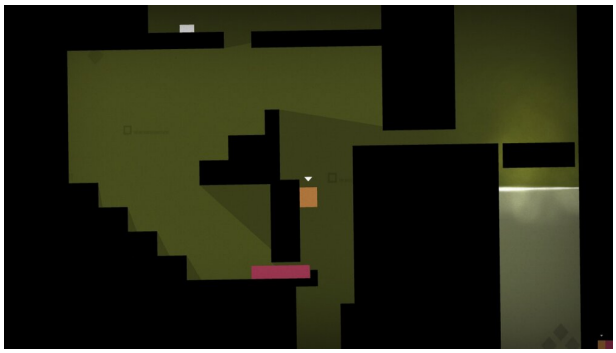
IMAC
INGÉNIEUR
ESIPE

Synthèse d'image I
Programmation-Algorithmique +
Projet
Les Ptimacs

Maintenant que vous êtes des développeurs affirmés, il est temps de mettre en pratique les nouvelles notions obtenues aux cours de Synthèse d'Image et d'Algorithmes Avancés. Le projet de ce semestre consiste en la réalisation d'un programme permettant de jouer à un jeu de plateforme qui met en scene un groupe de cubes qui doivent collaborer pour atteindre leur rêve. Ce programme sera inspiré du jeu Thomas Was Alone.

Pour ce faire, vous allez donc devoir réaliser un programme en C/C++r en utilisant la librairie OpenGL pour faire l'affichage et en implémentant des algorithmes pour gérer la physique, notamment les collisions.
Le but étant d'exploiter/introduire des compétences sur :

- Les fonctions de dessin 2D OpenGL.
- Les bases de la programmation de jeu vidéo.
- Les bases de la physique 2D.
- Les structures d'arbres notamment le quadtree.

Modalités de rendu

- **Nombre de personnes par projet :** 2 ou 3
- **Livrable :**
  — Les sources C/C++ ainsi qu'un système de compilation permettant de compiler sur un système Linux avec les packages/librairies usuel.le.s SDL, SDL2, OpenGL, SFML, GLU, GLUT, SDL_image, SDL_ttf disponibles ainsi que make, gcc, g++ et cmake). Si vous avez une spécificité de compilation (une librairie qui n'est pas listée ici ou un autre système de compilation (python, premake...)), notifiez le aux chargés de TD + responsables de cours et s'ils sont d'accord, mettez un README qui explique la spécificité.
  — Un rapport au format PDF décrivant le projet (en particulier les aspects qui diffèrent du jeu original, inutile de répéter le sujet), votre méthode de travail, des détails techniques si vous souhaitez détailler une fonctionnalité, les difficultés rencontrées et enfin les améliorations possibles.
- **Soutenance :** Si les conditions le permettent, vous pourrez défendre votre projet durant une soutenance en faisant une démonstration de votre projet et en reprenant les différents points du rapport. Votre présentation durera 10 minutes et sera suivi d'une série de questions du jury pour compléter votre soutenance.
- **Date de rendu (non définitive) :** 23 mai 2022
- **Date de soutenance :** ?

1 Le jeu

Il s'agit de vous approprier le jeu *Thomas was alone*, un jeu très simple et pourtant ludique. Le but du jeu est de faire traverser un ensemble de rectangles mobiles un parcours qu'ils seraient incapables de franchir seuls.

1.1 Déroulement

Le début du parcours est défini par les emplacements des rectangles "personnages" que nous allons déplacer. Chacun d'eux a une taille différente. A chaque instant, un des rectangles est sélectionné et la vue du jeu est (relativement) centrée sur ce rectangle. On peut le déplacer ou changer de rectangle à l'aide des touches du clavier. Le but du jeu est de pouvoir déplacer l'ensemble des rectangles "personnages" pour les amener à leur position finale, matérialisée par un ensemble de rectangles blancs de même taille que les rectangles mobiles.

2 Les entités du jeu

2.1 Les rectangles

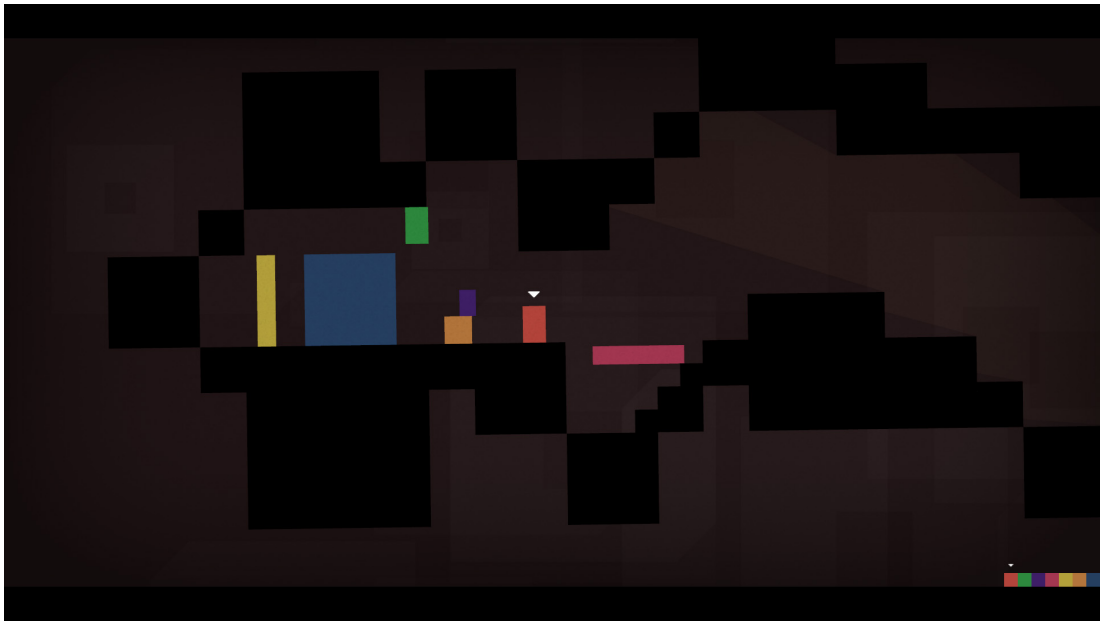
Les rectangles sont définis par leur attributs de base :
- taille
- couleur
- position de départ
- position d'arrivée
- puissance de saut

Vous pouvez en définir d'autres au besoin.
Tous les rectangles se manipulent avec le clavier de la même manière :
- droite
- gauche
- saut vers le haut, orientable à droite et gauche pendant le saut

Il faudra aussi définir une touche pour changer d'un rectangle à l'autre (par exemple : la touche TAB).

2.2 La carte

La carte définie par un ensemble de rectangles statiques et éventuellement par une identité visuelle.



2.3 Les niveaux

Un niveau est défini par un fichier contenant :
— l'ensemble des rectangles statiques formant la carte
— le nombre et les attributs des rectangles personnages
— d'autres paramètres optionnels que vous souhaiterez peut-être ajouter

2.4 Pour aller plus loin

Ce projet est très libre dans la façon que vous aurez de le concevoir, mais aussi dans les options que vous voudrez y apporter. Vous pouvez par exemple :
— faire une carte dynamique pour les binômes (background ou rectangles supports)
— faire un mode multijoueur
— rajouter de l'eau
— rajouter des boutons ou des actionneurs
— rajouter du son
— ...

Par contre, veillez à ne pas commencer à rajouter d'options tant qu'une version basique et jouable du jeu n'est pas réalisée.

3 Développement

Pour faire l'application vous devez respecter les contraintes décrites dans cette partie, tout écart devra être au préalable validé par vos évaluateurs et justifié dans le rapport et la soutenance.

3.1 Architecture Logiciel

Pour faire ce jeu vous allez devoir gérer une main loop ou game loop. Il s'agit d'une boucle conditionnelle qui fait tourner le jeu. Il va notamment

- gérer les inputs utilisateur
- calculer les interactions des différents éléments du jeu
- dessiner une frame

Cette boucle peut être gérer par un singleton qui représente le jeu même.
Pour gérer des menus, vous pouvez utiliser un state machine pour gérer les différents écrans de jeu.
vous pouvez trouver des tutoriels pour faire des jeux avec SDL dans les internets, par exemple http://www.sdltutorials.com

3.2 Synthèse d'Images - GUI

L'application devra être fluide et dotée d'une identité visuelle originale. Le jeu est assez simple d'un point de vue graphique et interaction donc une attention particulière sera donnée à l'IHM générale du jeu et à la fluidité du jeu.

3.2.1 GUI

L'application devra proposer à minima un écran de début de jeu, le jeu et un écran de fin de jeu (réussite). Les écrans de début et de fin de jeu devront proposer des textures. Le jeu peut également proposer des textures bien évidemment. Les choix dans les menus peuvent être fait via le clavier ou la souris.

Par ailleurs durant le jeu il sera nécessaire de montrer **visuellement** quel rectangle est, à tout moment, sélectionné par l'utilisateur par tout moyen qui vous semblera intéressant.

3.2.2 La carte

La carte devra être plus grande que la zone affichée. Il vous faudra donc gérer du "scrolling" afin de mettre le rectangle au milieu de la vue caméra. Évidemment à chaque changement de rectangle sélectionné, la vue devra se mettre "à jour".

3.2.3 Les déplacements

Les rectangles devront avoir un mouvement fluide, notamment au regard des contraintes (moteur physique, gravité) imposées. Dans ce cadre la gestion du temps sera probablement critique.

3.2.4 Trinôme uniquement : éléments mobiles

Ne concerne que les trinômes. Votre application devra comprendre des éléments de décor mobiles comme des plateformes en mouvement par exemple. Ces éléments devront avoir un mouvement fluide et devront pouvoir être en interaction avec les rectangle (autrement dit les rectangles doivent pouvoir grimper dessus par exemple).

3.3 Algorithmique - Physiques

Les rectangles (et peut être d'autres éléments du jeu) sont soumis à des règles physiques, notamment la gravité mais aussi les autres forces d'un système dynamique qui permettent notamment de pas rentrer dans le sol ou dans un mur.

3.3.1 Collision 2D

Le système de collisions de ce jeu sera simplifié. Le jeu ne mettant en scene que des éléments cubique et parallèles au sol, il suffit juste d'appliquer les collisions de type AABB, **Axis Aligned Bounding Box**. Les éléments étant alignés avec les axes X et Y, il suffit de tester si un élément ne rentre pas dans les coordonnées d'un autre, par exemple pour le sol, il faut juste s'assurer que notre cube ne dépasse pas le seuil en Y définir par le sol sur lequel il se trouve. Le site jeux.developpez.com a mis en ligne des articles (dont je ne retrouve plus l'auteur) qui peuvent vous aider à gérer les collisions, surtout si vous passez sur d'autres systèmes que le AABB.

Trinôme : les collisions devront prendre en compte vos éléments mobiles ce qui peut entraîner des difficultés dans la gestion du quadtree (voir plus loin). Les collisions avec les éléments mobiles peuvent se déterminer à part mais si vous trouver une meilleure façon, cela sera pris en compte.

3.3.2 Gravité

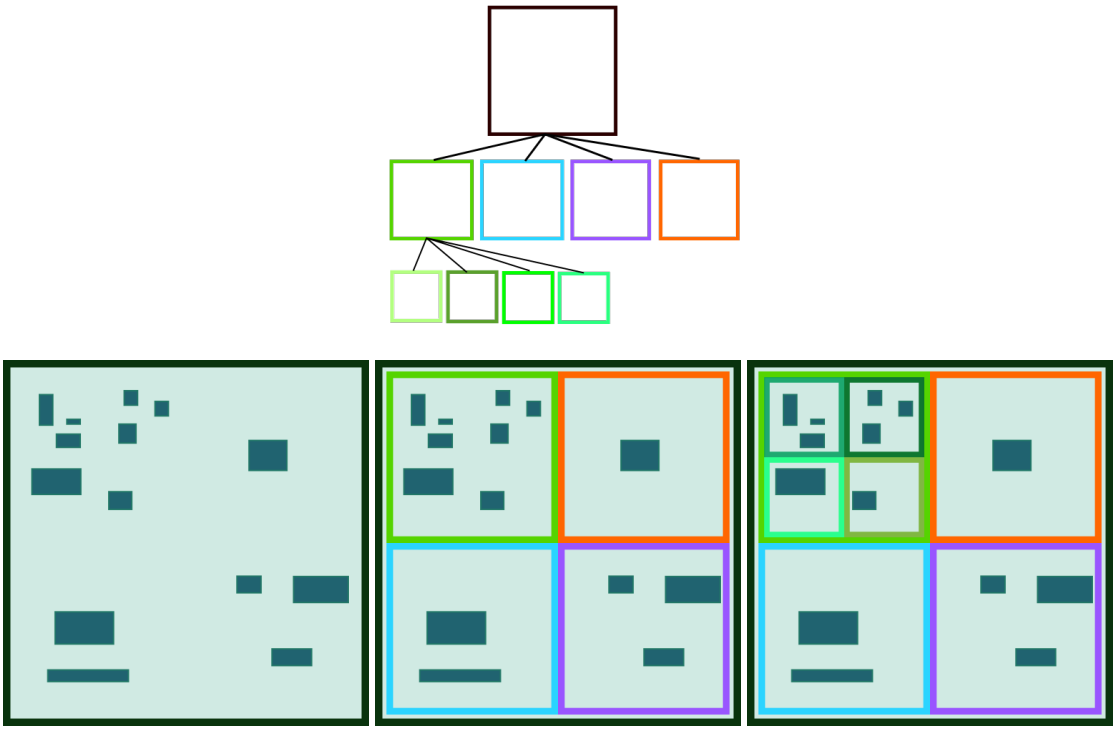
La gravité est très simple à gérer, il suffit d'appliquer un déplacement en Y vers le bas à chaque frame, ou, comme expliqué dans la section suivante, une accélération dirigée sur l'axe Y vers le bas.

3.3.3 Inertie et Accélération

Pour aller plus loin dans la physique du jeu, vous pouvez prendre en compte l'inertie et l'accélération. Pour ce faire, à chaque fois que vous souhaitez faire déplacer un élément, ne modifiez pas sa vitesse, modifier l'accélération, puis à chaque frame modifier la vitesse automatiquement en fonction de l'accélération puis la position en fonction de la vitesse. Cette accélération sera aussi impactée par des forces de frottements qui permettront de décélérer naturellement. Cela donnera un aspect plus lisse aux déplacements.

3.3.4 Le Quad Tree

Le *Quad Tree* ou Arbre Quaternaire est un arbre dont chaque noeud peut avoir jusqu'à 4 enfants. Semblable à l'arbre binaire, cette structure permet d'organiser des données pour mieux les retrouver, il est souvent utilisé pour partitionner des données dans un espace bidimensionnel, notamment pour ranger des éléments d'une carte. Chaque noeud de l'arbre représente un carré partitionnant l'espace et chaque enfant est une partition de son parent, la racine de l'arbre est un carré représentant la totalité de l'espace.



Cet arbre permettra de restreindre les calcules de collisions aux éléments proches des cubes du joueurs. On cherchera uniquement les noeuds de l'arbre qui contiennent les cubes à bouger.
Voici les propriétés à respecter

— Chaque noeud qui n'est pas une feuille aura 4 enfants qui le divisent en quatre espaces (supérieur droit, supérieur gauche, inférieur droit, inférieur gauche)
— Chaque noeud représente un espace et possède 4 coordonnées qui délimite celui-ci
— Chaque feuille contient une liste d'au maximum 4 éléments 2D

Au moment des calculs de collisions, il faudra récupérer le ou les noeuds de l'arbre qui contiennent les éléments susceptibles d'être touchés. Pour ça il faudra regarder la coordonnée de l'élément mobile, et chercher, à chaque niveau de l'arbre, la région (noeud) dans laquelle il se trouve. En récupérant une ou plusieurs feuilles, on pourra récupérer une liste d'éléments qui sont proches de notre cube et on pourra éviter des calculs inutiles avec du décors qui est à l'autre bout de la carte.

Note : Il existe l'équivalent 3D, l'octree qui range des points selon l'axe X, Y et Z.

4 Notation

Partie Commune (6)
- Rapport
- Soutenance
- Compilable par l'évaluateur
- Architecture logiciel et propreté du code
- Fonctionnalités du jeu

Partie Synthèse d'Images (10)
- Qualité et originalité graphique
- Qualité de la GUI
- Gestion du scrolling et des déplacements des rectangles

Partie Algorithmique (10)
- Collisions
- QuadTree

Bonus (4) (Soutenance, Originalité, Fonctionnalités supplémentaires...)

5 Références

https://www.bithellgames.com/thomas-was-alone
https://www.developer.com/languages/game-programming-fundamentals/
https://www.codeproject.com/Articles/1215961/Making-a-D-Physics-Engine-Mass-Inertia-and-Forces
https://jeux.developpez.com/tutoriels/theorie-des-collisions
http://devmag.org.za/2011/01/18/11-tips-for-making-a-fun-platformer/
https://www.cs.drexel.edu/~santi/teaching/2013/CS480-680/slides/CS480680-W6-Physics.pdf

6 Remarques

— Il est très important que vous réfléchissiez avant de commencer à coder aux principaux modules, algorithmes et aux principales structures de données que vous utiliserez pour votre application . Il faut également que vous vous répartissiez le travail et que vous déterminiez les tâches à réaliser en priorité.
— Ne rédigez pas le rapport à la dernière minute sinon il sera bâclé et cela se sent toujours.
— Le projet est à faire par binôme. Il est impératif que chacun d'entre vous travaille sur une partie et non pas tous "en même temps" (plusieurs qui regarde un travailler). sinon vous n'aurez pas le temps de tout faire. C'est encore plus vrai pour les trinômes.
— Utilisez des bibliothèques ! Notamment pour les types abstraits.
— N'oubliez pas de tester votre application à chaque spécification implémentée. Il est impensable de tout coder puis de tout vérifier après. Pour les tests, confectionnez vous tout d'abord de petites cartes (taille 5 par 5 par exemple) avec un chemin extrêmement simple. Si cela marche vous pouvez passer à plus gros ou plus complexe.

— Vos chargés de TD et CM sont là pour vous aider. Si vous ne comprenez pas un algorithme ou avez des difficultés sur un point, n'attendez pas la soutenance pour nous en parler.

6.0.1 README Exemple

```
Getting Help
------------

Need help understanding certain concepts in USD? See
[Getting Help with USD](http://openusd.org/docs/Getting-Help-with-USD.html) or
visit our [forum](https://groups.google.com/forum/#!forum/usd-interest).

If you are experiencing undocumented problems with the software, please
[file a bug](https://github.com/PixarAnimationStudios/USD/issues/new).

Supported Platforms
-------------------

USD is currently supported on Linux platforms and has been built and tested
on CentOS 7 and RHEL 7.

We are actively working on porting USD to both Windows and Mac platforms (Please
see [VERSIONS.md](VERSIONS.md) for explicitly tested versions). Support for both
platforms should be considered experimental at this time. Currently, the tree
will build on Mac and Windows, but only limited testing has been done on these
platforms.

Dependencies
------------

The following dependencies are required:
 - C++ compiler
 - C compiler
 - [CMake](https://cmake.org/documentation/)
 - [Boost](https://boost.org)
 - [Intel TBB](https://www.threadingbuildingblocks.org/)

The following dependencies are optional:
 - [Python](https://python.org)
```