UQAC

Université du Québec à Chicoutimi

Stranded Away
Rapport de développement

Auteurs :
Simon Lauzeral (LAUS26089703)
Pierrick Barbarroux (BARP19069709)
Adrien Nouvellon (NOUA13129601)
Thomas Gubeno (GUBT19019806)

Conception / Développement de jeux vidéos
8INF960 Session Automne

15 mars 2020

Table des matières

Chapitre 1

Concept et mécaniques de jeu

1.1 Concept du jeu

Un samedi soir de juin 2074, un jeune ingénieur du nom de Sam sort avec son vaisseau spatial personnel pour aller retrouver des amis dans un bar situé sur une planète éloignée. Au cours de la soirée, la bande d'amis, en recherche de sensations fortes et alcoolisés, décident d'effectuer une virée dans dans un champ d'astéroïdes ... Le lendemain, en rentrant chez lui, le jeune homme se rend compte de son erreur de la veille, et constate avec effroi la dégradation du module d'hyperespace de son vaisseau. Il entame alors un périple seul au fin fond de l'espace, dans l'espoir de trouver un endroit dans lequel il pourrait réparer son vaisseau.

Le jeu prend place sur une autre planète sur laquelle vous êtes contraint de poser votre vaisseau, dans une époque futuriste. Vous devez réparer votre vaisseau en récoltant les bonnes ressources au cours de votre aventure. Vous y rencontrerez des autochtones, vivant dans une époque technologique très reculée qui se méfieront de vous. Vous devrez alors établir votre campement sur la planète et vous débrouiller seul pour collecter les ressources dont vous avez besoin.
Grâce à vos connaissances plus évoluées, vous pourrez vous servir d'eux, en leur volant des ressources ou bien choisir de les aider, pour arriver à vos fins mais attention, chaque action a des conséquences.

1.2 Mécaniques principales

Les mécaniques de base de Stranded Away sont similaires à celles que l'on retrouve dans la vaste majorité des jeux de tout-type, à savoir :

— Le déplacement du personnage-joueur à l'aide des touches ZQSD.
— Le combat avec les différents ennemis, au corps à corps ou bien à distance avec des projectiles.
— Un système d'inventaire permettant au joueur de collecter des objets.
— Une interaction avec l'environnement du joueur.

En plus de ces mécaniques "classiques", nous avons également implémenté un système de karma, valeur numérique variant selon les actions du joueur et impactant certains évenements de l'histoire ou bien directement l'environnement (avec un karma négatif, le taux d'apparition d'ennemis est augmenté).

Un cycle jour/nuit est également présent dans le jeu, avec une heure variant tout au long de l'aventure (cf. **6.2.2 Informations du monde**). Certains évenements sont disponibles à des périodes de la journée (ou de la nuit spécifique), et le temps peut être accéléré en utilisant le lit présent à l'intérieur du vaisseau.

Finalement, les dernières mécaniques implémentées sont les mécaniques des différents ennemis, qui possèdent tous une intelligence artificielle spécifique. Ces intelligences sont définies dans la partie suivante, **3 Comportement des Intelligences Artificielles**

Chapitre 2

Comportements des Intelligences Artificielles

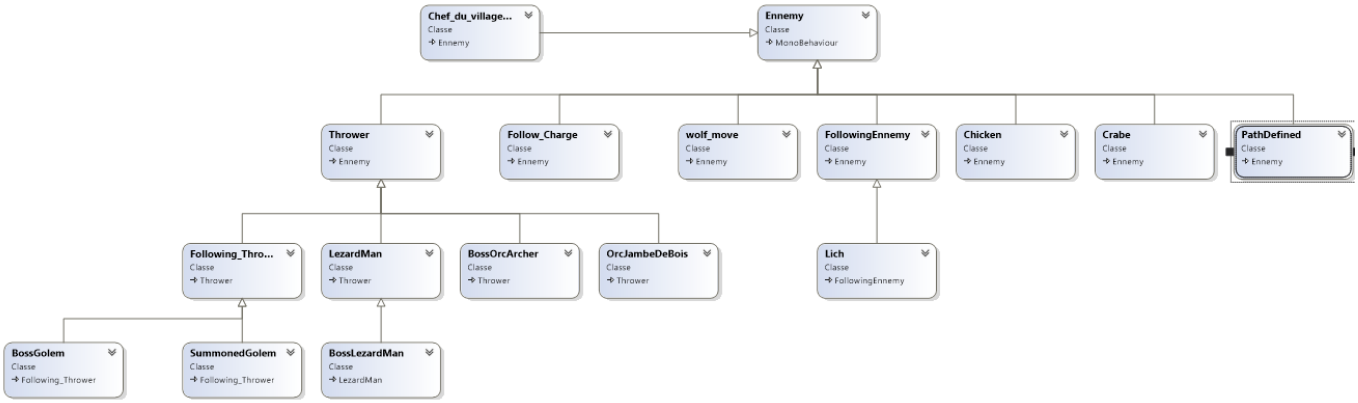

FIGURE 2.1 – Hiérarchie des classes "Ennemis"

2.1 Classe Mère

Stranded Away est un jeu d'aventure et comporte donc de nombreux ennemis dans chacune de ses zones. En terme d'IA, ces ennemis se répartissent en cinq catégories (classes c#). Chacune de ces classes vient compléter (hérite) une classe mère correspondant aux caractéristiques communes à tous les ennemis. Cette classe mère comprend les éléments suivants :

— Points de vie
— Dégâts de collision avec le joueur
— Gestion de la mort
— Gestion de la prise de dégâts (frames d'invulnérabilité et recul)
— Gestion des animations de déplacement et de mort
— Vitesse de déplacement

2.2 Monstres classiques

À cela va s'ajouter les comportements spécifiques de chacune des IA. Les cinq catégories évoquées précédemment sont les suivantes :

— **Suiveurs** : Ce sont les ennemis classiques, ils vont se contenter de suivre le joueur pour lui infliger des dégâts de collision lorsqu'ils rentrent en contact avec lui.
— **Lanceurs** : Ce sont les ennemis qui vont lancer des projectiles, ils possèdent deux zones, la première est la zone de détection du joueur, c'est à dire la distance à laquelle ils vont passer en mode "attaque" et se rapprocher du joueur jusqu'à atteindre leur deuxième zone qui correspond à leur distance maximale

de tire. Une fois la distance maximale de tire atteinte, le lanceur va s'arrêter et commencer à tirer sur le joueur. En plus de cela, les projectiles eux même peuvent posséder des propriétées comme ralentir le joueur.
- **Suiveurs + Lanceurs** : Ces ennemis combinent les deux comportement précédent, c'est à dire que même lorsqu'ils arrivent dans leur zone de lancer, il s'arrete pour lancer un projectile mais continue de poursuivre le joueur pour lui infliger des dégats de collisions.
- **Chargeurs** : Ces ennemis vont se rapprocher du joueur puis, une fois arrivé à une certaine distance, le charger à une vitesse bien supérieur à leur vitesse normal mais seulement en ligne droite, ce qui les rend quand même esquivable.
- **Crabes** : Un seul ennemi hérite de ce comportement (le crabe), c'est un ennemi qui a pour but de bloquer des passages étroits (de type couloir), il va faire des aller-retours de gauche à droite où de haut en bas.

2.3 Boss

La boule de feu, il envoie un projectile infligeant des dégâts au joueur

L'explosion d'éclair, il crée une explosion sous les pieds du joueur, ce qui lui inflige des dégâts.
] Les 5 catégories précédentes caractérisent les ennemis classiques que l'on peut trouver dans une zone, mais chaque zone comporte aussi son propre boss qui reprend un comportement classique en lui ajoutant une ou des particularités, en plus d'avoir de la vie et des dégâts accrus. Les 5 boss sont les suivants :

- **Orc Archer** : Il reprend le comportement d'un lanceur classique mais possède une compétence en plus qui est d'invoquer deux orcs (lanceurs) classiques à des endroits prédéfini de son arène (4 points d'apparitions).
- **Yeti** : il reprend le comportement d'un lanceur + suiveur avec une capacité supplémentaire qui est de faire tomber un rocher directement sur le joueur.
- **Golem** : il reprend le comportement d'un lanceur+suiveur avec comme particularité de se diviser en deux golems classiques lorsqu'on le tue.
- **Homme Lézard** : il reprend le comportement d'un lanceur+suiveur avec la capacité d'invoquer des tortues à intervalle de temps régulier et aussi de pousser le joueur vers les bords de son arène qui sont composés de pics qui le blessent s'il les touche.
- **Chef du village** : Ce boss possède un comportement unique, il ne se déplace jamais mais possède trois sorts qu'il va envoyer de manière aléatoire à interval de temps régulier. Ses trois sorts sont les suivants :
  - La téléportation, il possède 4 emplacements différents sur lesquels il peut se téléporter
  - La boule de feu, il envoie un projectile infligeant des dégâts au joueur
  - L'explosion d'éclair, il crée une explosion sous les pieds du joueur, ce qui lui inflige des dégâts.

L'ensemble des classes des IA a été codé entièrement par notre équipe car nous voulions expérimenter nous même la conception d'IA sans outils tierce tel que des behaviour tree (qui présente certain avantage) afin de se constituer une première expérience dans ce domaine.
Sur le plan purement technique, les différents comportements sont codés dans la fonction update de chacune des classes (ou une fonctions tierce qui est appelé dans le update par soucis de lisibilité du code) et utilise des coroutines pour tout ce qui dépend de timers.

Chapitre 3

Assurance Qualité

Au cours de ce projet nous n'avons pas mis en place de système de tests à proprement parlé si ce n'est que lorsque l'un de nous avait finis une fonctionnalité, au moins un autre membre de l'équipe s'assurait de son bon fonctionnement en la testant lui même de son côté pour ne pas être influencé par les habitudes de tests de celui qui l'avait développé.

Chapitre 4

Planification du projet

Pour planifier notre projet, nous avons utilisé l'outil Trello. Cela nous a permis de bien répartir les tâches entre nous. Chaque colonne du trello correspondait à un sprint. Dans un premier temps, nous avons fait la liste des fonctionnalités que nous devions implémenter. Puis nous avons divisé le temps qui nous était donné, à savoir deux mois, pour réaliser ce projet. Certains sprints duraient une seule semaine, d'autre deux. Il y avait en tout 5 sprints.

Puis nous avons réparti les tâches précédemment créées dans chaque sprint. D'abord les tâches primordiales et basiques : création et déplacement du personnage, création d'un ennemi, création des premières zones, créations des objets ... En effet, nous avons commencé par créer les objets qui seraient récurrents dans la suite du projet.

Nous avons ensuite profité d'un sprint d'une semaine pour créer une grosse partie de l'interface pour que le jeu prenne forme. D'autres tâches venaient se glisser entre plusieurs sprints. Certaines nécessitaient que d'autres éléments soient réalisés d'abord. Parfois, il s'agissait simplement de tâches qu'il restait une fois que toutes les autres avaient été distribuées. Nous les avons donc réparties équitablement entre les différents sprints tout en faisant attention à ce qu'il soit possible de les réaliser avec les composants déjà terminés.

Vinrent ensuite les IAs des ennemis et le système de combat (knockback, hitstun,...). Nous avions beaucoup d'ennemis à créer et nous leur avons donné divers comportements (suiveurs, lanceurs, casteurs, ...).

Le dernier sprint était principalement constitué de l'ajout des dialogues, des tutoriels de jeu et de certaines cinématiques. Après ce sprint, il nous restait du temps pour peaufiner certaines choses et supprimer une partie des bugs.

Nous avions parfois du retard sur certains sprints mais celui-ci était assez rapidement rattrapé et n'entachait pas la suite du déroulement du projet.

Chapitre 5

Choix de conception

5.1 Elements intéractifs

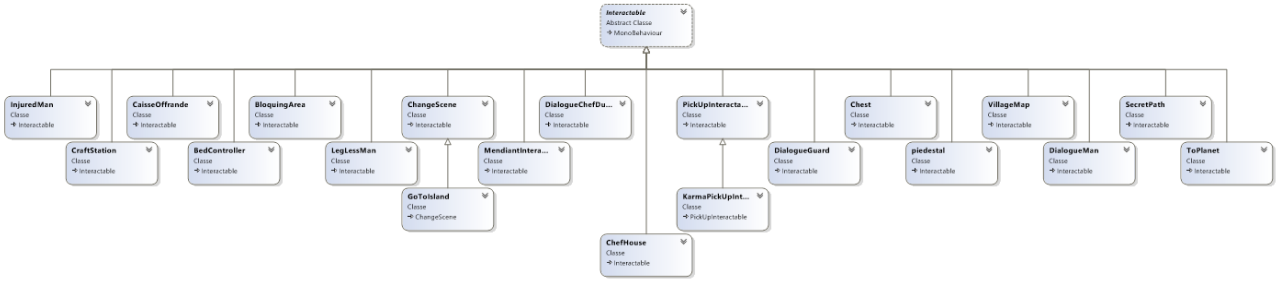

FIGURE 5.1 – Hiérarchie des éléments interactifs

Les éléments interactifs sont l'un des éléments les plus importants de **Stranded Away** car chaque ressource, chaque pnj chaque changement de zone en est un, il était donc important d'avoir un système fiable qui nous permet de réduire la duplication de code au maximum.
C'est pour ca que toutes les classes héritent d'une même classe "Interactable" qui est une interface.

Cette interface implémente les fonctions "OnTriggerEnter" "OnTriggerExit" et "OnTriggerStay" fournis par la classe MonoBehaviour de unity. Les deux premières fonctions servent à afficher sur l'écran du joueur un sprite lui disant sur quel touche il faut appuyer pour interagir avec un élément du jeu, tout en lui indiquant qu'il peut interagir avec cet élément. La troisième fonction sert à gérer l'événement lorsque le joueur appuie sur la dite touche ; lorsque le joueur appuie dessus on va appeler une methode (virtuelle) "void Interact()" qui devra être implémentée par les classes filles pour faire ce que l'on veut.

Par exemple, la classe "PickUpInteractable" sert à ramasser une ressource ou un objet sur le sol, donc dans sa fonction interact on va tout d'abord ajouter l'objet voulu dans l'inventaire du joueur (si il a la place) puis désactiver le gameobject afin que le joueur ne puisse pas le prendre deux fois.

5.2 Objets

Etant un RPG, notre jeu est obligé de comprendre de nombreux objets, ici encore nous avons une classe-mère qui va donner son comportement aux classes filles qui n'auront pour objectif que de spécifier une action précise. Ainsi,la classe "Item" comprend seulement deux attributs (le nom et l'icône de l'objet) et une méthode virtuelle "Use()". Cette méthode sera appelée via des éléments d'interface à savoir des boutons ; cela implique que le développement des objets a été pensé par rapport à l'inventaire que nous proposons dans le jeu qui est géré avec des boutons.
Comme décrit sur le diagramme, il existe quatre types d'objets :
— Les objets basiques qui n'ont aucune action directe.
— Les consommables qui vont servir à se redonner de la vie et enlevant l'objet de l'inventaire
— Les armes de melée et les armes à distance qui vont pouvoir être équipées par le joueur.

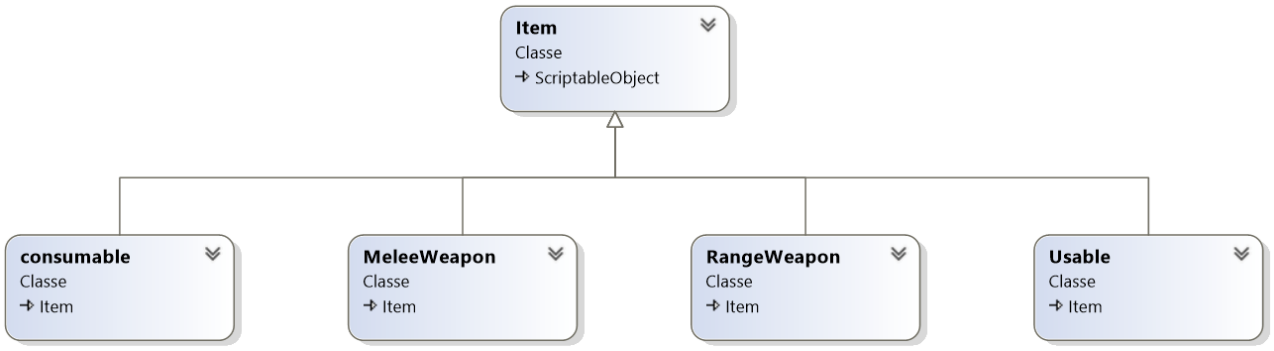

FIGURE 5.2 – Hiérarchie des classes "Objet"

Ce système d'objet nous permet d'être très flexible, il est très facile d'en ajouter un ou même de créer de nouvelles catégories si les besoins futurs du jeu le demande.

5.3 Inventaire

L'inventaire est géré par trois classes différentes : "Inventory", "InventoryUI" et "InventorySlot", cette division permet de rendre l'affichage de l'inventaire plus flexible et adaptatif aux changements. Si l'on voulait changer l'UI on aurait juste à modifier la classe InventoryUI plutôt que de revoir tout le système, ou bien si l'on veut changer les informations affichées dans une case de l'inventaire, seule la modification de InventorySlot est requise. De plus la classe InventorySlot est aussi utilisée pour gérer l'inventaire du coffre, ainsi on gardera une logique entre les différentes interfaces d'inventaire sans avoir à modifier plusieurs classes.

5.4 Ennemis

Cette catégorie représente la dernière structure concernant les choix de conception. Elle a déjà été détaillée dans la partie sur l'intelligence Artificielle.

Chapitre 6

Interface

6.1 Menus de jeu

Nous avons tenu à organiser les différents menus d'une manière cohérente et pratique. Au total, le jeu est composé de 4 menus :

— Le menu principal, que le joueur découvre au lancement du jeu. Ce menu permet de commencer une nouvelle partie, charger une partie déjà commencée et quitter le jeu.

— Le menu de pause, qui s'ouvre lorsque le joueur appuie sur la touche "Echap" du clavier au cours de la partie. Ce menu permet de sauvegarder la partie actuelle, reprendre la dernière sauvegarde effectuée par le joueur et de revenir au menu principal.
Pendant que ce menu est ouvert, le jeu est en pause, et le joueur ne peut plus interagir avec son environnement (et vice-versa)

FIGURE 6.1 – Menu Pause

— Le menu d'inventaire, qui s'ouvre lorsque le joueur appuie sur la touche "I" du clavier. Il permet au joueur de réorganiser son inventaire, y compris l'inventaire rapide (cf. **4.2.3 Inventaire Rapide**)

FIGURE 6.2 – Inventaire

— Le menu de confection, auquel le joueur accède depuis son vaisseau, lui permet de créer un objet parmi la liste des objets disponibles dans le jeu.

Ce menu est composé d'une fenêtre affichant chaque objet confectionnable accompagné de la liste des matériaux nécessaires à sa confection.

Les objets pouvant être crées à l'aide des matériaux déjà présents dans l'inventaire du joueur sont affichés en blanc, tandis que les objets que le joueur ne peut pas créer actuellement sont grisés, pour que le joueur puisse avoir une indication instantanée des objets qu'il est en mesure d'acquérir.

FIGURE 6.3 – Menu de Confection

6.2 Interface graphique (GUI)

Au sein même du jeu, l'utilisateur dispose d'une interface épurée pour ne pas encombrer sa vision, mais également complète pour lui permettre d'accéder à toutes les informations nécessaires à sa progression.

FIGURE 6.4 – Interface graphique

6.2.1 Barre de vie

Le joueur dispose premièrement d'un espace dans lequel est renseigné son capital de points de vie (représenté par plusieurs coeurs remplis qui se vident selon les dégâts reçus). Cet espace est situé en haut à gauche de l'écran.

6.2.2 Informations du monde

En haut de l'écran, le joueur dispose d'informations relatives à son environnement : une mini-carte affichant la zone autour de lui lui permettant de se repérer, en haut à droite de l'écran, et une horloge lui indiquant l'heure au centre.

6.2.3 Inventaire Rapide

En bas à gauche de l'écran, le joueur peut visualiser 4 de ses objets, qu'il a précédemment assigné dans son "inventaire rapide" et auxquels il peut accéder à l'aide des touches "1", "2", "3" et "4" sur le clavier. Il peut modifier les objets assignés en ouvrant son inventaire complet (cf. **3.1 Menus de jeu**).

6.2.4 Boîtes de dialogues

Lors de la discussion avec les autochtones ou avec l'IA de son vaisseau, le joueur verra apparaître une boîte de dialogues en bas de son écran, qui affichera le texte ainsi qu'un portrait de son interlocuteur.

FIGURE 6.5 – Boite de dialogues

Chapitre 7

Répartition des tâches dans l'équipe

Une fois l'ensemble des tâches réparties équitablement dans les 5 sprints dont nous disposions, nous nous sommes retrouvés avec en moyenne 8 tâches de taille correcte par sprint.
Ainsi, la répartition des tâches fut assez simple, nous avons sélectionné 2 tâches chacun qui nous plaisaient, en fonction de nos capacités et de nos connaissances, car nous n'avions pas tous le même niveau dans chaque domaine.

Il avait été convenu que si les tâches attribuées étaient réalisées trop rapidement, alors elles n'étaient pas égales avec les autres, et dans ce cas les membres du groupe devaient fournir de l'aide à ceux en difficulté avec leurs tâches.

Par la suite, et au cours de chaque sprint, les tâches ont dû être réparties de nouveau et distribuées à 3 personnes au lieu de 4 (cf. **7 Post-Mortem des sprints**)

Chapitre 8

Post-Mortem des sprints

8.1 Premier sprint

Au cours de ce sprint nous devions implémenter les bases de notre jeu à savoir les interactions avec l'environnement, les déplacements du personnage, l'inventaire avec un système d'objet utilisable ainsi que les premiers ennemis.

8.1.1 Points positifs

Nous avons réussi à coder toutes les fonctionnalités fondamentales de manière propre et qui ont su rester fiables jusqu'à la fin du projet. Cela nous a donc fourni des bases solides sur lesquelles s'appuyer pour le reste du développement.

8.1.2 Points négatifs

Ce qui nous a pénalisé au cours de ce sprint est d'avoir été seulement trois à développer au lieu de quatre, cela a causé la non-implémentation de certaines tâches au cours de ce premier sprint comme finir le système d'attaque du personnage ou bien la création de la carte du vaisseau spatial. Nous avons donc pris un petit retard sur notre planification initiale.

8.1.3 Conclusion

Ce sprint a rempli ses principaux objectifs mais sans aller jusqu'au bout. Ce qu'il faudrait revoir c'est la capacité de l'équipe à motiver l'ensemble des membres du groupe afin que chacun travaille équitablement ou au moins fasse les tâches qui lui avaient été assignées. Il est certain que dans un projet d'école, la motivation vis à vis du projet ne sera pas la même pour tout les membres du groupe et c'est aux plus motivés d'essayer de motiver les autres plutôt que d'essayer de compenser en en faisant plus car un travail prévu pour deux ne peut pas être réalisé par une seule personne dans un temps restreint.

8.2 Deuxième sprint

Ce sprint comprenait beaucoup de design, il y avait toutes les zones à créer avec leurs minimap plus des compléments sur le système de combat. En plus de cela nous avions rajouté au dernier moment le système de karma avec les quêtes secondaires ce qui faisait de ce sprint l'un des plus chargés.

8.2.1 Points positifs

Ce sprint a représenté un grand pas en avant pour notre jeu car nous avions à la fin presque tout le décor de notre jeu, nous pouvions enfin voir ou au moins un bon aperçu de ce à quoi il allait ressembler. Ensuite la création de toutes ces zones nous a aussi permis de travailler de facon plus autonome, lorsque l'on travaillait sur un élément spécifique d'une zone, nous pouvions directement développer dans la scène et ajuster en fonction du terrain.

8.2.2 Points négatifs

De même qu'au premier sprint nous avons été en effectif réduit pour ce sprint et avons donc décidé au vu de la charge du sprint de laisser des tâches pour plus tard et de se focaliser à bien faire certaines tâches, de cela a

résulté un retard encore plus important qu'à la fin du premier sprint mais qui ne nous semblait pas ingérable en travaillant beaucoup. Nous avons aussi eu des problèmes au niveau des dépendances entre les différentes tâches de ce même sprint ; afin de finir certaines tâches il était indispensable que d'autres soient finies avant, ce qui a pu retarder l'implémentation de fonctionnalités et donc s'ajouter aux causes du retard de ce sprint.

8.2.3 Conclusion

Le problème de motivation du premier sprint s'est retrouvé encore dans celui-ci, ce qui montre que nous avons été incapable d'y remedier et nous en avons encore souffert, cependant ce sprint a augmenté le moral de l'équipe car le jeu devenait de plus en plus réel et nous avions de plus en plus envie de l'améliorer du mieux que nous pouvions.

8.3 Troisième sprint

Ce sprint était le dernier avant la première revue de sprint, il ne contenait donc aucune fonctionnalité critique qui risquerait d'avoir des effets de bords trop important

8.3.1 Point positif

Ce sprint marque le début réel de notre équipe, nous étions enfin quatre pour coder les fonctionnalités, ce qui nous a permis de rattraper une partie du retard que nous avions pris lors des deux premiers sprints. Lors de la revue de sprint, le jeu comportait certain bug dû à l'ajout tardif de certaines fonctionnalitées mais nous les avons corrigé dans la demi-heure qui suivit la revue, donc nous étions contents d'avoir bien progressé sur ce sprint.

8.3.2 Point négatif

Peu de choses à dire ici, bien qu'il nous restait un peu de retard par rapport au planning, nous étions en train de le rattraper et les quelques bugs ont vite été corrigés.

8.3.3 Conclusion

Nous avons été satisfait de notre avancée jusque là et de comment s'est passée la revue de sprint. De plus étant quatre, nous allions pouvoir avancer à un meilleur rythme pour la suite des sprints.

8.4 Quatrième sprint

Ce sprint avait pour objectif de finir tous les systèmes de combat, c'est à dire l'implémentation de toutes les IA et finaliser le système de prise de dégâts.

Ce sprint était sûrement le plus intéressant de tous car nous avions Les IA à développer, nous avons pu découvrir des notions intéressantes et les appliquer dans notre jeu en créant différents types d'intelligences artificielles.

Nous avions placé uniquement les IA dans ce sprint (en plus de quelques tâches secondaires) car nous pensions que leur développement était extrêmement chronophage. Cependant nous nous sommes bien débrouillés et avons pu coder ces dernières en plus de fonctionnalités supplémentaires (telles que l'interface de confection) dans les temps, et avons pu rattraper le retard que nous avions accumulé sur certaines fonctionnalités.

8.5 Cinquième sprint

Le cinquième script, se terminant le jour de la présentation orale, a volontairement été rempli de tâches superficielles, n'ayant pas un impact direct sur le jeu mais ayant pour but de peaufiner ce dernier.
Parmi ces tâches on retrouvait par exemple la création de tous les dialogues du jeu, des différentes cinématiques et des tutoriels, ainsi que quelques améliorations telles que les animations de mort des ennemis.

Le jour de la présentation, la plupart de ces fonctionnalités avaient été implémentées, mais il nous manquait toujours les différentes cinématiques, dû au fait qu'aucun membre de notre groupe n'avait de connaissances dans ce domaine. Cette fonctionnalité non implémentée n'impacte pas réellement le jeu en lui-même, mais aurait été un bel ajout. Sa présence dans le dernier sprint indique que ce n'était pas une tâche sur laquelle nous allions

nous focaliser.

Nous avons cependant profité, comme prévu, de ce sprint pour finaliser l'ensemble des fonctionnalités non-terminées ou améliorables des sprints précedents.

Chapitre 9

Post-mortem global du projet

Le projet dans son ensemble a constitué une expérience enrichissante pour la majorité du groupe, qui a pu développer un jeu en partant uniquement de sa conception, jusqu'à obtenir un prototype convaincant.

Le manque d'implication de certains membres a rendu la tâche ardue en début de projet, mais au fil des sprints, notre jeu prenait de l'ampleur, et notre motivation avec. De plus, l'ajout de certaines fonctionnalités était très satisfaisant et nous incitait à pousser le développement plus loin.

Malgré les légers retards explicités précedemment, nous avons réussi à nous tenir à notre planning, qui était très cohérent pour une estimation, et la charge de travail était correcte au cours de chaque sprint.
Le travail d'équipe était efficace, et nous avons réussi à fournir un code lisible chacun de notre coté, ce qui a permis de ne pas perdre de temps à essayer de comprendre les codes des autres membres.
La disponibilité des différents membres, que ce soit en face à face ou via discord a également permis de résoudre de nombreux problèmes, qui auraient pu créer de gros retards sur le planning.

Finalement, le jeu dont nous disposons actuellement nous satisfait amplement, même si certaines fonctionnalités prévus à l'origine n'ont pas pu être implémentées, et nous sommes arrivé à la hauteur de nos attentes, nous sommes donc très satisfaits de notre projet et de l'ensemble de la matière en général.

Chapitre 10

Pourcentage de participation

Il est difficile d'évaluer sa propre implication ainsi que celle des autres membres du groupe, mais après concertation avec les différents membres du groupe, nous sommes arrivés à la répartition suivante :

— Adrien NOUVELLON : 5%, il a réalisé une seule tâche sur le trello pendant toute la durée du projet, et n'a commencé qu'à partir du 3ème sprint. Il a cependant aidé à la réalisation de la présentation et du GDD.

— Thomas GUBENO : 30%, il a réalisé la majorité des tâches qui lui étaient demandées, et a aidé d'autres membres (notamment Simon) sur les tâches qui nécessitaient plusieurs personnes. Il a cependant eu une baisse de régime dû à la non-activité de certains membres.

— Pierrick BARBARROUX : 25%, il a réalisé toutes les tâches qui lui étaient demandées, mais souvent au dernier moment ou en retard, ce qui compliquait la réalisation d'autres tâches en relation.
Son code était cependant clair et fonctionnel, et il a participé à la réalisation de plusieurs tâches très chronophages.

— Simon LAUZERAL : 40%, il a mené le projet et a réalisé la totalité des tâches qui lui étaient attribuées, en plus de réaliser celles qui étaient laissées à l'abandon par Adrien. C'est Simon qui s'est occupé de toute l'organisation du projet en général, à la fois dans son architecture et dans sa réalisation, et était en contact permanent avec les autres membres pour leur donner des instructions / les aider pour une tâche compliquée, tout au long du projet.

Chapitre 11

Annexe

11.1 Description de chaque classe

Dans cette partie nous parcourerons l'ensemble des classes (scripts) que nous avons créées, pourquoi nous les avons créées et ce qu'elles sont censés faire.

Coffre

— **ChestInventory** : gère l'inventaire du coffre. Permet de pouvoir ajouter des items dans le coffre ou d'en enlever par exemple.

— **ChestInventoryUI** : gère l'affichage de l'inventaire du coffre dans la fenêtre du jeu ainsi que ses interactions en parallèle avec la classe ChestInventory.

Table de craft

— **CraftItem** : gère l'ensemble des interactions avec la table de craft (création d'un item,...) lorsque le joueur navigue dans l'UI de la table de craft.

— **openCraftStation** : ouverture et fermeture de l'interface de craft. Cette classe sert uniquement à ouvrir et fermer l'UI de la table de craft.

— **CraftStation** : hérite de Interactable. Ouvre l'interface de craft via la classe openCraftStation lorsque le joueur interagit avec la table de craft.

Cycle jour/nuit

— **BedController** : hérite d'Interactable. Permet au joueur d'utiliser le lit dans le vaisseau de Sam pour faire passer le temps. Cette classe permet donc de changer l'heure du jeu via l'interaction du joueur avec le lit.

— **SwitchDayNight** : gère la transition du jour vers la nuit et inversement.

Dialogue

— **DialogueChefDuVillage** : hérite d'Interactable.

— **DialogueGuard** : hérite d'Interactable. Gère les dialogues du garde lorsque le joueur arrive au village pour la première fois. Deux dialogues différents sont lancés selon si le joueur dispose de la lampe torche dans son inventaire ou non.

— **DialogueIAArrivee** : lance le dialogue de l'IA lorsque le joueur arrive sur la planète inconnue.

— **TriggerDialogueChef** : lance le dialogue du chef du village lorsque le joueur vole le joyau du village.

— **Dialogue** : classe décrivant un dialogue et donc ses attributs (nom du personnage, sprite du personnage, phrase du dialogue).

— **DialogueManager** : gère l'affichage des objets Dialogue (lancement du dialogue, écriture sur l'UI, fermeture du dialogue, ...)

— **DialogueTrigger** : appelle la fonction StartDialogue du DialogueManager. Est utilisé pour les events.

Ennemis

— **Ennemy** : classe commune à tous les ennemis du jeu, où du moins à tous les objets auxquels Sam peut infliger des dégâts. Gère la vie, les dégâts, la période d'invincibilité, les knockbacks et hitstuns des ennemis.

Les classes suivantes héritent de la classe **Ennemy** :

— **Chicken** : si un poulet est tué, le karma de Sam diminue.

— **Crabe** : gère le déplacement du crabe et appelle la fonction ManageHealth() à chaque frame.

— **Follow_Charge** : IA des ennemis qui charge le joueur lorsque celui-ci se trouve dans un certain périmètre de l'ennemi.

— **Following_Throwers** : hérite de Thrower. IA des ennemis qui suivent le joueur et lance des projectiles si ce dernier est dans un certain périmètre.

— **FollowingEnnemy** : IA des ennemis qui suivent le joueur.

— **Lich** : gère le comportement de la liche vis-à-vis de Sam.

— **OrcJambeDeBois** : est attaché à l'orc qui lache une jambe de bois lorsqu'il meurt. Ce script gère ce comportement là. hérite de Thrower.

— **PathDefined** : classe vide, ne sert nulle part.

— **Thrower** : IA des ennemis qui lance un projectile si le joueur est dans un périmètre assez proche de l'enneme. Ne gère pas le déplacement de l'ennemi, cela est géré dans la classe Followin_Throwers

— **wolf_moves** : gère le déplacement latéral des loups. Les loups alternent entre droite et gauche.

Interactable

— **Interactable** : classe permettant au joueur d'interagir avec la touche E avec les éléments de la scène. Sam devra être assez près d'un objet pour interagir avec lui. La détection est gérée par des colliders.

Les classes suivantes héritent de la classe interactable

— **BlockingArea** : est rattaché à chaque objet qui bloque l'accès à une zone. Permet de détecter si le joueur peut briser l'obstacle en vérifiant qu'il dispose du bon objet dans son inventaire.

— **ChangeScene** : effectue les transitions d'une scène à l'autre.

— **ChefHouse** : permet de réaliser la quête pour la liche où l'on doit uriner dans la maison du chef.

— **Chest** : permet d'ouvrir le coffre.

— **DialogueMan** : lance un dialogue.

— **GoToIsland** : hérite de ChangeScene. Override la fonction interact(). Vérifie si le joueur a le moteur du bateau lui permettant d'aller sur l'île.

— **InjuredMan** : gère l'interaction avec l'homme blessé dans la montagne et la quête qui en découle (karma, ...).

— **KarmaPickUpInteractable** : hérite de PickUpInteractable. Override la fonction interact(). Cette classe ajoute la baisse du karma de Sam si ce dernier prend un item qu'il ne devrait pas (exemple : pomme).

— **LegLessMan** : gère l'interaction et donc la quête avec l'homme sans jambe du village. On doit lui rapporter la jambe de bois qu'on récupère en tuant un orc dans la forêt.

— **MendiantInteraction** : est attaché au mendiant dans le village. On peut lui donner des pommes ce qui fait augmenter notre karma. Cependant on ne peut dépasser un certain nombre de pommes par jour.

— **PickUpInteractable** : est attaché à tous les objets que l'on peut ramasser (loot,...). Ce script permet de ramasser des objets et de les stocker dans notre inventaire.

— **piedestal** : ce script gère la venue du chef lorsque Sam vole le joyau du village.

— **SecretPath** : Sam passe par un passage secret pour aller voler le joyau du village. Ce script gère la transition entre l'entrée du passage (le point d'interaction) et le lieu où est caché le joyau du village.

— **ToPlanet** : avant d'arriver sur la planète, Sam est dans son vaisseau. Le joueur effectue un premier tutoriel à l'interieur du vaisseau. Ce script vérifie que le joueur a bien effectué le tutoriel avant de pouvoir sortir du vaisseau pour découvrir la planète sur laquelle il vient d'atterrir.

Inventaire

— **Inventory** : gère la manipulation des objets de l'inventaire de Sam (ajout, supression,...).

— **InventorySlot** : gère l'utilisation des items quand l'inventaire est ouvert.

— **InvetoryUI** : gère l'affichage de l'inventaire en utilisant Inventory et InventorySlot. Permet d'assigner des objets de l'inventaire dans les slots d'utilisation rapide.

— **Consumable** : est attaché aux items que Sam peut utiliser pour se soigner. Permet d'utiliser ces items et soigne Sam d'une certaine valeur de points de vie.

— **MeleeWeapon** : est rattaché aux armes de mélée. Permet de s'équiper d'une arme de mélée lorsqu'elle est utilisée via l'inventaire.

— **Range Weapon** : est rattaché aux armes à distance. Permet de s'équipper d'une arme à distance lorsqu'elle est utilisée via l'inventaire.

Usable :

Offrande

— **CaisseOffrande** : permet de récupérer les offrandes dans la caisse à offrande vers l'inventaire de Sam.

— **ManageOffrande** : gère les offrandes laissées par les villageois après que Sam aie vaincu un boss.

— **ProjectileBehaviour** : est rattaché à chaque projectile. Gère le déplacement, la vitesse, les dégâts, le knockback et le hitstun d'un projectile.

— **AssignedItem** : permet d'utiliser les items qui sont assignés dans l'inventaire rapide.

— **QuickInvManager** : gère l'inventaire rapide de Sam.

Sam

— **PlayerData** : permet de garder en mémoire les données relatives à Sam (vie, karma,...) entre chaque transition de scène.

— **SamAttackController** : gère les attaques au corps à coprs et à distance de Sam, en particulier le temps d'attente entre deux attaques.

— **SamController** : gère les déplacements, la vitesse, l'état, la vie, les inputs,... de Sam.

Tutoriel

— **tutoCraft** : gère les dialogues du tutoriel de craft au début du jeu.

Autres

— **ArenaSpikes** : gère les dommages dûs à la collision des pics dans l'arène sur l'île.

— **CameraManager** : gère les déplacements de la caméra.

— **HideMiniMap** : active ou désactive la minimap selon si le joueur est dans le vaisseau spatial ou non.

— **LoadOnLaunch** : gère le chargement des scènes via les sauvegardes.

— **MainMenu** : controlleur du menu principal.

— **PauseManu** : controlleur du menu pause.

— **SaveSystem** : permet de sauvegarder une partie.

— **ZoneName** : gère l'affichage du nom de la zone dans laquelle se trouve Sam.