Alex Crozat & Sarah Nguyen

Projet de Programmation C : Jeu d'infiltration
Documentation technique

Geometry

Ce module regroupe toutes les fonctionnalités de géometrie.

Structures :

La structure **Point** est représentée par les coordonnées **x** et **y**.
La structure **Coords** représente les coordonnées **i** et **j**.

Fonctions:

- **double euclidian_distance(double x_o, double y_o, double x, double y)** : Calcule la distance euclidienne entre les points O (x_o,y_o) et M (x,y).

L3_random

Ce module regroupe toutes les fonctions de générations de nombres aléatoires.

Fonctions:

- **int init_random()** : Initialise une librairie aléatoire.

- **float random_seed()** : Retourne une graine entre 0 et 1.

- **int L3_randint(int min, int max)** : Retourne un entier entre min et max inclus.

- **float L3_random(float min, float max)** : Retourne un flottant entre min et max.

Entity

Ce module comprend toutes les fonctions d'une entité que ce soit un joueur ou un gardien.

Macros:

- SPEED : la vitesse à 20
- STEPS : 60 calculs par seconde

Gestion de la vitesse du joueur :
- **PLAYER_MIN_SPEED** : vitesse initiale du joueur à 0,1
- **PLAYER_MAX_SPEED** : vitesse maximale du joueur à 0,9
- **PLAYER_MAX_OVERDRIVE_SPEED** : vitesse maximale du joueur à 1,2 en accélération surchargée

- **PLAYER_ACCELERATION** : accélération par frame à 0,03

Gestion du gardien :
- **MIN_GUARDIAN_SPAWN_RANGE** : distance euclidienne d'au moins 20 cases de la position initiale du personnage
- **GUARDIAN_SIGHT** : champ de vision du gardien à 4
- **GUARDIAN_PANIC_SIGHT** : champ de vision du gardien élargie à 6
- **GUARDIAN_MIN_SPEED** : vitesse minimale du gardien à 0.3
- **GUARDIAN_MAX_SPEED** : vitesse maximale du gardien à 0.8
- **PANIC_GUARDIAN_SPEED** : vitesse en mode panique à 1

- **GUARDIAN_TURNS** : tour du gardien mis à 1/50

Structures :

La structure **Entity** permet de représenter le joueur ainsi que les gardiens.
Chacune est définie par :
- une **direction** : étant une énumération indiquant le cardinal (NORTH, SOUTH, EAST, WEST)
- un **type** : une énumération indiquant son type (**PLAYER** : joueur, **GUARDIAN** : guardien, **PANIC_GUARDIAN** : guardien en mode panique)
- une coordonnée en abscisse **x**
- une coordonnée en abscisse **y**
- son rayon **r** : doit être inférieur ou égal à 1/2
- sa vitesse **speed**

L'énumération **MoveType** indique à quel état on se situe :
- **MOVE_START** : démarrage ou changement de direction
- **SPEED_UP** : accélération quand la touche de direction reste enfoncée
- **OVERDRIVE** : accélération surchargé

Fonctions:

- **Entity * init_entity(Type type)** : Initialise une entité.
Alloue la mémoire pour Entity. Initialise les données x, y, speed et direction en fonction du type entré en paramètre.

- **int update_entity_position(Entity* e, Point p_col)** : Met à jour la position de l'entité.
Actualise la position de l'entité en faisant avancer les coordonnées x et y dans sa direction et vérifie qu'il n'y a pas de collision avec un mur.

- **int check_collision(Entity* e, Point p_col)** : Vérifie si l'entité entre en collision avec le point entré en paramètre.

- **Point collision_bound(Terrain* t, Entity* e)** : Gère la collision entre l'entité et les murs du terrain en corrigeant la position de l'entité. Retourne les coordonnées du prochain point non traversable ou (0,0) s'il n'y a pas de contraintes de mouvement.

- **int place_guardian(Terrain* board, Entity* e)** : Place l'entité gardien sur le terrain.
Attribue à gardien des coordonnées x et y aléatoires sur le terrain .

- **int change_guardian_trajectory(Entity* e)** : Change la trajectoire du gardien de manière aléatoire.

- **int move_normal_guardian(Entity* e, Point p_col)** : Fait bouger le gardien en mettant à jour sa position (update_entity_position) et sa trajectoire (change_guardian_trajectory).

- **int move_panic_guardian(Entity* e, Point p_col)** : Fait bouger le gardiene mode panique en mettant à jour sa position (update_entity_position) et sa trajectoire (change_guardian_trajectory).

Terrain

Ce module comprend toute la gestion du terrain de jeu.

Macros:

- **TERRAIN_WIDTH**: largeur du terrain à 60
- **TERRAIN_HEIGHT**: hauteur du terrain à 45
- **MINSIDE** : la taille minimale d'une pièce à 9
- **DOOR_WIDTH**: doit être inférieur à MINSIDE
- **RELIC_NUMBER**: nombre de relique à 3
- **MIN_RELIC_SPAWN_DISTANCE**: distance minimale entre les reliques à 30

Structures :

La structure **Terrain** représente la salle par :
- la salle composée par des tuiles : **tiles**
- un DynamicArray de **collected_mana** : où sont enregistrés les coordonnées des tuiles de mana collectées (voir le module DynamicArray)
- un DynamicArray de **collected_relics** : où sont enregistrés les coordonnées des tuiles de reliques collectées

Il existe différents types de tuile nommé **tile** :
- **GROUND**
- **WALL** : un mur
- **MANA** : une trace de mana
- **ENTRANCE** : l'entrée où le joueur se situe en début de partie et où il doit revenir où achever la partie
- **RELIC** : une relique
- **STOLEN_RELIC** : une relique collectée
- **RELIC_CANDIDATE** : une des tuiles pouvant devenir une relique

Fonctions:

- **Terrain * init_terrain()** : Initialise un terrain.
Alloue de la mémoire pour Terrain, collected_mana.array et stolen_relics.array.

- **void cut_along_max_length(Terrain* terrain, int L_i, int l_i, int L_j, int l_j, int horizontal)** : Génère deux sous-pièces selon le côté le plus long de la pièce, en dessinant un mur intérieur avec une ouverture DOORSIZE.

- **int terminate_room_generation(Terrain* terrain, int x_i, int y_i, int x_j, int y_j)** : Met fin à la génération de la salle.

- **int make_inner_walls(Terrain* terrain, int x_i, int y_i, int x_j, int y_j)** : Vérifie si l'algorithme de génération de pièce se termine et détermine le côté le plus long de la pièce actuelle.

- **int generate_relics(Terrain* terrain)** : Génère les reliques à des lieux disponibles d manière aléatoire.

- **int generate_mana(Terrain* terrain, int nb_mana)** : Génère le nombre de mana donné à des lieux aléatoires.

- **int generate_terrain(Terrain *terrain)** : Génère les murs intérieurs.

- **int point_spotted(float x_g, float y_g, float x_p, float y_p, Terrain* t)** : Vérifie si le point p (x_p, y_p) peut être repéré par le point g-x_g, y_g) dans le terrain.

Dynamic_array

Ce module comprend toutes les fonctions d'une liste dynamique.

Structures :

La structure **DynamicArray** (utilisée dans le module Terrain) est représentée par (:
- une liste de Coords nommée **array** : coordonnées (voir module Geometry)
- **size** : la taille de la liste précédente

Fonctions:

- **int push(Coords coords, DynamicArray * s)** : Ajoute point dans la liste.
Actualise la taille.

- **int pop(Coords * coords, DynamicArray * s, int index)** : Supprime un point de la liste à l'indice donné et la sauvegarde dans coords. Actualise la taille.

- **int pop_random(Coords * coords, DynamicArray * s)** : Supprime un point aléatoirement.

- **void print_dynamic_array(DynamicArray s)** : Affiche en console la liste.

Game

Ce module comprend toute la gestion du jeu.

Macros:

- GUARDIANS_NUMBER : nombre de gardiens à 5
- PANIC_TIME : durée du mode panique à 30 secondes

Structures :

La structure du jeu **Game** est représentée par :
- la réserve de **mana**
- le temps écoulé **time**
- le nombre de relique encore disponible dans le jeu **available_relics**
- un entier **invisibility** : indiquant si l'invisibilité du joueur est déclenchée
- un entier **overdrive** : indiquant si l'accélération surchargée est déclenchée
- **panic_steps** : nombre de pas de calcul (frames par seconde) de la durée du mode panique
- l'entité **player** : représentant le joueur (voir module Entity)
- la liste de gardien nommée **guardians**: représentant les entités gardiens (voir module Entity)
- le **terrain** (voir module Terrain)

Fonctions:

- **Game * init_game()** : Initialise le jeu.
Alloue de la mémoire pour jeu, initialise random, le terrain, le joueur, les gardiens, ainsi que les autres valeurs.

- **void free_game(Game * g)** : Libère la mémoire alloué par Game.

- **int collect(Game * g)** : Met à jour les tuiles collectés par le joueur.
Les tuiles contenant autrefois du mana maintenant collectés deviennent alors des tuiles GROUND, et les tuiles de reliques deviennent STOLEN_RELIC.

- **int move_player(Game * g, Cardinal direction)** : Ajuste la vitesse et la position de l'entité joueur en appelant la fonction player_manage_speed() et update_entity_position en prenant en compte les collisions en appelant collision_bound(g->terrain, g→player) avant.

- **int player_manage_speed(Game * g, int choice)** : Gère la vitesse du joueur. Pour cela, il vérifie le choix :
- s'il s'agit d'une redirection ou d'un départ (MOVE_START) il réinitialise la vitesse à son minimum,
- s'il doit accélérer (SPEED_UP) il vérifie s'il est encore possible d'accélérer sinon il atteint la vitesse maximum
- s'il passe en accélération surchargée (OVERDRIVE), il vérifie que le joueur possède suffisamment de mana et qu'il n'est pas à la vitesse maximale en overdrive pour accélérer, consommer 2 manas et appeler generate_mana() (voir module terrain), sinon si la vitesse du joueur dépasse la vitesse max en overdrive on la corrige.

- **void trigger_panic(Game * g)** : Tourne les gardiens en mode panique.

- **void toggle_normal(Game * g)** : Retourne les gardiens à leur forme normale.

- **int check_win(Game* g)** : Verifie les conditions de victoire : c'est à dire si le joueur se situe bien à l'entrée et qu'il n'y a plus de reliques disponibles dans le jeu.
- **int guardians_vision(Game* g)** : Gère la vision des gardiens des joueurs et des reliques volées.

Leaderboard

Macros:
- MAX_CHARACTER_NAME : nombre de caractères maximum pour le nom du joueur à 20
- NUMBER_MAX_LEADER_DISPLAY : top 15 affiché dans le tableau

Structures :
La structure **Leader** est composée par les informations d'affichage d'une partie gagnée par un joueur dont :
- le **score**
- le temps **time**
- son nom **name**

La structure **Leaderboard** représente le tableau de classement composée de :
- une liste **array** de Leader
- la taille de la liste **array**, **ranking_score_index** et **ranking_time_index**
- une liste d'indices **ranking_score_index** : où sont rangés les indices des différents Leader par ordre de classement (score décroissant)
- une liste d'indices **ranking_time_index** : où sont rangés les indices des différents Leader par meilleur temps (temps croissant)

Fonctions:
- **Leaderboard * init_leaderboard()** : Initialise un leaderboard vide.
Alloue la mémoire pour Leaderboard, ainsi que pour les listes ranking_score_index et ranking_time_index.

- **Leader * init_leader(int score, int time, char * name)** : Initialise un leader.
Alloue la mémoire pour un Leader et initialise son score, temps et nom.

- **void free_leaderboard(Leaderboard * b)** : Libère la mémoire alloué par Leaderboard.

- **int add_new_leader(Leaderboard * board, int score, int time, char ** name)** : Ajoute un nouveau Leader dans Leaderboard.
Comme name est un pointeur où est stocké le nom lue en mémoire à la lecture du fichier, on utilise memcpy() pour la copier à une autre adresse mémoire propre à elle. Ensuite, on ajoute son indice dans ranking_score_index et ranking_time_index qui ne sont pas encore triés pour l'instant.

- **int read_leaderboard(Leaderboard * board)** : Affiche en console les données dans Leaderboard.

- **int read_ranking_score_index(Leaderboard * board)** : Affiche en console l'indice et le score du tableau ranking_score_index.

- **int read_ranking_time_index(Leaderboard * board)** : Affiche en console l'indice et le temps du tableau ranking_time_index.

- **int rank_leaders_score(Leaderboard * board)** : Trie le tableau ranking_score_index par meilleur score.

- **int rank_leaders_time(Leaderboard * board)** : Trie le tableau ranking_time_index par meilleur temps.

- **FILE * get_file(const char * path, const char * accessMode)** : Ouvre le fichier.

- **int save_data_in_file(char * name, int score, int time)** : Sauvegarde le nom, score et temps dans en écrivant dans le fichier.

- **int read_data_in_file(Leaderboard * board)** : Lit les données du fichier et sauvegarde les données en appelant add_new_leader.

Display

Ce module regroupe toutes les fonctions utilisant la bibliothèque MLV pour l'initialisation et l'affichage de la fenêtre ainsi que la gestion des événements du clavier et de la souris.

Macros:

Couleurs utilisés disponibles dans la bibliothèque MLV :
- GREEN : MLV_COLOR_GREEN
- BLUE : MLV_COLOR_BLUE
- RED : MLV_COLOR_RED
- GRAY : MLV_COLOR_DARK_GREY
- BLACK : MLV_COLOR_BLACK
- CYAN : MLV_COLOR_DARK_TURQUOISE
- MOSS_GREEN : MLV_COLOR_SEA_GREEN
- GOLD : MLV_COLOR_GOLD
- BROWN : MLV_COLOR_BROWN
- RED_PANIC : MLV_COLOR_DARK_RED

Affichage du jeu :
- **TILE_SIZE** : taille d'une tuile intitialisé à 10 (vous pouvez la modifier pour changer la taille de la fenêtre de jeu)
- **FRAMES_PER_SECOND** : nombre de frame par seconde égal à STEPS

Touches du clavier :
- DIRECTIONAL_KEYS_NUMBER : nombre de touches directionnelles (ZQSD + les touches directionnelles) au total : 8

- Z_KEY : touche z (MLV_KEYBOARD_z)
- Q_KEY : touche q (MLV_KEYBOARD_q)
- S_KEY : touche s (MLV_KEYBOARD_s)
- D_KEY : touche d (MLV_KEYBOARD_d)
- UP_KEY : touche directionnelle vers le haut (MLV_KEYBOARD_UP)
- DOWN_KEY : touche directionnelle vers le bas MLV_KEYBOARD_DOWN)
- LEFT_KEY : touche directionnelle vers la gauche (MLV_KEYBOARD_LEFT)
- RIGHT_KEY : touche directionnelle vers la droite (MLV_KEYBOARD_RIGHT)
- LSHIFT_KEY : touche shift de gauche (MLV_KEYBOARD_LSHIFT)
- RSHIFT_KEY : touche shift de droite (MLV_KEYBOARD_RSHIFT)
- SPACE_KEY : barre d'espace (MLV_KEYBOARD_SPACE)

Bouton start
- MENU_BUTTON_PLAY_X : coordonnée x du bouton
- MENU_BUTTON_PLAY_Y : coordonnée y du bouton
- MENU_BUTTON_PLAY_WIDTH : largeur du bouton
- MENU_BUTTON_PLAY_HEIGHT : hauteur du bouton

Affichage
- **MARGIN** : marge

Fonctions:

- **int init_window()** : Crée la fenêtre de jeu.

- **int draw_tile(int i, int j,MLV_Color color)** : Dessine la tuile dans la fenêtre de jeu.

- **int draw_grid(MLV_Color grid_color)** : Dessine la grille de jeu qui sépare les tuiles.

- **int display_menu()** : Affiche le menu : dessine le bouton start.

- **int display_game_win(char ** name)** : Affichage en cas de victoire et demande son pseudo. Le pseudo est ensuite enregistré dans name.

- **int display_game_lost()** : Affichage en cas de défaite.

- **int display_ranking_score(Leaderboard * board)** : Affiche les tableaux de classement par score et par temps pour le top 15 (valeur initialisé dans NUMBER_MAX_LEADER_DISPLAY) ou moins s'il n'y a pas eu assez de victoire enregistrée.

- **int display_game_information(Game * g)** : Affiche les informations du jeu : score et temps pendant la partie.

- **int display_terrain(Terrain * terrain)** : Affiche le terrain.

- **int display_entity(Entity* e)** : Affiche l'entité. S'il s'agit d'un gardien, il affichage aussi son champ de vision, de même pour le mode panique.

- **int WaitClick()** : Fonction qui attend un click sur le bouton « start » dans le menu.

- **int WaitClickOnBox(int x0, int y0, int w, int h)** : Fonction qui attend un clic entre les coordonnés données.
- **int check_2_inputs(MLV_Keyboard_button input1, MLV_Keyboard_button input2, int * param)** : Met à jour le paramètre param en fonction de l'état des touches input1 et input2.

- **int check_input(MLV_Keyboard_button input, int * param)** : Met à jour param en fonction de l'état de la touche input

- **int released_move_key(Game * g, MLV_Button_state state)** : Vérifie si la touche est pressée en renvoyant 0 et 1 sinon.

- **int control_player_with_keyboard(Game * g)** : Gère le contrôle du clavier mettant à jour le jeu. Il vérifie si une touche est pressée, appelle check_2_inputs() pour vérifier si l'accélération surchargé est appelée, apelle check_input() pour vérifier si l'invisibilité est appelé. Dans le cas où l'invisibilité est déclenché, il vérifie que le joueur possède assez de mana et génère les manas (generate_mana) si c'est le cas, sinon il annule l'invisibilité.
La fonction collect() (voir module game) est ensuite appelée pour mettre à jour les tuiles.
Enfin, il vérifie si une touche directionnelle est pressée pour modifier la direction du joueur (move_player voir module game).

- **void color_background(MLV_Color color, Terrain* t)** : Dessine les murs et la grille avec la nouvelle couleur.

Stealth

Fichier à exécuter.

Main:

Initialisation des variables du jeu :
- **frametime** et **extratime** : sont utilisés pour gestion du framerate.
- **quit** : initialisé à 0 permet rester dans la partie.
- **win** : initialisé à 0 au départ indique le résultat de la partie : 1 si gagné, 0 sinon.
- **n** : indice utilisé dans la boucle pour les gardiens.
- **name** : utilisé pour enregistré temporairement le pseudo du joueur.
- **board** : initialisation de Leaderboard à NULL pour l'instant.

Initialisation du jeu :
- **end_time** et **new_time**: sont utilisés pour gestion du framerate.
- initialise le jeu : **game**
- initialise la fenêtre

Menu :
- affiche le menu
- actualise l'affichage de la fenêtre

- reste dans la fenêtre tant que le bouton « start » n’a pas été cliqué

Boucle de jeu :
- dans la boucle while(!quit)

Affichage de fin :
- en cas de victoire : appelle display_game_win() pour obtenir le pseudo du joueur, les fonctions de leaderboard et affiche le classement.
- en cas d’echec affiche display_game_lost().

Free :
Libère les places en mémoire.