

Thomas Was **A Clone**

Projet de synthèse d'image

IMAC1 - Promotion 2017

Lucas Horand - Mégane Burckel


Introduction

Après quelques moi à avoir appris les bases de la programmation en C et en OpenGL, il est temps de s'y mettre !

Essayons de créer un mini jeu avec nos nouvelles connaissances : Thomas Was A Clone, une copie du jeu Thomas Was Alone.

Nous avons essayé de retranscrire l'esprit de ce jeu plus ou moins à notre sauce. Nous avons tenté divers algorithmes, et découvert différentes bibliothèques SDL pour y parvenir. Tout au long de ce rapport, seront présentés nos problèmes rencontrés à travers cette expérience, ainsi que les choix auxquels nous étions confrontés.

La version fournie est la version Collector Correcteur's Cut 3.0, contenant un cheat code exclusif : appuyer sur N permet de passer au niveau suivant.

The image shows a screenshot of the title screen for the game 'THOMAS WAS A CLONE'. The background is a solid purple color. The title 'THOMAS WAS A CLONE' is displayed in a white, pixelated, all-caps font, centered horizontally. Below the title, the text 'PRESS ENTER TO START' is also displayed in the same white, pixelated, all-caps font, centered horizontally.

THOMAS WAS A CLONE

PRESS ENTER TO START

Structure utilisée

Architecture des dossiers

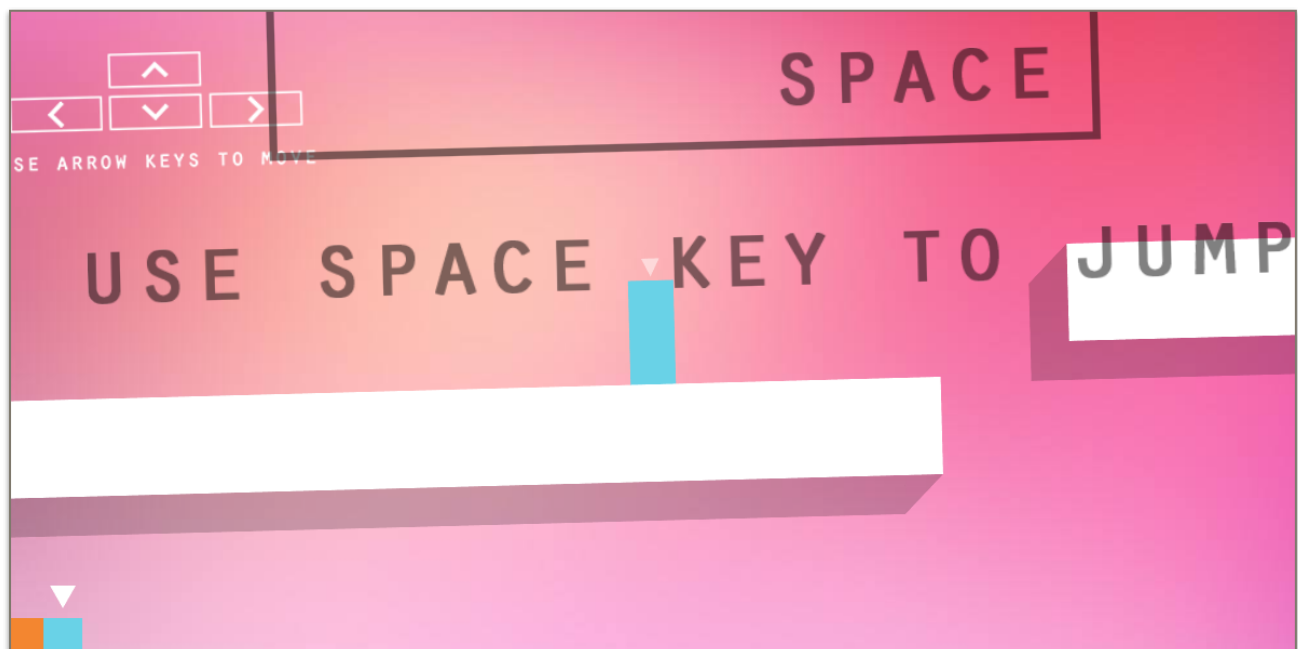
Afin de mieux nous organiser dans notre travail, nous avons divisé le répertoire de travail, en séparant dès le départ les **assets** (textures, musiques, ...), le code (dans un répertoire **src**, les **includes**, et les **obj (.o)** dans un répertoire lui aussi différent. Le Makefile se charge de dispatcher les fichiers aux endroits nécessaires.

Ce même makefile étend le domaine des inclusions (par seulement include.h) mais aussi un répertoire sous Ubuntu contenant SDL_Mixer.

Stockage des variables

Tous nos principaux objets (personnages, textures, murs) sont contenus dans des **listes**. Ce choix est principalement justifié par le fait de ne pas avoir à gérer la taille du tableau, qu'il s'agisse de réallots ou de parcours de tableau. De plus, cette décision s'inscrit dans l'optique d'avoir un jeu évolutif et pouvant plus facilement intégrer de nouvelles fonctionnalités, comme le fait de pouvoir ajouter ou supprimer n'importe quel personnage en cours de partie beaucoup plus simplement.

Le contenu des niveaux est lu dans des fichiers textes, à raison d'un fichier par niveau. Les lignes de fichier commençant par P ajoutent un joueur, W un mur, T une texture, et S définissent les options de base du niveau.



Principales fonctionnalités

Mouvements du personnage

Pour faire bouger nos petits cubes, nous gardons 3 variables modifiées par l'utilisateur grâce à la SDL : **keyJump**, **keyLeft** et **keyRight**.

Si le joueur a appuyé sur espace (ou haut), keyJump vaut 1.

Si ce dernier a appuyé sur la flèche de gauche, keyLeft vaut -1,

et si il a appuyé sur la flèche de droite, keyRight vaut 1.

Le personnage étant caractérisé, entre autres, par sa position d'arrivée, de départ, actuelle et sa vitesse horizontale et verticale ainsi que sa vitesse de déplacement,

sa position n'est changée qu'à la fin de la boucle principale, où sa position en X et Y s'incrémente respectivement de sa vitesse horizontale et verticale.

Sa vitesse est calculée de la manière suivante: **dir * thomas->moveSpeed**

où dir est égal à **keyLeft + keyRight**. En effet, si seul keyLeft est enfoncé, sa vitesse sera négative. Si seul KeyRight, elle restera positive, et si les deux touches sont enfoncées en même temps, le personnage n'avance plus.

Le personnage subit la gravité, définie arbitrairement à 0.3 mais pouvant tout à fait devenir une variable d'un niveau, s'inverser, etc. A chaque frame, la vitesse verticale s'incrémente de la gravité*-1, sauf si le personnage est au sol (ou sur un personnage).

Dans ce cas, sa vitesse verticale vaut **keyJump * jumpSpeed**. En effet, si le personnage est en train de sauter, il suffit de donner un boost à sa vitesse, et le personnage retombera de lui-même avec la gravité. Si il n'est pas en train de sauter, il arrête néanmoins de tomber !

Collisions

Les collisions furent la grosse difficulté de ce projet. Si gérer la collision entre un personnage et des murs fut plutôt aisé, il en est tout autre pour les collisions entre plusieurs personnages subissant chacun des collisions d'autres joueurs et de murs !

Pour savoir si deux objets se collisionnent, nous avons écrit la fonction **placeMeeting** qui prend en paramètre les coordonnées de deux objets, renvoie 1 si ils se collisionnent, 0 sinon.

Pour gérer les collisions entre les différents objets, nous parcourons la liste de tous les personnages, et pour chaque personnage la liste de tous les murs.

```
si      bottom1 < top2
      et  top1 > bottom2
      et  right1 > left2
      et  left1 < right2
alors  retourne 1.
```

Problèmes rencontrés

Collisions

Connaissant déjà les algorithmes de base de la collision, ce ne fut pas si ardu. Il fallait néanmoins bien penser à utiliser les valeurs pour la frame suivante (position X + vitesse Horizontale par exemple), et à faire **reculer le joueur** si il entre en collision avec un élément, afin de le coller contre ce dernier.

Collisions entre joueurs

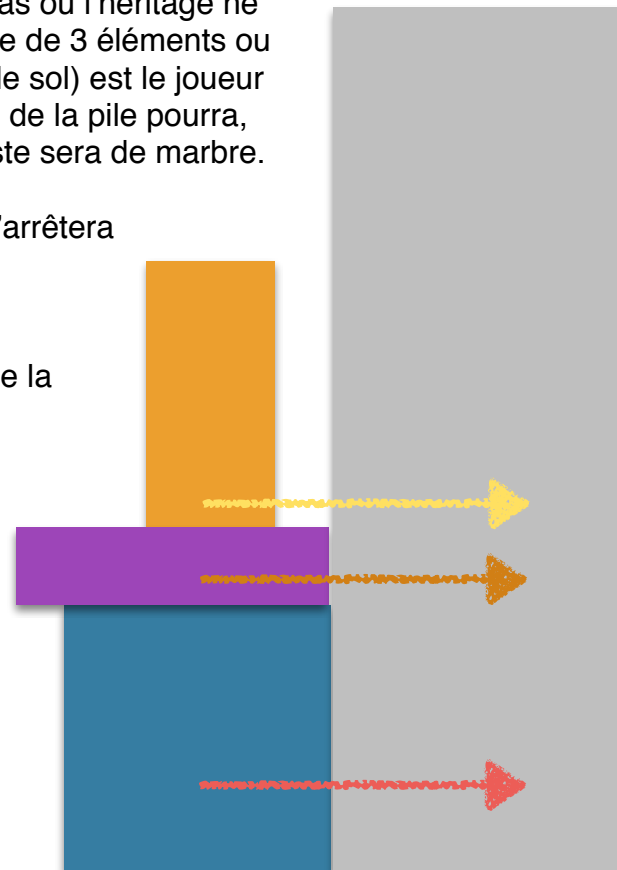
Une grande problématique qui nous a coûté une partie de notre âme. Les joueurs étant sur d'autres **héritent de la vitesse de ces derniers**. Nous avons ainsi ajouté une variable **isHolding** qui est un pointeur sur un autre bloc (son parent).

Ainsi, avant de s'occuper des collisions d'un bloc, nous faisons une boucle qui va, pour chaque personnage, récupérer si il y a la vitesse du bloc parent et l'appliquer à ce dernier. Néanmoins, un problème se pose : comment faire pour que dans une pile, chaque personnage hérite constamment de son parent, car la boucle s'occupe des joueurs un par un, et se faisant, un personnage traité en **fin de boucle** n'aura sans doute pas les mêmes valeurs que si ce dernier eusse été considéré en **début de boucle**...

La solution que nous avons trouvée consiste à faire une autre boucle si un parent est rencontré, qui va appliquer au parent la vitesse du parent du parent et au parent du parent la vitesse du parent du parent du parent, etc. Cette solution nous a semblé la plus opportune et la plus évidente, mais il reste un léger cas où l'héritage ne se passe pas, sans grande incidence sur le jeu : si une pile de 3 éléments ou plus se déplace et que le dernier parent (celui qui touche le sol) est le joueur actuel, qu'il se collisionne avec un mur, le premier joueur de la pile pourra, selon l'ordre de la pile, continuer à bouger alors que le reste sera de marbre.

Dans ce schéma, si le bloc bleu bouge à droite, le violet s'arrêtera alors que le orange continuera sa course effrénée.

Une solution au problème serait de démarrer par le pied de la pile.



Affichage de texte

Nous souhaitons afficher du texte pour indiquer au joueur comment se servir des commandes pour jouer au jeu et éventuellement raconter une histoire sur les personnages. Pour cela nous avons plusieurs solutions :

- Utilisation de la librairie Sdl_Ttf
- Utilisation des textures avec du texte sous forme d'image.
- Création d'une fonction de texte prenant une **spritesheet** de lettres, et découpant dedans pour reconstituer une chaîne de caractère sous forme d'une multitude de textures.

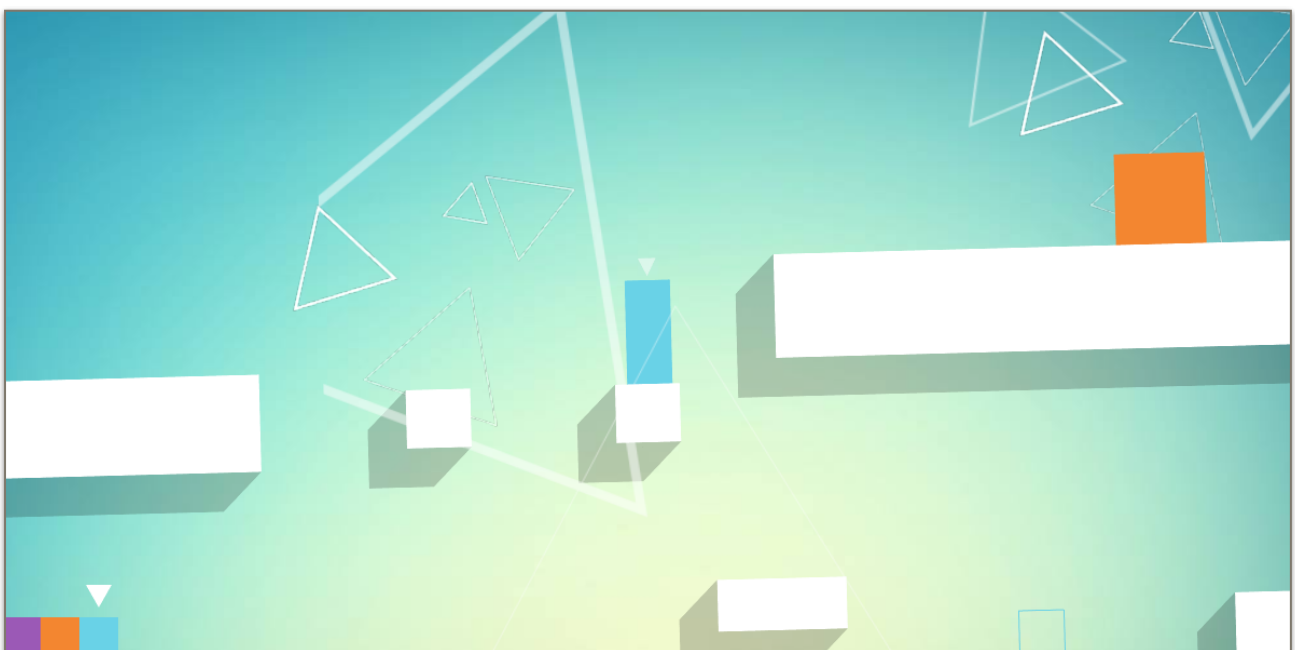
Après quelques essais non concluants avec la librairie SDL_ttf nous avons préféré l'utilisation d'images et de textures car cela permettait d'éviter l'installation d'une librairie spécifique, et c'était la solution la plus économique en temps puisque nous avons déjà codé une fonction permettant d'afficher des textures. De plus, la création d'une fonction de texte avancée aurait consommé beaucoup de ressources, peu nécessaire pour notre projet où le texte n'est pas mis en avant.

Création d'ombres

Pour donner une impression de volume aux éléments de notre parcours, nous avons eu envie d'utiliser des ombres. Nous pouvions...

- redessiner un cube de manière moins opaque
- projeter la largeur du personnage sur le sol qu'il occupe
- faire un pseudo raytracer (à partir d'un point lumineux, dessiner des polygones d'ombre en fonction des blocs pouvant occlure la source lumineuse).

Ayant déjà rencontré quelques difficultés avec les collisions, nous nous sommes tournés vers le plus simple, redessiner un cube moins opaque pour les murs, décalé par un léger offset. Nous avons néanmoins complété cette ombre avec deux triangles. Nous pouvons ainsi, si nous le souhaitons, agrandir ces ombres pour donner un aspect « long shadow ».



Fonctionnalités

Fonctionnel

- Création et dessin d'un personnage aux caractéristiques données
- Création et dessin de murs **et de leurs ombres**
- **Création de textures et affichages de ces dernières, comprenant la gestion de la transparence et un effet de parallaxe en fonction de la caméra**
- Mouvements du personnage et gravité
- Passage d'un personnage à l'autre avec la touche TAB **(et affichage d'une miniature des personnages)**
- Caméra suivant le personnage
- Collisions joueurs / murs
- Collisions joueurs / joueurs
- Gestion de la victoire (si tous les personnages sont sur leur position d'arrivée) et de leur mort (si un personnage sort du cadre de jeu)
- Gestion du vidage d'un niveau et remplissage avec le suivant.
- **Gestion des musiques, changeant à chaque niveau, ainsi que d'un bruitage de saut**

Nécessitant une légère amélioration

- Collisions au sein d'une pile importante de joueurs
- Support manette XBOX 360 : les boutons et les axes fonctionnent, mais sont trop différents en fonction des OS pour trouver un juste milieu. De même, sur un des Ubuntu utilisés, la latence pour le transfert manette/SDL était trop haute pour pouvoir l'implémenter dans le jeu (une grosse seconde entre le stick et la réaction).

Conclusion

Travailler sur un projet de cette ampleur fut une expérience très enrichissante, car sur la durée. Devoir travailler sur le long terme a impliqué de davantage prévoir ses algorithmes en amont ainsi que la structure de son code.

C'est un réel plaisir d'enfin voir l'algorithmique ainsi que la synthèse d'image finalement concevoir un réel objet ludique, utilisable et montrable même pour les personnes étrangères à l'informatique !

De même, nous sommes désormais plus apte à nous organiser dans le temps, car nous n'avions jamais été confronté à un tel projet. La grande question n'est pas « quand travailler pour terminer » mais « quand travailler pour garder du temps pour de nouvelles fonctionnalités ».

Nous souhaitons continuer ce projet sur notre temps personnel, afin d'y ajouter quelques fonctionnalités supplémentaires (inversion de gravité, plateformes rebondissantes)...

Mais surtout, ce projet est une grande source d'inspiration car preuve de ce que nous pouvons désormais accomplir, là où il fallait auparavant compter sur des moteurs de jeu préconçus.

