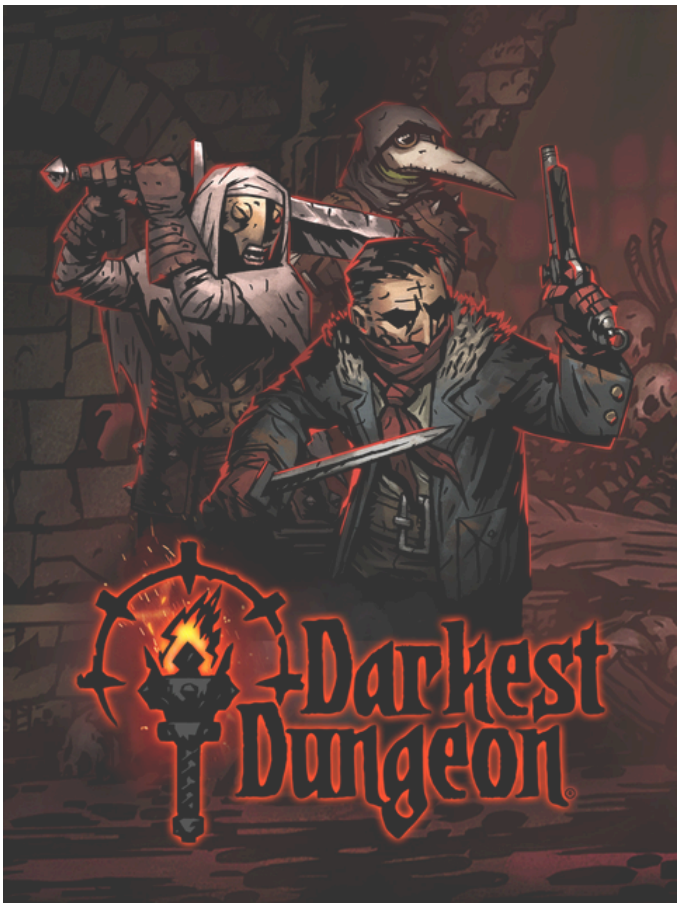
RAPPORT PROJET C

JANVIER 2025

DARKEST C DUNGEON



Inès Benhamida

Sommaire

Introduction

Ce projet consiste à créer une version simplifiée d'un jeu vidéo en C. Le but est de mettre en place des combats entre des personnages et des ennemis, où chaque personnage a des statistiques comme des points de vie, d'attaque, et de défense. Les joueurs peuvent aussi utiliser des accessoires pour améliorer leurs statistiques.
Le jeu est conçu pour montrer comment gérer des données dynamiques en C avec des listes chaînées. Cela permet de créer et de manipuler facilement des personnages, des ennemis et des accessoires tout au long du jeu.

Manuel utilisateur

Compilation

Pour compiler le projet, utilisez la commande suivante dans le terminal :clang -std=c17 -Wall -Wfatal-errors darkestc_dungeon_final.c -o darkestc_dungeon_final

Exécution

Pour exécuter le jeu, lancez simplement l'exécutable :

./darkestc_dungeon_final

Déroulement du Jeu

Menu du jeu :
Au lancement du jeu on a le choix de lancer une nouvelle partie, de charger un fichier de sauvegarde ou de quitter le jeu

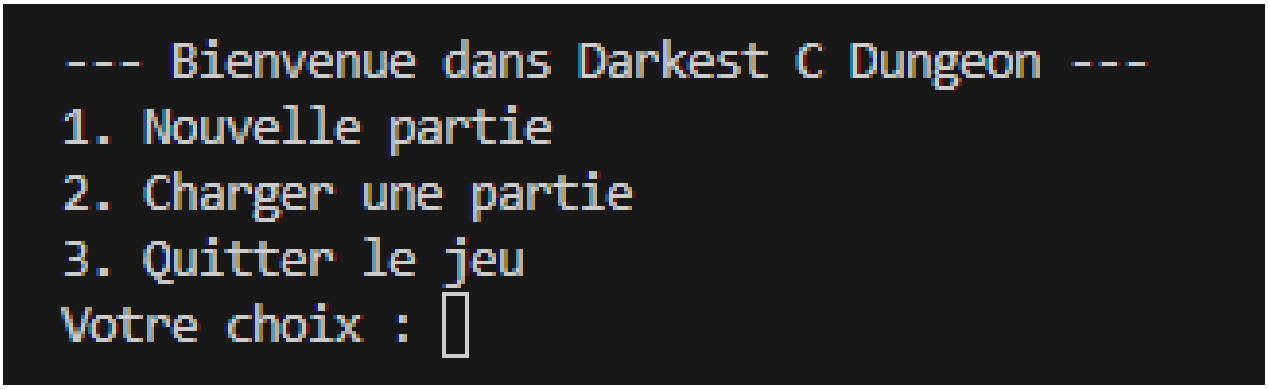


Figure 1 - Menu d'accueil du jeu

Affichage des Personnages et Accessoires :
Lorsque la partie démarre, les personnages disponibles et leurs statistiques sont affichés. Les accessoires disponibles dans l'inventaire sont également listés.

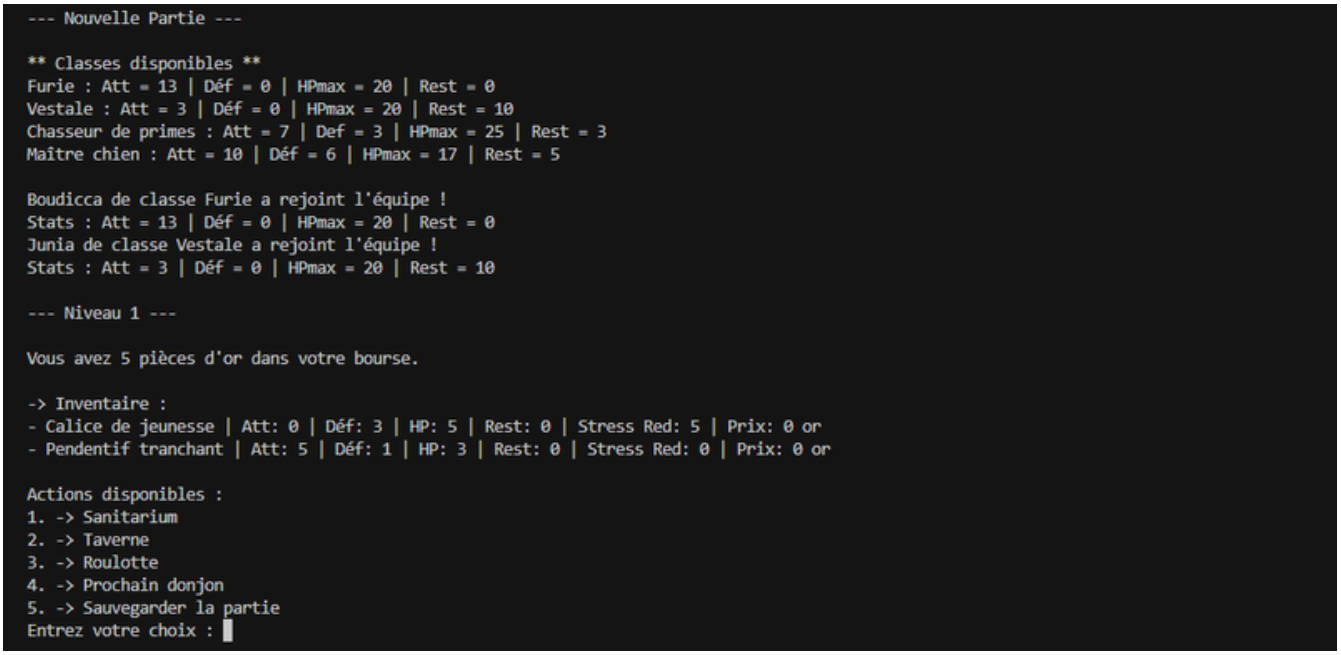


Figure 2 - Affichage du menu lors du lancement d'une nouvelle partie

Combats :
Avant le combat, le joueur doit choisir quels personnages vont prendre part au combat. Suite à cela il peut attribuer des accessoires à ses personnages. Il ne peut équiper uniquement que les accessoires présents dans son inventaire.

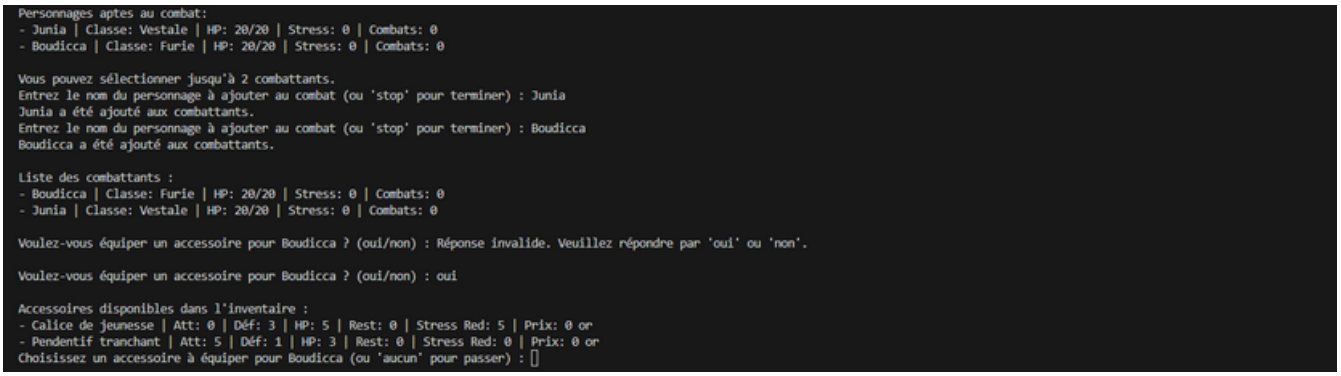


Figure 3 - Interface lors de la préparation d'un combat

Ensuite le joueur doit choisir quelles actions réaliser avec chacun des personnages, il joue au tour par tour, chacun de ses personnages agit avant que l'ennemi le fasse à son tour.

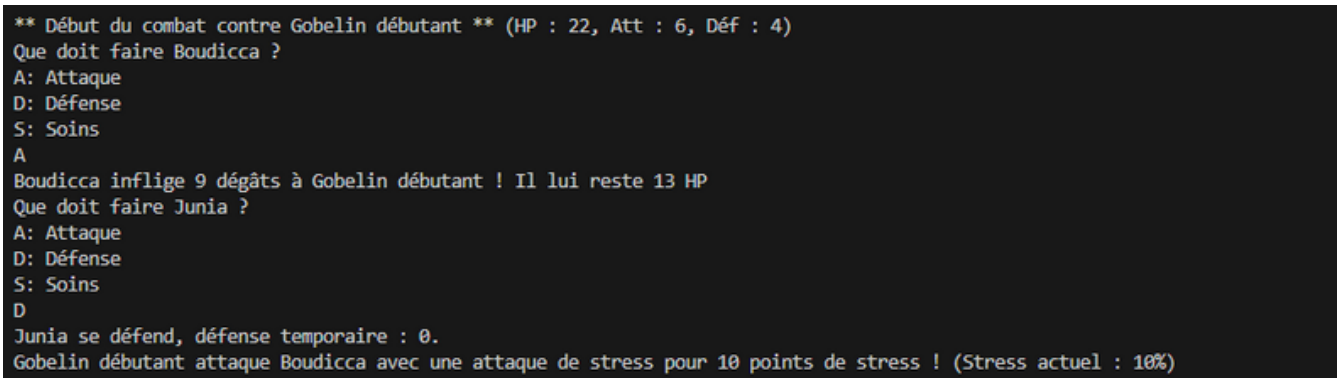


Figure 4 - Interface de combat

Une fois l'ennemi vaincu, le joueur récupère 10 pièces d'or et un accessoire aléatoire. Entre les combats le joueur a plusieurs choix:

- Envoyer des personnages au sanitarium pour qu'ils récupèrent de la santé lors du prochain combat
- Envoyer des personnages à la taverne pour qu'ils réduisent leur stress lors du prochain combat
- Acheter des accessoires à la roulotte avec les pièces d'or qu'il possède
- Sauvegarder la partie

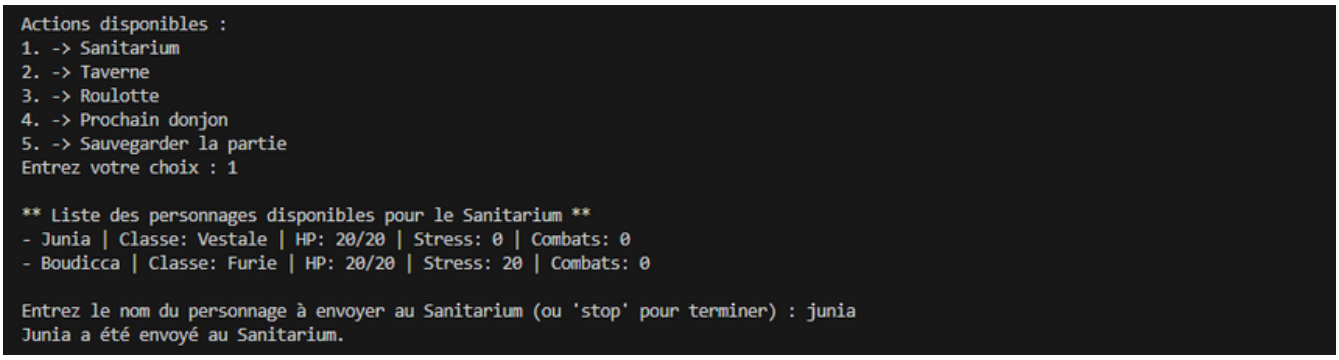


Figure 5 - Interface du sanitarium

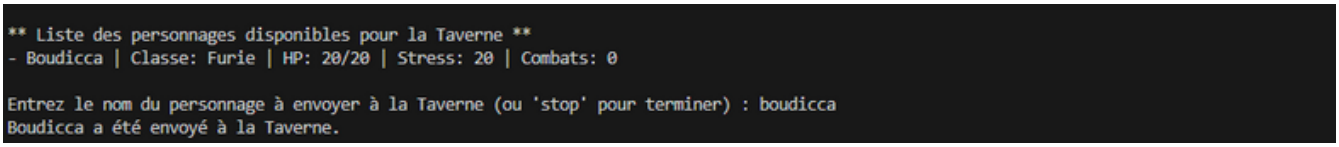


Figure 6 - Interface de la taverne

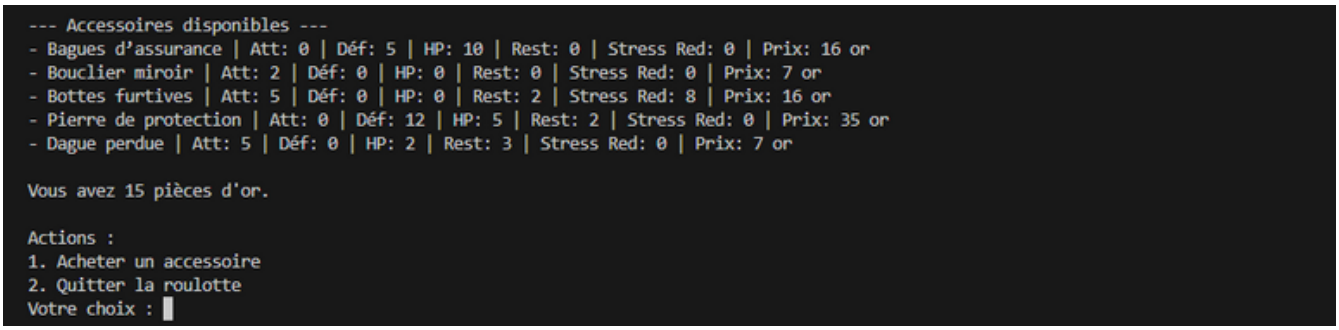


Figure 7 - Interface de la roulotte

Figure 8 - Onglet de sauvegarde du jeu

Structures du Jeu

Personnage

Chaque personnage a les caractéristiques suivantes :

- Nom : Le nom du personnage.
- Classe : Sa classe (par exemple : Furie, Vestale).
- Attaque (att) : La force des attaques qu'il peut infliger.
- Défense (def) : Sa capacité à réduire les dégâts.
- Points de vie (HP) : La quantité de vie restante et le maximum qu'il peut avoir.
- Stress : Une jauge qui monte pendant les combats et rend le personnage inutilisable s'il atteint 100.

Structure Accessoire

Un accessoire est défini par :

- nom : Nom de l'accessoire.
- attbonus, defbonus, HPbonus, restbonus : Bonus appliqués au personnage qui l'équipe.
- strred : Réduction des attaques de stress.
- prix : Coût de l'accessoire en or.
- suivant : Pointeur vers le prochain accessoire.

Structure Ennemi

Un ennemi est défini par :

- nom : Nom de l'ennemi.
- niveau : Niveau de difficulté.
- attenn, defenn, HPenn : Statistiques d'attaque, de défense et de points de vie.
- attstrenn : Attaque de stress.

```c
// Structure pour une classe
typedef struct {
    char nom[50];
    int att, def, HPmax, rest;
} Classe;

// Structure pour accessoire
typedef struct Accessoire {
    char nom[50];
    int attbonus, defbonus, HPbonus, restbonus, strred, prix;
    struct Accessoire *suivant;
} Accessoire;

// Structure pour un personnage
typedef struct Personnage {
    char nom[50];
    Classe classe;
    int HP, stress, nbcomb, defense_temporaire;
    Accessoire *accessoire1;
    Accessoire *accessoire2;
    struct Personnage *suivant;
} Personnage;

// Structure pour un ennemi
typedef struct {
    const char *nom;
    int niveau, attenn, defenn, HPenn, attstrenn;
} Ennemi;
```

Figure 9 - Structures du programme

Fonctions Créées

Gestion des Personnages

- ajouter_personnage : Ajoute un personnage à une liste chaînée.
- supprimer_personnage : Supprime un personnage mort d'une liste.
- afficher_personnages : Affiche les personnages d'une liste avec leurs statistiques.

Gestion des Accessoires

- ajouter_accessoire : Ajoute un accessoire à la roulotte.
- retirer_accessoire : Permet de retirer un accessoire d'une liste chainée pour l'attribuer à une autre.
- afficher_accessoires : Liste les accessoires disponibles.
- gerer_roulotte : Gère les accessoires disponibles à l'achat dans la roulotte et permet de les acheter pour les avoir dans l'inventaire.
- equiper_accessoire : Permet d'équiper les accessoires aux personnages combattants.
- retourner_accessoires : Permet de retourner les accessoires à l'inventaire en vue du prochain combat.
- recuperer_accessoire_sur_ennemi : Permet de mettre les accessoires lachés par les ennemis dans l'inventaire.
- liberer_accessoires : permet de libérer correctement la mémoire allouée dynamiquement pour une liste chaînée d'accessoires. On évite ainsi les fuites de mémoire.

Combat

- selectionner_combattants : Permet de choisir les personnages de l'équipe à envoyer au combat.
- combat : Gère un combat au tour par tour :
  - Les personnages choisissent leurs actions (attaquer, se défendre, restaurer).
  - L'ennemi attaque un personnage aléatoire.
  - Le combat continue jusqu'à ce que l'ennemi ou tous les personnages soient éliminés.
- gerer_personnages_morts : Supprime les personnages tombés au combat avec leurs accessoires équipés.

Autres

On a certaines fonctions définie par question de practicité, afin de les réutiliser dans d'autres fonctions du programme, on retrouve notamment :

- afficher_classes : afficher les classes de personnages présentes dans le jeu
- initialiser_roulotte : initialise les accessoires présents à la vente dans la roulotte
- initialiser_inventaire : initialise les accessoires donnés au joueur au début du jeu

Fonctions de sauvegarde

- sauvegarder_partie : Permet de sauvegarder l'avancée d'une partie dans un fichier .txt. On y écrit la composition de l'équipe, le niveau atteint, l'inventaire ...
- charger_partie : Permet de charger une partie sauvegardée précedemment à partir d'un fichier de sauvegarde .txt.

Étapes Importantes dans main()

Initialisation :

Création des classes, personnages et accessoires de départ.
Ajout d'un ennemi pour chaque niveau.
Préparation Avant Chaque Combat :
Envoi des personnages au sanitarium ou à la taverne.
Achat d'accessoires à la roulotte.

Combat :

Sélection des combattants par le joueur.
Sélection des accessoires des combattants.
Exécution de la fonction combat.
Récupération d'un accessoire à la fin des combats.

Progression :

Ajout d'un nouveau personnage tous les 2 niveaux.
Victoire après le 10e niveau.

Difficultés Rencontrées

1. Gestion des Listes Chaînées

La suppression dynamique de nœuds dans une liste chaînée (par exemple, lorsqu'un personnage meurt) a nécessité une attention particulière pour éviter les erreurs de pointeurs.

2. Calcul des Dégâts et Équilibrage

Les calculs des dégâts et du stress en fonction des statistiques ont demandé des ajustements pour garantir un bon équilibre entre difficulté et jouabilité.

.3. Incomplétude

Certaines fonctionnalités supplémentaires, comme les systèmes de queues ou intégrer des ennemis multiples dans un combat, n'ont pas été implémentées.

Réflexion sur l'Utilisation des Listes Chaînées

Les listes chaînées ont été utilisées pour gérer les personnages, les accessoires et d'autres entités dynamiques du jeu.

Avantages
Flexibilité : Elles permettent d'ajouter ou de supprimer facilement des éléments pendant l'exécution du programme.
Gestion Mémoire : Seuls les éléments nécessaires sont alloués, ce qui optimise l'utilisation de la mémoire.

Inconvénients
Complexité : Leur gestion peut être difficile, notamment lors des suppressions, et nécessite une attention particulière pour éviter des erreurs comme des fuites de mémoire.
Lenteur d'Accès : Pour accéder à un élément spécifique, il faut parcourir toute la liste, ce qui peut ralentir le programme lorsque la liste est longue

Conclusion

Le projet Darkest C Dungeon montre l'utilisation des listes chaînées pour gérer dynamiquement des entités en C, comme les personnages, les statistiques et les accessoires. Cette méthode est plutôt complexe mais permet une flexibilité dans le développement du jeu.

A travers ce projet nous avons notamment pu intégrer des mécanismes variés au jeu, tels que la gestion des personnages ou des accessoires, le tout en exploitant les structures de données de la manière la plus optimale possible.

La complexité de ce projet résidait notamment dans la manipulation des pointeurs et la gestion de la mémoire, nous avons pu appréhender ces concepts dans un contexte concret. C'était pour nous une opportunité d'appliquer plus en détails les concepts clés du langage C vus en cours.

Des améliorations futures à ce projet pourraient inclure l'ajout d'une interface graphique pour rendre le jeu plus interactif, la gestion des sauvegardes pour reprendre une partie, ainsi que l'intégration de nouveaux niveaux ou ennemis pour enrichir l'expérience.