RAPPORT DE PROJET
THE LIGHT CORRIDOR

Léo SALAUN - Heïdi LAGARDE

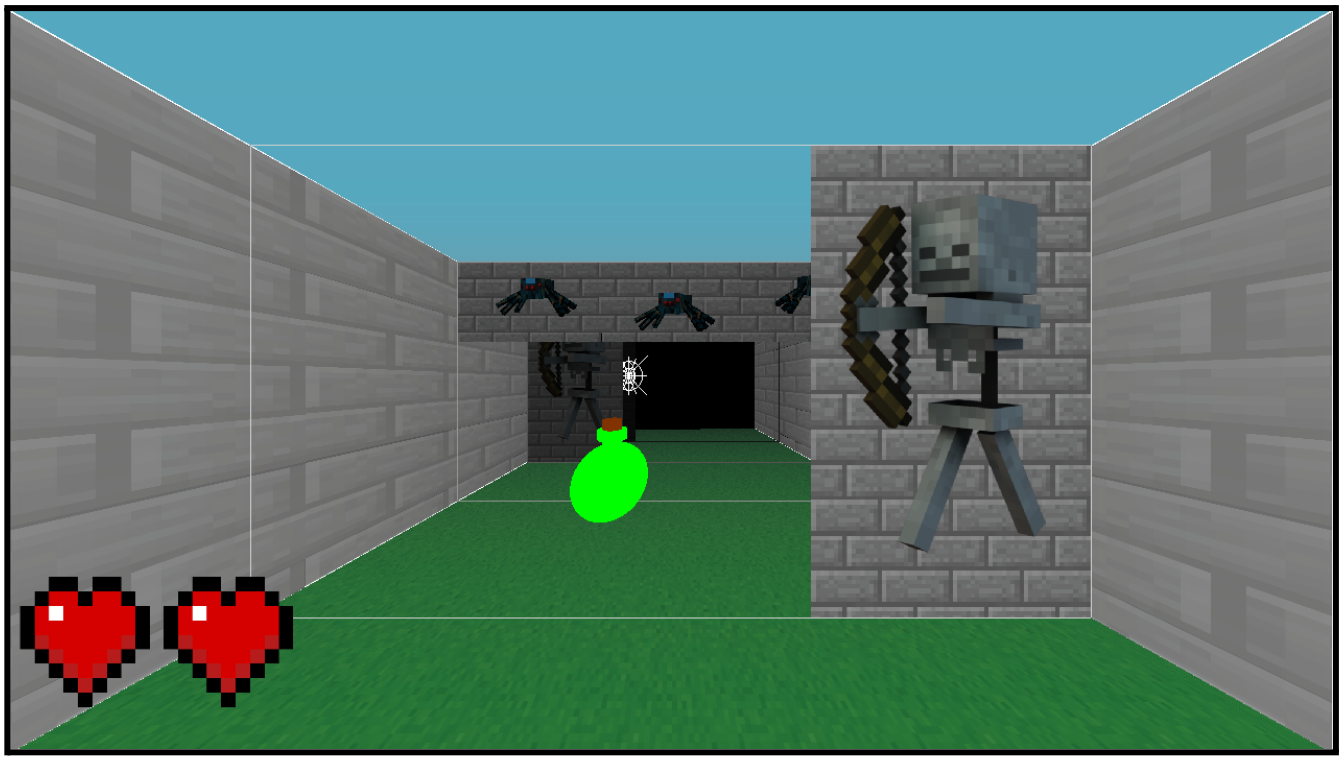


TABLE DES MATIÈRES

MODE D'INSTALLATION & FONCTIONNEMENT

Pour compiler et lancer le jeu, il suffit d'ouvrir un terminal muni des commandes bash, se placer à la racine du projet et exécuter les commandes suivantes :

Compilation du projet :

`make` ou `make app`

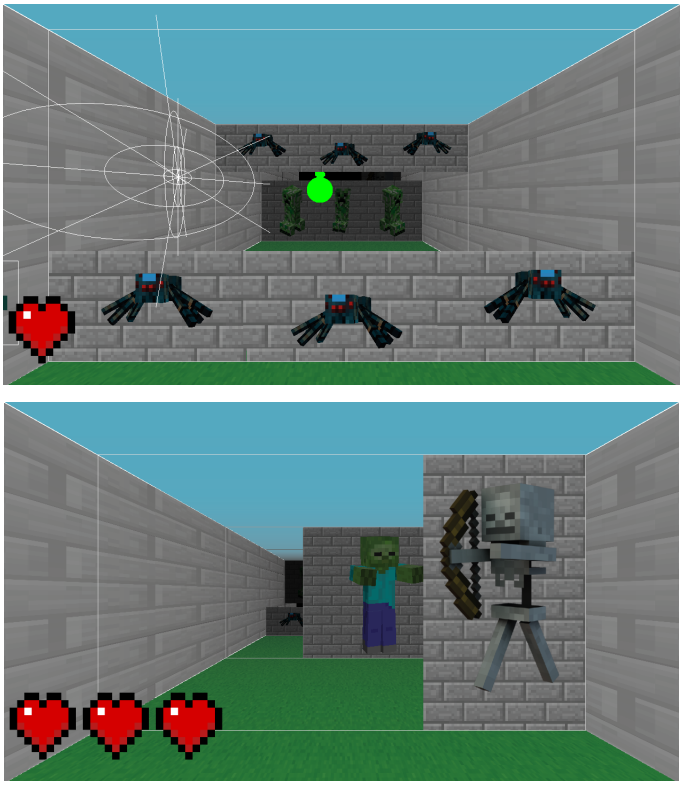
Exécution :

`bin/The_Light_Corridor.out`

RÉSULTATS OBTENUS

But du jeu :

Le jeu présenté est dérivé du jeu déjà existant Minecraft. Le but du jeu est d'arriver à la fin du parcours en gardant une balle en mouvement et en la faisant rebondir contre les murs à l'aide d'une raquette. Si l'on ne rattrape pas la balle après l'avoir lancé (si elle passe derrière la caméra), on perd une vie. Si on n'a plus de vie, on doit recommencer le jeu. Il faudra aussi éviter les obstacles sur le parcours. On peut aussi avoir des bonus qui permettent de récupérer une vie ou de recoller la balle à la raquette.

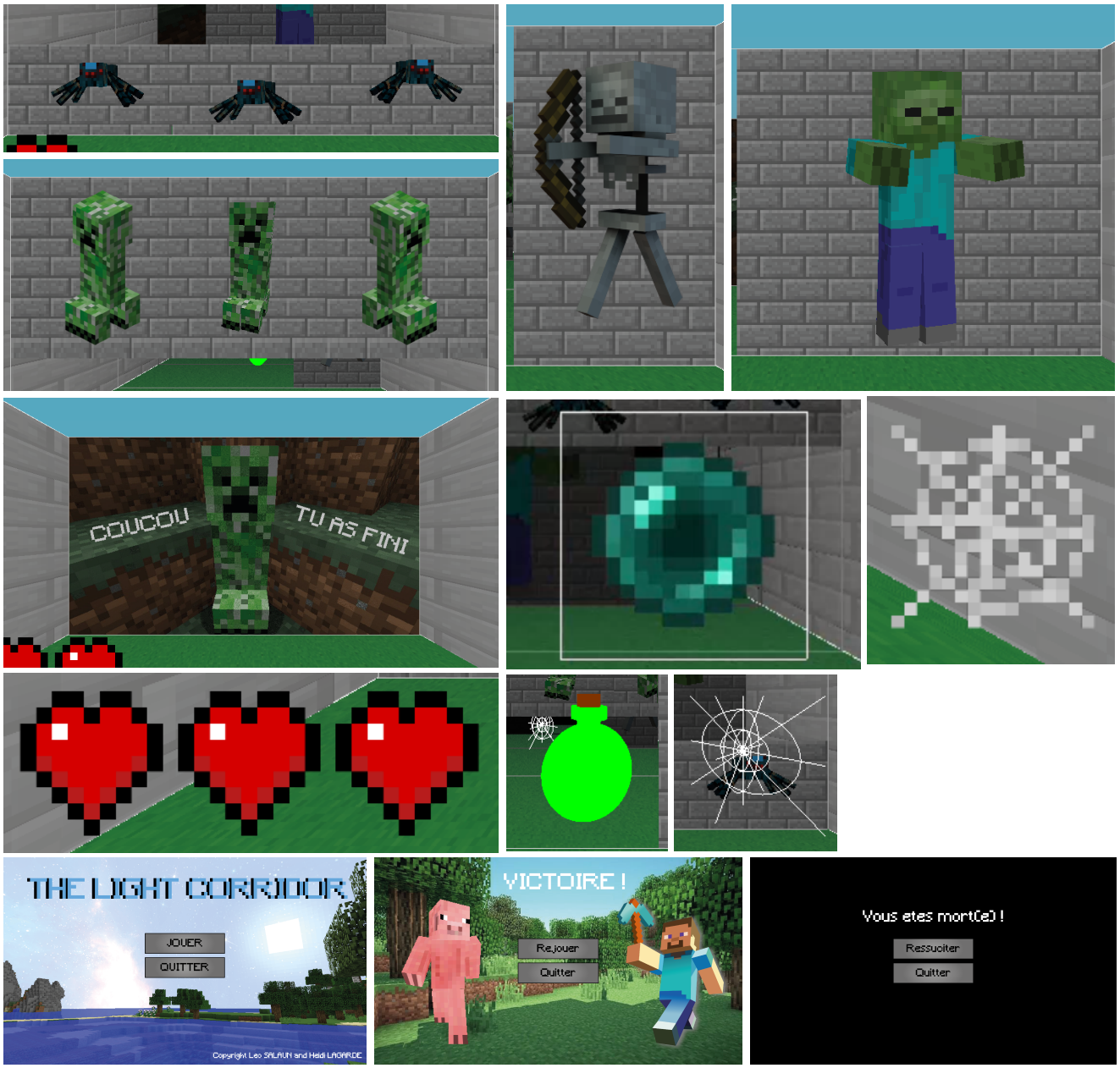


Exemples d'images du jeu

Design du jeu :

Le jeu s'inspire fortement de l'univers de Minecraft. Les design du jeu (les textures) ont d'abord été réalisé sur Adobe illustrator par Heïdi. Elle a réalisé le sol, le plafond, les murs, les menus et les obstacles (zombie, araignée, creeper,...).
Ils ont ensuite été importé un à un dans le code grâce à la fonction `stbi_load` qui a permis de charger les images à partir des fichiers présents dans le dossier "doc".
On compte au total 14 textures différentes :

- 3 pour les murs constituant le couloir (côtés, sol et plafond).
- 4 pour les murs faisant office d'obstacles (selon leur taille et inclinaison) et 1 pour le mur de fin.
- 1 pour la balle.
- 2 pour l'interface (vies et bonus de la colle).
- 3 pour les menus et écrans de mort et de victoire.



Exemples des textures et modèles 3D des bonus appliqués dans le jeu

MÉTHODE DE TRAVAIL

Arborescence du git :

Dans le dossier “doc”, nous pouvons retrouver tous les design du jeu (les obstacles avec les créatures, les menus, les bonus, la balle, les murs, le sol, le plafond,...).

Dans le dossier “obj” sont stockés les fichiers bonus.o, collisions.o, corridor.o, interface.o, obstacles.o, raquette.o, ... lors de la compilation du projet.
ainsi que les fichiers 3D_tools.o et drawscene.o qui nous permettent de modéliser les différents éléments du jeu.

Le dossier “src” a été organisé de sorte à ce que chaque sous-dossier contient le code associé à une partie spécifique de l’application. Nous retrouvons tous les fichiers qui permettent de faire marcher le jeu, chaque nom de fichier définissant à quoi il sert :

- “collisions” pour gérer les collisions et centraliser les autres parties.
- “corridor” pour modéliser le couloir.
- “raquette” pour gérer la raquette et la balle.
- “obstacles” pour définir et dessiner les obstacles.
- “bonus” pour concevoir et dessiner les bonus.
- “interface” pour représenter visuellement les informations relatives au joueur comme le nombre de vies.

Répartition du travail :

Ce projet a été réalisé en binôme par Léo Salaun et Heïdi Lagarde. Léo a joué un rôle important dans la réalisation du projet en raison de ses compétences plus approfondies en programmation. Il a su expliquer et aider Heïdi tout au long du projet. Elle a ainsi pu modéliser les murs, les obstacles, réaliser les textures et leur importation, définir à quel moment un menu devrait apparaître (menu d’échec, de victoire) et réaliser les bonus.

Le travail a majoritairement été fait sur le PC de Léo car l’ordinateur d’Heïdi n'était plus assez performant durant la réalisation de ce projet pour afficher le jeu. Nous nous donnions des rendez-vous aussi souvent que possible chez l’un ou l’autre après la fin des TD de synthèse d’image pour travailler ensemble sur le projet. Lors de l’importation des textures et afin que l’implication de Heïdi puisse être un minimum visible sur le répertoire git, Heïdi codait sur son ordinateur et réalisait des commits de ses changements pendant que Léo visualisait le travail sur le sien.

DÉTAILS TECHNIQUES

Structures définies :

Plusieurs structures ont été créées dans le but de faciliter la gestion des différentes composantes du jeu. Celles-ci incluent notamment :

- `Player`, qui représente le joueur ou plutôt son interface. Elle est constituée de `nbVies` (entier représentant nombre de vies du joueur), `forward` (booléen indiquant si le joueur peut avancer ou non), `sticky` (booléen indiquant si le bonus correspondant à la colle est actif) et `menu` (entier indiquant quel menu est actif ou 0 si une partie est en cours).
- `Ball`, qui représente la balle. Elle est constituée de `posX`, `posY` et `posZ` (réels décrivant la position de la balle dans le repère), `speeX`, `speeY` et `speeZ` (réels décrivant la vitesse de la balle dans le repère) et `sticky` (booléen indiquant si la balle est attachée à la raquette ou non).
- `Obstacle`, qui représente un obstacle. Elle est constituée de `pos` (réel décrivant la position en Z de l'obstacle), `size` (réels décrivant la taille de l'obstacle par rapport à celle du couloir) et `wall` (caractère indiquant si l'obstacle est sur la droite du couloir, sur la gauche, en haut ou en bas).
- `Bonus`, qui représente un bonus. Elle est constituée de `type` (entier décrivant si le bonus est du type 'gain de vie' ou 'colle'), `posX`, `posY` et `posZ` (réels décrivant la position du bonus dans le repère) et `visible` (booléen indiquant si le bonus est visible ou non).

Constantes utilisées :

Ces structures sont accompagnées de plusieurs constantes décrivant le cadre du programme :

- `NB_OBSTACLES` décrit le nombre d'obstacles à gérer.
- `OBSTACLE_SPACE` décrit l'espacement entre deux obstacles.
- `OBSTACLE_LENGTH` décrit la longueur affichée du couloir.
- `NB_BONUS` décrit le nombre de bonus à gérer.
- `ROTATE_SPEED` décrit la vitesse de rotation des bonus.

Ces structures et constantes nous ont permis de manipuler efficacement les différentes composantes de notre jeu. Les facettes les plus complexes à gérer de notre projet étaient certainement la gestion des obstacles et des collisions.

Gestion des obstacles :

Nous avons fait le choix de représenter nos obstacles sous la forme d'un tableau à taille fixe sachant que la longueur du couloir ne changerait pas. Nous avons également décidé qu'il serait plus simple de ne déplacer que les obstacles et les bonus en direction de la caméra afin de créer l'illusion d'une avancée pour le joueur. Ainsi seules les parties du couloir correspondant aux quatre obstacles suivants sont affichées et la couleur des obstacles varie selon la position de chacun par rapport à la caméra et à la balle. En effet, souhaitant commencer le projet le plus tôt possible afin de ne pas se retrouver débordés par la suite, nous avons pris la décision d'avancer malgré nos connaissances limitées en illuminations. Cela nous a poussé à exploiter des variations dans la couleur des obstacles ainsi que des dégradés dans les textures du couloir afin de simuler la présence d'une source de lumière, pour des résultats plus ou moins probants. Voici l'algorithme permettant de dessiner les obstacles tout en gérant leur position :

- Début
    - Faire une homothétie pour avoir un ratio de 16/9.
    - Pour chaque obstacle de la liste :
        - Si la position de l'obstacle est supérieure à 0 (devant la caméra) :
            - Calculer la couleur correspondant à la lumière émise par la caméra (appelée `lightCamera`) et celle correspondant à la lumière émise par la balle (appelée `lightBall`, vaut 0 si la balle est derrière l'obstacle).
            - Établir la couleur de l'obstacle en fonction de la source de lumière la plus proche (`lightCamera` ou `lightBall`).
            - Selon la position de l'obstacle (droite, gauche, haut, bas) :
                - Faire une homothétie en suivant la taille de l'obstacle.
                - Faire une translation en X et Y à la position souhaitée en suivant la taille de l'obstacle.
            - Selon la taille de l'obstacle :
                - Dessiner l'obstacle avec la bonne texture.
    - Calculer la couleur correspondant à la lumière émise par la caméra (appelée `lightCamera`) et celle correspondant à la lumière émise par la balle (appelée `lightBall`) pour le mur de fin.
    - Établir la couleur du mur de fin en fonction de la source de lumière la plus proche (`lightCamera` ou `lightBall`).
    - Faire une translation jusque derrière la position en Z du dernier obstacle.
    - Dessiner le mur de fin
    - Fin

Gestion des collisions :

Quatre fonctions nous permettent de simuler les collisions entre les différents éléments du jeu : `collCorridor` (collisions entre la balle et les murs du couloir), `collWall` (collisions entre les obstacles et la balle et la raquette), `collBonus` (collisions entre la raquette et les bonus) et `collRaquette` (collisions entre la balle et la raquette). Ces fonctions font globalement appel à des tests de conditions sur les positions en X, Y et Z des éléments du jeu et modifient les variables selon les résultats de ces tests. Voici les algorithmes détaillés de ces différentes fonctions :

`collCorridor` :

- Début
    - Si la position X de la balle est inférieure à 0 ou supérieure à 1280 :
        - Inverser la vitesse en X de la balle.
    - Si la position Y de la balle est inférieure à 0 ou supérieure à 720 :
        - Inverser la vitesse en Y de la balle.
- Fin

`collWall` :

- Début
    - Pour chaque obstacle de la liste :
        - Selon la position de l'obstacle (droite, gauche, haut, bas) :
            - Déterminer les bords de l'obstacle en X et en Y.
        - Si la distance en Z entre la balle et l'obstacle est inférieure à 0.5 ET la position en X et en Y de la balle est contenue dans les bords de l'obstacle définis précédemment OU la distance entre la balle et l'un des bords de l'obstacle est inférieure à 0.5 :
            - Inverser la vitesse en Z de la balle.
        - Si la position de l'obstacle se situe entre -0.5 et -0.9 (l'obstacle touche la caméra) ET la position en X et en Y de la souris est contenue dans les bords de l'obstacle définis précédemment :
            - Mettre `player.forward` à 0 (empêche le joueur d'avancer).
- Fin

`collRaquette` :

- Début
    - Si la balle n'est pas collée à la raquette ET la position en Z de la balle est supérieure à 0.5 (balle derrière la raquette) ET la distance en X et en Y entre la balle et la souris est inférieure à 148 ET le champ `player.sticky` est égal à 0 (bonus de la colle non actif) :
        - Inverser la vitesse en Z de la balle.
        - Calculer la vitesse en X et en Y de la balle en fonction de la distance par rapport à la souris.
        - Replacer la balle devant la raquette.
    - Sinon si la balle n'est pas collée à la raquette ET la position en Z de la balle est supérieure à 0.5 (balle derrière la raquette) ET la distance en X et en Y entre la balle et la souris est inférieure à 148 ET le champ `player.sticky` est égal à 1 (bonus de la colle actif) :
        - Coller la balle à la raquette.
        - Mettre sa vitesse à 0 en X et Y et à 0.25 en Z.
        - Immobiliser le joueur.
    - Sinon si la balle n'est pas collée à la raquette ET la position en Z de la balle est supérieure à 0.5 (balle derrière la raquette) :
        - Coller la balle à la raquette.
        - Mettre sa vitesse à 0 en X et Y et à 0.25 en Z.
        - Immobiliser le joueur.
        - Réduire le nombre de vies du joueur de 1.
    - Si la position du mur de fin est supérieure à 0.5 :
        - Afficher le menu de victoire.
- Fin

`collBonus` :

- Début
    - Pour chaque bonus de la liste :
        - Si le bonus est visible ET sa position en Z est entre -2 et 1 ET sa distance par rapport à la souris en X et en Y est inférieure à 100 ET (le bonus est du type 'gain de vie' OU le bonus est de type 'colle' tout en étant inactif) :
            - Rendre le bonus invisible.
            - Si le bonus est du type 'gain de vie' :
                - Augmenter le nombre de vies du joueur de 1.
            - Si le bonus est du type 'colle' :
                - Mettre `player.sticky` à 1.
- Fin

DIFFICULTÉS RENCONTRÉES

Difficultés liées à l'organisation :

La difficulté principale était liée à la différence de niveau entre Léo et Heïdi en termes de programmation en raison de leurs parcours respectifs. On a alors décidé de travailler ensemble la plupart du temps afin que l'on puisse s'investir le plus équitablement possible.

Difficultés liées au développement :

Nous avons eu beaucoup de difficulté à gérer le dessin des éléments avec les différents repères dans lesquels ils étaient manipulés. Cela découle principalement de la gestion des positions des éléments qui ne sont pas toutes calculées suivant la même échelle, rendant leur cohésion difficile. En résultent des translations et homothéties assez approximatives qui auraient gagné à être plus rigoureusement calculées.

De plus, nous avons mis du temps avant de réussir à gérer les collisions entre la raquette et les obstacles et bonus. Nous n'avions pas remarqué, au départ, que la fonction `mouse_button_callback` ne prenait pas en compte les clics continus et ainsi les collisions ne se faisaient pas lorsque le joueur avançait (impliquant de maintenir le bouton droit de la souris appuyé). Nous avons pu régler cela pour les obstacles en ajoutant une variable booléenne `isDown` dont la valeur est true s'il y a un clic droit continu. En revanche, la collision avec les bonus reste imparfaite. Nous n'avons malheureusement pas eu le temps de régler ce problème-là.

Enfin, comme dit plus haut, nous avons dû simuler une illumination en utilisant des variations de couleurs sur les obstacles. Si cela fonctionne pour la lumière provenant de la caméra, le résultat est un peu plus étrange pour celle émise par la balle. Nous n'avons pas non plus eu le temps de régler de problème.

AMÉLIORATIONS POSSIBLES

En premier lieu, il convient bien entendu de régler les erreurs pré-existantes. Cela comprend la gestion des collisions entre la raquette et les bonus, mais aussi la mise en place d'une illumination réaliste. Ce serait aussi bien de revoir la collision entre la balle et les grands murs car celle-ci leur passe au travers au niveau des bords. Il serait également intéressant d'animer les murs du couloir afin de donner pleinement l'impression que le joueur avance.

Une fois ceci fait, on pourrait densifier ce jeu très primaire en rallongeant le couloir et en ajoutant des niveaux. On pourrait également ajouter de nouveaux types d'obstacles (tailles variables, obstacles mouvants, formes plus complexes que de simples rectangles) et de bonus (téléportation, ralentissement de la balle, suppression d'obstacles, ...) voire implémenter des boss de fin pour chaque niveau. Tout ceci serait inspiré des éléments présents dans le jeu Minecraft afin de conserver une direction artistique cohérente.

Sur un plan esthétique, on pourrait ajouter des visuels en fonction des éléments à ajouter. On pourrait proposer différents décors inspirés de Minecraft et éventuellement les modéliser en 3D plutôt que d'utiliser des textures 2D. Cela pourrait s'accompagner d'animations et d'effets tels que des sources de lumière spécifiques ou des effets de particule.

Dans le but de rendre le jeu plus immersif et ainsi plonger dans l'univers Minecraft, nous pourrions ajouter des effets sonores avec déjà une musique de fond et ensuite des bruitages. Chaque bruitage serait relié avec un obstacle du jeu (Quand le joueur s'approche d'un squelette, d'un zombie, d'une araignée ou encore un creeper, le bruitage serait différent).