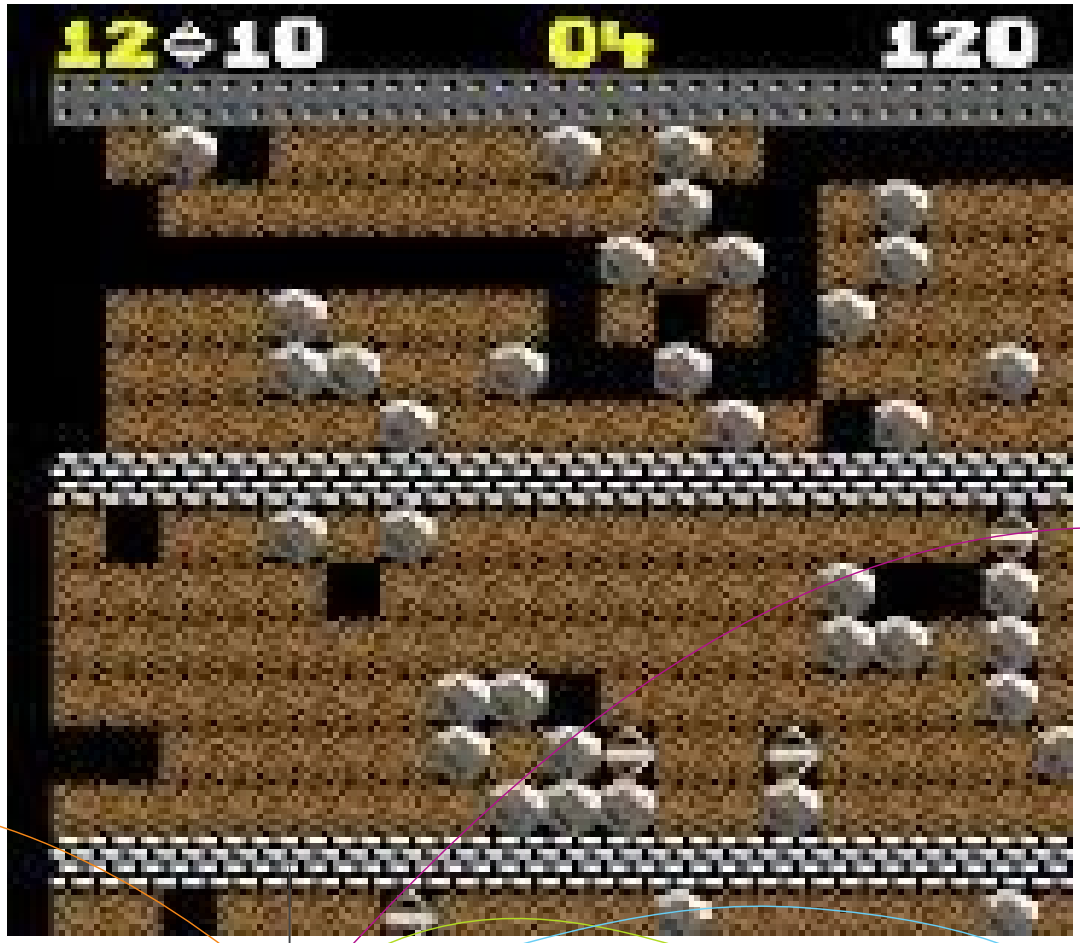
INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
ROUEN NORMANDIE



Nils Cercariolo
Lucien Coste
Adrien Pasquesoone
Djan Yigit
Roxane Leduc
Benjamin Fontaine

Analyse du projet

Jeu choisi :
Boulder Dash

Table des matières

1 Cahier des charges

1.1 Description du projet

Le groupe a décidé de réaliser le jeu Boulder Dash en I4.2. Le joueur contrôle Rockford, qui doit creuser dans des grottes pour collecter des pierres précieuses et des diamants et atteindre la sortie dans un temps limité tout en évitant divers types de créatures dangereuses ainsi que les chutes de pierres et le danger constant d'être écrasé ou piégé par une avalanche ou tué par une explosion souterraine.

1.2 Liste des fonctionnalités

Le rendu final de notre projet comportera les fonctionnalités suivantes :

- Gérer le menu (Commencer une partie, tableau des scores, gestion des profils (nom + raccourcis), quitter)
- Création du niveau (Convertis un fichier texte en affichage SDL)
- Se déplacer en haut, bas, gauche, droite + creuser un carré devant soi (En fonction des raccourcis du profil)
- Gérer les vies (Gain de vie, perd la partie si plus de vie)
- Gérer de la physique des pierres et des points (Les pierres roulent sur le côté sauf sur la terre, les points et les pierres sont soumis à la gravité)
- Gérer le temps imparti (150 secondes par niveau) (Affichage du chronomètre, perd si le temps est écoulé)
- Gestion des collisions (Pousser une pierre si il n'y a rien derrière, on ne peut pas creuser les murs, tenir les pierres sauf si elles ont une case de chute)
- Gérer la sortie (Passage au niveau suivant)
- Gérer le score (Calcul et affichage du score, enregistrement du meilleur score en fin de partie)
- Gestion de plusieurs niveaux (Charge un autre niveau parmi une banque de niveaux)

Nous avons prévu des fonctionnalités optionnelles, qui ne seront implémentées que s'il nous reste du temps à la fin du semestre :

- Gestion du son (Musique d'ambiance, événements)
- Gestion des animations (Entrée du niveau (porte d'entrée), explosion, sortie du niveau (porte de sortie))
- Gestion de la pause (Touche pause qui permet de mettre en pause le jeu)
- Gestion de l'enregistrement (Enregistre le nom, les raccourcis, le meilleur score)
- Apparition et gestion de monstres

Versions prévues :

V1 (05/04/2021) :

— Affichage et génération du plateau, Déplacements du joueur et physique des pierres et des minerais à collecter, Sélection des textures, Gérer la victoire

V2 (30/04/2021)

— Création et gestion du menu, Gestion de la mort, Gestion du temps, Gestion du score, Gestion de plusieurs niveaux, Gestion de l'enregistrement

V3 Finale (30/05/2021)

— Gestion du son, Gestion de pause, Gestion des animations

2 Conception globale

L'analyse descendante correspondante est donnée en figure 6.

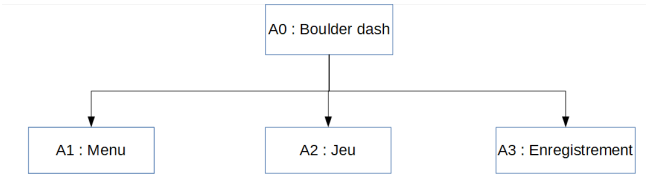


FIGURE 1 – Analyse descendante du projet.

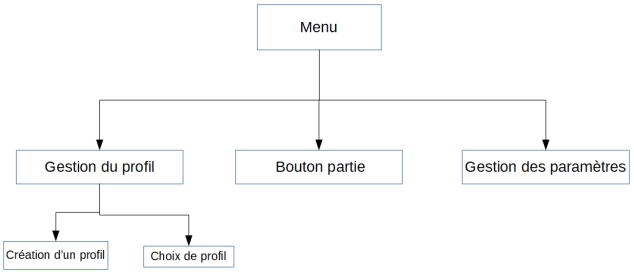


FIGURE 2 – Analyse descendante du projet.

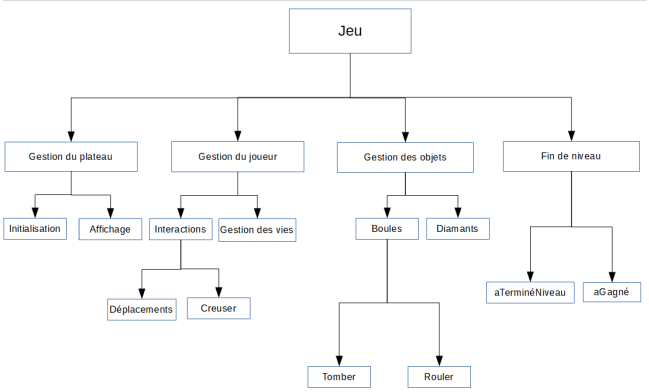


FIGURE 3 – Analyse descendante du projet.

3 Conception préliminaire

```
Types :

Parametre : Structure
Haut, Bas, Droite, Gauche, Creuser : Integer

Profil : Structure
Nom : String
Touches : Parametre

Vecteur : Structure
x, y : Integer

Entites : Structure
Coord : Vecteur
Texture : psdl_texture

Plateau : array [1..LargeurMax, 1..LongueurMax] of Entites

Rockford : Structure
Entite : Entites
Vie, Score, compteurNiveau : Integer

Signatures :

procedure p_Tomber(E/S p : plateau)

procedure p_Rouler(E/S p : plateau)

procedure p_Boules(E/S p : plateau)

procedure p_Diamants(E/S p : plateau, joueur : Rockford, compteurDiamant : Integer)

procedure p_GestionDesObjets(E/S p : plateau, joueur : Rockford, compteurDiamant : Integer)

procedure p_Deplacements(E para : Parametres; E/S p : plateau; joueur : Rockford)
```

```
procedure p_Creuser(E para : Parametres; E/S p : plateau; joueur : Rockford)

procedure p_Interactions(E para : Parametres; E/S p : plateau; joueur : Rockford)

procedure p_GestionDesVies(E/S p : plateau; joueur : Rockford)

procedure p_GestionDuJoueur(E para : Parametres; E/S p : plateau; joueur : Rockford)

fonction f_aTermineNiveau(E p : plateau; joueur : Rockford) : Booleen

fonction f_aGagne(E joueur : Rockford) : Booleen

procedure p_FinDeNiveau(E p : plateau; joueur : Rockford)

fonction f_Initialisation(E fichier : Text) : plateau

procedure p_Affichage(E p : plateau)

procedure p_GestionDuPlateau(E p : plateau; fichier : Text)

procedure p_Jeu(E para : parametres; fichier : Text; E/S p : plateau; joueur : Rockford)

procedure p_Menu(E nomJoueur : String; para : parametres; E/S profilJoueur : Profil)

procedure p_CreationDeProfil(E nomJoueur : String; E/S ProfilJoueur : Profil)

procedure p_ChoixDeProfil(E: nomJoueur : String; E/S ProfilJoueur : Profil)

procedure p_GestionProfil(E nomJoueur : String; E/S profilJoueur : Profil)

procedure p_BoutonPartie()

procedure p_GestionParametres(E para : parametres; E/S profilJoueur : Profil)

procedure p_Enregistrement(E profilJoueur : Profil, joueur : Rockford)
```

4 Introduction

Dans le cadre de ce projet à effectuer au cours de notre deuxième année à l'INSA, nous avons choisi de réaliser une version numérique du jeu Boulder Dash. Lors de la rédaction du cahier des charges, nous nous sommes fixés pour objectif d'être, à terme, en mesure de contrôler Rockford (un mineur) qui doit creuser dans des grottes afin de collecter des diamants et atteindre la sortie dans un temps limité, tout en évitant divers types de créatures dangereuses ainsi que les chutes de pierres et le danger constant d'être écrasé ou piégé par une avalanche ou tué par une explosion souterraine.

Ce rapport présente de manière détaillée nos réflexions, de la conception de l'analyse descendante jusqu'à la réalisation du jeu. Nous commencerons donc par analyser la conception globale « revisitée » de notre programme, ce qui nous permettra de revenir sur les différents changements que nous y avons apportés. Dans un second temps, nous présenterons certaines procédures et fonctions de notre jeu, puis, nous expliquerons brièvement ce qui est mis en œuvre dans chacune de nos unités. Ensuite, nous effectuerons un retour sur notre travail de groupe et notre organisation. Pour finir, nous évoquerons les difficultés que nous avons pu rencontrer lors de la programmation, ainsi que les améliorations et perspectives envisageables

5 Nouvelle analyse descendante

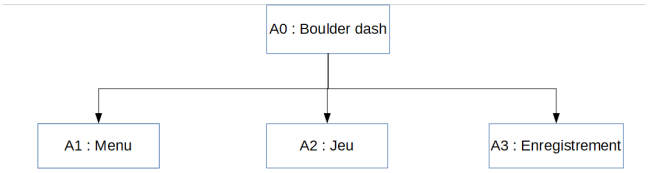


FIGURE 4 – Nouvelle analyse descendante du projet.

Nous avons donc modifié quelque peu notre analyse descendante mais la majeure partie y est restée. Comme il n'y a pas qu'une seule analyse descendante, nous nous sommes permis de modifier des emplacements de procédures lorsque cela ne changeait rien. Nous avons décidé de supprimer la procédure d'enregistrement des raccourcis claviers par manque de temps. Nous avons cependant mieux détailler la partie concernant les menus. La gestion de niveau s'est faite au niveau du joueur car cela correspondait mieux aux variable associées au joueur comme nous vérifions chaque image son emplacement. Les 2 méthodes marchent bien donc nous avons privilégié la plus simple et efficace.

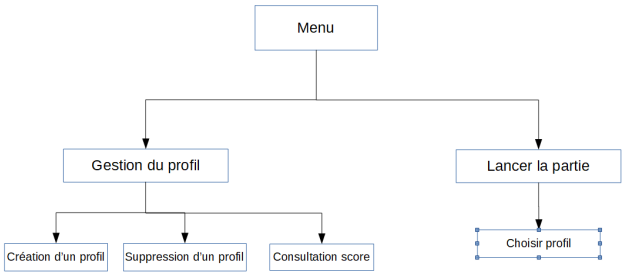


FIGURE 5 – Nouvelle analyse descendante du projet

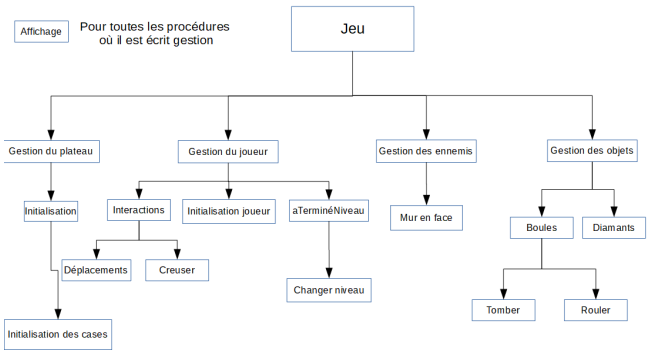


FIGURE 6 – Nouvelle analyse descendante du projet

6 Présentation de certaines procédures et fonctions et description des unités

Dans l'unité :

— playerunit

Cette première unité a pour objectif de réunir toutes les fonctions et procédures ayant un lien avec la gestion du joueur. On y retrouve les procédures et fonctions gérant ces animations, le système de mort et de game over, ces déplacements... La majorité des procédures de cette unité modifie l'état de la variable player qui correspond à une structure regroupant toutes les caractéristiques du joueur.

Les procédures sont :

— initializePlayer

Cette procédure à pour but de fixer les différentes textures associées au personnage jouable via la procédure initializeSprite. Elle a également pour fonction de fixer les valeurs de base caractérisant l'état du jeu, à savoir le niveau en cours, le nombre de vies restantes, le score, le nombre de diamants récupérés, l'état de la mort, et l'apparition de la porte de sortie. Enfin la procédure initialise la position de départ du joueur par un appel à la procédure initializePlayerPosition.

— handlePlayerMovement

Cette procedure s'occupe de toute la gestion des mouvements du personnage. La procedure se charge de récupérer les inputs saisis par le joueur et d'effectuer l'action associée. Le personnage peut en effet creuser une case située à côté de lui par appel à la procédure creuser. Le joueur peut également choisir de déplacer le personnage sur une case adjacente ce qui nécessite l'appel de la procédure canPlayerMove afin de vérifier que la case choisie par le joueur pour déplacer son personnage n'est pas occupée par un obstacle. La procédure checkCollisions est également appelée afin de permettre au personnage d'interagir avec son environnement en poussant une pierre ou en récupérant un diamant par exemple. Enfin la procédure animatePlayer permet de selectionner la bonne texture en fonction des mouvements effectués par le personnage.

— drawPlayer

Cette procédure sert à afficher les textures associées au personnage.

— checkLevelFinish

Cette procédure se charge de vérifier si le joueur a terminé le niveau afin de passer au suivant quand tel est le cas grâce à la procédure changeLevel. La procédure calcule également le score du joueur en fonction du temps restant

— checkPlayerDeath

Cette procédure s'occupe de déterminer dans un premier temps si le joueur est dans une situation où il doit mourir ou non. Si tel est le cas alors le booléen représentant l'état de mort du personnage passe à True. Dans un deuxième temps, si le joueur est mort la procédure permet de déclencher l'animation de mort du personnage et permet de réessayer le niveau grâce à la procédure retryLevel.

Dans l'unité :

— drawunit

Cette seconde unité prend en charge l'affichage des différents éléments relatif à l'environnement du jeu en dehors du personnage contrôlé par le joueur. Les différentes procédures composant cette unité s'appuient sur la SDL pour gérer l'affichage. On retrouve également dans cette unité différents types personnalisés permettant de représenter la map et les différentes cases du plateau pouvant représenter différents éléments de décors avec lesquels le joueur pourra interagir.

Les procédures sont :

— initializePlateau

Cette procédure a pour objectif d'initialiser le plateau de jeu avant le début de la partie. Pour cela elle fait appel à la procédure initializeCase qui associe à chaque élément du décor une texture ou un tableau de sprite si l'élément est amené à être animé. Cette dernière permet aussi d'associer chaque case du plateau à ses caractéristiques (index, dimensions...) en se basant sur les dispositions de map stockées dans les fichiers du jeu par le biais de la procédure loadMapFromFile.

— drawPlateau

Cette procédure permet de rendre les différentes textures et sprites associés à chaque élément du plateau si ces derniers sont visibles selon l'angle de caméra actuel.

— drawGrid

Cette procédure permet d'afficher une grille sur le plateau afin de séparer les différentes cases.

— showPlateauText

Cette procédure affiche un plateau dans le terminal. Chaque élément sera représenté par une lettre correspondant à l'index qui lui est associé.

— destroyPlateTextures

Cette procédure permet de détruire les différentes textures chargées en RAM. Elle est appelée à la fin de chaque itération du programme principal afin de libérer de l'espace mémoire.

Dans l'unité :

— cameraunit

Cette unité réunit les différentes procédures qui permettent de gérer le déplacement de la caméra en fonction des déplacements du joueur sur le plateau de jeu.

Les procédures sont :

— initCamera

Cette procédure initialise la position de la caméra au début d'une partie ainsi que ses dimensions.

— applyCamera

— updateCamera

Cette procédure permet de rafraichir la caméra.

Dans l'unité :

— hudunit

Cette unité permet d'afficher le HUD du jeu, à savoir toutes les informations relatives au score, au nombre de points de vie du joueur etc.

Les procédures sont :

— initializeHUD

Cette procédure a pour rôle de charger les différents éléments nécessaires à l'affichage du HUD, à savoir la police d'écriture, les informations à afficher, la couleur du texte à afficher ...

— drawInfo

Cette procédure permet de rendre les différentes informations relatives au joueur, à savoir le nombre de vie restant et le numéro du niveau.

— drawHUD

Cette procédure permet de rendre les différentes informations du HUD, à savoir le score du joueur, les diamants restants à ramasser et le temps restant.

Dans l'unité :

— physiqueunit

Le rôle de cette unité est de gérer la physique des éléments mouvants du décors à savoir les diamants et les pierres.

Les procédures sont :

— updateEntities

Cette procédure est appelée à chaque tour de boucle du programme principal et a pour fonction de vérifier si les pierres et les diamants du plateau doivent être mis en mouvement. Si l'élément doit être mis en mouvement il peut alors soit tomber, on fera alors appel à la procédure makeEntityFall soit glisser et on fera alors appel à la procédure makeEntitySlip.

Dans l'unité :

— enemiesUnit

Le rôle de cette unité est de gérer les déplacements des ennemis.

Les procédures sont :

— updateEnemy

Cette procédure a pour rôle de contrôler les mouvements des ennemis. A l'aide de la fonction IsMurInFace, la trajectoire de l'ennemi est rectifiée si ce dernier se dirige vers un mur.

Dans l'unité :

— menus

Cette unité regroupe les fonctions et procédures s'occupant de la gestion des menus et de leur affichage.

Les procédures sont :

— mainMenu

Cette procédure a pour fonction d'afficher le menu principal du jeu qui permet d'accéder aux options suivantes : jouer, profil ou quitter. Les options sont affichées dans différents cadres cliquables. La détection du clique se fait par appel à la fonction inRectangle.

— profileMenu

Cette procédure a pour fonction de s'occuper de la gestion du menu des profils. Ce menu permet d'accéder aux options suivantes : création de profil, supprimer un profil, afficher le score. Les options sont affichées dans différents cadres cliquables. La détection du clique se fait par appel à la fonction inRectangle.

— afficherTexte

Cette procédure a pour fonction d'afficher le texte saisi en entrée dans une fenêtre sous format SDL.

— demandeNom

Cette fonction permet de demander au joueur de saisir son nom. Cette fonction est appelée à chaque fois qu'on demande à l'utilisateur de saisir son nom.

Dans l'unité :

— GestionProfil

Cette unité regroupe les procédures et fonctions utilisées pour la gestion de profils.

Les procédures sont :

— CreationProfil

Cette procédure permet de demander au joueur le nom qu'il souhaite donner à son profil, via un appel à la fonction demandeNom. Il est ensuite vérifié si ce nom de profil existe déja via la fonction ExistenceProfil. Une fois le nom saisi, il est stocké dans un fichier.

— SupprimerProfil

Cette procédure permet de supprimer un nom de profil dans le fichier stockant les différents noms de profil. Il est d'abord demandé au joueur de saisir le nom de profil qu'il souhaite supprimer via un appel à la fonction demandeNom. On vérifie ensuite l'existence du profil via ExistenceProfil et on supprime le profil demandé.

— AffichageScore

Cette procédure permet d'afficher le score stocké dans un fichier associé à un nom de profil, après vérification de l'existence de ce derneir.

— EnrengistrerScore

Cette procédure permet de stocker un score associé à un nom de profil dans un fichier.

7 Procédures en pseudo-code

```
Procédure animatePlayer
Les variables précédées de sprite sont des tableaux contenant les textures de l'animation
correspondante.

const : TICK_INTERVAL = 120


procedure animatePlayer(E/S p:Player, E : xDir,Dir : Naturel)

Declaration frame : Naturel

Debut
  Si p.isOnExit alors
    frame <- arrondir(GetTicks() / TICK_INTERVAL) modulo MaxIndice(animationVictoire)
    p.texture <- animationVictoire[frame + 1]
  Sinon Si p.isDeath alors
      frame <- arrondir(GetTicks() / TICK_INTERVAL) modulo MaxIndice(animationMort)
      p.texture <- animationMort[frame + 1]
    FinSi
  Sinon
    Si yDir = -1 alors
      frame <- arrondir(GetTicks() / TICK_INTERVAL) modulo MaxIndice(animationHaut)
      p.texture <- animationHaut[frame + 1]
    Sinon Si yDir = 1 alors
        frame <- arrondir(GetTicks() / TICK_INTERVAL) modulo MaxIndice(animationBas)
        p.texture <- animationBas[frame + 1]
      FinSi
    Sinon Si xDir = -1 alors
        frame <- arrondir(GetTicks() / TICK_INTERVAL) modulo MaxIndice(animationGauche)
        p.texture <- animationGauche[frame + 1]
      FinSi
    Sinon Si xDir = 1 alors
        frame <- arrondir(GetTicks() / TICK_INTERVAL) modulo MaxIndice(animationDroite)
        p.texture <- animationDroite[frame + 1]
      FinSi
```

```
    Sinon
      frame <- arrondir(GetTicks() / TICK_INTERVAL) modulo MaxIndice(animationStatique)
      p.texture <- animationStatique[frame + 1]
    FinSi
  FinSi
Fin

Fonction loadMapfromFile

Le type Map est un tableau de caractères de taille la taille maximum du plateau

fonction loadMapFromFile(E filename : String, plate : Plateau) : Map;
Declaration m : Map
fichier : Text
i, j : Naturel
ligne : String

Debut
  assigner(fichier,filename)
  reset(fichier)

  lire(fichier, plate.name)                ## les 3 premières lignes du fichier texte sont des
  lire(fichier, plate.remainingDiamond)       informations sur la map
  lire(fichier, plate.timeLeft)

  Pour i <- 1 à HauteurPLateuMAx faire        ## convertir le fichier texte en tableau
    lire(fichier,ligne);
    Pour j <- 1 à LongueurPlateauMax faire
      m[i][j] <- ligne[j]
      case ligne[j] of
        'P' : plate.spawnCoord[0] <- j
              plate.spawnCoord[1] <- i
              m[i][j] <- 'R'
        'E' : plate.exitCoord[0] <- j
              plate.exitCoord[1] <- i
      FinCase
    FinPour
```

```
  FinPour
  close(fichier)
  retourner m
Fin
```

8 Retour sur le travail de groupe

Nous avons formé un groupe de 6 élèves pour ce projet : Adrien, Lucien, Benjamin, Djan, Nils et Roxane. Les tâches ont été équitablement et systématiquement réparties, que ce soit pour la conception préliminaire, le code ou le rapport.

Concernant l'alternance présentiel/ distanciel, nous avons formé un sous-groupe de 3 élèves qui venait en présentiel, afin de poser les questions soulevées lors des sessions de travail précédentes et de montrer aux professeurs référents l'avancée de notre projet. En considérant les conditions de travail de ce créneau (bruit dans la salle, réseau Eduroam instable, pas toujours la possibilité de charger les ordinateurs portables ...), ce n'est pas durant ces sessions que nous avons le plus avancé.

Certains élèves du groupe étaient, au départ, plus à l'aise que d'autres en SDL grâce au Projet-Info réalisé au 3ème semestre de STPI. Le niveau en SDL dans le groupe était varié, ainsi certains ont développé des parties plus algorithmiques et fonctionnelles (comme la gestion des profils ou même le déplacement de Rockford), tandis que d'autres se sont penchés sur l'interface Humain-Machine (graphismes, affichages ...) par exemple. De manière générale, nous nous sommes tous intéressés à la SDL et avons appris le nécessaire.

Depuis le début de la crise sanitaire, nous avons été habitués à travailler à distance, même en groupe, et avons tout de suite pensé à créer un serveur discord dédié pour ce projet. Nous avons ajouté un salon audio pour les streams et les réunions, et des salons textuels pour du partage de documents, images, idées... Une fois le travail réparti et réalisé par chacun, ou dès qu'un membre avait quelque chose à présenter, nous nous réunissions sur un stream discord. On récoltait les avis et les remarques de chacun pour avancer sur un projet qui nous convenait à tous. Tous les membres du groupe appréciant coder et monter diverses projets, la réalisation de ce jeu s'est faite avec enthousiasme et bonne humeur. Cependant, nous aurions tout de même aimé avoir plus d'occasions pour travailler ensemble en présentiel, car le travail par discord reste moins efficace qu'une session de code en groupe. Une fois une version satisfaisante ou une modification majeure dans la structure réalisée, nous organisions le dossier de travail et le déposions dans notre Git. Nous avons souvent lors de la rédaction de procédures

simples ou de petites unités, partagé ces fichiers via notre serveur discord sans passer par Git. Nous avons cependant trouvé cet outil efficace et professionnel pour la gestion du projet dans sa globalité. Il était simple de voir où nous en étions, et on avait aussi ce sentiment de sûreté qu'apporte un serveur : cela permet d'éviter de perdre des documents et d'être certain que tout le groupe travaille bien sur la même version. Tous les membres du groupe ont créé un git repository sur leur machine, mais comme mentionné précédemment, nous avons utilisé cet outil pour des changements significatifs et des versions fonctionnelles.

Pour conclure, durant ce projet d'informatique nous nous sommes tous entre-aidés, concertés, et avons réussi à garder une bonne ambiance de travail malgré les différends et les erreurs dans le code.

9 Difficultés rencontrées et améliorations possibles

Ce projet nous a permis d'identifier plusieurs difficultés auxquelles nous avons dû faire face. Tout d'abord, installer la SDL a demandé beaucoup de temps pour certains d'entre nous. Les premières tentatives d'installation ne fonctionnaient pas et nous n'étions pas tous sous les mêmes systèmes d'exploitation, ce qui ne nous facilitait pas la tâche. Ensuite, gérer la position du Rockford via ses coordonnées (savoir où se situe le joueur et afficher la partie concernée à l'écran) n'a pas non plus été chose facile. La gestion des animations et des transitions nous a demandé beaucoup de travail car nous cherchions à les rendre plus ou moins fluides et la gestion du temps a aussi été problématique, puisque nous nous sommes rendus compte que les délais peuvent différer d'un ordinateur à l'autre (nous avons dû alors effectuer quelques modifications afin de rendre le code plus adaptable). Enfin, le dernier point que nous souhaitons mentionner dans cette partie concerne la physique des pierres et notamment les éboulements. En effet nous avons commencé par créer une procédure qui ne vérifiait que les cases où le joueur a pu avoir une action, seulement cette solution n'était pas compatible avec la mécanique des éboulements présente dans le jeu original qui peut survenir sur un rayon bien plus vaste autour du joueur. Nous avons donc transformé cette procédure afin qu'elle puisse vérifier à chaque itération du programme principal tous les éléments pouvant se déplacer. Un autre point qui a été difficile à gérer avec la physique des pierres a été la vitesse de chute de ces dernières. En effet, il a fallu trouver une solution afin de dissocier la vitesse de chute des pierres de la vitesse de déplacement du joueur afin de lui permettre de les éviter.

Nous sommes très satisfaits de notre projet, il comporte les fonctionnalités principales du jeu. Cependant, quelques améliorations sont toujours possibles. En effet, une refonte des menus peut être un axe d'amélioration car actuellement, la structure envisagée n'est pas optimale car on recréé une fenêtre à chaque action. Nous nous sommes particulièrement concentrés sur le jeu en lui-même. Une

autre piste d'améliorations pourrait être la mise en place de nouveaux niveaux ainsi que la gestion de la pause comme nous l'imaginions. Nous pouvons aussi imaginer ajouter d'autres types d'ennemis. Un mode expert avec une seule vie pourrait aussi être envisagé pour les joueurs plus expérimentés.

10 Conclusion

En premier lieu, ce projet de jeu Boulder Dash nous a permis d'améliorer considérablement notre niveau en programmation. C'est ensuite une expérience tout à fait enrichissante du point de vue du travail en équipe. Ainsi, nous avons appris à confronter nos idées et à débattre mais aussi à nous organiser et à respecter des délais.

L'exercice qui nous a été demandé nous sera utile dans le cadre de notre futur métier. En effet, un ingénieur doit avoir un profil complet afin de pouvoir aisément s'adapter à toutes formes de situations. Il doit pouvoir naviguer entre différents secteurs d'activités et ce, de manière rapide et efficace, afin d'être toujours plus productif. Faire preuve de productivité nécessite d'être en mesure de travailler en équipe. Il lui est impératif de savoir communiquer avec ses collaborateurs et de comprendre « l'autre », ses manières de travailler, mais aussi ses opinions et convictions propres. C'est pourquoi, au-delà d'avoir un aspect technique tout à fait intéressant, programmer ce jeu a été un bon moyen pour nous d'apprendre à nous écouter et à partager nos savoir-faire.