Projet de Programmation C : Donjon Rogue

L'objectif de ce projet est de développer une application graphique d'un jeu d'exploration de donjon de type *roguelike*. Il s'agit typiquement de jeux en tour par tour où les ingrédients aléatoires (cartes, monstres, *etc.*) jouent un rôle essentiel.

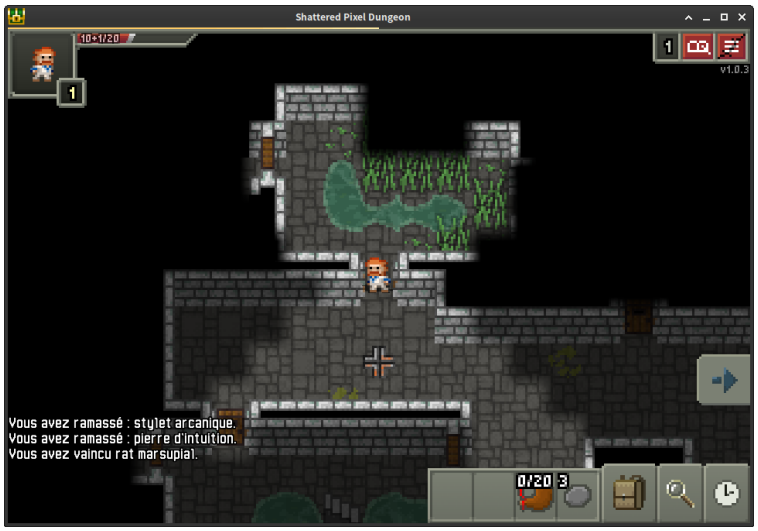


Figure 1: Capture d'écran de *Shattered Pixel Dungeon*

1 Description et déroulement du programme

Une fois le jeu lancé, après un écran d'accueil (totalement optionnel), une fenêtre graphique contenant un menu s'ouvre et permet à l'utilisateur de commencer une nouvelle partie ou de reprendre une éventuelle partie existante. Une fois le choix pris, le jeu commence, avec un personnage au centre de l'écran, sur fond de donjon. Le donjon se compose de plusieurs étages, et l'utilisateur explore le donjon avec le personnage. Le terrain de chaque étage est généré aléatoirement, rempli avec des monstres, des trésors, un escalier pour remonter à l'étage précédent, et un escalier pour descendre au prochain étage. Le personnage se déplace avec les touches directionnelles (et optionnellement ZQSD), et les actions sont effectuées en cliquant dans l'interface (et optionnellement avec des raccourcis clavier). La partie se termine quand le personnage meurt, et les statistiques de cette partie sont affichées.

2 Mécanisme du jeu

2.1 État du personnage

Le personnage possède les attributs suivants :

- **Hp** (Hit Point, point de vie), **Mp** (Mana Point, point de magie);
- **Atk** (attaque), **Int** (intelligence), **Def** (défense);
- **Exp** (point d'expérience), **Lv** (niveau du personnage).

Hp et Mp représentent des jauges de points de vie et de magie qui pourront augmenter ou diminuer au cours de la partie. Si Hp descend à 0, le personnage meurt et la partie se termine. Ces deux jauges ont une valeur maximale de $10 \times$ Def pour Hp et $10 \times$ Int $- 50$ pour Mp.

Au début de la partie, les valeurs Atk, Int et Def du personnage sont fixées à 10, et les jauges Hp et Mp sont pleines : le personnage commence donc avec 100 Hp et 50 Mp.

L'Exp est accumulée à chaque victoire en combat et est dépensée pour monter de niveau. Passer du Lv n au Lv $n+1$ nécessite $350 + 50 \times n$ points d'expérience, qui sont automatiquement soustraits au passage au niveau supérieur. Le passage de niveau remet les jauges Hp et Mp à leur maximum et octroie 3 points au joueur à affecter librement aux attributs Atk, Int et Def.

2.2 Terrain

Chaque étage du donjon est compris dans une grille de taille 63×43, composée de cases de salles (où le personnage peut se déplacer et qui peuvent contenir des monstres et des trésors générés aléatoirement) et des cases de mur. Les bords de la grille sont toujours des murs, et les salles doivent former une seule composante connexe.

Chaque étage possède un escalier montant au centre, où commence le joueur lorsqu'il vient d'atteindre l'étage en descendant depuis l'étage supérieur. Chaque étage contient également un escalier descendant pour aller à l'étage suivant, qui doit être relié à l'escalier montant par au moins un chemin. Les autres parties du terrain sont à générer aléatoirement, avec des **murs**, des **monstres** et des **trésors** (voir Section 3).

Les monstres sont décrits par deux attributs, **Hp** et **Atk**, qui seront utilisés en combat. Les trésors contiennent des objets générés aléatoirement et affichés un par un. Le joueur peut choisir de prendre chaque trésor au moment où il est affiché. Il y a quatre types d'objets : **potions**, **armures**, **armes**, **baguettes magiques**.

2.3 Combat

Après chaque tour du personnage, chaque monstre adjacent effectue une attaque qui inflige des dégâts aléatoires entre 80% et 120% de son attribut Atk. Chaque attaque a une probabilité de 5% d'infliger un **coup critique**, dont les dégâts sont **triplés**. Si plusieurs monstres sont adjacents au joueur, chaque monstre effectue une attaque dans un ordre à déterminer.

À son tour, le personnage peut aussi choisir de lancer une attaque en mêlée ou une attaque magique contre un monstre adjacent. Les attaques de mêlée suivent le même calcul de dégâts (y compris pour les coups critiques) que ceux des monstres. Les attaques magiques consomment 20 Mp, et suivent le même calcul de dégâts que pour les attaques de mêlée, en remplaçant l'attribut Atk par $2 \times$ Int.

Le monstre est éliminé quand son Hp tombe à 0 et le personnage gagne un certain nombre d'Exp, **dont la valeur est laissée à votre choix**. En général, la valeur totale d'Exp qu'on peut obtenir dans un étage de donjon doit être environ suffisante pour que le personnage gagne deux niveaux.

2.4 Objets et Trésor

Le personnage peut porter un maximum de 12 objets. Les potions sont des consommables, qui octroient un bonus une fois utilisées. Les autres objets sont des pièces d'équipement qui améliorent les attributs du personnage. Les objets peuvent être jetés à tout moment, et les objets jetés disparaissent.

Il y a 5 types de potions :

- **Potion de soin** : restaure 10% de l'Hp (par rapport au max) ;
- **Potion de magie** : restaure 10% du Mp (par rapport au max) ;
- **Potion de régénération** : restaure 20 Hp et 10 Mp tous les 3 tours pendant 30 tours, sauf si le personnage a déjà cet effet ;
- **Potion de précision** : augmente la probabilité d'attaque critique de 5% à 15% pendant 30 tours ;
- **Potion d'apprentissage** : augmente les points d'Exp gagnés de 30% pendant 30 tours.

Les équipements (armures, armes, baguettes magiques) ont un attribut **Qualité**, qui s'ajoute, quand équipé, aux attributs correspondants : Def pour les armures, Atk pour les armes, et Int pour les baguettes magiques. Les équipements sont équipés automatiquement, et si un équipement de qualité supérieure est disponible, le personnage remplace automatiquement l'ancien équipement du même type. Les équipements trouvés au n-ième étage ont une qualité aléatoire entre 1 et n. Les objets sont tous trouvés dans des trésors. Une fois fouillé, le trésor disparaît.

2.5 Action du personnage

Le jeu se déroule par tour. À chaque tour, le personnage peut :

- se déplacer d'une case vers une salle adjacente, et quand le personnage se déplace sur un escalier, il le suit immédiatement et se retrouve dans l'étage correspondant;
- fouiller les objets d'un trésor adjacent; si l'inventaire est plein le joueur doit pouvoir choisir un objet à jeter pour pouvoir accueillir un nouvel objet;
- lancer une attaque contre un monstre adjacent.

Après le tour du personnage, les monstres adjacents effectuent chacun une attaque contre le personnage. Consommer des potions et inspecter l'inventaire sont des actions spéciales qui ne comptent pas comme un tour, et peuvent être effectuées à tout moment.

2.6 Sauvegarde de partie

Pour que la partie puisse être reprise, il faut sauvegarder la partie en cours quand l'utilisateur quitte l'application en fermant la fenêtre. Voici quelques éléments que la sauvegarde doit contenir :

- Le personnage, avec sa position, son orientation et ses attributs ;
- Le terrain, incluant la position des escaliers, les monstres et leurs attributs, et les trésors et objets qu'ils contiennent ;
- Toute autre information nécessaire.

Lorsque l'utilisateur quitte le programme, toutes les informations nécessaires à la reprise de la partie doivent être sauvegardées. La sauvegarde devra être en **binaire** pour empêcher les tricheurs de modifier les données du jeu. Le format est laissé à votre choix.

3 Génération de terrain

La taille des étages étant fixée (63×43), on peut représenter le terrain comme un tableau de taille fixée dont chaque élément représente une case. Typiquement, comme il n'y a qu'un petit nombre de possibilités pour chaque cases (case de mur, case de salle, monstre, trésor, escalier montant, escalier descendant), on peut représenter les cases par la structure suivante :

```
1 typedef enum {WALL, ROOM, MONSTER, TREASURE,
2                STAIR_UP, STAIR_DOWN} Celltype;
3
4 typedef struct {
```

```
5       Celltype type;
6       union {
7           Monster monster;
8           Treasure treasure;
9       }
10  } Cell;
```

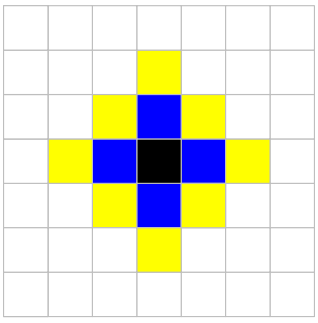
Quand on commence une partie, ou quand on descend à un nouvel étage, le jeu doit générer un nouveau terrain pour l'étage correspondant, y compris les monstres, les trésors et leurs objets.

Vous êtes libres de suivre la méthode de votre choix pour générer le terrain, tant qu'elle respecte les contraintes définies dans les sections précédentes. La section suivante présente une telle approche, mais d'autres méthodes sont également possibles. En général, on commence par remplir tout le terrain avec des cases de mur, on place l'escalier montant au centre, puis on génère les autres cases dans l'ordre suivant : salles, trésors, monstres puis escalier descendant.

3.1 Une méthode de génération de labyrinthe

Supposons que le terrain est déjà rempli par cases de mur, avec un escalier montant au centre. Pour générer les cases de salle, on utilise l'algorithme ci-dessous, dont l'idée est de trouver les cases de mur adjacentes à exactement une case de salle et les convertir en cases de salle, tout en s'assurant qu'il n'y a pas déjà trop de cases de salle à proximité.

Pour cela, on définit la *distance de Manhattan* de deux cases (i_1, j_1) et (i_2, j_2) par $|i_1 - i_2| + |j_1 - j_2|$. Dans la figure suivante, les cases bleues sont à distance 1 de la case noire, tandis que les cases jaunes sont à distance 2. On dit qu'une case est *admissible* si c'est une case de mur avec exactement une case de salle à distance 1 et au plus 2 cases de salle à distance 2.

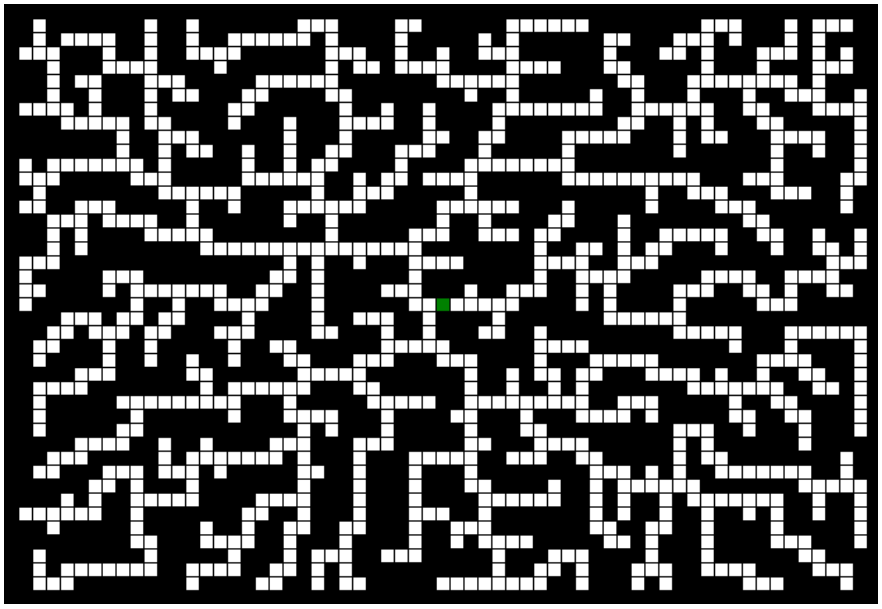


L'algorithme se déroule de la manière suivante :

1. Initialiser une liste `toexpand` des cases de mur adjacentes à l'escalier montant.
2. Tirer une case aléatoire `c` dans `toexpand` et la retirer de la liste.

3. Si `c` n'est pas admissible, revenir à l'étape 2, sinon convertir `c` en une case de salle et passer à l'étape suivante.
4. Regarder les quatre cases adjacentes à `c`, et ajouter celles qui sont admissibles dans `toexpand` si elles n'y sont pas déjà et si elles ne sont pas aux bords.
5. Si `toexpand` n'est pas vide, revenir à l'étape 2, sinon passer à l'étape suivante.
6. Identifier tous les cases de salle adjacentes à trois cases de mur. Passer ces cases en cases de mur, puis terminer.

Cette méthode de génération garantit que toutes les cases de salle sont atteignables à partir de l'escalier montant. Voici un exemple du résultat, avec l'escalier montant en vert.



3.2 Génération des autres éléments

Une fois les cases de salles générées, on remplace certaines d'entre elles par des trésors. Un trésor est toujours généré dans une case de salle à coté de l'escalier montant (point de départ) du premier étage comme bonus au début de la partie. Les autres trésors sont générés dans une case de salle accessible à partir de l'escalier montant et **adjacente à une seule case de salle**, dans laquelle un monstre est toujours généré pour garder le trésor. Dans chaque trésor se trouvent deux objets, générés aléatoirement. **La fréquence de génération de chaque type d'objet est laissée à votre choix**.

Vous pouvez aussi générer d'autres monstres que ceux qui gardent les trésors, **dont la fréquence de génération est laissée à votre choix**. Lorsqu'un monstre est généré, ses attributs Hp et Atk sont tirés aléatoirement. **Les distributions de ces deux attributs sont**

laissées à votre choix. En général, il faut que les monstres soient plus forts quand on va plus loin dans le donjon.

Finalement, on prend une case de salle éloignée de l'escalier montant (distance de Manhattan au moins 60) qui est encore vide pour y placer l'escalier descendant.

La jouabilité du jeu repose fortement sur l'équilibre entre la progression du personnage et l'augmentation de difficulté de chaque étage. **Il est donc crucial de bien choisir les paramètres qui sont laissés à votre choix.**

4 Interface graphique

À tout moment du jeu, le personnage se situe au centre de la fenêtre, qui affiche les 13×9 cases autour. Quand le personnage se déplace, il s'oriente visuellement dans la direction choisie et l'affichage est mis à jour. S'il y a un monstre dans la case où le personnage veut se déplacer, alors le déplacement est interprété comme une attaque en mêlée. Le déplacement vers un trésor est interprété comme une fouille. L'interface graphique doit aussi proposer toutes les autres actions possibles, par exemple l'attaque magique.

En plus de l'affichage principal, il faut aussi prévoir des petits modules pour les fonctionnalités suivantes :

- Affichage des attributs ;
- Manipulation de l'inventaire ;
- Fouille de trésor ;
- Distribution des points d'attribut obtenus au passage de Lv.

Le même module peut servir pour plusieurs fonctionnalités.

Pour réaliser l'interface graphique, vous utiliserez obligatoirement la bibliothèque graphique C de l'université : `libMLV`. Une documentation de cette bibliothèque est accessible à l'adresse `http://www-igm.univ-mlv.fr/~boussica/mlv/api/French/html/index.html`. Elle est déjà installée sur toutes les machines de l'université.

Ce choix de la libMLV est une obligation. Il y a mieux, il y a aussi moins bien... On attend de vous que vous vous familiarisiez avec une bibliothèque écrite par un autre, et que sa documentation vous suffise à construire une véritable application graphique.

5 Modularité

Votre projet doit organiser ses sources sémantiquement. Votre code devra être organisé en modules rassemblant les fonctionnalités traitant d'un même domaine. Un module sans entêtes pour le main, un module pour le moteur du jeu, un module pour la gestion

du terrain et un module graphique sont un minimum. Le moteur du jeu sera vraisemblablement très volumineux et on peut même imaginer le subdiviser au besoin (module pour le combat, module pour l'inventaire et les objets...).

Votre projet devra être compilable via un Makefile qui exploite la compilation séparée. Chaque module sera compilé indépendamment et seulement à la fin, une dernière règle de compilation devra assembler votre exécutable à partir des fichiers objets en faisant appel au linker. Le flag -lMLV est normalement réservé aux modules graphiques ainsi qu'à l'assemblage de l'exécutable (sinon, c'est que votre interface graphique dégouline de partout et que vos sources sont mal modulées).

6 Pour aller plus loin

Pour les plus récalcitrants d'entre vous, toute amélioration sera considérée comme un plus pour l'évaluation. Voici quelques suggestions possibles de raffinements pour ce projet. Ces propositions sont gradués avec un indice de difficulté entre parenthèse.

- (blob) Ajouter du son pour les actions du personnage.
- (blob) Générer plusieurs types de monstres, avec des caractéristiques différents.
- (blob) Donner la possibilité de choisir la classe du personnage (guerrier, magicien, etc.), chaque classe disposant d'attributs initiaux différents et de capacités passives.
- (gobelin) Ajouter des animations de déplacement, d'attaque et de fouille.
- (gobelin) Ajouter des potions ou des objets avec d'autres comportements.
- (gobelin) Introduire des étages de boss, par exemple tous les 5 ou 10 étages, dépendant des paramètres que vous avez choisis.
- (paladin) Ajouter des portes verrouillées, dont les clés sont placées aléatoirement dans certains trésors. Il est évident que le trésor contenant la clé d'une porte doit être atteignable sans passer par ladite porte !
- (paladin) Ajouter des compétences actives ou passives que le personnage peut acquérir en montant de niveau.
- (paladin) Minimiser la taille de votre sauvegarde en ne gardant que la graine aléatoire utilisée dans la génération des cases de chambre de chaque étage. Il faut vérifier la cohérence des données dans la sauvegarde.
- (dragon) Introduire une ou plusieurs autres méthodes de génération du terrain qui donnent des résultats intéressants pour ce jeu. Vous pouvez générer les différents étages avec des méthodes différentes. Vous êtes aussi libre de modifier la taille de terrain pour que ce soit compatible avec votre propre méthode de génération.

- (dragon) Ajouter un mécanisme d'enchantement pour les équipements, qui leur donne des effets permanents (de régénération, de protection, etc.) au prix de jetons très rares. Dans ce cas, il faudra laisser à l'utilisateur le choix de remplacer ou non ses équipements lorsque de nouveaux équipements sont trouvés.
- (Jörmungandr) Ajouter un étage caché de mini-jeu, dans lequel le personnage doit résoudre des casse-têtes de type Sokoban pour obtenir des objets spéciaux. La sauvegarde doit aussi marcher ici. La manière d'arriver à cet étage caché est laissée à votre imagination.

Attention, **tout changement déteriorant la jouabilité sera lourdement pénalisé** ! Faites attention à bien calibrer les paramètres et les nouveaux mécanismes !

7 Conditions de développement

Le but de ce projet est moins de pondre du code que de développer le plus proprement possible. C'est pourquoi vous développerez ce projet en utilisant un système de gestion de versions Git. Le serveur à disposition des étudiants de l'Université est disponible à cette adresse : `https://forge-etud.u-pem.fr/`. Comme nous attendons de vous que vous le fassiez sérieusement, votre rendu devra contenir un dump du fichier de logs des opérations effectuées sur votre projet, extrait depuis le serveur de gestion de versions ; afin que nous puissions nous assurer que vous avez bien développé par petites touches successives et propres (commits bien commentés), et non pas avec un seul commit du résultat la veille du rendu. L'évaluation tient compte de la capacité du groupe à se diviser équitablement le travail.

Voici quelques **remarques importantes** :

- L'intégralité de votre application doit être développée exclusivement en langage C (la documentation technique dynamique fait évidemment exception). Toute utilisation de code dans un autre langage (y compris C++) vaudra 0 pour l'intégralité du projet concerné.
- En dehors des bibliothèques standards du langage C et de `libMLV`, il est interdit d'utiliser du code externe : vous devrez tout coder vous-même. Toute utilisation de code non développé par vous-même vaudra 0 pour l'intégralité du projet concerné.
- Tout code commun à plusieurs projets vaudra 0 pour l'intégralité des projets concernés.

8 Conditions de rendu

Vous travaillerez en **binôme** et vous lirez avec attention la Charte des Projets. Il faudra rendre au final une archive `tar.gz` de tout votre projet (tout le contenu de votre projet `git`), les sources de votre application et ses moyens de compilation. Il sera alors crucial de lire des recommandations et conseils d'utilisation de `git` sur la plate-forme `e-learning`. Vous devrez aussi donner des droits d'accès à votre chargé de TP et de cours à votre projet via l'interface `redmine`.

Un exécutable devra alors être produit à partir de vos sources à l'aide d'un `Makefile`. Naturellement, toutes les options que vous proposerez (ne serait-ce que `-help`) devront être gérées avec `getopt` et `getopt_long`.

La cible `clean` doit fonctionner correctement. Les sources doivent être propres, dans la langue de votre choix, et commentées. C'est bien de se mettre un peu à l'anglais si possible.

Votre archive devra aussi contenir :

- Un fichier `log_dev` correspondant au dump des logs de votre projet (nom des commits, qui? et quand?), extrait depuis le serveur de gestion de versions que vous aurez utilisé.
- Un fichier `Makefile` contenant les règles de compilation pour votre application ainsi que tout autre petit bout de code nécessitant compilation (comme les tests par exemple).
- Un dossier `doc` contenant la documentation technique de votre projet ainsi qu'un fichier `rapport.pdf` contenant votre rapport qui devra décrire votre travail. Si votre projet ne fonctionne pas complètement, vous devrez en décrire les bugs.
- Un dossier `src` contenant les sources de votre application.
- Un dossier `include` contenant tous les headers de vos différents modules.
- Un dossier `bin` contenant à la fois les fichiers objets générés par la compilation séparée.
- Aucun fichier polluant du type `bla.c~` ou `.%smurf.h%` généré par les éditeurs. Votre dossier doit être propre !
- Si possible, la partie `html` générée par l'utilitaire `doxygen` à partir de votre application de visualisation. Ceci est optionnel mais tellement plus propre.

Sachant qu'un script automatique scrupuleux va analyser et corriger votre rendu avant l'oeil humain, le non-respect des précédentes règles peut rapidement avorter la correction de votre projet. Le respect du format des données est une des choses des plus critiques.