

Rendu : Projet de RPG

Règles du jeu :

- Vaincre les ennemis
- Sortir du donjon en allant jusqu'à la sortie

Dans notre jeu, vous retrouverez 3 symboles principaux :

Votre joueur :



Les ennemis :



La sortie :



Problématiques :

- Comment créer la carte ?
- Comment faire en sorte que le joueur puisse se déplacer ?
- Comment faire pour avoir des combats ?

Problèmes précis :

- Comment faire une carte sans avoir besoin de préparer plusieurs salles à l'avance directement à la main ?

Solution proposée :

Pour la solution, nous avons décidé au début de créer 15 types de salles (toutes les salles possibles) qu'on sélectionnerait en fonction des précédentes (avec la première salle sélectionnée aléatoirement) afin d'avoir une carte générée aléatoirement.

Ces 15 salles étaient les suivantes :



Comme le procédé aurait été long, nous nous sommes alors posé la question précédente. Nous avons finalement décidé de séparer la création des salles et des couloirs afin que ce soit beaucoup plus simple à créer : une fonction pour créer les salles et 2 autres fonctions pour créer les couloirs verticaux et horizontaux.

Fonctions pour les salles :

```
47
48 /* entoure la salle de MURs et la remplit de VIDE */
49 void initialise_salle(struct salle* s)
50 {
51     // murs :
52     for (int col = 0; col < SAL_COL; col++) // murs horizontaux
53     {
54         s->contenu[0][col] = MUR; // haut
55         s->contenu[SAL_LIN - 1][col] = MUR; // bas (- 1 pour ne pas sortir de la salle)
56     }
57     for (int lin = 0; lin < SAL_LIN; lin++) // murs verticaux
58     {
59         s->contenu[lin][0] = MUR; // gauche
60         s->contenu[lin][SAL_COL - 1] = MUR; // droite
61     }
62
63     // vide :
64     for (int lin = 1; lin < SAL_LIN - 1; lin++) // valeurs extrêmes non comprises car des murs s'y trouvent
65     {
66         for (int col = 1; col < SAL_COL - 1; col++) // idem
67         {
68             s->contenu[lin][col] = VIDE;
69         }
70     }
71     return;
72 }
73
74
75 /* ajoute les éléments de la salle dans le map */
76 void ajoute_salle(struct salle s, int map[MAP_LIN][MAP_COL])
77 {
78     // empêche de placer les éléments de la salle à des endroits trop loin dans le map
79     assert(s.pos_lin >= 0 && s.pos_lin + SAL_LIN <= MAP_LIN);
80     assert(s.pos_col >= 0 && s.pos_col + SAL_COL <= MAP_COL);
81
82     for (int lin = 0; lin < SAL_LIN; lin++) // lignes de la salle
83     {
84         for (int col = 0; col < SAL_COL; col++) // colonnes de la salle
85         {
86             map[s.pos_lin + lin][s.pos_col + col] = s.contenu[lin][col];
87         }
88     }
89     return;
90 }
91
92 }
```

Nous avons écrit une fonction qui crée la salle (**initialise_salle()**) puis une autre fonction qui la place dans la carte (**ajoute_salle()**). Afin de placer toutes les salles, nous avons écrit **creation_salles()** qui, à l'aide de plusieurs boucles, va créer toutes les salles et les placer automatiquement aux bonnes coordonnées en utilisant ces 2 fonctions.

Fonctions pour les couloirs :

```
119
120 /* ajoute les couloirs horizontaux, et un ennemi devant
121  * chaque couloir si le paramètre ennemi vaut true */
122 void ajoute_couloirs_h(int map[MAP_LIN][MAP_COL], bool ennemi)
123 {
124     // seuil (colonne) à ne pas dépasser pour ne pas sortir de la map :
125     int seuil_col = MAP_COL - SAL_COL - LONG_COULOIR_H;
126     // décalage (colonne) d'un couloir au suivant (- 2 car le couloir commence dans la salle) :
127     int decalage_col = LONG_COULOIR_H + SAL_COL - 2;
128     // décalage (ligne) d'un couloir au suivant (- 2 car le couloir commence dans la salle) :
129     int decalage_lin = LONG_COULOIR_V + SAL_LIN - 2;
130
131     int debut_coul_lin = ORI_COULOIR_H_LIN; // ligne de début du couloir actuel
132
133     // parcourt les rangées de couloirs horizontalement
134     for (int rangee = 0; rangee < calcul_nbr_rangee_couloirs(); rangee++)
135     {
136         // debut_coul_col : colonne du début du couloir actuel
137         for (int debut_coul_col = ORI_COULOIR_H_COL; debut_coul_col < seuil_col; debut_coul_col += decalage_col)
138         {
139             for (int col = 0; col < LONG_COULOIR_H; col++) // ajoute le couloir, colonne par colonne
140             {
141                 map[debut_coul_lin + 0][debut_coul_col + col] = MUR;
142                 map[debut_coul_lin + 1][debut_coul_col + col] = VIDE;
143                 map[debut_coul_lin + 2][debut_coul_col + col] = MUR;
144             }
145             if (ennemi) // place un ennemi devant le couloir
146             {
147                 map[debut_coul_lin + 1][debut_coul_col - 1] = ENNEMI;
148             }
149             debut_coul_lin += decalage_lin; // décale la ligne de début du couloir
150         }
151     }
152     return;
153 }
154 }
```

Chaque couloir (vertical ou horizontal) est bloqué par un ennemi. On peut désactiver l'apparition d'ennemis si on le souhaite.

Pour afficher les éléments de la carte, on change la valeur de la case que l'on veut dans le tableau map. Le tableau est ensuite affiché grâce à la fonction **affiche_map()** :

```
/* affiche la map au centre de l'écran, moins un décalage pour la zone de texte */
void affiche_map(int map[MAP_LIN][MAP_COL])
{
    // coordonnées du coin supérieur gauche de la map dans l'écran
    int start_lin, start_col;
    // centre du décalage entre la map et l'écran :
    start_lin = (SCR_LIN - MAP_LIN) / 2 - TAILLE_ZONE_TXT_LIN; // - TAILLE_ZONE_TXT_LIN : décalage
    start_col = (SCR_COL - MAP_COL) / 2;

    // l'affichage doit bien se faire dans l'écran
    assert(start_lin >= 0);
    assert(start_lin + MAP_LIN <= SCR_LIN);
    assert(start_col >= 0);
    assert(start_col + MAP_COL <= SCR_COL);

    int current_lin = start_lin; // sera itéré pour placer le caractère
    int current_col = start_col; // à cet emplacement sur l'écran

    /* affichage des éléments de la map */
    for (int lin = 0; lin < MAP_LIN; lin++) // lignes
    {
        for (int col = 0; col < MAP_COL; col++) // colonnes
        {
            int tap = map[lin][col]; // case actuelle

            affiche_char_val(tap, current_lin, current_col);

            current_col++; // colonne suivante
        }

        current_lin++; // ligne suivante
        current_col = start_col; // revenir à la première colonne
    }
    refresh();
    return;
}
```

→ Comment faire en sorte que le joueur puisse se déplacer sans traverser les objets et les murs ?

En ce qui concerne le déplacement et la collision :

```
86
87 /* Modifie la position du joueur en fonction de l'entrée au clavier (on suppose que
88  * le déplacement est valide) ; le cas échéant, lance un combat ou la fin du jeu */
89 void deplacement(struct joueur* j_ptr, int map[MAP_LIN][MAP_COL], int inputchar, struct ennemi* e_ptr)
90 {
91     // le joueur doit être dans la map
92     assert_joueur_dans_map(*j_ptr, map);
93
94     efface_zone_texte(); // pour pouvoir afficher les dialogues si besoin
95     map[j_ptr->pos_lin][j_ptr->pos_col] = VIDE; // efface le joueur de sa position
96
97     if (inputchar == KEY_UP) // flèche du haut pressée
98     {
99         j_ptr->dir = HAUT; // tourne le joueur vers le haut
100
101         if (test_elt_en_haut(*j_ptr, map, VIDE)) // si la voie est libre
102         {
103             j_ptr->pos_lin -- 1; // vrai déplacement
104         }
105         else if (test_elt_en_haut(*j_ptr, map, ENNEMI))
106         {
107             if (combat(j_ptr, e_ptr)) // lance un combat ; s'il est gagné :
108             {
109                 map[j_ptr->pos_lin - 1][j_ptr->pos_col] = VIDE; // remplace la case où était l'ennemi par du vide
110             }
111         }
112         else if (test_elt_en_haut(*j_ptr, map, END))
113         {
114             efface_ligne_texte(0);
115             efface_ligne_texte(1);
116             affiche_texte(1, 0, "Bravo, vous avez atteint la fin !");
117         }
118     }
119 }
```

On récupère la touche sur laquelle le joueur appuie. Dans cette capture d'écran, on a pris le cas où le joueur appuie sur la flèche du haut. On vérifie donc ce qu'il y a en face du joueur et on vérifie les différents cas possibles (c'est-à-dire : rien → on se déplace, ennemi → on enclenche le combat, sortie → on a fini le jeu).

Ces vérifications sont effectuées grâce à des fonctions test.

Exemple quand le joueur se déplace vers le haut :

```
/* renvoie true si l'element recherché est dans la case au-dessus du joueur */
bool test_elt_en_haut(struct joueur j, int map[MAP_LIN][MAP_COL], int element)
{
    // le joueur doit se trouver dans la map
    assert_joueur_dans_map(j, map);

    // cas où joueur en haut de la map
    if (j.pos_lin == 0)
        return true;

    // case au-dessus du joueur
    else if (map[j.pos_lin - 1][j.pos_col] == element)
        return true;

    else
        return false;
}
```

→ Comment faire pour avoir des combats ?

```
/* renvoie true si l'element recherché est dans la case au-dessus du joueur */
bool test_elt_en_haut(struct joueur j, int map[MAP_LIN][MAP_COL], int element)
{
    // le joueur doit se trouver dans la map
    assert_joueur_dans_map(j, map);

    // cas où joueur en haut de la map
    if (j.pos_lin == 0)
        return true;

    // case au-dessus du joueur
    else if (map[j.pos_lin - 1][j.pos_col] == element)
        return true;

    else
        return false;
}
```

Il a fallu modifier la fonction de détection pour qu'elle puisse détecter les ennemis.

```

/* Modifie la position du joueur en fonction de l'entrée au clavier (on suppose que
 * le déplacement est valide) ; le cas échéant, lance un combat ou la fin du jeu */
void deplacement(struct joueur* j_ptr, int map[MAP_LIN][MAP_COL], int inputchar, struct ennemi* e_ptr)
{
    // le joueur doit être dans la map
    assert_joueur_dans_map(*j_ptr, map);

    efface_zone_texte(); // pour pouvoir afficher les dialogues si besoin
    map[j_ptr->pos_lin][j_ptr->pos_col] = VIDE; // efface le joueur de sa position

    if (inputchar == KEY_UP) // flèche du haut pressée
    {
        j_ptr->dir = HAUT; // tourne le joueur vers le haut

        if (test_elt_en_haut(*j_ptr, map, VIDE)) // si la voie est libre
        {
            j_ptr->pos_lin -= 1; // vrai déplacement
        }
        else if (test_elt_en_haut(*j_ptr, map, ENNEMI))
        {
            pivote_joueur_vers_map(*j_ptr, map); // le joueur se tourne vers l'ennemi
            affiche_map(map); // et on l'affiche
            if (combat(j_ptr, e_ptr)) // lance un combat ; s'il est gagné :
            {
                map[j_ptr->pos_lin - 1][j_ptr->pos_col] = VIDE; // remplace la case où était l'ennemi par du vide
            }
        }
    }
}

```

Donc si un ennemi est détecté, on lance la séquence de combat, et si jamais l'ennemi est vaincu, on le remplace par une case de vide, libérant ainsi l'accès aux couloirs.

Le système de combat fonctionne au tour par tour, où le joueur et l'ennemi choisissent une action (attaquer ou se défendre). L'ennemi choisit une action de manière aléatoire.

```

if (entree == 'a') // le joueur attaque
{
    // Génère un nombre aléatoire : 0 (attaque) ou 1 (défense)
    action_ennemi = rand()&1;

    if (action_ennemi == 0) // l'ennemi attaque
    {
        efface_ligne_texte(2);
        efface_ligne_texte(3);
        affiche_texte(2, 0, "L'adversaire attaque aussi !");
        e->pv -= j->atk;
        j->pv -= e->atk;
        affiche_degats_subis(3, 'b', e->atk, j->atk);
    }
    else // l'ennemi se défend
    {
        efface_ligne_texte(2);
        efface_ligne_texte(3);
        affiche_texte(2, 0, "Vous attaquez, mais l'adversaire se défend.");
        e->pv -= (j->atk / 2);
        affiche_degats_subis(3, 'e', (j->atk / 2), 0);
    }
}

```

On diminue ensuite la vie du joueur et de l'ennemi en fonction des actions.

Et on répète ce procédé jusqu'à ce que l'un des deux combattants n'ait plus de vie.

```

if (j->pv <= 0) // défaite
{
    efface_ligne_texte(1);
    efface_ligne_texte(2);
    efface_ligne_texte(3);
    affiche_texte(2, 0, "Vous avez perdu...");
    j->pv = pv_max;           // En cas de défaite on rend les pv du joueur (peut être changé pour plus de difficulté)
    e->pv = pv_ennemi;       // En cas de défaite, on remets les pv de l'ennemi a son max (sinon trop facile)

    return false;           // Indique que le combat n'est pas gagné
}

/* On peut changer manuellement la difficulté du jeu en changeant combien de pv on gagne après une victoire */

else if (e->pv <= 0) // victoire
{
    efface_ligne_texte(1);
    efface_ligne_texte(2);
    efface_ligne_texte(3);
    affiche_texte(2, 0, "Vous avez gagné. Votre attaque augmente !");
    j->atk += 1;

    affiche_texte(3, 0, "Vous regagnez tous vos PV, et vos PV max. augmentent !");
    j->pv = pv_max;           // Remet les pv du joueur au niveau d'avant le combat
    j->pv += 5;               // rajoute 5 pv a la fin d'un combat
    e->pv = pv_ennemi + 5;    // la vie des ennemis augmente au fur et a mesure pour rajouter un peu de difficulté
    e->atk = e->atk + 5; //rand()&1; // Augmente de 1 ou de 0 l'attaque de l'adversaire pour corser le jeu

    return true;             // Renvoie true si le combat est gagné pour pouvoir supprimer l'ennemi de la map
}

```

Une fois le combat remporté, on redonne de la vie aux deux combattants, pour pouvoir continuer l'aventure et recombattre les ennemis plus loin. Il est donc à noter qu'il n'y a qu'un type d'ennemi sur la carte.

Bilan :

Ce qui fonctionne :

- notre personnage se déplace
- les collisions fonctionnent
- notre génération de carte fonctionne
- le système de combat fonctionne

Ce qui ne fonctionne pas :

- la génération aléatoire de carte n'a pas été faite

Pour continuer :

En ce qui concerne la génération aléatoire de carte, il faudra faire en sorte qu'il y ait toujours un chemin qui mène à la sortie et qu'il n'y ait pas de salles non connectées à la carte.

L'implémentation d'un algorithme de pathfinding permettrait d'ajouter cette fonctionnalité.

On pourrait également rajouter :

- Un système de niveau, avec des cartes de plus en plus grandes et des styles différents. Les fonctions employées dans le jeu sont assez modulaires, ce qui permettrait de ne pas être limités si nous voulions poursuivre le développement.
- Un système d'inventaire, avec différentes armes, des munitions et du mana. Encore une fois, notre système n'est pas figé et pourrait être étendu.