Le Mans Université

Licence Informatique *2ème année*

Rapport de projet
The Hive

Anaïs Mottier
Mathilde Mottay
Clément Mainguy
Moustapha Tsamarayev

Superviseur : Azzeddine Benabbou
Groupe : TD2 - TP4

Dépôt Git : `https://github.com/Grayspot/The-Hive`

Année scolaire 2019/2020

Table des matières

1 Introduction **3**

1.1 Principe du jeu . . . 3

2 Organisation du travail **4**

2.1 Identification et priorisation des tâches . . . 4

2.2 Planification des tâches . . . 4

2.2.1 Macro-planning . . . 4

2.2.2 Planning par séance . . . 4

2.3 Répartition des tâches . . . 5

2.4 Outils pour collaborer . . . 6

3 Conception **7**

3.1 Déroulement d'une partie . . . 7

3.1.1 Une nouvelle carte à chaque partie . . . 7

3.1.2 Les paramètres du joueur . . . 7

3.1.3 Les items . . . 7

3.1.4 3 issues possibles . . . 8

3.1.5 Sauvegarde/chargement . . . 8

3.2 Déroulement d'un tour . . . 9

3.2.1 Fouiller les hexagones . . . 9

3.2.2 Gérer son inventaire . . . 9

3.2.3 Gérer son équipement . . . 10

3.2.4 Se déplacer . . . 10

3.2.5 Pêcher . . . 10

3.2.6 Regarder la carte . . . 10

3.2.7 Se reposer et guérir . . . 11

3.2.8 Passer au tour suivant . . . 11

3.3 Système de combat . . . 11

3.4 Quêtes . . . 11

3.4.1 Quête bandits . . . 11

3.4.2 Quête soin . . . 12

3.4.3 Quête recherche . . . 12

3.4.4 Quête montagne . . . 12

3.4.5 Quête frontière . . . 12

3.4.6 Quête bunker . . . 12

4 Développement **13**
4.1 Fichiers externes pour le système de sauvegarde/chargement 13
4.2 Génération de la carte 13
4.3 Mise en oeuvre inventaire et équipement 13
4.4 Mise en oeuvre récupération et fouille des items 14
4.5 Fonctionnement du système de combat 14
4.5.1 Calcul des dommages 14
4.5.2 Comportement de l'ennemi pendant le combat 14
4.6 Codage des quêtes 16
4.7 Difficultés développement interface graphique SDL 16
4.7.1 Complexité de la réalisation 17
4.7.2 Manque d'anticipation 17
4.8 Outils 17

5 Informations complémentaires **18**
5.1 Commandes makefile 18
5.2 Mode demo 18
5.3 Annexes 18

6 Conclusion **19**
6.1 Résultats 19
6.2 Bilan du projet 19

1 Introduction

Ce document a pour objet de présenter le travail réalisé dans le cadre du module 174UP02 **Conduite de projet** de la formation de licence 2 Informatique de l'université du Mans.

Nous avions à élaborer un jeu, les objectifs étant notamment de créer et mettre en oeuvre des algorithmes, gérer un projet et mettre en place les outils de développement vus en cours.

1.1 Principe du jeu

The Hive est un jeu de survie. Le joueur est un survivant dans un monde post-apocalyptique ravagé par une guerre nucléaire. Situé dans une zone contaminée, il doit survivre dans ce monde hostile et trouver une des sorties le plus vite possible dans un nombre de tour imparti.

Au début de son aventure, le joueur n'a ni ressources, ni équipement. Pour survivre et espérer trouver une sortie, il doit explorer les prairies, montagnes, villes, lacs... des environs. En fouillant la région, il devrait collecter certains items utiles à sa survie : des objets divers (carte, kit médical...), de la nourriture mais aussi des armes et des armures. En effet, il risque de faire des rencontres plus ou moins amicales qui mèneront parfois au combat.

2 Organisation du travail

2.1 Identification et priorisation des tâches

Notre groupe se réunit pour la première séance du projet afin d'établir une liste des règles du jeu, identifier les fonctionnalités à incorporer et discuter de l'aspect visuel de la première version du jeu. Chacun donne ses propositions et nous essayons de trouver ensemble des compromis sur nos désaccords.

Les fonctionnalités du jeu sont ensuite priorisées selon leurs importances et en suivant une certaine logique : il est préférable par exemple de terminer la génération aléatoire de la carte et s'assurer qu'elle soit fonctionnelle avant de commencer le codage du déplacement du joueur. Cette étape de priorisation des tâches est particulièrement utile pour planifier le projet à long terme.

2.2 Planification des tâches

Pour organiser notre travail, nous utilisons un **macro planning prévisionnel** et un **planning par séance**.

2.2.1 Macro-planning

Le macro-planning prévisionnel est un diagramme de Gantt proposant une vue générale sur plusieurs séances avec les grands intitulés de tâches. Pour chaque tâche figure les prévisions de dates de début/fin, du nombre de séances et une estimation en pourcentage de réalisation de la tâche (mise à jour au fil des séances).

Lien vers le macro-planning :
`https://docs.google.com/spreadsheets/d/1_34T7yRQEfpWN6XVMSUa7jgUmXFABmIoQxDsxnfsyBM/edit?usp=sharing`

2.2.2 Planning par séance

En début de séance, chaque membre du groupe se fixe des objectifs personnels à atteindre. Le planning par séance est plus détaillé que le macro-planning car il correspond aux prévisions des sous-tâches (pour le système de combat, coder le calcul des dommages, le comportement de l'ennemi. . .sont des sous-tâches). Dans le macro-planning, un nombre de séances est prévu pour réaliser une tâche, ce qui permet à chacun, en parallèle, d'organiser les séances selon son rythme, sa méthode pour finir le travail en temps voulu.

A la fin de chaque séance, le planning est complété pour indiquer ce qui a été réalisé, ce qui a du retard. Le travail non-fait est soit reporté, soit terminé pendant le temps libre avant la prochaine séance.

L'outil **Gitkraken Glo** est utilisé pour le planning par séance. Il permet d'assigner des personnes aux sous-tâches, de compléter facilement la progression et d'étiqueter pour une meilleure lisibilité.

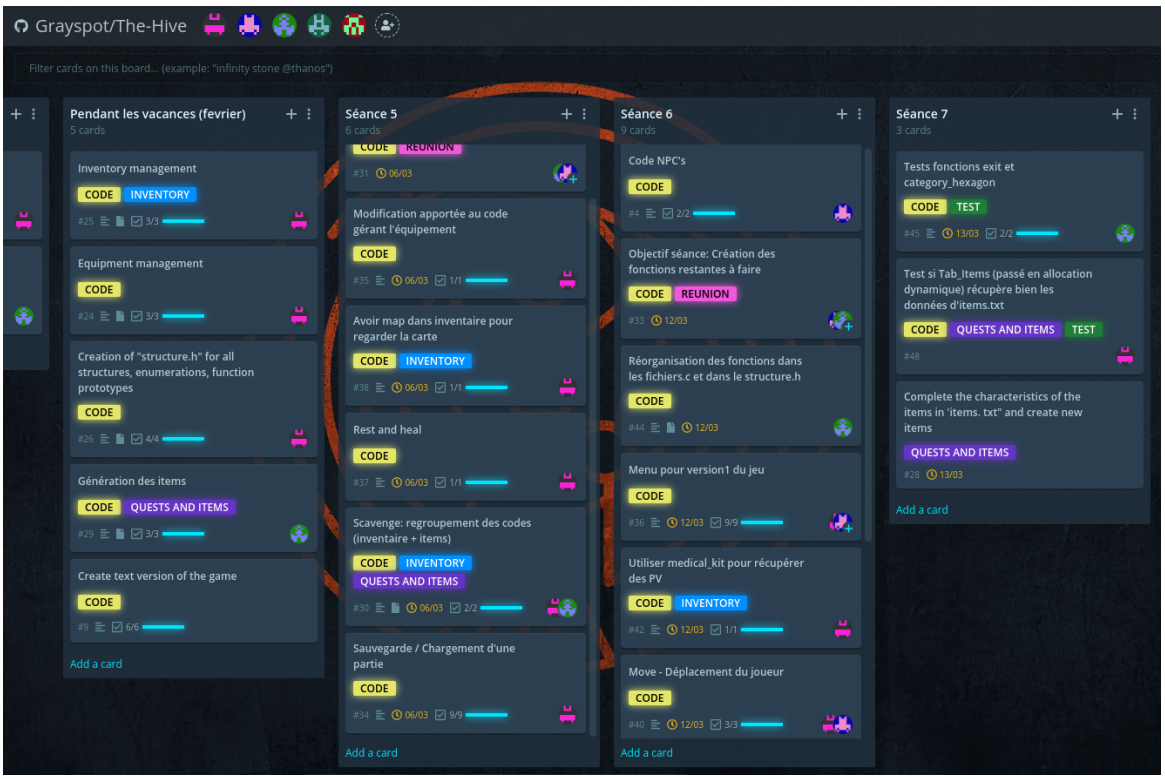

FIGURE 1 – Gitkraken Glo : Planning par séance

2.3 Répartition des tâches

La répartition des tâches, présentée dans le tableau ci-dessous, se déroule de manière équitable (en temps et difficulté) et selon les préférences de chacun. Il est convenu dès le début du projet que Clément s'investisse entièrement à l'interface graphique SDL du jeu.

TABLE 1 – Répartition des tâches

Description des tâches	**Membres du groupe**
Création des items	Groupe (suivant les besoins de chacun)
Génération aléatoire de la carte	Moustapha Tsamarayev
Génération aléatoire des items sur la carte	Anaïs Mottier Mathilde Mottay
Déplacement du joueur	Moustapha Tsamarayev
Gestion des caractéristiques du joueur	Anaïs Mottier Mathilde Mottay
Gestion inventaire et équipement joueur	Mathilde Mottay
Écriture et codage des quêtes	Anaïs Mottier Moustapha Tsamarayev
Système de combat	Moustapha Tsamarayev Clément Mainguy Mathilde Mottay (récupération des items)
Gestion tour suivant	Moustapha Tsamarayev
Fonctionnalités mineures	Mathilde Mottay (pêcher, se reposer et guérir) Anaïs Mottier (aide pour le joueur)
Interface Graphique (SDL)	Clément Mainguy
Système de sauvegarde et chargement	Mathilde Mottay
Tests unitaires	Anaïs Mottier Mathilde Mottay

2.4 Outils pour collaborer

Pour collaborer sur ce projet, plusieurs outils sont utilisés :

— **GitHub** pour héberger et partager le code source

— **Gitkraken** pour intégrer les modifications réalisées par chaque membre du groupe

— **Discord** pour communiquer régulièrement sur l'avancée du projet et s'entraider. Cet outil s'avère très utile pendant la période de confinement pour continuer de travailler au mieux ensemble à distance.

3 Conception

3.1 Déroulement d'une partie

Une partie de The Hive dure au **maximum 15 tours**. Pour rappel, l'objectif du joueur est de rester en vie et trouver une sortie le plus vite possible.

Remarque : Dans le cas où le joueur commence une nouvelle partie, une présentation des règles du jeu s'affiche à l'écran.

3.1.1 Une nouvelle carte à chaque partie

La carte est constituée de différentes zones appelées "hexagones" (en référence à l'interface graphique SDL de la carte). La dimension de la carte est fixée à 15 x 15, soit un total de 225 hexagones. Générée aléatoirement à chaque partie, seules les bordures (mer, wasteland, frontière et montagne) restent inchangées pour des raisons de bon-sens.

12 types d'hexagone sont répartis en 4 catégories : **nature** (prairie, forêt, lac, montagne, mer), **urbain** (ville, marché, favela), **militaire** (camp militaire) et **autre** (camp des bandits, frontière, wasteland).

3.1.2 Les paramètres du joueur

Le joueur commence la partie avec un sac à dos (inventaire) vide et sans équipement (armes/armures). Ses points de vie sont initialisés à 100, tout comme ses points d'énergie, et il a 5 points d'action. Ce sont les valeurs maximales pour ces paramètres. Il dispose de 15 tours pour s'enfuir.

A quoi servent ces paramètres ?

- **Points de vie (PV)** : Ces points permettent au joueur de rester en vie. Il peut perdre ses points de vie lors de combats et/ou au cours de certaines quêtes.
- **Points d'énergie (PE)** : Plus un joueur a de points d'énergie, plus il récupère après chaque tour de points d'action. Se reposer/guérir ou manger/boire un item lui permet de récupérer des points d'énergie.
- **Points d'action (PA)** : Les points d'action permettent au joueur de se déplacer sur la carte. Le joueur récupère des points d'action après chaque tour.

Le joueur est positionné aléatoirement sur la carte, mais il ne peut pas commencer la partie par une quête ou un combat.

3.1.3 Les items

Il existe plusieurs types d'items :

Table 2 – Items créés selon leur type

Type d'items	**Items créés**
Armes	pistolet, gros bâton, fusil, couteau, arc et flèches, batte de baseball
Armures	gilet pare-balles, casque
Objets divers	canne à pêche, kit médical, corde, bâton de marche, carte
Nourriture	fruits, poisson, boîte de conserve, soda, boisson énergétique, bouteille d'eau

Chaque item a une probabilité d'être trouvé sur une zone nature (lac, forêt, prairie...), urbaine (ville, favela, marché...) ou militaire (camp militaire).

Chaque item existe dans un but précis :

- Les items de type nourriture permettent au joueur de récupérer de l'énergie. Chaque aliment ou boisson a une valeur énergétique différente.
- Si le joueur est en possession de l'item carte, il peut accéder dans le menu principal à une vue générale des environs. Sans cet item, le joueur ne peut voir que les 8 hexagones qui l'entourent pour se déplacer.
- Si le joueur possède une canne à pêche, il pourra dans les hexagones de type lac ou mer pêcher et peut-être attraper un poisson.
- Les items corde et bâton de marche sont utilisés dans la quête montagne. La montagne est une des sorties possibles. Le joueur aura plus de chance de réussir à la gravir avec ces items dans son inventaire.
- Le kit médical permet au joueur de récupérer des points de vie s'il a besoin de soins ou d'aider un homme dans la quête soin.
- Les armes et les armures permettent au joueur de s'équiper pour avoir plus de chance de sortir vainqueur de combats ou lors de la quête des bandits par exemple.

3.1.4 3 issues possibles

Le jeu peut se terminer de 3 façons différentes :

1. Le joueur réussit à trouver une des 3 sorties possibles et s'enfuit de la zone :
   - Il réussit à obtenir le pass, trouver le bunker et s'y réfugie.
   - Il réussit à franchir la frontière.
   - Il réussit à gravir la montagne pour passer de l'autre côté.
2. Le joueur meurt pendant un combat ou lors d'une quête.
3. Passé 15 tours, si le joueur n'a pas trouvé de sortie, il reste bloqué et est promis à une mort certaine.

3.1.5 Sauvegarde/chargement

Le système de sauvegarde/chargement du jeu donne la possibilité au(x) joueur(s) de jouer jusqu'à 3 parties nommées, avec possibilité d'effacer une partie. Le joueur a la possibilité de sauvegarder sa progression à tout moment (sauf pendant combat et quêtes).

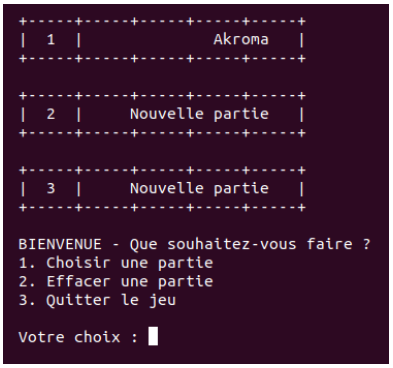

FIGURE 2 – 3 parties

3.2 Déroulement d'un tour

A chaque tour, le joueur voit ses caractéristiques avec sa localisation et le nombre de tours qu'il lui reste ainsi qu'un menu principal où plusieurs choix s'offrent à lui.

FIGURE 3 – Menu principal par tour

3.2.1 Fouiller les hexagones

Pour trouver des ressources, le joueur peut choisir de fouiller les hexagones. 0 à 5 items sont générés, aléatoirement, selon leur probabilité d'apparition pour la catégorie de l'hexagone inspecté. Le joueur doit faire preuve de déduction et stratégie pour essayer de trouver certains items en particulier. Un hexagone ne peut être fouillé qu'une seule fois et les hexagones de type frontière et wasteland sont toujours vides.

Le joueur a le choix d'ajouter ou non le ou les item(s) trouvé(s) à son inventaire. Si son inventaire est déjà plein, le joueur a la possibilité d'échanger l'item trouvé avec un de son inventaire.

3.2.2 Gérer son inventaire

L'inventaire du joueur a une capacité maximale de 10 items et ne peut contenir plus de 2 fois le même item.

Le joueur peut gérer son inventaire comme il le souhaite :

- En savoir plus sur un item pour connaître ses caractéristiques
- Se débarrasser d'un de ses items
- Manger ou boire un item (s'il a au moins un item de type nourriture)
- Utiliser le kit médical pour se soigner (s'il en a un)

3.2.3 Gérer son équipement

Le joueur peut porter une arme à sa main gauche et/ou droite (fusil, couteau, arc et flèches, batte de baseball...), une armure sur sa tête (casque) et/ou son corps (gilet pare-balles).

Il peut gérer son équipement comme il veut :

— Retirer un équipement (s'il porte au moins un équipement)

— S'équiper d'un item (s'il a dans son inventaire, au moins, un item de type armure ou arme dont il n'est pas déjà équipé). S'il souhaite s'équiper d'un item qui se porte sur une zone de son corps déjà équipée, un échange est proposé.

3.2.4 Se déplacer

Pour explorer les environs, le joueur peut se déplacer sur la carte. Il peut voir les différents types d'hexagones qui l'entourent, dont les coordonnées sont valides, et choisir sur lequel il souhaite aller. Il doit cependant avoir le nombre de points d'action nécessaire pour pouvoir se déplacer dans un hexagone en particulier. Certaines zones requièrent plus de points d'action que d'autres. Par exemple, se déplacer vers une montagne nécessite 3 points d'action, tandis que se déplacer vers une prairie n'en nécessite qu'un. L'arrivée du joueur sur un hexagone peut déclencher un combat ou une quête.

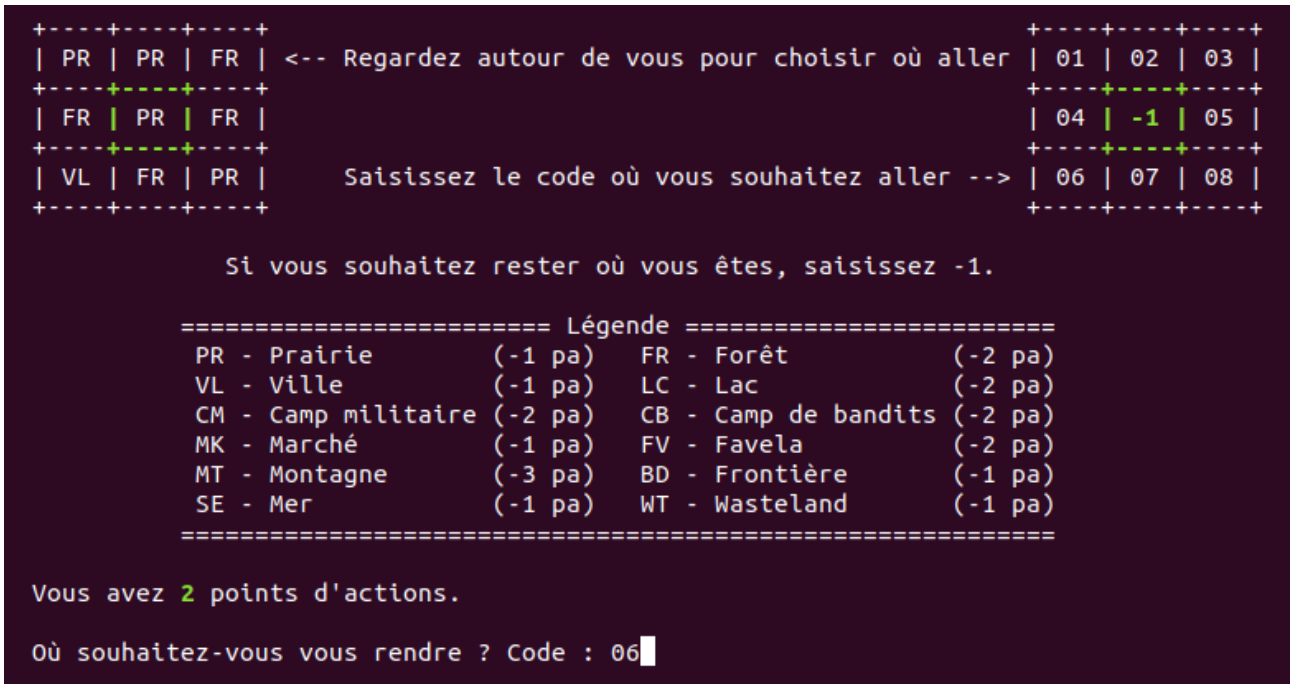

FIGURE 4 – Le joueur souhaite se déplacer dans une ville.

3.2.5 Pêcher

Si le joueur dispose d'une canne à pêche dans son inventaire et se situe sur un hexagone de type lac ou mer, il peut pêcher et peut-être attraper un poisson.

3.2.6 Regarder la carte

Si le joueur a une carte dans son inventaire, il peut regarder la carte entière.

3.2.7 Se reposer et guérir

Le joueur peut choisir de se reposer et guérir pour récupérer des points de vie et d'énergie. Il passe directement au tour suivant.

3.2.8 Passer au tour suivant

Le joueur décide lui-même de passer au tour suivant (souvent par manque de points d'action pour se déplacer). Dans ce cas, il perd au maximum 15 points d'énergie et récupère des points d'action. Le nombre de points d'action récupéré dépend du nombre de points d'énergie du joueur. Plus il a de l'énergie, plus son action augmente.

3.3 Système de combat

Les combats sont positionnés aléatoirement sur la carte. Lors de son arrivée sur certains hexagones, le joueur peut tomber sur des maraudeurs. Le joueur a la possibilité d'accepter ou non de combattre. Cependant, s'il n'a aucune arme sur lui, il prend d'office la fuite.

Les maraudeurs sont générés aléatoirement. Ils peuvent être de plusieurs classes (léger, moyen ou lourd) et ne sont pas toujours équipés de la même arme ou armure. Le tableau ci-dessous présente les différentes combinaisons possibles avec leur pourcentage d'apparition.

Table 3 – Caractéristiques des différents types de maraudeurs possibles

Classe	Léger (65%)	Moyen (30%)	Lourd (5%)
PV	40	80	120
Arme	Batte de baseball (70%) Couteau (30%)	Batte de baseball (70%) Pistolet(30%)	Batte de baseball (55%) Pistolet (40%) Fusil (5%)
Armure	Pas d'armure	Gilet pare-balles (si batte de baseball comme arme)	Gilet pare-balles (100%)

Lors du combat, le joueur peut se rapprocher de l'ennemi, s'en éloigner (dans les limites du champ de bataille), se défendre, attaquer avec une arme ou essayer de fuir. Si le joueur réussit à prendre la fuite, il est déplacé automatiquement sur un des hexagones qui l'entourent.

L'ennemi prend ses décisions intelligemment, en fonction de la distance qui le sépare du joueur, de sa position sur le champ de bataille, de sa couverture et de celle du joueur.

3.4 Quêtes

En explorant les hexagones de la carte, le joueur peut être amené à réaliser des quêtes qui, pour certaines, fonctionnent en parallèle. A chaque partie, le joueur ne peut les jouer qu'une seule fois et leurs coordonnées sur la carte changent. Certaines quêtes offrent la possibilité au joueur de s'enfuir de la zone. Cependant, en fonction des décisions qu'il prend, les quêtes peuvent également le conduire à la mort.

3.4.1 Quête bandits

Le joueur arrive au camp des bandits. Le lieu n'est pas très accueillant, il n'y a personne. En se rapprochant, le joueur observe des objets volés, des armes...Des voix se rapprochent...Il a le choix entre partir au plus vite (déplacement sur un autre hexagone qui l'entoure), attendre

le retour des bandits ou voler des objets. Dans les deux derniers cas, le joueur prend le risque de combattre les bandits et de mourir s'il n'est pas bien équipé.

3.4.2 Quête soin

La quête soin se déroule hors des bordures de la carte (mais pas sur un lac). Le joueur rencontre une personne blessée, il a le choix entre l'ignorer, lui voler ce qu'elle possède ou lui apporter de l'aide. S'il possède un kit médical ou de la nourriture, il réussit à aider l'homme qui le remercie en lui donnant un pass pour ouvrir le bunker.

3.4.3 Quête recherche

Le joueur croise un homme qui connaît bien les environs et lui propose de le mettre en sécurité, en échange d'un service. L'homme demande au joueur de lui trouver un objet en particulier. Deux possibilités s'offrent à lui, accepter ou refuser. Si le joueur accepte :

— il a l'item, il choisit de lui donner ou non.

— il n'a pas l'item, il doit partir à sa recherche et le lui ramener. Si le joueur réussit à apporter l'item à l'homme, il obtient comme convenu les coordonnées du bunker.

Remarque : Une aide s'affiche tout au long du déroulement de cette quête dans le menu principal. Le joueur peut continuer à jouer normalement, même s'il n'a pas fini la quête.

3.4.4 Quête montagne

Située dans une montagne, cette quête propose une sortie de jeu. Il faut de l'énergie, du courage et surtout le matériel adapté pour la gravir (corde et bâton de marche). Le joueur a la possibilité de tenter son ascension. S'il refuse ou change d'avis pendant la montée, il redescend sur terre et continue le jeu. En revanche, s'il accepte et persévère, il prend le risque de mourir mais a davantage de chance de réussir s'il est bien équipé.

3.4.5 Quête frontière

Le joueur peut essayer de traverser la frontière. Cependant, elle est protégée par des soldats. Le déroulement de cette quête dépend du parcours du joueur :

— S'il réalise la quête soin et apporte son aide à l'homme blessé, il obtient un laissez-passer sans problème. Dans le cas, où il n'a pas réussi à apporter de l'aide, il a 70% de chance de franchir la frontière.

— S'il ne réalise pas la quête soin mais possède un pass, il a 50% de chance de s'en sortir. Sinon, seulement 15% de chance.

3.4.6 Quête bunker

La quête bunker est également une quête de sortie. Elle se déroule hors des bordures de la carte (mais pas sur un lac). Le joueur découvre un bunker isolé, il a besoin du pass pour s'y réfugier.

4 Développement

4.1 Fichiers externes pour le système de sauvegarde/chargement

Chaque partie sauvegardée correspond à un fichier externe dont le nom dépend du numéro de la partie jouée : **sauv1.csv**, **sauv2.csv** et **sauv3.csv**. Ces fichiers sont dans le dossier sauv et sont créés quand le joueur commence une nouvelle partie. Dans le cas d'une simple sauvegarde de la progression, le contenu du fichier est mis à jour. La fonction remove est utilisée pour supprimer un fichier si le joueur décide d'effacer une partie.

4.2 Génération de la carte

La carte du jeu est représentée par une **matrice de type cell_t** et de dimension 15x15. La structure cell_t correspond à un hexagone et contient son type (énumération hex_t), sa catégorie (énumération categ_hexa), un indicateur de combat, un indicateur de quête et un dernier indicateur permettant de savoir si l'hexagone a été fouillé par le joueur.

Algorithme génération de la carte :

1. La carte est initialisée par des hexagones de type prairie (catégorie nature), sans combat, sans quête et sans hexagones fouillés.
2. Les contours de la carte sont initialisés : mer, wasteland, montagne et frontière.
3. Un nombre déterminé d'hexagones indispensables pour la carte est créé (comme camp militaire/bandit, marché, ville, favela, lac par exemple) sur des coordonnées aléatoires.
4. Pour assurer une hétérogénéité des types d'hexagones dans la carte, la matrice est parcourue. Pour chaque case, le nombre d'hexagones-voisins similaires est compté. Selon ce nombre, après des calculs de probabilité, l'hexagone courant est modifié ou non.
5. Les combats et les quêtes sont positionnés aléatoirement.
   Néanmoins, le positionnement des quêtes respecte certaines contraintes :
   - Quête montagne : Hexagone de type montagne
   - Quête frontière : Hexagone de type frontière
   - Quête bunker : Hexagone hors bordure de catégorie nature sauf lac
   - Quête bandits : Hexagone de type camp des bandits
   - Quête soin : Hexagone hors bordure - ni lac, ni camp des bandits
   - Quête recherche : Hexagone hors bordure - ni urbain, ni camp des bandits

   Remarque : Toutes les quêtes se situent sur des hexagones différents.

4.3 Mise en oeuvre inventaire et équipement

L'inventaire est un tableau dans la structure perso_t correspondant au joueur. Pour y apporter des changements, les principales fonctions add_item_to_inventory (ajout d'un item) et delete_item_in_inventory (suppression d'un item) utilisent l'attribut index des items (champ dans la structure item_t), c'est-à-dire la position des items dans l'inventaire. Le champ nb_items_inventory de la structure perso_t est également mis à jour lors de ces modifications.

La structure perso_t représentant le joueur est aussi constituée de 4 pointeurs sur des objets de type item_t : un pointeur sur sa main gauche, sa main droite, son corps et sa tête. Les items pointés représentent donc l'équipement du joueur et sont mis à NULL, s'ils ne pointent sur aucun équipement. Chaque item a un attribut nommé equipable permettant de savoir où

il peut être porté et ainsi, équiper un item au bon endroit. Les fonctions gérant l'équipement utilisent aussi l'index des items pour s'assurer de retirer ou équiper le bon item, en cas de double dans l'inventaire.

4.4 Mise en oeuvre récupération et fouille des items

Le fichier externe **items.csv** (dossier data) contient tous les items du jeu avec leurs caractéristiques. Pour être utilisés dans le programme, ils sont extraits grâce à la fonction creation_tab_item qui stocke le contenu du fichier dans un tableau item_t Tab_Items, alloué dynamiquement.

La structure item_t est constituée de 3 champs : pc_nature, pc_urbain et pc_military. Chacun représente respectivement la probabilité, en pourcentage, de trouver l'item dans un hexagone de catégorie nature, urbain et militaire. On utilise ensuite la fonction rng qui génère 0 ou 1 en fonction de la probabilité, en pourcentage, passée en paramètre. C'est ainsi qu'on met en oeuvre aléatoirement la fouille des items sur les hexagones.

4.5 Fonctionnement du système de combat

4.5.1 Calcul des dommages

Le tableau d'entiers attack[3], dans la structure item_t, représente la valeur d'attaque d'une arme selon la distance.

Exemple pour le pistolet :

- attack[0] = 5 : Si la distance entre le joueur et l'ennemi vaut 0, la valeur d'attaque est de 5.
- attack[1] = 25 : Si la distance entre le joueur et l'ennemi vaut 1, la valeur d'attaque est de 25.
- attack[2] = 15 : Si la distance entre le joueur et l'ennemi vaut 2, la valeur d'attaque est de 15.

Le tableau d'entiers hitchance[3], dans la structure item_t, représente la probabilité, en pourcentage, d'une arme à toucher l'ennemi selon la distance et fonctionne de la même manière.

La fonction damage_calculator utilise ces 2 valeurs, mais aussi la distance entre le joueur et l'ennemi, la couverture du non-attaquant et la valeur de défense de son armure (s'il en porte une) pour calculer les dommages causés par l'attaquant.

4.5.2 Comportement de l'ennemi pendant le combat

Pendant le combat, le joueur joue contre l'ordinateur. Ce dernier ne prend pas ses décisions au hasard. Il est stratégique et se comporte différemment selon l'arme qu'il porte, la distance qui le sépare du joueur, sa position dans le champ de bataille et ses points de vie.

Algorithme :

$\Longrightarrow$ **Si l'arme de l'ennemi est un fusil**

- Si distance supérieure à 2 :
  - L'ennemi avance vers le joueur
- Si distance égale à 0 :
  - 60% de chance (si sa position le permet) que l'ennemi s'éloigne du joueur
  - Sinon, 30% de chance que l'ennemi se défende
  - Sinon, l'ennemi essaie d'attaquer
- Si distance égale à 1 :
  - Si PV ennemi supérieurs à 50, l'ennemi essaie d'attaquer
  - Sinon, 50% de chance que l'ennemi se défende
  - Sinon, l'ennemi essaie d'attaquer
- Si distance égale à 2 :
  - 40% de chance que l'ennemi avance vers le joueur
  - Sinon, 30% de chance que l'ennemi se défende
  - Sinon, l'ennemi essaie d'attaquer

$\Longrightarrow$ **Si l'arme de l'ennemi est un pistolet**

- Si distance supérieure à 2 :
  - L'ennemi avance vers le joueur
- Si distance égale à 0 :
  - 80% de chance (si sa position le permet) que l'ennemi s'éloigne du joueur
  - Sinon, 30% de chance que l'ennemi se défende
  - Sinon, l'ennemi essaie d'attaquer
- Si distance égale à 1 :
  - 40% de chance (si sa position le permet) que l'ennemi s'éloigne du joueur
  - Sinon, 30% de chance que l'ennemi se défende
  - Sinon, l'ennemi essaie d'attaquer
- Si distance égale à 2 :
  - Si PV ennemi > 50, l'ennemi essaie d'attaquer
  - Sinon, 50% de chance que l'ennemi se défende
  - Sinon, l'ennemi essaie d'attaquer

$\Longrightarrow$ **Si l'arme de l'ennemi est une arme corps à corps (couteau, batte de baseball...)**

- Si distance supérieure à 0 :
  - 80% de chance que l'ennemi avance vers le joueur
  - Sinon, 20% de chance que l'ennemi se défende
- Si distance égale à 0 :
  - L'ennemi essaie d'attaquer

4.6 Codage des quêtes

Chaque quête possède une fonction principale, les plus complexes font appel à d'autres fonctions internes. Les quêtes sont représentées dans une structure de type quete_t. Cette structure contient pour chaque quête un indicateur représentant l'avancement de la quête, sauf pour la quête recherche. En effet, celle-ci n'est pas représentée par un entier, mais par une autre structure de type search_t. La structure search_t contient un indicateur d'avancement, un item (item_t), un indicateur de découverte de l'item, la position X du bunker et la position Y de celui-ci. Au début de la partie, les entiers de quete_t sont initialisés à -1, signifiant "non activé".

Remarque : chaque quête ne peut être jouée entièrement qu'une seule fois par partie.

Algorithme général des quêtes :

1. Description de la situation liée à la quête en cours et initialisation de son indicateur d'avancement avec la valeur 0 (signifiant "en cours").
2. Le joueur décide de la suite de la quête, en fonction des choix qui lui sont proposés. Il a le choix entre jouer la quête, de différentes manières ou d'une seule, ou de ne pas la jouer.
3. Déroulement de la quête en fonction du choix fait précédemment.

   $\Longrightarrow$ Si le joueur décide de continuer la quête, alors elle se poursuit de la manière dont il l'a choisie. Au fil de son avancée, le joueur peut être amené à faire de nouveaux choix :

   — demande de confirmation du choix précédent à la vue des derniers évènements.

   — demande de nouveaux choix pour la suite de la quête.

   Si le joueur finit la quête, l'indicateur d'avancement est mis à 1 (signifiant "terminé"), sinon il reste à 0.

   $\Longrightarrow$ Si le joueur ne souhaite pas jouer la quête, le jeu reprend là où il en était.

Remarque : l'indicateur de la quête soin peut être égal à 1, 2 ou 3, en fonction des choix du joueur pendant cette quête. Cet indicateur joue un rôle dans la quête frontière.

La quête recherche est un peu différente, mais le principe de l'algorithme reste le même. Elle se déroule en deux temps :

1. Découverte de la quête, le joueur accepte ou non de la jouer. Il doit récupérer un item, s'il l'accepte, on lui donne les instructions à suivre. Le jeu reprend son cours, mais le joueur garde en tête l'objectif de trouver l'item.
2. L'item est trouvé, il rejoint l'homme sur le lieu où il l'a rencontré et le lui donne.

4.7 Difficultés développement interface graphique SDL

Le développement de l'interface graphique en SDL2 a débuté plus tard que les autres fonctionnalités. En effet, nous souhaitions établir certaines fonctionnalités comme la génération de la carte avant de commencer le développement de son interface graphique.

Deux problèmes majeurs ont été rencontrés lors de la réalisation de l'interface graphique : la complexité de la réalisation et le manque d'anticipation.

4.7.1 Complexité de la réalisation

N'ayant jamais utilisé les bibliothèques SDL2, il y a une période d'adaptation pour la prise en main et l'affichage graphique, ne serait-ce qu'un simple bouton. Ce travail nécessite beaucoup de recherche pour apprendre à mettre en oeuvre l'interface imaginée pendant la phase de conception. Ces difficultés, imprévues, ont entraîné un retard dans l'organisation et impactent le travail final.

4.7.2 Manque d'anticipation

Nous n'avions pas anticipé que le code permettant de jouer au jeu dans la console et celui gérant l'interface graphique soient difficilement compatibles. Le rassemblement des deux codes est compliqué, notamment à cause du retard des fonctions de bases de la partie interface graphique. Ce problème n'est pas encore résolu. Il est impossible, pour le moment, de proposer au joueur beaucoup de contenu en interface graphique. La gestion du menu principal et des quêtes n'existe qu'en version console.

FIGURE 5 – Affichage de la carte en SDL

4.8 Outils

Différents outils sont utilisés tout au long du développement du projet :

- **gdb** pour déboguer (cf. Annexe 1)
- **valgrind** pour vérifier les éventuelles fuites de mémoire (cf. Annexe 2)
- **jeux de tests** pour effectuer des tests unitaires (cf. Annexe 3)
- **Doxygen** pour créer une documentation de tout le code du jeu
- **makefile** pour la compilation séparée
- **LaTeX** pour concevoir ce rapport de projet

5 Informations complémentaires

5.1 Commandes makefile

make all : compiler le jeu et les tests
make clean : effacer les fichiers objets
make remove : effacer les fichiers objets et exécutables
make docs : générer la documentation Doxygen et l'afficher directement dans le moteur de recherche

5.2 Mode demo

Il existe un mode demo qui permet de choisir les items souhaités. Ce mode est pratique pour tester certaines fonctionnalités sans avoir à chercher les items.
Exécuter **./game –demo** ou **./game -d**

5.3 Annexes

Annexe 1 - Débogage avec gdb

Annexe 2 - Utilisation outil valgrind

Annexe 3 - Jeux de tests
Remarque : Les fichiers test sont dans le dossier src et commencent toujours par "test_*.c". Ils sont générés lors de la compilation dans le dossier bin avec le jeu.

6 Conclusion

6.1 Résultats

Les fonctionnalités principales du jeu, décrites dans la partie conception de ce rapport, sont toutes réalisées. Néanmoins, en raison des difficultés mentionnées précédemment, l'interface graphique du projet n'est pas satisfaisante. Le jeu fonctionne correctement quand il est exécuté dans la console, mais l'incorporation du code SDL perturbe son bon déroulement.

Le planning prévisionnel a fait l'objet de modifications pendant la durée du projet, mais a été dans l'ensemble respecté. Certaines tâches ont nécessité plus de temps que prévu (système de combat, quêtes...), les imprévus dans le développement de l'interface graphique, ainsi que quelques problèmes de connexion pendant le confinement, ont conduit à décaler et réaménager au mieux certaines séances de travail.

Si l'on disposait davantage de temps, le problème avec la SDL serait réglé en priorité. On développerait, à nouveau, plusieurs fonctions afin que la gestion d'une partie via la console et via l'interface graphique soient semblables.

Certaines améliorations seraient ensuite envisagées :

- Proposer au joueur, en début de partie, de choisir une compétence spéciale qui apporterait des avantages. Par exemple, la compétence "scout" qui lui permettrait d'avoir besoin de moins de points d'action pour se déplacer, ou encore, la compétence "bon métabolisme" qui pourrait lui permettre de récupérer plus de points d'énergie en mangeant/buvant...
- Proposer en début de partie au joueur de choisir entre différentes tailles de carte, correspondant à différents niveaux de difficultés.
- Créer des quêtes plus interactives et difficiles à résoudre.
- Créer une interface graphique pour l'inventaire, l'équipement, le déplacement du joueur, un menu interactif et des animations pour les quêtes.

6.2 Bilan du projet

N'ayant pas choisi un sujet proposé par les professeurs en début de module, créer The Hive de toute pièce a permis de stimuler notre créativité dans l'élaboration des règles du jeu, des fonctionnalités, des quêtes, de l'univers...

Nous avons aussi amélioré nos compétences techniques en développant des algorithmes et des fonctions pour les différentes fonctionnalités. Réaliser les tests d'un algorithme ou d'une fonction avant de l'intégrer dans le reste du code est aussi devenu un réflexe.

Nous aurions aimé proposer une version du jeu complète et fonctionnelle avec l'interface graphique. Néanmoins, les difficultés rencontrées dans sa mise en oeuvre nous ont appris à ne pas nous reposer uniquement sur nos compétences. Novices dans l'utilisation de la SDL2 et de ses bibliothèques, la réalisation de cette tâche a nécessité beaucoup de recherche avant d'obtenir un résultat. La durée prévue à été sous-estimée, entraînant du retard. L'anticipation et la planification n'ont donc pas été au rendez-vous pour cette partie du projet.

Ce premier projet en informatique nous a aussi appris à développer notre capacité à travailler en groupe. La cohésion entre les membres du groupe est bonne avec un esprit d'entraide.