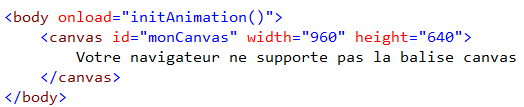
|  |
| --- |
| Laboratoire 6: La programmation d’un jeu : le vortex |

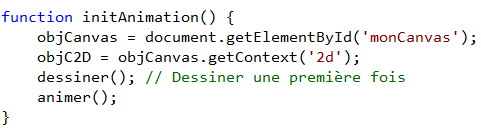
Théorie

1. Le jeu

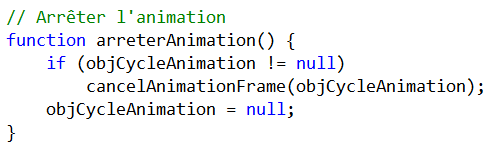
Dans ce laboratoire, nous allons programmer un petit jeu style « casse-brique ».

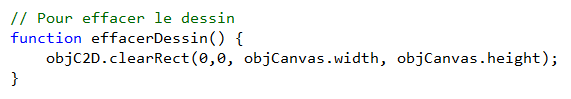
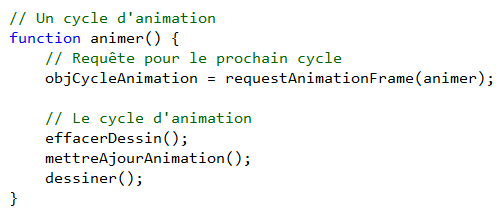
Ouvrez la page Web **1-SqueletteJeu.htm.** Cette page Web est une page blanche. Elle contient le squelette du jeu.

Le corps de cette page Web n’a qu’une seule balise. C’est à l’intérieur du canevas que le jeu va être programmé.

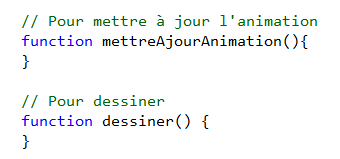


La fonction **initAnimation()** sera utilisée pour initialiser tous les objets qu’il y aura dans ce jeu et pour démarrer le jeu.

La fonction **animer()** programme le cycle d’animation et la fonction **arreterAnimation()** arrête ce cycle.



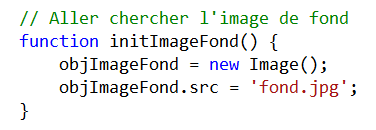
La fonction **effacerDessin** efface tout ce qui a été dessiné dans le canevas.

Les fonctions **mettreAJourAnimation()** et **dessiner()** servent respectivement à mettre à jour les objets animés qu’il y aura dans ce jeu et pour les dessiner. Pour l’instant, il n’y en a pas.

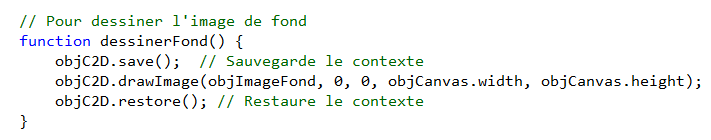
Actuellement, ce programme ne fait pas grand-chose. Ce programme efface un canevas vide à chaque 1/60 de seconde.

2. Le dessin du fond

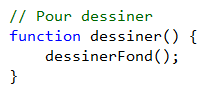
Ouvrez la page Web **2-DessinDuFond.htm.** Dans ce jeu, il y a une image qui couvre la totalité du canevas.



Tout d’abord, dans la fonction **initImageFond**, nous allons chercher l’image de fond (le fichier **fond.jpg**). Ici, il n’est pas nécessaire d’attendre que l’image soit complètement chargée, car le canevas s’efface et se redessine 60 fois par seconde. Il est évident, qu’à un certain moment donné, l’image de fond sera complètement dessinée.

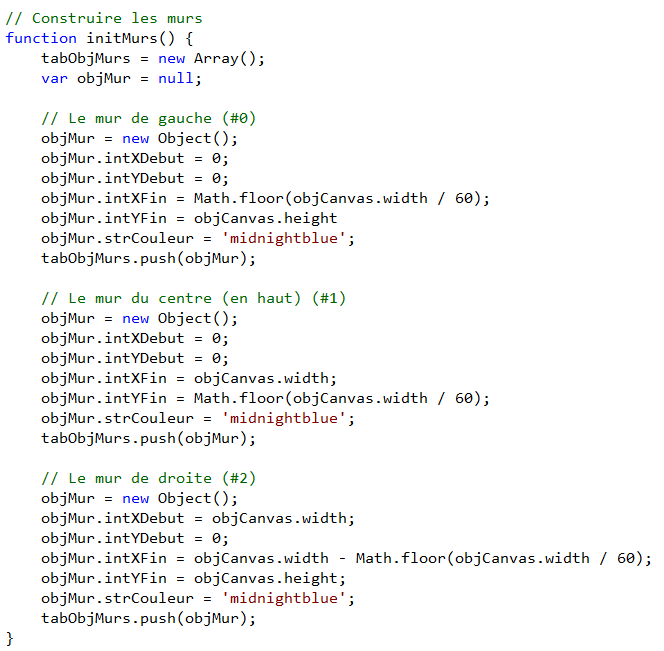
Puis, dans la fonction **dessinerFond**, nous la dessinons en prenant soin de sauvegarder le contexte au début de la fonction et de le restaurer à la fin (mais ici ce n’est pas nécesssaire).

La fonction **dessinerFond** est appelée dans la fonction **dessiner**.



Encore une fois, ce programme ne fait pas grand-chose. Ce programme efface puis redessine la même image à chaque 1/60 de seconde.

3. Le dessin des murs



Ouvrez la page Web **3-DessinDesMurs.htm.** Dans ce jeu, il y a des murs (qui vont être utilisés plus tard pour faire rebondir une balle).

La fonction **initMurs** sert à paramétrer les murs.

En fait, il y a trois (3) murs : le mur de gauche (le mur #0), le mur du haut (le mur #1) et le mur de droite (le mur #2).

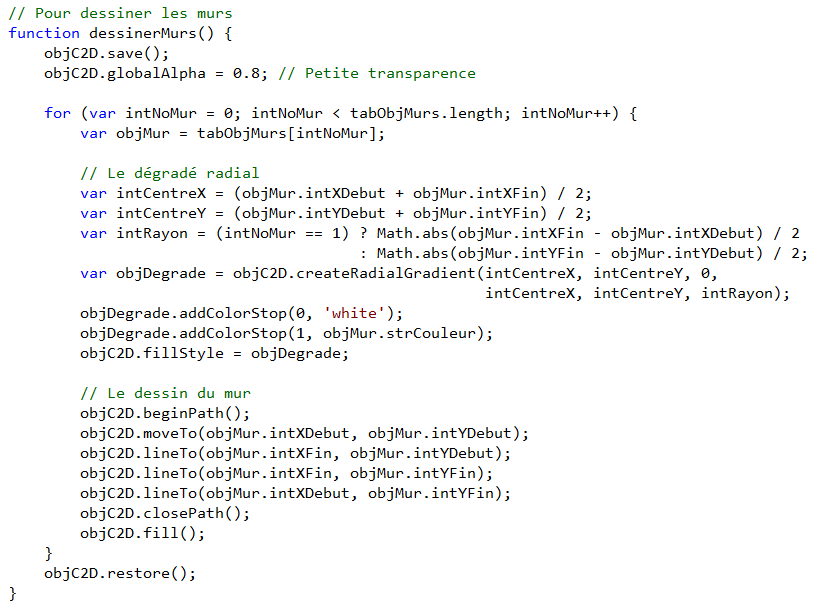
Chacun de ces murs est un objet placé dans un tableau (dans la variable **tabObjMurs**).

Pour chacun des murs, les propriétés **.intXDebut**, **.intYDebut, .intXFin, .intYFin** et **.strCouleur** contiennent respectivement la position en X au début du mur, la position en **Y** au début du mur, la position en X à la fin du mur, la position en **Y** à la fin du mur et la couleur du mur.

Observez que le mur de gauche débute à l’extrémité gauche du canevas et que le mur de droite débute à l’extrémité droite du canevas. Cela fait en sorte que la fin du mur de gauche et que la fin du mur de droite sont situées à l’intérieur du canevas.

|  |  |  |
| --- | --- | --- |
| **Mur #0** |  | **Mur #2** |

**.intXDebut .intXFin .intXFin .intXDebut**

Dans la fonction **dessinerMurs**, chacun des murs se dessine et le style appliqué est un dégradé radial.

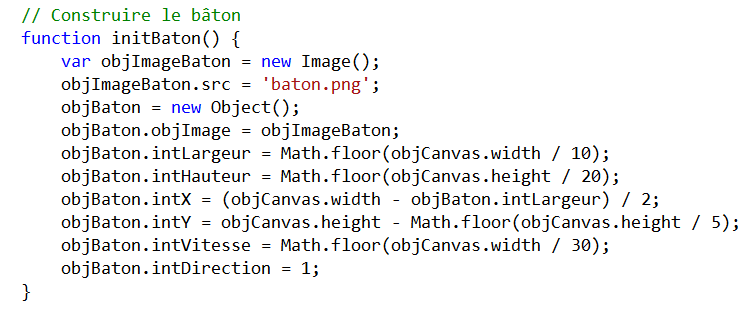
Observez qu’il y a une légère transparence (20% de transparence) dans le dessin du mur. Cela fait en sorte que nous apercevons légèrement l’image de fond en arrière des murs lorsque les murs se dessinent.

Prenez le temps de bien comprendre la manière que les murs ont été dessinés.

4. Le dessin et l’animation du bâton

Ouvrez la page Web **4-DessinEtAnimationDuBaton.htm.** Dans ce jeu, il y a un bâton (qui va être utilisé plus tard pour frapper une balle).

Ici, le bâton a la forme d’une soucoupe volante.

La fonction **initBaton** sert à paramétrer le bâton.

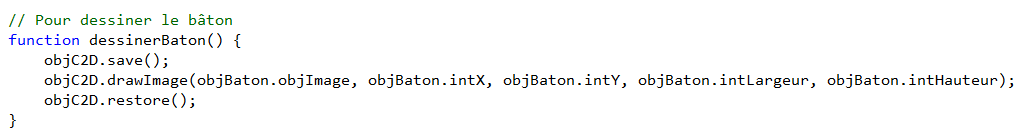
La propriété **.objImage** contient l’image du bâton (ici **baton.png**). À cause de l’animation, on ne doit pas attendre que l’image soit complètement chargée.

Les propriétés **.intX, .intY, .intLargeur** et **.intHauteur** contiennent respectivement la position et la taille du bâton. Au point de départ, le bâton est centré horizontalement et est situé au 4/5 du canevas verticalement.

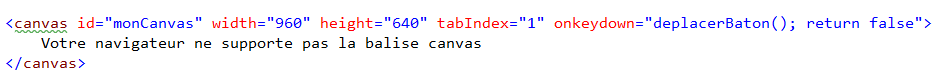
La propriété **.intVitesse** contient la vitesse de déplacement du bâton, c’est-à-dire le nombre de pixels que le bâton va se déplacer lorsque ce dernier va être déplacé par l’utilisateur.

La propriété **.intDirection** contient la direction de déplacement du bâton. 1 signifie qu’il se déplace vers la droite et -1 signifie qu’il se déplace vers la gauche.

La fonction **dessinerBaton** sert à dessiner l’image du bâton.



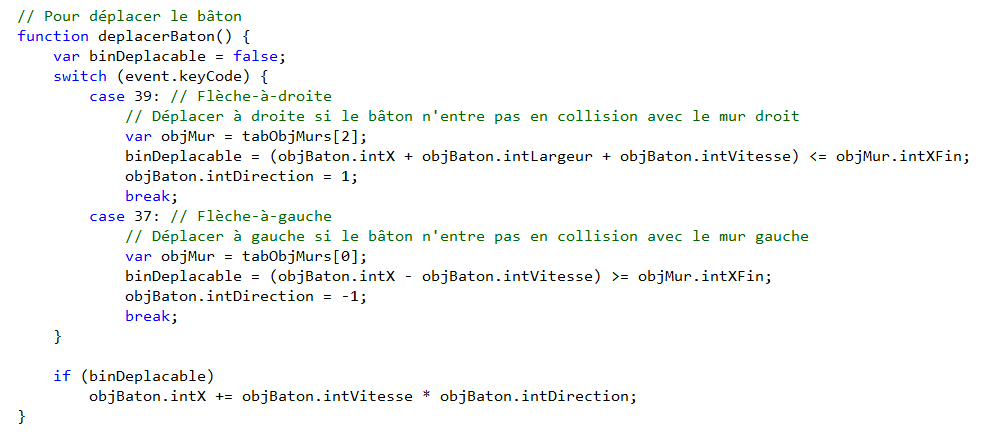
Le cœur de l’animation du bâton est la fonction **deplacerBaton** qui est appelé à chaque fois que l’utilisateur relâche une touche du clavier. L’instruction **return false** est importante. Elle annule le comportement par défaut des touches du clavier contrôlé par le navigateur.

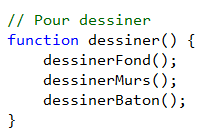


Lorsque l’utilisateur relâche la touche « **flèche-à-droite** », nous vérifions si le bâton n’entre pas en collision avec le mur droit. Si c’est le cas, le bâton est déplaçable.

Lorsque l’utilisateur relâche la touche « **flèche-à-gauche** », nous vérifions si le bâton n’entre pas en collision avec le mur gauche. Si c’est le cas, le bâton est déplaçable.

Si le bâton est déplaçable (il ne l’est pas si le bâton touche le mur), nous mettons à jour sa position en X.

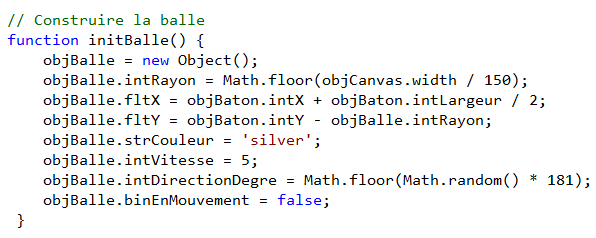


Il est important de prendre note qu’il n’est pas nécessaire de redessiner le bâton. Tout ce que nous devons faire, c’est de mettre à jour la position en X du bâton. Le cycle d’animation va faire en sorte que le canevas va s’effacer et se redessiner automatiquement à chaque 1/60 de seconde.

Testez le déplacement du bâton à l’aide des touches « **flèche-à-droite** » et « **flèche-à-gauche** ».

5. Le dessin et l’animation de la balle

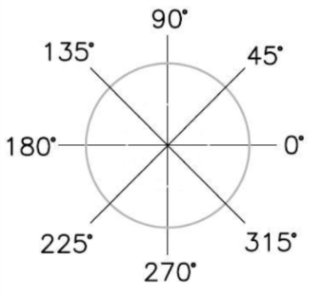
Ouvrez la page Web **5-DessinEtAnimationDeLaBalle.htm.** Dans ce jeu, il y a une balle. La balle est certainement l’objet le plus complexe à programmer.



La fonction **initBalle** sert à paramétrer la balle.

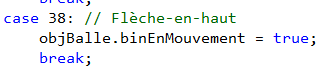


Les propriétés **.intRayon**, **.fltX**, **.fltY** et **.strCouleur** contiennent respectivement le rayon, la position et la couleur de la balle. Au point de départ, la balle est centrée horizontalement et est située juste au-dessus du bâton.

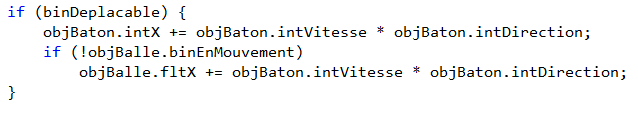
La propriété **.intVitesse** contient la vitesse de déplacement de la balle, c’est-à-dire le nombre de pixels que la balle va se déplacer lorsque cette dernière va se promener dans l’espace intersidéral.

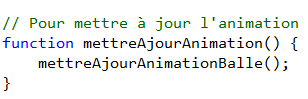
La propriété **.intDirectionDegre** contient la direction de la balle en degrés. Au point de départ, cet angle est situé entre 0 et 180 degrés inclus au hasard. Donc, au point de départ, la balle va se diriger vers le haut.

La propriété **.binEnMouvement** indique si la balle est en mouvement ou non. Au point de départ, la balle n’est pas en mouvement.

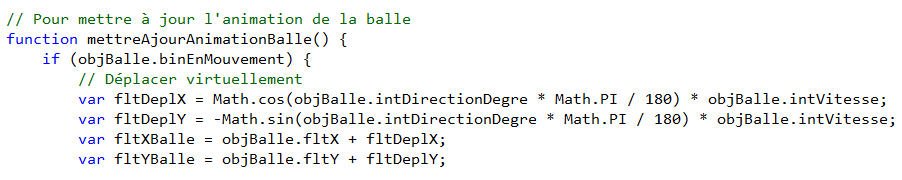
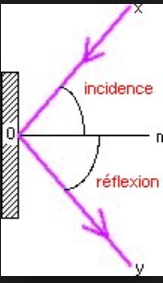


La balle sera en mouvement lorsque l’utilisateur va relâcher la touche « **flèche-en-haut** » dans la fonction **deplacerBaton**.

Prenez note, si la balle n’est pas en mouvement, que la balle suit le déplacement du bâton. Lorsque la balle n’est pas en mouvement, c’est le bâton qui décide de la position de la balle.

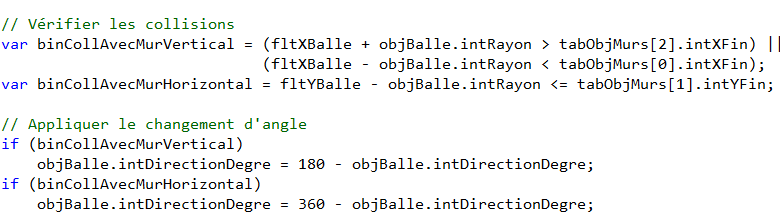
À partir du moment que la balle est en mouvement, elle se déplace toute seule. Dans la fonction **mettreAJourAnimation**, la fonction **mettreAJourAnimationBalle** est appelée. Cette fonction met à jour la nouvelle position de la balle lors de son déplacement automatique (à chaque 1/60 de seconde).

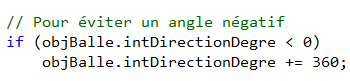
|  |  |
| --- | --- |
| Pour calculer la nouvelle position de la balle à la suite de son déplacement, nous appliquons les formules suivantes où **(x,y)** est la position actuelle de la balle, où **d** est la direction de la balle (en degrés), où **v** est la vitesse de déplacement de la balle et où **(x’,y’)** est sa nouvelle position calculée. | ***x’ = x + cos(d) \* v***  ***y’ = y - sin(d) \* v*** |

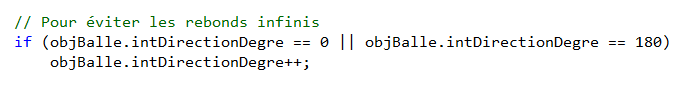


Observez que nous calculons cette nouvelle position virtuellement. La raison c’est, qu’avant de redessiner la balle, nous devons vérifier si la balle est entrée en collision avec un des murs. En effet, si la balle entre en collision avec un des murs, nous devons faire rebondir la balle, c’est-à-dire modifier sa direction.

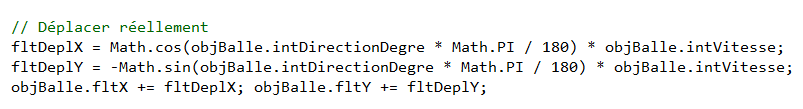
Pour que le rebond soit réaliste, la formule appliquée est  ***d’ = 180 – d*** si la balle entre en collision avec un mur vertical ou ***d’ = 360 – d*** si la balle entre en collision avec un mur horizontal.

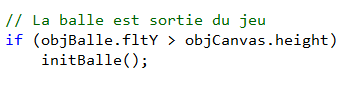


Après avoir appliqué la nouvelle direction, nous vérifions si l’angle de direction de la balle est un nombre négatif. Si c’est le cas, nous additions 360 pour que cet angle soit positif. Prenez note que si l’angle de direction est négatif, cela fonctionne pareil. Mais c’est plus facile à « déboguer » lorsque l’angle est positif.

Nous vérifions également si l’angle de direction de la balle est égal à 0 ou à 180 degrés. Si c’est le cas, nous additionnons 1 à l’angle. Ceci est pour éviter des rebonds infinis. En effet, lorsque l’angle est égal à 0 ou à 180 degrés, la balle rebondit sur le mur de gauche et sur celui de droite à l’infini.

Finalement, nous déplaçons réellement la balle (en modifiant sa position réelle).

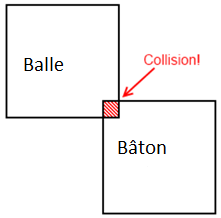


Si la balle sort à l’extérieur du canevas, nous réinitialisons la balle. Cela va faire en sorte que la balle va se redessiner juste au-dessus du bâton et sa nouvelle direction va être déterminée au hasard (entre 0 et 180 degrés).

Avant de poursuivre le laboratoire, testez le jeu. Appuyer sur la flèche-en-haut pour lancer la balle.

6. La collision de la balle avec le bâton

Ouvrez la page Web **6-CollisionAvecBâton.htm.**

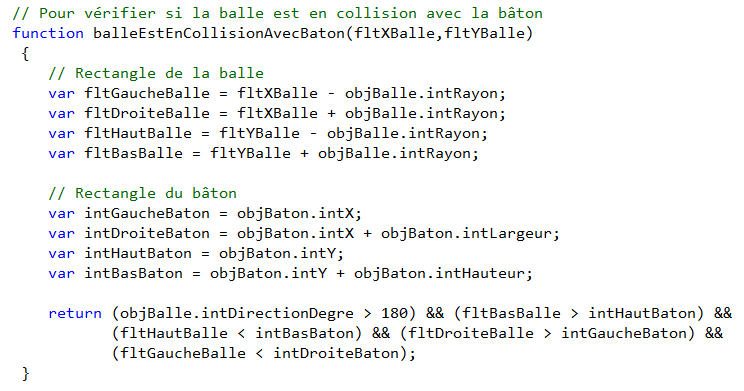
Étant donné que les murs sont situés aux extrémités du canevas, vérifier la collision de la balle avec un mur est relativement simple. Il suffit de vérifier si la balle a franchi ou non la frontière intérieure du mur.

Vérifier la collision de la balle avec le bâton est un peu plus complexe. Il faut délimiter la balle ainsi que le bâton puis tester si les deux rectangles sont en intersection. Si c’est le cas, cela signifie que la balle est entrée en collision avec le bâton. Si ce n’est pas le cas, cela signifie que la balle n’est pas entrée en collision avec le bâton.

Ce n’est pas tout à fait vrai. En effet, la forme de la balle est circulaire et le bâton n’a pas une forme parfaitement rectangulaire. Donc, il est possible que les deux rectangles soient en intersection et qu’il n’y ait aucune collision. Pour être précis, nous devrions vérifier, pixel par pixel, s’il y a eu une collision ou non.

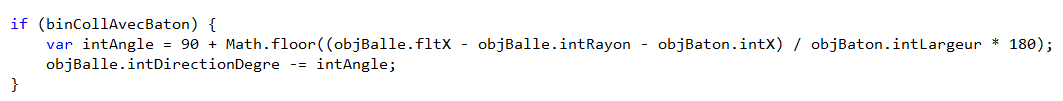
Mais vérifier une collision pixel par pixel est complexe et demande un temps non négligeable. Étant donné que la balle est relativement petite et que les collisions sont relativement rapides, dans la majorité des cas, le joueur ne devrait pas s’apercevoir qu’il n’y a pas eu de réelle collision.



Dans la fonction **mettreAjourAnimationBalle**, nous appelons la fonction **balleEstEnCollisionAvecBaton** en lui passant en paramètre la position virtuelle de la balle.

Cette dernière fonction retourne **true** si les deux rectangles sont en intersection ou retourne **false** dans le cas contraire.

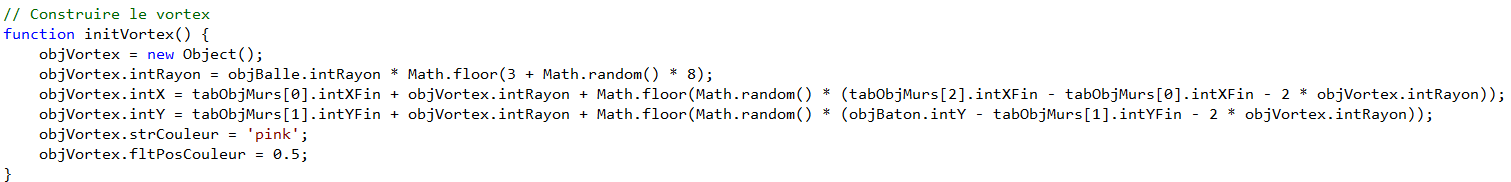
Lorsque la balle entre en collision avec le bâton, le rebond de la balle ne se fait pas exactement de la même manière qu’avec les murs. En fait, le rebond dépend de la position de la balle sur le bâton. Plus que la balle frappe le bâton du côté gauche, plus la balle a tendance à rebondir du côté gauche et plus que la balle frappe le bâton du côté droit, plus la balle a tendance à rebondir du côté droit. **Testez la collision de la balle avec le bâton**.



7. Le dessin et l’animation du vortex

Ouvrez la page Web **7-DessinEtAnimationDuVortex.htm.** Dans ce jeu, il y a un vortex. Le but du jeu est de détruire le vortex à l’aide de la balle.

La fonction **initVortex** sert à paramétrer le vortex.



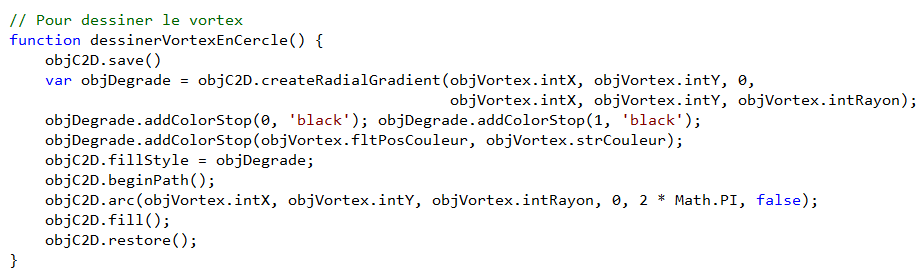
Les propriétés **.intRayon**, **.intX**, **.intY** et **.strCouleur** contiennent respectivement le rayon, la position et la couleur du vortex.

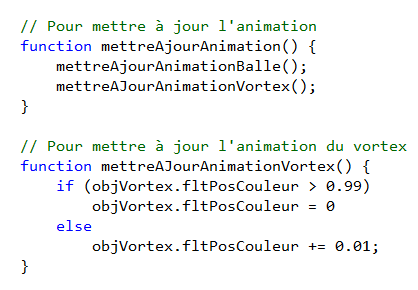
Le rayon du vortex est un nombre au hasard entre 3 fois et 10 fois la taille de la balle. La position du vortex est une position au hasard mais en s’assurant que le vortex va être dessiné entre le mur gauche et le mur droit et entre le mur du haut et le bâton.

La propriété **.fltPosCouleur** est un nombre réel entre 0 et 1 qui détermine à quel endroit à l’intérieur du vortex que la couleur va s’afficher. Au point de départ, elle s’affiche au milieu du vortex (0.5).

Au fur et à mesure de l’animation, cette position est modifiée pour donner l’impression que la lumière part de l’intérieur du vortex vers l’extérieur.

La fonction qui dessine le vortex est la fonction **dessinerVortexEnCercle**. Cette fonction dessine un cercle plein dont le remplissage est un dégradé radial.



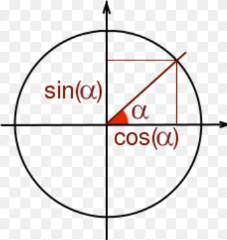


L’animation du vortex est réalisée à l’aide de la fonction **mettreAJourAnimationVortex**. A chaque 1/60 de seconde, cette fonction augmente de 0.01 la position de la couleur à l’intérieur du vortex. C’est cela qui donne l’impression que la lumière va de l’intérieur vers l’extérieur du vortex.

Ici, l’utilisation d’un cercle pour dessiner le vortex n’est pas très adéquate. En fait, au lieu de dessiner un cercle, nous devrions dessiner une spirale (car un vortex a habituellement la forme d’une spirale).

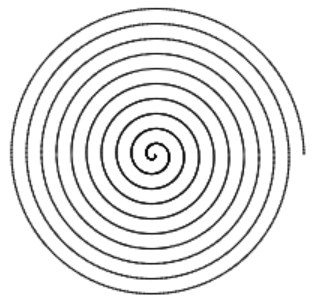
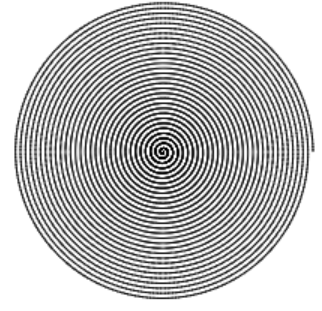
Le problème c’est de savoir comment dessiner cette spirale.

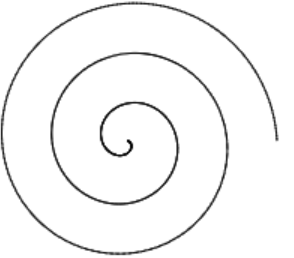
|  |  |
| --- | --- |
| Pour calculer la coordonnée ***(x,y)*** d’un point situé sur la circonférence d’un cercle de centre ***(xc ,yc)*** et de rayon ***r*** pour un angle **α** quelconque, il faut appliquer les formules suivantes. Ces formules sont les équations trigonométriques du cercle. | ***x = xc + cos(α) \* r***  ***y = yc + sin(α) \* r*** |

En programmation, si on fait varier l’angle **α** de 0 degré jusqu’à 360 degrés à l’intérieur d’une boucle et si on dessine chacun des points ***(x,y)*** résultants, il est possible de tracer tous les points situés sur la circonférence d’un cercle et ainsi dessiner le cercle.

Pour dessiner une spirale, le principe est le même sauf que le rayon doit débuter à 0 (au centre de la spirale) et doit augmenter de manière progressive jusqu’à atteindre le rayon total.

Plus le nombre de circonvolutions dans la spirale augmente et moins le rayon doit augmenter rapidement. Dans l’exemple ci-dessous, les trois spirales ont exactement le même rayon. La différence entre les trois spirales est que la 1ère spirale a 3 circonvolutions, la 2ième spirale a 10 circonvolutions et la 3ième spirale a 30 circonvolutions.



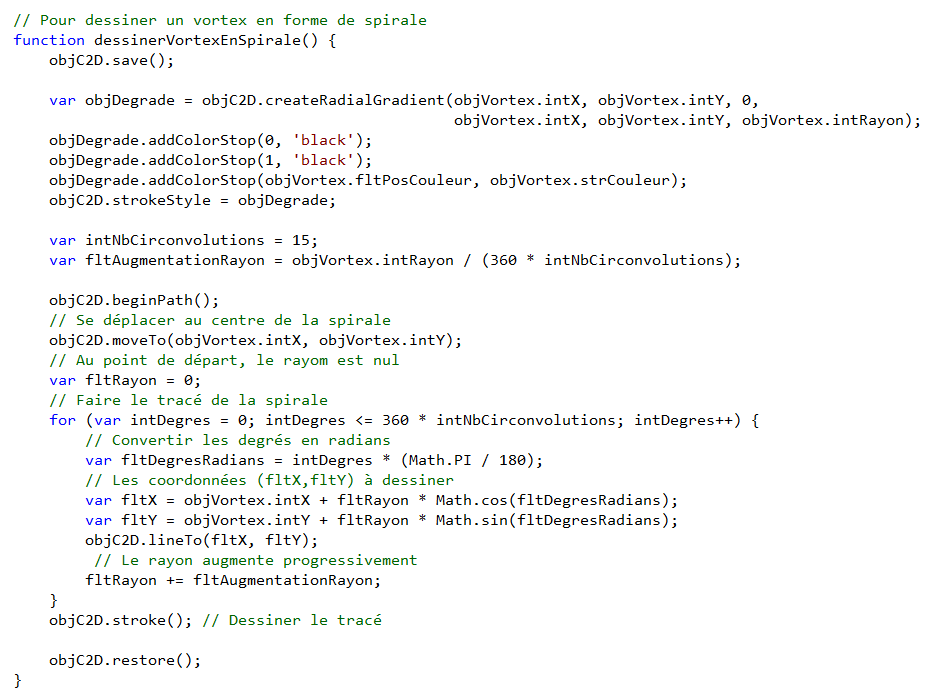


Étant donné que le rayon augmente moins rapidement lorsque la spirale a un plus grand nombre de circonvolutions, la distance entre chacune des circonvolutions est plus petite.

La fonction **dessinerVortexEnSpirale** sert à dessiner le vortex mais cette fois-ci en lui donnant une forme de spirale.

Dans la fonction **dessiner**, remplacez **dessinerVortexEnCercle()** par **dessinerVortexEnSpirale()** (à la ligne 210) et observez le vortex sous forme de spirale.

Tentez de comprendre les instructions qui dessinent le vortex en spirale (lisez les commentaires). Ici, il y a 15 circonvolutions. Vous pouvez, si vous voulez, diminuer ou augmenter le nombre de circonvolutions (pour voir l’effet).



Nous aimerions donner l’illusion que le vortex tourne. Pour ce faire, nous allons utiliser la propriété **.fltPosCouleur** qui donne la position de la couleur dans le vortex. Nous allons nous en servir pour déterminer à partir de quel angle le vortex va se dessiner.

Pour ce faire, remplacez l’instruction suivante :

***for (let intDegres = 0; intDegres<=360\* intNbCirconvolutions; intDegres++)***

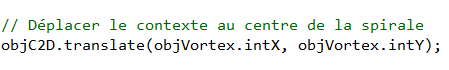
par l’instruction suivante :

***for (let intDegres = 360\*objVortex.fltPosCouleur; intDegres<=360\*(intNbCirconvolutions+objVortex.fltPosCouleur); intDegres++)***

Exécutez et observez qu’on a l’impression que le vortex tourne.

Ouvrez la page Web **7B-DessinEtAnimationDuVortex.htm.**

Nous avons « reprogrammé » la fonction **dessinerVortexEnSpirale**. L’effet est le même que dans la fonction précédente sauf que nous l’avons programmé de manière différente.



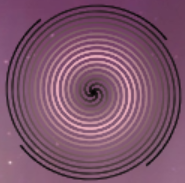
Tout d’abord, au début de la fonction, nous déplaçons le contexte au centre de la spirale.

Cela signifie, qu’à partir de maintenant, le centre de la spirale n’est plus le point **(objVortex.intX, objVortex.inY)** mais le point **(0,0).**

Par conséquent, le centre du dégradé est le point **(0,0)**.

Lorsque nous nous déplaçons au centre de la spirale, nous nous déplaçons au point **(0,0)**.

Finalement, lorsque nous calculons les coordonnées à dessiner, nous n’additions pas le centre (car il est **(0,0)**).

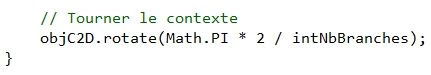
Maintenant, dans la fonction **dessiner()**, remplacez **dessinerVortexEnSpirale()** par **dessinerVortexEnSpiraleAvecPlusieursBranches()** (ligne 211) et observez le vortex.

Ici, le vortex a quatre branches (au lieu d’une seule comme c’était le cas auparavant).

Voici la manière que nous avons programmé cela.

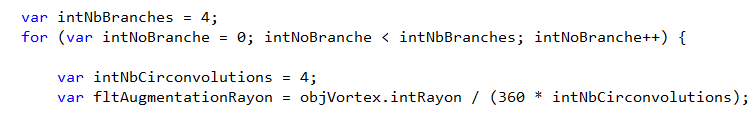
Tout d’abord, nous déterminons le nombre de branches puis, dans une boucle, nous dessinons autant de spirales qu’il y a de branches. Ici, il y a quatre branches donc quatre spirales sont dessinées.



A la fin, après avoir dessiné une spirale, nous tournons le contexte. Cela fait en sorte que la prochaine spirale va se dessiner à la suite d’une rotation qui varie en fonction du nombre de branches. C’est cela qui donne l’impression qu’il y a plusieurs branches.

Il est important de noter, qu’à chaque fois, c’est toujours la même spirale qui est dessinée mais chacune des spirales subit une rotation qui lui est spécifique avant de se dessiner.

Je vous suggère fortement de modifier le nombre de branches (ligne 286) et le nombre de circonvolutions (ligne 289) de chaque spirale pour voir le type de vortex que cela donne.



Pour bien visualiser le vortex, je vous suggère d’augmenter son rayon à 120 pixels (enlevez le commentaire à la ligne 101).

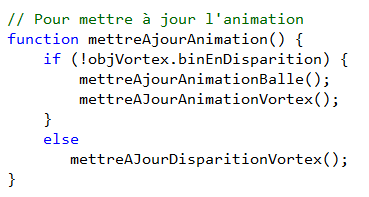
8. La balle dans le vortex

Ouvrez la page Web **8-BalleDansLeVortex.htm.**

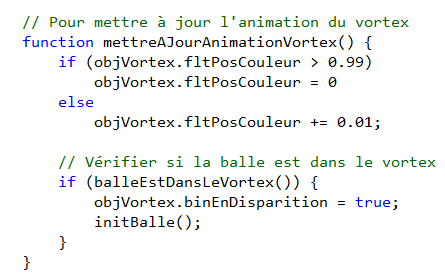
Dans ce jeu, dès que la balle sera entièrement située à l’intérieur du vortex, le vortex va disparaître progressivement en absorbant la balle.

Par conséquent, il va y avoir deux animations : une première animation lorsque la balle va se déplacer librement dans l’espace et une deuxième animation lorsque le vortex va disparaître progressivement (en compagnie de la balle).

C’est la raison pour laquelle la propriété **.binEnDisparition** a été ajoutée dans le vortex. Cette propriété indique si le vortex est en disparition progressive ou non. Au point de départ, il ne l’est pas.



La fonction qui met à jour l’animation a également été modifiée. Si le vortex n’est pas en disparition progressive, nous animons la balle et le vortex de manière normale. Par contre, si le vortex est en disparition progressive, nous faisons disparaître le vortex à l’intérieur d’une animation.

Ici, nous vérifions si la balle est entièrement située à l’intérieur du vortex dans l’animation normale du vortex.

Si la balle est entièrement située à l’intérieur du vortex, nous indiquons que le vortex est en disparition progressive puis nous initialisons la balle. Cela va faire en sorte que le vortex va disparaître progressivement et que la balle ne sera plus là.

Pour vérifier si la balle est entièrement située à l’intérieur du vortex, nous n’allons pas utiliser le rectangle de collision comme nous l’avons fait pour vérifier la collision entre la balle et le bâton car cela n’est pas assez précis.

Ce que nous allons utiliser c’est l’équation cartésienne du cercle centrée à l’origine : ***x2 + y2 = r2*** où ***(x,y)*** est un point quelconque situé sur la circonférence du cercle et où **r** est le rayon du cercle. Tous les points situés à l’intérieur du cercle satisfont l’équation suivante : ***x2 + y2 < r2*** et tous les points situés à l’extérieur du cercle satisfont l’équation suivante : ***x2 + y2 > r2***.

***x2 + y2 < r2***

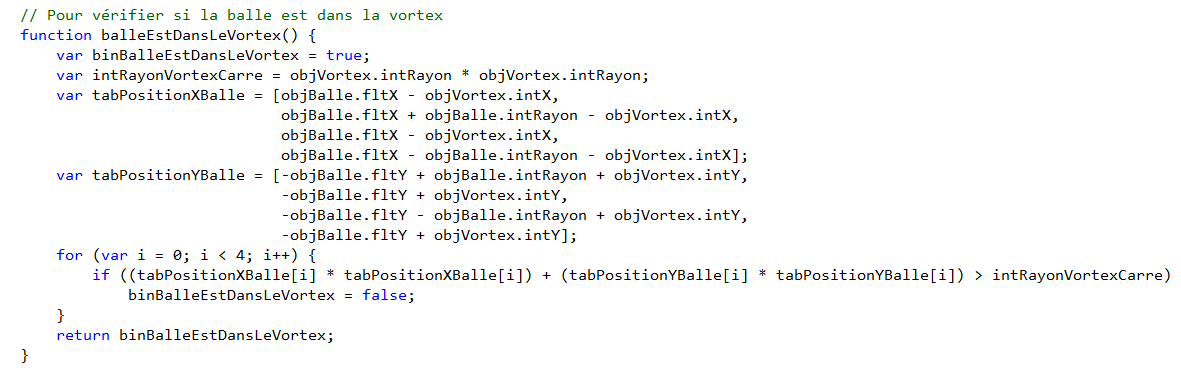
***x2 + y2 = r2***

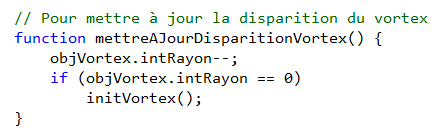
***x2 + y2 > r2***

Si la somme ***x2 + y2*** d’un des quatre points extrêmes de la balle est plus grande ou égale au rayon du vortex au carré, cela signifie que la balle n’est pas entièrement située à l’intérieur du vortex. Par contre, si la somme ***x2 + y2*** des quatre points extrêmes de la balle est plus petite que le rayon du vortex au carré, cela signifie que la balle est entièrement située à l’intérieur du vortex.

|  |  |  |  |
| --- | --- | --- | --- |
| La balle n’est pas entièrement située à l’intérieur du vortex. |  | La balle est entièrement située à l’intérieur du vortex. |  |

C’est ce que fait la fonction **balleEstDansLeVortex**. Tentez de comprendre chacune des instructions.





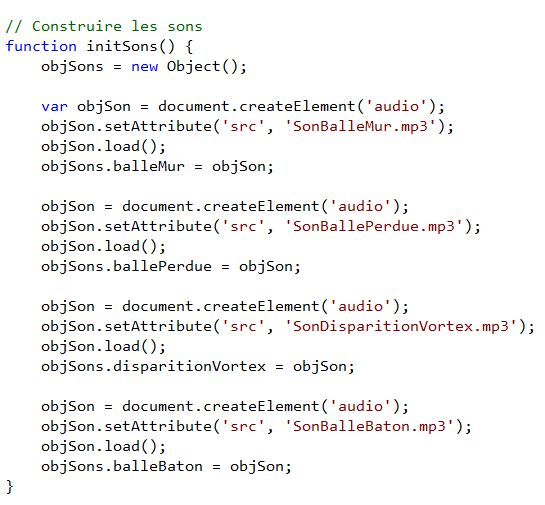
Pour faire disparaître le vortex progressivement, à chaque 1/60 de seconde, nous diminuons son rayon de 1 pixel. Lorsque son rayon atteint 0, nous créons un nouveau vortex.

Testez le jeu. Organisez-vous que la balle frappe le vortex (la balle doit être entièrement située à l’intérieur du vortex) et observez la disparition progressive du vortex puis l’apparition d’un nouveau vortex.

9. Les sons

Ouvrez la page Web **9-Les sons.htm.**

Dans ce jeu, lorsque la balle frappe le mur, lorsque la balle frappe le bâton, lorsque la balle entre à l’intérieur du vortex et lorsque la balle disparaît à l’extérieur des limites du jeu, nous allons faire jouer un son particulier. Il va de soi que vous devez avoir des écouteurs pour entendre les sons.

Pour les besoins de la cause, nous avons placé tous les sons dans le même objet (dans l’objet **objSons**).

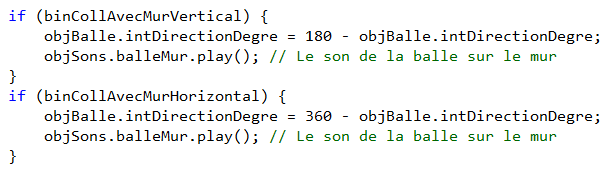
Pour chacun des quatre sons, le principe est le même.

Tout d’abord, nous créons un nouvel élément de type « **audio »**. Cela a comme conséquence de créer dynamiquement une balise **<audio></audio>**. Cette balise n’est pas sur la page Web mais elle existe.

Par la suite, nous affectons, à cet élément, la source du son (c’est-à-dire le nom du fichier qui contient le son). Ici, ce sont tous des fichiers .**mp3**. Ce format audio est reconnu par tous les navigateurs.

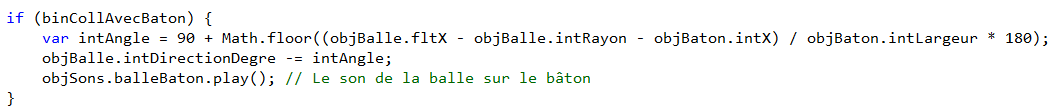
Par la suite, nous chargeons ce son en mémoire (***.load()***).

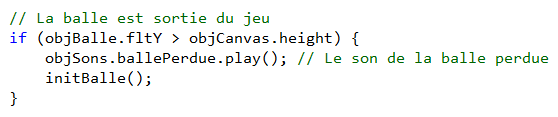
Finalement, nous affectons le son à la bonne propriété. Par exemple, **objSons.balleMur** représente le son de la balle qui entre en collision avec un mur.

Pour faire jouer le son, il faut utiliser la méthode **.play().**

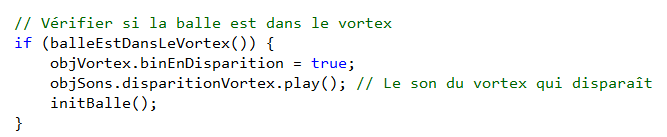
Ici, lorsque la balle entre en collision avec un des murs, nous faisons jouer le son **objSons.balleMur**.

Lorsque la balle entre en collision avec le bâton, nous faisons jouer le son **objSons.balleBaton.**





Lorsque la balle sort à l’extérieur du jeu, nous faisons jouer le son **objSons.ballePerdue.**



Lorsque le vortex disparaît de manière progressive, nous faisons jouer le son **objSons.disparitionVortex.**

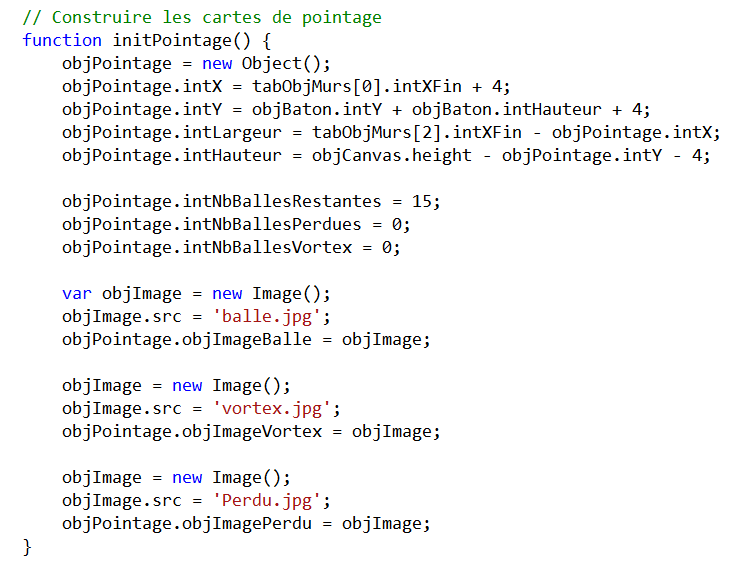
Si vous avez des écouteurs, prenez le temps de bien écouter chacun des sons pendant le déroulement du jeu.

10. Le pointage

Ouvrez la page Web **10-Le pointage.htm.**

Dans ce jeu, nous comptons les points. Les différentes cartes de pointage s’affichent en permanence tout au long du jeu.

La fonction **initPointage** sert à paramétrer la zone réservée aux cartes de pointage.

Les propriétés **.intX**, **.intY**, **.intLargeur** et **.intHauteur** contiennent respectivement la position et la taille de la zone réservée aux cartes de pointage. Cette zone est située dans le bas du canevas juste en-dessous du bâton.

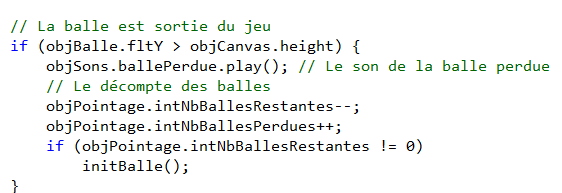
Les propriétés **.intNbBallesRestantes**, **.intNbBallesPerdues** et **.intNbBallesVortex** contiennent respectivement le nombre de balles qui restent à lancer (au point de départ, il y en a 15), le nombre de balles perdues et le nombre de balles qui ont atteint le vortex.

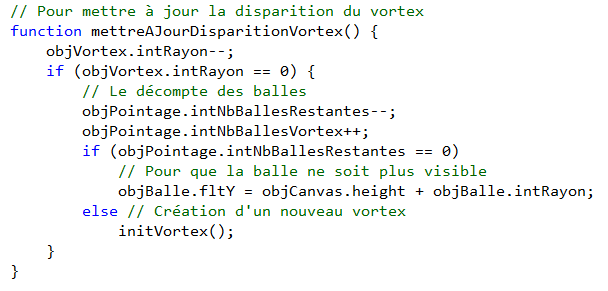
Les propriétés **.objImageBalle**, **.objImageVortex** et **.objImagePerdu** contiennent les différentes images qui sont dessinées lorsque les cartes de pointage sont dessinées.

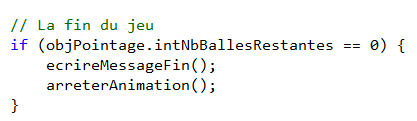
La fonction **dessinerPointage** est relativement longue. Vous pouvez consulter la programmation de cette fonction si cela vous chante.

Voici les cartes de pointage. La 1ère carte contient le nombre de balles qui restent à lancer, la 2ième carte contient le nombre de balles qui ont atteint le vortex et la 3ième carte contient le nombre de balles perdues.



Ici, à chaque fois qu’une balle sort à l’extérieur du jeu, nous diminuons le nombre de balles restantes et nous augmentons le nombre de balles perdues. S’il ne reste plus de balles, nous ne créons pas de nouvelles balles (car le jeu est terminé).

De la même manière, à chaque fois qu’une balle atteint le vortex, à la fin de la disparition progressive du vortex, nous diminuons le nombre de balles restantes et nous augmentons le nombre de balles qui ont atteint le vortex. S’il ne reste plus de balles, nous modifions la position de la balle pour qu’elle ne soit plus visible sinon nous créons un nouveau vortex.



A la fin du dessin des cartes de pointage (les cartes de pointage sont les derniers objets à être dessinés à l’intérieur du cycle d’animation), nous vérifions s’il ne reste plus de balles. Si c’est le cas, nous écrivons un message puis nous arrêtons complètement le cycle d’animation (car le jeu est terminé).





11. Références

<http://www.w3schools.com>

<http://chimera.labs.oreilly.com/books/1234000001654/ch05.html#moving_in_a_straight_line>

<http://www.gamedev.net/page/resources/_/technical/game-programming/collision-detection-r735>

<http://code18.blogspot.ca/2012/01/dessiner-une-spirale-dans-un-canvas.html>