Rapport de projet

Projet Labyrinthe

Présentation de ce qui a été réalisé au cours de notre projet de réalisation d'un mini jeu de labyrinthe.

Boisson Léo
Bussereau Keryann
Ciron Fabien

18/12/2015

Table des matières

Table des matières 1

I. Introduction 3

II. Analyse du problème 4

A. Énoncé 4

B. Compréhension 4

C. Mise en application choisie 4

III. Organisation du travail 6

IV. Développement 7

A. Le Labyrinthe 7

1. Première tentative 7

2. Seconde tentative 7

3. Troisième tentative 7

4. Le lissage 8

5. Les couleurs 8

6. Les statistiques 8

7. La génération de départ de la nourriture 9

B. Les insectes 10

1. Génération des insectes 10

2. Modification de la structure d'un insecte ainsi que de sa position 10

3. Suppression des insectes 10

C. Les déplacements et interactions entre les insectes 11

1. Déplacements aléatoires 11

2. Déplacements contrôlés 11

3. Interactions entre les insectes 11

4. Découverte du labyrinthe 12

D. L'interface utilisateur tour par tour 13

1. Sauvegarde/Chargement 13

2. Menu et interface avec l'utilisateur 13

E. Le menu principal 14
1. Menu et règles du jeu 14
2. Lancement du jeu 14
V. Résultats 15
VI. Conclusion 16
VII. Annexes 17
A. Annexe 1 : Le labyrinthe 17
B. Annexe 2 : Interface avec l'utilisateur 21

I. Introduction

Notre sujet se nommait « La vie dans un labyrinthe » et avait comme but de faire évoluer des insectes dans un labyrinthe Générer / Sauvegarder un labyrinthe, mettre des insectes.

Nous avions comme objectif de réaliser un labyrinthe de type jeu de la vie, nous avons donc décidé de rester assez proche du sujet en gardant l'idée d'insectes, ces insectes se déplacent dans un labyrinthe en étant attirés par la nourriture.

Nous avons convenu de deux manières de finir une partie : soit de découvrir l'ensemble des cases vides du labyrinthe, ce qui correspond à la victoire, soit que tous les insectes meurent, ce qui correspond à la défaite. Nous avons aussi réinterprété les règles afin d'en faire un jeu plus interactif en permettant à l'utilisateur d'ajouter de la nourriture ou des insectes à chaque tour.

Au début de la partie, nous avons fait le choix de générer 5 insectes à des endroits aléatoires du labyrinthe. Les insectes supplémentaires sont soit placés par l'utilisateur (zéro ou un, aléatoirement), soit sont le fruit de la reproduction de deux insectes de sexe différents. Les insectes meurent soit à cause de l'âge, soit par inanition, soit en se battant entre eux quand ils sont de même sexe et adjacents.

Pour le déplacement des insectes, nous avons d'abord implémenté un mouvement aléatoire. Après cela, nous avons développé une méthode de déplacement où les insectes se dirigent vers la nourriture la plus proche avant de poser des conditions pour que les deux fonctions interviennent selon que la nourriture est plus ou moins proche d'eux.

La nourriture est placée en début de partie, dix qui sont placées aléatoirement, puis l'utilisateur peut faire le choix d'en placer à chaque tour mais ne pourra en placer qu'entre un et trois, aléatoirement.

Nous avons de plus intégré la possibilité de sauvegarder un labyrinthe et de pouvoir charger des parties précédentes.

Nous avons ajouté des statistiques sur la population d'insecte telle que nous donnons le numéro de l'insecte, son sexe, sa durée de vie restante (âge et nourriture) et son emplacement en x et y.

II. Analyse du problème

A. Énoncé

L'objectif est de créer un labyrinthe contenant des insectes et de la nourriture, dans le style "jeu de la vie". Les insectes doivent se déplacer de manière semi-aléatoire, ceux-ci devant être capables d'assurer leur propre survie. Ils doivent notamment être capable de se diriger vers de la nourriture quand il y en a à proximité. Les règles de vie et de mort des insectes n'ont pas été spécifiées, ainsi que celles gérant la génération de nouveaux insectes. Il faut aussi pouvoir sauvegarder et charger une partie et pouvoir afficher des statistiques sur la population au sein du labyrinthe.

B. Compréhension

La première chose qui nous est venu à l'esprit, c'est qu'un "jeu de la vie" ne nous semblait pas représenter suffisamment d'intérêt pour que quelqu'un ait envie d'y jouer. Nous avons décidé que nous allions laisser à l'utilisateur plus de possibilités d'influer sur la partie en pouvant placer des insectes ou de la nourriture à chaque tour.

Ensuite nous avons réfléchit au labyrinthe, et nous avons compris que nous allions devoir utiliser une matrice mais avons immédiatement écarté l'idée d'une matrice d'entiers car, bien que plus simple, ne nous permettant pas de contenir les informations voulues. Nous nous sommes dit qu'une matrice de structure allait nous être utile pour avoir accès facilement aux informations nécessaires à nos différentes fonctions. Nous avons aussi choisi de "masquer" la partie du labyrinthe qui n'a pas été parcourue par les insectes, le but du jeu devenant de découvrir le labyrinthe avant que tous les insectes meurent.

Nous avons aussi déterminé que les insectes allaient être des structures mais différentes des structures de la matrice afin de simplifier l'initialisation de la matrice tout en ayant toujours les informations sur les insectes que nous voulions.

C. Mise en application choisie

Nous avons d'abord mis en place quelques types énumérés afin de simplifier la compréhension du code, tel que "t_discover" qui sert à savoir si une case a été découverte ou non, "t_sexe" pour connaître le sexe de chaque insecte et "t_etat" afin de savoir dans quel état est chaque case du labyrinthe (une case vide, une case de mur, une case de nourriture ou une case contenant un insecte).

Ensuite, grâce aux types énumérés précédent, nous avons définit la structure "t_lab" qui représente une case du labyrinthe. Cette structure contient "t_discover", "t_etat" et "int insecte" qui est l'indice de l'insecte dans un tableau si l'etat est à insecte.

Puis nous avons définis le type "t_fourmi" qui contient les informations de chaque insecte que sont "t_sexe", "int nourriture" qui contient la quantité de nourriture restante à l'insecte et "int age" qui contient le nombre de tour qu'il reste à vivre à l'insecte.

Toutes les structures ont été définies dans deux fichiers header, on a donc « struct_ins » contenant la structure d'un insecte et « struct_lab » qui contient la structure de chaque case du labyrinthe. Nous n'avons ainsi plus qu'à inclure ces fichiers là où nous en avons besoins, ainsi lors d'un changement dans la structure nous n'avons pas besoin de modifier chaque en tête de nos fichiers c mais uniquement nos headers.

Nous avons aussi défini quelques fonctions qui nous étaient nécessaires. Il nous fallait une fonction pour initialiser le labyrinthe et une autre pour l'afficher. Il nous fallait aussi une fonction pour déterminer le prochain déplacement d'un insecte et une autre pour vérifier si un autre insecte est adjacent pour appeler les interactions. Puis nous avons choisi de faire une fonction régissant la vie des insectes qui contiendrait les règles générales de naissance, de vie et de mort des insectes. Nous avons aussi choisi de faire une fonction pour vérifier si la partie est finie ou non, ainsi que toutes les fonctions d'interface utilisateurs nécessaires.

III. Organisation du travail

Nous avons d'abord pris quelques heures pour analyser le problème avant de commencer à développer. L'intérêt majeur de cette partie était d'avoir une idée précise de ce que nous voulions faire, et être d'accord ensemble des structures que nous allions utiliser ainsi que des règles à utiliser afin de pouvoir travailler dans le même objectif.

Nous avons défini un certain nombre de structures et trois axes majeures du programme se sont démarqués.

Le premier axe a été le labyrinthe, les méthodes de génération et comment le sauvegarder/charger. Nous avons aussi inclus dedans la méthode d'affichage.

Le deuxième axe a été le déplacement des insectes et les méthodes qui pouvaient faire qu'ils n'agissent pas complètement aléatoirement.

Le troisième axe a été les insectes en eux-mêmes, leurs interactions ainsi que les interactions entre l'utilisateur et le jeu.

Nous en avons discuté entre nous puis chacun a fait part aux autres de la partie qui l'intéressait le plus. Nous sommes tombés rapidement d'accord sur quelle partie était à la charge de qui, chacun voulant s'occuper d'une partie différente des autres. Léo a donc décidé de se charger du labyrinthe, Fabien s'est chargé des déplacements pendant que Keryann s'occupait des insectes et des interactions.

IV. Développement

A. Le Labyrinthe

Nous avons convenu d'un labyrinthe en largeur (afin de correspondre au mieux à la taille des écrans sur lesquels nous avons travaillés). Il s'agit du module « insecte.c ». Cette partie a été majoritairement réalisée par Léo sauf pour les statistiques et la génération de la nourriture réalisée par Keryann.

1. Première tentative

Ma première tentative a été de procéder par « îlots », c'est à dire d'initialiser le labyrinthe avec des cases vides, puis de placer aléatoirement des îlots de murs dans le labyrinthe. Le principal problème de ce mode de génération était qu'il créait des zones qui n'étaient raccordés à aucune autre zone. De plus, le labyrinthe manquait de continuité : on pouvait facilement deviner d'où partaient les îlots, ce qui ne donnais pas une apparence très naturelle.

2. Seconde tentative

Ma deuxième tentative a été de procéder par « extrudage ». C'est à dire de remplir complètement le labyrinthe avec des murs, puis de « creuser » des cases vides dedans. Pour commencer, je plaçais la « case courante » (la case que l'on va regarder) tout à gauche, puis je la faisais aller jusqu'à la droite du labyrinthe, de manière pseudo aléatoire (avec une probabilité plus forte qu'elle aille vers la droite que dans les 3 autres directions). Malgré cela, on obtenait tout de même un « couloir », donc pas la carte tortueuse avec des salles que l'on désirait. (v*oir figure 1*)

3. Troisième tentative

Pour la troisième tentative, j'ai décidé de garder la technique par « extrudage », car elle permet au moins d'éviter d'avoir des zones séparées des autres zones. L'idée étant maintenant de placer la « case courante » au centre du labyrinthe, puis de partir de manière aléatoire égale (même probabilité d'aller en haut, en bas, à gauche ou à droite). On arrête « d'extruder » lorsque l'on a enlevé la moitié des murs dans le labyrinthe. Les premiers essais furent plutôt concluant sauf dans un cas : lorsque la « case courante » part complètement d'un côté, et y reste jusqu'à ce que la moitié des cases ai été enlevé de ce côté. Pour résoudre cela, j'ai mis un compteur, qui, à chaque itération, voit si la « case courante » est plutôt à gauche ou à droite. Si elle extrude plus d'un quart du nombre total des cases d'un côté, la case courante est remise au point de départ, soit au milieu du labyrinthe. De plus, on va influencer sur l'orientation des futurs mouvements : si la « case courante » était à gauche du labyrinthe pendant trop d'itérations, elle est remise au milieu, mais désormais, la probabilité qu'elle iras à gauche seras plus faible (de manière proportionnelle, la probabilité qu'elle ira à droite sera plus élevée).

C'est cette version de la génération aléatoire que j'ai décidé de conserver : On obtient bien un labyrinthe complexe, avec des embranchements, des salles, et des îlots. De plus, on a la certitude qu'il n'y a pas de zones isolés du reste dedans *(exemples* : voir *figure 2 et 3*).

4. Le lissage

Afin d'obtenir des labyrinthes plus « arrondies » et moins abrupt, j'ai créé une fonction que j'ai appelée « lissage ». Elle a pour but de parcourir le labyrinthe, et de supprimer les murs trop isolés ainsi que les bords trop escarpés. Pour cela, elle regarde pour chaque mur son nombre de voisin. Si trois cases ou plus autour du mur que l'on est en train de regarder sont vides, alors on passera également la case que l'on regarde en vide. Pour s'assurer que le labyrinthe soit bien « lissé », la fonction renvoie 1 si elle a supprimé des murs, et 0 sinon. Ensuite, lors de la génération du labyrinthe, on utilise cette fonction tant qu'elle revoie 1. J'ai pu constater qu'en moins de 10 itérations, elle lissait le labyrinthe de manière correcte. (*voir figure 4*)

5. Les couleurs

De base, l'affichage était fait avec des caractères : des # pour les murs, un espace pour le vide, un % pour les insectes, et un * pour la nourriture. Nous nous sommes très vite rendu compte que cet affichage n'était pas lisible lorsqu'il y avait beaucoup d'information au même endroit *(voir figure* 5), et nous avons opté comme solution d'utiliser les options de couleur des terminaux ANSI. En effet, il est possible de définir la couleur du caractère que l'on affiche, mais également du fond. De fait nous affichons les murs et les espaces vide comme un carré de couleur unie, en mettant le caractère et le fond à la même couleur, et en laissant les insectes et la nourriture avec les % et *, mais en affichant ces caractères de manière à ce qu'ils ressortent sur la couleur du vide. Cela nous permet même d'afficher des informations contextuelles, comme par exemple le fait qu'un insecte arrive au bout de ses réserves de nourritures, ou bien à la fin de sa durée de vie, qui seras représentée par un changement de couleur.

De plus, cet affichage nous permet d'appliquer la vision du labyrinthe. Lorsqu'on commence la partie, rien n'est découvert, et donc les cases découvertes sont toutes noires. Les insectes ont un champ de vision de 2 cases, et découvrent le labyrinthe au fur et à mesure de la partie.

6. Les statistiques

Afin d'avoir des nouvelles de nos insectes comme leurs positions et leurs sexes qui nous permettent de savoir s'ils vont se combattre ou s'accoupler ou le temps qu'il leur reste à vivre, nous avions besoin d'afficher des statistiques, c'est pourquoi cette partie d'affichage de statistiques était très importante vis-à-vis de la stratégie du jeu. Il s'agit simplement d'un affichage de tout ce que contient chaque case occupée d'une fourmi vivante du tableau de fourmis.

7. La génération de départ de la nourriture

Afin de permettre aux insectes d'avoir un objectif et qu'ils survivent un minimum de temps nous avions besoin d'une génération aléatoire de cases de nourriture c'est pourquoi j'ai créé cette fonction à la mission simple mais très importante.

B. Les insectes

Dans cette partie nous traitons tout ce qui touche à la structure d'un insecte ainsi qu'au tableau de cette structure nommé fourmi. Il s'agit du module « insecte.c ». Cette partie a été entièrement réalisée par Keryann.

1. Génération des insectes

Afin de générer les insectes de différentes manières en fonction de l'endroit d'où on génére cet insecte j'ai fait une fonction de génération d'insectes « gene_ins » qui génère une case reçue en paramètre un insecte d'un sexe tiré au hasard ou non (un paramètre choixsexe le définit). Toutes les fonctions qui vont suivre appellent la fonction « gene_ins ».

Le paramètre de choix de sexe est utile pour la fonction qui génère les insectes au départ car au départ nous nous retrouvions avec uniquement des insectes de même sexe comme le rand qui définit le sexe est indexé sur le temps. Cette fonction qui génère les insectes au départ est la fonction « gene_ins_deb » qui génère un nombre d'insecte déclaré dans la fonction.

Il y a également une fonction « bebe » qui, elle, crée un insecte, lorsque deux fourmis s'accouplent, dans une case aléatoire qui doit déjà être découverte.

Nous avons une fonction « ajout_insecte » qui ajoute un insecte dans une case découverte donnée par l'utilisateur, la possibilité d'ajouté un insecte n'est possible, en moyenne, que dans un cas sur deux.

2. Modification de la structure d'un insecte ainsi que de sa position

Dans cette fonction nommée « modif_pos » on décrémente le nombre de tour de nourriture restant ainsi que le nombre de tours de vie restants, on ré incrémente également ce nombre de tours de nourriture lorsqu'un insecte va sur une case de nourriture. Lors d'un déplacement on modifie la position d'un insecte dans un labyrinthe en modifiant l'état des cases de départ et d'arrivés ainsi que l'entier insecte de ces mêmes cases. Enfin on met à jour les x et y de l'insecte présents dans l'énumération. C'est également dans cette partie que l'on décidé si l'insecte doit mourir de vieillesse ou de malnutrition.

3. Suppression des insectes

La fonction « delete_insecte » a pour rôle, comme son nom l'indique, de supprimer un insecte en le transformant en nourriture, pour cela on met à jour le tableau de fourmi dans lequel tous les indices des fourmis se décale. L'idée ici est donc, lorsqu'on supprime une fourmi du jeu, de la supprimer totalement du tableau et donc d'être sûr de ne plus la traiter et de ne pas prendre trop de place dans le tableau.

C. Les déplacements et interactions entre les insectes

Dans cette partie nous gérons le déplacement des insectes, c'est-à-dire que nous décidons où l'insecte doit aller, c'est également ici que nous gérons les interactions entre deux insectes. Il s'agit du module « deplacement.c » Cette partie a été réalisée en majorité par Fabien sauf pour la fonction « combat » et la fonction « decouvrir » réalisés par Keryann.

1. Déplacements aléatoires

Dans un premier temps, nous avons convenu d'utiliser une méthode où les insectes se déplaçaient aléatoirement afin que nous puissions tester la validité du code déjà produit. Pour se faire, je me suis servi de la fonction « rand » de la bibliothèque <time>. J'ai créé une variable pour contenir le nombre obtenu entre 0 et 99, puis cette variable est testée pour que le déplacement soit également possible vers les quatre points cardinaux. J'ai aussi créé une fonction « insecte_adjacent » qui détecte si un autre insecte est directement adjacent afin de gérer les interactions entre les insectes.

2. Déplacements contrôlés

Dans un second temps, j'ai voulu mettre en place une fonction faisant que les insectes auraient des déplacements plus "intelligents". Je voulais mettre en place un algorithme de parcours de graphe de Dijkstra, mais je me suis rapidement rendu compte que ça allait être très complexe à réaliser en C. Le professeur m'a ensuite aiguillé vers une autre solution, plus simple, et n'impliquant pas de graphe à proprement parler : utiliser une matrice des distances. J'ai donc créé deux fonctions, « deplacement » et « pluscourte_dist », afin de pouvoir gérer les déplacements des insectes.

La fonction « pluscourte_dist » crée une matrice correspondant au labyrinthe mais ne contenant que la distance à la position d'origine passée en paramètre. Puis elle parcourt le chemin de l'arrivée vers le départ avant de renvoyer la case dans la direction de l'objectif.

La fonction « deplacement » crée un tableau contenant toutes les positions des cases de nourriture puis parcours ce tableau en cherchant la plus proche. Ensuite la fonction vérifie la présence d'autres insectes avant de faire appel à la fonction de modification de position créée par Keryann. Si la plus proche nourriture est trop loin, elle va remplacer son propre déplacement par la fonction de déplacement aléatoire.

3. Interactions entre les insectes

J'ai réalisé cette fonction « combat » avec une règle très simple : deux insectes de mêmes sexes combattent à la mort, deux insectes de sexes opposés s'accouplent pour donner naissance à un nouvel insecte. La seule condition autre est que l'issue du combat est dû à la quantité de

nourriture détenue par un insecte ainsi que par son âge. Cette fonction est appelée par « insecte_adjacent » en fonction des éventuels insectes voisins.

4. Découverte du labyrinthe

Nous avions fixé des règles pour la découverte du labyrinthe, un insecte découvre de deux case en croix autour de lui et d'une case dans les diagonales ce qui nous donne un losange. Les cases de nourritures ne sont pas découvertes si aucun insecte n'est passé par là. La fonction « decouvrir » effectue uniquement une découverte en ce sens.

D. L’interface utilisateur tour par tour

Dans ce module nous gérons l’interface avec l’utilisateur lors de chaque tour avant d’agir sur les insectes. Toute cette partie est sous forme de menu dans lequel on peut agit de différente manières les fonctions de sauvegarde et de chargement y sont également présentes. Il s’agit du module « utilisateur.c ». Cette partie a été réalisée par Léo pour toute la partie sauvegarde et chargement et par Keryann pour toute la partie menu et ajout de nourriture.

1. Sauvegarde/Chargement

Il nous semblait important de pouvoir sauvegarder une partie, afin de la recharger plus tard pour la continuer. Pour cela, nous avons convenu d'un seul emplacement de sauvegarde (afin de simplifier la mise en œuvre), qui correspond à un fichier texte, nommé SAVED_GAME. Dans ce fichier, on enregistre toutes les informations sur une seule ligne, de la manière suivante :

- D'abord le labyrinthe, en alternant entre l'état de la case (mur, insecte, rien …) et le fait que la case a été découverte ou pas. Les données sont inscrites dans l'ordre d'un parcours de matrice standard.

- Ensuite, on inscrit le nombre d'insectes, puis les informations relatives à chaque insecte (avec des espaces entre chaque nombre, puisqu'ils peuvent être supérieurs à 9). Ces informations sont celles contenues dans la structure insecte, et correspondent au sexe, au nombre de tour de nourriture qu'ils leurs restent, au nombre de tours de durée de vie qu'ils leurs restent, et à leurs positions en x et en y.

2. Menu et interface avec l’utilisateur

L’utilisateur à le choix entre créer un nouvel insecte avec une chance sur deux de ne pas pouvoir en créer un ou créer entre une et trois nourriture (déterminé aléatoirement). L’utilisateur peut également décider de passer cette étape et de ne rien placer pour passer au tour suivant. Nous avons également ajouté un code caché pour l’utilisateur qui nous permet de découvrir toutes les cases du labyrinthe et donc de gagner. Ce menu donne également accès à un abandon de la partie, à la sauvegarde ou au chargement d’une partie. L’ajout de la nourriture se fait comme celui d’un insecte on demande les coordonnées d’endroits découverts à l’utilisateur. On peut voir dans la figure 6 en annexe le rendu du premier tour après avoir lancé le jeu.

E. Le menu principal

Dans ce module : « main.c » on a simplement le menu principal, les règles du jeu, le lancement du jeu et une vérification de si on a gagné. Cette partie a été réalisée par Keryann.

1. Menu et règles du jeu

Le menu est assez simple puisqu'il propose simplement trois choix : voir les règles du jeu, lancer la partie ou quitter. Les règles du jeu sont comme elles ont été énoncés plus haut avec en plus une explication des codes couleurs.

2. Lancement du jeu

La fonction de lancement est en réalité la pièce maitresse de ce jeu puisque c'est elle qui décide du déroulement du jeu, on génère un labyrinthe grâce au module concernant ce même labyrinthe, on y place les insectes et la nourriture, on découvre le labyrinthe autour des insectes puis on l'affiche. Ensuite on rentre dans une boucle qui ne s'arrête que lors d'une défaite ou d'une victoire qui nous est indiquée grâce à la fonction « verifvictoire ». Dans cette boucle on demande à l'utilisateur l'action qu'il souhaite réaliser à l'aide du module d'interface utilisateur tour par tour. Puis pour chaque utilisateur on effectue un déplacement, on utilise ainsi le module de déplacment. Enfin on affiche le labyrinthe et les statistiques du tour suivant.

V. Résultats

Comme tous projets informatiques, il y a eu des imprévus et des contretemps qui ont failli nous mettre en retard. Nous avons d'abord été confrontés au problème de l'organisation d'un groupe travaillant sur un même sujet en même temps. Nous avons réussi à passer outre grâce au temps passé à la planification et la répartition des tâches effectué au début du projet.

Le problème suivant que nous avons eu a été de réussir à transposer correctement dans un terminal un jeu qui par définition est graphique. Nous avons dû nous adapter car nous ne pensions pas avoir suffisamment de temps pour réaliser une interface graphique.

Nous avons ensuite été confrontés à quelques légers problème de coordination, certains membres du groupe n'ayant pas toujours le temps de travailler avec les autres en dehors des cours.

Dans l'organisation nous avons aussi buté sur l'organisation de nos fichiers c, typiquement nous avions commencés par avoir chacun nos fichiers c cependant nous avons rapidement du ranger tout cela dans des fichiers c qui sont en fait des modules ayant du sens. L'utilisation d'un gestionnaire de source nous a beaucoup aidé pour cela.

Nous avons assez rapidement mis des commentaires au format interprété par doxygen ainsi nous avons pu produire une doc convenable qui est disponible sur notre git : https://github.com/j4r0g/Projet_Labyrinthe.

Le dernier problème que nous avons rencontré a été lors du nettoyage et de l'optimisation du code. Après quelques réarrangements dans le code, le programme a commencé à ne plus fonctionner correctement. Nous avons dû passer plus de temps que prévu à débugger au dépend du temps que nous avions initialement prévu pour la rédaction de ce rapport.

VI. Conclusion

De par ce projet, nous avons pu constater l'importance d'une bonne organisation.

Nous avons beaucoup appris au cours de ce projet. Pas seulement en terme de code ou d'algorithmique mais aussi en terme de gestion d'une équipe, comment travailler ensemble, comment poursuivre un même but en essayant d'éviter les incompréhensions.

Nous n'avons pas tout à fait respecté le cahier des charges qui nous a été fourni, préférant faire l'impasse sur certaines fonctionnalités pour gagner du temps bien que ce programme n'est pas terminé. Nous sommes tout de même satisfaits du travail fourni, du résultat obtenu, bien que cela soit très perfectible, et de l'expérience que nous as apporté ce projet en terme de gestion d'un groupe pour atteindre un objectif donné dans un délai fixé.

VII. Annexes

A. Annexe 1 : Le labyrinthe

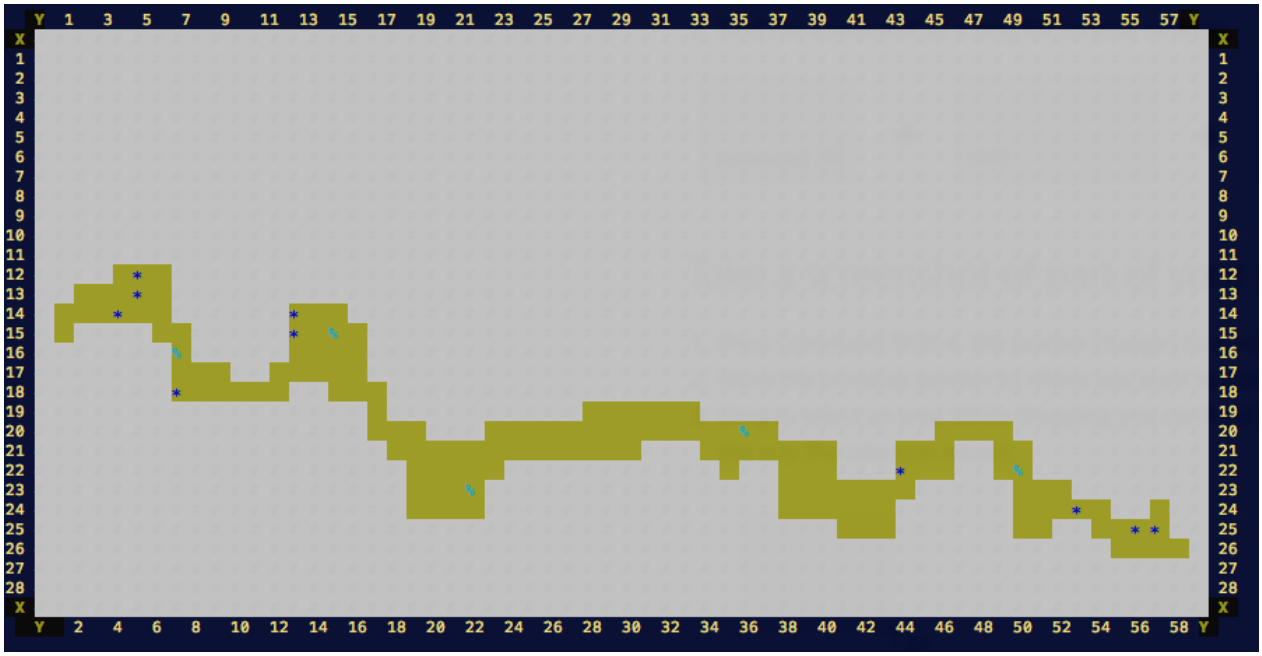

Figure 1

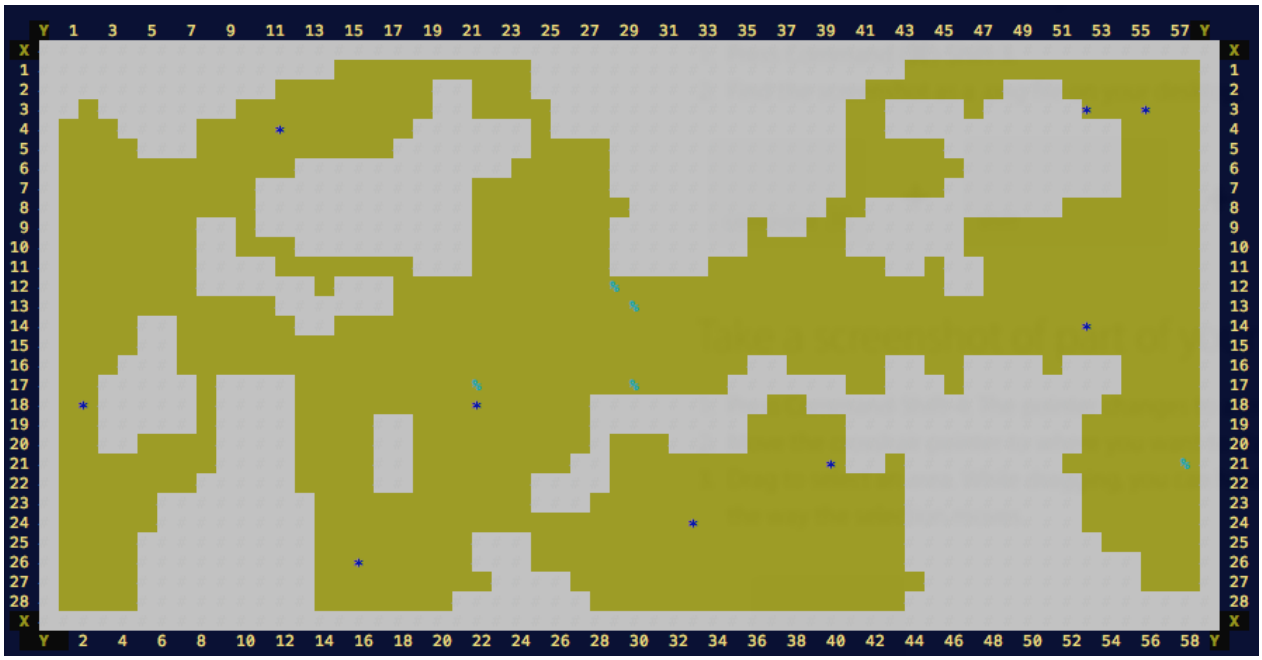

Figure 2

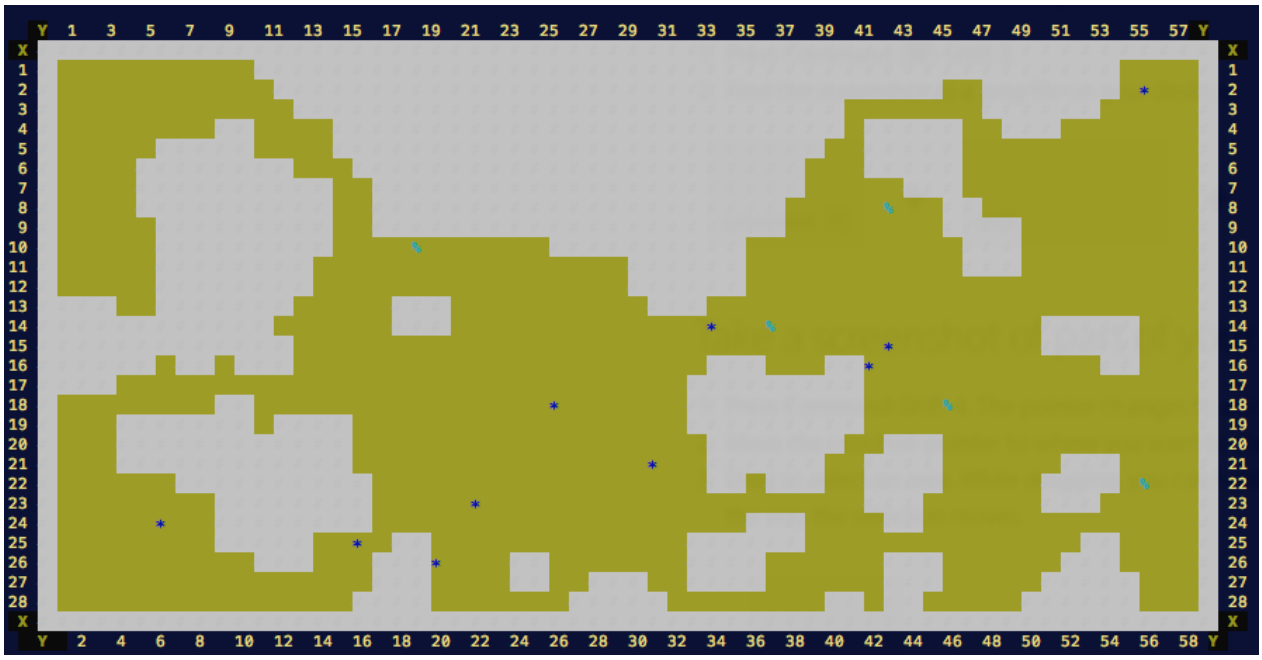

Figure 3

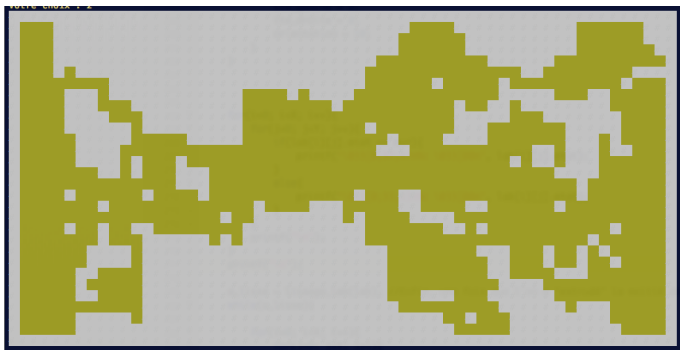

Figure 4.1

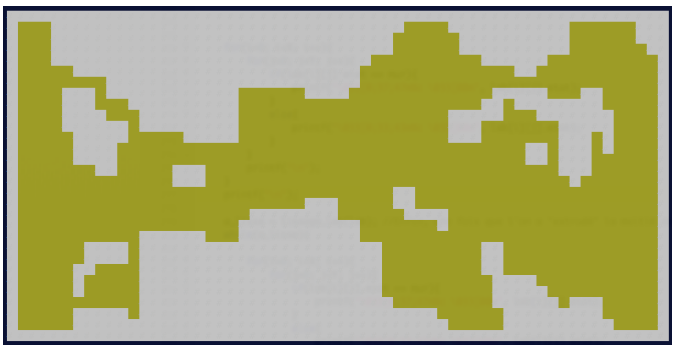

Figure 4.2

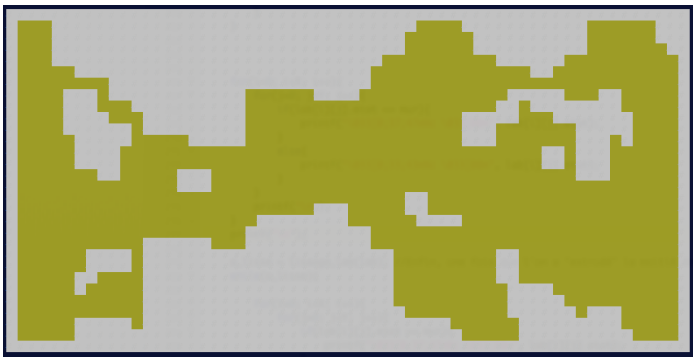

Figure 4.3

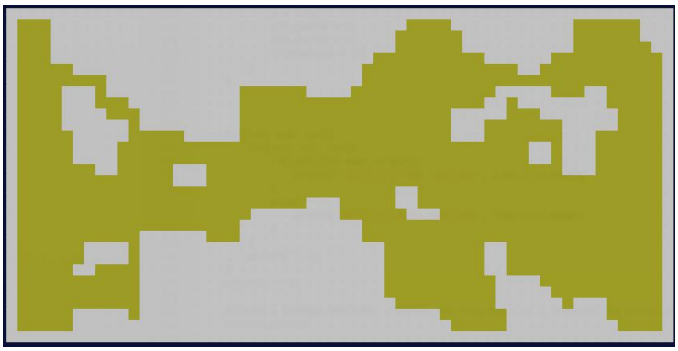

Figure 4.4

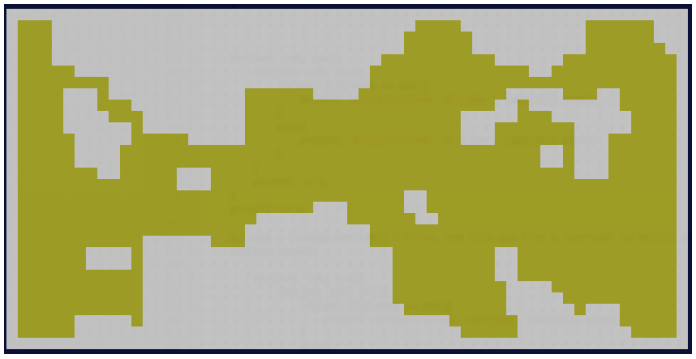

Figure 4.5

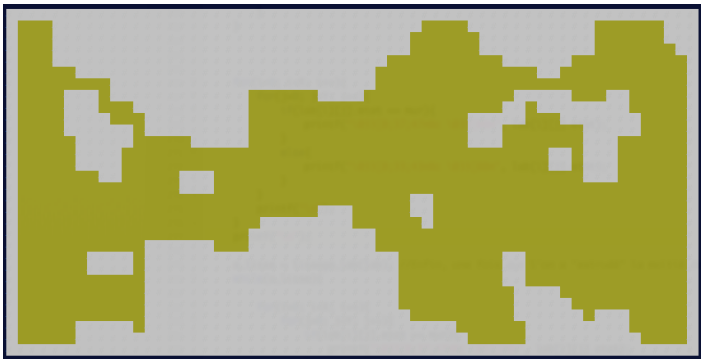

Figure 4.6

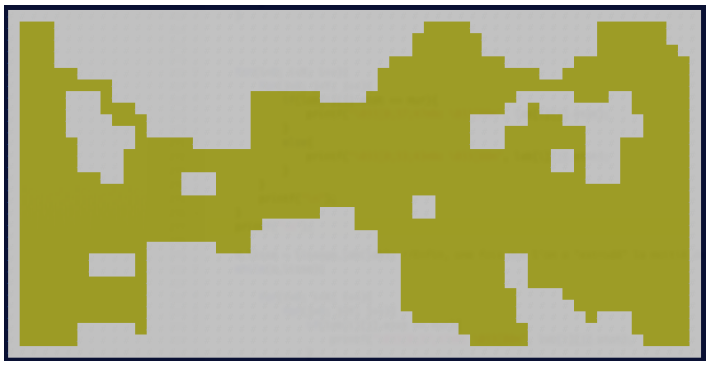

Figure 4.7

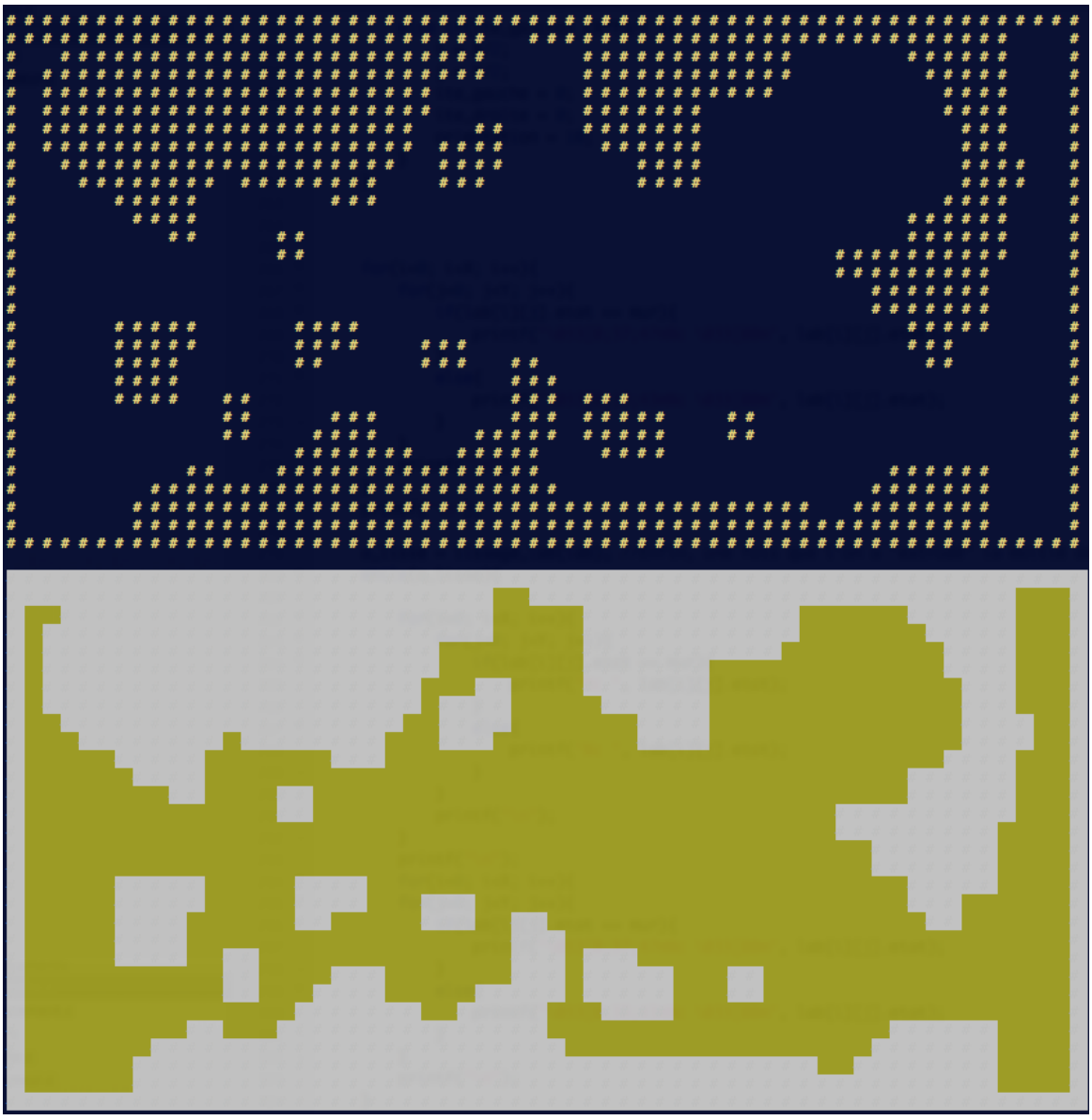

Figure 5

B. Annexe 2 : Interface avec l'utilisateur

Figure 6 : Un tour de jeu (ici le premier)