



Scraft - Premier rapport de soutenance

Vincent AUPEST, Mathis BERTRAND

Felix MUHLKE, Louis VIDELIER

Table des matières

Introduction	4
1 Origine et nouvelle nature du projet	5
1.1 Origine	5
1.2 Nouvelle nature du projet	6
1.3 Fiche du projet	6
1.4 État de l'art	7
2 Distribution des tâches	8
2.1 Subdivisions réalisées et nouvelles catégories	8
2.1.1 Subdivisions	8
2.1.2 Suppression et ajout de catégories	8
3 Découpage du projet	10
3.1 Découpage en parties	10
4 Planification et avancement	11
4.1 Intelligence Artificielle	11
4.1.1 Auteur	11
4.1.2 Description	11
4.1.3 Implémentation	12
4.1.4 Différentes entités	12
4.2 Design	13
4.2.1 Auteur	13
4.2.2 Description	13
4.2.3 Fonctionnement	13
4.2.4 Rendu	13
4.3 Items	14
4.3.1 Auteurs	14
4.3.2 Description	14
4.3.3 Réalisation	14
4.3.4 Classes	15
4.4 Inventaire	16
4.4.1 Auteur	16
4.4.2 Description	16
4.4.3 Réalisation	16

4.5	Joueur	17
4.5.1	Auteur	17
4.5.2	Fonctionnel	17
4.5.3	Réalisation	17
4.6	Props	19
4.6.1	Auteurs	19
4.6.2	Description	19
4.6.3	Réalisation	19
5	Structure du projet	20
5.1	Fonctionnel	20
5.2	Méthodologique et Technologique	21
5.2.1	Executable	21
5.2.2	Gestion du projet	22
5.2.3	Communication	22
5.3	Opérationnel	23
	Conclusion	23

Introduction

Ce rapport constitue un résumé de l'avancement de notre projet, Scraft, de la période de validation du Cahier des Charges jusqu'à la première soutenance.

Tout d'abord, il est important de mentionner que certains éléments de notre planning ont dû être modifiés en raison des problèmes auxquels nous avons dû faire face.

De plus, le multi-joueur étant une fonctionnalité à l'échelle globale de notre jeu, son développement s'est avéré plus complexe que prévu (tous les éléments devant être ajoutés à l'écosystème multi-joueur).

Enfin, en raison de dépendances entre les fonctionnalités, certains ajouts ont dû être fait avant d'autres, ce qui nous a contraint de modifier quelques points de notre agenda.

Nous verrons donc dans ce rapport, l'évolution des différents parties nécessaires au bon fonctionnement de notre jeu. Pour chacune de ces parties nous mentionnerons les auteurs en charge de la dite fonctionnalité et de la méthode choisie pour en réaliser son implémentation.

1 Origine et nouvelle nature du projet

1.1 Origine

La création du groupe s'est faite rapidement, puisque dès l'annonce du début du projet S2 le groupe comptait déjà trois membres actifs. Mathis étant le quatrième membre essentiel au bon fonctionnement du projet, le groupe Megawaves est ainsi né.

Ensuite, l'idée de faire un jeu vidéo a été une évidence dès les premières mises en commun du groupe. Nous nous sommes très rapidement mis d'accord sur la création d'un jeu de survie.

L'idée de Scraft est née après des heures de communications entre les membres de l'équipe, mais aussi à l'inspiration de jeux déjà existant comme Rimworld, ou encore Factorio.

- **Vincent AUPEST** : Ancien élève de terminale du lycée, avec la spécialité NSI, j'ai déjà effectué plusieurs "projets" de jeux vidéos avec le langage de programmation Python dans le cadre de ma scolarité. Aujourd'hui, je me sens prêt à assurer ce projet ainsi que sa direction.
- **Mathis BERTRAND** : Ayant toujours été proche de l'univers des jeux vidéos, mon intérêt envers l'informatique s'est fait naturellement, mais sans jamais vraiment sans approcher. La spécialité NSI que j'ai suivi au lycée a directement été une révélation pour moi. Dès mes premiers cours de NSI j'étais convaincu sur mon avenir. N'ayant jamais conçu de petits jeux ou de projets d'une telle envergure. Ce projet sera l'occasion de donner une vraie application directe à mon travail. Mes espérances pour notre jeu sont d'en apprendre énormément sur le développement globale d'un projet.
- **Felix MUHLKE** : N'ayant pas fait NSI (ou option Informatique dans mon lycée), je n'ai pas la même assurance que certains de mes camarades dans la création d'un jeu vidéo. Et pourtant, depuis que j'ai découvert l'informatique par mes propres moyens j'ai toujours eu envie d'en faire un. Le faire seul et sans assistance m'avait un peu effrayé, je suis donc impatient à l'idée de commencer ce projet en groupe. J'ai d'ailleurs déjà pu constater le plaisir de faire un projet en groupe lors de l'élaboration du concept de notre jeu.

- **Louis VIDELIER** : Venant d'un petit lycée de province, j'ai toujours aimé ce qui était en lien avec la programmation et l'informatique au cours de ma scolarité (Scratch au collège, Python en seconde et au cours de mes deux ans de NSI). Grand amateur de programmation orientée objet, je me sens prêt à relever les défis de ce projet (Réseau, API avec Unity, etc.) et prendre mes premières marques dans le monde du *gamedev*.

1.2 Nouvelle nature du projet

Face à la difficulté pour nous de trouver des textures adéquat au thème précédemment sélectionné, nous avons décidé à l'unanimité de changer le thème pour le "Post-apocalyptique". Le nom de "Scraft" ne fait plus que référence a "scrap" pour ferraille et "craft" pour fabrication.

1.3 Fiche du projet

- Genre : Multijoueur / TPS / PVE
- Style : Action / Aventure / Survie
- Plateforme : PC Windows
- Moteur de jeu : Unity
- Nombre de joueurs : 1-4

1.4 État de l'art



Scraft s'inscrit dans la grande lignée des jeux de survie, mais comme chaque jeu, il aura ses propres particularités qui lui permettront de se distinguer, dans notre cas celles-ci sont :

- Son style graphique : avec des sprites 32×32 pixels et 16×16 , le tout étant réalisé dans un set de couleurs varié.
- Son gameplay unique : une survie classique pouvant tourner au chaos au fur et à mesure du temps en raison de la quantité de monstre à chaque vague.

De plus cette oeuvre s'inspire comme vu précédemment, de Rimworld (c) pour le côté survie, de Factorio (b) pour la petite partie industrielle du jeu, mais aussi de The Escapists (a) pour la POV¹ des personnages.

1. Point de vue

2 Distribution des tâches

En raison de la nature des éléments à implémenter, certaines catégories ont été subdivisées afin de mieux décrire les participations de chaque membre dans chaque domaine.

2.1 Subdivisions réalisées et nouvelles catégories

2.1.1 Subdivisions

Inventaire/Craft → Inventaire + Craft

Animation et graphismes → Animations + graphismes

2.1.2 Suppression et ajout de catégories

Bases du jeu → \emptyset

\emptyset → Player

\emptyset → Intelligence Artificielle

\emptyset → Objectif

\emptyset → Sauvegarder/Charger

\emptyset → Arbre des recherches

Nous avons décidé de supprimer la catégorie "Bases du jeu" au profit de nouvelles ; "Player", "Intelligence Artificielle" et "Objectif".

Enfin nous avons ajouté les catégories "Sauvegarder/Charger" et "Arbre des recherches", deux fonctionnalités précédemment manquantes. La catégorie "Équilibrage" a également été ajoutée.

<i>Tâches</i>	Louis	Felix	Mathis	Vincent
Player			X	
Réseau	O	O	X	O
Animations		O	X	
Sprites			O	X
Items	X			O
Interface			O	X
Inventaire			X	
Craft	X			
Intelligence Artificielle		X		
Génération de la carte	O			X
Objectif		X		
Son	X			
Jour & Nuit & Lumière			X	
Construction	O	X		O
Équilibrage				X
Sauvegarder/Charger		X		
Arbre des recherches	X	O		
Mise en avant	X			O

FIGURE 2 – X : Référent, O : Suppléant

Une fois le développement commencé, nous nous sommes rendu compte que nous avions parfois mal réparti les tâches. Voici les modifications à souligner :

Nous avons décidé d'implémenter le multijoueur au fur et à mesure du développement pour ne pas avoir à modifier à nouveau le code déjà écrit. À chaque fonctionnalité terminée, on implémente sa partie réseau de manière à la faire fonctionner en multijoueur et on passe à la suivante. Ainsi chacun d'entre nous réalisera, à un certain moment de sa participation, l'intégration réseau de ses objets.

De manière générale nous nous sommes rendu compte que travailler à plusieurs sur une même fonctionnalité n'est pas optimal car nous sommes parfois ralentis par les autres, et les conflits liés à Git. Afin de simplifier le développement, le nombre de personnes par fonctionnalité a été réduit.

3 Découpage du projet

3.1 Découpage en parties

1. Player : Mouvements/Animations/Attaque en multijoueur
2. Items : Sprites/attributs/Prefabs
3. IA : Comportements/Animations en multijoueur
4. Inventaire : stockage/UI¹ en multijoueur
5. Construction
 - Item
 - Craft²
 - Bâtiments
 - Props (entité physique fixe)
6. Mise en place de l'objectif (Histoire/Apparition)
 - Vagues
7. Jour/Nuit + Lumières
8. Sons
9. Implémentation du TECH-TREE³ (+ ajouts fonctionnalités)
10. Debug et optimisation
11. Mis en avant et accessibilité
 - Site
 - Trailer⁴
 - .exe

1. Interface utilisateur
2. Fabrication via une interface
3. Arbre des recherches
4. Vidéo de présentation

4 Planification et avancement

<i>Soutenance</i>	1 (actuel)	2	3
Player	60%	80%	100%
Réseau	33%	66%	100%
Animations	40%	75%	100%
Design	33%	66%	100%
Interface	20%	50%	100%
Inventaire	70%	100%	100%
Craft	0%	50%	100%
Intelligence Artificielle	30%	60%	100%
Génération de la carte	0%	50%	100%
Objectif	0%	50%	100%
Son	0%	0%	100%
Jour & Nuit & Lumière	0%	0%	100%
Construction	0%	50%	100%
Équilibrage	20%	50%	100%
Sauvegarder/Charger	0%	0%	100%
Mise en avant	15%	40%	100%

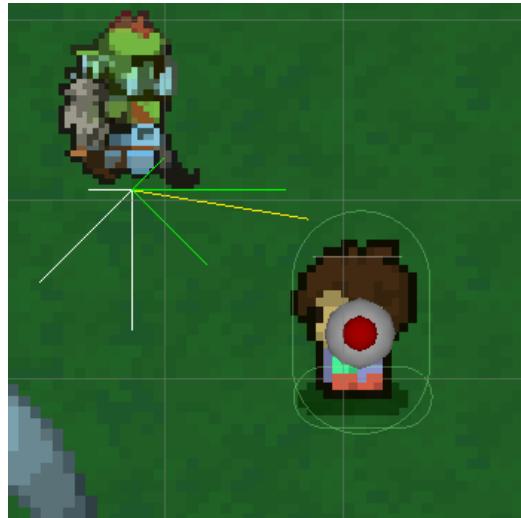
4.1 Intelligence Artificielle

4.1.1 Auteur

MUHLKE Felix - Responsable de l'implémentation du comportement des entités.

4.1.2 Description

L'objectif principal du jeu est la survie. En effet, le joueur va rencontrer des hordes de monstres durant son périple. C'est pourquoi nous avons dû créer un système d'intelligence artificielle capable de traquer les joueurs les plus proches. Tout cela dans une zone prédéfinie qui constitue la zone dans laquelle l'ennemi est capable de "voir" le joueur et ainsi de le suivre pour l'attaquer.



(a) Vecteurs du mouvement de l'ennemi

4.1.3 Implémentation

Ce système a été réalisé en étudiant les obstacles et le(s) joueur(s) autour de chaque monstre afin de déterminer quelle direction choisir pour atteindre le joueur. Cela fonctionne en utilisant deux listes, "intérêt" et "danger", chacune étant composée de 8 nombres flottants compris entre 0 et 1, obtenus à partir de produits scalaires de vecteurs. Puis en manipulant ces deux listes, on obtient un vecteur moyen qui nous donne la direction optimale locale. Cela est permis grâce à une zone de détection des joueurs et des obstacles, qui, afin d'éviter que les monstres ne suivent les joueurs partout sur la carte, est limitée. Il s'agit donc d'une implémentation assez naïve puisqu'elle est très locale.

4.1.4 Différentes entités

Nous avons donc implémenté le mouvement d'un ennemi. Il nous reste à implémenter ses dégâts et sa santé ainsi qu'implémenter le comportement des masses d'ennemis. Les ennemis n'étant pas les seuls personnages non joueurs (PNJ), il nous faut également implémenter des animaux inoffensifs, qui seront donc des entités passives, se déplaçant de manière aléatoire dans une certaine zone, ainsi que des animaux neutres, qui se comporteront comme des ennemis lorsqu'ils se font attaqués par un joueur.

4.2 Design

4.2.1 Auteur

AUPEST Vincent - Réalisation des Sprites.

4.2.2 Description

Pour les graphismes généraux du projet, nous avons sélectionné une banque de Sprites répondant à la majeure partie de nos critères.

Malheureusement, aucune banque de Sprites ne répondait à 100% de nos critères, c'est pourquoi nous consacrons une partie entière du projet à la conception de Sprites.

À l'état actuel, nous avons déjà produit une bonne partie des Sprites nécessaire au bon fonctionnement de l'inventaire et du personnage (environ 160).

4.2.3 Fonctionnement

Aseprite - Logiciel de création de PixelArt



Nous avons décidé d'utiliser Aseprite comme logiciel de dessin, pour sa simplicité d'utilisation et sa qualité de rendu.

Quant à la conception des Sprites, nous avons deux types de Sprites, la version 16x16 pour l'inventaire, et son équivalent en 32x32 pour l'affichage dans la main du joueur. Nous devions différencier ces 2 cas pour avoir un rendu propre en inventaire, et à l'échelle dans la main du joueur.

4.2.4 Rendu



(a) Minerais de fer



(b) Lance en silex



(c) Navet



(d) Plastron en uranium



(e) Pioche en pierre

4.3 Items

4.3.1 Auteurs

VIDELIER Louis - Responsable de la majeure partie de l'implémentation. Réalisation de la hiérarchie des classes nécessaire à implémenter les différents items.

AUPEST Vincent - Réalisation de la totalité des prefabs de chaque item.

4.3.2 Description

Les items sont les objets qui représentent des ressources, au sol, ou dans un inventaire (coffre, inventaire du joueur, etc.). Par exemple, lorsqu'un arbre est coupé par un joueur, des items apparaissent au sol (bois, bois dur, bâtons). Le joueur peut ensuite les récupérer, les utiliser pour fabriquer de nouveaux objets, ou les reposer au sol s'il n'en a pas besoin ou pour les donner à un autre joueur.

Les items sont donc des objets complexes par leur concept de modéliser de l'information sans toujours être matérialisé par un GameObject dans Unity.

4.3.3 Réalisation

Les items sont composés de deux scripts C#. Le premier, un MonoBehaviour, permet de catégoriser chaque type d'item (une arme, un consommable, un outil, etc.). Le second, un ScriptableObject, contient toutes les valeurs pour un type d'objet. Par exemple, pour un consommable, le nombre de points de vie gagnés en consommant l'objet.

A partir du GameObject qui représente notre item (qui est donc muni du MonoBehaviour de la famille de classes qui décrivent les items) on en garde une copie sous forme de prefab Unity, à laquelle on lie les données du ScriptableObject (et inversement ; on lit le prefab au ScriptableObject).

Cette double-référence nous permet de pouvoir créer des GameObjects à partir de notre prefab en ayant toutes les données, puis, de ne représenter l'item dans un inventaire uniquement grâce au ScriptableObject.

En utilisant ce système, on permet à tous les items du même type de se comporter de manière identique. En effet ; tous les consommables agissent sur les mêmes valeurs, mais avec des valeurs différentes, on utilise donc un seul MonoBehaviour "ConsumableItem" pour décrire leurs effets sur le joueur

à partir des données du ScriptableObjet. Cependant, il est tout de même nécessaire de diviser les types d'items en classes différentes, car tous les items ne fonctionne pas de la même manière (un outil ne peut pas être utilisé comme un consommable) ils ont donc des scripts différents. De même chaque classe d'item à donc besoin de données différentes, c'est-à-dire un ScriptableObject différent.

4.3.4 Classes

Les diagrammes UML représentants les relations entre les différentes classes nécessaires pour l'implémentation du système d'items sont disponible au format *.png* ci-joins ;

[Premier diagramme]

[Second diagramme]

4.4 Inventaire

4.4.1 Auteur

BERTRAND Mathis - Responsable de la totalité de l'implémentation et design de l'inventaire.

4.4.2 Description

Chaque joueur possède son propre inventaire qui lui permet d'équiper, stocker, jeter et obtenir des informations sur les items qu'il possède. L'inventaire est une interface graphique interactive qui s'ouvre et se ferme. L'inventaire est composé de quatre parties : 4 emplacements réservés pour équiper l'armure sur le joueur, 30 emplacements de stockage, 1 champ affichant les informations de l'item sélectionné (nom, description, durabilité, dégâts etc...) et 10 emplacements supplémentaires pour la barres d'actions qui permettront au joueur d'utiliser directement 10 de ses items en jeu sans ouvrir son inventaire. Dans l'inventaire les items "regroupables" sont stockés par 100 (par emplacement).

4.4.3 Réalisation

L'affichage de l'inventaire est réalisé avec l'interface graphique de Unity (les UI). L'inventaire une fois affiché utilise un système de "Drag and Drop"¹ pour déplacer à l'aide de la souris les différents items, on "attrape" un item sur un emplacement et on le "relâche" dans un autre. Actuellement la partie stockage et barre d'action est partiellement implémentée. Nous pouvons ajouter, supprimer et déplacer des éléments.

1. Déplacer et poser



(a) Aperçu de l'inventaire

4.5 Joueur

4.5.1 Auteur

BERTRAND Mathis - Responsable de la totalité de l'implémentation et design du joueur.

4.5.2 Fonctionnel

Chaque utilisateur contrôle son propre personnage. Il est ici question du joueur tel que le personnage, capable de se déplacer et d'utiliser les objets qu'il a à sa disposition.

Le joueur a une animation pour le rendre vivant et une animation pour déclencher une attaque. Il ne peut tenir et utiliser qu'un item à la fois. L'utilisateur déplace son joueur avec son clavier et vise avec sa souris pour attaquer dirigeant toujours son regard vers le pointeur de la souris. Le joueur tient en main l'item sélectionné dans sa barre d'action, par défaut si l'emplacement sélectionné est vide le joueur porte un gant de boxe. De même, l'item tenu est toujours en direction du pointeur de la souris pour bien identifier la direction de l'attaque.

4.5.3 Réalisation

Actuellement l'intégralité des fonctionnalités sur le joueur sont fonctionnelles en multijoueur. La déplacement du joueur, son animation, la position de son arme et l'animation de son arme sont synchronisés entre tous les clients.

Le joueur possède une animation quand il reste sans bouger dans les 4 directions cardinales, ainsi qu'une animation quand il se déplace. Lorsqu'on attaque notre item tenu en main a une animation de mouvement vers l'avant avec une rotation vers la où il vise pour mimer un coup.

Afin de différencier les joueurs les uns des autres, nous avons utilisé des shaders¹ sur le visuel du joueur pour que chacun puisse modifier personnellement la couleur des cheveux, de peau, du T-shirt, de la veste, du pantalon ainsi que la couleur des chaussures de son personnage. Les couleurs du personnages sont également synchronisés entre les utilisateurs.

La réalisation de toutes ses fonctionnalités m'a beaucoup posé de problèmes, pour la synchronisation multijoueur, étant le premier à me pencher sur le sujet et n'ayant jamais fait de multijoueur auparavant. J'ai mis beaucoup de temps à comprendre l'interface et les notions du framework² Mirror.



(a) Aperçu de deux joueurs face à face.

1. Un shader ou nuanceur (le mot est issu du verbe anglais *to shade* pris dans le sens de « nuancer ») est un programme informatique, utilisé en image de synthèse, pour paramétriser une partie du processus de rendu réalisé par une carte graphique ou un moteur de rendu logiciel.

2. Cadriciel

4.6 Props

4.6.1 Auteurs

AUPEST Vincent - Référent Props. Responsable développement première implémentation.

VIDELIER Louis - Suppléant Props. Responsable développement ré-écriture des Props.

4.6.2 Description

Les objets physiques du monde (arbres, murs, portes, etc.) sont des objets à part entière ; le joueur doit pouvoir être en mesure de les utiliser indépendamment de leur type (le joueur doit pouvoir interagir avec une porte de la même manière qu'avec un four ou tout autre objet physique). En raison de leur nature diverse, nous les nommons "Props" comme équivalent à "Construction" (cette définition inclue également les plantes et autres objets naturels). Ceux-ci sont encore en cours de développement, au travers de MonoBehaviours généralistes et de ScriptableObjects (à la manière des Items).

4.6.3 Réalisation

Leur réalisation a commencée de manière naïve lorsque nous ne connaissions pas les ScriptableObjects. Ceux-ci sont actuellement en cours de réécriture afin de simplifier la production d'une importante quantité de Props pour tous les besoins futurs de notre jeu.



(a) Exemple de Props

5 Structure du projet

5.1 Fonctionnel

Cette partie évoque l'intégralité des fonctionnalités qui seront présentes dans Scraft.

- Situation initiale : Les différents membres apparaissent à proximité les uns des autres.
- Création d'un arbre des connaissances communs entre les joueurs.
- Système multijoueur en *Peer-to-Peer*¹ (Sur Internet ; ou en LAN²) pour pouvoir jouer directement, sans l'intermédiaire d'un serveur. Chaque joueur peut héberger le jeu en local et ainsi, jouer seul.
- Système d'items qui peuvent être utilisés par les joueurs pour interagir avec leur environnement. Ainsi qu'un système de besoins de nourriture.
- Écosystème de machines, d'électricité entre les machines qui en produisent, les machines qui en consomme, etc.
- Système de génération d'une carte (par région, biome, puis hotspot)
- Système d'inventaire, de remise à zéro lorsque le joueur meurt.
- Des vagues d'ennemis régulières. Ainsi que des entités déjà présentent sur la carte.
- Une configuration des paramètres personnalisés.

1. L'architecture *Peer-to-Peer*, est une architecture en réseau où les différents clients échangent entre eux, sans nécessiter de serveur central. Dans le monde du jeu-vidéo, un jeu en *Peer-to-Peer* (abrégé en P2P) est opposé au modèle CGS (*Central Game-Server*), où les joueurs doivent communiquer au travers d'un serveur héberge par l'éditeur.

2. *Local-Access-Network*, soit le réseau local.

5.2 Méthodologique et Technologique

Au cours du développement de Scraft, nous utiliserons différents logiciels pour concevoir le jeu, gérer sa production, générer de la documentation, ou encore créer l'identité graphique du jeu.

5.2.1 Executable

Pour concevoir le programme du jeu (le fichier exécutable, qui permettra aux joueurs de jouer au jeu), nous utiliserons les technologies suivantes ;

Unity - Moteur de jeu



Le moteur Unity nous permettra de pouvoir directement travailler sur l'environnement de notre jeu, plutôt que sur l'affichage, la physique, etc. Il s'agit d'un moteur de jeu.

C# - Langage de programmation



Le langage de programmation C# est un langage de programmation Orienté Objet de haut niveau utilisé par Unity pour le comportement des objets du jeu.

Rider - IDE



L'IDE Rider est une environnement de développement créé par JetBrains et adapté pour le langage de programmation C#, utilisé par Unity.

5.2.2 Gestion du projet

Pour gérer le projet en lui même, nous utiliserons les technologies suivantes ;

Git - Outil de versionnage



Git est un outil développé par Linus Torvald (plus connu comme étant le créateur du noyau Linux), il permet de gérer le développement d'un logiciel ou d'un projet de manière incrémentale sous la forme de "repos" pour stocker les différentes versions, avec un modèle client-serveur.

GitHub - Host Git et Organisation



GitHub est une plateforme rachetée par Microsoft qui permet la gestion de projet avec Git, accompagné une surcouche pour gérer les membres d'un projet au sein d'une organisation.

5.2.3 Communication

Afin de communiquer entre nous ou pour concevoir des rapports concernant l'avancée du projet, nous utiliserons les technologies suivantes ;

Discord - Tchat en ligne



Discord est une plateforme de communication en ligne ouverte à tous où les utilisateurs se retrouvent au sein de groupes ou de serveurs pour échanger autour de sujet communs. Nous l'utiliserons pour échanger lors des phases de développement du jeu. (3230 messages au total)

Overleaf - Rédaction L^AT_EX en équipe



Overleaf est une plateforme que nous utiliserons pour rédiger nos rapports et autres documents au format L^AT_EX.

5.3 Opérationnel

Comme nous comptons développer notre multijoueur en Peer-To-Peer, nous n'aurons pas besoin de louer un serveur. Nous aurons cependant besoin d'un hébergeur de sites web ce qui représente environ 3€/mois.

Conclusion

Malgré certaines surprises aux fondements du projet (notamment au sujet de la partie multijoueur) nous avons su nous réorganiser et nous mettre au travail en ayant une idée plus claire de nos idées.

Pour conclure, nous sommes satisfaits du contenu que nous rendons à la date de ce rendu intermédiaire.