

PROJET L1 – C.M.I.
Spécialité informatique

Skizzle

Maëlle BEURET
Rémi CÉRÈS
Mattéo DELABRE

Année : 2015 – 2016
Soutenu le : 29/04/2016



Réseau Figure
CURSUS MASTER EN INGÉNIERIE

Table des matières

1	Introduction	2
1.1	Choix du jeu	2
1.2	Cahier des charges	3
2	Organisation du projet	4
2.1	Organisation du travail	4
2.2	Outils de développement	6
3	Analyse du projet	7
3.1	Découpage du projet	7
3.2	Découpage du code	8
4	Développement	12
4.1	Moteur physique	12
4.2	Niveaux du jeu	12
4.3	Interface du jeu	13
4.4	Univers graphique	13
5	Manuel d'utilisation	14
5.1	Menu du jeu	14
5.2	Objets	16
5.3	Jeu	17
5.4	Éditeur	18
6	Conclusion	20
6.1	Perspectives	20
6.2	Conclusions	20
	Webographie	22

Chapitre 1

Introduction

Dans le cadre du module Projet C.M.I. du second semestre de L1, nous avons développé en équipe un jeu vidéo nommé « Skizzle ». Notre groupe est composé de trois personnes : Maëlle BEURET, Rémi CÉRÈS et Mattéo DELABRE.

L'objectif de ce projet est la création d'un jeu vidéo fonctionnel. Le jeu mobilise les bases d'algorithmique apprises au premier semestre ainsi que nos connaissances et recherches personnelles. La création de ce jeu nous a permis de renforcer nos capacités de travail en collaboration, en communication et en gestion de projet en général.

Le développement du projet s'est déroulé sur une période d'un mois et une semaine : du vendredi 4 mars 2016 au lundi 11 avril 2016 inclus. Chaque vendredi, lors de la séance de trois heures consacrée au projet, les trois membres de l'équipe se réunissent pour résumer le travail effectué la semaine passée et planifier celui de la semaine à venir.

1.1 Choix du jeu

Nous avons d'abord réalisé une étude comparative entre trois jeux vidéos possibles. Pour ces jeux vidéos, le choix était libre. Notre choix s'est porté sur un jeu avec un principe original inspiré des jeux de plates-formes, de coopération et de réflexion. Nous l'avons appelé « Skizzle », à mi-chemin entre l'anglais *skill* et *puzzle*.

Le jeu se joue à deux joueurs. Chaque joueur est affecté à une balle qu'il contrôle par le clavier. Le but est pour ces joueurs de faire traverser leur balle à travers des niveaux prédéfinis. Pour ce faire, les joueurs doivent utiliser différents phénomènes physiques et constructions mises en place à la fois dans le niveau et dans le moteur physique du jeu.

Les niveaux se présentent sous la forme de casse-têtes courts à difficulté progressive. Les deux joueurs doivent souvent réfléchir et s'entraider pour pouvoir parvenir à la fin. Ce n'est que lorsque les deux joueurs franchissent la ligne d'arrivée que la partie est gagnée.

1.2 Cahier des charges

Le jeu doit fonctionner sur tous les systèmes courants (Linux, OS X, Windows). À l'ouverture du jeu, un menu doit permettre d'orienter le joueur vers les différents états de jeu disponibles, notamment le jeu ou l'éditeur. Avant d'ouvrir le jeu, on doit pouvoir choisir le niveau à jouer. Avant d'ouvrir l'éditeur, on doit pouvoir choisir le niveau à éditer ou si l'on veut créer un nouveau niveau.

Le jeu se joue à deux joueurs qui incarnent chacun une balle contrôlable au clavier. Une partie doit se présenter sous la forme d'un niveau où les joueurs ont une position initiale et une position à atteindre pour gagner. Chaque partie est limitée en temps. La durée limite est définie selon le niveau.

Les mécanismes physiques à implémenter sont la force d'attraction chargée (coulombienne), la force de gravité, les forces de frottement et les collisions entre objets. La force de gravité est appliquée selon un vecteur de norme constante mais de direction et sens modifiables selon les conditions du niveau.

La caméra des joueurs doit être centrée sur la position intermédiaire des deux joueurs et doit s'orienter dans la direction inverse de la direction actuelle de la gravité, en tout temps.

Chaque niveau est composé d'objets. Chaque objet possède une masse, un coefficient de frottement statique et dynamique, un coefficient de restitution, une charge, une position, une vitesse et un calque d'affichage (couche). Un objet est soit une balle d'un joueur soit un bloc. Un bloc peut être neutre ou posséder une particularité. Les blocs particuliers sont définis complètement dans la section 5.2.

Dans chaque niveau est définie une zone de jeu. Cette zone est un polygone contrôlé par un nombre arbitraire de points. Si un objet avec une masse non-infinie se trouve en dehors de cette zone (notamment un joueur ou bien un bloc déplaçable), il doit être immédiatement tué. S'il s'agissait d'un joueur, la partie se termine.

Les niveaux doivent être éditables par un éditeur. L'éditeur permet de créer un nouveau niveau ou d'éditer un niveau existant. Il permet de placer des objets prédéfinis sélectionnables depuis la barre d'outils. Il permet notamment de situer les positions initiales des joueurs et du(des) bloc(s) d'arrivée. On doit pouvoir modifier la polarité des objets depuis l'éditeur. Le niveau modifié doit pouvoir être sauvegardé. On doit pouvoir tester le niveau en cours d'édition depuis l'éditeur sans perdre son travail. On doit pouvoir modifier la taille de la zone de jeu.

Chapitre 2

Organisation du projet

2.1 Organisation du travail

Lors de la réunion initiale les tâches ont été réparties selon les capacités de chacun. En dehors des réunions, chaque membre du groupe a travaillé en autonomie, en communiquant son avancement au reste du groupe.

Les réunions suivantes étaient consacrées au passage en revue des modifications effectuées par chacun, à l'analyse des tâches restant à effectuer et à leur distribution entre les membres du groupe, toujours selon leurs capacités et leur disponibilité. La répartition finale des tâches telles qu'elles sont été accomplies est présentée dans la figure 2.1.

Certaines parties du développement nécessitaient plusieurs personnes et étaient ainsi partagées entre certains membres du groupe. La conception et les tests des niveaux, le fond des menus ainsi que les décors furent réalisés par Rémi et Maëlle, et la gestion de projet par tout le groupe.

Tout au long de la réalisation du projet, nous communiquions par Skype afin de s'informer de l'avancement et réfléchir à des solutions lorsqu'un problème était rencontré.

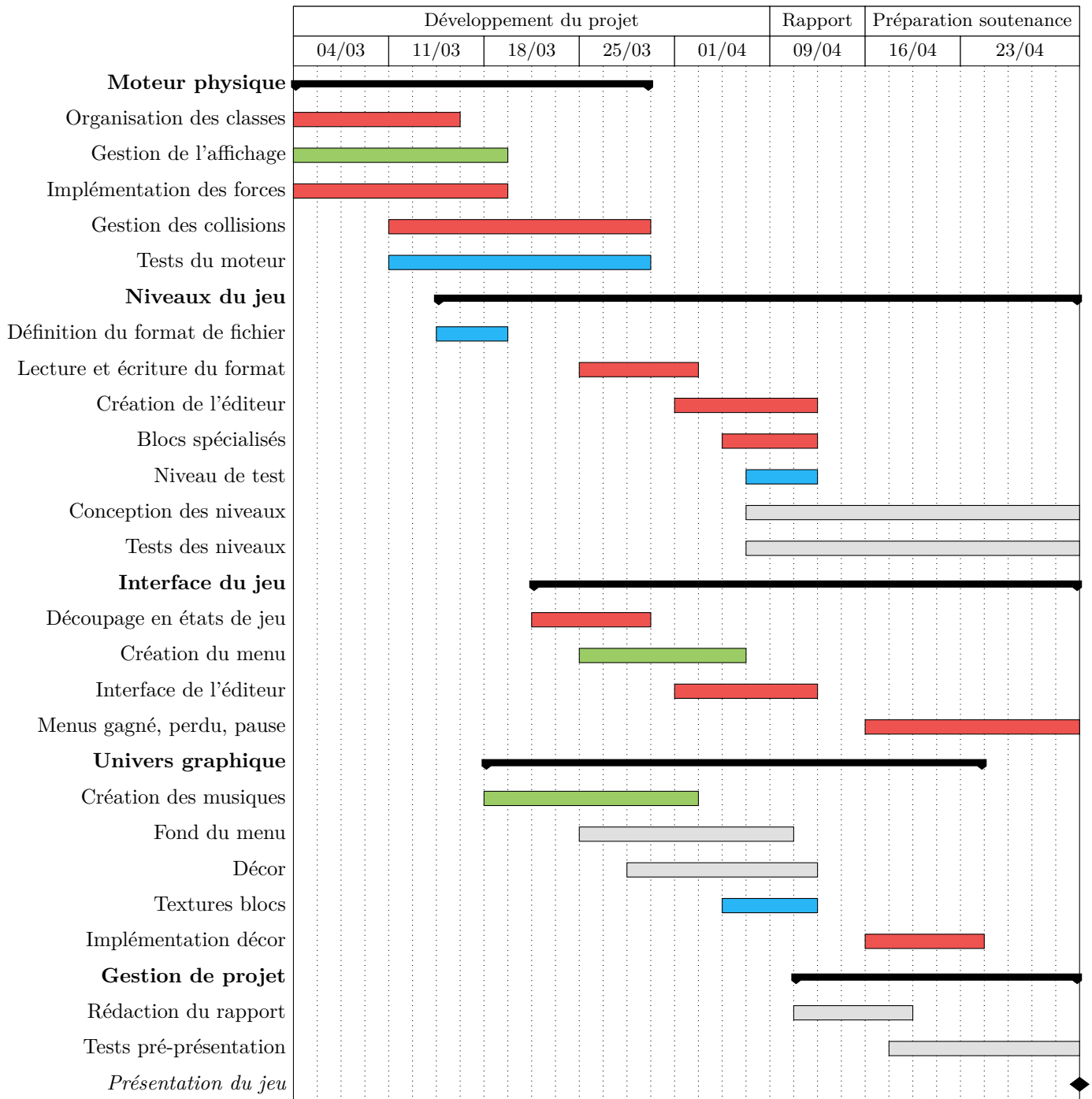


FIGURE 2.1 – Diagramme de la répartition des tâches. En vert, les tâches affectées à Maëlle ; en bleu, les tâches affectées à Rémi ; en rouge, les tâches affectées à Mattéo ; en gris, les tâches résolues en groupe

2.2 Outils de développement

Tout le code du programme est centralisé sur un dépôt GitHub et les membres du groupe utilisent Git pour synchroniser leur travail, travailler indépendamment, vérifier le travail des autres ou récupérer d'anciennes versions. [1]

Pour le développement du jeu, nous avons choisi le C++ car il s'agit du langage que nous apprenons cette année : d'une part, nous possédons donc des acquis avec ce langage, et d'autre part ce projet nous permet de fixer nos connaissances du C++. Une autre raison est la quantité importante de documentation sur le web, de nombreuses bibliothèques et d'une communauté importante.

Nous avons fait le choix de la SFML comme librairie graphique et d'utiliser la spécification C++11 qui apporte des fonctionnalités non-négligeables (notamment concernant la gestion de la mémoire avec les pointeurs intelligents). [2]

Pour écrire le code, nous avons utilisé différents éditeurs de texte (atom et gedit). Nous compilons notre programme avec le compilateur g++. Pour faciliter la compilation, nous avons utilisé CMake.

Chapitre 3

Analyse du projet

3.1 Découpage du projet

Le jeu est constitué d'une suite de niveaux organisés de manière semblable à ceux d'un jeu de plateformes. Ces niveaux contiennent des entités.

Les entités du jeu sont multiples : blocs, blocs spéciaux, joueurs, éléments de décor. Ces entités – ou objets, interagissent entre elles par un certain nombre de phénomènes physiques « naturels ». Pour répondre à ce besoin, **un moteur physique** est nécessaire. Le moteur physique est chargé de gérer les forces s'appliquant aux objets du jeu, de répondre aux collisions entre objets et de faire évoluer les objets en conséquence des forces qui leur sont appliquées.

Plusieurs moteurs physiques en 2D existent déjà dans le langage que nous avons choisi, notamment Box2D. [3] Nous avons choisi d'implémenter le moteur physique du jeu par nous-mêmes pour répondre aux besoins particuliers (notamment la force d'attraction) et car cela nous permet de mettre en pratique les savoirs acquis au premier semestre dans le module de physique générale.

Les **niveaux du jeu** sont constitués de ces entités et d'autres métadonnées. Pour pouvoir éditer les niveaux, les sauvegarder et y rejouer plus tard, il est nécessaire de pouvoir les stocker en dehors de la mémoire. Nous avons pour ce faire choisi de définir un format de fichier binaire permettant leur stockage sur le disque. Des fonctions pour coder et décoder ce format devront être écrites.

Skizzle propose différents **états de jeu**, notamment, on peut à tout moment se trouver dans l'éditeur, dans le jeu en lui-même ou sur la vue des règles. Pour pouvoir accéder à ces états, nous devons créer un menu. L'ensemble des états du jeu doit être abstrait pour pouvoir être géré dans la classe principale. Certains états du jeu proposeront des éléments interactifs (boutons, barres d'outils, zones de texte) qui doivent être implémentés.

Enfin, les différents **objets du jeu** sont représentés à l'écran en dessinant des textures. Nous avons également choisi d'ajouter des musiques au jeu pour le rendre plus convivial. D'autres éléments graphiques doivent être créés, par exemple le fond du menu. Tous ces éléments sont regroupés dans l'univers graphique du jeu.

3.2 Découpage du code

Nous avons choisi d'organiser notre code selon le paradigme objet. La plupart du code est sorti en dehors du `main`, dont la seule fonction est d'instancier la classe `Manager` qui gère de manière abstraite le jeu et de démarrer le premier état du jeu : le menu.

3.2.1 États, gestion des états et des ressources

Un état du jeu modélise un écran pouvant être affiché. Une classe abstraite `State` chapeaute toutes les classes d'états et permet de requérir l'implémentation d'une interface commune :

- `enable()` : cette méthode initialise l'état avant qu'il commence à être affiché. L'état implémentant cette méthode doit mettre en place les ressources globales utilisées comme la lecture de la musique, le titre de la fenêtre, les éléments de l'interface quand cette méthode est appelée ;
- `processEvent(event)` : cette méthode est appelée avec un événement lorsque celui-ci est extrait par la SFML lors de la boucle principale. L'état est censé décider s'il souhaite traiter cet événement et, si oui, modifier ses variables en conséquence ;
- `frame()` : cette méthode est appelée lorsque l'état doit dessiner une frame à l'écran. Pour éviter d'encombrer la boucle principale, l'état doit dessiner sa frame le plus rapidement possible.

Les états suivants sont implémentés et descendent de la classe `State` : `Rules` pour afficher les règles du jeu, `Menu` pour afficher le menu du jeu, `Level` pour afficher les niveaux (soit l'éditeur, soit le jeu en lui-même).

On définit `Manager` la classe qui gère les éléments principaux du jeu. Notamment, `Manager` maintient une pile d'états qui est initialisée contenant une seule instance de la classe `Menu` et peut être empilée ou dépilée par les états. Par exemple, le menu peut empiler un nouvel état instance de `Rules` pour « démarrer » la vue affichant les règles. En tout temps, l'état en haut de la pile est celui qui est actif (il reçoit les événements et est dessiné).

La librairie SFML permet de charger les ressources comme la musique, les images et les polices. Cependant, recharger ces ressources à chaque utilisation serait inefficace. La classe `ResourceManager` permet de mutualiser ces ressources : les états lui demandent les ressources à obtenir et le gestionnaire de ressources s'arrange pour ne charger la ressource qu'à la première demande et à la garder en mémoire par la suite. Le gestionnaire des ressources mutualise l'accès aux polices, textures et à la lecture de la musique.

La figure 3.1 résume les classes de gestion d'états et de ressources présentées.

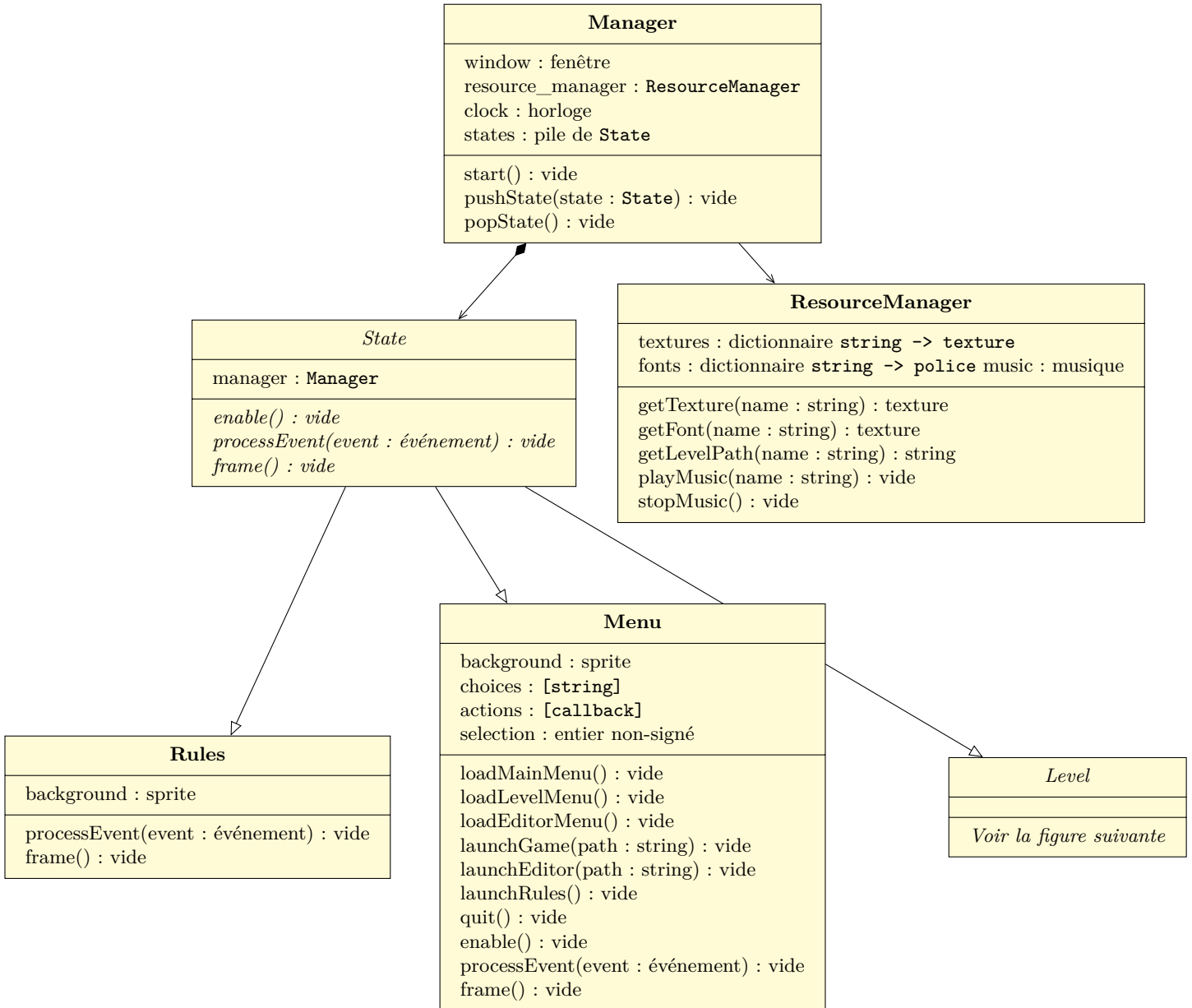


FIGURE 3.1 – Gestion des états et des ressources dans le jeu

3.2.2 Niveau et objets

La classe `Level` définit les niveaux, qui sont des collections d'objets. Elle définit la méthode pour dessiner tous les objets d'un niveau, le charger, le sauvegarder dans un fichier, ajouter ou supprimer des objets. Elle ne définit pas la méthode `frame()` que tous les états doivent implémenter, elle n'est donc pas un état en tant que tel.

Deux classes dérivent de `Level` : `Game` pour jouer aux niveaux et `Editor` pour les éditer. L'abstraction en `Level` permet d'éviter la duplication de code notamment en ce qui concerne la gestion des objets contenus.

Les classes de niveaux manipulent des collections d'objets. Les objets modélisent toutes les entités du jeu : les joueurs, les blocs et les blocs spéciaux. Une classe abstrait les fonctionnalités de tous les objets, `Object`.

Les classes `Block`, définissant l'apparence et le comportement des blocs, et `Player`, définissant l'apparence et le comportement des joueurs, descendent directement d'`Object`. Enfin, on définit des blocs spéciaux, qui peuvent réaliser des actions particulières :

- le bloc de gravité modifie la direction de la gravité dans un niveau lorsqu'une entité entre en contact avec lui. Il ne peut être activé qu'une seule fois par partie ;
- le bloc changeur échange la polarité de l'entité entrant en contact avec lui. Il ne peut être activé qu'une seule fois par partie ;
- le bloc tueur tue le joueur entrant en contact avec lui et fait perdre la partie (le niveau passe en mode « perdu ») ;
- le bloc d'arrivée tue le joueur entrant en contact et lorsqu'il ne reste plus de joueurs fait gagner la partie (le niveau passe en mode « gagné »).

La figure 3.2 résume les classes de niveaux et d'objets.

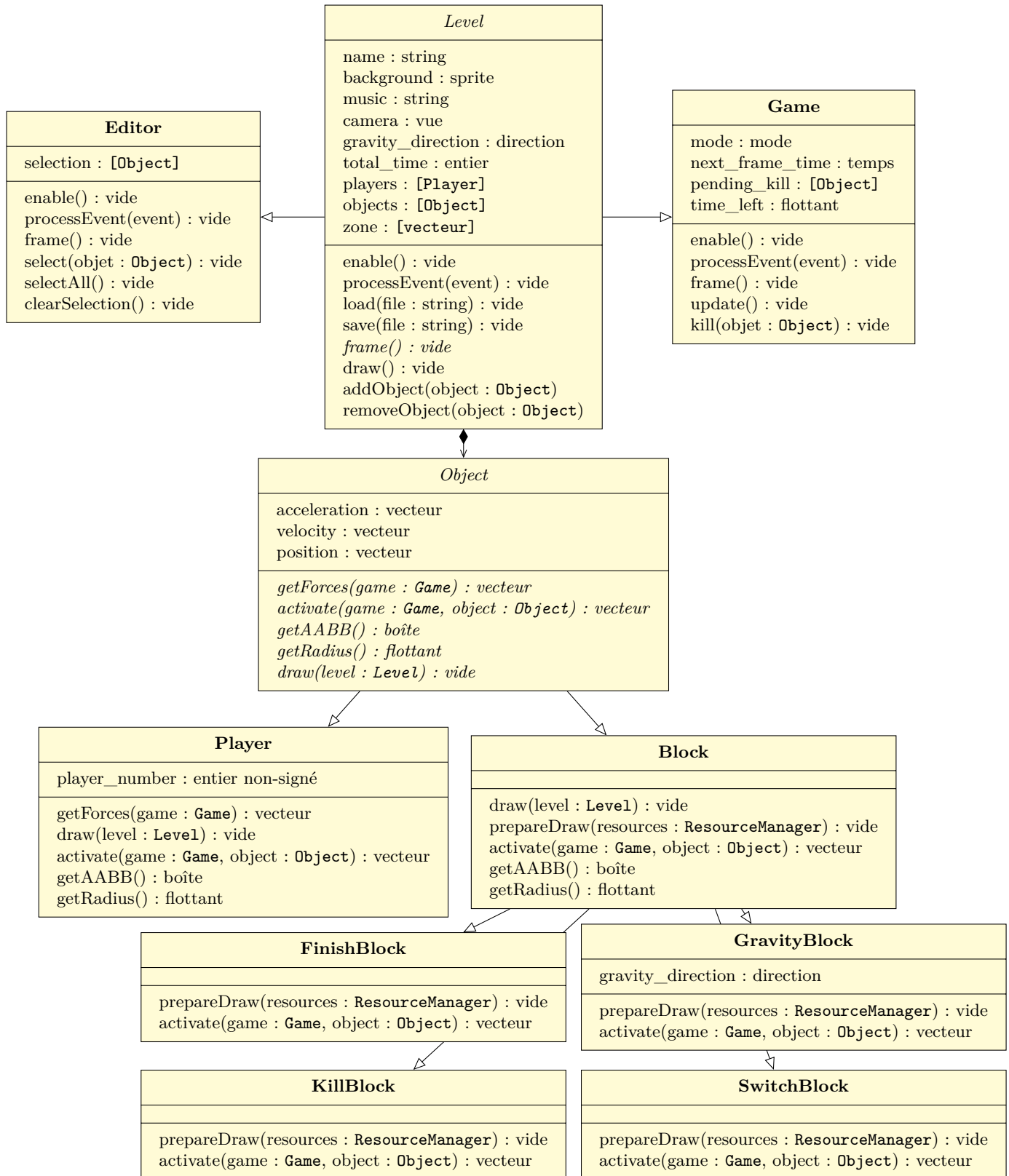


FIGURE 3.2 – Classes du niveau

Chapitre 4

Développement

4.1 Moteur physique

Cette partie du projet est basée en partie sur les algorithmes du pré-rapport. L’affichage est géré par des fonctions `draw()` permettant de boucler sur les différents objets afin de mettre à jour le dessin affiché en fonction de leur position. Il fallut également gérer la caméra avec l’objet `view` de la SFML, et l’adapter au redimensionnement de la fenêtre. Enfin, pour un meilleur rendu graphique, nous avons chargé et appliqué des textures aux objets (textures provisoires lors de cette étape).

Lors de l’implémentation des forces et des collisions, nous avons rencontré quelques difficultés, notamment au niveau des collisions. En effet, grâce aux tests du moteur dans différentes situations, nous avons pu repérer des erreurs de collisions. Cela mena à la création d’une nouvelle classe `Object` réunissant les balles et les blocs dans un même tableau et définissant les propriétés communes des objets. L’implémentation des collisions est basée sur un solveur de collisions discret. Nous avons finalement passé plus de temps que prévu sur cette partie du projet. [4, 5, 6]

4.2 Niveaux du jeu

Tout d’abord, il fallut définir le format des fichiers de niveaux. Pour cela, nous avons au préalable fait des recherches sur les normes de base, puis nous avons listé tout ce qu’il fallait sauvegarder dans le niveau. Il fallut ensuite lire et écrire le format. Afin de pouvoir créer des niveaux plus facilement, mais également permettre aux utilisateurs de jouer à leurs propres niveaux, nous avons choisi de créer un éditeur de niveaux en mode graphique. Cela nous prit plus de temps que prévu, c’est pourquoi il nous restait peu de temps pour la création de niveaux. Nous avons donc réfléchi à des niveaux courts mais demandant de la réflexion afin de les rendre intéressants malgré le manque de temps.

4.3 Interface du jeu

Afin de mieux gérer l’affichage entre les menus, l’éditeur et les niveaux, nous avons décidé de découper le jeu en états. Nous avons créé une classe abstraite gérant tous ces états, ainsi qu’une classe par état : Menu pour le menu principal, Rules pour les règles du jeu, Editor pour l’éditeur de niveaux et Game pour le jeu, ainsi qu’une classe abstraite Level pour gérer les niveaux.

Le menu principal se base ainsi sur ce découpage. Chaque option renvoie à un état différent ou modifie l’état du menu si cela envoie sur un autre menu (choix de niveaux).

L’éditeur est la partie la plus interactive du jeu. En sus des nombreux raccourcis clavier et souris décrits dans le manuel d’utilisation, une boîte à outils est présente sur le côté et permet de sélectionner le type d’objet à placer dans le niveau.

Cependant, il reste certaines lacunes au niveau de l’interface, notamment au niveau des composants interfaçant avec l’utilisateur. Par exemple, faute de temps, nous n’avons pas pu implémenter de moyen de modifier le nom du niveau modifié depuis l’éditeur, de modifier la texture de fond ou la musique jouée. Ces modifications doivent être faites « à la main » dans le fichier du niveau.

Par ailleurs, lorsque l’on gagne, perd ou met en pause le niveau, cet état est bien sauvegardé mais aucun élément d’interface n’a pu être implémenté à temps pour permettre à l’utilisateur par exemple de passer au niveau suivant ou de recommencer. Pour le moment, l’état est affiché en texte dans la console pour s’assurer que le changement d’état est bien fonctionnel.

Nous espérons pouvoir ajouter ces fonctionnalités en nous basant sur la librairie SFGUI d’ici à la soutenance, car elles nous paraissent importantes. [7]

4.4 Univers graphique

Nous avons décidé d’ajouter des musiques au jeu afin de le rendre plus vivant. Pour cela, nous avons fait appel à un étudiant de l’IUT de Montpellier, Maxime PETITJEAN, pour aider Maëlle à créer des musiques pour chaque niveau ainsi que pour le menu à l’aide du logiciel *Ableton*.

Nous avons également décidé de créer nos propres textures pour un rendu du jeu plus personnel. Après les dessins préalablement réalisés par Maëlle sur papier, Rémi s’est chargé de les numériser en repassant les contours pour créer une image vectorielle sur Inkscape. Puis Maëlle les a colorées avec le même logiciel, pendant que Rémi se chargeait des textures des blocs que nous avons au préalable définies à l’oral en groupe.

Nous n’avons pas eu le temps d’implémenter le décor avant le rendu du code, mais il est prévu que cela soit fait avant le passage à l’oral.

Chapitre 5

Manuel d'utilisation

5.1 Menu du jeu

5.1.1 Menu principal

Au lancement du jeu le menu principal apparaît. La figure 5.1 montre une copie d'écran du menu. Le menu est composé de quatre boutons.

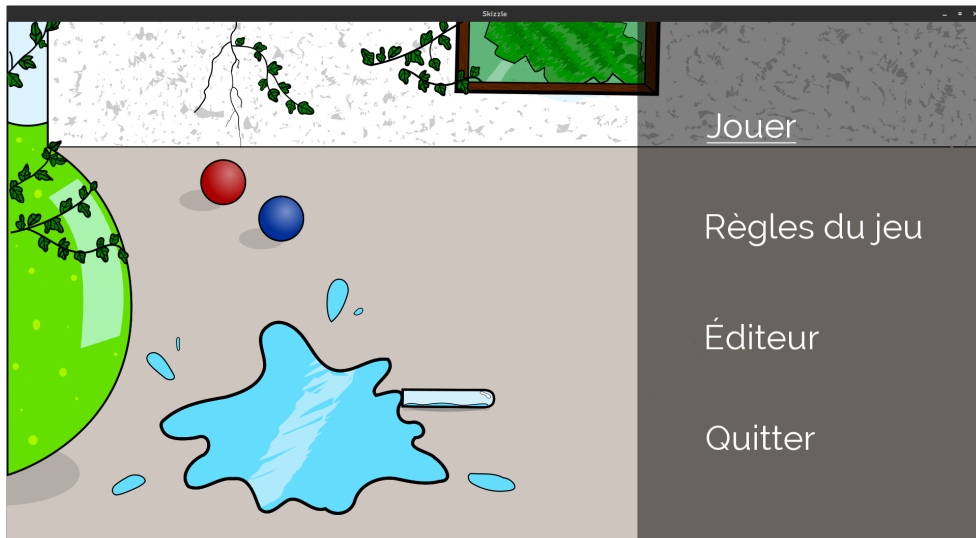


FIGURE 5.1 – Menu principal

Bouton jouer Il permet d'accéder au menu de sélection des niveaux.

Bouton règles du jeu Il démarre la vue affichant les règles du jeu.

Bouton éditeur Il permet d'accéder au menu permettant de créer un niveau ou d'en éditer un existant.

Bouton quitter Il quitte le jeu.

5.1.2 Menu de sélection du niveau

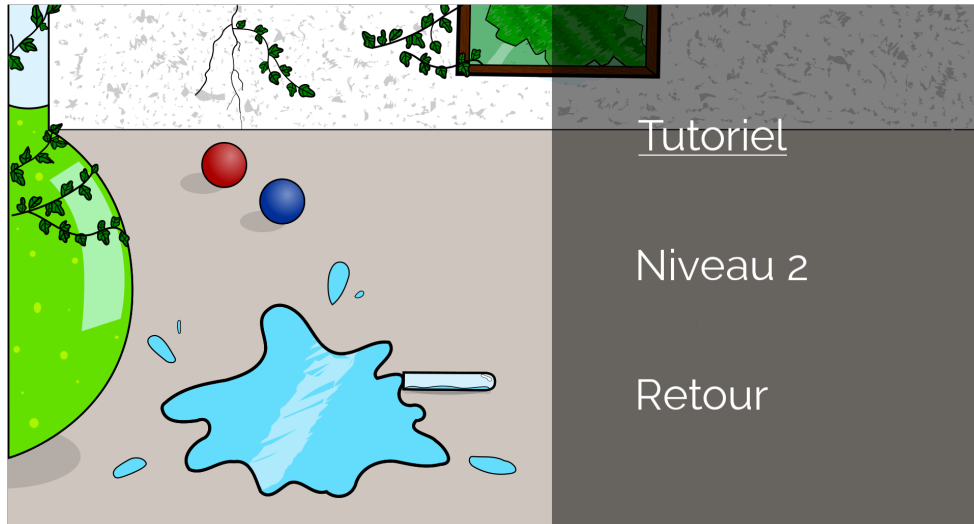


FIGURE 5.2 – Menu de sélection du niveau

Le menu de sélection des niveaux, dont une copie d'écran est présentée en figure 5.2, se compose de deux types de boutons. Les boutons niveaux permettent au joueur de choisir un niveau à jouer et le bouton retour pour revenir vers le menu principal.

5.1.3 Menu de l'éditeur

Le menu de l'éditeur, dont une copie d'écran est présentée en figure 5.3, se compose de trois types de boutons. Créer un nouveau niveau pour démarrer la création d'un niveau à partir de rien, éditer un niveau déjà existant, et retour qui renvoie vers le menu principal.

Dans les menus, les touches , ou le passage de la souris sur un des boutons permettent de sélectionner un élément du menu. Le bouton actuellement sélectionné est souligné.

La touche ou un clic avec la souris permettent de valider le choix.

Les touches et permettent de revenir au menu précédent.

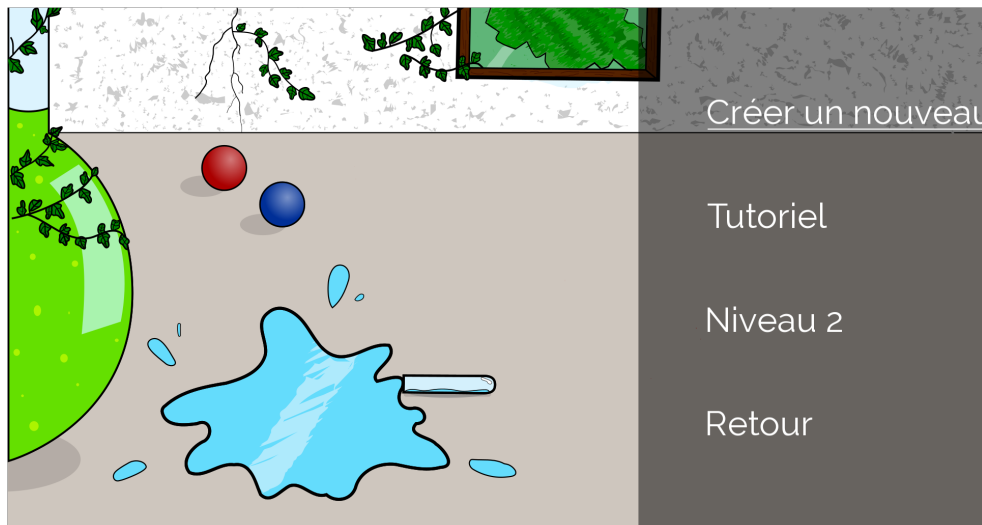


FIGURE 5.3 – Menu de l'éditeur

5.2 Objets

Les niveaux se composent d'une collection d'objets qui interagissent entre eux par les lois physiques ou par l'action du joueur, selon les types d'entités. Nous définissons dans cette section les comportements des différents objets.

5.2.1 Balles

1

Les balles sont des objets contrôlés par les joueurs. Chaque joueur est affecté à une balle, identifiée de manière unique par son numéro.

Polarité



Les objets de couleur bleue repoussent tous ceux de la même couleur et attirent les objets de couleur rouge.



Les objets de couleur rouge repoussent tous ceux de la même couleur et attirent les objets de couleur bleue.

Blocs



Le bloc de base ne réalise aucune action particulière, il sert uniquement à délimiter des zones dans le niveau.



La caisse est un bloc qui peut être déplacé par les joueurs ou une autre caisse. Elle active les blocs de gravité comme les joueurs.



Le bloc de gravité modifie le sens de la gravité du niveau lorsqu'un joueur ou une caisse l'active en rentrant en collision avec lui. Ce bloc n'est activable qu'une seule fois par partie.



Le bloc inverseur inverse la charge des balles qui l'activent en rentrant en collision avec lui. Ce bloc n'est activable qu'une seule fois par partie.









Le bloc tueur fait perdre la partie si un des deux joueurs l'active en rentrant en collision avec lui.



Le bloc d'arrivée fait gagner la partie lorsque tous les joueurs l'ont activé en rentrant en collision avec lui.

5.3 Jeu

Les joueurs ne peuvent déplacer leurs balles que vers la gauche ou la droite. Le joueur 1 utilise  pour se déplacer vers la droite et  pour se déplacer vers la gauche. Le joueur 2 utilise  pour se déplacer vers la droite et  pour se déplacer vers la gauche.

Les joueurs peuvent mettre le jeu en pause avec la touche , ou bien quitter le niveau en appuyant sur .

La partie peut être perdue pour trois raisons : si un joueur a activé un bloc tueur, si un joueur est sorti de la zone de jeu ou si le temps s'est écoulé totalement.

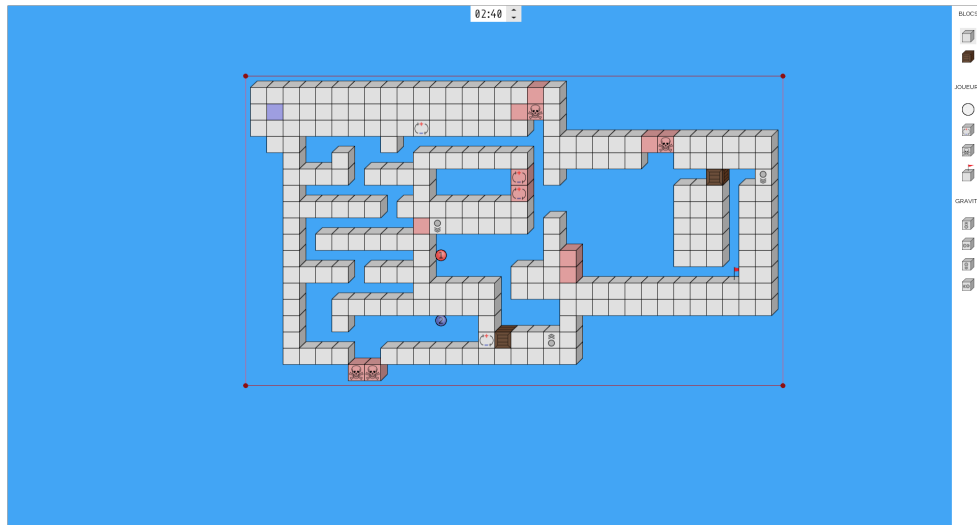


FIGURE 5.4 – Exemple de niveau en cours d'édition dans l'éditeur

5.4 Éditeur

5.4.1 Gestion des objets

La figure 5.4 montre un exemple de niveau en édition dans l'éditeur. On peut sélectionner le type d'objet à placer en cliquant à la souris dans la barre latérale droite.

Un clic gauche sur une zone libre permet de placer le type d'objet actuellement sélectionné. Le maintien du clic gauche en déplaçant la souris permet de placer plusieurs objets à la fois.

Pour modifier la charge d'un objet, le curseur doit être placé sur celui-ci. La touche `Ctrl` doit être enfoncée tout en faisant glisser la mollette de la souris, ou bien en faisant glisser deux doigts sur le pavé tactile vers le haut ou le bas. L'objet change de couleur en conséquence.

La sélection d'un objet se fait en cliquant sur celui-ci. Pour sélectionner plusieurs objets, on maintient `Ctrl` et on clique sur les objets. On peut effectuer une sélection rectangulaire en maintenant la touche `Shift` et en faisant glisser la souris sur la zone à sélectionner, en maintenant le clic gauche enfoncé. Lorsqu'un objet est sélectionné, sa bordure devient rouge, comme montré dans la figure 5.5.



FIGURE 5.5 – Exemple d'objet sélectionné

La suppression d'un objet s'effectue en cliquant dessus avec le bouton droit de la souris. Pour supprimer tous les objets actuellement sélectionnés, il suffit d'appuyer sur la touche `Suppr`.

5.4.2 Compte à rebours

Le compte à rebours se situe en haut au centre de la fenêtre. Durant la partie, s'il arrive à 0, les joueurs meurent et la partie est perdue.

La valeur du compte à rebours peut être modifiée dans l'éditeur en cliquant sur les flèches se situant à côté, ou en glissant la molette en gardant la souris sur le compteur.

5.4.3 Zone de jeu

La zone jouable est représentée dans l'éditeur par un polygone rouge composé de quatre points. Chacun de ces points peut être déplacé en cliquant dessus afin de modifier la taille et la forme de la zone.

5.4.4 Gestion de la caméra

Le déplacement de la caméra s'effectue en déplaçant la souris vers une bordure de la fenêtre. La molette de la souris peut être utilisée pour un défilement vertical, ou horizontal si la touche **Shift** est enfoncée.

5.4.5 Commandes générales

Il est possible de tester le niveau à tout moment en appuyant sur la touche **Espace**. Pour revenir à l'édition il suffit d'appuyer à nouveau sur **Espace**.

Afin de sauvegarder le niveau il est nécessaire de réaliser la combinaison de touches suivantes : **Ctrl** + **S**. (en mode édition). Si vous ne sauvegardez pas vos changements, ils seront perdus.

Pour quitter l'éditeur et revenir au menu la touche **Échap** doit être utilisée.

Chapitre 6

Conclusion

6.1 Perspectives

Par manque de temps, de nombreuses perspectives d'amélioration n'ont pas pu aboutir. Nous aurions pu améliorer l'interface graphique (ce qui est prévu avant l'oral), faire plus de niveaux et donc ajouter des musiques pour ces nouveaux niveaux, ou encore ajouter les menus pause, victoire et défaite.

6.2 Conclusions

6.2.1 Fonctionnement de l'application

Nous avons remarqué qu'un message d'erreur « l'application ne répond pas » s'affiche dans certaines circonstances que nous n'avons pas réussi à isoler, bien que le programme réponde toujours.

Lors de la rotation de la caméra à la suite d'un changement dans la direction de la gravité, la caméra ne réalise pas toujours le mouvement le plus rapide et peut effectuer une rotation à 270 degrés au lieu de 90 degrés dans certaines circonstances.

Dans certains cas les blocs déplaçables, si placés en trop grandes quantités, se traversent. De plus, l'algorithme de collisions étant discret, un objet ayant une vitesse trop élevée peut passer à travers un autre.

Enfin, la force d'attraction ne réagit pas tout à fait comme nous l'avions imaginée initialement : elle est moins souple. Cela est probablement lié à un mauvais ajustement dans les constantes du moteur physique. Cette force fonctionne malgré tout.

Cependant, les menus ainsi que les retours aux menus fonctionnent bien, l'affichage est fluide, l'éditeur est fonctionnel et comporte de nombreuses fonctionnalités intuitives. Les ressources ne posent pas de problème, les textures se chargent et les musiques se déclenchent correctement et bouclent comme prévu.

Globalement, le jeu est fonctionnel et ne pose problème que dans des cas très particuliers, il reste tout de même stable malgré le temps limité que nous avons pour régler les problèmes.

6.2.2 Fonctionnement du groupe de travail

Le principal problème du groupe fut le sens des priorités. Nous nous sommes attardés sur des détails avant que le tout soit fonctionnel, ce qui a perturbé notre gestion du temps. Malgré cela, nous avons réussi à nous organiser pour gagner en efficacité en partageant les tâches les plus chronophages.

Le fait de refaire fréquemment le diagramme de Gantt nous a permis de rattraper en partie le retard que nous avons pris. Au niveau de la communication, nous n'avons eu aucun problème car nous nous retrouvions très souvent sur Skype ainsi qu'à l'université afin de discuter de tout problème et optimiser nos chances de trouver rapidement une solution.

Malgré une gestion du temps perturbée, notre organisation et notre communication nous ont permis de compenser les défauts du groupe, et ainsi d'avancer à un rythme convenable.

Ainsi, grâce à notre détermination et notre coopération, nous avons su franchir les difficultés et ainsi proposer un jeu fonctionnel et amusant.

Webographie

- [1] Membres du groupe. Dépôt GitHub du projet. <https://goo.gl/9Aa0Zu>.
- [2] University of Michigan. Using c++11's smart pointers. <http://goo.gl/nwsKu2>.
- [3] Erin Catto. Box2d : A 2d physics engine for games. <http://goo.gl/uTnXH4>.
- [4] Randy Gaul. Impulseengine.
- [5] Allen Chou. Game physics : Stability – slops. <http://goo.gl/j0aEGA>.
- [6] Entropy Interactive. The game loop. <http://goo.gl/cin0jt>.
- [7] Stefan Schindler. Sfgui. <http://goo.gl/2fwBku>.