Rapport

Atkanopong

Résumé des fonctionnalités

Éléments demandés et codés qui fonctionnent

- Un niveau jouable à deux (11 niveaux sont disponibles en tout)
- Au moins 4 types de bonus pour les briques
- Affichage de briques texturées
- Affichage du nombre de points de vie de chacun des joueurs
- Chargement du niveau depuis un fichier

Éléments demandés et codés qui ne fonctionnent pas

- La gestion des collisions fonctionne, modulo un bug qui arrive dans les circonstances suivantes : la barre d'un joueur se déplace très vite vers la balle et celle-ci heurte la barre sur un côté vertical. Parfois, au lieu de ricocher, la balle se « téléporte » de l'autre côté de la barre. Je ne suis pas parvenu à corriger ce bug pour l'instant mais je pense que le problème tient à la balle qui doit entrer en collision une seconde fois à la frame suivante et être positionnée au mauvais endroit.

Éléments demandés mais pas codés

Aucun

Éléments non demandés et codés qui fonctionnent

- La mise en pause
- La possibilité de choisir des thèmes différents
- L'organisation des niveaux en packs de niveaux (voir les commentaires dans les fichiers pour comprendre)
- Le choix de la vitesse de la balle
- Le mode de jeu contre l'ordinateur
- Le son et le fait de pouvoir régler le volume
- La possibilité d'étendre ses fichiers de niveau sur autant de lignes qu'on veut et d'ajouter des commentaires avec le caractère #
- Six types de bonus au lieu des quatre demandés
- Le menu d'accueil

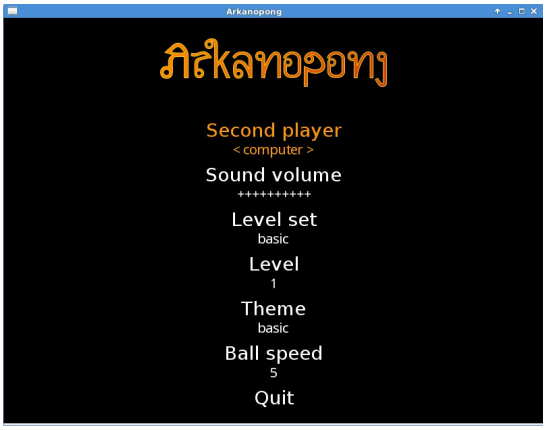
Éléments non demandés mais pas codés ou qui ne fonctionnent pas

Aucun.

Remarques

Pour lancer le jeu, se placer dans son répertoire contenant les sous-répertoires **bin**, **data**, etc, et faire ./bin/main.

Le menu



Pour utiliser le menu :

- Utilisez les touches **Haut** / **Bas** pour sélectionner l'option précédente / suivante.
- Utilisez les touches **Gauche** / **Droite** pour modifier l'option sélectionnée.
- Appuyez sur **Entrée** à tout moment pour lancer la partie, ou appuyez sur **Entrée** quand vous êtes sur **Quit** ou appuyez sur **Echap** à tout moment pour quitter le jeu.

Gestion des niveaux

Les niveaux doivent être mis dans des packs de niveau. Pour tester un niveau que vous avez créé, voici les étapes :

- Créer un répertoire dans data/level/ que vous appelerez comme vous voudrez (par exemple : 42).
- Ajouter deux lignes dans data/level/sets.list : une ligne avec le nom du groupe de niveaux créé (42 ici), et une ligne après celle-ci contenant le nombre de niveaux (1 seul si vous n'avez qu'un niveau à créer)
- Dans data/level/42/, copier votre fichier de niveau en l'appelant *1.level*. Si vous avez d'autres niveaux, nommez-les 2.level. 3.level, etc.
- Lancez le jeu, choisissez 42 comme **Level set**, choisissez *1.level* dans **Level** (il devrait être sélectionné par défaut), puis appuyez sur **Entrée** pour lancer une partie sur ce niveau.

Jouer contre l'ordinateur

Pour choisir de jouer contre l'ordinateur ou un humain :

- Lancez le jeu et sélectionnez l'option **Computer** au niveau de **Second player**. C'est l'option par défaut.

Mettre en pause

Le jeu se met automatiquement en pause dès que la fenêtre perd le focus : si vous la minimisez ou vous déplacez sur une autre fenêtre. La pause se termine dès que la fenêtre du jeu reprend le focus.

Les briques

En allant de gauche à droite, les différents types de briques sont :

- La brique de base
- La brique qui élargit le pad du joueur possédant la balle ayant touché la brique
- La brique qui rétrécit le pad du joueur adverse
- La brique qui donne un bouclier au joueur (le bouclier peut endurer quatre coups avant de disparaître)
- La brique qui libère une nouvelle balle. Si cette balle-bonus sort de l'écran, elle disparaît.
- La brique qui retire un point de vie au joueur adverse
- La brique qui ajoute un point de vie au joueur possesseur de la balle ayant touché la brique

La balle et son possesseur

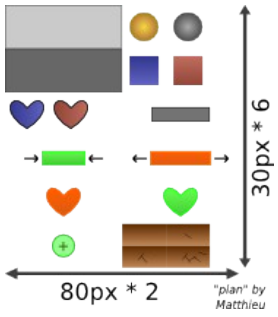
Chaque balle a un possesseur. Son possesseur est tout simplement le dernier joueur a avoir touché cette balle. Dans le cas des balles bonus, le possesseur initial de ce genre de balle est le possesseur de la balle qui a heurté la brique ayant fait apparaître cette balle.

Les thèmes

Chaque thème se matérialise sous la forme suivante :

- Une ligne portant son nom dans le fichier data/themes/themes.list
- Un fichier PNG/JPG/etc (l'extension est automatiquement détectée) contenant les graphismes de ce thème, nommé nom-du-thème.png (ou jpg ou autre)
- Un fichier PNG/JPG/etc contenant le fond d'écran de ce thème, nommé nom-du-thème-background.png (ou jpg ou autre)

Voici la structure du fichier basic.png :



Le fichier doit mesurer au moins 160 pixels de large et 180 pixels de haut et chaque élément doit être placé à une position précise (les coordonnées exactes de chaque élément sont données en pixels dans le fichier include/Theme.h).

Par exemple,

- La brique de base est positionnée en 0,0 et doit mesurer 80x30 pixels.
- La brique servant d'arrière-plan aux briques bonus doit mesurer 80x30 pixels et être positionnée en 0,30.

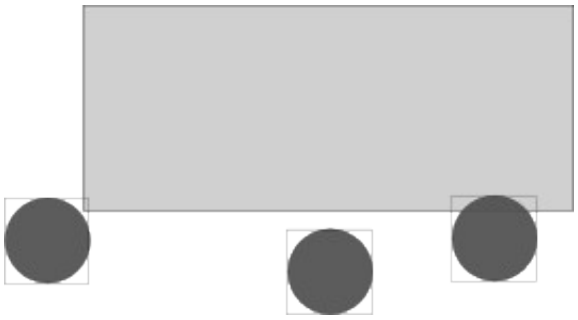
- L'image du bouclier dans ses quatre états de vie (de la gauche vers la droite, puis du haut vers le bas) est positionnée en 80,150. Chaque sous-partie de cette image doit mesurer 30x15 pixels.
- Etc.

Il est possible d'avoir un fichier de thème plus grand que les dimensions requises (comme dans l'exemple) pour ajouter des annotations et des crédits.

Problèmes rencontrés

La gestion des collisions

Pour gérer les collisions de la balle avec une brique ou une barre, j'ai choisi d'utiliser la technique des boîtes englobantes :



La collision de deux rectangles est une opération extrêmement simple et rapide à réaliser, seulement, dans notre cas, il faudra affiner un peu, dans le cas où la balle arrive sur l'un des coins d'un bloc. Sur l'image, on peut voir que la balle en bas à gauche est en collision avec le bloc si on la considère comme un rectangle, mais en la considérant comme le cercle qu'elle est, elle n'est pas en collision. Il faut donc gérer ce cas-ci.

L'algorithme complet pour détecter les collisions fonctionne comme suit :

1. Calcul des distances (verticale et horizontale) entre le centre de la balle et le centre du bloc

2. Si (la distance horizontale est supérieure à la moitié de la largeur du bloc + le rayon de la balle), ou (la distance verticale est supérieure à la moitié de la hauteur du bloc + le rayon de la balle) alors il ne peut y avoir aucune collision **(cas 1)**

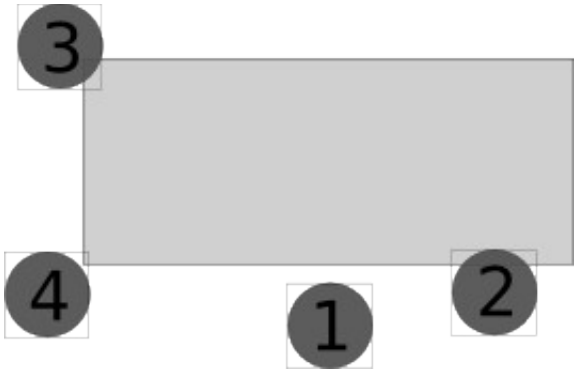
3. Si la distance horizontale est inférieure à la moitié de la largeur du bloc ou la distance verticale est inférieure à la moitié de la hauteur du bloc, alors il y a collision **(cas 2)**

4. Sinon, calculer la somme entre

- la distance du centre du cercle à un coin du rectangle en abscisses (mettre au carré)
- la distance du centre du cercle à ce même coin du rectangle en ordonnées (mettre au carré)

5. Si cette valeur est inférieure ou égale au rayon du cercle au carré, alors il y a une collision **(cas 3)**

6. Sinon, il n'y a pas de collision **(cas 4)**.



Un problème va de pair avec la gestion des collisions : ajuster la position de la balle pour qu'elle ne reste pas en partie à l'intérieur du bloc. Pour cela, j'ai choisi de coller la balle au côté par lequel celle-ci est entrée en collision avec la forme.

Afficher du texte et des nombres

Pouvoir afficher du texte était obligatoire pour avoir un menu fonctionnel. En regardant sur internet, j'ai pu constater que les gens qui veulent rendre du texte avec OpenGL utilisent souvent Freetype ou une bibliothèque tierce facilitant la tâche. Seulement, Freetype m'a paru très compliquée à utiliser et il ne m'était pas possible d'utiliser certaines bibliothèques car celles-ci ne sont pas installées sur les machines de l'université. J'ai donc pensé à certaines solutions :

- Sauver le texte dans des images que je chargerais ensuite avec SDL_image. Je n'ai pas choisi cette solution car le menu doit pouvoir afficher des noms de thèmes et de packs de niveaux, que je ne peux pas prévoir à l'avance.
- Utiliser SDL_TTF. C'est la solution que j'ai choisie.

Un autre problème s'est posé : SDL_TTF rend du texte sous forme d'image stockée sur la RAM de l'ordinateur. Je ne pouvais donc pas rendre du texte à chaque frame avec SDL_TTF, car cela demande d'envoyer l'image rendue à la carte graphique et c'est une opération lente. J'ai donc choisi de fonctionner comme suit :

Lancement du jeu
↓
Chargement des images et rendu des textes sous forme de SDL_Surface
↓
Envoi des images rendues à la carte vidéo
↓
Utilisation des textures obtenues tout le long du programme

Par contre, comment afficher des nombres (pour le numéro du niveau sélectionner) ? Je savais que je ne

pouvais pas rendre tous les nombres possibles puisqu'il y en a une infinité (un pack contenant 100000 niveaux est possible). Pour régler le problème et économiser la mémoire vidéo, j'ai rendu les chiffres de 0 à 9 puis j'ai écrit des fonctions pour afficher n'importe quel nombre en utilisant ces chiffres.

La taille de la fenêtre et la taille des niveaux

Un niveau peut être de hauteur et de largeur variables. Afin de m'adapter aux différentes possibilités, les solutions suivantes s'offraient à moi :

- Changer la taille de la fenêtre en fonction des dimensions du niveau. Désavantages : risque d'avoir une fenêtre minuscule ou deux fois plus grande que l'écran de l'ordinateur.
- Taille de fenêtre constante, mais zoomer / dézoomer avec une opération de scale après avoir dessiné le niveau, pour qu'il s'affiche en entier. Désavantages : un petit niveau va apparaître énorme (les graphismes seront plus moches), un grand niveau va apparaître très petit (plus difficiles à distinguer à l'œil nu).
- Taille de fenêtre constante, mais utiliser une caméra pour pouvoir voir les différentes parties du niveau. Désavantages : que doit suivre la caméra ? Il y a plusieurs balles et plusieurs joueurs et s'agissant plus ou moins d'un jeu de vitesse, cette solution était trop encombrante.
- Taille de fenêtre constante, avec une limite à la dimension d'un niveau. C'est la solution que j'ai retenue, et que je vais détailler par la suite.

J'ai choisi des dimensions de 800x600 pour la fenêtre, car je suis certain que quelque soit l'ordinateur sur lequel le jeu est lancé, la définition de l'écran sera a priori suffisante pour tout afficher.

Si un niveau est plus petit que la taille de la fenêtre, le reste du niveau est rempli (lors de son chargement) de vides.

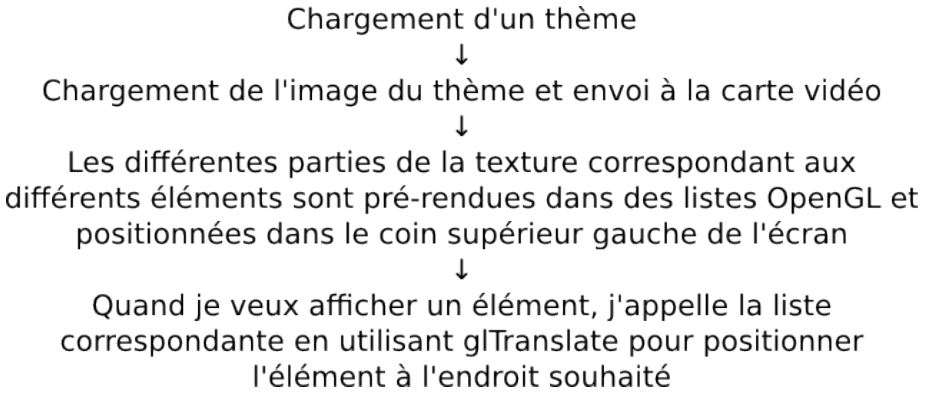
Si un niveau est trop grand, il n'est pas chargé.

Ce n'est pas la solution la plus flexible mais c'est la plus simple à mettre en œuvre et le rapport avantages/désavantages est satisfaisant, sachant que sur 800x600 pixels, il y a assez de place pour 10x15 briques.

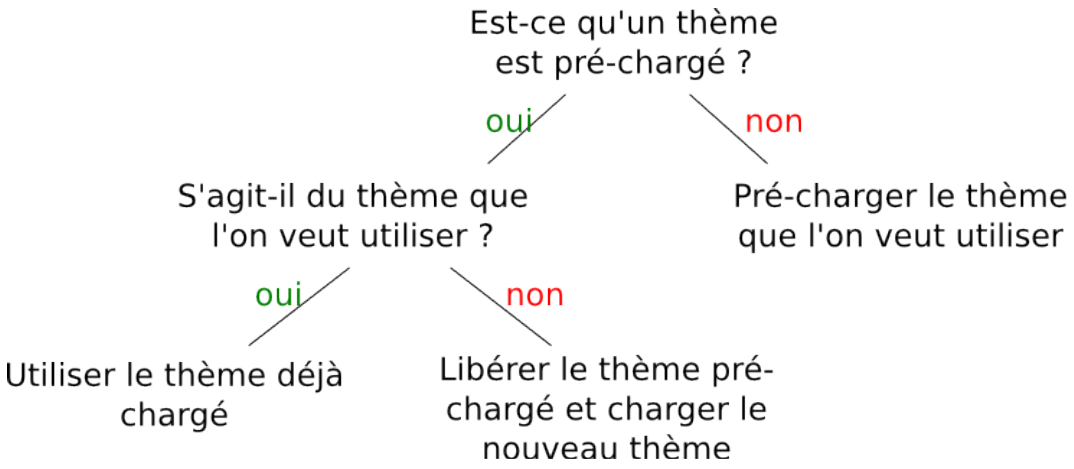
La gestion des thèmes

Comment sauvegarder les différents composants graphiques de mon jeu sans me retrouver avec un certain nombre de fichiers à gérer ?

J'ai choisi de mettre tous ces composants sur une seule image (sauf le fond d'écran).



Par ailleurs, afin d'économiser la mémoire vidéo et d'accélérer le démarrage d'une partie, j'utilise l'algorithme suivant lorsque le joueur lance une nouvelle partie :



Structures de données particulières

Au lieu de faire un listing de toutes mes structures de données, j'ai choisi de ne parler que de celles qui méritent commentaire :

Timer.h

Ce fichier contient une structure de données nommée Timer et contenant deux variables membres, **start** et **ms**.

En utilisant les fonctions listées dans le même fichier, elle permet de gérer un timer en lui donnant un certain framerate (par exemple, on peut créer un timer qui doit « ticker » 60 fois par seconde) et de faire dormir le programme jusqu'au prochain tick. Cette factorisation du code m'a simplifié la gestion du framerate dans le menu et en cours de jeu.

La variable start est mise à jour à chaque tick pour correspondre au temps à ce moment précis, quand à la variable ms, elle indique le nombre de microsecondes entre chaque tick.

Ball.h

Ce fichier contient une structure Ball contenant

- le numéro (**last_player**) du dernier joueur à avoir touché la balle (0 pour le joueur 1, 1 pour le joueur 2)
- le **type** de balle (normale ou bonus)
- sa position en **x** et **y** par rapport au coin supérieur gauche de l'écran et dans le système de coordonnées de la fenêtre
- sa vitesse horizontale **x_speed** et verticale **y_speed**

On trouve aussi dans ce fichier un type opaque nommé **Ball_list**. Le fichier contient de nombreuses fonctions utilisant ce type, qui permet de gérer une liste de balles de longueur potentiellement infinie.

Level.h

Les seules variables membres du type de structure Level sont :

- La largeur (**width**) et sa hauteur (**height**). À noter que ces variables indiquent un nombre de briques et pas un nombre de pixels
- Ses blocs (**blocks**) sous la forme d'un tableau à deux dimensions (la première dimension est leur index vertical).

Paddle.h

Le type de structure Paddle contient les informations nécessaires pour gérer un joueur :

- sa position horizontale **x** et sa position verticale **y** (qui ne change jamais)
- sa largeur et sa hauteur (**width** et **height**)
- sa vitesse (**speed**)
- son numéro (0 pour le joueur 1, 1 pour le joueur 2)
- l'état de son bouclier (**shield**) codé par une énumération (THEME_SHIELD_100, THEME_SHIELD_75, _50, _25 puis __THEME_SHIELD_LAST pour pas de bouclier).

Settings.h

La dernière structure de données dont il est intéressant de parler est le type Settings :

- un booléen **first_run**. S'il vaut **false**, le menu va restaurer les options d'une précédente partie au lieu d'afficher les options par défaut
- le booléen **play_with_computer**
- le **volume** sonore
- le nom d'un pack de niveaux (**level_set**)
- le numéro d'un niveau (**level**)
- la vitesse de départ des balles (**ball_speed**)
- le nom d'un thème (**theme**)

Cette structure est remplie par la boucle utilisée pour rendre et gérer le menu et est utilisée pour retenir les informations sur la partie que le joueur souhaite lancer.

Améliorations possibles

- Corriger le bug énoncé vers le début
- J'ai réalisé vers la fin que je n'avais pas fait de séparation claire entre l'affichage et la physique. Par exemple, j'utilise THEME_BALL_WIDTH pour détecter les collisions avec la balle, or cette variable indique la largeur en pixel de l'image de la balle. Ce qui fait que tout le code est fortement dépendant du code de Theme.h et qu'il me faudrait procéder à des changements

substantiels si, par exemple, je souhaitais ajouter un type de bonus qui agrandisse ou rétrecisse une balle.

Crédits

La police d'écriture utilisée pour le titre du jeu dans le menu est AW_Siam, désignée par Andreas Weygandt (http://www.weygandt.de/aw_siam/).