

A dark blue vertical bar runs along the left edge of the page. A blue arrow points to the right from this bar, containing the text '2024-2025'.

2024-2025

Rapport de projet - Darkest C Dungeon

Several thin, curved lines in dark blue and light grey originate from the bottom left corner and sweep upwards and to the right.

Vincent PLESSY

Table des matières

1. Présentation du projet	2
2. Instructions de compilation et d'exécution	2
Compilation	2
Exécution	2
3. Architecture du code	2
4. Structures de données	3
Structures principales	3
5. Fonctions principales	4
Gestion des personnages	4
Système de combat	4
Gestion des accessoires	4
Sauvegarde et chargement	4
6. Étapes du programme principal	4
7. Difficultés rencontrées	5
8. Réflexion sur l'utilisation des listes chaînées	5
Avantages des listes chaînées	5
Alternatives possibles	5
9. Perspectives d'amélioration	5

1. Présentation du projet

Le projet "Darkest C Dungeon" est une adaptation simplifiée du jeu Darkest Dungeon en langage C. C'est un jeu de rôle tour par tour où le joueur dirige une équipe de héros qui doivent combattre des ennemis de plus en plus puissants à travers 10 niveaux.

Les caractéristiques principales du jeu sont :

- Gestion d'une équipe de héros avec différentes classes
- Système de combat au tour par tour
- Gestion des points de vie et du stress des personnages
- Système d'équipement avec des accessoires
- Lieux de repos (sanitarium et taverne) pour récupérer entre les combats
- Système de sauvegarde et chargement de partie

2. Instructions de compilation et d'exécution

Compilation

Le projet utilise un Makefile pour la compilation. Les commandes sont :

```
# Compiler le projet
make

# Nettoyer les fichiers compilés
make clean
```

Exécution

```
# Lancer le jeu
./game
```

3. Architecture du code

Le projet est divisé en plusieurs fichiers pour une meilleure organisation :

- **structures.h** : Définition des structures de données principales
- **character.h/c** : Gestion des personnages
- **combat.h/c** : Système de combat
- **accessory.h/c** : Gestion des accessoires
- **save_load.h/c** : Système de sauvegarde/chargement
- **main.c** : Programme principal

4. Structures de données

Structures principales

```
// Type de classe (enumération)
typedef enum {
    CLASS_FURIE,
    CLASS_VESTALE,
    CLASS_CHASSEUR_DE_PRIMES,
    CLASS_MAITRE_CHIEN
} ClassType;

// Structure de classe
typedef struct {
    ClassType type;
    int att;
    int def;
    int HPmax;
    int rest;
} Class;

// Structure de personnage
typedef struct Character {
    char name[50];
    Class class;
    int HP;
    int stress;
    Accessory *acc1;
    Accessory *acc2;
    int nbcomb;
    int is_defending;
    struct Character *next;
} Character;

// Structure d'accessoire
typedef struct Accessory {
    char name[50];
    int attbonus;
    int defbonus;
    int HPbonus;
    int restbonus;
    int strred;
    int price;
    struct Accessory *next;
} Accessory;

// État global du jeu
typedef struct {
    int current_level;
    int gold;
    Character *available_characters;
    Character *sanitarium_characters;
    Character *tavern_characters;
    Character *fighting_characters;
    Accessory *available_accessories;
    Accessory *shop_accessories;
```

```
} GameState;
```

5. Fonctions principales

Gestion des personnages

- `Character* create_character(const char* name, ClassType class_type)`
- `void display_character(const Character* character)`
- `Character* add_character_to_list(Character* list, Character* new_char)`
- `Character* remove_character_from_list(Character* list, Character* char_to_remove)`

Système de combat

- `void start_combat(GameState* state, Enemy* enemy)`
- `void perform_enemy_action(Enemy* enemy, Character* fighters)`
- `void apply_damage(Character* target, int damage)`
- `void apply_healing(Character* target, int healing)`
- `void apply_stress(Character* target, int stress, int stress_resistance)`

Gestion des accessoires

- `Accessory* create_accessory(const char* name, int attbonus, int defbonus, int HPbonus, int restbonus, int stred)`
- `Accessory* add_accessory_to_list(Accessory* list, Accessory* new_acc)`
- `Accessory* remove_accessory_from_list(Accessory* list, Accessory* acc_to_remove)`

Sauvegarde et chargement

- `int save_game(const char* filename, GameState* state)`
- `GameState* load_game(const char* filename)`

6. Étapes du programme principal

1. **Initialisation**
 - Mise en place du générateur de nombres aléatoires
 - Création de l'état initial du jeu
 - Menu principal (Nouveau jeu/Charger/Quitter)
2. **Boucle de jeu principale**
 - Affichage du niveau actuel
 - Gestion du sanitarium et de la taverne
 - Sélection des combattants
 - Combat
 - Gestion de l'après-combat
 - Option de sauvegarde
 - Passage au niveau suivant
3. **Fin de partie**
 - Victoire (niveau 10 complété)
 - Défaite (tous les personnages morts)
 - Libération de la mémoire

7. Difficultés rencontrées

1. Gestion de la mémoire

- La gestion des listes chaînées a nécessité une attention particulière pour éviter les fuites mémoire
- Le transfert des personnages et accessoires entre différentes listes a demandé une gestion précise des pointeurs

2. Système de sauvegarde

- La sérialisation et désérialisation des structures complexes
- La gestion des chaînes de caractères dans le fichier de sauvegarde

3. Équilibrage du jeu

- Ajustement des statistiques des personnages et ennemis
- Équilibrage des mécaniques de stress et de combat

8. Réflexion sur l'utilisation des listes chaînées

Avantages des listes chaînées

1. Flexibilité

- Ajout et suppression dynamique d'éléments sans réallocation
- Facilité de transfert des éléments entre différentes listes

2. Gestion de la mémoire

- Allocation dynamique selon les besoins
- Pas de limite de taille prédéfinie

3. Adaptation au contexte du jeu

- Correspond bien au besoin de déplacer les personnages entre différents états
- Facilite la gestion des accessoires équipés/disponibles

Alternatives possibles

1. Tableaux dynamiques

- Plus simple à implémenter
- Accès direct aux éléments
- Mais nécessite des réallocations fréquentes

2. Tableaux fixes

- Plus simple encore
- Mais limite le nombre maximum d'éléments
- Moins flexible pour les transferts

La solution avec listes chaînées, bien que plus complexe à implémenter, offre une meilleure flexibilité et correspond bien aux besoins du jeu.

9. Perspectives d'amélioration

1. Fonctionnalités additionnelles

- Système de niveau pour les personnages
- Plus de classes et d'ennemis

- Système de combat plus complexe
- 2. **Améliorations techniques**
 - Interface graphique avec la bibliothèque MLV
 - Optimisation de la gestion mémoire
 - Meilleure gestion des erreurs
- 3. **Équilibrage**
 - Ajout de plus de variété dans les accessoires
 - Amélioration du système de difficulté
 - Plus de stratégies de combat possibles