Rapport de projet : Bubble-Bobble

IHDC B132

Cédric Evrard

1 Introduction

L'object de ce projet est de réaliser notre propre version du jeu vidéo Bubble-Bobble [1]. Ce jeu vidéo sera réalisé à l'aide du langage de programmation C ainsi qu'avec les bibliothèques graphiques OpenGL et GLUT.

La réalisation du jeu est divisée en 2 étapes. Une premire étape, imposée, sera de réaliser le coeur du jeu, c'est à dire un personnage fonctionnel et la présence d'ennemis, le tout dans un seul niveau. La deuxième partie sera libre et l'objectif de celle-ci sera d'améliorer le jeu de la meilleur des façons via, par exemple, l'ajout d'une intelligence artificielle pour les ennemis, la création de plusieurs ennemis, de plusieurs niveaux, ...

2 Projet

2.1 Présentation des écrans

2.1.1 Écran d'accueil

L'écran d'accueil sera affiché à l'ouveture du programme. Sur cet écran (Figure 1), il y aura le titre du jeu ainsi qu'un menu de sélection permettant d'accéder aux autres écrans de l'application.

Figure 1: Écran d'accueil

2.1.2 Écran de contrôles

Cet écran (Figure 2) sera affiché avant chaque partie afin de présenter au joueur les contrôles du jeu afin qu'il puisse facilement comprendre comment jouer au jeu.

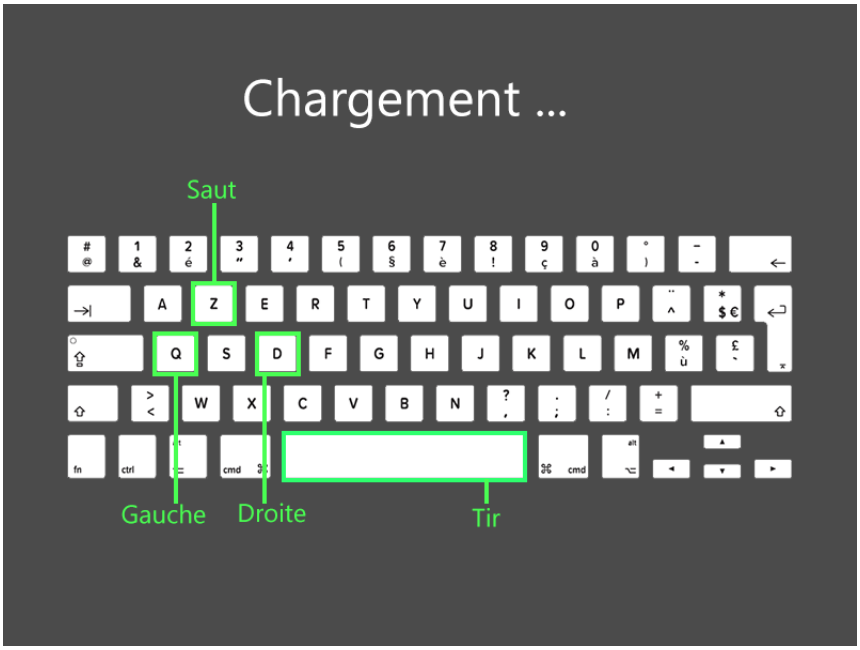

Figure 2: Écran des contrôles affichés au chargement

2.1.3 Écran de jeu

Sur l'écran de jeu (Figure 3), écran principal de l'application, différentes informations seront données au joueur, telles que le niveau, le score (affiché en haut à gauche), le niveau (en haut à droite) et le nombre de vie, (en bas à gauche).

2.2 Fonctionnement du coeur du jeu

Cette section décrira les diffrents éléments pour le fonctionnement du jeu, à savoir comment le monde est représenté ou comment la détection entre deux éléments du jeu sera faite.

2.2.1 Représentation de la carte

La carte sera divisée en une grille de 32 cellules de large sur 25 cellules de haut. Chaque cellule de cette grille sera un mur ou une zone vide. Ensuite, chaque bloc aura une taille de 32 pixels de large sur 32 pixels de haut et sera rempli avec une texture.

2.2.2 Représentation des personnages

Les personnages du jeu, que cela soit pour le personnage joueur ou pour les ennemis, seront représentés à l'aide d'une image qui sera positionnée sur un axe x-y. Le personnage sera entouré d'une "hitbox" [2] (voir exemple figure 4), un rectangle qui permettra de savoir quand deux éléments entre en collision. C'est à dire, lorsque les deux rectangles d'une "hitbox" se rencontrent.

Le même système sera utilisé pour les bulles lancées par le joueur. Même si une bulle représentée par

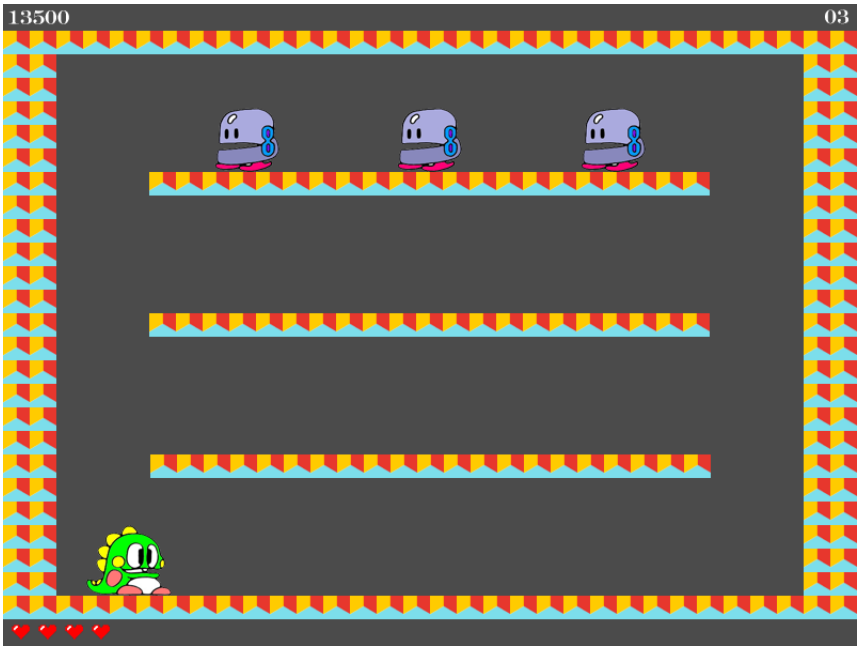

Figure 3: Écran de jeu

Figure 4: La “hitbox” est représentée en rouge, celle-ci sera invisible en jeu

une image ronde, la “hitbox” de celle-ci sera aussi un rectangle afin de faciliter les calculs liés à la détection de “hit”.

2.2.3 Gestion des mouvements

La gestion des mouvements se fera via les touches ‘Q’ (mouvement à droite), ‘D’ (mouvement à gauche) et ‘Z’ (mouvement de saut). Lorsque le joueur va à gauche ou à droite, le jeu va vérifier que le joueur ne rentre pas en collision avec un mur. Pour cela, un algorithme va regarder la position du joueur par rapport au tableau qui représente la carte. Si le joueur entre dans une cellule du tableau représentant un mur, il sera arrêté dans son mouvement.

Lorsque le joueur saute (via la touche ‘Espace’), le jeu va détecter si celui-ci était en mouvement lors du saut ou non. Si celui-ci n’était pas en mouvement, le saut sera réalisé à la vertical par rapport à sa position de départ. Dans le cas d’un saut alors que le joueur est en mouvement, le saut sera réalisé sous la forme d’une hyperbole.

Lors de la période durant laquelle saute le joueur monte, il ne sera pas bloqué par les murs présents

dans le niveau (seul les murs qui entourent la carte seront bloquant). Par contre, lors de la chute, les murs bloqueront le joueur.

Enfin, lors de la chute du joueur (que ça soit après un saut ou lors d'une chute, le joueur aura la possibilité de choisir la direction de sa descente, ainsi que de ralentir la chute en appurant sur la touche de saut.

2.2.4 Élimination des ennemis

Lorsque le joueur lance une bulle sur un ennemi, celui-ci sera directement transformé en bonbon. Le joueur devra ensuite aller le récupérer pour gagner des points. Les bonbons resteront à l'écran un temps défini, après quoi, ils disparaitront de l'écran.

2.3 Stuctures utilisées

2.3.1 Structure de "hitbox"

Une "hitbox" sera représenté par une position dans l'écran de jeu ainsi qu'une hauteur et une largueur.

```
typedef struct hitbox {
  int positionX;
  int positionY;
  int hauteur;
  int largueur;
} Hitbox;
```

2.3.2 Structure de personnage

Les personnages, que ça soit le personnage joueur ou les ennemis, seront représentés à l'aide d'une hitbox, un nombre de vie, une chaine de caractère qui donnera le chemin vers la texture du personnage ainsi qu'un booléen indiquant s'il s'agit d'un personnage amical ou non.

```
typedef struct personnage {
  Hitbox hitbox;
  int nombreVie;
  char *image;
  bool amical;
  int nombrePoint;
  int vitesse;
} Personnage;
```

2.3.3 Structure de la liste des personnages

Chaque élément de la liste chaîné sera représenté via une structure contenant une référence vers un personnage ainsi qu'une deuxième référence vers l'lément suivant.

```
typedef struct personnageListItem {
  Personnage *personnage;
  PersonnageListItem *next;
} PersonnageListItem;
```

2.3.4 Structure des scores

Sera utilisé pour sauvegarde le score des joueurs ainsi que le nom du joueur.

```
typedef struct score {
  int score;
  char *nom;
} Score;
```

2.4 Stuctures de données dynamiques utilisées

2.4.1 Gestion des collisions

Afin de faciliter la recherche de collision entre deux éléments du jeu, une liste chaînée contenant les informations de position et de taille des “hitbox” sera utilisée. Cette liste aura pour but de parcourir le plus rapidement possible les “hitbox” et savoir si deux de celles-ci se croisent.

2.4.2 Ennemis

Les ennemis seront eux aussi présents dans une liste chaînée afin de pouvoir facilement ajouter ou retirer des ennemis du jeu.

2.5 Utilisation des fichiers

Des fichiers seront utilisés pour les niveaux ainsi que pour la sauvegarde des scores des joueurs.

2.5.1 Sauvegarde des niveaux

Chaque niveau sera sauvegardé dans un fichier, le fichier sera au format binaire afin de pouvoir facilement sauvegarder l'ensemble des structures *hitbox* composant le niveau.

Les fichiers des niveaux seront généré par le programme au lancement de celui-ci si ceux-ci n'existent pas. Dans les versions suivantes du jeu, il sera possible via une interface de crer des niveaux directement dans le jeu afin de faciliter leur création.

2.5.2 Sauvegarde des scores

Seul les 10 scores les plus importants seront gardés dans un fichier binaire. L'utilisation du fichier binaire permettra de directement aller lire puis écrire les nouveaux scores à l'endroit souhaité.

2.6 Découpage du programme

Le programme sera divisé en plusieurs composants afin de faciliter la gestion du code.

2.6.1 Core

Le *Core* contiendra les fonctions appelé par GLUT pour réaliser l'affichage, la gestion du clavier, ...

```
void init(void);
void display(void);
void reshape(int w, int h);
void processNormalKeys(unsigned char key, int x, int y);
void processSpecialKeys(int key, int x, int y);
```

2.6.2 Engine

L'*engine* contiendra les fonctions qui permettront de gérer le déplacement du joueur, l'intelligence des ennemis, les collisions entre le joueurs et les ennemis ou encore lorsqu'un ennemi est touché par une bulle et que celui-ci doit se transformer en bonbon.

```
bool collision(Hitbox *hitbox1, Hitbox *hitbox2);
void deplacement(Personnage *personnage);
bool toucher(Hitbox *hitbox1, Hitbox *hitbox2);
```

2.6.3 FileManager

Le *FileManager* contiendra toutes les fonctions permettant de charger ou d'écrire dans les fichiers utilisés par le joueur.

```
void chargementNiveau(char *path);
void chargementImage(char *path);
void chargementScore(char *path);
void sauvegardeScore(char *path, Score score);
```

Les fonctions de chargement ne reçoivent pour le moment que le chemin du fichier à charger car la façon de retourner le résultat va dépendre de l'implémentation des chargement à proprement dit.

3 Conclusion

La principale difficulté de se projet va résider dans le fait d'optimiser la gestion des hitbox afin que l'application ne soit pas ralentie par la recherche de collision. En effet, à chaque tour de la boucle principale de GLUT, on va devoir vérifier si deux éléments sont entrés en collision ou non. Il faut donc optimiser au maximum cette gestion.

La deuxième difficulté sera liée à la gestion des mouvements du personnage joueur et des ennemis. Ceux-ci pouvant se déplacer librement, il faudra faire attention à ne pas bloquer le personnage joueur dans un mur ou à l'inverse, ne pas l'autoriser à se déplacer trop librement sur la carte et donc sortir de celle-ci.

4 References

[1] "Bubble Bobble - StrategyWiki," https://strategywiki.org/wiki/Bubble_Bobble.

[2] "Hitbox - VALVe Developer Community," https://developer.valvesoftware.com/wiki/Hitbox.