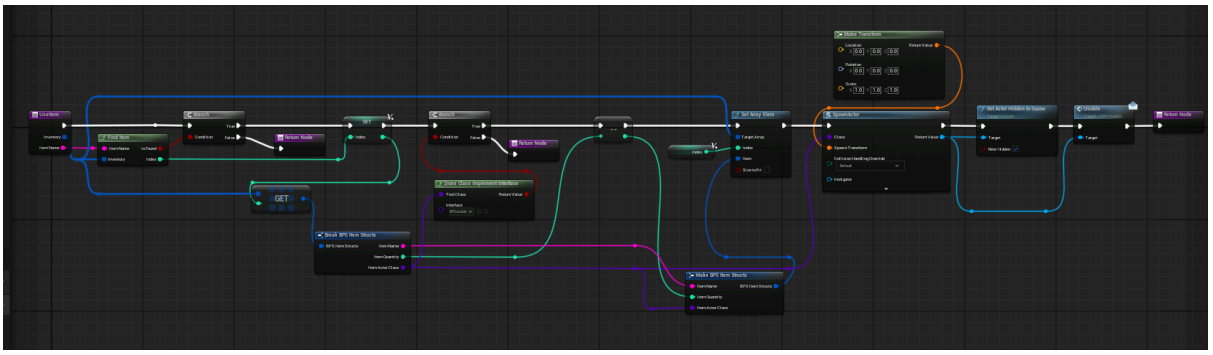
J'ai choisi une Liste plutôt qu'un Tableau, parce que le joueur peut ramasser autant d'objets qu'il veut, sans limites. Une liste est plus pratique dans ce cas, car elle n'a pas de taille prédéfinie, contrairement à un tableau où le nombre d'éléments doit être déterminé dès le début. De plus, avec un Tableau, si le joueur atteint la limite, j'aurais été obligé de reconstruire le tableau avec un index plus grand et de déplacer tout son inventaire dans le tableau plus grand pour éviter une erreur de type "out of bound".

Function ADD:




Explication brève: dans la fonction ADD, cette fonction a pour but de voir si l'item que le joueur prend existe déjà ou non. Donc, quand le joueur prend un item, cela vérifie si la List.Length > 0 si c'est faux, cela va garder dans la Liste: le nom, la quantité voulue et la classe (Actor Class). Si à l'inverse c'est vrai, cela va vérifier si l'item existe déjà avec son nom avec l'aide de la fonction "Find Item" si c'est vrai, on va juste ajouter la quantité, si cela est faux, on va garder les informations dans la List.

Function UseItem:



Explication brève: Ici, lorsque le joueur utilise un item, cela va récupérer l'index où se trouve l'item et avec le "Get Clone" on va récupérer la quantité qu'on va soustraire avec 1 et on va récupérer la classe Actor pour récupérer l'interface de UseItem et ce que l'UseItem fait.

Mais, sinon, je n'ai pas eu de problème lors de la création de la liste. C'était plutôt facile à mettre en place.

## *Structure de données – Niveau 2*

Pour la structure de données de niveau 2, j'ai décidé d'y aller avec la LinkedList. Cette structure a été utilisée dans mon jeu pour un système de combos. En d'autres mots, lorsque le joueur récupère un objet appelé (PowerCell), elle devient la tête (head) de la LinkedList si la liste est vide. Sinon, un nouveau nœud est ajouté à la fin de la liste (tail). Cela permet de sauvegarder dans l'ordre tous les PowerCells que le joueur récupère.

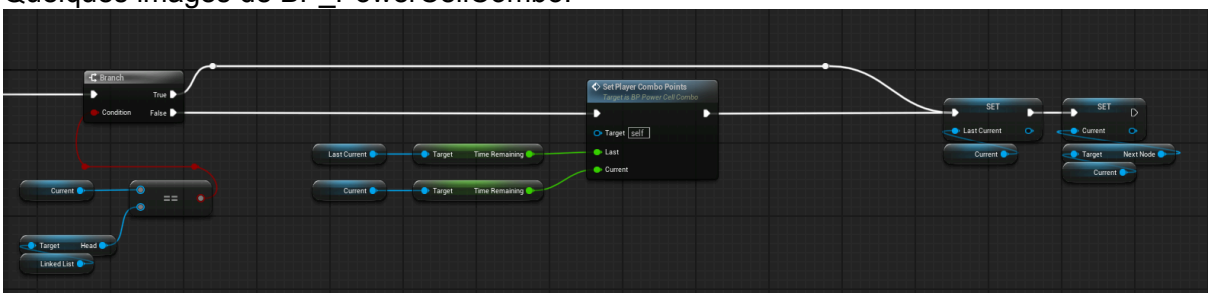
Lorsque le joueur va ramasser un deuxième PowerCell, cela va calculer la différence entre le nœud précédent (Last.RameningTime) et lui du PowerCell (Current.RemeningTime). Avec la différence, cela va déterminer le combo donné: plus que les secondes sont plus proches de 0, le combo va être plus grand et, si la différence est plus grande, la valeur va être plus petite, voire absente. Ensuite, cette valeur est multipliée par le score du joueur.

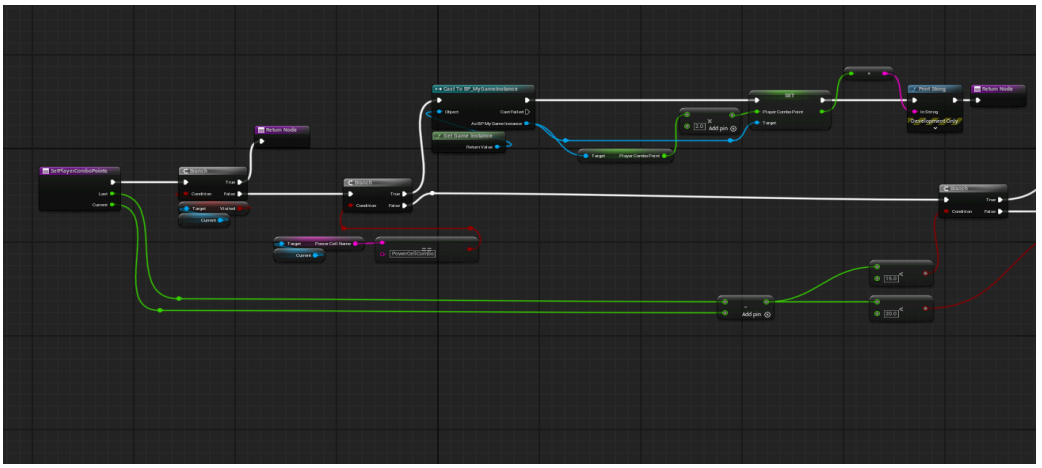
De plus,j'ai également créé un item appelé Combo_x2. Lorsque le joueur utilise cet objet, cela va créer un nœud temporaire (PowerCell_Combo) dans la LinkedList, puis démarrer un minuteur. Si, durant le minuteur, le joueur collecte un autre

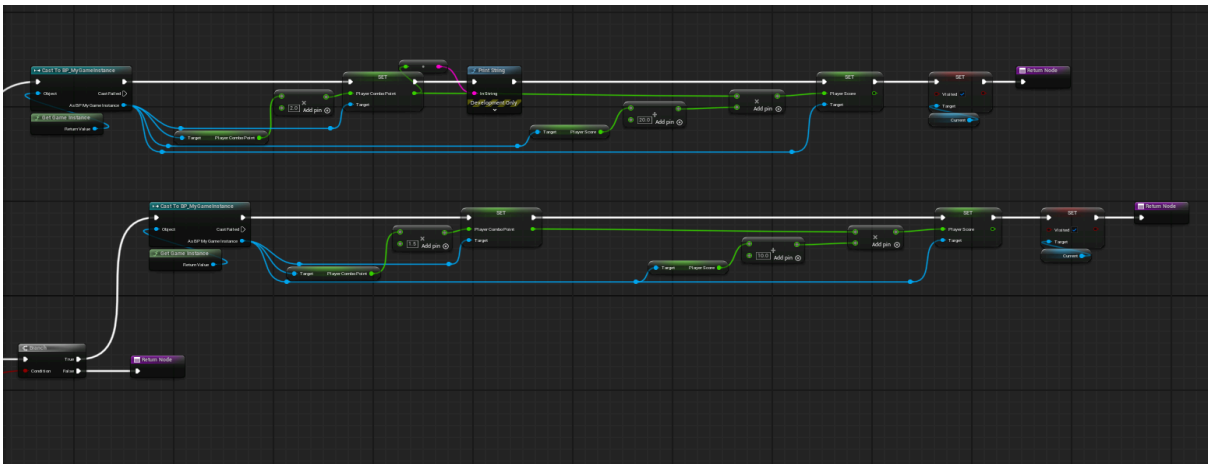
PowerCell, cela  va vérifier si, dans la liste, il y a un PowerCell_Combo. Si c'est oui, cela va doubler, le combo obtenu va être doublé avant d'être multiplié par le score du joueur. Une fois que la minuterie finie, tous les nœuds temporaires (PowerCell_Combo) vont être supprimés de la LinkedList, ensuite ces nœuds vont tous être reconnectés.

Puis, pourquoi j'ai choisi la LinkedList est dû au fait que la LinkedList est plus avantageuse ici. Comme je supprime des nœuds (PowerCell_Combo) dans la liste, cela prend une complexité de O(1) lors de la suppression et de la reconnexion des nœuds comparer à la Liste qui prend une complexité de O(n).

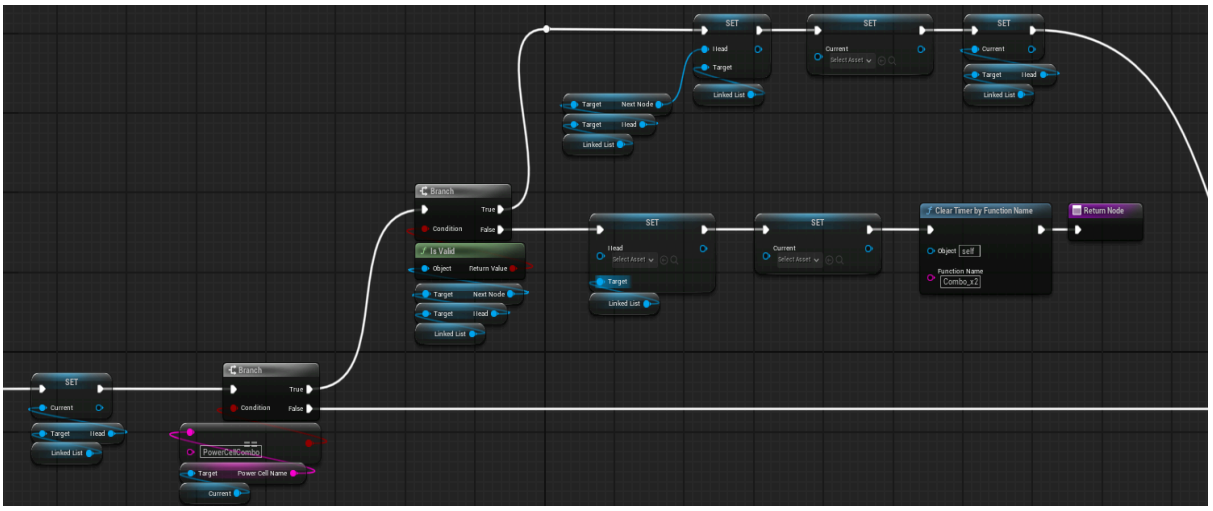
Quelques images de BP_PowerCellCombo:

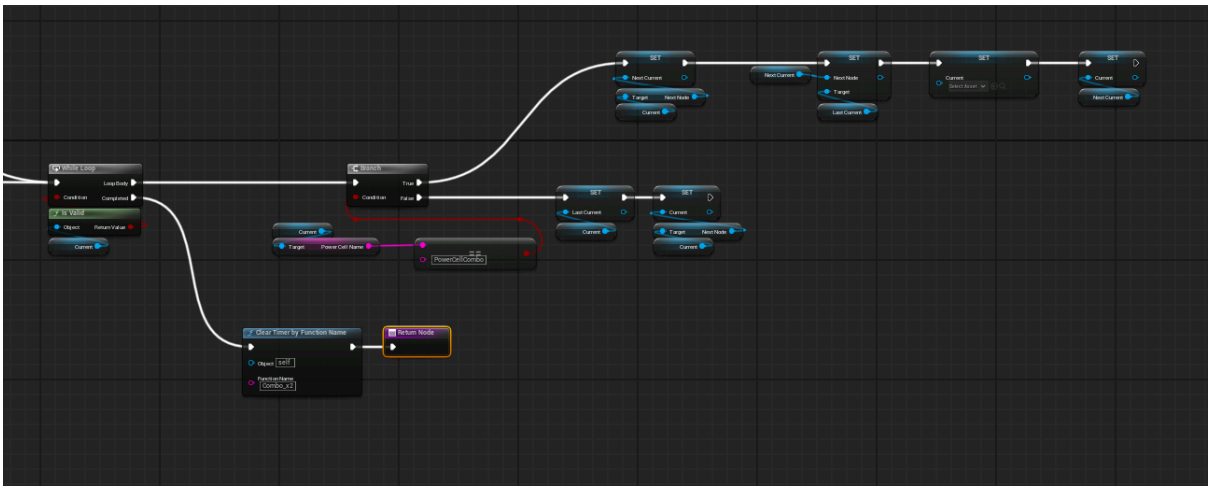






Quelques images de BP_Combo_x2 (item):





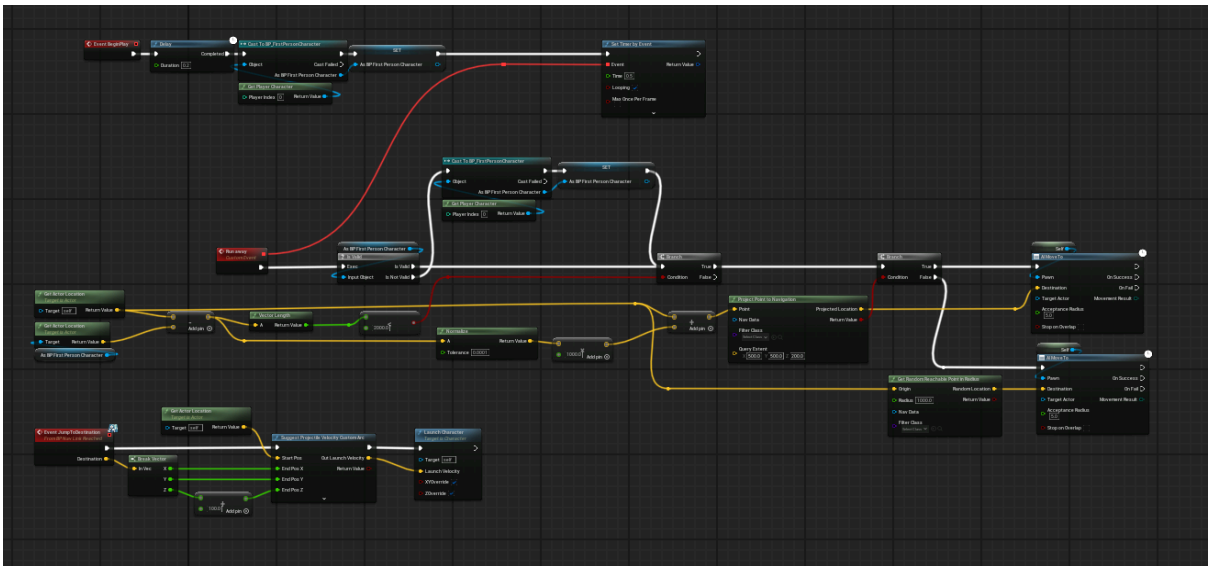
En ce qui concerne l'implémentation, j'ai trouvé cela assez difficile, car il n'y avait pas vraiment de ressources (documentations ou tutoriels vidéo) sur la manière d'implémenter une LinkedList en Blueprint. J'ai donc dû chercher des alternatives pour obtenir le même résultat : créer un objet Blueprint avec les variables souhaitées, puis définir une fonction Node qui les renvoie les variables, créer un nouveau Blueprint avec une nouvelle fonction Append pour instancier la tête ou ajouter les nœuds à la fin de la liste, etc.

## *Structure de données – Niveau 3*

Pour la structure 3, j'ai utilisé le graph comme structure de données. Comme dans mon jeu, je voulais avoir un ennemi qui s'enfuit du joueur quand il le voit. J'ai décidé

d’utiliser le NavMesh d’Unreal, qui est une sorte de graphe de navigation pour l’IA, et j’ai ajouté le nœud « AI Move To », qui calcule le chemin le plus court à travers entre deux points (utilisant l’algorithme A*). Dans notre cas, il s’agit de fuir le joueur. De plus, je lui ai ajouté des NavLinks pour lui permettre quand il voit du vide de sauter pour se rendre de l’autre côté avec l’aide du NavLink.

Image du BP_AI:



## *Conclusion*

La structure que j’ai le plus approfondie au cours du projet est la LinkedList, puisque, comme je l’ai mentionné dans la section 2, peu de vidéos ou de documents existent sur la façon d’implémenter cette structure de donnée en Blueprint. J’ai donc dû réfléchir par moi-même à la manière dont je pouvais adapter ce qu'on a vu en cours, qui est très différente de l’aspect code.

De plus, j’ai trouvé les arbres (Trees) et les graphes (Graphs) et l’algorithme A* étaient difficiles à maîtriser dans le cadre de ce cours. Cela n’a rien à voir avec la façon dont ils fonctionnent en tant que structures, mais plutôt avec la manière dont on les intègre à Visual Studio, Unity et Unreal. Comme on a consacré moins de pratique avec LeetCode, je trouve que les 3 aspects précédents sont ceux que j’ai trouvés plus durs à comprendre.

Autrement, la suite de mon parcours dépendra en grande partie de mon stage prévu l'hiver prochain : si je m'y plais et qu'il débouche immédiatement sur un poste, je resterai probablement dans cette entreprise. Dans le cas contraire, j'envisagerais de poursuivre mes études à l'université pour approfondir mes compétences, soit en programmation de jeux vidéo, soit en intelligence artificielle.

Les projets qui m'ont le plus marqué sont le jeu de clones pour résoudre des énigmes et le labyrinthe généré aléatoirement. Tous deux offrent la possibilité de jouer quasiment à l'infini : d'un côté, le concept de clones crée une multitude de façons de résoudre ou de concevoir des puzzles ; de l'autre, le labyrinthe aléatoire garantit une expérience toujours renouvelée.

Pour être honnête, je ne pense pas poursuivre le développement de mon propre jeu cet été. En revanche, je l'utiliserai comme exemple de mise en œuvre d'algorithmes et de raisonnement autour de ceux-ci. Comme je vais constituer mon portfolio en vue de mon prochain stage ou emploi, je m'appuierai sur ce projet pour démontrer mes compétences et me préparer au monde professionnel.