KaizerWald
Castle Defense

Florian Duruz

Un Travail présenté dans le cadre d'une formation

CFC

Formation Professionnelle Accélérée

Centre professionnel Nord Vaudois

Suisse

Mai 2022

RÉSUMÉ

TABLE DES MATIÈRES

Table des figures **4**

Glossaire **5**

1 Analyse préliminaire **6**
1.1 Introduction . . . 6
1.2 Objectifs . . . 7
1.3 Planification initiale . . . 8

2 Analyse / Conception **10**
2.1 Convention de nommage . . . 10
2.2 Concept . . . 12
2.2.1 Menu Principal . . . 13
2.2.2 Régiments . . . 15
2.3 Pathfinding HPA . . . 31
2.3.1 Introduction . . . 31
2.3.2 Décomposition des tâches . . . 32
2.3.2.1 Cluster . . . 33
2.3.2.2 Les Portes . . . 34
2.4 Stratégie de test . . . 36
2.5 Risques techniques . . . 37
2.6 Dossier de conception . . . 38

3 Réalisation **39**
3.1 Dossier de réalisation . . . 39
3.2 Description des tests effectués . . . 39
3.3 Erreurs restantes . . . 39

4 Conclusion **40**
4.1 Objectifs atteints / non-atteints . . . 40
4.2 Points positifs / négatifs . . . 40
4.3 Difficultés particulières . . . 41
4.4 Suites possibles pour le projet . . . 41
4.5 Bilan Personnel . . . 41

A Appendix **42**
A.1 Journal de Travail 42

Bibliographie **44**

TABLE DES FIGURES

1.1 Total War : Tir en formation 7

2.1 Feuilles de Style 14
2.2 Note : pour simplifier la compréhension, l'espacement entre les unités est de 0 . 23
2.3 Réarrangement automatique d'une collection type List 30
2.4 Réarrangement suivant l'algorithme 30

GLOSSAIRE

API (Application Programming Interface) : Collection de fonctionnalités permettant aux services d'une applications de communiquer entre eux. 30, 43

Framework : Collection de librairies ayant des fonctionnalités associées via une API unifiée. 43

Mod : Un mod (abréviation de modification) est une modification par une personne tierce d'un jeu vidéo existant, se présentant sous la forme d'un greffon qui s'ajoute à l'original, pour ajouter une fonctionnalité ou modifier les fonctionnalités existantes. . 6

Real Time Strategy Game : Jeu de stratégie en temps réel. 20

CHAPITRE 1

ANALYSE PRÉLIMINAIRE

1.1 Introduction

Le Projet consiste à la création d'un jeu de stratégie en temps réel sous la forme d'un Castle Defense(défense de château) un jour surtout représenté dans le monde du modding et un Mod très populaire dans le RTS[1] Warcraft 3 ; ce projet est réalisé dans le cadre d'un travail d'un travail personnel individuelle(TPI) pour l'établissement du CPNV.

L'objectif de ce travail est avant tout de travailler sur l'intelligence artificielle plus précisément sur l'algorithme lié à la recherche de chemin qui est le centre du projet, le combat sera aussi abordé mais restera dans un forme plus basique et se concentrera sur l'analyse du comportement des entités dans le cadre d'un combat au sein d'un groupe/régiment. Ce projet s'inscrit dans la continuité du Pré-TPI qui avait pour but de me familiarisé et de poser les bases des algorithmes utilisés dans ce projet.

Les algorithmes suivants ont été mis en place :

- **A*** : probablement l'algorithme le plus populaire qui a l'avantage de toujours trouver le chemin le plus court et est aussi très performant.
- **FlowField** : Algorithme développer pour répondre un à besoin bien spécifique aux RTS, permet à tout un groupe d'avoir la direction a prendre via un champs de vecteurs placé sur le terrain, chaque Vecteur Indiquant le chemin à prendre.

Il y a cependant un problème millénaire qui frappe les jeux utilisant ces algorithmes, plus le terrain est grand plus le processeur peine à calculer les chemins, ce phénomène est à multiplier par le nombre d'entités devant calculer ces chemins.

Un algorithme plus sophistiqué à vu le jour pour répondre à ce problème, le HPA ou hierarchical Pathfinding qui sera au coeur du projet et qui régira les mouvement des entités dans le projet.
Une attention particulière sera aussi apporté à l'architecture et à la structure du code, ce projet étant destiné à avoir une continuité, il est impératif que le code soit lisible afin d'assurer un maintien continu.

1. Real Time Strategy Game (Jeu de stratégie en temps réel)

1.2 Objectifs

Gestion de groupe des entités

Cette partie comprendra l'interaction du joueur avec des entités fonctionnant en groupe les principaux objectifs comprendront :

- **Sélection par groupe** : Une entité ne sera jamais sélectionnable individuellement, quand le jouer sélectionne une unité tout sont régiment sera sélectionné.
- **Déplacement en groupe** : Sûrement la partie la plus complexe, ce sujet à fait coulé beaucoup d'encre et ne semble pas avoir trouvé de solution uniforme, chaque solution à ce jour correspondant à des besoins bien spécifiques.

  L'objectif sera de développer un algorithme en conjonction avec l'algorithme Pathfinding [2] (décrit plus bas) permettant aux entités de se déplacer comme une seule unité et de donner un semblant de cohésion lors des déplacements.
- **Combat en formation** : Autant pour les attaques à distance que au corps à corps, les régiments devront se battre comme une unité et ne pas briser le rang, dans le cadre du corps à corps, la formation pourra être déformée mais jusqu'à une certaine limite fixée arbitrairement.

  Ce point sera un élément central en terme de jeu, car la formation aura un impacte directe sur la capacité du régiment en combat.

  Combat à Distance : Les troupes ne pourront tirer que sur une rangée(voir image : 1.1) inspiré du système de tir des anciens jeux total war, une formation allongée offrira donc une plus grande capacité de feu, cependant cela se fera au détriment de la capacité de défense(décrite au point suivant)
  Combat en Mêlée : Une formation plus compacte offrira une meilleure capacité de défense quand le régiment est engagée au corps à corps.

Figure 1.1 – Total War : Tir en formation

Algorithme Pathfinding HPA [3]

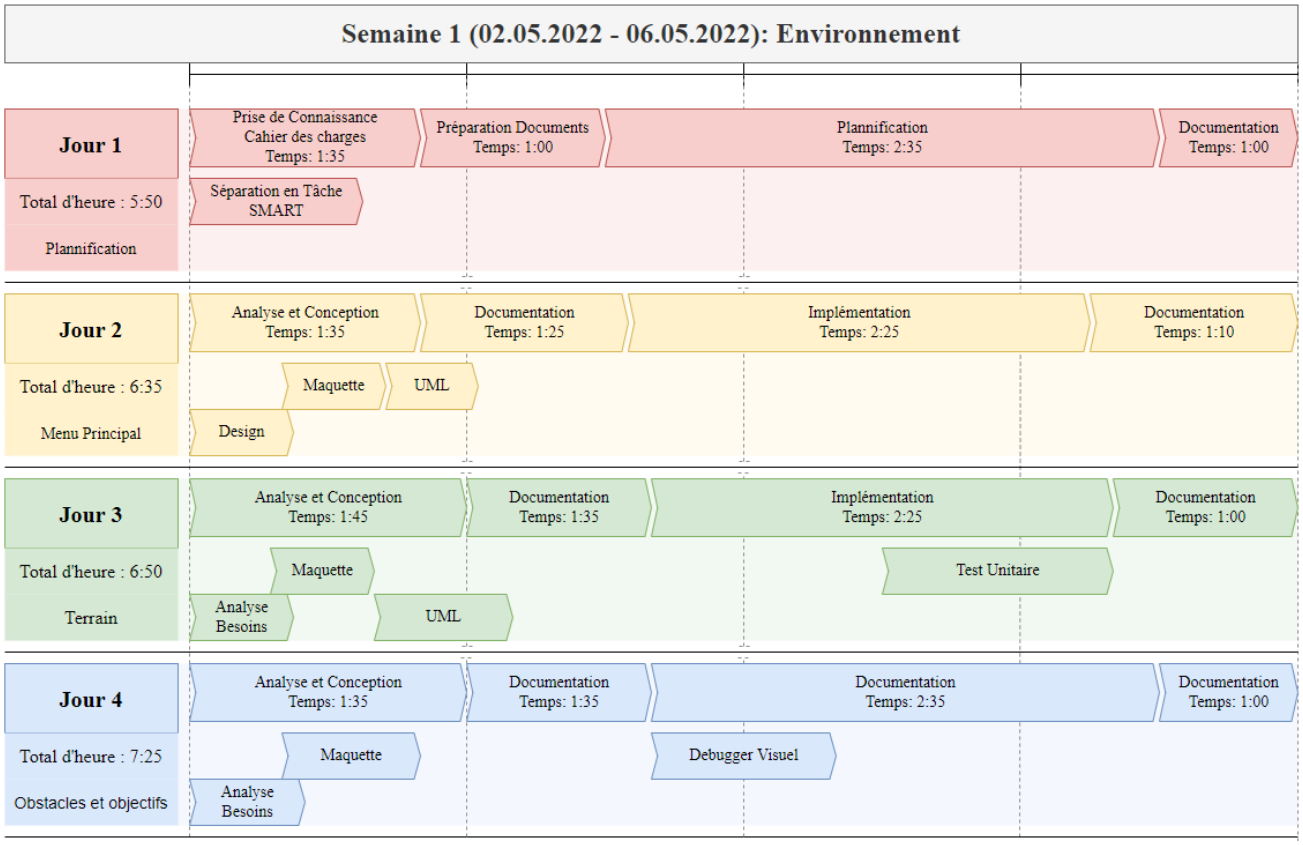
Sur la base des travaux effectués lors du pré-tpi, il faudra mettre en place un système de hierarchical Pathfinding A* en apportant un changement afin d'adapter l'algorithme au besoin de projet, à savoir l'utilisation d'un Flowfield (ou champs de vecteurs).

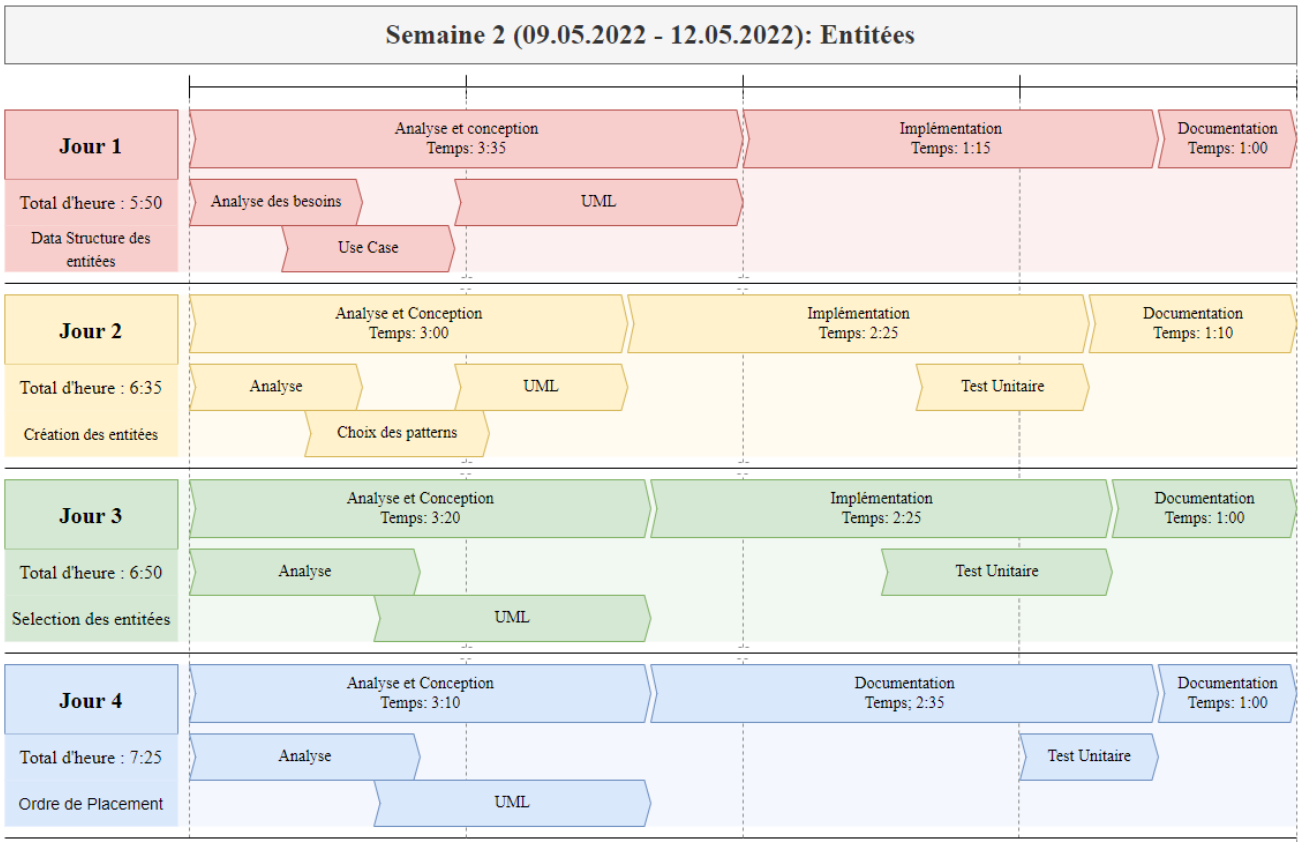
Pour ce faire, les éléments suivant devront être implémenté :

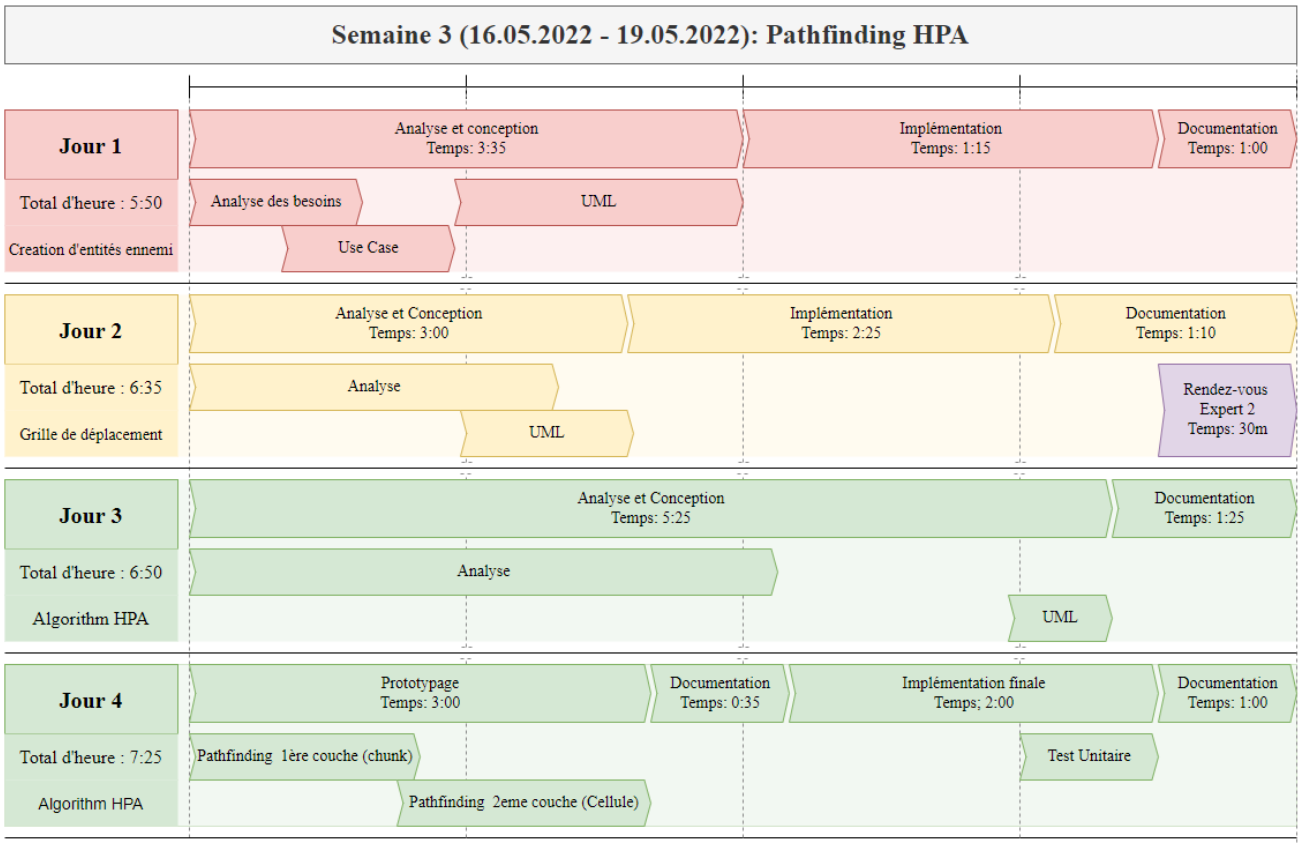
- **Génération d'une grille partitionnée** : Il faudra en premier lieu générer un grille sur tout le terrain en utilisant l'outil mis en place précédemment. Cette dernière devra être partitionnée afin de pouvoir cibler les futures calculs à une région spécifique.
- **Structure de donnée** : Chaque partition devra contenir un certains nombre d'informations afin de permettre le bon fonctionnement de l'algorithme(détaillés dans la partie analyse de ce document)
- **Adaptation de l'algoritme HPA au Flowfield** : La version traditionnelle du HPA utilise uniquement le A*(il s'agit en réalité d'une abstraction en ou plusieurs couche de cet algorithme), il est malgré tout possible d'y inclure des algorithmes différents dont le flowfield qui sera utilisé.

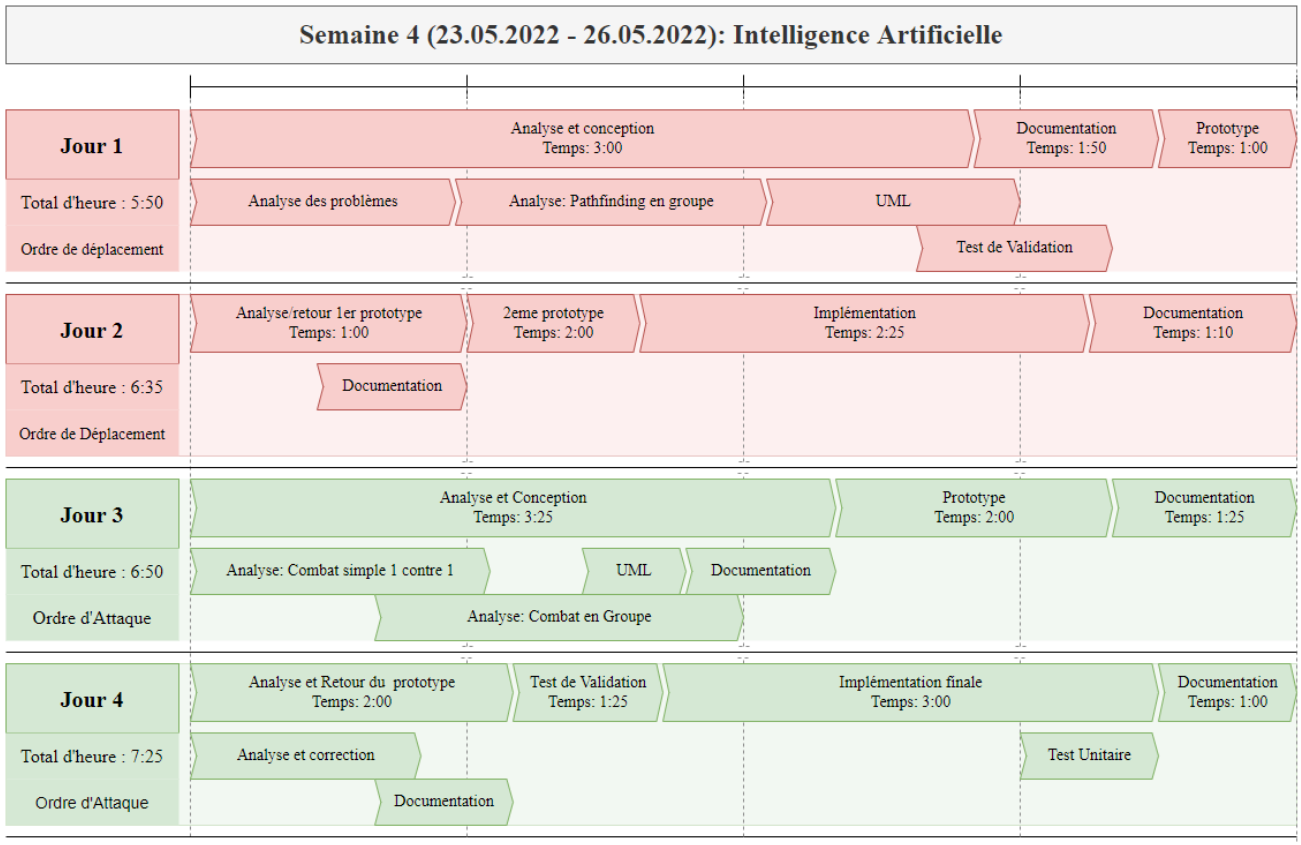
2. Pathfinding : Recherche de Chemin
3. Hierachical Pathfinding A*

1.3 Planification initiale






Semaine 3 (16.05.2022 - 19.05.2022): Pathfinding HPA
Jour 1
Total d'heure : 5:50
Creation d'entités ennemi
Analyse et conception
Temps: 3:35
Implémentation
Temps: 1:15
Documentation
Temps: 1:00
Analyse des besoins
UML
Use Case
Jour 2
Total d'heure : 6:35
Grille de déplacement
Analyse et Conception
Temps: 3:00
Implémentation
Temps: 2:25
Documentation
Temps: 1:10
Analyse
UML
Rendez-vous
Expert 2
Temps: 30m
Jour 3
Total d'heure : 6:50
Algorithm HPA
Analyse et Conception
Temps: 5:25
Documentation
Temps: 1:25
Analyse
UML
Jour 4
Total d'heure : 7:25
Algorithm HPA
Prototypage
Temps: 3:00
Documentation
Temps: 0:35
Implémentation finale
Temps; 2:00
Documentation
Temps: 1:00
Pathfinding 1ère couche (chunk)
Test Unitaire
Pathfinding 2eme couche (Cellule)


Semaine 4 (23.05.2022 - 26.05.2022): Intelligence Artificielle
Jour 1
Total d'heure : 5:50
Ordre de déplacement
Analyse et conception
Temps: 3:00
Documentation
Temps: 1:50
Prototype
Temps: 1:00
Analyse des problèmes
Analyse: Pathfinding en groupe
UML
Test de Validation
Jour 2
Total d'heure : 6:35
Ordre de Déplacement
Analyse/retour 1er prototype
Temps: 1:00
2eme prototype
Temps: 2:00
Implémentation
Temps: 2:25
Documentation
Temps: 1:10
Documentation
Jour 3
Total d'heure : 6:50
Ordre d'Attaque
Analyse et Conception
Temps: 3:25
Prototype
Temps: 2:00
Documentation
Temps: 1:25
Analyse: Combat simple 1 contre 1
UML
Documentation
Analyse: Combat en Groupe
Jour 4
Total d'heure : 7:25
Ordre d'Attaque
Analyse et Retour du prototype
Temps: 2:00
Test de Validation
Temps: 1:25
Implémentation finale
Temps: 3:00
Documentation
Temps: 1:00
Analyse et correction
Test Unitaire
Documentation

CHAPITRE 2

ANALYSE / CONCEPTION

2.1 Convention de nommage

Général

S'aligne en général avec les conventions de C#[4] sauf pour le typage qui doit être explicite. La raison est que les algorithmes utilisent beaucoup de mathématique et les types sont sujet à beaucoup de conversions, afin de facilité la lecture, le typage est explicite afin que le lecteur n'ai pas à traquer les conversions de type. De plus on fait gagner du temps au lecteur en affichant directement au lecteur, ce dernier n'ayant pas besoin de regarder à droite du "=" pour trouver le type.

- **Indentation** : A la ligne(ou block) selon les conventions C#.
- **Indentation Inline** : Les méthodes de type "Setter" de champs doivent utiliser le body-expression de C#(=>) au lieu des crochets({ }).
- **Typages** : Explicite uniquement.

Langue du projet

S'alignant sur les conventions principalement utilisées en Suisse romande

- **Code** : Anglais.
- **Commentaire** : Français.
- **Commits github** : Français.
- **Documents externes** : Français.

Code

Suis globalement les conventions Usuelles de C# à l'exception des variables privés qui normalement a le préfixe "_", ce dernier a été abandonner car il s'agit d'une vieille pratique qui viendrait selon certaines sources du C et qui a été reprise mais qui ne fait pas sens car contrairement au C nous n'avons pas besoin de préciser si la variable et globale ou locale, ce préfixe n'ajoute donc rien.

- **Méthode** : CamelCase - 1ère Lettre Majuscule - Sans préfixe.
- **Interface** : CamelCase - 1ère Lettre Majuscule - préfixe "I".

- **Champs privée** : pascalCase - 1ère Lettre Minuscule - pas de préfixe.
- **Champs publique** : CamelCase - 1ère Lettre Majuscule - pas de préfixe.
- **Propriété** : CamelCase - 1ère Lettre Majuscule - pas de préfixe.
- **Variable privée** : pascalCase - 1ère Lettre Minuscule - pas de préfixe.
- **Constante** : SNAKE_CASE - Tout en Majuscule - pas de préfixe.

Particularité Unity

- **SerializedField** : Les champs ou proriétés ayant l'attribut [SerializedField] prennent une majuscule (même si le champs est privé).
- **Indentation** : les événements liés à Monobehaviour(Awake, Start, Update,etc...) ne doivent jamais utiliser le Body-expression de C#, Même si il n'y a que une ligne.

2.2 Concept

Le concept complet avec toutes ses annexes : Par exemple :

- Multimédia : carte de site, maquettes papier, story board préliminaire, …
- Bases de données : interfaces graphiques, modèle conceptuel.
- Programmation : interfaces graphiques, maquettes, analyse fonctionnelle…
- …

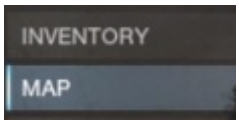
2.2.1 Menu Principal

Le Menu principale peut sembler être un sujet triviale, par définition le joueur n'est pas sensé y passer beaucoup de temps, cependant s'agissant de la première image du jeu présenté au joueur et donc la première impression donnée au joueur il est important de soigner son aspect ; le menu principal sera en effet le premier élément jugé par le joueur.

Design : Qu'est-ce qu'un bon menu principal ?

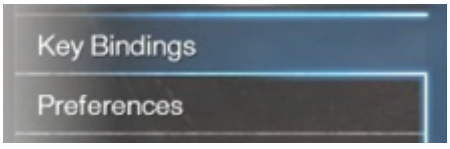
Malgré ne nombreuse recherche il ne semble pas y avoir un convention ou des règles établies, j'ai donc décider de m'inspirer d'un jeux ayant un thème similaire : New World crée par Amazone et dont l'interface graphique avait été très appréciée.
Sans entrée dans le design, prenons quelques points qui de mon point de vu ont un rôle dans le succès de l'interface.

- **Fondu** :
  la fenêtre n'est pas un carré régulier, les bords sont irrégulier donnant une impression de fondu adoucissant l'effet de rupture avec le jeu,
- **Sobriété** :



  Aucune couleurs vives même les éléments actifs restent dans des tons ternes.
- **Transition douce** :

  Un bouton ne change jamais entièrement de couleur, la transition est comme une lumière qui serait sous le bouton nous laissant suggérer la forme qu'avait le bouton, ne donnant pas l'impression désagréable que le bouton a été entièrement changé.
- **Le visuel suggère la transition qui va suivre** :



  Dans le cadre d'un choix multiples, la lumière est présente sur le côté ou va apparaître la réponse

Unity a depuis un moment effectué une transition majeure dans l'élaboration des interfaces utilisateur ; via le package UI Toolkit, la nouvelle interface utilise maintenant un système proche de ce qui se fait en développement web au delà de quelques nomenclatures, le système est en tout point identique au html fonctionnant via un fichier UXML d'où il est possible d'écrire manuellement l'interface via le code ; Unity laisse aussi la possibilité de créer l'interface en plaçant les éléments manuellement directement sur le canvas.

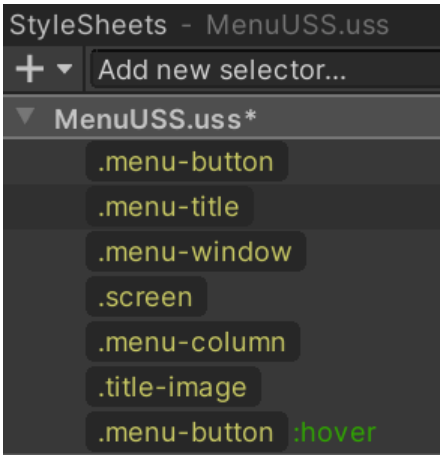
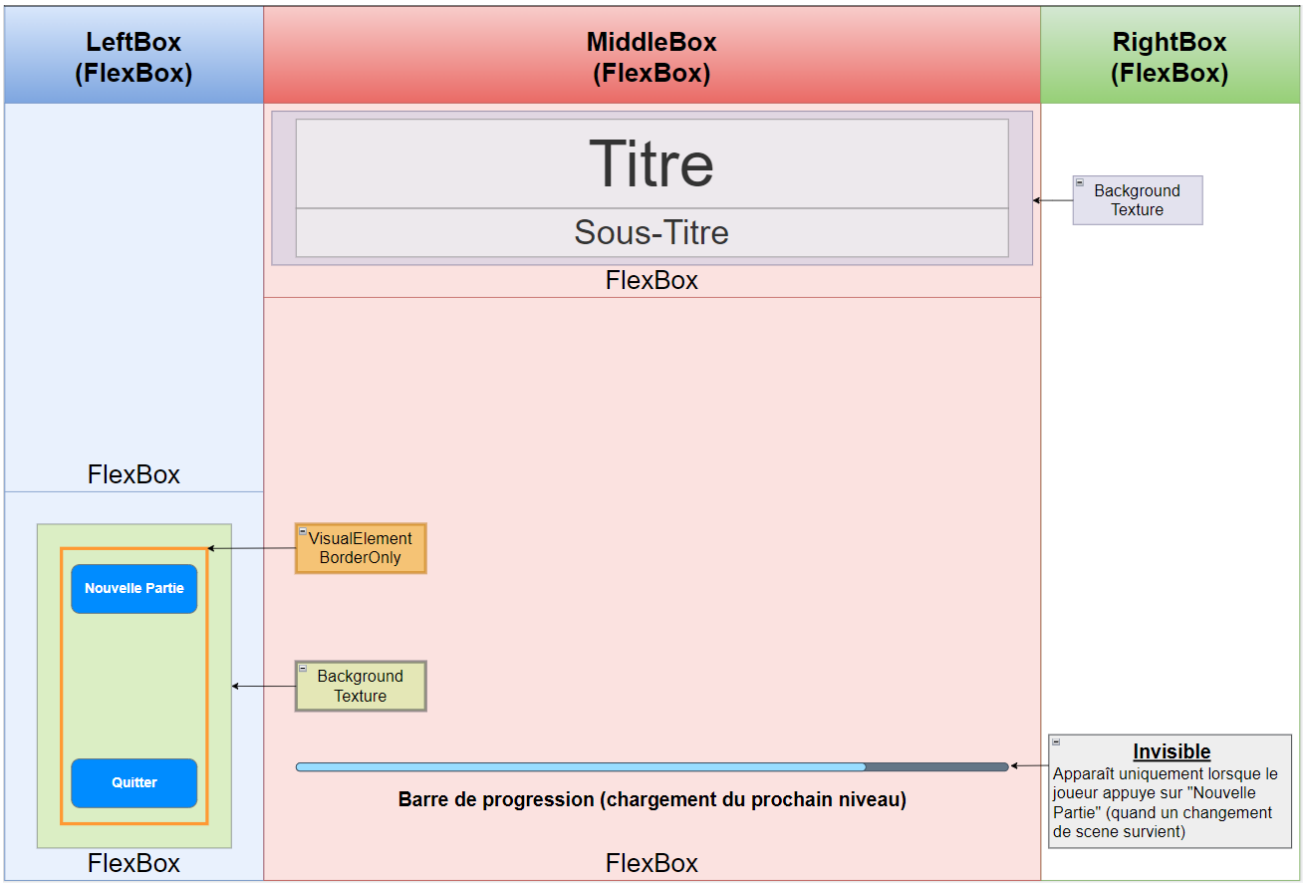


Figure 2.1 – Feuilles de Style

La Mise en forme elle est faite via un code USS(équivalent de CSS), là encore Unity propose un raccourci sur l'éditeur via une fenêtre d'inspection donnant un accès directe aux paramètre modifiable d'un élément sélectionné.

Il est cependant conseillé dans le cadre de la mise en forme de créer des feuilles de style USS qui pourront être reprise et utiliser aussi dans l'interface graphique (Voir Figure : 2.1) afin de garder une constance dans le design des éléments et éviter de recréer plusieurs fois le même objet et de pouvoir les réutiliser dans d'autres projets, en effet toute modification css via l'interface graphique crée une nouvelle propriété dans le fichier UXML.

Maquette du menu Principale



2.2.2 Régiments

La structure des entités en tant que régiment/groupe demande de mettre au clair quelque aspect avant l'élaboration de la conception :

Interactions

Reprenant le cahier des charges, il nous faudra dans le cadre des interactions, concevoir :

- La Présélection des régiments.
- La Sélection des régiments.
- Le Placement des régiments.

Chacun de ces éléments correspondra à un objet qui pourra être activé ou désactivé, mais à qui revient cette responsabilité ?

Registre : Correspond à la liste des éléments contenu dans les systèmes suivants :

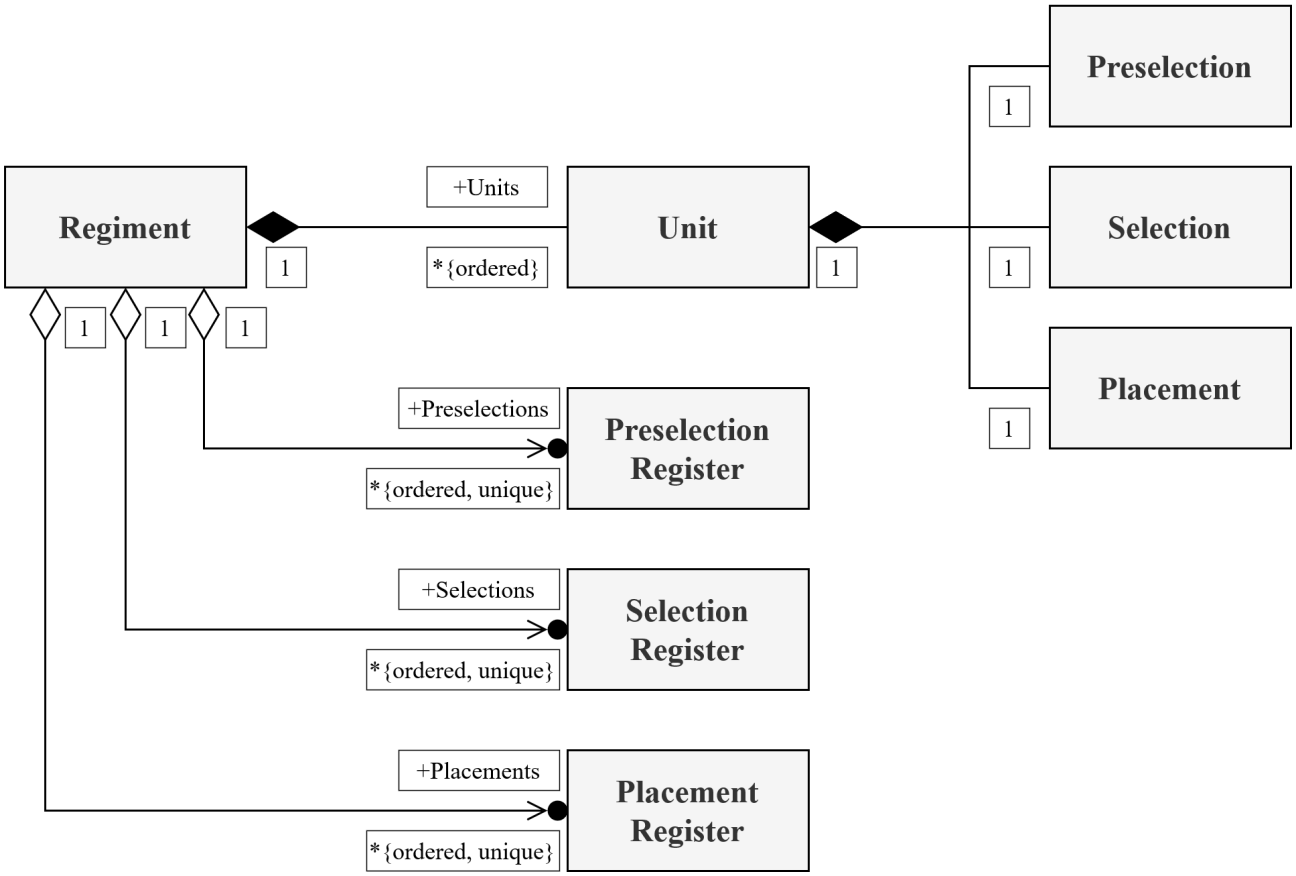
Présélection

Sélection

Placement

deux schémas peuvent être proposé chacun ayant des implications profondes dans la structure du programme :

Relation directe des intéractions avec les unités



Cette forme prévoit un lien directe entre les unités et les différents indicateur dans une relation parent-enfant ou l'unité serait le parent.

Avantages

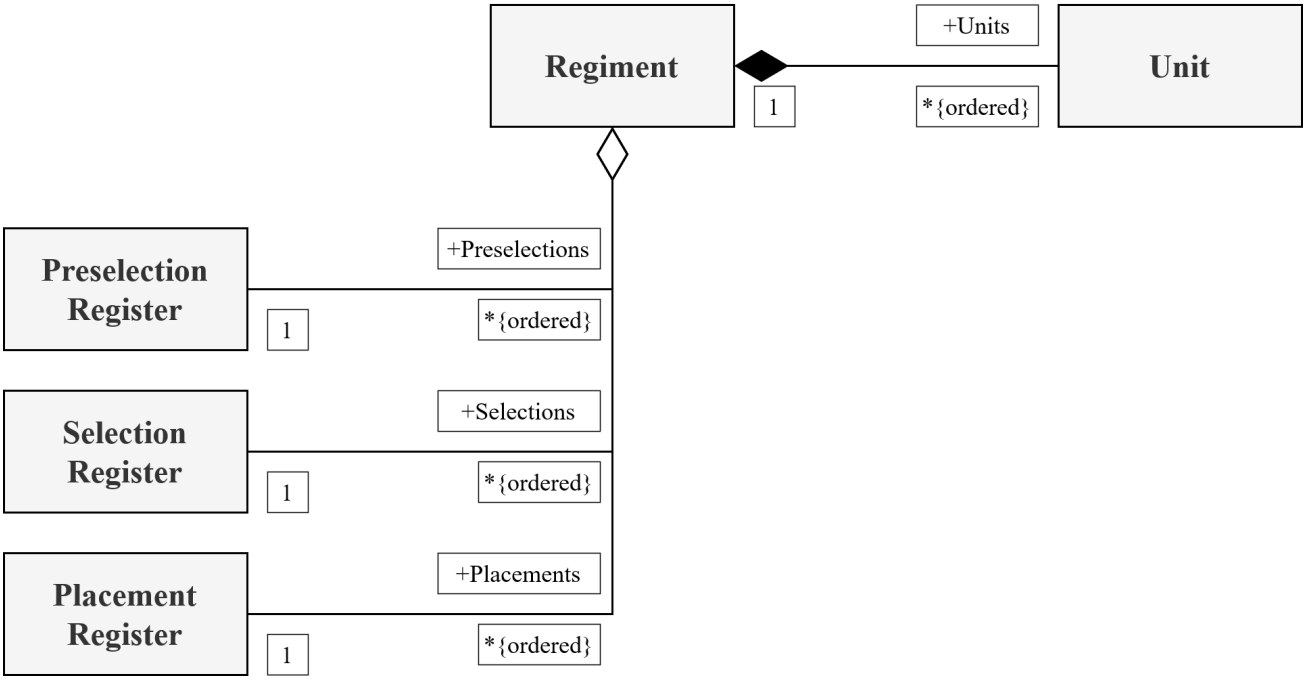
- Gestion des déplacements gérés par Unity.
- Gestion de la destruction des marqueurs simplifiés
- Besoin réduit de passer par un système complexe d'interfaces/abstract

Inconvénients

- Architecture ne respectant le le SOC[1] il y a une longue chaîne de dépendances entre le système en charge de activer/-désactiver les marqueurs ce qui rend le programme sensible aux changements (un changement dans la classe qui gère le régiment peu casser le système de séléction par exemple).
- Accès au ressources d'un système à un autre potentiellement problématique.

1. Separation of Concerns : Séparations des résponsabilités, chaque système doit avoir une unique responsabilité

Relation indépendante des intéractions avec les unités



L'autre solution est de rendre totalement indépendant chaque marqueur des unités du régiment dans le sens ou un marqueur ne serait pas lié à une référence spécifique d'une unité en jeu.

Avantages

- Gestion des registres à la destruction d'une unité, moins complexe.
- Gestion de la destruction des marqueurs simplifiés
- Séparation propre et claire des responsabilités

Inconvénients

- Gestion des mouvements du marqueurs manuels
- Implémentation d'un système d'interface/abstract complexe pour obtenir un code lisible.
- Gestion de cas particuliers liés à la gestion manuel du mouvement des marqueur.

Après consultation avec le chef de projet, la deuxième solution est préférée, pour la raison suivante :

Souhaitons-nous pouvoir sélectionner une unité individuellement (sans sélectionner tout son régiment ? Non, une autre raison est là séparation de chaque fonctionnalité en système distinct permettant de respecter l'exigence SOC.

Le partage des ressources ce fera vie un coordinateur qui référencera les trois systèmes et sera en charge de faire transiter les ressources d'un système à un autre.

Diagramme de classe : Régiments

Régiments : Présélection

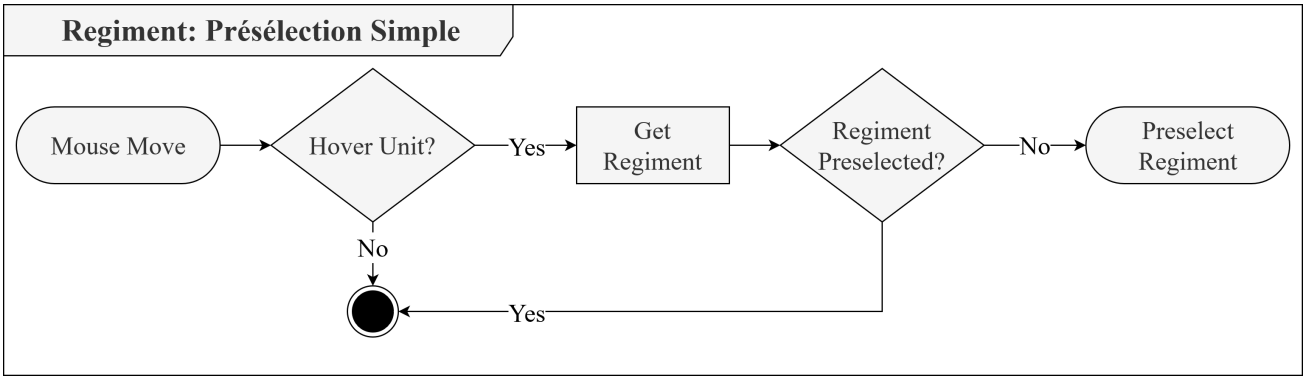
Pourquoi ? Les entités fonctionnant en groupe il est important que le joueur puisse déterminer quelles troupes seront sélectionnée avant d'engager l'action dans les cas ou plusieurs régiments seraient engagés au même endroit et leurs unités seraient alors peut être les une sur les autres.

La Présélection fonctionnera de pair avec la sélection(voir chapitre suivant : Sélection d'un régiment) en effet ce système informant à l'avance quels régiments seront sélectionnés, il lui suffira d'envoyer un message à la classe concernée pour effectuer la sélection.

Il y aura deux types de présélection distincts :

Présélection Simple

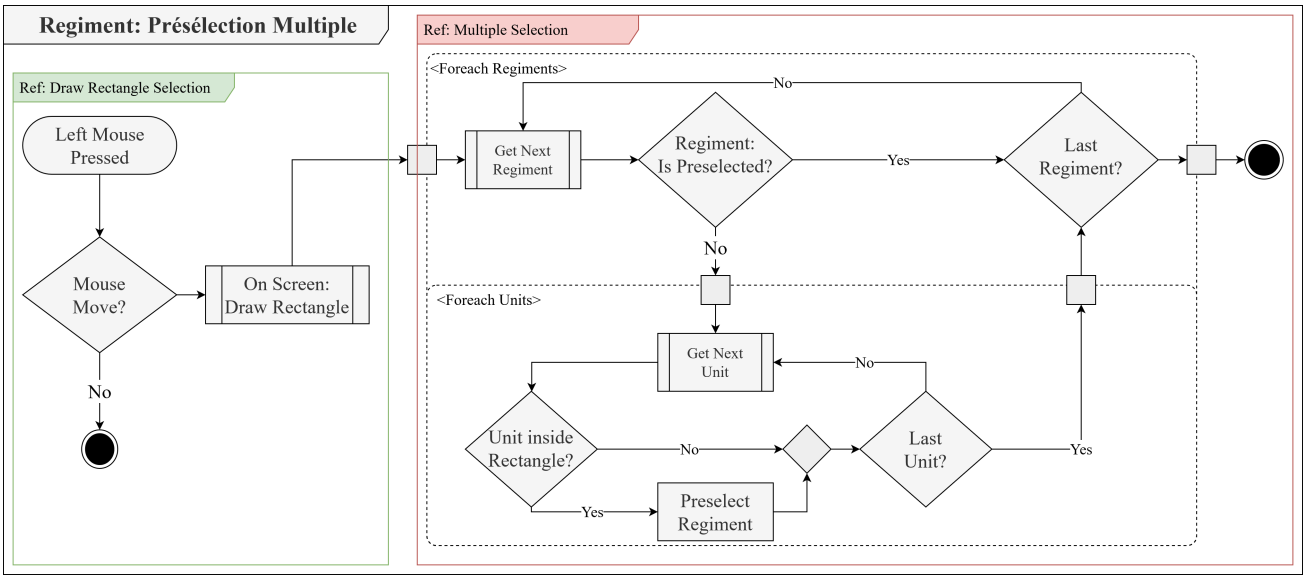
Cette dernière ce fera lorsque le curseur passe par dessus une unité, dès lors le système va récupéré le régiment auquel appartient la cible, puis va provoquer la présélection de tous ses membres.
Schématique la structure du programme donnera quelque chose de ce type :



Présélection Multiple

La Présélection devra permettre au joueur de sélectionner plusieurs entités en maintenant le clique gauche de la souris enfoncé, en bougeant la souris, un rectangle se dessinera, ainsi toutes les unités sous le rectangle verront leur régiment être sélectionner.

Il y aura donc deux séquences : le "dessin" du rectangle de sélection et la détection des régiments présélectionnés.



Régiments : Sélection

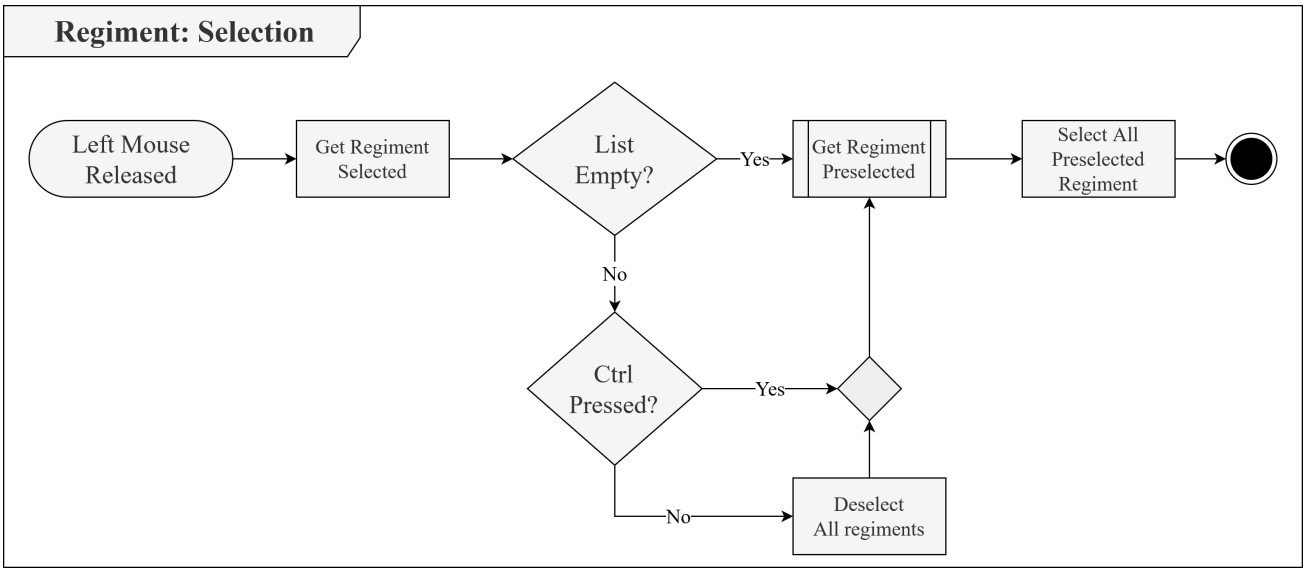
La sélection sera un système très court car reprenant bon nombre du travail qui a déjà été fait par la présélection(voir : chapitre présélection), avec cependant une fonctionnalité supplémentaire : "Ajouter uniquement" ; qu'entend-on par là ?
La fonctionnalité de base est la suivante :

"Quand je relâche le clique gauche de ma souris, désélectionne tous les régiments, et sélectionne tous les régiments préselectionnés".

Note Importante

l'Accès aux régiments préselectionnés qui est une ressource externe au système de sélection se fera via un coordinateur mentionné précédemment.

Mais le joueur voudra peut-être ajouter des régiments à la sélection actuelle sans avoir à les présélectionner à nouveau ceux déjà sélectionnés ; une mécanique connue dans les Real Time Strategy Game et de désactiver la possibilité de désélectionner sitôt que une touche est maintenue enfoncée (généralement : Ctrl Gauche du clavier).



Attention : Remarquez que la séléction survient après la préselection ; ce qui implique que la déprésélection des régiments survient avant vidant la liste des présélections avant que le système de sélection ne puisse la consulté.

Régiments : Placement

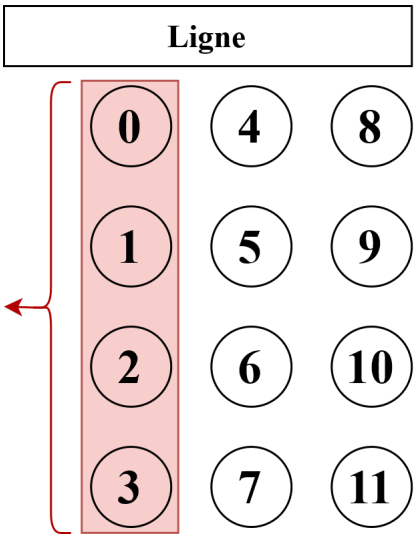
La partie la plus technique dans l'intéractions des régiment tant le nombre de cas particulier sont légions et les adaptations nécessaires pour parer à toutes éventualités augmentent la complexité de l'algorithme.

Afin de facilité la compréhension nous irons étape par étape ajoutant les complexités une à une afin de facilité l'analyse et l'implémentation, et dans le pire des cas de permettre de proposer une implémentation même imparfaite qui permette de tester les autres fonctionnalités du projet.

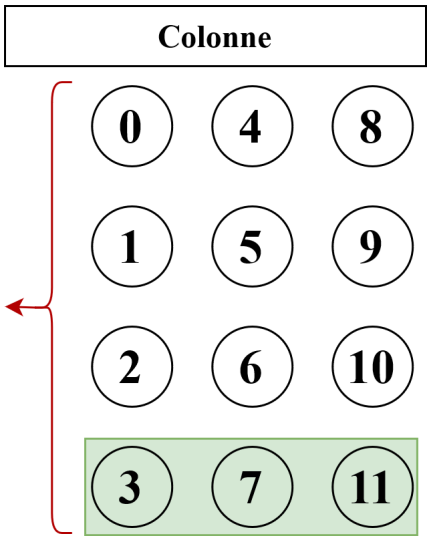
Nomenclature

- **Distance du curseur** : distance parcouru par le curseur entre le début du clique gauche et sa position actuelle :
- **Taille minimal d'un régiment(ligne ou colonne)** : cette dernière comprend le nombre d'unités autorisés multiplié par la taille d'une des unités en question en prenant en compte l'espacement entre les unités.
- **Configuration** : On parle dans le cadre du régiment des variables dynamiques suivantes : Nombre actuelle : d'unité par ligne, d'unité par colonne, d'unité sur la dernière colonne.

Les termes "Colonnes" et "Lignes" seront souvent utilisés, au vu du caractère dynamique des formations il est important de poser les définitions suivantes :



Les Lignes correspondent aux rangées perpendiculaire à la direction où regarde le régiment.

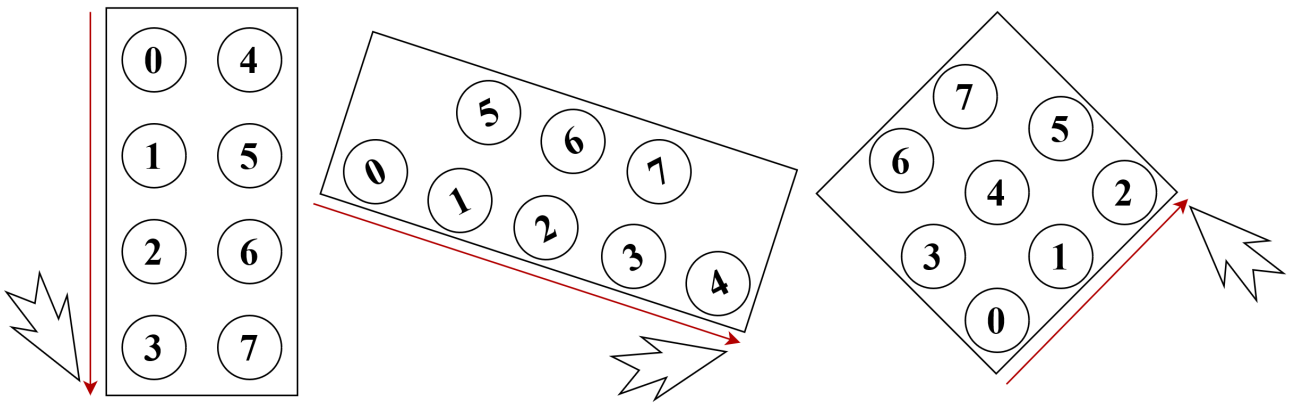


Les Colonnes correspondent aux rangées parallèles à la direction où regarde le régiment.

Implémentation de base

Pour cette partie nous allons nous concentrer sur le cas d'une seule sélection, le résultat souhaité et de positionner dynamiquement les régiments sélectionné en effectuant la séquence suivante : Bouton droit de la souris pressé + bouger la souris.
Les paramètres suivants seront à prendre en compte :

- La direction du défilement de la souris
- Le minimum d'unité par rangée qui est autorisé pour le régiment sélectionné
- Le maximum d'unité par rangée qui est autorisé pour le régiment sélectionné



- Le nombre d'unités sur la dernière rangée.

Note Importante

Le nombre de colonne et de ligne d'un régiment changeant dynamiquement, l'utilisation d'un array multidimensionnelle est proscrit !
Pour récupérer les valeurs x(Index dans la ligne) et y(N° ligne) il faudra appliquer la formule suivante :
Soit :

int i : L'indexe de l'unité dans le régiment.

int width : Nombre d'unités par ligne dans la configuration actuelle du régiment.

```
int y = i / width;
int x = i - (y * width);
```

Avant d'atteindre le point de déformation

La première phase consiste à vérifier la distance parcourue par le curseur, les régiments ayant, quelque soit leur type une taille de ligne minimal, aussi lorsque la distance parcourue par le curseur n'est pas supérieure d'au moins une taille d'unité (unité qui compose le régiment) le régiment conservera sa configuration.

Ce qui implique que lorsque la "distance du curseur" est inférieur à la taille minimal du régiment cette dernière devra tout de même apparaître, cependant, seul une rotation sera effectuée. *Note : bien que le terme rotation est utilisé, le positionnement sera recalculé pour des raisons de temps, la rotation en question étant un calcul complètement différent.*

Algorithme

Quelques variables utilisées seront

- **Position de départ(Start)** : Position du curseur au moment du clique droit (cette position sera celle de la première unité du régiment).
- **Position Actuelle(End)** : Position actuelle du curseur.
- **Distance parcourue par le curseur** : Distance(End-Start).
- **Direction du placement** : Direction normalisé du vecteur formée par End-Start.
- **Espace entre les unités** : Correspond à l'espacement fixe entre deux unités d'un régiment et qui est définie par la classe du régiment.
- **Taille d'unité** : Correspond à la taille d'une unité d'un régiment et est défini dans la classe du régiment.

Nombre d'unités par ligne

La première variable à mettre à jour est le nombre d'unité par ligne, il nous faudra pour cela calculer le nombre d'unités pouvant être placé sur la longueur tracée par le curseur.

Pour obtenir le nombre d'unités par ligne il nous faut faire le calcul suivant :

$$\textbf{Nombre d'unités par ligne} = \frac{\text{Distance parcourue par le curseur}}{\text{Taille d'unité} + \text{Espace entre les unités}} + 1$$

Pourquoi ajouter 1 ?

La première unité n'est pas comprise, dans les fait le calcul exprime plutôt le nombre de séparation entre les unités, en partant du point "start" au point "end". Que nous pouvons interpréter comme le nombre de coordonnées valides après la première unité (impliquant qu'elle n'est pas incluse).

Dans les deux cas, il nous faut ajouter la position "Start" correspondant à la première unité.

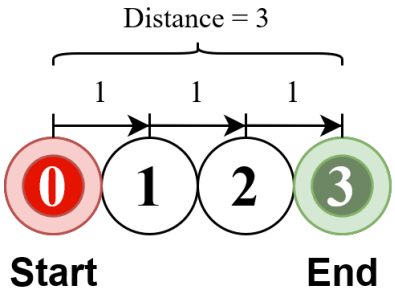
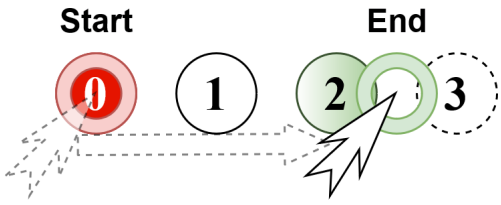
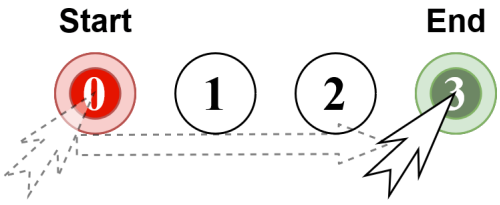


Figure 2.2 – Note : pour simplifier la compréhension, l'espacement entre les unités est de 0

Il faudra aussi arrondir à l'inférieur la valeur obtenue, Attention cependant, la longueur étant un type float, il faudra utiliser un comparateur prévu pour ce type dans notre cas, Unity fourni : Mathf.Approximate(float a, float b).



la ligne comprendra les coordonnées :
{ 0, 1, 2 }



la ligne comprendra les coordonnées :
{ 0, 1, 2, 3 }

Nombre de lignes complètes

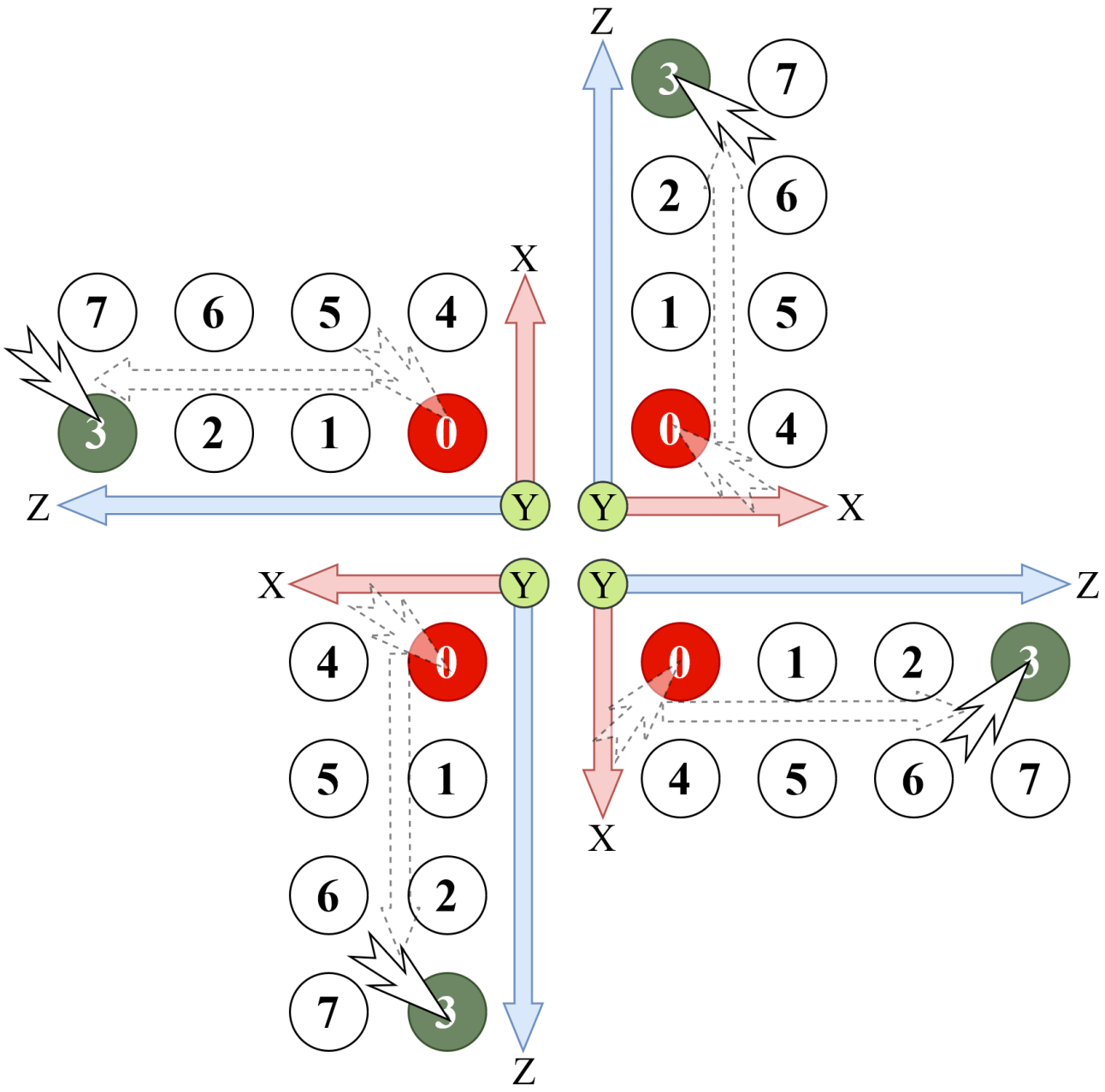
Il nous faut obtenir combien de lignes sont complètes, par cela nous entendons que le nombre de troupe est égal au nombre d'unité par ligne obtenu au calcul précédent :

$$\textbf{Nombre de lignes complètes} = \frac{\text{Nombre total d'unités}}{\text{Nombre d'unités par ligne}}$$

Note : Il est conseillé d'utiliser un type non décimal pour la formule, autrement il faudra arrondir à l'inférieur le résultat obtenu.

Direction des colonnes arrières

Dans cette partie il nous faut déterminer la direction du régiment afin de placer les lignes d'unités dans le sens souhaité ; le comportement souhaité du point de vu du joueur est le suivant :



Ce qu'il faut retenir

1) Le déplacement du curseur est représenter par l'axe Z dans le calcul.
2) Le vecteur recherché correspond vecteur perpendiculaire négatif à Z.

Il est important de noter qu'étant dans un environnement en trois dimensions, le calcul du

vecteur perpendiculaire comporte une particularité, en effet il nous faudra utiliser l'axe Y, Nous n'irons pas dans les détails, cependant cette information nous permet d'utiliser le vecteur négatif de Y où la valeur de la coordonnée Y = (0, -1, 0).

Pour le calcul nous utiliserons la formule proposer par le moteur Unity : Vector3.Cross(Vector3 lhs, Vector3 rhs) où lhs = direction du curseur, et rhs = vecteur Y ou (0, -1, 0) Le calcul prendra ainsi la forme suivante :

Direction colonne = $Vector3.Cross$(Direction du curseur, Vector3(0, -1, 0))

Important : la direction ainsi obtenue doit être normalisée pour que les calculs suivants soient correctes.

Nombre d'unité sur la dernière ligne du régiment

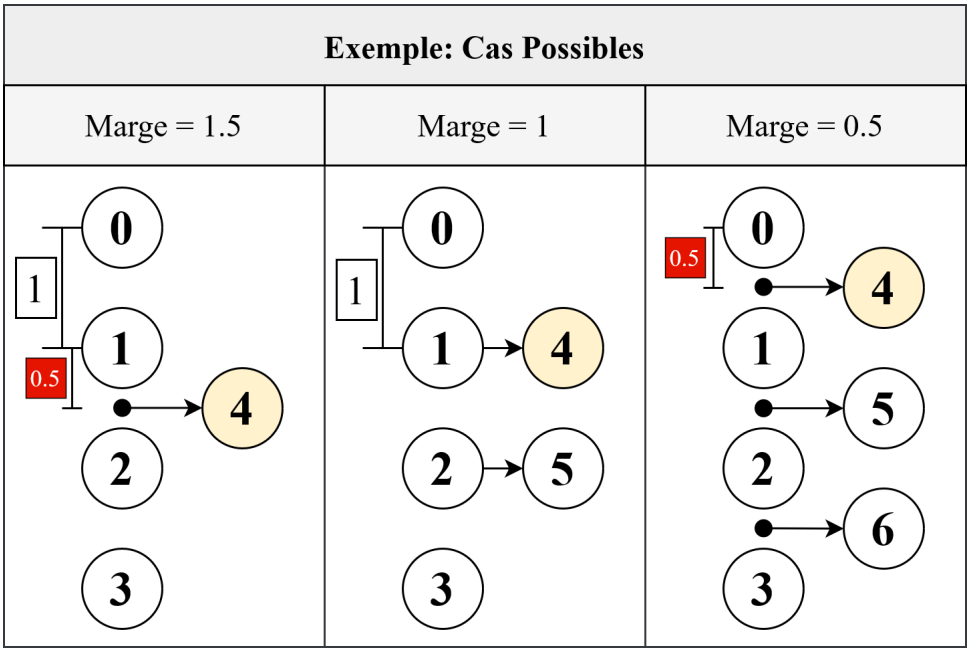
Enfin nous pouvons obtenir le nombre d'unités sur la dernière ligne du régiment par la formule suivante :

Nombre d'unités dernière ligne =
Nombre total d'unités − Nombre de lignes complètes ∗ Nombre d'unités par ligne

Note : Il est conseillé d'utiliser un type non décimal pour la formule, autrement il faudra arrondir à l'inférieur le résultat obtenu.

L'importance de la dernière ligne

La dernière rangée est assez technique, en effet cette dernière peut ne pas être pleine pouvant donner des cas ou les unités seront décalée d'une demi taille (taille d'une unité du régiment + espacement dans le régiment) afin que la dernière ligne soit bien centrée comme le montre les exemples suivant :



Les Valeurs ci-dessus sont a prendre comme une marge à ajouter avant chaque placement d'unités sur la dernière ligne(voir l'algorithme complet plus loin).

Le calcul pour obtenir cette valeur est le suivant :

$$\textbf{Marge} = \frac{\text{Nombre d'unité par ligne} - \text{Nombre d'unités dernière ligne}}{2}$$

Algorithme complet

```
1  Vector3[] GetRegimentFormation(Regiment regiment)
2  {
3      float UnitSize = regiment.UnitSize;
4      float UnitOffset = regiment.UnitsOffsetInRegiment;
5
6      float totalUnitSize = UnitSize + UnitOffset;
7
8      //Position de la souris sur le terrain au moment du clique droit
9      Vector3 StartMouse;
10     //Position actuelle de la souris sur le terrain
11     Vector3 EndMouse;
12
13     float MouseDistance = Vector3.Distance(EndMouse - StartMouse);
14
15     //Guard Clause
16     if(MouseDistance < regiment.minRowDistance) return;
17
18     Vector3[] formationPositions = new Vector3[regiment.TotalUnits];
19
20     int NumberUnitPerLine = 1 + MouseDistance/(UnitSize+UnitsOffset);
21
22     float numRows = regiment.TotalUnits / (float)NumberUnitPerLine;
23     //nombre de lignes
24     int totalNumRow = Mathf.CeilToInt(numRows);
25     //nombre de lignes complètes
26     int numCompleteRow = Mathf.FloorToInt(numRows);
27
28     int NumberUnitLastRow = regiment.TotalUnits - numCompleteRow*
       NumberUnitPerLine;
29
30
31
32     for (int i = 0; i < regiment.TotalUnits; i++)
33     {
34         // Coordonnées dans le régiment et non dans l'espace
35         int y = i / NumberUnitPerLine;
36         int x = i - (y * NumberUnitPerLine)
37
38         //Position sur la ligne (coordonnée x)
39         Vector3 linePosition;
40         Vector3 lineDirection = (EndMouse - StartMouse).normalized;
41         linePosition = StartMouse + x*(lineDirection*totalUnitSize);
42
43         //Dernière Ligne ?
44         if(y == totalNumRow && totalNumRow != numCompleteRow)
45         {
```

```
46              float offset = (NumberUnitPerLine - NumberUnitLastRow) / 2f;
47              linePosition += offset;
48          }
49
50          //Position sur la colonne (coordonnée y)
51          Vector3 columnPosition;
52          Vector3 yAxis = new Vector3(0, -1, 0); // Vector3.down
53          Vector3 columnDirection = Vector3.Cross(lineDirection, yAxis);
54          columnPosition = linePosition+y*(columnDirection*totalUnitSize);
55
56          formationPositions[i] = linePosition + columnPosition;
57      }
58  }
```

Attention : lors de l'implémentation :
0) il fallait checker si y = total de ligne - 1 ! car total de ligne ne prend pas en compte le 0.
1) Ne pas oublier de Min-Max le nombre d'unité par ligne selon le régiment
2) Externaliser la direction de la ligne et de la Colonne (pour ne pas répéter le calcul)
3) Externaliser la marge aussi.
4) Oubli de calculer la direction : Quaternion.LookRotation(-columnDirection, Vector3.up)
Possibilité d'amélioration :

Le coût de l'algorithme augmente proportionnellement au nombre d'unités contenu dans le régiment, aussi il serait intéressant d'utiliser le multithreading proposé par Units, avec la package Job System.

Placement à plusieurs régiments

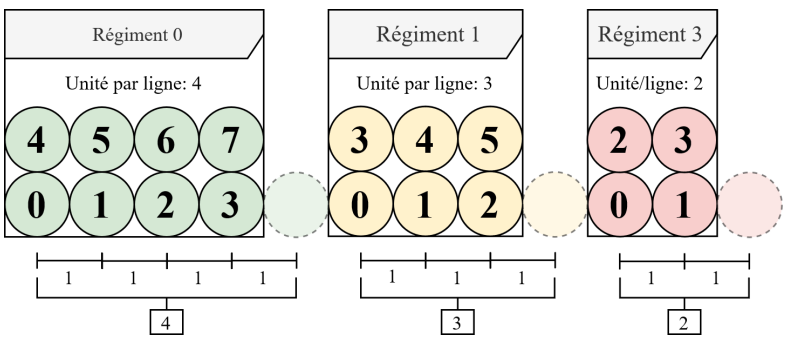
Lorsque plusieurs régiments seront sélectionnés, il faudra ajouter les éléments suivants à l'algorithme :

- Il faudra que la distance parcourue permette d'augmenter la taille de chaque régiment.
- L'emplacement des régiments devra être au plus proche de leur position actuelle.
- Il y aura une marge à ajouter pour l'unité de départ des régiments.
- La taille actuelle et après placement sera à prendre en compte.

Distance parcourue : Nombre d'unité à ajouter/réduire par ligne

---

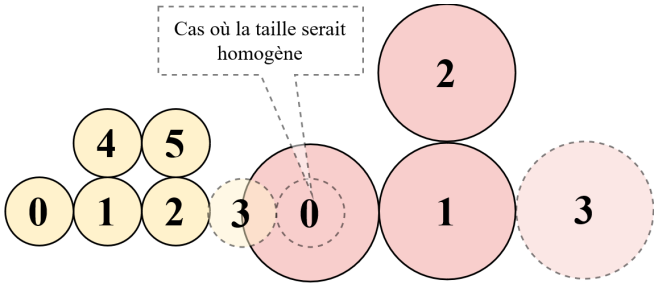
La première Adaptation consiste à redéfinir la règle qui autorise le lancement de l'algorithme, nous avons en effet plusieurs régiments sélectionné et le comportement souhaité est une "déformation" homogène, autrement dit, il faut que chaque régiment augmente ou réduit leur nombre d'unités par ligne de manière égal.



Attention : à ne pas confondre Nombre d'unités et nombre d'espace entre unités.nous allons utiliserons exclusivement l'espacement entre les unités pour les calculs suivant

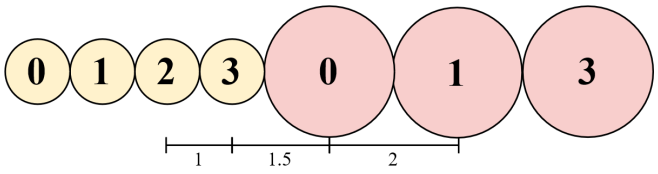
Note Concernant l'espacement entre régiments

Une question à se poser est l'espacement entre les régiments, en effet si dans le cadre du prototype, les unités ont toutes la même taille qu'arrivera-t-il si l'on devait ajouter des unités avec des tailles différentes ?

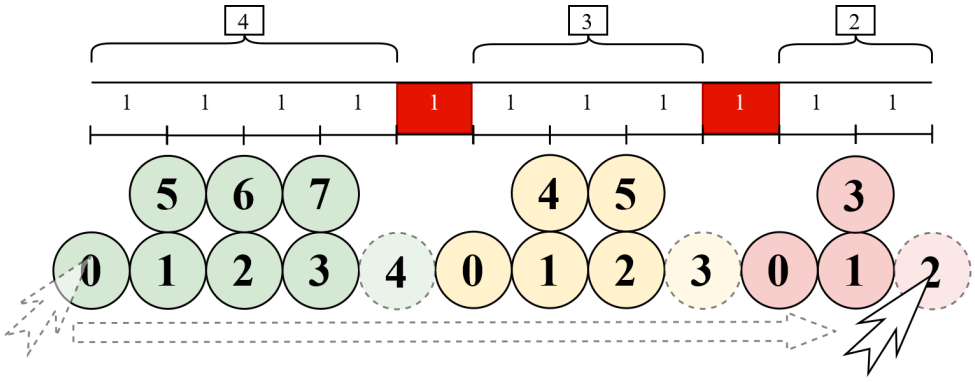


Il y aurait un risque que les unités se retrouve les une sur les autres au placement, Il faudra donc effectuer le calcul suivant :

$$\textbf{Espacement entre régiment} = \frac{\text{Espace entre Unité N} + \text{Espace entre Unité N-1}}{2}$$



Le résultat devra être ajouter à la vérification de la distance parcourue par le curseur.
Implication pour la distance parcourue par le curseur :



Important : Dans le cadre du placement multiple, il s'agit davantage de calculer le nombre d'unité à ajouter à la valeur "*Nombre d'unités par ligne*", aussi il faudra que les régiments conservent en mémoire la valeur correspondant au nombre d'unités par ligne actuelle.
le calcul est en deux étapes :

1. Calculer la distance minimale des régiments sélectionnés :

Il s'agit là de prendre la distance taille de la longueur(ou taille de ligne). Attention il s'agit de la taille allant du centre de la première unité jusqu'au centre de la dernière unité(voir partie précédente : Implémentation de base, partie "Nombre d'unités par ligne").

Il faudra ensuite ajouter l'espacement entre les régiments (voir page précédente : "Note Concernant l'espacement entre régiments"). Il faudra très certainement passer au travers d'une boucle utilisant le même ordre que pour le placement des troupes.

2. Diviser le tout par le nombre de régiments sélectionnés.

Enfin le calcul déterminant le nombre d'unités ajouté sera le suivant :

$$\textbf{Nombre d'unités ajouté*} = \frac{\text{Distance curseur} - \text{Taille minimale des régiments**}}{\text{Nombre de régiments sélectionnés}}$$

* Si le type choisi est à décimal, il faudra arrondir à l'inférieur.
**Comprend aussi l'espacement entre les régiments

Finalement il faudra a chaque itération (sur les régiment) ajouter à la position start(position de départ du cusrseur) un décalage égal à la taille de la ligne du régiment précédemment placé (+ l'espacement entre régiment calculé précédemment) qu'il faudra multiplier par la direction du vecteur formé par la position de départ et final du curseur.

Un autre aspect a prendre en compte est le réarrangement dynamique d'un régiment quand une de ses unités est tué. Les choix de la collection est de ce primordiale, il y a un comportement précis suivant un algorithme demandant un contrôle précis dans l'agencement des unités dans la collection choisie.

Il est aussi important que le réarrangement soit effectif et ne doit en aucun cas être simulé via des valeurs temporaires ou en déplaçant uniquement les unités dans la configuration voulu, car cela provoquera des problèmes quand le joueur repositionnera le régiment ; les unités n'ayant pas changé effectivement leur indexe dans le régiment, tenteront de reprendre leur position lors du prochaine ordre de repositionnement pour se conformer à la position qu'il leur est assigné dans le régiment, exposant au joueur une vision de chaos et de désordre lui indiquant un manque flagrant d'attention apporté au produit ; bien que fonctionnellement le résultat soit le même, d'autres fonctionnalitées pourrait être impactées négativement comme le combats, les unités mettant plus de temps à se reformer sur de courtes distances et donc mettant potentiellement plus de temps se mettre en position de tir.

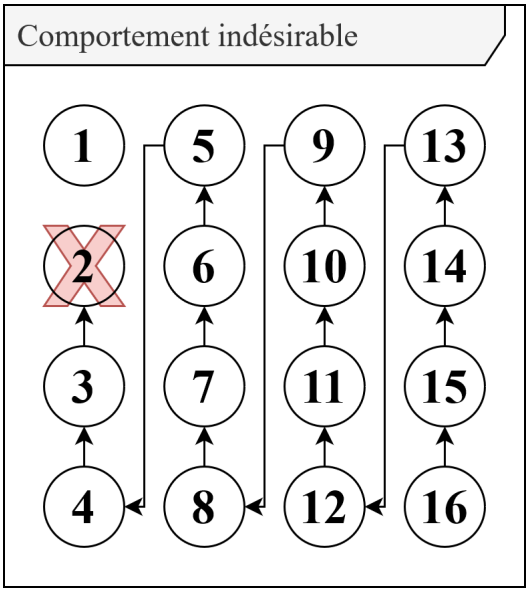


Figure 2.3 – Réarrangement automatique d'une collection type List

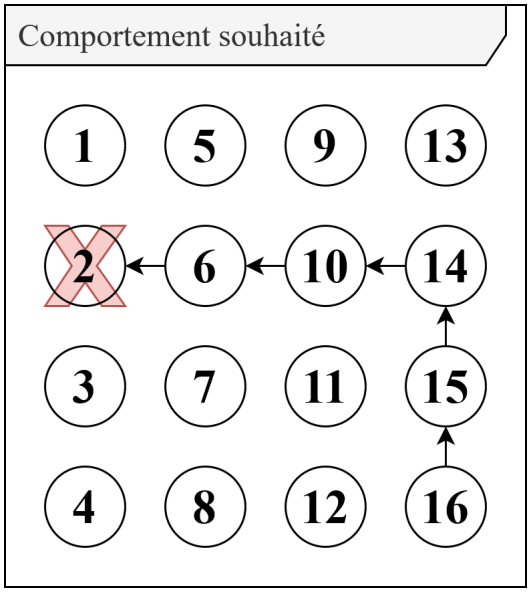


Figure 2.4 – Réarrangement suivant l'algorithme

Sur toutes les collections à disposition en C# c'est l'array qui est choisi pour les raisons suivantes :

- Flexibilité
- Manuel : Pas de réarrangement automatique
- Compatibilité avec l'API du moteur de jeu Unity
- Performance

Algorithme de réarrangement

L'algorithme utilisé devra prendre en compte les paramètres suivants :

- Largeur(Ligne) du régiment en nombre d'unité
- Longueur(Colonne) du régiment en nombre d'unité
- Index(dans le régiment) de l'unité détruite

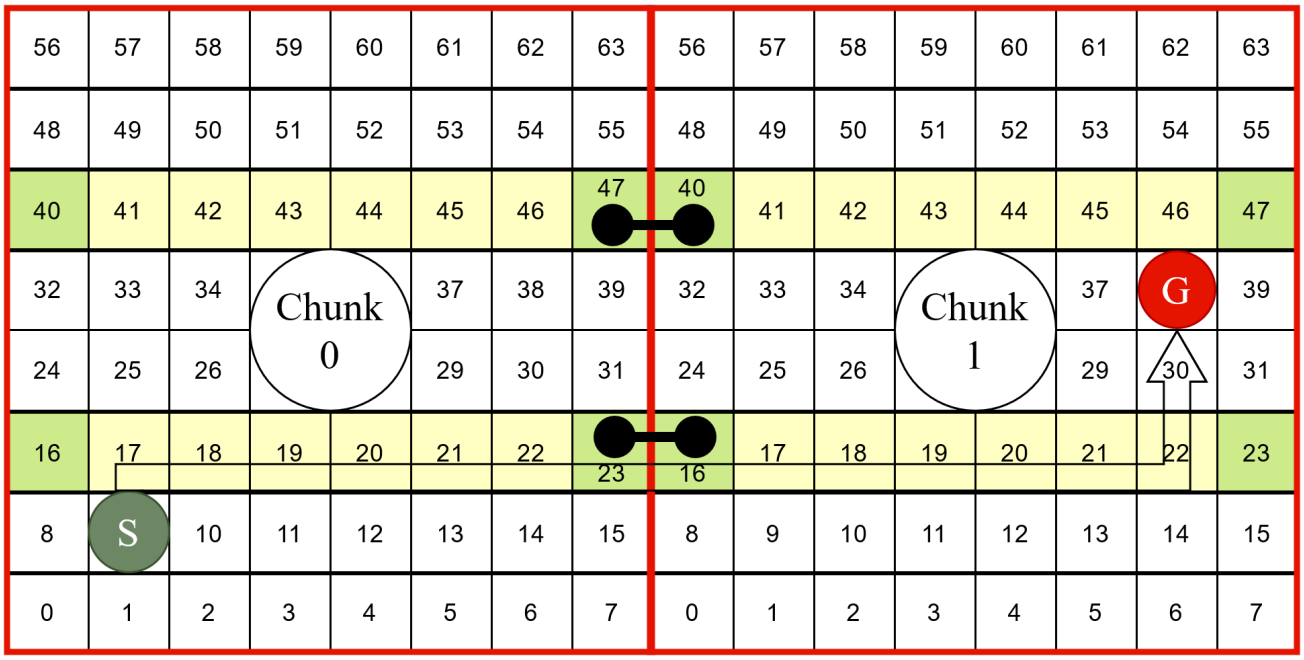
Note : Les régiments changeant dynamiquement certaines de leur propriétés comme la largeur et la longueur, les paramètres seront fixes et égal à l'instant où l'ordre de réarrangement est lancé.

2.3 Pathfinding HPA

2.3.1 Introduction

Qu'est-ce que le Hierarchical Pathfinding A* ou HPA ?

Il s'agit d'une abstraction du système de pathfinding A* et qui a pour but de réduire le temps de calcul du processeur sur de grande surface. Ce travail est fait sur la base du travail de recherche effectué par Adi Botea, Martin Müller et Jonathan Schaeffer [1]
Exemple prenons un terrain de 16x8.



Le principe est de découper la grille en chunks (ci-dessus : 8x8 en rouge), et de précalculer les chemins entre deux portes définies(ci-dessus : cellules vertes) ; ainsi lorsque une unité doit aller du point "S" au point "G", ce dernier n'aura pas à calculer les chemin dans les chunks.

Le calcul dynamique se fera sur les chunk eux-même, réduisant considérablement le temps d'itérations du calcul de l'A*.

Avantages :

- Chemins précalculés à la compilation
- Introduction d'obstacles dynamiques facilité et extrêmement performant.
- Recalcul de chemin très performant.
- Calcul de chemin impossible performant.

Désavantages :

- Mise en place d'une structure de données complexe.
- Coût en Mémoire RAM non négligeable selon la structure mise en place.
- Identification et Gestion des cas limites.
- Qualité du chemin réduit(le chemin choisi ne sera pas forcément le plus court).

Note : les implémentations du HPA sont nombreuses, chaque implémentation devant s'adapter au besoin du projet. Dans le cadre de ce projet et au besoin en performance de ce dernier, au vu du temps disponible, nous nous concentrerons sur l'approche la plus directe possible.

2.3.2 Décomposition des tâches

La complexité du HPA vient principalement de la structure de données complexe qu'elle demande en effet, l'algorithme demande de stocker un grand nombre de données pour être efficaces et si l'implémentation de cette structure n'est pas solide des problèmes apparaîtront dans le pire des cas vers la fin de l'implémentation(quand il est trop tard).

Afin de ne pas nous perdre dans la conception, tout comme la gestion du régiment, nous allons implémenter pas à pas l'algorithme ajoutant les complexité une à une.
Les découpages seront les suivants :

Pour un Cluster :

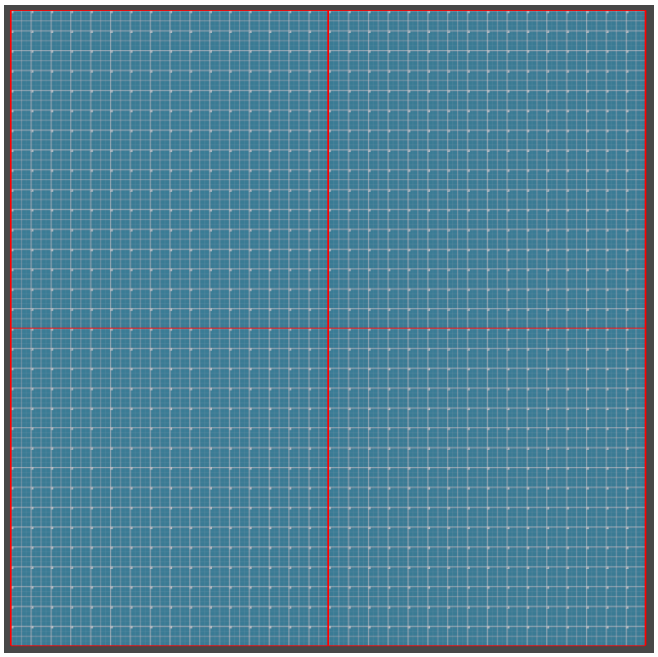
- Composition d'un cluster
- Placements des portes (servira à connecter les clusters entre eux).
- Calcul des chemins dans le cluster.
- Calcul des Distances entre les portes.

Connexion entre deux Cluster :

- Composition d'un cluster
- Placements des portes (servira à connecter les clusters entre eux).
- Calcul des chemins dans le cluster.
- Calcul des Distances entre les portes.

2.3.2.1 Cluster

Le Cluster est la représentation des données dans une partitions de la grille dont nous pouvons voir les limites en rouge sur l'image ci-dessous :

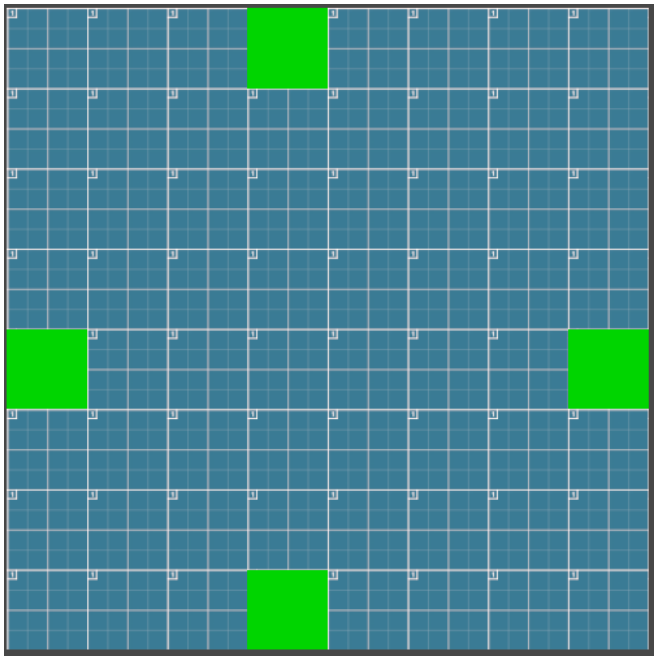


Ci-dessus : Une représentation de la grille en jeu (32x32), les lignes rouges montrent la séparation entre les partitions de taille (16x16).

Nous allons dans un premier temps établir la structure de données interne à un cluster :

Nous savons qu'un cluster doit avoir 4 portes, permettant d'établir les liens avec ses voisins, la position des portes doit normalement suivre un certains nombre de règles.

Nous n'aborderons pas ces règles ici et allons aller au plus simple et simplement les poser sur les cellules au centre des bords de la grille (arrondie à l'inférieur si il n'y a pas de chiffre rond possible.



Ci-dessus : les portes sont représentées en vert.

Il nous faut maintenant nous poser les questions suivantes :

- Y'a-t-il des situations ou il pourrait y avoir plusieurs portes par bord ?
  Oui, nous en verrons les détails plus loin, plusieurs articles[2] décrivent des situations ou plusieurs portes sont nécessaires notamment quand des obstacles sont présents.
- Doit-on pouvoir accéder à une porte en identifiant le bord auquel elle est rattachée ?
  Il est trop tôt pour le dire, cependant cela serait bénéfique dans une optique de debug, nous allons donc utiliser une collection permettant cela.

Synthèse
Suivant les besoins établi précédemment il nous faut les éléments suivant :

Un enumérateur pour définir les bords du cluster (Nord,Est,Sud,Ouest).

une liste pour chaque des côtés du cluster contenant les portes attachées à chacun de ses bords.

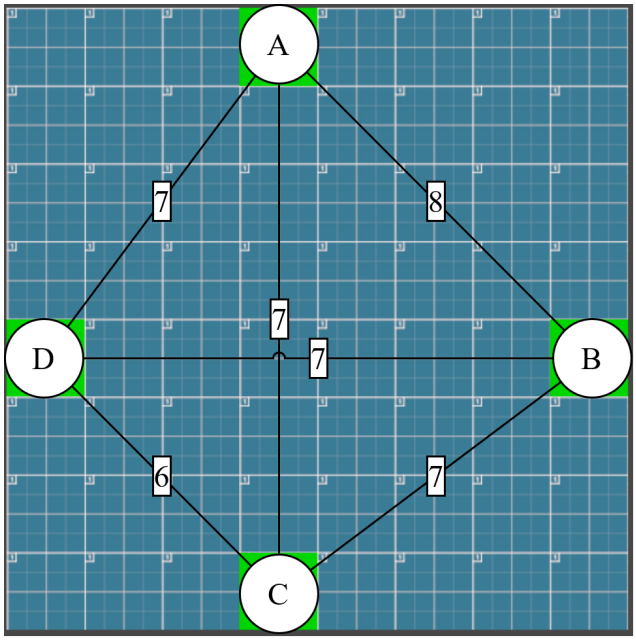
Représentation C#

```
enum Cardinals
{
 North,
 Est,
 South,
 West,
}
struct Cluster
{
Dictionary<Cardinals, List<Gate*> > Gates;
}
*Défini plus loin
```

2.3.2.2 Les Portes

Les portes sont l'équivalent des noeud dans les graphiques de l'Algorithme de Dijkstra[6], cependant contrairement aux algorithme conventionnels du même type, il nous faudra stocker un grand nombre d'informations et surtout, il faudra pouvoir les récupérer.
En se basant sur l'illustration ci-dessous nous pouvons établir les besoins suivants :

- L'indexe de la cellule
- Les chemins possibles vers les autres portes(du même cluster).
- La distance entre les différents chemins.



Tout d'abord comment fonctionne le pathfinding A* ?

Variables importantes

H : Distance entre la Destination et la position actuelle
G : Distance entre le Début et la position actuelle
F : Coût Total (H + G)

2.4 Stratégie de test

Décrire la stratégie globale de test :

- types de des tests et ordre dans lequel ils seront effectués.
- les moyens à mettre en œuvre.
- couverture des tests (tests exhaustifs ou non, si non, pourquoi ?).
- données de test à prévoir (données réelles ?).
- les testeurs extérieurs éventuels.

Tests d'Acceptations : Menu Principal

Ordre	Contexte	Condition	Resultat
1	J'ouvre le Jeu		Le Menu et et ses composantes apparaissent
2	J'appuye sur le bouton "Nouvelle Partie"		La barre de chargement apparaît
3	La barre de chargement est complète		la scène de jeu est chargée
4	J'appuye sur "Quitter"		Le jeu se ferme

Tests d'Acceptations : Présélection Régiment

Ordre	Contexte	Condition(s)	Resultat
1	Le curseur passe au dessus d'une unité	unité non présélectionnée	Préselectionne tout son régiment
2	Le curseur n'est plus au dessus d'une unité	unité préselectionnée	Dépéselectionne l'unité
3	Je maintiens le clique droit de la souris enfoncé + je déplace ma souris		Dessine un carré à l'écran
4	Je maintiens le clique droit de la souris enfoncé + Je déplace ma souris	Passe au dessus d'une unité	Préselectionne tout son régiment
5	Je maintiens le clique droit de la souris enfoncé + Je déplace ma souris	Passe au dessus de plusieurs unités ayant appartenant à des régiments différents	Préselectionne le régiment de chaque unité

Tests d'Acceptations : Sélection Régiment

Ordre	Contexte	Condition(s)	Resultat
1	Je relâche le clique gauche de la souris	Un ou plusieurs régiments sont séléctionnées + Touche Ctrl non enfoncé	Désélectionne tous les régiments précédemment sélectionnés
2	Je relâche le clique gauche de la souris	Un ou plusieurs régiments sont séléctionnées + Touche Ctrl enfoncé	Sélectionne + dépréselectionne tous les régiments

2.5 Risques techniques

Algorithme de Pathfinding HPA

Bien que fonctionnant de la même manière que le A*, le node graph utilisé par le HPA est différent dans sa modélisation(voir le chapitre : **??**)

Réduction du risque :

- Commencer par une implémentation uniquement A* afin d'avoir une fondation solide.
- Il faudra très probablement implémenter un débugger visuel permettant de marquer le terrain, permettant ainsi de traquer visuellement ce que fait l'algorithme.

Mouvement de groupe des entités

Il s'agit là d'un secret bien gardé, en effet si beaucoup se sont penché sur les mouvement de groupes d'entités, le maintien d'une formation semblent être une implémentions marginale, les articles traitant de ce sujet sont rares[5] (car peut-être trop spécifique jeux-vidéos ?).

Réduction du risque :

Fixer les objectifs à atteindre de l'algorithme afin d'en déterminer les cas limites et les exécuter un par un afin de ne faire face qu'à un bug à la fois.

Temps

Les sujets traîtés sont complexes et sont surtout mis dans le contexte d'un projet demandant, de les faire interagir avec des systèmes extérieurs, bien que les ayant étudier dans des formes plus simples, cela c'est toujours limités à des projets d'expérimentation/d'étude. Il est clair que le projet ne sera pas un produit fini mais un prototype qui pourra lui déboucher sur un produit fini.

Réduction du risque :

Implémenter dans un premier temps chaque système dans des projets séparés, puis les assembler un à un afin de cibler les conflits entre les systèmes.

2.6 Dossier de conception

Fournir tous les document de conception :

- Le choix des outils logiciels pour la réalisation et l'utilisation
- Réaliser les maquettes avec un logiciel.
- Organigramme.
- Architecture du programme.
- Pseudo-code / structogramme.

CHAPITRE 3

RÉALISATION

3.1 Dossier de réalisation

Cette partie comprendra le déroulement du projet, la façon dont les implémentations ont été réalisé, leur fonctionnement, les difficultés rencontrés et la résolution de ces difficultés. Devront aussi apparaître :

1. Les compromis fait.
2. Les changements par rapport aux plans initiaux et pourquoi.
3. Présenter les alternative à une implémentation.
4. Motiver le choix d'une alternative par rapport à une autre.

3.2 Description des tests effectués

Pour chaque partie testée de votre projet, il faut décrire :

1. les conditions exactes de chaque test.
2. les preuves de test (papier ou fichier).
3. tests sans preuve : fournir au moins une description .

3.3 Erreurs restantes

S'il reste encore des erreurs :

1. Description détaillée.
2. Conséquences sur l'utilisation du produit.
3. Actions envisagées ou possibles.

CHAPITRE 4

CONCLUSION

4.1 Objectifs atteints / non-atteints

Atteints :

- Objectif1.
- Objectif2.
- Objectif3.
- Objectif4.

Non-Atteints :

- Objectif5.
- Objectif6.
- Objectif7.
- Objectif8.

4.2 Points positifs / négatifs

Positifs :

- Positif1.
- Positif2.
- Positif3.
- Positif4.

Négatifs :

- Negatif5.
- Negatif6.
- Negatif7.
- Negatif8.

4.3 Difficultés particulières

4.4 Suites possibles pour le projet

Évolutions & Améliorations

4.5 Bilan Personnel

I Will reference someone[3]

ANNEXE A

APPENDIX

A.1 Journal de Travail

Framework test frame API test api

BIBLIOGRAPHIE

[1] Jonathan Schaeffer Adi Botea, Martin Müller. Near optimal hierarchical path-finding, 2004. Last accessed 17.05.2022, https://webdocs.cs.ualberta.ca/ mmueller/ps/hpastar.pdf.

[2] Narendra S Chaudhari Amandeep Singh Sidhu. Hierarchical pathfinding and ai-based learning approach in strategy game design, 2008. Last accessed 19.05.2022, https://www.hindawi.com/journals/ijcgt/2008/873913/.

[3] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[4] Microsoft. C# coding conventions, 2021. Last accessed 13.04.2022, https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions.

[5] Dave Pottinger. Implementing coordinated movement, 2021. Last accessed 05.05.2022, https://www.gamasutra.com/view/feature/3314/coordinated_unit_movement.php ?print=1.

[6] Wikipedia. Algorithme de dijkstra, 2022. Last accessed 19.05.2022, https://fr.wikipedia.org/-wiki/Algorithme_de_Dijkstra.