Rapport de Projet informatique

Gauthier Wiemann - David Albert - Roméo Bellevergue

Table des matières

Introduction

Dans le cadre de notre enseignement à l'INSA de Rouen, nous avons effectué un projet informatique dans lequel il nous a été demandé de concevoir un programme à l'aide des connaissances acquises lors de la première année.

Nous avons choisi de créer un jeu de toute pièce avec nos propres règles. Nous avons décidé de concevoir un jeu de combat au tour par tour opposant deux créatures. Le principe est simple : choisir où est-ce que la créature doit attaquer celle adverse parmi la tête, le buste et les jambes, puis où est-ce qu'elle doit se défendre. À cela s'ajoute la gestion d'une capacité spéciale qui doit aider le joueur à remporter le combat plus rapidement. La créature adverse est gérée par l'ordinateur. Cependant le jeu est plus vaste que cela car la créature doit aussi se faire soigner dans une infirmerie. Dans celle-ci, le joueur a besoin d'argent. C'est à la chasse, un ensemble de mini-jeux, qu'il peut en gagner.

Nous reviendrons en premier lieu dans ce rapport sur notre cahier des charges. Dans une deuxième partie nous vous présenterons l'analyse descendante finale. S'en suivra ensuite un guide d'utilisation du programme, un mot sur les perspectives d'évolution de celui-ci et enfin un retour sur notre travail en groupe.

Cahier des charges

Descriptif détaillé du jeu à programmer fait en début de projet

Objectif : Réaliser dans le cadre du projet informatique un jeu vidéo que l'on nomme « L'Arène ».

Le principe du jeu consiste à diriger une créature que nous allons choisir en début de partie parmi plusieurs. Celle-ci pourra au fur et à mesure de ses combats gagner de l'expérience et ainsi passer des niveaux et gagner des points de compétence.

Le jeu se structure autour d'un menu principal à 6 onglets : une arène pour les combats contre d'autres créatures, une infirmerie pour se soigner, un terrain de chasse pour gagner de l'argent, un onglet pour afficher les statistiques de sa créature, un onglet enregistrer/quitter et un onglet « règles du jeu » .

Fonctionnement de l'arène : On combat contre une créature récupérée aléatoirement dans la liste de tous les compte utilisateurs. Chaque créature est divisée en trois zones. Chaque créature attaque tour à tour et la créature la plus rapide commence. Chaque tour se divise en trois phases. La première est le choix de l'utilisation, ou non, d'une attaque spéciale puissante et utilisable qu'une seule fois par combat. Le deuxième est de choisir la zone que l'on veut défendre sur sa créature et la dernière est le choix de la zone de la créature adverse à attaquer. La quantité de dégâts infligés dépend de la défense de l'attaqué et de l'attaque de l'attaquant.

Fonctionnement de l'infirmerie : On soigne sa créature contre de l'argent (1 Gold = 1 Point de Vie).

Fonctionnement de la zone de chasse : On chasse des animaux pour amasser de l'argent afin de soigner sa créature. Pour les chasser, un mini jeu sera implémenté.

Liste des fonctionnalités prévues en début de projet

Enregistrement de sa partie (√ Fait)
Choix de la créature (√ Fait) Affichage des caractéristiques de sa créature (√ Fait)
Choix des améliorations (√ Fait) Possibilité de se soigner (√ Fait) Evolution de la créature (√ Fait)
Fonctionnalités de combats (√ Fait)

Liste des fonctionnalités ajoutées au cours de la réalisation du jeu

Interface graphique (SDL)
Plusieurs mini-jeux
Compte administrateur
fonctionnalité sauvegarde

Analyse descendante

Définition des types du jeu principal

Dans le programme principal Jeu_arene

```
VAR         listing : ListCompte;
                monCompte : Compte;
```

Dans l'unité type_jeu

```
CONST MAX = 1000;
Type
        Cible = (tete, buste, jambes);

        Capacite = (force, resistance, agilite);

        Creature = record
                nom , image: string;
                niveau , xp, pv, spe : integer;
                competence : array[force..agilite] of integer;
        end; {regroupe tout ce qui caractéristique une créature}

        Compte = record
                creature : Creature;
                MDP : string;
                numero, argent : integer;
        end; {regroupe créature, mot de passe, numéro de compte et argent du compte}

        ListCompte = record
                compte : array[1..MAX] of Compte;
                nombre : integer; {taille utilisée du tableau de compte}
        end; {regroupe l'ensemble des comptes avec 'nombre' le nombre de compte}

        FComptes = File of ListCompte;
        {permet d'enregistrer la liste des comptes dans un fichier}
```

Dans l'unité SDL_procedures_functions

```
CONST         L_Ecran = 900; {Largeur de l'écran}
                  H_Ecran = 600; {hauteur de l'écran}

VAR           ecran : PSDL_Surface;
```

```
{Déclaration d'une variable globale = utilisable dans toute l'unité
= on utilise dans cette unité le même écran de dimension L_Ecran*H_Ecran}
```

Dans l'unité jeu_chasse

```
CONST          NB_JEUX = 4;
                   HAUT = 70;
                   H_Ecran = 100+ NB_JEUX * HAUT;
                   L_Ecran = 718;

VAR            screen : PSDL_Surface;
```

Signatures des procédures et fonctions du jeu principal

Initialiser/Quitter

```
procedure initSDL_Main; {initialise la bibliothèque SDL}
procedure quitSDL_Main; {ferme la bibliothèque SDL}
procedure intro; {Animation d'accueil du programme-jeu}
```

Connexion

```
function chargerComptes(nom : string): ListCompte;
{On récupère la liste de tous les comptes utilisateurs placés dans le fichier
correspondant (nom en entrée) et on les mets dans un listing}
function connexion(var listing : ListCompte): Compte;
{L'utilisateur choisi s'il veut faire un recommencer une partie
ou continuer son ancienne partie}
function chargerPartie(listing : ListCompte): Compte;
{Charger une partie existante dans la liste de tous les comptes}
function nouvellePartie(var listing : ListCompte): Compte;
{Crée un nouveau compte pour le nouveau (pas forcément) joueur}
function choixCreature(listing : ListCompte): Creature;
{Accès à une image pour le choix d'une des 6 créatures élémentaires
pour démarrer l'aventure : il faut cliquer dessus}
```

Menu princial / accès aux différentes parties

```
procedure menu(monCompte : Compte ; listing : ListCompte);
{Accès au menu du jeu : il comprend plusieurs items}
procedure regle; {Affichage des règles générales du jeu}
procedure afficherStats_SDL(var v_compte : Compte; normal :Boolean);
{ En mode : *  "normal = true" : Affiche les stats de la créature
*  "normal = false" : Demande au joueur de choisir une compétence à augmenter}
procedure infirmerie( var monCompte : Compte);
{Choix du nombre de PV à échanger contre de l'argent}
procedure arene( listing : listCompte ; var monCompte : Compte);
{initialise le coeur du jeu : l'arène où vont s'affronter deux créatures}
```

Fonctions utiles dans la procédure arène

```
function adversaireAleatoire( listing : ListCompte; niveau : integer) : Creature;
{Choisi un adversaire aléatoire parmi la liste des comptes
```

```
dans un fourchette de = ou - 2 niveau}
procedure gagnerXP(gainXP : integer; var monCompte : Compte);
{Calcule l'xp gagner à la suite d'un combat}
function calculXP(niv_adv , monNiv : integer) : integer;
{Ajoute l'xp à la créature et le fait monter de niveau si cela est nécessaire}
procedure choixAmeliorations(quantite : integer; var monCompte : Compte);
{procédure de choix d'amélioration lorsque la créature a passé de niveau}
procedure combat( var adversaire, maCreature : Creature);
{procédure orchestrant le déroulement de toute la phase de combat dans le jeu}
```

Fonctions utile dans la procédure combat

```
function utiliserSpe():boolean; {gère l'utilisation de la spé}
function choixCible(monTour : Boolean): Cible; {gère le choix des cibles}
function calculDmg(attaquant, defenseur : Creature; speA, speD: boolean) : integer;
{fonction de calcul des dommages reçus}
function esquive(speed_attack, speed_defense : integer) : Boolean;
{fonction gérant l'esquive}
```

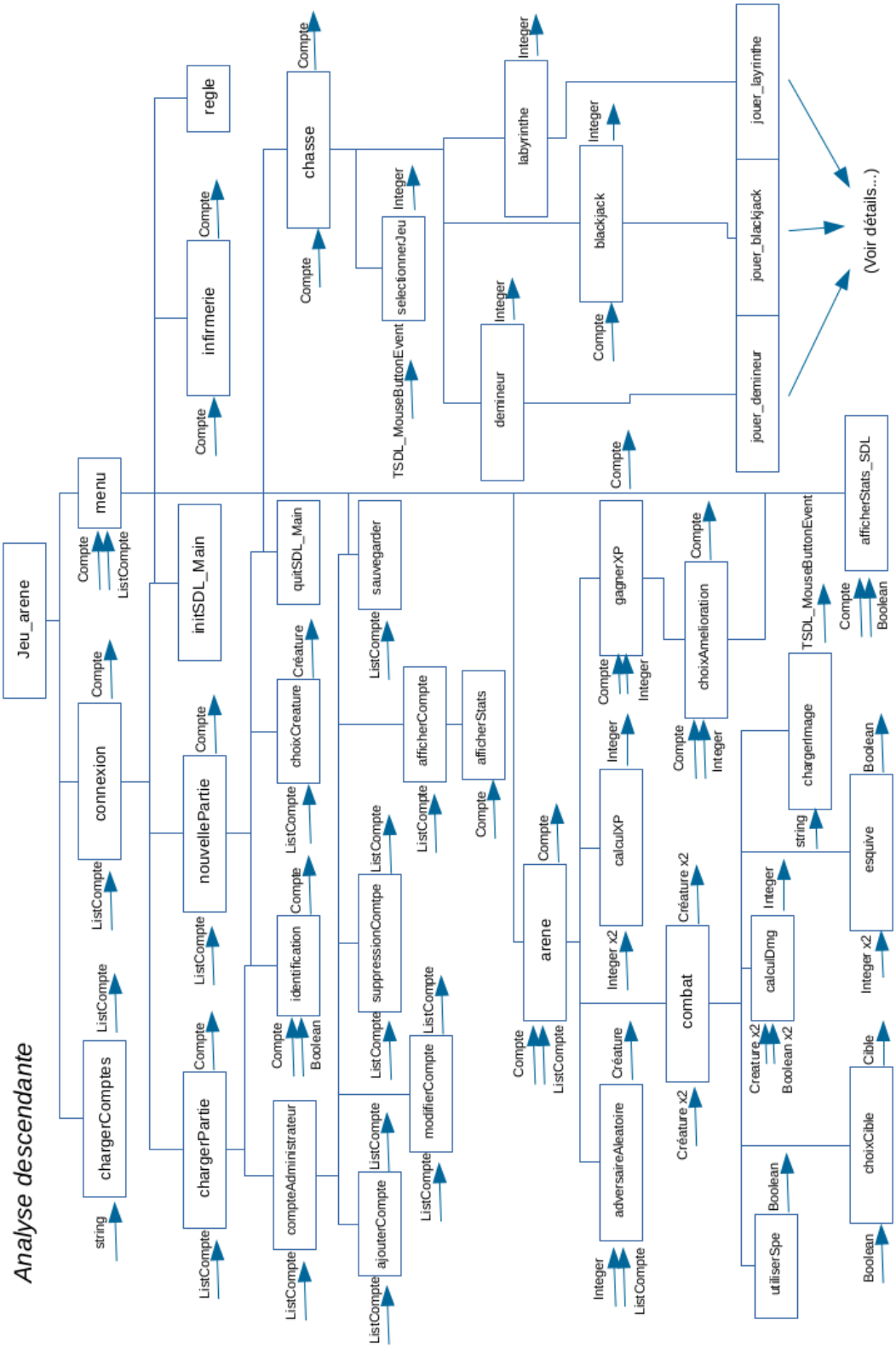
Unité jeu_admin

```
procedure compteAdministrateur(listing : ListCompte); {menu du compte administrateur}
procedure ajouterCompte(var listing : listCompte); {permet d'ajouter un compte}
procedure modifierCompte(var listing : listCompte); {permet de modifier un compte}
procedure suppressionCompte(var listing : listCompte); {permet de surrpimer un compte}
procedure afficherCompte(listing : listCompte); {affiche les vairiables des comptes}
```

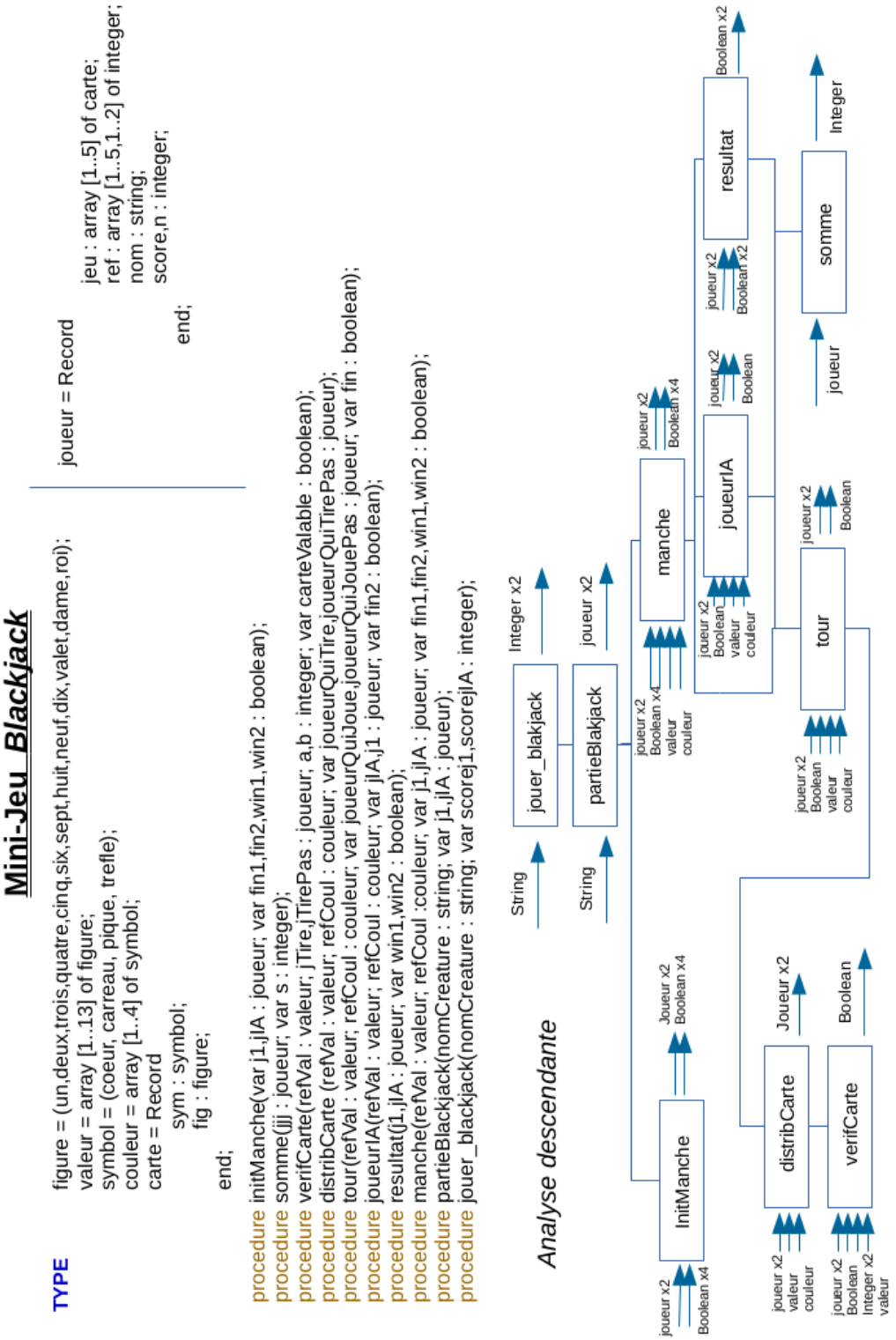
Unité jeu_fonctionnalite

```
procedure sauvegarder(listing : ListCompte);
{Enregistrer le tableau de compte dans un fichier et donc la progression du joueur}
procedure afficherStats(v_compte : Compte);
{Afficher sur le terminal les stats d'une créature
(utilisée seulement dans l'unité administrateur)}
function chargerImage(img : String): PSDL_Surface;
{Charger l'image correspondant à celle de notre créature
(voir afficherStats_SDL et combat)}
procedure ecrire(ecran : PSDL_Surface; txt : String; taille,color,posx,posy : Integer);
{Fonction TRES utilisée qui permet d'écrire sur l'écran SDL ce que l'on veut,
où on veut et avec la taille et la couleur au choix
* taille = taille de police ordinaire
* color : 1 = rouge / 2 = bleu / 3 = vert / autre = noir
* posX, posY en pixels
* Elle permet de nous simplifier l'écriture avec SDL}
procedure identification(var monCompte : Compte; nouveau : Boolean);
{Création d'un compte dans le cas d'un nouveau joueur (nouveau = true)
et Récupération d'un compte dans le cas d'un joueur existant (nouveau = false)}
```

Analyse desendante du jeu principal



Définition des types, signatures des procédures et fonctions, et analyses descendantes des mini-jeux



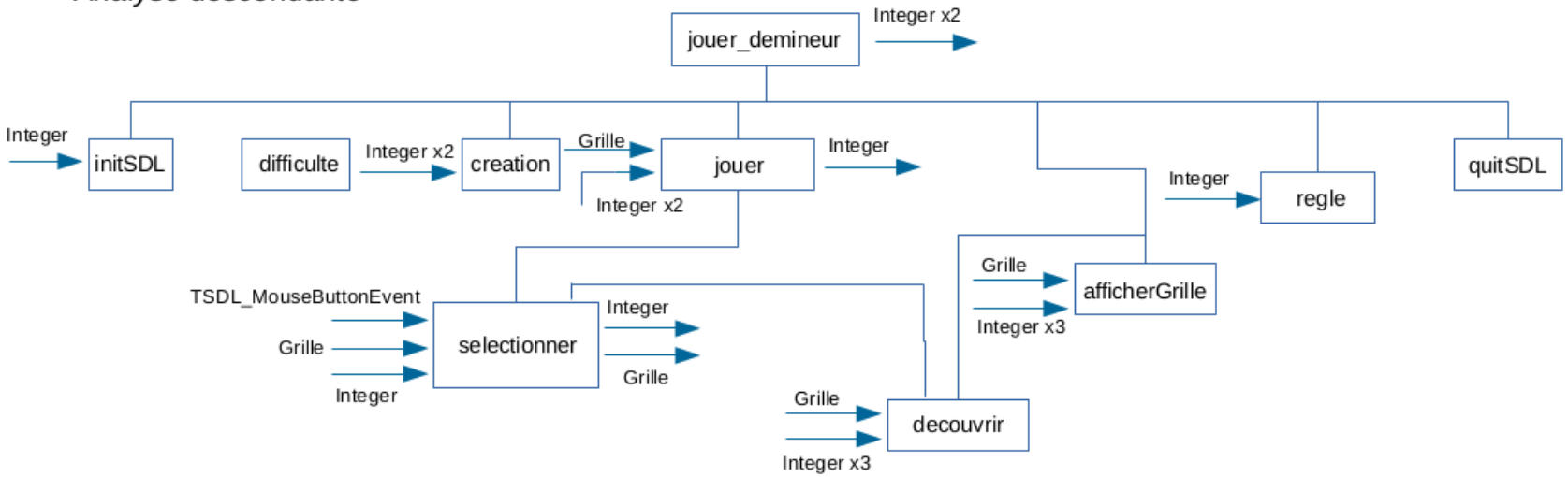
Mini-Jeu *DEMINEUR*

```
CONST   MAX=40;
        COTE = 32;

TYPE    Face = array[1..2] of char;
        Grille = array[1..MAX,1..MAX] of Face;
```

```
procedure jouer_demineur(var taille, bombe : Integer);
procedure initSDL(taille_demineur : integer);
procedure difficulte(var taille, bombe : integer);
procedure creation( taille, bombe: integer; var demineur: grille);
procedure regle(taille : integer);
procedure decouvrir(x,y , style : integer;  demineur: Grille);
procedure afficherGrille(taille, style: integer; dem:grille);
procedure selectionner(mouseEvent:TSDL_MouseButtonEvent; var demineur : Grille; var reste : Integer);
procedure jouer(demineur : Grille; taille : integer; var bombe : integer);
procedure quitSDL;
```

Analyse descendante



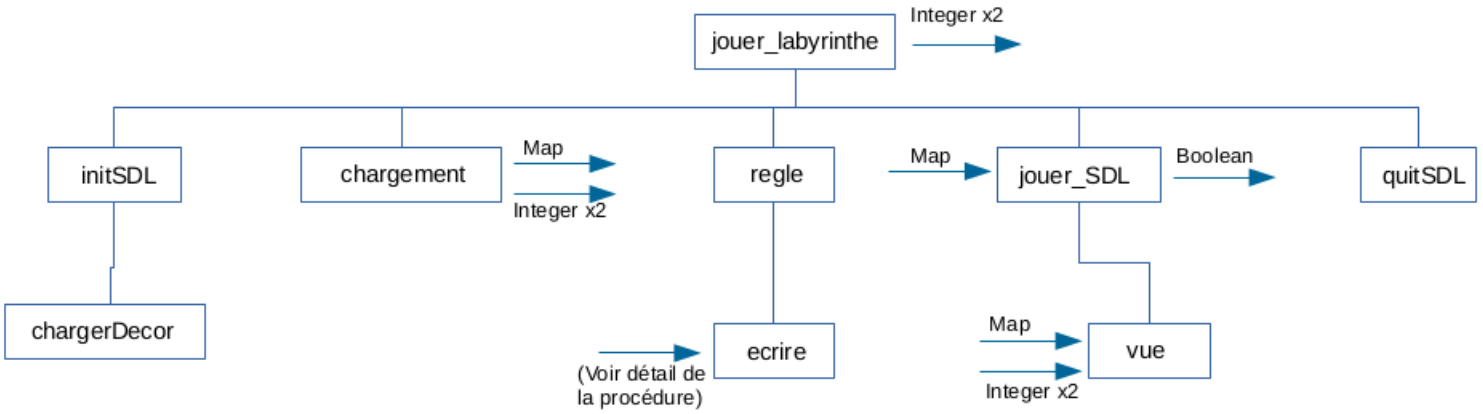
Mini-Jeu *LABYRINTHE*

```
CONST   MAX = 1000;
        COTE = 64;
        Champs_Vision = 7;

TYPE    Map = array[1..MAX,1..MAX] of Char;

procedure jouer_labyrinthe(var tailleX, tailleY : integer);
procedure initSDL;
procedure chargerDecor;
procedure chargement(var laby : Map; var longueur, hauteur : Integer);
procedure regle;
procedure vue(laMap: Map; posX,posY: integer);
procedure jouer_SDL(laMap : Map; var gagner : Boolean);
procedure quitSDL;
```

Analyse descendante



Guide d'utilisation

Exécutez le fichier Jeu_arene.pas. Une fenêtre s'ouvre alors et vous propose soit de charger une partie déjà existante, soit de créer une nouvelle partie.

Si votre compte existe déjà, cliquez sur « charger partie ». Vous devez entrer ensuite le nom de votre compte et votre mot de passe.

Si votre compte existe déjà, cliquez sur « nouvelle partie ». Vous devez choisir un nom de code et un mot de passe, puis choisir une créature parmi six qui sera la seule créature asociée à votre partie.

Une créature est définie par ses trois compétences, à savoir : la force (capacité à faire des dégâts), la résistance (capacité à subir les dégâts) et l'agilité (capacité à attaquer en premier et à esquiver les attaques). Chaque créature possède également une capacité spéciale. Il en existe trois : deux fois plus de force lors d'une attaque, la vision de la défense de l'adversaire, et une défense assurée lors d'une attaque de la créature adverse. La capacité spéciale n'est utilisable qu'une seule fois par combat. On ne peut pas changer sa capacité spéciale, il faut donc bien choisir lors de la création de sa partie.

Une fois connecté à votre compte, le menu s'affiche. Vous pouvez circuler librement dans les différentes parties du jeu : le combat, l'infirmerie, la chasse, les statistiques, les règles, la sauvegarde. Il y a également la possibilité de sortir du jeu avec le bouton « quitter ». La partie centrale de notre jeu est le combat dans l'arène. La créature adverse est choisie aléatoirement parmi les autres créatures d'un niveau proche de celui de votre créature. La première créature à attaquer est celle avec le plus d'agilité. Au début de chaque tour, il vous est proposé d'utiliser votre capacité spéciale (qui n'est utilisable qu'une seule fois par combat !). Après avoir choisi d'utiliser sa capacité ou non, vous devez, tour à tour, choisir la zone où vous voulez attaquer et la zone où vous voulez défendre votre créature. Le combat se termine lorsque l'une des deux créatures n'a plus de pv. Une fois que vous avez enchaîné assez de combat, votre créature gagne un niveau et vous avez la possibilité d'améliorer une ou plusieurs de vos trois compétences.

A la fin de votre combat, votre créature a sûrement peu de vie, il faut donc aller la soigner à l'infirmerie, en cliquant sur « infirmerie » dans le menu. Pour être soigné, vous devez payer l'infirmière avec quelques pièces d'or. Pour quitter l'infirmerie, cliquez sur la croix en haut à gauche de la fenêtre. Si vous n'avez pas de pièces d'or, il faut en gagner : la chasse est là pour ça.

En cliquant sur « chasse » dans le menu, quatre mini-jeux vous sont proposés : un labyrinthe dans lequel vous devez amener le lapin à trouver une des deux carottes disponibles, un démineur de taille et difficulté aléatoire, un black-jack et un jeu de clic où un clic équivaut à une pièce d'or. Pour quitter un mini-jeu ou le menu chasse, cliquer sur la croix en haut à gauche de la fenêtre.

Les statistiques de votre créature sont consultables à tout moment en cliquant sur « statistiques » dans le menu. Pour quitter les statistiques, cliquer sur la croix en haut à droite de la fenêtre. Une partie « règles » est aussi disponible dans ce même menu. Pour sauvegarder votre partie il suffit de cliquer sur « sauvegarder » dans le menu. Enfin, pour quitter le programme, cliquer sur « quitter » mais n'oubliez pas de sauvegarder avant !

De plus, il existe un compte administrateur permettant de gérer la liste complète des comptes des joueurs. Une fois dans le menu administrateur, on peut créer ou supprimer un compte, afficher un ou la liste complète des comptes et sauvegarder les changements. Pour accéder au compte utilisateur le nom de compte et le mot de passe sont définis dans le code source.

Les perspectives d'avenir

C'est en effet la question qui nous trotte tous à la tête. Que pourrait-on faire pour améliorer ou enrichir le projet ? Quelles sont les perspectives d'avenir ? Et bien laissez nous vous dire qu'elles sont nombreuses ! Évidemment, nous n'allons pas toutes les citer, mais on peut tout de même en lister quelques unes.

Tout d'abord, une des ambitions premières serait de faire en sorte que tout joueur puisse jouer sur son compte depuis différents ordinateurs. Pour cela il faudrait que le fichier « Comptes_Utilisateurs » soit accessible depuis différents ordinateurs. Par exemple via un dossier de partage d'un réseau (réseau de l'INSA par exemple) ou encore via un serveur privé. Cependant nous pouvons imaginer que nous rencontrerions des problèmes d'écrasement de données lors de l'exécution de la fonction « sauvegarde » si plusieurs joueurs jouaient en même temps (celle-ci réenregistre toute la liste de compte au lieu d'enregistrer seulement notre partie). Il faudrait donc modifier cette dernière procédure.

D'autre part, plusieurs modifications pour améliorer la jouabilité peuvent être implémentées :

- Mettre des boutons « Retour » au lieu de devoir cliquer sur la croix pour revenir en arrière.

- Pouvoir choisir le niveau de difficulté du labyrinthe. On peut en partie déjà le faire en passant par le code source et en augmentant ou diminuant la valeur de la constante Champs_Vision de l'unité chasse_labyrinthe (exemple : mettre entre 3 et 15 au lieu de 7).

-Faire en sorte que, lorsque le joueur réussit à trouver la carotte du labyrinthe, ce dernier change pour la fois d'après.

-Pouvoir choisir le niveau de difficulté du démineur. La dimension (4x4 $<$ dimension $<$ 40x40) et la difficulté (10 % $<$bombes$<$ 25 %) du démineur sont générées aléatoirement. Il serait plus « sympa » de pouvoir choisir entre : très facile, facile, moyen, difficile, très difficile ou KAMIKAZE.

- Faire un gros travail sur les statistiques (le calcul des dommages, du gain d'expérience, de l'esquive, etc.) pour éviter les problèmes de non-jouabilité au bout d'un certain niveau (ex : esquive systématique si trop de rapidité).

Avant de terminer cette partie, ajoutons tout de même deux ou trois perspectives qui pourraient augmenter largement l'intérêt du jeu.

D'un point de vue de la programmation nous pourrions réaliser une mini-IA en ajoutant une variable à l'enregistrement Compte qui serait un Tableau de cible et qui récupérerait la totalité des cibles choisies par l'utilisateur pour que l'ordinateur n'utilise pas un Random pour choisir sa cible mais choisisse en fonction de celui qui joue. Du point de vue du jeu en lui même, nous pourrions faire en sorte que deux joueurs, sur un même ordinateur, puissent combattre l'un contre l'autre, en mode 1 vs 1, avec chacun la créature de leur compte respectif. Nous pourrions faire en plus de l'infirmerie, une sorte de magasin où il est possible d'acheter des équipements ou des compétences supplémentaires (un peu comme dans le jeu Pokemon). Nous pourrions faire en sorte de pouvoir changer le nom de notre créature et notre mot de passe, de rajouter du son lors des combats et en fond, de pouvoir diriger plusieurs créatures, de pouvoir échanger nos créatures avec d'autres utilisateurs ou contre de l'argent, etc.

Ainsi les perspectives d'évolution du jeu son nombreuses.

Travail de groupe

Notre groupe de trois s'est constitué assez rapidement lors de la première séance. La recherche du sujet a été plutôt rapide, nous avions en effet plus ou moins les mêmes motivations et idées : réaliser un jeu assez sophistiqué et "cool". On a donc décidé d'inventer notre propre jeu en s'inspirant d'autres déjà existant, comme Pokemon par exemple.

Pour réaliser notre programme, nous avons fait tout d'abord une ébauche d'analyse descendante lors des premières semaines. Durant 3 heures, enfermés dans une salle de cours, nous avons inventé, modifié, débattu, analysé, critiqué et élaboré à la fois les types, et l'analyse descendante. Il faut avouer que le travail d'harmonisation des idées a été plus long que nous l'avions imaginé. La rédaction en Pascal des procédures et fonctions de notre jeu a été, du moins au début, quelque peu désordonnée. En effet, certaines procédures ont été écrites/codées sans que la répartition des tâches n'ait été explicitée. Ceci doit nous avoir fait comprendre dès le début que dans un travail de groupe, une bonne communication et une concertation mutuelle fréquente sont indispensables. De même, nous avons appris qu'un travail de groupe implique des prises de décisions collectives, et que si elles ne le sont pas, alors cela perturbe l'ambiance entre collaborateurs, et donc la bonne marche du projet.

Toujours dans l'esprit du travail de groupe, nous avons vite appris à commenter nos lignes de code pour faciliter non seulement la relecture des autres membres du groupe, mais aussi pour faciliter la mise en commun des différentes parties codées.

En lien avec la SDL, et pour rendre notre jeu toujours plus attirant, nous avons esquissé quelques dessins représentant les différents types de personnage du jeu. La SDL a été une difficulté de programmation supplémentaire car il fallu apprendre de zéro à utiliser la bibliothèque SDL. Enfin, la réalisation de ce rapport, qui marque la fin de notre projet, a été faite de façon à structurer notre pensée quant au travail effectué, et nous permet ainsi de dresser un bilan dont nous sommes relativement satisfaits.

Conclusion

À la fin de ce projet, nous en concluons que ce fut une expérience vraiment positive. Pendant près de quatre mois nous avons travaillé sur un projet qui nous plaisait tout en approfondissant nos connaissances en programmation.

En effet, en plus de l'application de notre apprentissage en cours de I1 et de I2, nous nous sommes entreprit d'apprendre à utiliser la bibliothèque SDL, notre première rencontre avec l'utilisation d'une interface graphique.

Nous avons aussi pu apprendre à travailler en groupe sur un projet sur moyen terme, à respecter un cahier des charges et une échéance.

Cependant, nous avons pris beaucoup de plaisir dans ce projet car ce jeu nous a progressivement tenu à cœur, et le voir enfin finalisé nous emplit un peu de fierté. De plus, il était très amusant et intéressant de concevoir un jeu dans sa globalité en partant de rien. Nous avons eu de nombreuses discussions palpitantes avant de pouvoir enfin se mettre à coder.

Nous sommes finalement fiers de pouvoir vous présenter notre projet informatique et nous espérons que vous aurez beaucoup de plaisir à l'utiliser tout comme nous en avons eu à y jouer et à le concevoir.