

Le Mans Université
Licence Informatique 2ème année
Module 174UP02 Rapport de projet
VoidBorn
Lien vers le Github

Baptiste M, Lucas R, Nathan M, Ilann T

3 avril 2025

Table des matières

1	Introduction	3
2	Analyse	3
3	Organisation du Projet	3
3.1	Planning Prévisionnel	3
3.2	Répartition des tâches	4
4	Développement	4
4.1	Création des ressources	4
4.1.1	Objet	5
4.1.2	Sprites et Personnages	5
4.1.3	Tuiles et Biomes	5
4.2	Implémentation	6
4.2.1	Map	6
4.2.2	Personnages	6
4.2.3	Compétences	7
4.2.4	Objets	7
4.2.5	Monstres	8
4.2.6	Évènement	9
4.3	Réseau	9
4.3.1	Networking	9
4.3.2	Client	10
4.3.3	Serveur	10
4.4	Rendu Graphique	10
4.4.1	Menu Principal	10
4.4.2	Jeu	14
5	Résultats et conclusion	16
6	Annexes	16
6.1	Débogage	16

1 Introduction

Nous sommes un groupe de quatre étudiants à l'université du Mans. Notre objectif à été de créer un jeu en respectant un délai d'environ 3 mois. Lors de notre phase de réflexion sur le jeu, nous souhaitions créer un jeu multijoueur en deux dimensions à l'aide de la librairie graphique SDL2 en langage C. Lors d'une partie de notre jeu, deux équipes s'affrontent pour détruire la base adverse tout en protégeant la sienne. Nous nous sommes fixé comme contrainte de développer ce jeu en réseau.

2 Analyse

Dans le menu, chaque joueur choisit son pseudo et la classe de son personnage, lui conférant des capacités uniques qu'il peut utiliser au cours de la partie pour prendre l'avantage sur l'équipe adverse. Dans le menu, vous avez le choix entre être le serveur de la partie, être joueur d'une partie ou être les deux à la fois.

Après cela, vous arrivez dans une salle d'attente pour que les autres joueurs se joignent à vous. Une fois que la personne servant de serveur lance la partie, le jeu se lance et il n'est plus possible de rejoindre la partie, même si vous étiez dedans, que vous quittez le jeu et que vous tentez de vous reconnecter.

La map de notre jeu se décompose en plusieurs parties que nous appelons tuiles, chacune ayant un biome(neige, plaine, etc) choisi aléatoirement.

Les joueurs ne peuvent visualiser qu'une tuile du jeu à la fois. Cependant, il est possible de changer de tuile en passant par des portes placées à différents endroits aux bords de la tuile à la manière de The Binding of Isaac.

Des obstacles sont générés aléatoirement sur la map, limitant les déplacements du joueur. Des monstres sont également présents sur la map, se dirigeant vers le joueur le plus proche pour l'attaquer. Lorsque le monstre est tué il lâche un objet que le joueur peut rammasser et ajouter à son inventaire.

Chaque objet a une rareté et confère au joueur différents effets. Plus la rareté de l'objet est élevée, plus les effets bonus conférés au joueur sont importants.

Un système de combat est mis en place entre les joueurs d'équipes différentes lorsqu'ils se situent sur la même tuile. Les points de vie ainsi que les objets du joueur sont affichés sur son écran à la manière d'un inventaire Minecraft.

La gestion des entités dynamiques et des tuiles de la map sont gérées grâce à l'implémentation d'une liste générique. Le serveur communique avec les clients en envoyant toutes les informations nécessaires à l'affichage du jeu par le client qui renvoie des informations au serveur en fonction des changements opérés par le joueur.

3 Organisation du Projet

3.1 Planning Prévisionnel

Ci-dessous notre planning prévisionnel sur 3 mois. (Voir Figure 1)

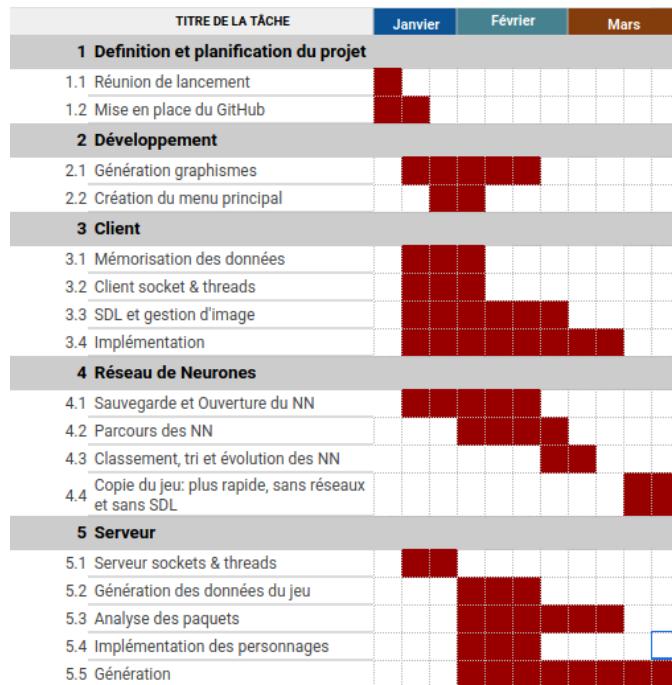


FIGURE 1 – Diagramme Gant

3.2 Répartition des tâches

Nous nous sommes répartis les tâches de la manière suivante (se référer au numérotage sur le diagramme Gant Figure 1) :

- Baptiste : 1.1, 1.2, 3.1, 3.2, 3.3, 3.4, 5.1, 5.2, 5.4, 5.5
- Lucas : 1.1, 3.2, 4.1, 4.2, 4.3, 4.4, 5.1, 5.5
- Ilann : 1.1, 2.2, 3.1, 5.4
- Nathan : 1.1, 2.1, 3.1, 5.2, 5.5

4 Développement

Cette partie abordera le développement du jeu dans sa globalité. De la création des ressources graphiques à la création du client/serveur en passant par la gestion des de la carte, des objets etc.

4.1 Creation des ressources

Cette section abordera la generation des ressources graphiques de base permettant la realisation du jeu.

4.1.1 Objet

Il y a quarante objets dans le jeu. Tous générés avec un prompt chat gpt. Quelques exemples d'objets ci-dessous (Figure 2, Figure 3).



FIGURE 2 – Bague du mage noir



FIGURE 3 – Orbe mystique

4.1.2 Sprites et Personnages

Il y a quatre personnages dans le jeu :

- Archer
- Mage
- Ninja
- Vampire

Chaque personnage à un spritesheet de 12 images (3 images par direction). Quelques exemples de spritesheet ci-dessous (Figure 4, Figure 5).



FIGURE 4 – spritesheet Mage



FIGURE 5 – spritesheet Archer

4.1.3 Tuiles et Biomes

Il y a six biomes différents dans le jeu.

Chaque biome a :

- une texture de base pour le sol (voir Figure 6)
- une texture pour un premier obstacle (voir Figure 7)
- une texture pour un deuxième obstacle (voir Figure 8)

— Une texture pour indiquer la sortie d'une tuile (voir Figure 9)
Exemple des 4 textures du biome plaine ci-dessous.

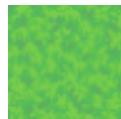


FIGURE 6 – Texture base

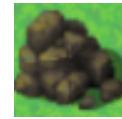


FIGURE 7 – Texture obstacle1



FIGURE 8 – Texture obstacle2

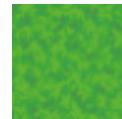


FIGURE 9 – Texture sortie

4.2 Implémentation

Nous allons maintenant présenter les différentes structures de données mises en œuvre pour la réalisation du jeu.

4.2.1 Map

La map est inspirée des jeux comme The Binding of Isaac, dans lesquels le joueur se déplace de salle en salle dans les quatre directions.

On appelle ces salles des tuiles. De plus, la map est constituée de différents biomes qui déterminent l'apparence de la tuile. Similaires aux biomes de Minecraft, les textures diffèrent selon les biomes (forêt, plaine, montagne, désert, neige, glace). Pour que le jeu soit re-jouable et intéressant, la map est générée procéduralement, en associant à chaque tuile un biome, puis en créant les tuiles en plaçant les textures selon le biome choisi.

Dans le programme écrit en C. La map est une matrice de tuile. Chaque tuile est une structure, qui contient l'information du biome, une matrice des textures qui la composent et les éléments du jeu qui y sont associés. On représente les informations par des listes. Elles sont variables au cours du jeu et selon les tuiles. Il y a quatre listes qui tiennent compte des joueurs, des objets, des monstres, et des effets des compétences actives afin de les afficher facilement.

4.2.2 Personnages

Les personnages sont choisis par les joueurs au début de la partie. Ils sont caractérisés par une classe, comme dans certains jeux de rôle. Cette classe détermine les statistiques de base du personnage et les compétences accessibles. Les statistiques sont la vie, la force, la magie et la vitesse. Chaque personnage a un point fort qui le distingue des autres.

Le personnage est représenté par une structure C. Ses statistiques sont deux tableaux de quatre valeurs, une pour sauvegarder ses statistiques de base et l'autre pour avoir accès

à ses statistiques améliorées par les objets. Un tableau de pointeurs permet de retrouver les objets possédés par le joueur.

La fonction creer_perso permet d'initialiser un joueur à partir de la classe choisie. Celle pour détruire est appelée à la fin du programme.

Pour gérer le déplacement des joueurs, on utilise SDL qui écoute les évènements du clavier. En utilisant l'état des flèches directionnelles, on peut faire changer la direction du joueur avec changer_dir. Puis la fonction avancer est appelée et change la position du personnage grâce à sa vitesse et sa direction en vérifiant qu'il ne dépasse pas les bords de la tuile. Pour simplifier le calcul, on utilise une table de hachage qui donne une structure qui est le multiplicateur en x et en y selon la direction.

Exemple : le joueur se déplace vers le haut et vers la droite, qui est converti en une valeur énumérée (type enum en C).

Cette valeur donne un indice pour le tableau deplacement. La valeur qui correspond à cette direction est la structure de valeurs $x = 1$ et $y = -1$, car un déplacement vers la droite signifie ajouter une valeur x (la vitesse du joueur) et un déplacement vers le haut implique de diminuer la valeur y du joueur de sa vitesse aussi.

4.2.3 Compétences

Les compétences sont les capacités des personnages, elles peuvent être de simples attaques, des techniques de déplacement ou bien des sorts.

4.2.4 Objets

Les objets sont essentiels au jeu. Ils permettent d'augmenter les statisques de base de notre personnage. Il y a 40 objets dans le jeu, séparés en 4 catégories de rareté :

- commun
- rare
- épique
- légendaire

Chaque objet a 4 statisques différentes :

- vie
- force
- magie
- vitesse

Ensuite pour calculer le bonus apporté au joueur par chaque objet on différencie trois cas de figure :

- bonus nul
- bonus à additionner
- bonus à multiplier

Pour garder une certaine cohérence dans les calculs de statisques du personnages un ordre de calcul a été décidé. On affectue d'abord les calculs de somme puis les calculs de produit. Pour ce faire chaque statisque de l'objet a un couple de valeur. La première valeur

corresponds à ce qui est à sommer ou multiplier à la statique de base du personnage. La deuxième valeur est un ordre de priorité permettant d'effectuer les calculs dans le bon ordre.

Exemple : si un personnage a deux objets :

- 1^{er} objet : $\{\{0, \text{null}\}, \{10, \text{add}\}, \{0, \text{null}\}, \{-2, \text{add}\}\}$
- 2^e objet : $\{\{1.5, \text{mult}\}, \{0.8, \text{mult}\}, \{20, \text{add}\}, \{0, \text{null}\}\}$

On rappelle que les statiques sont de la forme {vie, force, magie, vitesse}. Dans cet exemple, on commencera par calculer les trois additions puis les trois multiplications (la somme étant représenté par «add» et le produit par «mult»).

Deux fonctions permettent de gérer les objets possédés par le joueur. La fonction ajouter_objet permet de placer un nouvel objet dans l'inventaire à une place libre. La fonction retirer_objet supprime l'objet à la position donnée. Ces fonctions sont appelées par des fonctions de plus haut niveau qui vérifient la présence d'un objet au sol et retirent ou ajoutent l'objet correspondant sur la tuile en fonction de l'action du joueur.

La fonction update_stats est appelée quand un joueur gagne ou perd un objet. Elle permet de recalculer les statistiques du personnage

4.2.5 Monstres

Le jeu intègre des personnages non joueurs. Ce sont des monstres, appelés mobs. Lorsqu'il sont tués il permettent d'obtenir des objets qui augmentent les capacités de base de notre personnage. Les mobs ont plusieurs fonctionnalités. La première est qu'ils se rapprochent du joueur le plus proche à l'aide d'une fonction de calcul de distance. Une fois que cette distance est calculée, la direction que le mob doit prendre est choisie avec un calcul d'angle. Un vecteur horizontal orienté à droite qui servira de base est créé ainsi qu'un vecteur représentant la direction personnage vers mobs. (Voir Figure 16) On calcule l'angle en radians formé par ces deux vecteurs. On obtient donc une valeur du cercle trigonométrique que l'on exploite de la manière suivante :

- Le monstre va en haut si $\frac{\pi}{4} \leq \text{angle} < \frac{3\pi}{4}$
- Le monstre va à gauche si $\frac{3\pi}{4} \leq \text{angle} < \frac{5\pi}{4}$
- Le monstre va en bas si $\frac{5\pi}{4} \leq \text{angle} < \frac{7\pi}{4}$
- Le monstre va à droite si $\frac{7\pi}{4} \leq \text{angle} < \frac{\pi}{4}$

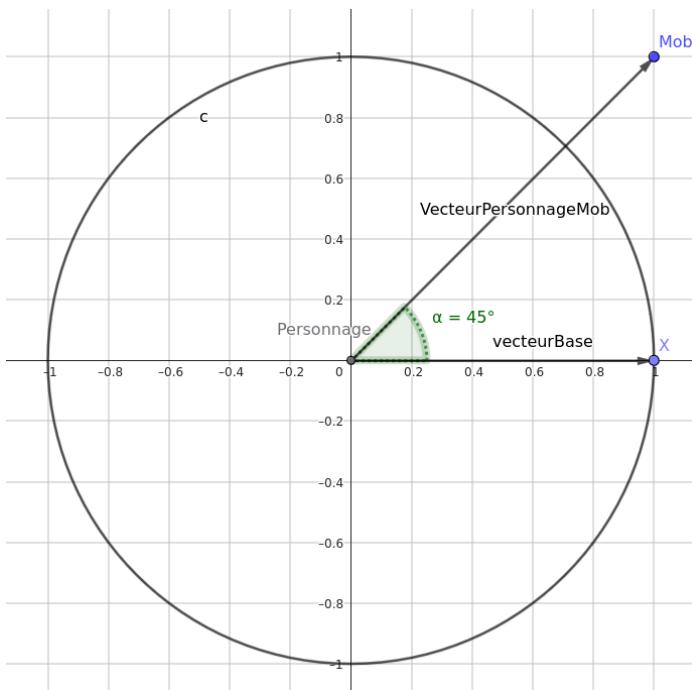


FIGURE 10 – Représentation d'un calcul d'angle en fonction de la position d'un personnage et d'un mob

4.2.6 Évènement

4.3 Réseau

Nous allons maintenant voir comment connecter les joueurs avec un serveur pour permettre la connexion et le jeu. Tout d'abord la communication à travers le réseau se fait par des écoutes et des envois dans différents flux. Le client n'a donc comme flux, seulement le serveur. Le serveur a bien plus de flux, un pour chaque joueur. La parallélisation est donc obligatoire.

4.3.1 Networking

Il y a tout d'abord une étape où le client se connecte au serveur, suite au choix réalisé dans le menu et suite à la l'initialisation du serveur. Plusieurs structures vont être créées et la parallélisation démarre.

L'envoi et la réception de données s'assure avec deux fonctions principales : La fonction `send` (qui n'est pas bloquante) permet l'envoi d'un paquet en unicast. La fonction `recv` (qui est bloquante) permet l'écoute du flux et la réception des paquets.

La propriété bloquante de `recv` force l'utilisation de threads. Ainsi dès la première connexion, une fonction parallèle d'écoute est créée. Le programme possède donc à ce moment :

- Le programme principal qui a appelé la création des autres threads.

— Un (ou plusieurs pour le serveur) thread(s) d'écoute.
 La connexion entre eux se fait par une file. Les threads d'écoute enfilent et le programme principal défile les informations envoyées par le serveur. Ces informations passent ainsi du serveur à l'écoute au main pour ensuite être traitées.
 Cette file permet aussi au serveur de traiter les différents clients.

Différents codes peuvent être utilisés lors de l'envoi de paquets (Voir Table 1) :

Code	Nom	Catégorie
11	JOUEUR_CHANGE_DIR	Joueur
12	JOUEUR_MV	Joueur
13	JOUEUR_MV_TUILLES	Joueur
...
20	ADD_OBJET	Objet
21	RM_OBJET	Objet
...
301	UPDATE_ZONE	Compétence de zone
302	RM_ZONE	Compétence de zone
...

TABLE 1 – Différents codes d'action entre le client et le serveur

4.3.2 Client

4.3.3 Serveur

4.4 Rendu Graphique

Cette partie traitera de l'utilisation de la librairie SDL2 qui sert pour le rendu graphique du jeu.

4.4.1 Menu Principal

Nous allons maintenant présenter la mise en œuvre du menu principal de notre jeu. Voici à quoi ressemble le menu lorsqu'on le lance. (voir Figure 11)



FIGURE 11 – Image menu principal lors du lancement du jeu

Affichage général

Le menu peut être dans différents états représentés par un type enum pos_actuelle, comme MENU_PRINCIPAL, DANS_PARAM ou SORTIE_MENU. Cet enum est mis à jour en fonction des boutons cliqués par le joueur.

La fonction aff_menu utilise l'enum pour afficher la bonne partie du menu. Elle utilise différentes fonctions comme clear_ecran qui sert à effacer tout ce qui est affiché sur l'écran et à remettre l'image d'arrière-plan. Elle utilise également les fonctions dessine_texte et dessine_image qui simplifient la lisibilité du code en remplaçant une longue ligne de SDL permettant d'afficher une image ou un texte à la fois par un simple appel à ces fonctions avec l'indice de leur tableau.

Il existe deux tableaux principaux pour la gestion des images et des textes que nous verrons plus en détail par la suite. Ces tableaux sont initialisés avec les coordonnées de chaque image et de chaque texte en s'adaptant à la taille de la fenêtre avec laquelle on lance le menu.

Images

Plusieurs images sont présentes dans le menu. On y retrouve différentes catégories :

- Personnages : les personnages disponibles pouvant être sélectionnés par le joueur
- Boutons : les boutons principaux du menu sont des images utilisant un style boisé
- Arrière-plan : une image dans le thème de la nature

Les images des personnages vus de face sont chargées en fonction du personnage sélectionné par le joueur. Pour se faire, on détruit puis recréée la texture correspondante dans tab_img à l'aide de deux fonctions :

- creer_image crée une SDL_Texture à partir du chemin vers le fichier correspondant à l'image

— `detruit_image` détruit la `SDL_Texture` à partir de l'indice de `tab_image`. Toutes les images chargées sont stockées dans un tableau de structure `img_t`. Chaque structure `img_t` possède deux champs : une `SDL_Texture`¹ et la position de l'image représentée par un `SDL_Rect`².

L'image d'arrière-plan est une `SDL_Texture` que l'on affiche sur tout l'écran. L'image est créée via la fonction `creer_image` et n'est pas stockée dans le tableau de `img_t`.

Nous utilisons principalement quatre images de boutons boisés pour servir d'arrière-plan aux textes présents. D'autres boutons sont présents en plus petit nombre comme un bouton de retour qui ramène au menu principal, des boutons plus et moins pour changer le volume de la musique du menu etc.

Textes

Les textes sont quant à eux chargés et sauvegardés dans un tableau de structure `texte_t` qui contient les mêmes champs que `img_t`, avec en plus un champ `contenu` qui est une chaîne de caractères correspondant au texte que l'on va afficher.

Pour créer des textes en SDL, on importe une police d'écriture (Go-Regular pour notre projet) et on choisit une couleur pour chaque texte. Nous utilisons principalement la couleur blanche pour les textes et dans certains cas la couleur est rouge.

La plupart des textes sont fixés et sont utilisés pour le nom des boutons mais certains sont modifiés en fonction des interactions avec l'utilisateur. Le nom du joueur, le nom du personnage choisi, le volume de la musique et la saisie de l'adresse IP en font partie. Pour créer un texte, nous avons deux fonctions :

- `creer_texte`
- `creer_texte_rouge`

Elles prennent les mêmes paramètres : un pointeur sur `texte_t` et une chaîne de caractères.

Elles créent une texture avec la couleur correspondante, la stockent dans le champ correspondant à la `SDL_texture` et mettent le nouveau texte dans le champ `contenu`.

Pour modifier les textes, on utilise une fonction `maj_texte` qui prend en paramètre un pointeur sur `texte_t` et une chaîne de caractères qui est le nouveau texte. On compare les deux textes et s'ils sont différents, on détruit le texte actuel et on le recrée avec la nouvelle chaîne de caractères.

Lors de la saisie de l'adresse IP (lorsque l'enum est en position `DANS_REJOINDRE`), on applique la fonction `saisie_touche_ip` qui sert à ajouter ou supprimer des caractères à l'IP en fonction du `SDL_Keycode`³ passé en paramètre.

Lorsque l'on clique sur le bouton rejoindre, on applique une expression régulière sur l'IP pour la valider ou non (Voir Figure 12 et 13). En cas d'échec, un texte s'affiche en rouge

1. image ou surface graphique que l'on peut afficher à l'écran à l'aide de la fonction `SDL_Render_Copy`.

2. structure de SDL à 4 champs : (coordonnées x et y de départ, longueur et largeur de la zone où sera affichée l'image).

3. un enum SDL correspondant à une touche pressée au clavier.

pour prévenir l'utilisateur d'une erreur de saisie.



FIGURE 12 – Affichage lorsque l'IP est valide



FIGURE 13 – Affichage lorsque l'IP est invalide

Gestion des événements du menu

Les événements en SDL sont gérés à l'aide d'une structure `SDL_Event` fournie par SDL. Le `SDL_Event` est utilisé, dans notre projet, pour détecter les clics de la souris et les touches pressées au clavier. Pour détecter ces entrées, nous regardons quel est le type du `SDL_Event`. Il peut être de type `SDL_KEYDOWN` (touche clavier) ou `SDL_MOUSEBUTTONDOWN` (clic souris).

souris

Les clics souris sont assez simples à gérer : si l'événement est de type clic souris, on crée un `SDL_Point`⁴ correspondant aux coordonnées du curseur de la souris. Nous avons une succession de conditions qui testent si le clic de la souris est dans les coordonnées d'un bouton à l'aide de la fonction `SDL_PointInRect`. Les conditions testent également la position de l'enum `pos_actuelle` pour que si par exemple, on clique sur le bouton de sauvegarde des paramètres alors qu'on n'est pas dans l'onglet paramètres, rien ne se passe. En fonction des boutons cliqués, différentes actions se passent. Il est possible de naviguer entre les différentes sections du menu de cette manière.

clavier

Les saisies clavier servent à modifier des chaînes de caractères mais aussi à se déplacer dans les menus à l'aide des flèches et des touches entrées. La saisie de l'adresse IP se fait via un switch dans la fonction `saisie_touche_ip` vue précédemment. Elle teste si on clique sur les chiffres du pavé numérique ou sur un point pour l'ajouter à la chaîne de l'IP. Si on appuie sur la touche effacer, ça enlève le dernier caractère de la chaîne s'il y en a au moins un. Pour la saisie du nom du joueur, c'est la même chose mais en plus simple : chaque caractère alphanumérique est ajouté à la chaîne du nom du joueur. Si on appuie sur effacer ça fait la même chose que pour la saisie de l'IP.

Lorsqu'un bouton est sélectionné, son affichage change en fonçant légèrement l'image de

4. une position sur la fenêtre représentée par deux coordonnées x et y.

fond. Chaque partie du menu a un tableau d'entiers correspondant aux indices des boutons pouvant être sélectionnés à l'aide des flèches. Un entier bouton_select correspond à l'indice du bouton sélectionné dans le tableau actuel.

Un pointeur sur tableau (tabBoutons) contient tous les tableaux d'indices. On récupère le tableau correspondant à l'endroit où on se situe dans le menu en faisant tabBoutons[pos_actuelle] pour le mettre en paramètre dans la fonction d'affichage du menu. bouton_select fonctionne circulairement : il est positionné sur le premier bouton du tableau et lorsque l'on clique sur les flèches haut ou bas, on se déplace dans le tableau en gérant les dépassements avec un modulo.

(image codeMenu - description : voici un exemple de gestion des entrées clavier par le joueur)

Si l'événement SDL est de type SDL_KEYDOWN, on crée un SDL_Point qui correspond aux coordonnées du bouton sélectionné en ajoutant un aux abscisses et aux ordonnées pour qu'il soit à l'intérieur des coordonnées.

Lorsque l'on clique sur echap, cela force l'arrêt du menu.

4.4.2 Jeu

Dans cette partie, nous présenterons les moyens d'affichage des différents composants du jeu.

Dans le jeu, le rendu de la map est généré en plusieurs étapes. Premièrement, la fonction init_biomes charge toutes les textures du jeu qui composent les tuiles (voir Figure 6, 7, 8 et 9). Ensuite, les fonctions init_map et init_tuile génèrent pour chaque tuile une instance d'un biome. Enfin, il reste la phase d'exploitation : quand un joueur entre sur une tuile. La fonction charger_tuile s'occupe de créer le rendu qui sera affiché sur l'écran (qui est fixe) à partir de la matrice de la tuile (voir exemple biome Figure 14 et 15). Dans la boucle du jeu, cette texture est affichée avant les autres éléments du jeu à l'écran, grâce à la bibliothèque SDL, car elle est le décor.



FIGURE 14 – Biome Forêt

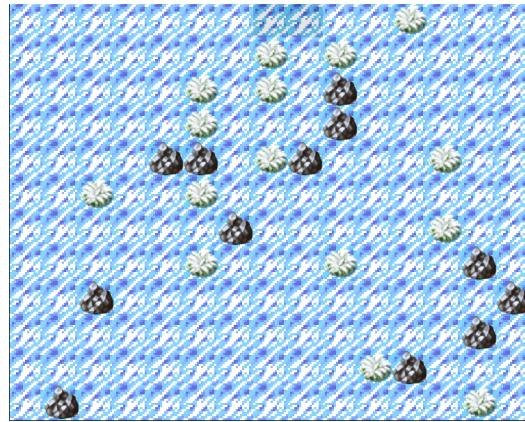


FIGURE 15 – Biome glace

Pour l'affichage des personnages. Nous avons développé une fonction `charger_sdl_joueurs`, qui prend en paramètres le tableau des joueurs et un tableau à deux dimensions qui stocke les différentes textures des personnages. La fonction s'occupe de charger les bonnes images en fonction de la classe de chaque joueur. Dans la boucle principale du client, la liste des personnages présents est parcourue, on affiche alors la texture du personnage qui correspond à sa direction, à sa position courante.

L'affichage des objets est similaire, les textures sont préchargées, puis, grâce à la liste qui tient compte de ceux qui sont présents sur la même tuile que le joueur, ils sont affichés. Les fonctions de destruction et de libération des ressources sont appelées à la fin du programme. Les textures des objets, personnages, biomes doivent être détruites par la fonction `SDL` adéquate. Tous les éléments des listes de chaque tuile sont aussi supprimés par `detruire_tuile`, faisant appel à `detruire_liste`.

5 Résultats et conclusion

6 Annexes

6.1 Débogage

```
void init_joueurs_server(perso_t * joueurs, int nb){
    char * data;
    int equipe=1;//alterne à chaque boucle
    classe_t classe;
    char *nom;
    int ind;
    /*On ne sait pas dans quel ordre on va lire, on utilise l'indice
    que le client envoie.*/
    while(nb){//nb de messages attendus
        if(fileVide(serv_file))
            sleep(1);
        else{
            data=defiler(serv_file);
            sscanf(data, "%d %d %s", &ind, (int*)&classe, nom);
            creer_perso(joueurs + ind, classe, nom, ind, equipe!=equipe);
            free(data);
            nb--;
        }
    }
}
```

FIGURE 16 – Fonction à tester

```

Thread 1 "serv" received signal SIGSEGV, Segmentation fault.
__vfscanf_internal (s=s@entry=0x7fffffffdea0,
    format=format@entry=0x55555555b1c7 "%d %d %s",
    argptr=argptr@entry=0x7fffffffde88, mode_flags=mode_flags@entry=2)
at vfscanf-internal.c:1100
1100 vfscanf-internal.c: Aucun fichier ou dossier de ce type.
(gdb) bt
#0 __vfscanf_internal (s=s@entry=0x7fffffffdea0,
    format=format@entry=0x55555555b1c7 "%d %d %s",
    argptr=argptr@entry=0x7fffffffde88, mode_flags=mode_flags@entry=2)
at vfscanf-internal.c:1100
#1 0x00007ffff7c7d382 in __GI_isoc99_sscanf (s=0x7ffe8000b60 "0 3 lulu ",
    format=0x55555555b1c7 "%d %d %s") at isoc99_sscanf.c:31
#2 0x0000555555581ed in init_joueurs_server (joueurs=0x7fffffff0c0, nb=2)
at serv_jeu.c:117
#3 0x000055555556a8c in main_server (port=2020, nb_clients=2) at serv.c:54
#4 0x00005555555743f in main (argc=2, argv=0x7fffffff458) at serv.c:170
(gdb) print ind
No symbol "ind" in current context.
(gdb) up
#1 0x00007ffff7c7d382 in __GI_isoc99_sscanf (s=0x7ffe8000b60 "0 3 lulu ",
    format=0x55555555b1c7 "%d %d %s") at isoc99_sscanf.c:31
31 isoc99_sscanf.c: Aucun fichier ou dossier de ce type.
(gdb) print ind
No symbol "ind" in current context.
(gdb) up
#2 0x0000555555581ed in init_joueurs_server (joueurs=0x7fffffff0c0, nb=2)
at serv_jeu.c:117
117         sscanf(data, "%d %d %s", &ind, (int*)&classe, nom);
(gdb) print ind
$1 = 0
(gdb) print classe
$2 = mage
(gdb) print nom
$3 = 0x18 <error: Cannot access memory at address 0x18>
(gdb) whatis nom
type = char *

```

FIGURE 17 – *nom* ne doit pas être un pointeur mais un tableau