Documentation technique pour le fichier jeu.c

Nous sommes Ilyass et Bryan, étudiant en première année à l'ESIPE, spécialité informatique. Voici la documentation technique de notre fichier jeu.c pour le jeu de type Shoot'em up en temps réel.

Ce fichier contient plusieurs fonctions pour gérer différents aspects d'un jeu, dont l'écriture des scores, la libération de mémoire, la vérification de la présence d'ennemis et le gestion du timing d'apparition des ennemis.

Constantes

La constante `LIFE_WEIGHT` est utilisée pour pondérer le nombre de vies restantes lors du calcul du score. De même, `ENEMY_WEIGHT` pondère le nombre d'ennemis tués et `TIME_WEIGHT` le temps écoulé.

Principales fonctions

Voici un aperçu des principales fonctions et de leur utilisation :

logique du jeu

- `int enemy_present(Enemy* new_enemy)`: cette fonction vérifie si un ennemi est présent. Si l'ennemi est NULL ou hors de l'écran, la fonction renvoie 0, sinon elle renvoie 1.
- `void change_direction(Enemy* new_enemy)`: cette fonction change la direction d'un ennemi quand il atteint un bord de l'écran. Si l'ennemi atteint le bord droit de l'écran, sa direction est changée vers la gauche, et vice versa.
- `void update_hearts(Player* player)`: cette fonction met à jour les coeurs (représentant les vies) du joueur. Elle parcourt la liste des coeurs du joueur et libère ceux qui sont en dehors de la fenêtre.
- `int enemy_disappears(Enemy*** enemies_ptr, int *time_passed, int* duration, Player* new_player)`: cette fonction vérifie si un ennemi est arrivé au bas de l'écran. Si c'est le cas, elle libère la mémoire associée à cet ennemi et réinitialise le compteur de temps. Si tous les ennemis ont été détruits, elle réinitialise également le tableau des ennemis.
- `int enemy_timer(int *duration)`: cette fonction gère le timing d'apparition des ennemis. Elle incrémente le compteur de durée jusqu'à 500, puis renvoie 1 pour indiquer qu'un nouvel ennemi doit être créé.
- `Fire* delete_fire(Fire* current_fire, Fire** enemy_fire)` : cette fonction supprime un tir d'ennemi en le libérant de la mémoire et en réalignant les pointeurs dans la liste chaînée de tirs. Si le tir à supprimer est le premier de la liste, le pointeur `enemy_fire` est mis à jour. Sinon, la fonction parcourt la liste pour trouver le tir précédent et mettre à jour son pointeur `next`.
- `void game_over(Player* player)` : cette fonction gère la fin de la partie. Elle affiche un message à l'écran indiquant que le joueur a perdu, écrit le score final du joueur dans un fichier, puis attend que l'utilisateur appuie sur une touche avant de libérer la fenêtre MLV et de quitter le programme.

Gestions des collisions

- `void check_collision_fire_enemy(Fire* fire, Enemy** enemies, int num_enemies, Player* new_player)` : cette fonction vérifie si un tir du joueur touche un ennemi. Si c'est le cas, elle diminue le nombre de vies de l'ennemi, met à jour le score du joueur, et supprime le tir et éventuellement l'ennemi. La suppression d'un ennemi est effectuée en libérant sa mémoire et en mettant à NULL le pointeur correspondant dans le tableau des ennemis.
- `void check_collision_player_enemy(Player* player, Enemy** enemies, int num_enemies)` : cette fonction vérifie si le joueur entre en collision avec un ennemi. Si c'est le cas, elle diminue le nombre de vies du joueur, met à jour son score, et supprime l'ennemi en libérant sa mémoire et en mettant à NULL le pointeur correspondant dans le tableau des ennemis. Si le joueur n'a plus de vies, la fonction `game_over()` est appelée.
- `void check_collision_enemy_fire_player(Enemy** enemies, int num_enemies, Player* player)` : cette fonction vérifie si le joueur entre en collision avec un tir d'ennemi. Si c'est le cas, elle diminue le nombre de vies du joueur et supprime le tir de l'ennemi en utilisant la fonction `delete_fire()`. Si le joueur n'a plus de vies, la fonction `game_over()` est appelée.
- `void check_collision_player_heart(Player* player)` : cette fonction vérifie si le joueur récupère un cœur (représentant une vie supplémentaire). Si c'est le cas, elle augmente le nombre de vies du joueur et supprime le cœur en libérant sa mémoire et en mettant à jour le pointeur `next` du cœur précédent ou du pointeur de tête de liste si le cœur à supprimer était le premier de la liste.
- `int collides_with(int x1, int y1, int width1, int height1, int x2, int y2, int width2, int height2)`: cette fonction vérifie si deux rectangles se chevauchent. Elle prend en entrée les coordonnées (x, y) et les dimensions (largeur, hauteur) de deux rectangles et renvoie 1 si ces rectangles se chevauchent, sinon 0.

Autres fonctions

- `void write_score_to_file(Player* player)`: cette fonction enregistre le score d'un joueur dans un fichier. Elle prend en entrée un pointeur vers l'objet `Player` et ajoute le score, le nombre de vies restantes et le nombre d'ennemis tués à un fichier nommé "scores.txt". Si le fichier ne peut pas être ouvert pour une raison quelconque, un message d'erreur est affiché.
- `void free_fire(Fire* fire)`, `void free_player(Player* new_player)`, et `void free_enemy(Enemy* new_enemy)`: ces fonctions libèrent la mémoire associée à leurs objets respectifs. Pour chaque objet, elles libèrent les images et les structures liées. Si l'objet passé en argument est NULL, ces fonctions n'effectuent aucune action.