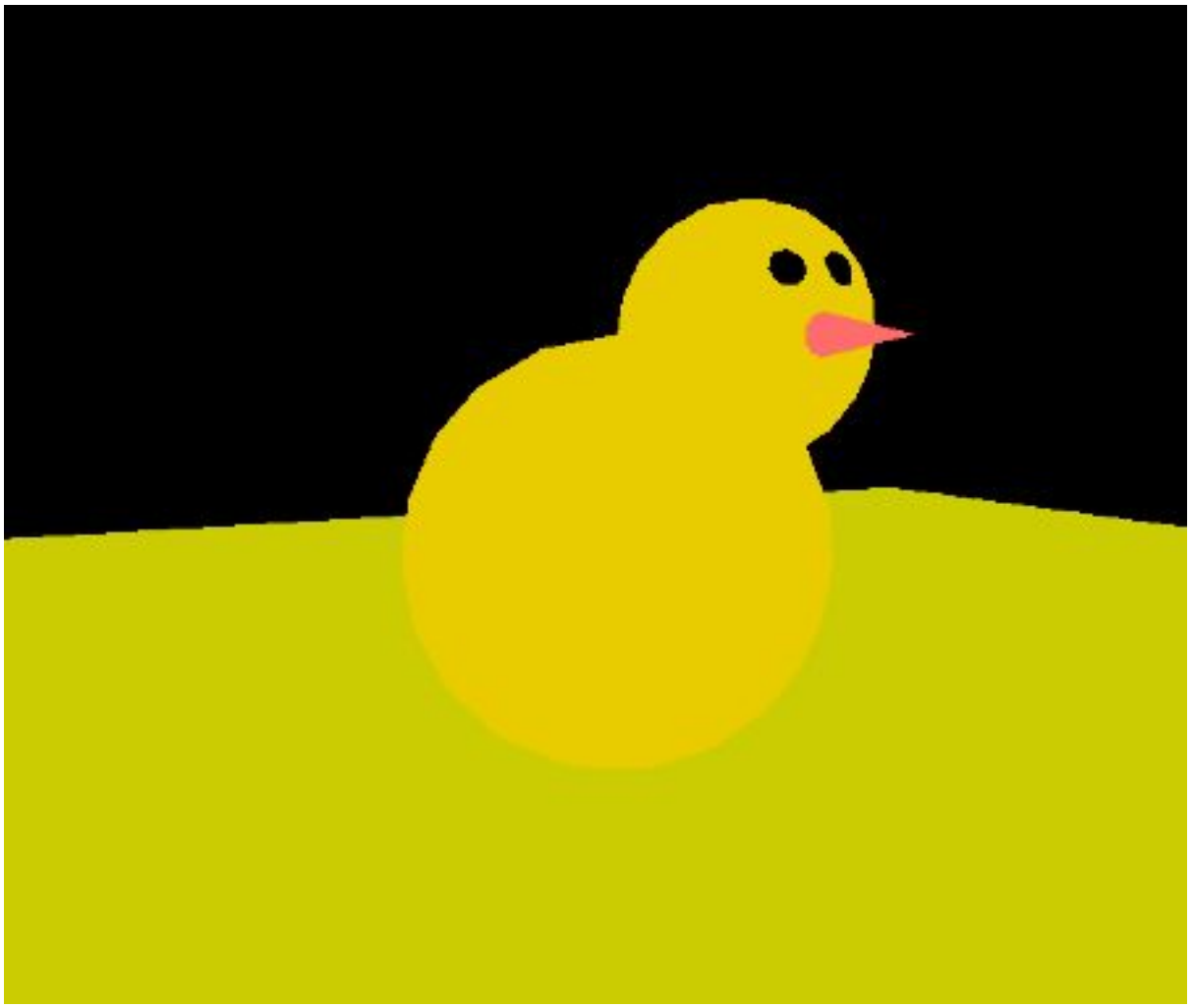
Octave Perrin de Brichambaut
Manon Vlieghe

P1RV: implémenter un jeu

Sujet 8:

Implémenter un jeu (objectifs et design de votre choix en accord avec l'encadrant) en C++ OpenGL ou OSG



Sommaire

Déroulement du projet:

Dans le cadre du premier projet d'option RV (P1RV), nous avons dû implémenter un jeu sur OpenGL ou OSG.

Nous avons consacré nos premières séances à la conception d'un scénario car nous trouvions plus logique de se fixer des objectifs puis de chercher des moyens de le réaliser.

Cependant, nous avons rapidement réalisé les limites de nos compétences et de la documentation;

- contrairement à Unity, nous ne sommes pas parvenus à trouver de cours suffisamment complet qui soit gratuit;
- La documentation nous était très chronophage à comprendre en l'absence d'une banque d'exemples d'utilisations, ici encore en comparaison avec d'autres supports de jeu ;

Nous avons donc changé d'approche et cherché progressivement quels moyens de jeu étaient implémentables sans trop de difficultés, puis exploités ceux-ci de diverses manières.

Introduction:

Pour démarrer, nous avons fait un choix dans nos outil, qui a consisté à se placer essentiellement (au final exclusivement) sur OpenGL.

Comme nous avons eu les cours d'OpenGL avant ceux d'OSG, c'est sur ce support que nous avons préféré nous lancer, d'autant que nous avions alors le TP_camera comme support pour débuter: nous avons donc complété cette base tout au long de notre travail.

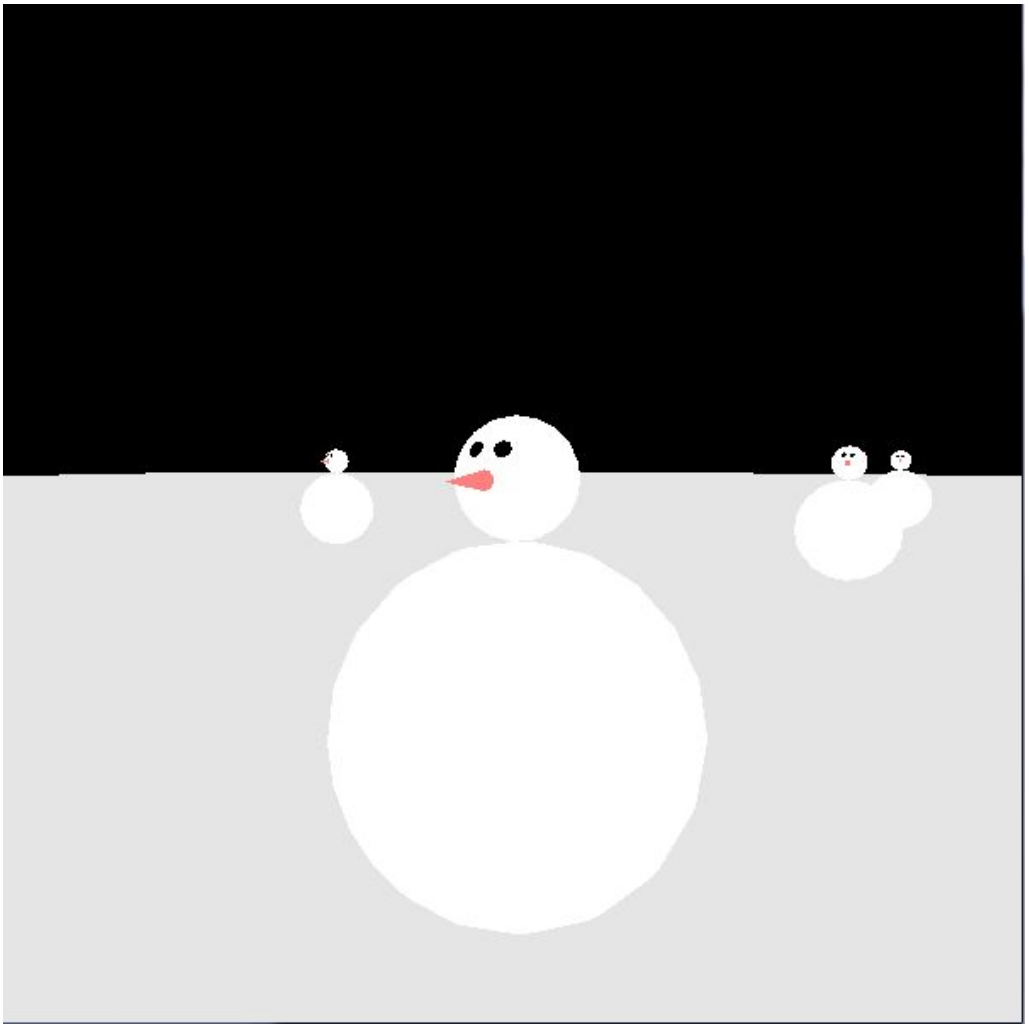


image1: Bonhomme de neige du TP camera

I-Classe Cube

Afin de créer nos premières interactions, nous avons créé la classe *Cube* qui permet de créer des blocs à l'emplacement et aux dimensions variables;

Celles-ci possèdent la fonction "*poke*" prenant en arguments notre position actuelle et l'endroit où l'on va (devant nous si on avance) et renvoie l'endroit on l'on va en tenant compte du cube: si on fonce sur un côté du cube, "*poke*" renvoie la projection du déplacement sur le plan du côté de sorte que l'on longe le côté du cube au lieu de passer au travers.

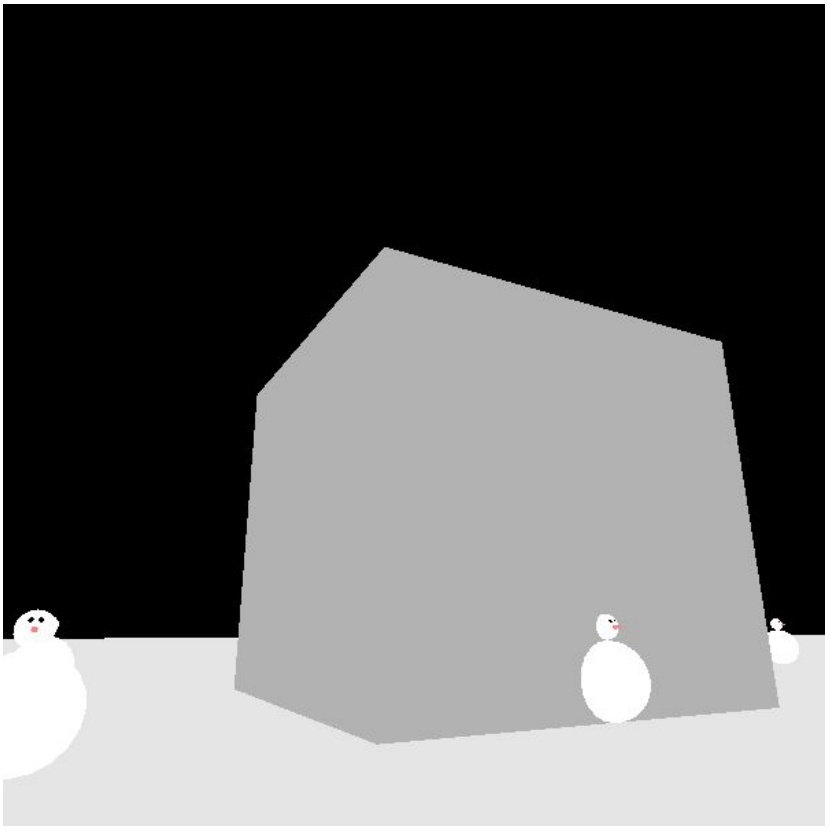


Image 2: Création de cube dans le champs de bonhomme de neige

Nos mouvements étaient cependant encore restreints au plan dans lequel on commençait (sauf en flymode, évidemment) . Afin de pouvoir ajouter la notion d'accélération, nous avons créer un Vector3D "*futur*" qui représente notre déplacement pour la frame suivante.

Ainsi, à chaque frame, on réduit *futur.Vy* d'une constante arbitraire (ainsi, on tombe de plus en plus vite) et on met les valeurs du déplacement comme précédemment pour Vx et Vz . Enfin, on applique l'effet des cubes, uns par uns pour que l'on puisse leur foncer dessus où tenir dessus.

II-Le saut

Une fois fait, nous avons voulu sauter; parce que juste tomber de plate-forme en plate-forme ne semble pas très intéressant comme gameplay. Pour sauter, nous avons donc commencé par rajouter une valeur positive à *futur.Vy* lorsqu'on appuie sur la barre d'espace. Mais si cela permettait de sauter, on pouvait également sauter en l'air.

Pour empêcher de sauter en l'air, il nous a fallu détecter si oui ou non on était au sol. pour ce faire, nous avons créé le booléen "*grounded*" qui détecte si oui ou non on est sur une plate-forme (et donc si oui ou non on peut sauter)

Originellement, *grounded* devenait vrai si on s'apprêtait à rentrer dans une plate-forme par le haut. Cependant, il arrivait que la commande de saut ne réponde pas.

En affichant à chaque frame le booléen grounded avec un cout, on obtient ceci:

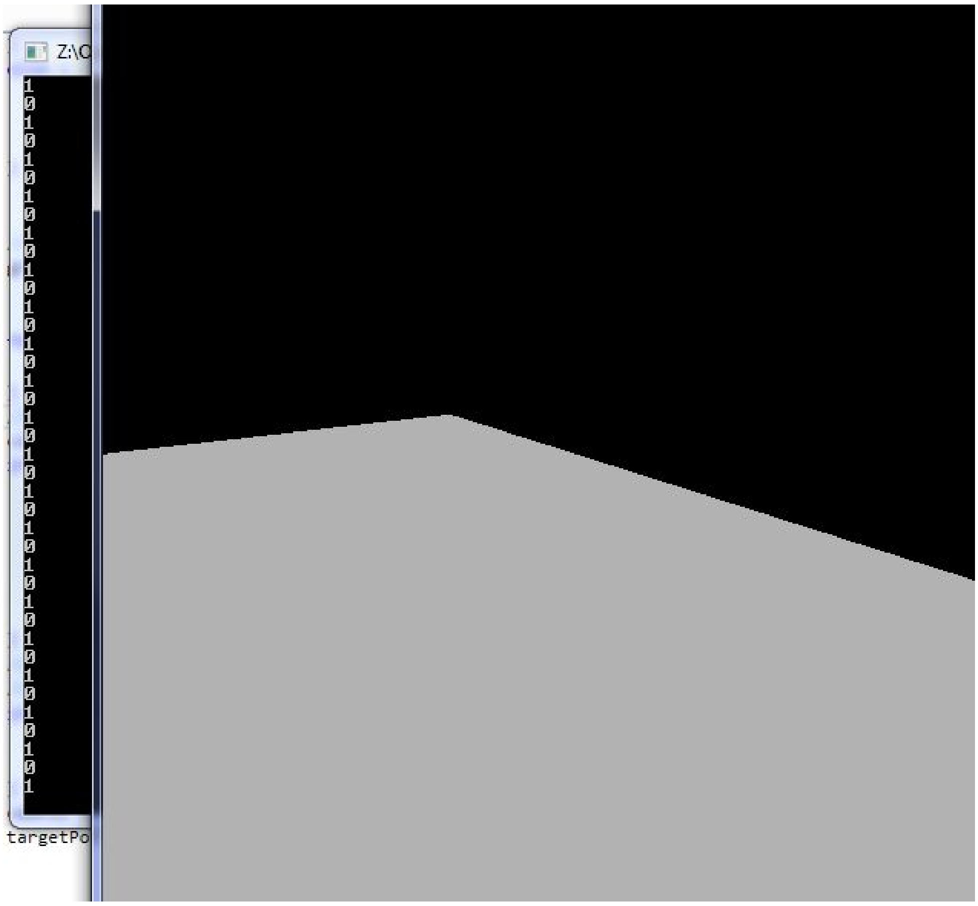


Image 3: Sortie de cout<<grounded<<endl; dans la console

En effet, si notre vitesse nous emmenait dans le cube, elle était réduite à 0 (selon l'axe y), et *grounded* valait vrai; mais alors on n'allait plus dans le cube! donc grounded prenait la valeur faux et on ne pouvait plus sauter, et ce jusqu'à ce que la gravité nous redonne assez d'élan pour retomber dans le cube.

Pour résoudre ce problème, nous avons rajouté une condition: si notre vitesse verticale est nulle et très proche de la surface du cube, alors grounded vaut vrai.

Nous avons donc pu sauter en toute liberté:

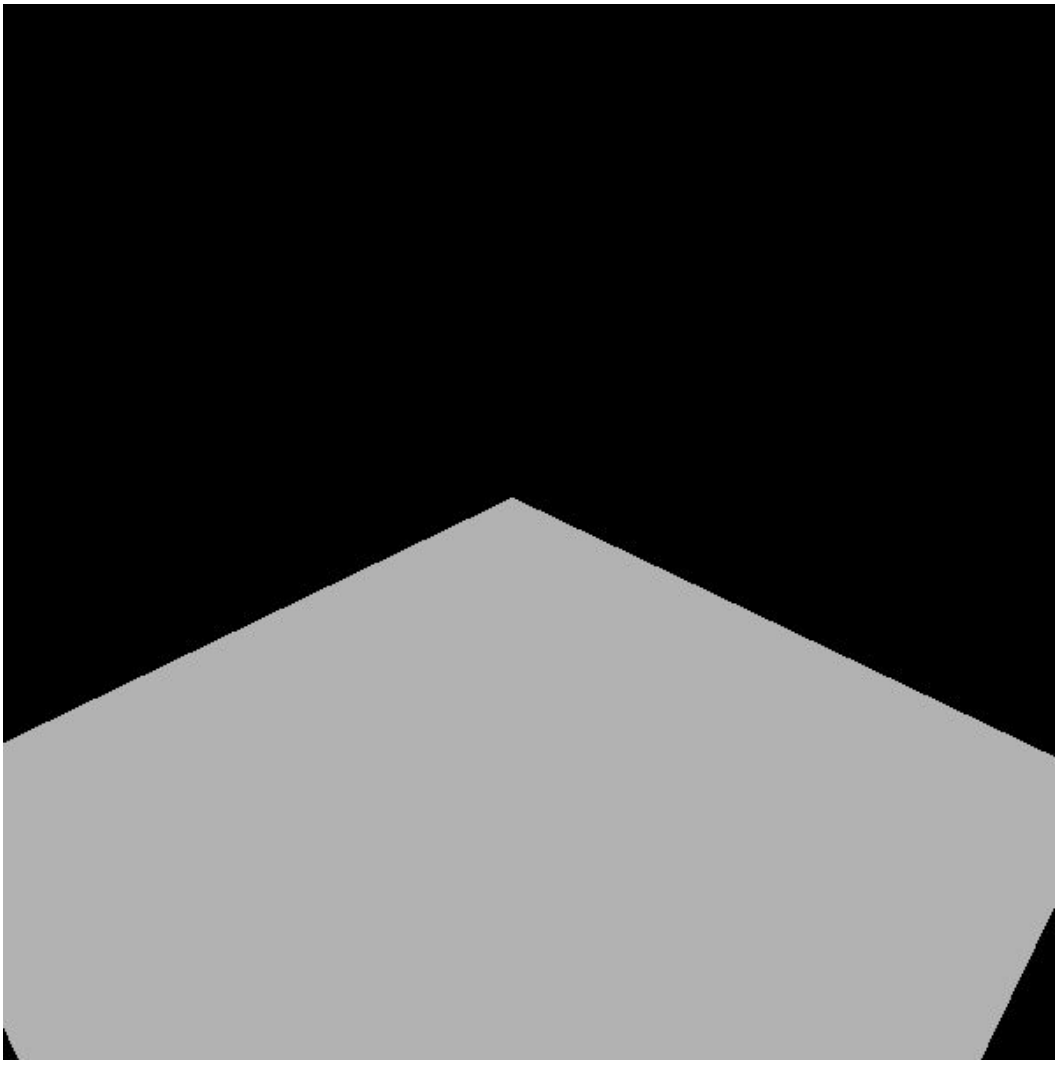


Image 4: Un saut bien mérité

Ensuite, il nous était possible de créer des niveaux intéressants avec de simples plateformes (la question du comment créer plusieurs niveaux n'était pas encore abordée).

Pour rajouter du contenu, nous avons fait le choix, respectant les normes clasiques du jeux de plateforme, de faire différentes plateformes pour différents

effets. Nous avons donc ajouté des cubes de différentes couleurs pour différents effets.

Pour ce faire, nous avons transformé la classe *Cube* en une classe abstraite (et *poke* en fonction virtuelle) de sorte que le *poke* ait des conséquences différentes selon le cube concerné.

Chaque classe a sa couleur pour permettre au joueur de distinguer les plateformes, et après un où deux essais de savoir sur quoi il va poser les pieds.

Ainsi furent créés les sous-classes suivantes:

-*Platform*: le cube normal:

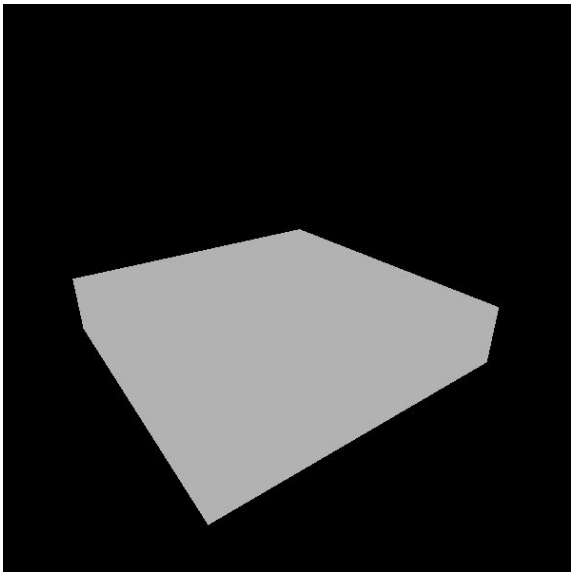


Image 5 : Un objet de la classe Platform de dimensions 5 par 1 par 5)

-*Trumpoline*: quand on arrive par dessus ou par dessus, change le signe de *futur.Vy*:

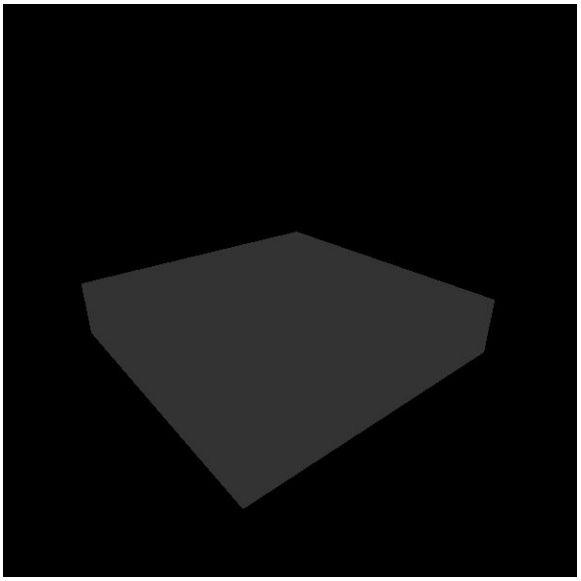


Image 6 : Un objet de la classe Trumpoline de dimensions 5 par 1 par 5)

-*Jumpoline:* quand on arrive par dessus, *futur.Vy* prend une valeur fixée au préalable:

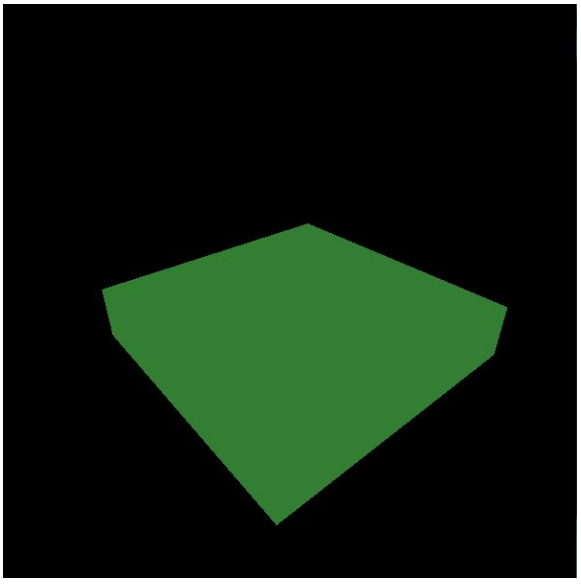


Image 7 : Un objet de la classe Jumpoline de dimensions 5 par 1 par 5)

-*Elevator*: quand on rentre dans le cube par le côté, on monte à une vitesse préfixée, une fois en haut, on peut marcher sur le cube comme sur une plateforme.

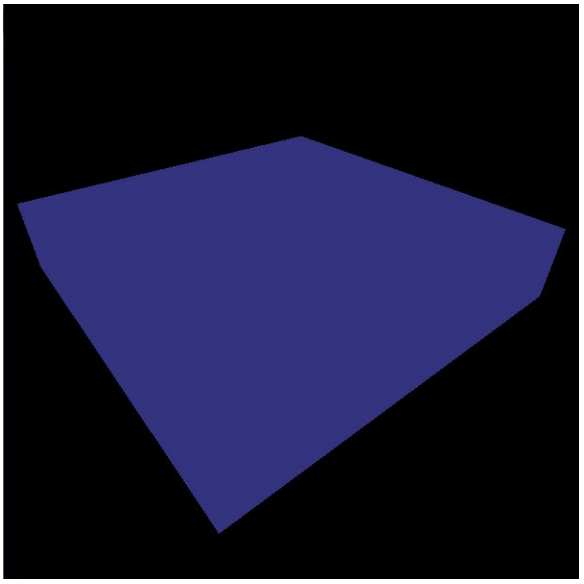


Image 8 : Un objet de la classe Elevator de dimensions 5 par 1 par 5)

Nous avons également ajouté un but (on a fait une nouvelle classe pour ça car on pensait faire un poke différent même si cela n'a finalement pas été le cas) sur lequel on a affiché un poussin.

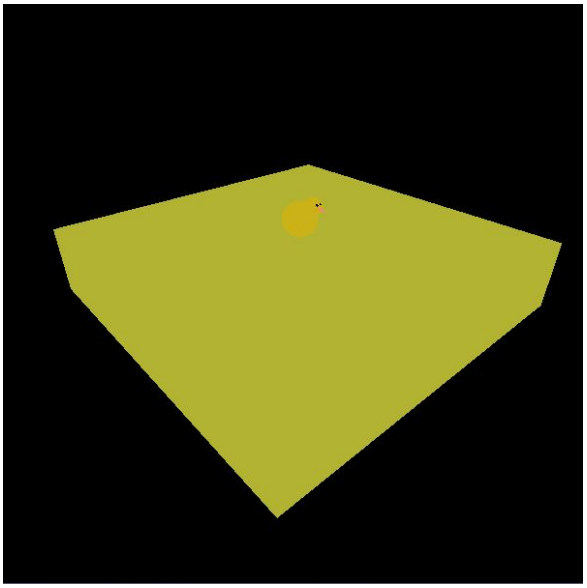


Image 9 : Un objet de la classe Goal de dimensions 5 par 1 par 5)

III-Les niveaux

Tout ça mis en place, on peut enfin créer un niveau à peu près complexe mettant en relation les différents cubes et leurs effets:

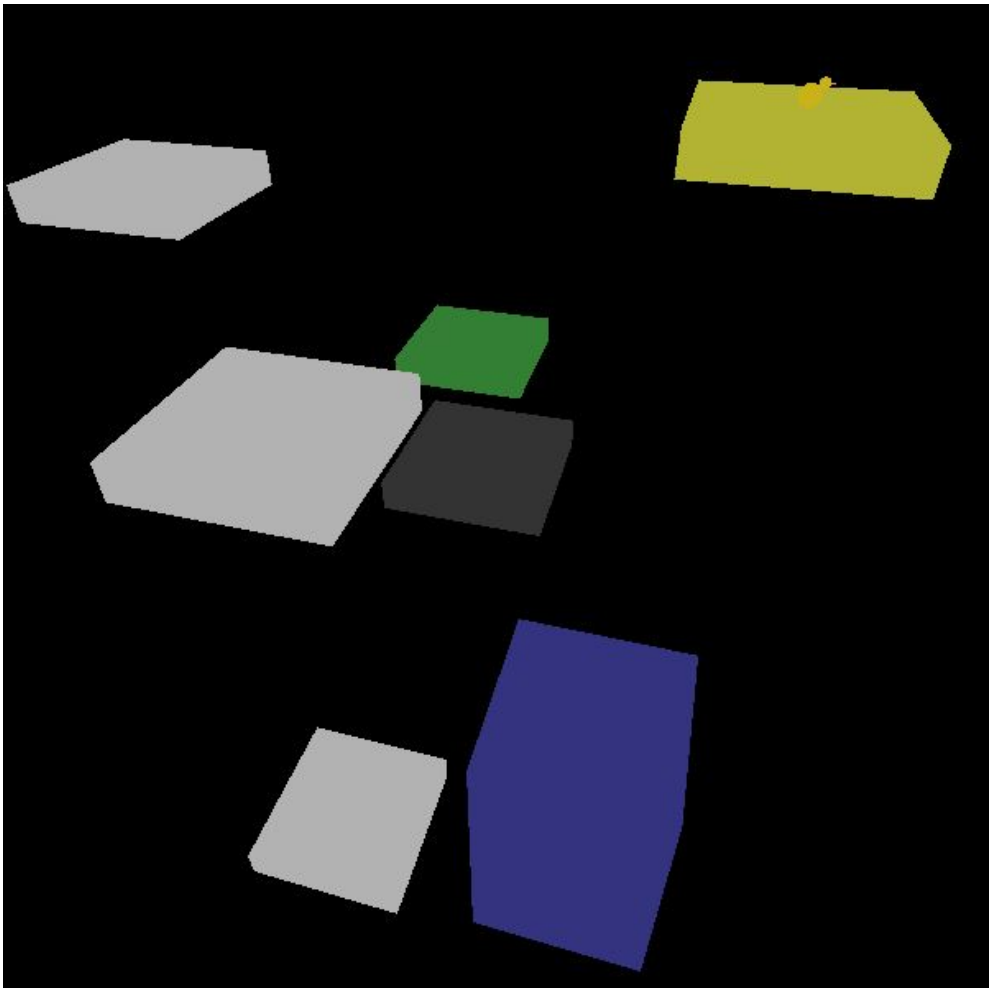


Image 10: Une idée de niveau

Un niveau étant insuffisant, nous avons créé une fonction prenant en argument un entier (le numéro du niveau) et qui déconstruit le niveau précédent, construit le niveau suivant et renvoie le joueur au point de départ. Cette fonction ne s'applique que lorsqu'on est très proche du but (vers le milieu de la plateforme jaune) et on ajoute un à l'incrément de niveau,
ainsi, on peut jouer de niveau en niveau.

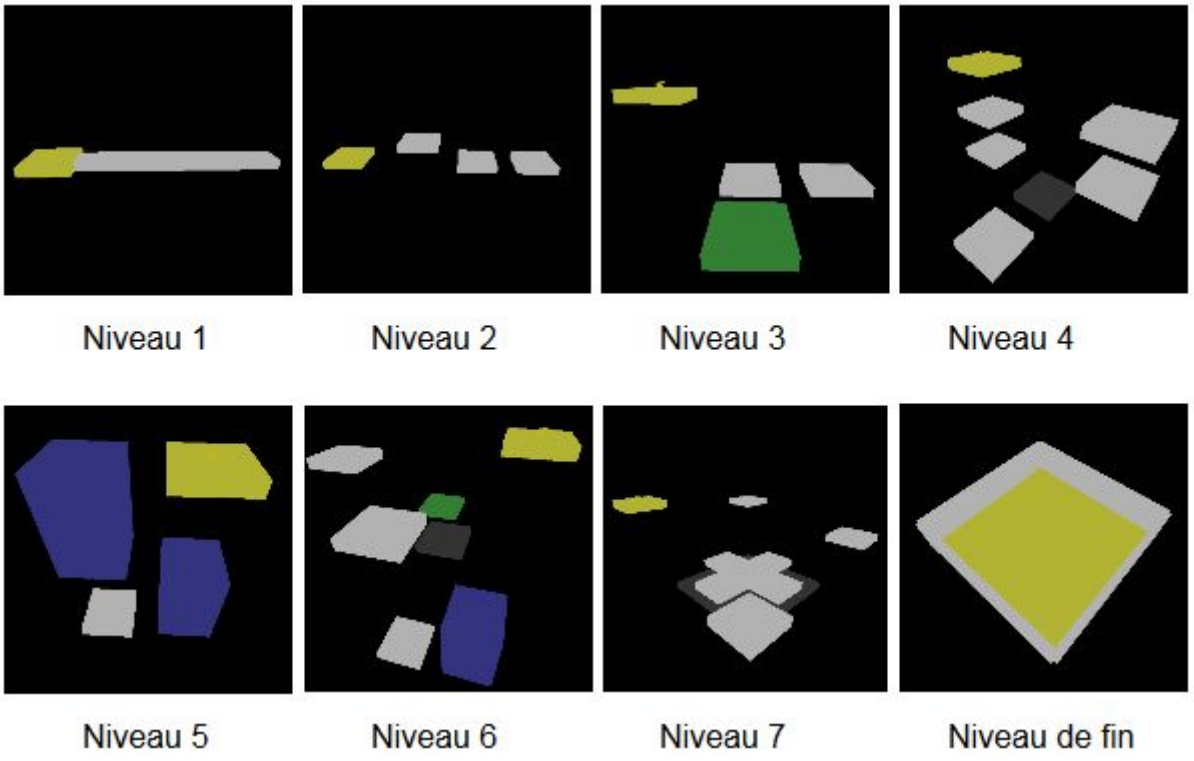


Image 11: Vue aérienne des niveaux

En terme de construction, les 5 premiers niveaux servent de tutoriel: le premier n'est qu'un couloir vers le poussin: pas de difficulté, mais on peut comprendre qu'il faut aller vers le poussin:

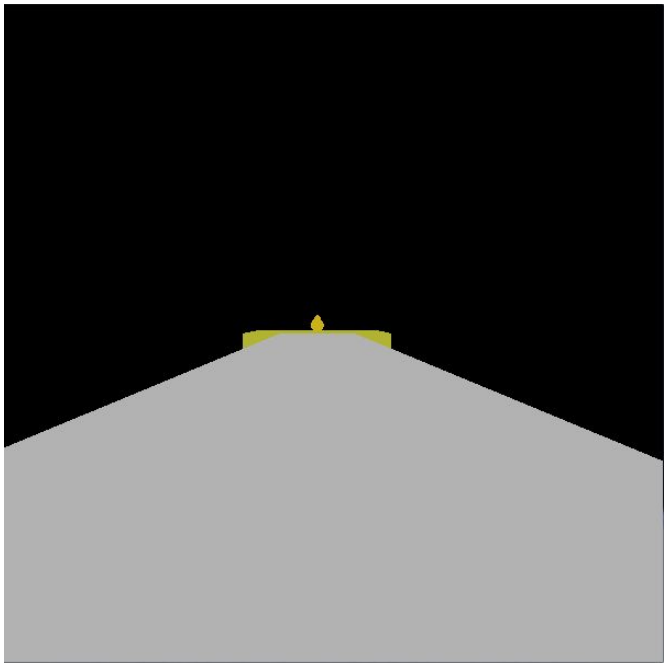


Image 12: Vue du premier niveau depuis le départ

Le deuxième niveau met en avant le saut, déclenché par la barre d'espace, permettant à la fois de surmonter le vide, mais aussi d'aller sur des plateformes en hauteur:

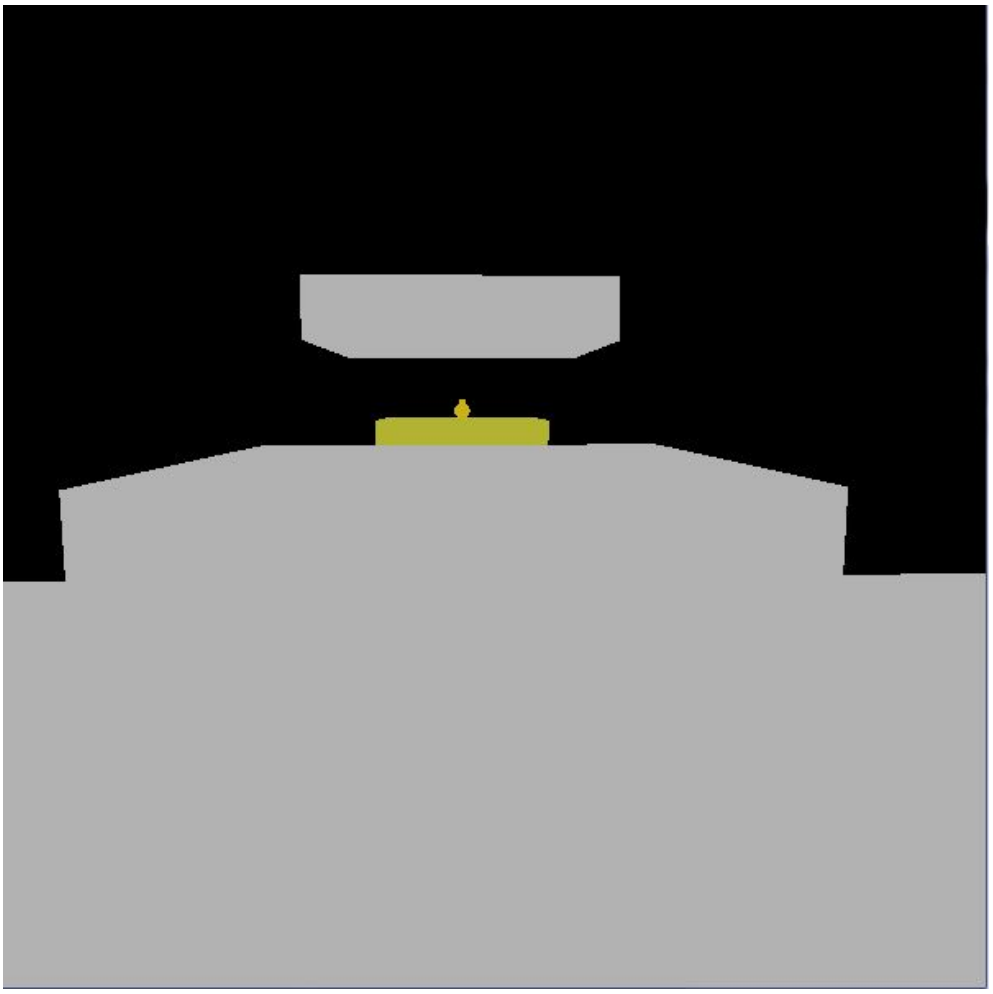


Image 12: Vue du deuxième niveau depuis le départ

Les trois suivants permettent de découvrir des cubes à effets spéciaux:

Image 13: Vue des niveaux 3,4,5 respectivement de gauche à droite

En outre, si on tombe dans le vide trop bas, on est ramené au départ.

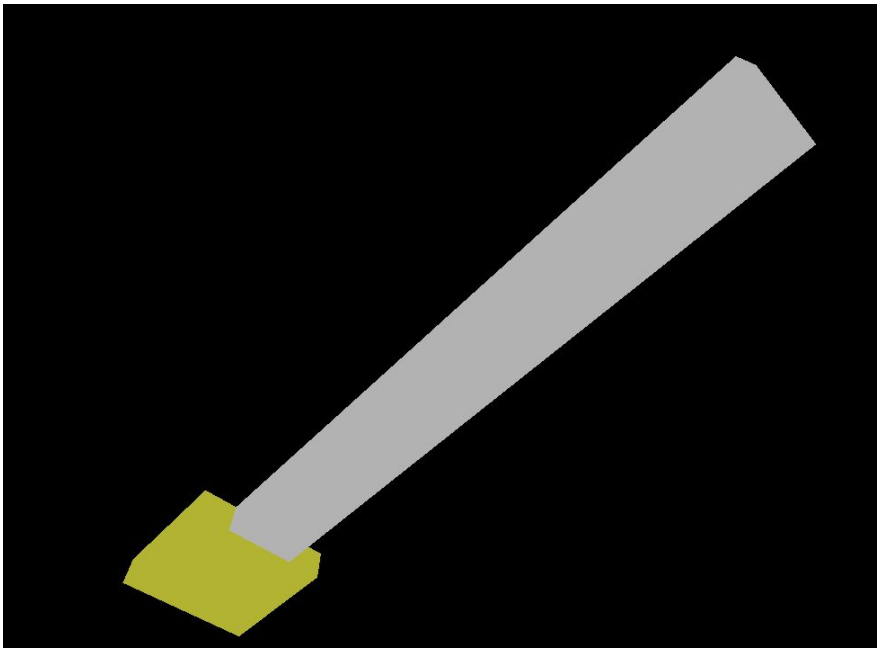


Image 14: Chute

IV-Résumé

Pour plus de clarté, nous avons jugé utile de faire un schéma détaillant un fonctionnement simplifié du script général du jeu:

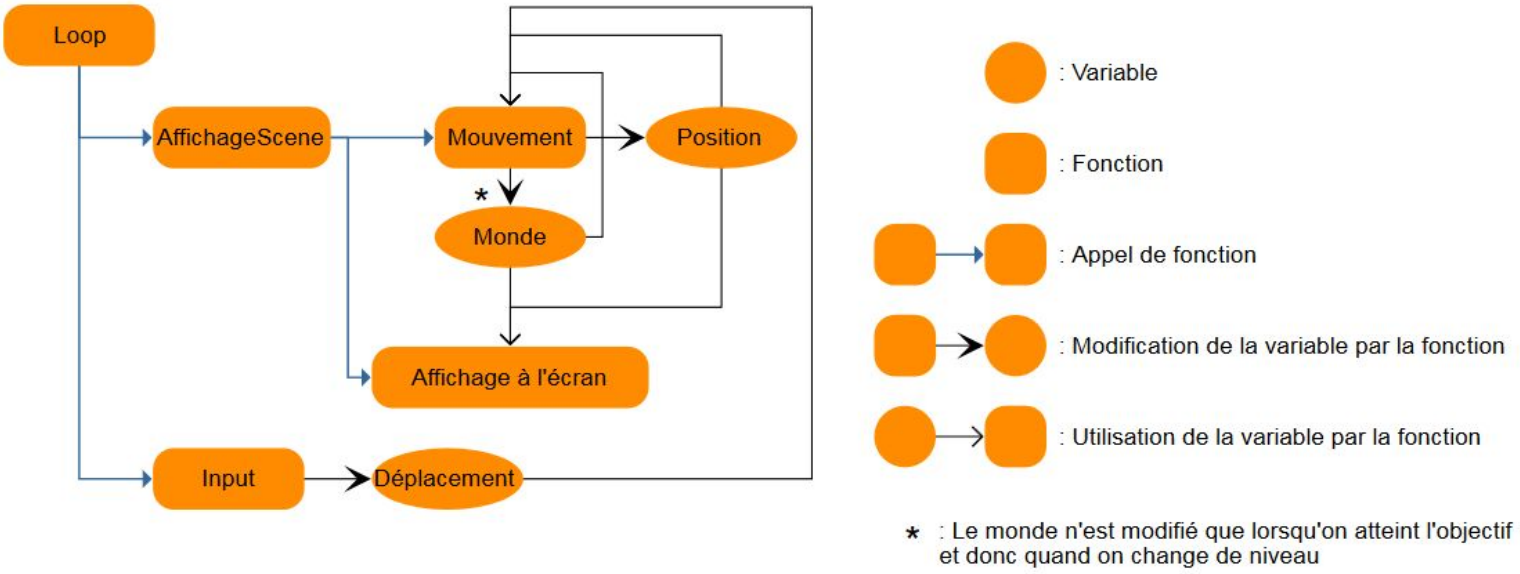


Image 14: Graphe représentatif du code effectué

V-Continuations /rapprochement du sujet d'origine

1) Ajout de texte

L'ajout de texte est la seule amélioration qu'il nous aurait été facile d'implémenter dans un temps court (une semaine aurait surement été suffisant).

Cela nous aurait permis entre autre d'afficher le niveau au début de chacun, où encore un message du type "vous avez gagné" à la fin du jeu.

2) Interactions avec le joueur

Dans l'hypothèse où nous aurions implémenté la présence de texte, nous aurions pu, mais ç'aurait été cette fois-ci plutôt long à implémenter, mettre un gameplay et proposer des choix à l'utilisateur, à l'aide d'un switch(key).

3) Environnement

Une amélioration que cette fois ci nous n'avons simplement pas priorisé dans notre liste de tâches, car elle relève uniquement du visuel et pas des mécaniques de jeu, aurait été de texturiser les cubes;

On aurait pu de la sorte créer un ciel avec un cube très large englobant les autres, créer peut-être un danger mortel tel qu'un sol en lave, qui serait un autre moyen de justifier la mort du joueur . Cela permettrait aussi peut-être de rendre les divers types de plateforme plus reconnaissables

Conclusion

Si nous devions recommencer, nous souhaiterions améliorer les points suivants:

- Nous avons perdu un temps considérable en essayant de comprendre le code de (3) afin d'en réutiliser les mécanismes. Cela a été très improductif car le code était clairement au-delà de notre portée avec notre niveau d'expérience, ce choix a été peu judicieux.
- Une idée qui nous est venue un peu tard est qu'à l'aide d'OSG, nous aurions pu faire se déplacer les plateformes, ce qui nous aurait permis de rajouter un mécanisme de jeu supplémentaire simple;

De notre point de vue, Nous avons pu travailler avec des classes abstraites, sous-classes, réfléchir aux méthodes qui aboutissaient aux mécanismes qui nous semblent si évidents dans n'importe quel jeu que nous pouvons trouver, et approfondir notre maîtrise d'objets simples sur OpenGL.

Travailler sur ce sujet nous aura été amusant et instructif, et malgré que notre résultat soit facilement améliorable, nous sommes contents du travail réalisé dans le temps imparti

Bibliographie

Openclassroom OpenGL:

- https://openclassrooms.com/fr/courses/167717-creez-des-programmes-en-3d-avec-opengl/164454-introduction-a-opengl (1)

- https://openclassrooms.com/fr/courses/966823-developpez-vos-applications-3d-avec-opengl-3-3/960359-introduction (2)

Jeu que nous avions essayé d'exploiter au départ:

- https://www.youtube.com/watch?v=aviL3HX3UEc (3)