RAPPORT PROJET PROGRAMMATION WEB AVANCEE

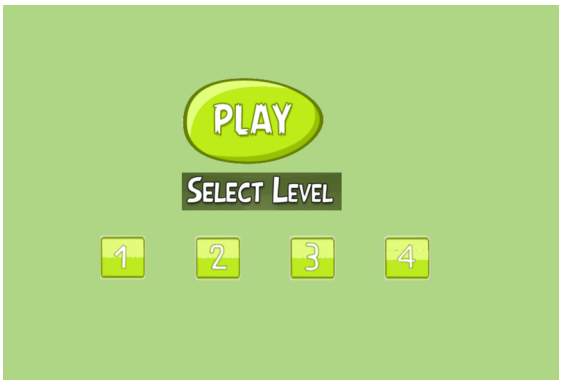
Mini Angry Birds

Camille Bignet *Majid Akharaz*

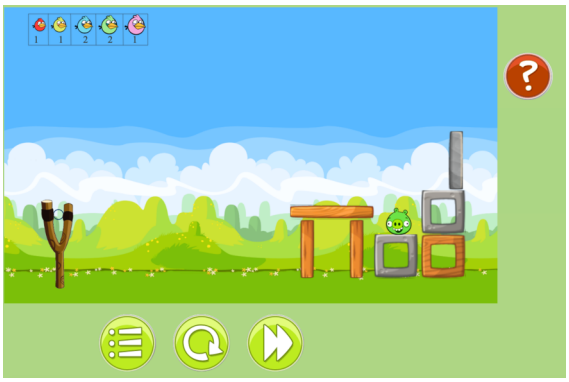
Le menu :

Nous avons codé un menu nous permettant de choisir le niveau que nous voulons jouer ou bien tout simplement cliquer sur « Play » nous faisant commencer du niveau 1.
Ensuite lorsque le jeu démarre, si nous gagnons un niveau, un message « Level cleared » apparait et nous pouvons cliquer dessus pour aller automatiquement au niveau suivant.
En revanche lorsque nous perdons le niveau, un message « Level Failed », nous devons donc le refaire.
Nous pouvons avoir accès aux règles du jeu et aux instructions en appuyant sur le bouton rouge avec « ? » en haut à droite de l'écran et recliquer sur ce bouton pour faire disparaitre le message.
Trois autres boutons sont disponibles sous l'écran ; le premier bouton permet de retourner sur le menu principal, le second permet de recommencer à zéro le niveau et enfin le dernier permet de passer directement au prochain niveau.

Le but du jeu est simple, il suffit de tirer sur les cibles avec les projectiles, afin de leur retirer des points de vie et les éliminer.
Il faut également faire attention de ne pas pousser la cible hors du cadre autrement le niveau sera automatiquement perdu.
De plus, le nombre de projectile est limité et varie en fonction des niveaux.

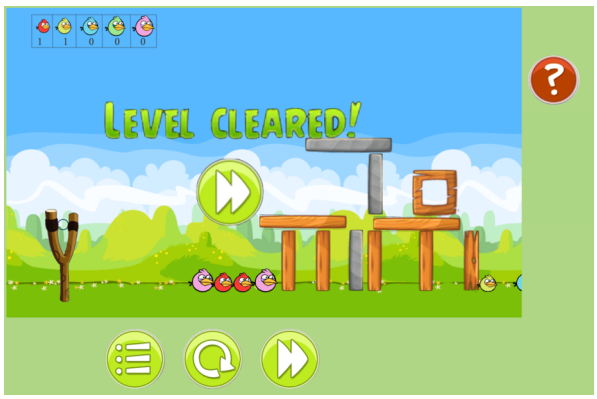


Le menu avec le bouton Play et les boutons des différents niveaux.



Ici nous avons le niveau 1 avec les trois boutons « menu », « replay » et « suivant » en dessous de l'écran de jeu. Ainsi que le bouton « help » sur la droite

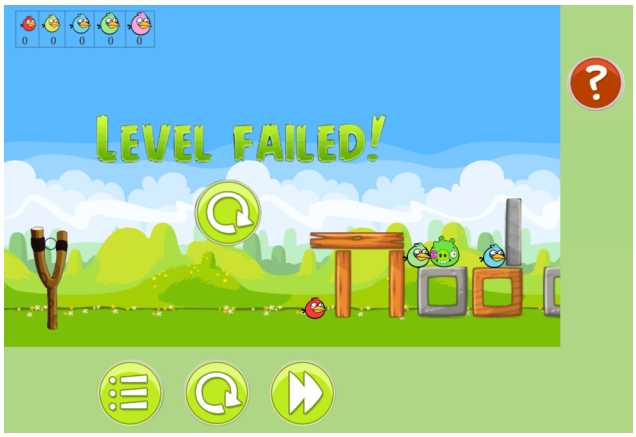
Les instructions qui apparaissent lorsque l'on clique sur le bouton « help ».



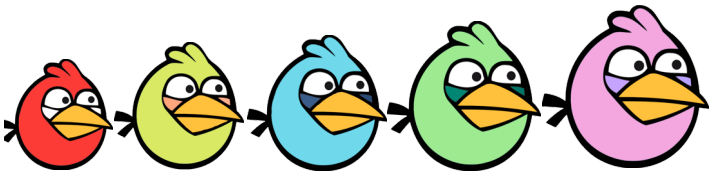
Lorsque l'on gagne un niveau

Les niveaux sont assez compliqués mais ils sont tous faisables.

Pour ce qui est des différents projectiles, nous avons en haut de l'écran, un tableau servant à nous indiquer combien de projectiles il nous reste et de quels types sont-ils.



Lorsque l'on perd un niveau



<u>Les projectiles :</u>

En effet, il existe différents types de projectiles variant suivant leur taille et surtout leur masse. Ce qui leur permet d'avoir une trajectoire/vitesse différente ainsi qu'un nombre de dégât occasionné différent.
Le joueur doit donc prendre en compte le type du projectile afin de mieux viser.

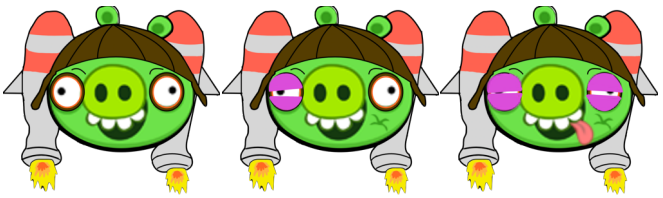
<u>Les cibles :</u>

Il existe également plusieurs types de cibles. Les cibles simples, les cibles volantes et les cibles glacées.

Les cibles simples ainsi que les cibles glacées réagissent comme les objets, elles rebondissent, subissent la gravité, les frictions, les collisions et perdent des points de vie lorsqu'un projectile les touches. Les points de vie perdus sont calculés en fonction des masses des objets et de la vitesse du projectile.

Les cibles glacées servent dans les niveaux enneigés, nous avions prévue de leur donner une particularité supplémentaire mais malheureusement nous manquons de temps.

Enfin les cibles volantes sont des cibles qui subissent les dégâts mais ne subissent pas la gravité. Elles ont une trajectoire prédéfinie (verticale) (Nous pourrions facilement rajouter d'autres trajectoire). Ainsi, lorsqu'un projectile entre en collision avec une cible



volante, cette dernière perd des points de vie de la même façon qu'une cible normale mais sa trajectoire n'est pas déviée.

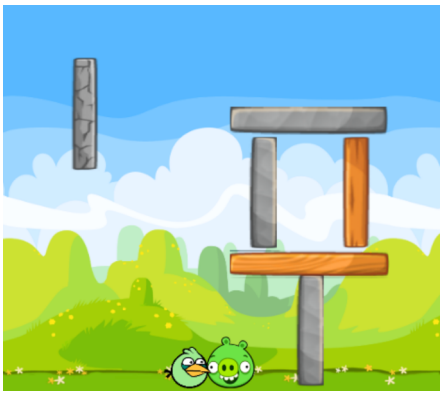
Les objets :

Les objets protègent les cibles et rendent l'accès à ces dernières plus difficile.

Il existe deux types d'objets, les objets simples et les objets volants.
Comme les cibles volantes, les objets volants suivent une trajectoire définie (également verticale) et perdent des points de vie au contact d'un projectile.

Nous avons ici, les différents objets en bois et en pierre ainsi que la version « abimé » d'un objet en bois.

Tous les objets ont des points de vie, et chaque objet possède deux aspects(image) différent(e)s afin de pouvoir signaler lorsque l'on passe sous un certain seuil de point de vie et donc que l'objet cédera bientôt.
En effet les objets partent avec un aspect de base et lorsqu'ils perdent trop de point de vie, ils ont ensuite un aspect fissuré puis finissent par disparaitre après la perte des derniers points de vie.
De plus les objets ont des matières différentes.



Dans cette image nous avons un objet Volant qui fait des aller retours de haut en bas, en version « abimé » d'un objet en pierre

Les objets en pierre casseront moins rapidement que les objets en bois car ils possèdent plus de point de vie à leur création.

Nous avons également rajouté des objets en glace et en neige, les objets en neige étant les plus fragiles.

On peut voir sur cette photo les objets en neige et en glace (Niveau 4)

Nous avons indexé toutes nos images d'objet dans un tableau ainsi que leur dimension ; de façon à ce que lorsque l'on récupère nos objets dans le fichier JSON nous avons juste à donner le type d'objet par exemple « boisVertical »

Implémentation :

Nous nous sommes inspiré du TP 2 pour commencer et pour gérer les collisions ainsi que l'article du lien que vous nous avez donné en bas du TP 2 (hamaluik.com) qui explique une technique plus élaborée de détection de collision afin que les collisions soient détectées même lorsque la vitesse des objets est trop élevée. Cette partie du code est donc **fortement** inspirée du TP 2 et de cet article.

Pour le lancement des projectiles nous nous sommes inspirés de ce tutoriel : « 2D Projectile Motion using Canvas and JS » afin que le joueur puisse lancer les projectiles avec une force variable avec la souris.
Il suffit de cliquer sur le petit cercle au centre des deux branches du lance pierre pour y placer un oiseau (lorsqu'il reste des oiseaux à tirer autrement la partie est perdue).

Pour charger nos niveaux nous utilisons un serveur et un fichier JSON ou nous répertorions tous les objets, les cibles, les projectiles disponibles, les fond d'écran en fonction des niveaux.
Chaque objet possède des coordonnées x et y, une masse, un t (afin de savoir s'il s'agit d'un objet en bois, pierre…), une image, et un type (pour savoir s'il s'agit d'un objet « normal » ou « volant ».
Ces objets sont rangés dans un tableau « objets ».

Les cibles quant à elles possèdent également des coordonnées x et y, une masse et un type (afin de savoir s'il s'agit d'une cible normale, glacée ou volante).

Remarque : Nous aurions pu encore simplifier en enlevant les variables de masse et « t » des objets, en les stockant plutôt dans notre index d'images.

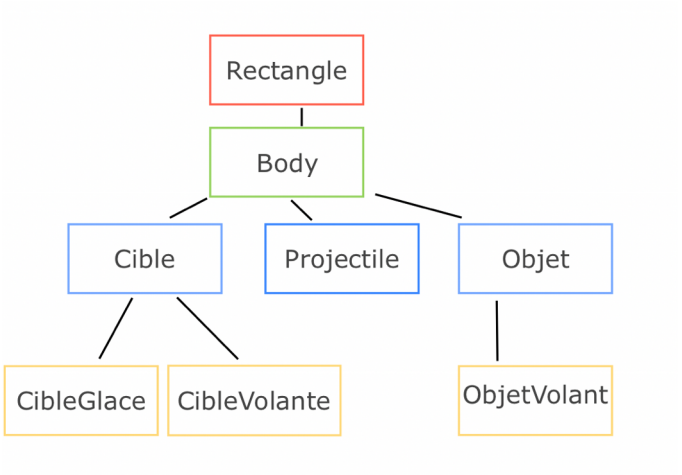
Pour l'affichage des informations de débuggage concernant les 60 images par secondes, nous ne les voyons pas dans les screens précédents ainsi que dans la vidéo (car nous avions oublié de l'afficher) mais ce fut rectifié par la suite et c'est à présent dans le code.

Organisation du code et des fichiers :

Nous avons les fichiers suivants :

- **index.HTML** : Page HTML
- **index.js** : fichier qui permet de gérer la partie « menu », faire apparaitre ou disparaitre le canvas ou les div, lancer le jeu lorsque l'on appuie sur un niveau ou sur le bouton « play ».

- **game.js** : Contient la fonction qui initialise la partie. Les fonctions permettant de charger les objets du fichier JSON, La fonction Update qu'on appelle avec setInterval toutes 60 millisecondes.
- **lanceur.js** : Ce fichier permet de gérer toute la partie du lance pierre, et récupérer les événement de la souris (les cliques, les mouvements…) afin de récupérer la puissance et l'angle de tir à chaque lancer de projectile.
- **constants.js** : Dans ce fichier nous y stockons la constante de gravité ainsi que les fonctions permettant de calculer des distances euclidiennes et des angles entre deux points. De plus une fonction permet de dessiner l'image d'arrière-plan.
- **vecteur.js** : Permet de créer les vecteurs et possède toutes les fonctions utiles de calculs avec les vecteurs.
- **rectangle.js** : Permet de créer des rectangles qui représenteront les objets. Et également d'implémenter les fonctions mDiff et hasOrigin utiles à la détection de collision.



    - **body.js** : hérite de la classe Rectangle, elle permet de gérer les collisions entre les « body » qui sont soit des projectiles, les objets ou des cibles. Cette classe permet également de calculer les dommages entre les objets (seule les projectiles occasionnent des dommages) et seule les objets et les cibles perdent des points de vie.
        - **cible.js** : Hérite de body et permet de créer des cibles « normales », de gérer l'aspect de la cible en fonction de ses points de vie (comme dit plus haut les points de vie d'une cible passent par trois stades avant de disparaitre lorsqu'elles n'ont plus de point de vie).
          Possède également une fonction permettant de dessiner les cibles, calculer leur trajectoire, vérifier qu'elles sont toujours dans le canvas (autrement la partie sera perdue)
            - **cibleGlace.js** : Hérite de Cible et permet de créer un autre type de cible : les cibles glacées qu'on utilise lors des niveaux avec de la glace.
            - **cibleVolante.js** : Hérite de Cible et permet de créer un autre type de cible : les cibles Volantes qui possèdent une fonction de calcul de trajectoire complétement différente puisqu'elle à une trajectoire fixée qui ne dévie pas même après une collision.
        - **objet.js** : Comme la classe Cible, la classe Objet hérite de la classe Body. Elle permet de créer des objets « normaux », les dessiner, calculer leur trajectoire (calculer leur position en fonction de leur vitesse, les rebonds, la friction), gérer leur aspect.
            - **objetVolant.js** : Hérite de la classe Objet et permet de créer un autre type d'objet : les Objets Volantes qui possèdent une

fonction de calcul de trajectoire différente avec une trajectoire fixe qui ne dévie pas même après une collision.
        - **projectile.js** : Hérite de body, permet de créer les projectiles, de gérer leur trajectoire et leur lancement.
- **image.js** : Permet de répertorier les images utilisées pour les différents objets, projectiles, et fond d'écran.
- **style.css** : Permet de gérer les positions des différentes div de la page HTML
- **data.JSON** : Permet de répertorier tous les niveaux avec leur objets correspondant.

Répartition des tâches :

Pour la répartition des tâches, nous avons principalement réfléchis et codé ensemble ; les classes Vecteur, Rectangles, et Body ont principalement été récupérées du TP2.
Camille s'est occupée des fichiers : Objet.js, ObjetVolant.js, index.js, image.js
Majid s'est occupé des fichiers : Cible.js, CibleVolante.js, CibleGlace.js, style.css
Pour le fichier data.JSON nous avons décidé ensemble de sa structure, et les objets que nous voulions mettre pour que les niveaux soient réalisables sans être trop faciles non plus, puis Majid s'est occupé de la rédaction du fichier.
Les fichiers projectile.js et lanceur.js ont été fait ensemble (à l'aide du tuto cité précédemment)
Le fichier game.js a également été réalisé ensemble notamment la partie sur le JSON qui nous a posé le plus de problème. Camille s'est occupée de la rédaction des fonctions de création des éléments du jeu à partir du fichier JSON.

De manière générale, nous nous sommes souvent concerté et entre aidé dans la réalisation des fichiers.

Pour les photos utilisées, nous les avons récupérées sur google image, et le plus souvent modifié légèrement comme le coloriage de nos oiseaux, nos cibles volantes que nous avons dessinées nous-mêmes. De plus nous avons également réalisé des trois aspects différents des cibles glacée et volantes, en leur dessinant des blessures. (A l'aide du logiciel Aperçu).