

**Rapport du projet informatique :**

**Année universitaire :** 2024 / 2025

**Groupe MI1-A :**

Lahouari Guetarni

Mathis Rameau

Sara Belghazi

**Coconut partY**



**Sommaire :**

* Introduction
* Répartition des tâches entre les membres du groupe
* Présentation des premières démarches et procédures
* Problèmes rencontrés
* Solutions et améliorations apportées au programme
* Conclusion

**Introduction :**

Dans le cadre de notre projet de fin de semestre en informatique, nous avons formé un trinôme et choisi un sujet autour de la création d’un jeu vidéo. Après avoir réfléchi ensemble, nous avons décidé de créer **"Coconut Party"**, un jeu de type **‘Tower-Defense’** dans lequel le joueur place des singes sur une carte afin de défendre la couronne de leur roi en repoussant des crabes cupides qui cherchent à s’en emparer.

Dans ce rapport, nous reviendrons sur les principales étapes de la création et du développement du jeu. Nous y présenterons les choix que nous avons faits, les problèmes que nous avons rencontrés, ainsi que les solutions trouvées et les améliorations apportées au programme.

**Répartition des tâches entre les membres du groupe :**

Afin de mener à bien le développement de notre jeu, nous avons commencé par organiser notre travail en répartissant les tâches entre les membres du groupe. Ceci nous a permis d'avancer de manière plus efficace et de mieux structurer notre projet.

Par exemple, Mathis s’est principalement chargé de la **création du plateau de jeu** (**gameBoard\_creation** **()**) et du **lissage de la carte** (**smooth\_gameBoard ()**), afin d’assurer une cohérence visuelle entre les zones de terre et de mer.

Lahouari quant à lui, a travaillé sur la **création du chemin aléatoire** (**creer\_chemin ()**) qui relie la case de départ à la couronne, une tâche complexe qui a nécessité plusieurs tentatives et ajustements (cette dernière a été renommée plus tard en (generate\_path)), et a créé la fonction **(directions ())** qui génère aléatoirement une direction pour le chemin (cette fonction a également été renommée plus tard par (choix\_direction()).

En parallèle, Sara a pris en charge l’implémentation des entités du jeu, notamment la création des crabes et des singes **(creer\_singes\_et\_crabes ())** (quia été modifiée et séparée en deux au fil du projet pour plus de clarté) et **la gestion des fichiers du projet**, en mettant les fonctions dans des fichiers .c et .h séparés, et en écrivant le **Makefile** pour faciliter la compilation du code.

**Présentation des premières démarches et procédures :**

Au début, nous avons commencé par créer un tableau 2D de cases à l’aide d’un double pointeur (Box\*\*) définies par une structure *Box*, pour former notre carte (map). Chaque case, dispose d’un type défini par une énumération pour représenter les différents terrains et éléments du plateau de jeu :

* **START** : case de départ des ennemis (crabes)
* **SEA** : pour l’eau
* **LAND** : pour la zone terrestre
* **CROWN** : pour la couronne (zone à défendre)
* **MONKEY** : pour le singe
* **CRAB** : pour les crabes
* **PATH** : pour le chemin suivi

Cette interface 2D représente donc l’**arène de jeu**, et nous avons implémenté un système de génération aléatoire, pour créer automatiquement une carte contenant des zones de **terre ferme** et des **zones maritimes (vagues)**, avec une probabilité réglée dans le code, pour diversifier et donc pour avoir une carte différente à chaque exécution du jeu.

Nous avons ensuite créé la fonction « **gameboard\_creation (int height, int width)** » qui alloue dynamiquement le tableau de cases 2D et initialise chaque case avec des coordonnées (pos\_x, pos\_y) et un type de terrain « LAND » ou « SEA » selon la probabilité mentionné dans le paragraphe précédent. Les bords sont toujours définis comme « SEA », ce qui donne un contour logique à la carte.

Et pour éviter les incohérences comme par exemple une case « SEA » dans les cases « LAND », nous avons créé une fonction de **lissage appelée « smooth\_gameBoard(Box\*\* game\_board, int height, int width)»** qui vérifie les types de terrain autour de chaque case, afin d’harmoniser le plateau de jeu.

Ensuite, nous avons créé une procédure appelée : « **play\_area(Box\*\* game\_board, Box\*\* path, int\* path\_length, int height, int width, int\* start\_x, int\* start\_y, int\* zone\_top)** » qui permet de définir la zone jouable au centre du plateau. Nous avons créé deux boucles « **for** » afin de parcourir tout le plateau et transformer les cases **SEA** en cases **LAND** pour avoir assez d’espace de jeu. Elle permet également de placer la case **«** **START »** en bas du plateau d’où sortiront les crabes et la case **«** **CROWN »** en haut que les crabes essaieront d’atteindre.

La création de la fonction « **creer\_chemin ()** » a été notre plus grande difficulté. Cette dernière **génère un chemin aléatoire** allant de la case « **START** » à la case « **CROWN** ». Elle utilise un système de direction contrôlée (**directions ()**), qui interdit certains mouvements pour éviter les allers-retours incohérents (comme par exemple : gauche puis droite).

Nous avons ensuite créé une fonction **« directions () » qui** prend en entrée la dernière direction suivie **(int derniere\_direction)** et renvoie une nouvelle direction aléatoire. En effet, elle génère aléatoirement une direction parmi trois possibilités **(1 : GAUCHE, 2 : HAUT, 3 : DROITE),** en s'assurant que la nouvelle direction n'est pas opposée à la précédente, et ce pour permettre d'éviter les retours en arrière.

Et pour peupler le chemin de crabes et de singes, nous avons créé la fonction « **creer\_singes\_et\_crabes ()** » où une certaine probabilité permet de placer des singes **(MONKEY)** sur le chemin, en tant que défenseurs, et que d’autres permettent de placer des **crabes (CRAB)**, se déplaçant vers la couronne.

Nous avons aussi écrit une fonction « **display\_gameBoard ()** » qui permet d’afficher l’état actuel du plateau dans le terminal, en associant chaque type de case à un symbole différent de **la table ASCII** :

* (**~**) *pour l’eau*
* (**.**) *pour le chemin*
* (**@**) *pour les singes*
* (**§**) *pour les crabes*
* (**X**) *pour la couronne*

Et enfin pour finir nos fonctions, nous avons libéré la mémoire allouée dynamiquement pour éviter les fuites mémoire à la fin du programme avec la procédure **« free\_gameBoard(Box\*\* game\_board, int height) ».**

Et donc pour la fonction principale **« main () »**,nous avons seulement initialisé l’aléatoire avec « **srand(time(NULL)) »** et enchainé les appels des fonctions cités auparavant :

* **(gameBoard\_creation)**
* **(smooth\_gameBoard)**
* **(play\_area)**
* **(creer\_chemin)**
* **(creer\_singes\_et\_crabes)**
* **(display\_gameBoard)**
* **(free\_gameBoard**)

**Problèmes rencontrés :**

Dans la présentation de notre démarche, nous avons précisé que la création de la fonction « **creer\_chemin ()** » a été notre plus grande difficulté. En effet, le système était **complexe à implémenter** : il fallait à la fois garantir que le chemin respecte les bords de la carte, et qu’il ne boucle pas indéfiniment.  
**Nous avons passé environ deux semaines et demie sur cette seule fonctionnalité**, à cause d’erreurs de logique, de dépassements de bornes et d’un manque de structuration du programme, car à cette étape du projet, tout notre code était encore déposé dans un seul fichier source, ce qui compliquait sa lecture, les tests, et sa modification.

De plus, nous ne maîtrisions pas encore parfaitement **GitHub**, ce qui a provoqué un désordre, sans gestion claire des versions. Cette expérience a mis à l’épreuve **notre patience et notre résilience et nous a donc poussé à mieux organiser notre travail…**

**Solutions et améliorations apportées au programme :**

Après ces difficultés, nous avons compris la nécessité d’une meilleure organisation, nous avons donc appris à mieux utiliser **GitHub**, pour réorganiser tout le projet. En effet, nous avons déplacé chaque fonctionnalité dans un **fichier.c** dédié, mis les déclarations dans des **fichiers.h** et créé le **Makefile** pour simplifier la compilation et l’exécution du programme.

Nous avons aussi modifié quelques fonctions comme **«creer\_singes\_et\_crabes () »** et ce, en la séparant en 2 fonctions :

* **« create\_crab(Box\*\* game\_board, Box\* path, int height, int width, int start\_x, int start\_y)»** qui prend en paramètres : le tableau 2D (Box\*\* game\_board) qui représente le plateau; (Box\* chemin) qui représente le chemin que doivent suivre les crabes ; les dimensions du plateau (**height** et **width**) et la position de départ des ennemis **(start\_x ;start\_y)** , est une fonction qui initialise un crabe sur le plateau de jeu en le plaçant à une position spécifique du chemin. En effet, elle vérifie si la case de départ est bien de type **« START »** puis place le crabe sur la 2ème case du chemin **(path[1]).** Cette fonction appelle aussi une autre fonction appelée **«attribute\_crab(Crab\* crab) »**  qui attribue les points de vie (hp) à un crabe en fonction de son niveau. Elle vérifie d'abord que le pointeur n’est pas nul (pour éviter un crash), puis assigne une valeur croissante de hp selon le niveau.
* Et **« create\_monkey(Box\*\* game\_board, Box\* path, int path\_length, int height, int width, int level)»** qui a pour objectif de créer un singe, de l’initialiser selon son niveau en appelant **attribute\_monkey(Monkey\* monkey),** qui prend en paramètre un pointeur vers une structure **Monkey** pour lui attribuer des statistiques en fonction de son niveau (**level**), et de le positionner sur une case valide du chemin dans le plateau de jeu en appellant une procédure **« ask\_monkey\_position(Box\*\* game\_board, Box\* path, int path\_length, int height, int width, int\* x, int\* y)»**  pour demander au joueur d’entrer les coordonnées (x ; y) de positionnement du singe.
* **« generate\_path(Box\*\* game\_board, int\* start\_x, int\* start\_y, int\* path\_length, int zone\_top, int zone\_bottom, int zone\_left, int zone\_right, int height, int width) »** est la version améliorée et robuste de la fonction **« creer\_chemin »,** en effet, elle génère un chemin **dynamique** stocké dans un tableau **(Box\* path)** qui relie le point de départ **START** à la case d’arrivée **CROWN.** Et pour éviter le risque de boucle infinie au cas où il est impossible d’atteindre la couronne, nous avons implémenté une condition d’arrêt, limitant le nombre d’itérations à 5000.
* **« choice\_direction(int last\_direction, Box box\_path, int zone\_top, int zone\_bottom, int zone\_left, int zone\_right, int width)»** qui prend en paramètres la dernière direction utilisée, la case actuelle du chemin, les limites de la zone de jeu (up, down, left, right), et la largeur du plateau est une fonction qui génère une direction aléatoire avec des probabilités différentes (40% left, 40% right, 20% up) et vérifie si cette direction est valide en prenant en compte plusieurs critères comme par exemple : éviter les retours en arrière (pas de gauche après droite ou inversement) et s'assurer que la position suivante reste dans les limites de la zone de jeu.
* La fonction principale **« main () »,** qui après avoir inclus les bibliothèques nécessaires et tous les **fichiers.h,** initialise le générateur aléatoire **(srand (time(NULL))**, et génère aléatoirement de nouveaux crabes dans une boucle **(WHILE)** (*5% de chance par itération*), tout en appelant les fonctions suivantes :

*(Quelques fonctions mentionnées ci-dessous seront vues dans les prochaines pages)*

* **display\_menu**
* **ask\_players\_name**
* **game\_board\_dimension**
* **gameBoard\_creation**
* **smooth\_gameBoard**
* **play\_area**
* **create\_monkey**
* **create\_crab**
* **move\_crab**
* **monkeys\_attack\_crabs**
* **display\_gameBoard**
* **usleep ()  : (*Permet une pause de quelques microsecondes entre chaque tour)***
* **free\_gameBoard**

Nous avons également rajouté de nouvelles fonctions comme :

* La fonction **« display\_menu () »** qui affiche le menu principal avec trois options : **Nouvelle partie** (relance le jeu depuis le début), **Quitter** (ferme le programme avec un message d'au revoir), et **Reprendre une partie** (charge une partie sauvegardée en lisant les données depuis un fichier ***(sauvegarde.txt)***).
* La fonction « **ask\_players\_name ()** » demande à l’utilisateur d’entrer son pseudo, le valide (non vide), et le retourne sous forme d’une chaîne de caractères allouée dynamiquement.
* La fonction **« move\_crab(Crab\* tab\_crab, Box\*\* game\_board, Box\* path, int path\_length, int nb\_crabs)»** gère le déplacement d'un crabe **CRAB** le long d'un chemin prédéfini sur le plateau de jeu et ainsi déplace le crabe d'une case vers la suivante sur le chemin, tout en mettant à jour l'affichage du plateau. La fonction parcourt tous les crabes actifs en commençant par le dernier. Pour chaque crabe, elle libère sa position actuelle sur le plateau, avance le crabe d'une case sur le chemin, puis marque sa nouvelle position. Si un crabe atteint la fin du chemin, il est désactivé.
* La fonction **«** **monkeys\_attack\_crabs(Monkey\* tab\_monkeys, Crab\* tab\_crab, int nb\_monkeys, int nb\_crabs, Box\*\* game\_board)»** permet aux singes placés d'attaquer les crabes qui se trouvent sur les cases voisines (haut, bas, gauche ou droite). Chaque singe peut infliger 1 point de dégât à un crabe adjacent. Si les points de vie (hp) d’un crabe tombent à 0 ou moins, il est désactivé (active = false).
* La fonction « **game\_board\_dimension (int\* height, int\* width) »** qui permet à l'utilisateur de définir les dimensions du plateau de jeu en demandant la hauteur et la largeur. Elle contient un « scanf » pour lire les entrées et une boucle **« DO WHILE »** qui recommence tant que les valeurs ne sont pas valides. Cela garantit que le plateau aura toujours des dimensions valides pour le bon fonctionnement du jeu.
* La fonction **« monkeys\_store(Monkey\* tab\_monkey, int\* nb\_monkeys, int\* nb\_ccn, Box\*\* game\_board, Box\* path, int path\_length, int height, int width) »** prend en paramètre un tableau de singes **(tab\_monkeys)**, le nombre de singes **(nb\_monkeys)** et le nombre de noix de coco **(nb\_ccn),** c’est un magasin où le joueur peut acheter et améliorer des singes en utilisant des noix de coco comme monnaie, la fonction vérifie si le joueur a assez de noix de coco (au moins 3) pour accéder au magasin. Ensuite, elle affiche un menu proposant trois options : acheter un singe, améliorer un singe ou mettre le jeu en pause.
* La procédure **« display\_crabs\_turn(int nb\_crabs[3], int tot\_crabs)» informe le joueur que des crabes arrivent et lance un décompte avant l'attaque.**
* La procédure **« countdown () »** effectue un décompte simple de 3 à 1, avec une pause de (500 millisecondes) entre chaque nombre avant le lancement du jeu.
* La fonction **« turn(Box\*\*game\_board, Turn current\_turn,int\* nb\_ccn, Monkey\*\* tab\_monkeys, int nb\_monkeys[3],Crab\*\* tab\_crabs, int nb\_crabs[3], int\* tot\_monkeys, int\* tot\_crabs, Box\* path, int path\_length, int height,int width,int start\_x,int start\_y, int zone\_top,char\* pseudo) »** commence par gérer le **début du tour** : si c'est le 1er ou 2ème tour, le joueur reçoit un singe gratuit **« create\_monkey »**. À partir du 3ème tour, il a accès à **« monkeys\_store » pour acheter ou améliorer des singes**. Ensuite, selon le numéro du tour, un certain nombre de **crabes ennemis** sont générés grâce à « **create\_crab** ». Cette fonction appelle aussi d’autres fonctions comme : **« display\_crabs\_turn »**, « **move\_crab »**, **« monkeys\_attack\_crabs »** et **« display\_gameBoard »**.
* La fonction **« turn\_creation(int num)»** sert à créer un nouveau tour de jeu, elle crée une structure **Turn** avec : le numéro du tour (**num**), un indicateur que le tour **est en cours** (**active = true**), et qu’il n’a pas encore échoué (**success = true**). Cela permet ensuite au jeu de savoir à quel tour on est, s’il est toujours actif, et si le joueur l’a réussi ou non.
* La fonction **« game\_over\_menu(int turn, int nb\_ccn, char\* pseudo)»** permet d’afficher le menu de fin de partie. Elle montre au joueur : Le nombre de **tours** atteints, Le nombre de **noix de coco gagnées**, Et propose de **recommencer** une partie ou de **quitter le jeu**.
* La fonction **« load\_game(Crab\* crab, int\* monkey\_count, int\* nb\_ccn)»** permet de charger une partie sauvegardée depuis un fichier texte "**sauvegarde.txt**". (*Cette fonction n’a pas pu être intégrée dans le programme final*).
* La fonction **« save\_game(Crab crab, Monkey\* monkeys, int monkey\_count, int nb\_ccn)»** enregistre l'état actuel du jeu dans le fichier "**sauvegarde.txt**". En effet, elle sauvegarde les données du crabe (**x, y, path\_index, hp**), le nombre de singes (**monkey\_count**), les noix de coco **(nb\_ccn)**, puis boucle sur tous les singes pour enregistrer leurs positions, temps de tir, et portée. (*Cette fonction n’a pas pu être intégrée dans le programme final*).
* Pour le côté esthétique, nous avons amélioré la fonction « **display\_gameBoard(Box\*\* game\_board, int height, int width, char\* pseudo, int nb\_ccn, int turn)»** qui parcourt tout le plateau case par case et l’affiche avec un système de couleurs en utilisant des codes **ANSI** pour différencier les éléments du jeu :
  + - **(LAND) : Vert**
    - **(SEA) : Bleu**
    - **(CROWN) : Jaune**
    - **(START) : Gris**
    - **(PATH) : Blanc**
    - **(MONKEY) : Violet**
    - **(CRAB) : Rouge**
* Enfin, nous avons rajouté une procédure **« clear\_screen »** qui nettoie l'affichage du terminal. En effet, elle utilise les séquences d'échappement ANSI pour repositionner le curseur au début de l'écran **(\033[H)** et effacer tout le contenu **(\033[J)**, elle appelle ensuite **« fflush(stdout) »**, une fonction qui renouvelle l'affichage du plateau de jeu entre chaque itération pour obtenir une animation plus fluide.

**Conclusion** :

Ce projet nous a permis de mettre en application des notions essentielles en programmation C abordées tout au long de notre année de préing 1 telles que : l’utilisation des structures, la gestion de la mémoire dynamique, la génération aléatoire et l’organisation du code avec une compilation facilitée par un Makefile.

Enfin certaines parties, comme la génération du chemin aléatoire entre la case de départ (**START**) et la couronne (**CROWN**), se sont révélées complexes et ont nécessité de nombreux essais, corrections et ajustements. Cette étape a mis à l’épreuve notre patience, mais nous a aussi permis de mieux structurer notre code au fur et à mesure.