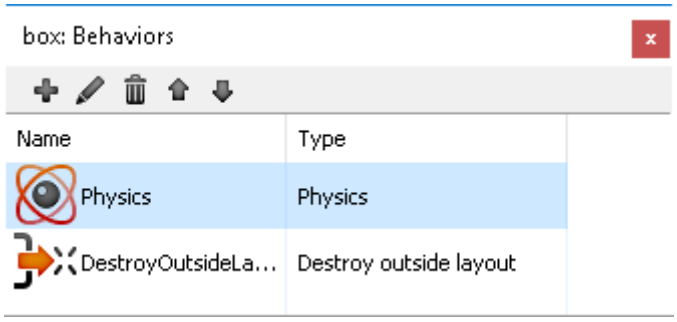
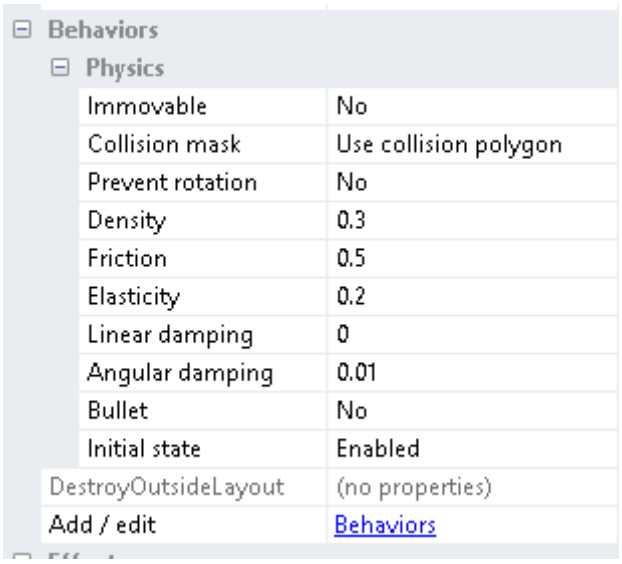
Tutoriel – Physics

Bases établies :

Tous les objets nécessitant un comportement « physique » doivent évidemment bénéficier du behavior « physic » pour fonctionner correctement, avec tous les paramètres concernés.



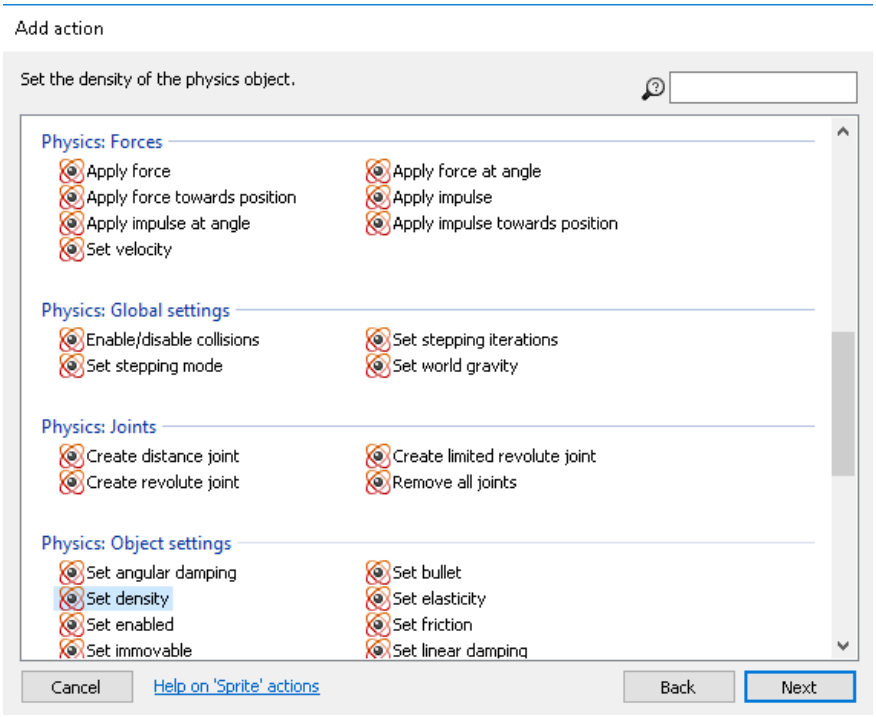
Tous nos objets en dehors du canon, de son socle, et des particules en ont besoin (boulet + écrans + cible).



Les paramètres du behavior physic sont simples et parlent normalement d'eux même. Chaque paramètre correspond à une notion de physique basique OU à une option vis-à-vis de l'objet (peut faire des rotations ou non, est soumis à la gravité ou non…). N'hésitez pas à me poser des questions.

Sur la feuille d'événement de votre projet (page de scripts simplifiés pour générer du JS), vous pouvez manipuler vos objets physiques (ajouter force ou vitesse…) et reparamétrer ce que vous voulez selon l'événement que vous voulez gérer.

En cas de doute, un rapide descriptif du paramètre ou de l'option choisie sera affiché en haut à gauche de la fenêtre, visible sur l'image suivante.



Au moment de votre départ, nous avions un canon qui marche à peu près et un décor fonctionnel sous behavior physique.

Vos objectifs :

À la suite de notre première séance de cours, il vous reste en théorie quatre étapes à compléter pour terminer l'exercice :

1. Tous les boulets ayant terminés leur trajectoire (boulets au sol immobiles ou pas loin) sans toucher la cible doivent disparaitre de l'écran

2. Notre cible doit s'autodétruire au contact du sol

3. Notre cible doit s'autodétruire au contact du boulet

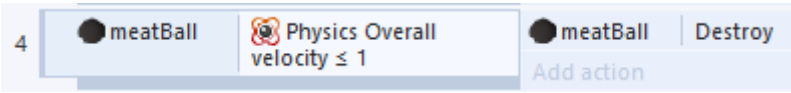
4. Notre cible doit faire apparaitre nos particules lorsqu'elle est détruite (explosion rougeâtre)

Essayez d'accomplir ces quatre tâches vous-même. Cela dit, la solution (ou plutôt, une des solutions possibles) se situe dans les pages suivantes, ainsi que plusieurs améliorations.

Détruire les boulets inutilisés :

Nous voulons gérer notre cible, à savoir la faire exploser puis relancer le niveau quand elle entre en contact avec notre boulet de canon. Avant ça, nous allons faire un peu de ménage : Les objets physiques sont couteux pour le CPU et la mémoire, aussi faut-il nous assurer qu'ils ne restent pas indéfiniment sur notre terrain lorsqu'ils ne servent plus.

Dans notre cas, les seuls objets physiques qui ne sont pas toujours essentiels sont les boulets de canon qui ont raté leur trajectoire. Ils reposent maintenant sur le terrain, mais ne servent plus à rien. Ils prennent juste de la place.

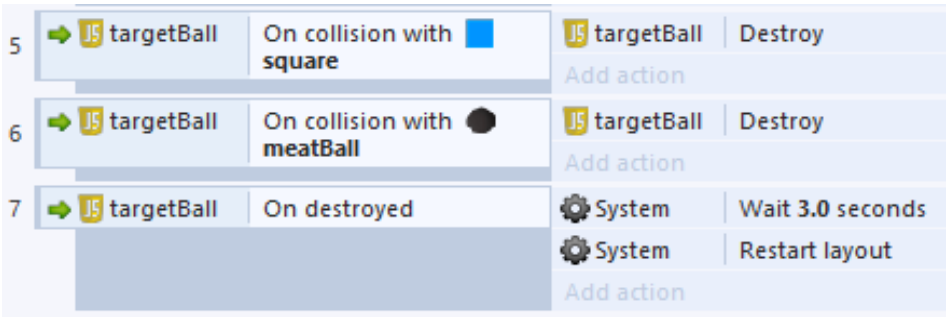


On instaure juste un événement précisant que si la vitesse d'un objet meatBall (mon boulet) est inférieur ou égal à 1, on doit détruire l'objet. Ainsi, les boulets ayant terminés leur course ou pas loin ne resteront pas longtemps à l'écran.

Ceci étant fait, revenons-en sur la cible.

Gérer la cible :

La gestion de l'objet cible est des plus simples.

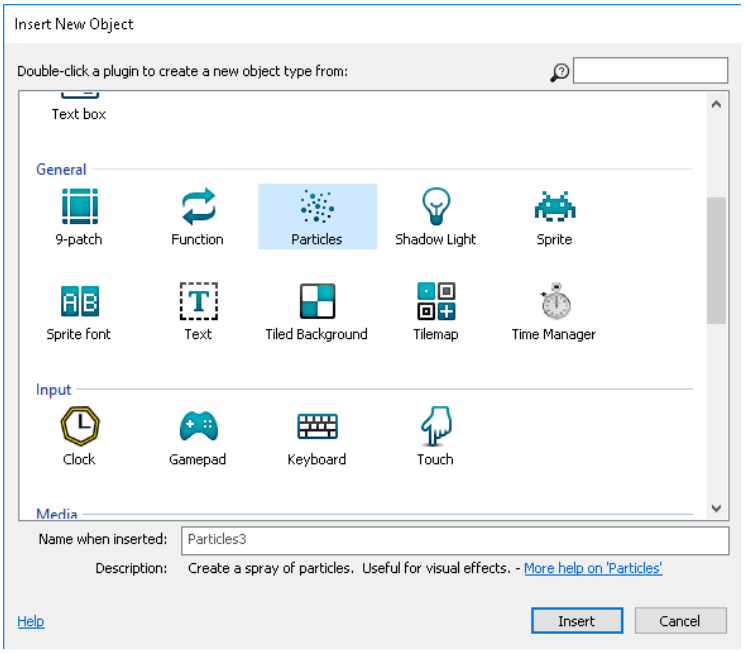


Premièrement, si notre cible touche le sol, elle va s'autodétruire. Même principe pour le boulet de canon. Dans les deux cas, on ajoute un événement sur notre objet cible pour tester s'il est en collision avec le sol ou le boulet.

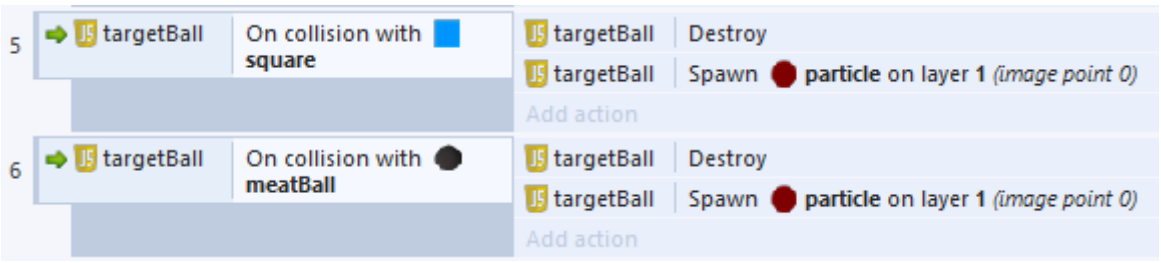
Enfin, en cas de destruction de l'objet cible, on demande au système d'attendre trois secondes avant d'enclencher le reset du layout (donc de notre niveau).

Gérer les particules :

Les objets « Particles » peuvent rapidement créer des effets visuels sympathiques en créant et en déplaçant indépendamment de nombreuses petites images, selon certains paramètres. Ce sont des objets polyvalents capable de nombreux types d'effets. Dans notre cas, nous voulons simuler une légère explosion rougeâtre lorsque notre cible est détruite.



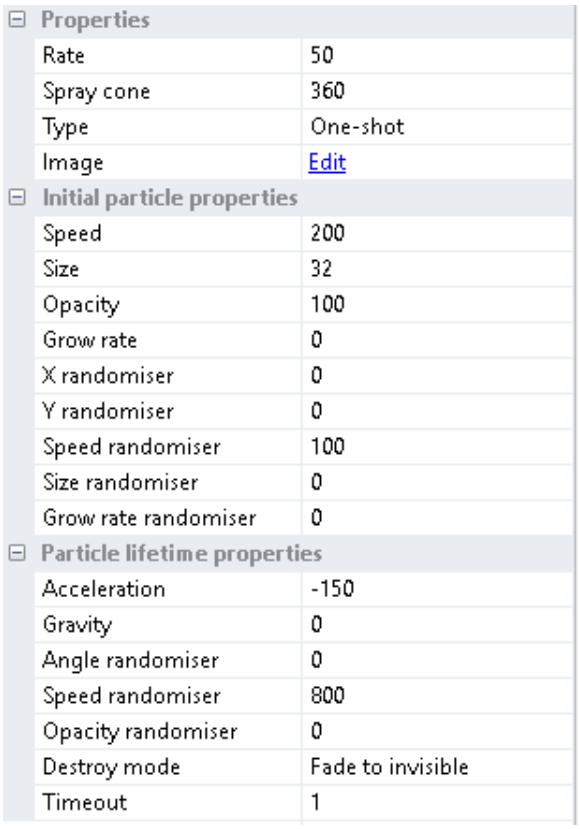
Ajoutez un nouvel objet particle, et sélectionnez l'image du même nom dans le dossier que je vous avais donné. Vous venez de créer votre première particule. Comme pour tous les objets, ses propriétés sont disponibles sur le volet de gauche.



Il ne reste plus qu'à faire spawn notre toute nouvelle particule au moment où notre cible est détruite. Encore faut-il la configurer pour obtenir l'effet que nous souhaitons.

Entrez ces valeurs dans le volet de gauche de notre particule.

Tutoriel – Physics



Properties	
Rate	50
Spray cone	360
Type	One-shot
Image	Edit
Initial particle properties	
Speed	200
Size	32
Opacity	100
Grow rate	0
X randomiser	0
Y randomiser	0
Speed randomiser	100
Size randomiser	0
Grow rate randomiser	0
Particle lifetime properties	
Acceleration	-150
Gravity	0
Angle randomiser	0
Speed randomiser	800
Opacity randomiser	0
Destroy mode	Fade to invisible
Timeout	1

Si vous avez un doute sur une des propriétés, vous pouvez cliquer sur son nom. Un léger descriptif sera affiché tout en bas à gauche de votre écran. Dans les grosses lignes :

Rate – Nombre de particules créées par secondes

Spray Cone – Création selon un angle/cone de X degrés

Type – Projection de particules en une fois ou en continue

Speed – Vitesse des particules en pixels par secondes

Size – Taille de chaque particule en pixels

Opacity – Opacité des particules (de 0 à 100)

Les Randomiser – Modifier aléatoirement certains paramètres de l'objet (abscisse, vitesse, taille…) selon un facteur de X

Acceleration – Facteur d'accélération de chaque particule en pixels par secondes. Ici configuré en négatif, nos particules devant rester à la vitesse constante définie dans Speed.

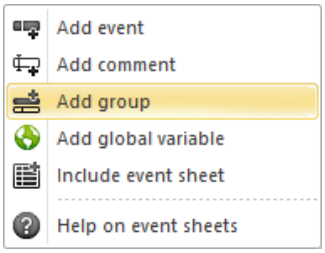
Gravity – Simuler X puissance de la gravité sur les particules

Destroy mode – Mode de destruction de toutes les particules

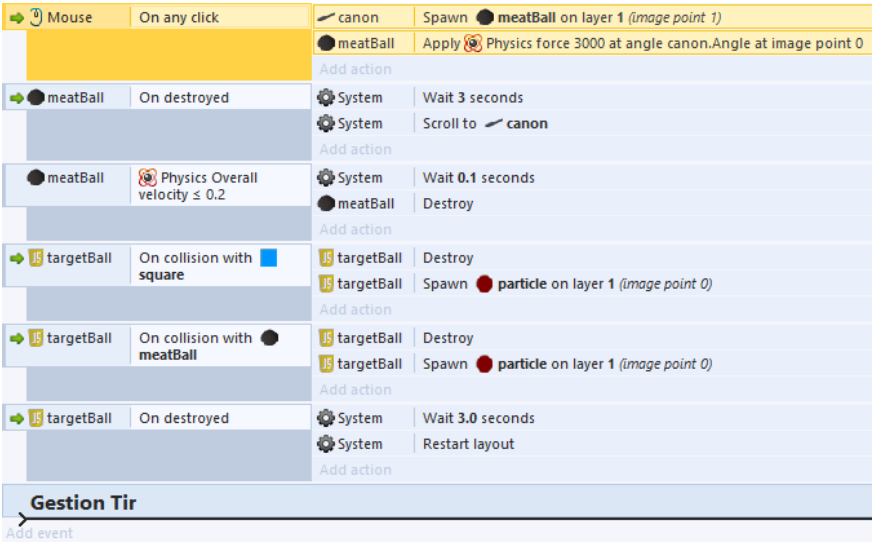
Timeout – Temps en secondes avant destruction des particules

Améliorations 1 – Grouper ses événements :

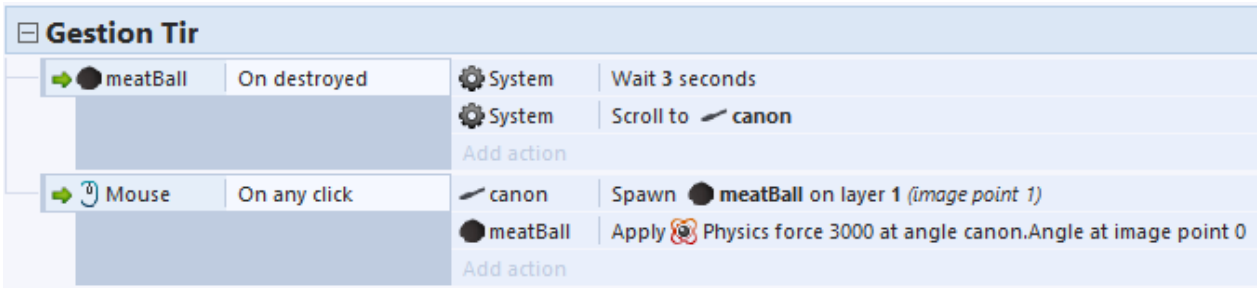
Faites clic droit sur votre feuille d'événement, et ajouter un nouveau groupe. Ils améliorent la lisibilité et l'organisation de toute de votre feuille. Nommez le « Gestion Tir ».



Faites glisser les deux événements responsables de la gestion du tir jusqu'au groupe pour les ajouter à ce dernier.



Vous devriez normalement obtenir ce rendu-là :



Améliorations 2 – Trainée derrière le boulet :

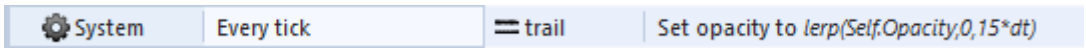
Etudions un peu la différence sprites/particules.

Dans mon projet, tous les objets physiques en mouvement laissent une petite trace derrière eux. Ça met leur trajectoire en valeur. Ceci pourrait nous être utile pour visualiser la trajectoire de nos boulets en mouvement.

Nous pourrions avoir recours aux particules. Cela dit, j'ai préféré utiliser des sprites. Techniquement, l'utilisation des particules est moins lourde pour le GPU. Cependant, les sprites nous offrent bien plus de contrôles que les particules pour des résultats visuellement identiques. En contrepartie, vous allez découvrir qu'ils sont bien moins simples à gérer.

Commencez par ajouter le sprite « trail » sur le canvas.

Dans la feuille d'événement, on va tout d'abord s'assurer que notre sprite devienne de plus en plus transparent au cours du temps, avant de totalement disparaitre. Aussi, il devra constamment adapter son angle à celui du boulet de canon.



Petite explication sur la fonction lerp(A,B,C) :

Fonction native permettant de trouver une valeur située entre deux autres valeurs. Les deux arguments A et B correspondent aux bornes choisies (dans notre cas, l'opacité courante de l'objet trail et 0, puisqu'on veut le rendre de moins en moins opaque à mesure que le temps passe depuis sa création).

L'argument C correspond à un nombre compris entre 0 et 1.

0 correspond à A, 1 correspond à B et 0.5 au milieu.

On peut également imaginer C comme un pourcentage.

lerp(0,300,0.5) ou lerp(300,0,0.5) = 150

(50% de la distance entre 0 et 300).

lerp(100,0,0.2) nous donnerait 80.

(20% de la distance entre 100 et 0)

lerp(0,100,0.2) nous donnerait 20.

(20% de la distance entre 0 et 100)

La fonction peut se résumer au calcul suivant : A+C*(B-A)

Petite explication sur la valeur dt :

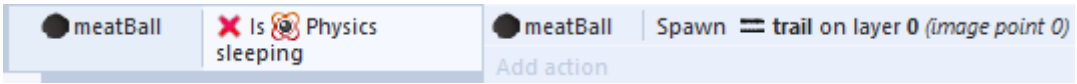
Delta-time (ou dt) est la durée écoulée en secondes depuis la dernière mise à jour du jeu (donc le « tick » le plus récent). Cette valeur est utilisée pour s'assurer que tous les calculs qui y ont recours sont indépendants des FPS de notre jeu.

C’est un tout petit nombre utilisé comme petite sécurité.

Retour à l’exercice :

Vous devriez maintenant comprendre le screenshot précédent, avec le calcul utilisant la fonction lerp. Nous voulons rendre notre sprite de moins en moins opaque. Nos bornes sont donc l’opacité courante de l’objet et 0. Plus la valeur du troisième argument est élevé, plus vite notre trainée disparaitra (car pourcentage choisi plus ou moins grand).

Personnellement, je trouvais que ça rendait bien avec 15. Vous êtes libres de tester diverses valeurs et d’en garder une. Maintenant, il va s’agir de lier le boulet à sa trainée.



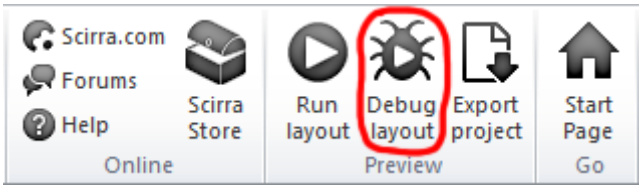
Pour ça, on vérifie que notre objet physique est actif ou non. S’il est actif, c’est qu’il est en mouvement. S’il dort, c’est qu’il est immobile, quelque part dans l’environnement.

Tant qu’il est actif, on crée des objets « trail » en son centre (image point 0) pour rendre sa trajectoire visible. Si vous testez le résultat maintenant, vous verrez que notre tâche est presque terminée. En revanche, en l’état, c’est extrêmement mal optimisé. C’en est même dangereux.

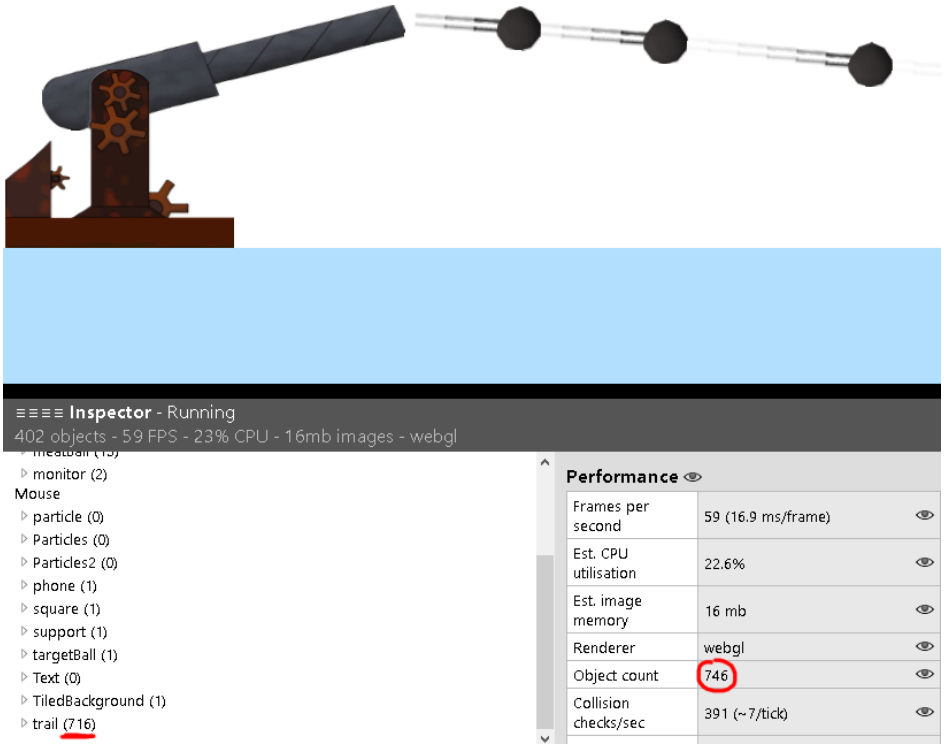
Tutoriel – Physics

Utilisation du debugger pour l'optimisation :

Lancez le projet via le debugger.



Le debugger nous permet de voir en temps réel si le projet tourne normalement, au travers de nombreuses informations.



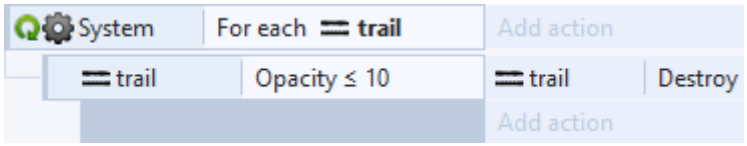
En tirant, vous verrez que le nombre total d'objets actuellement présents sur l'environnement va (trop) rapidement augmenter. Plus il y a d'objets, plus la mémoire est sollicitée et plus vous aurez de problèmes de latences.

Le panel de gauche liste tous les objets utilisés par ce même environnement, on peut donc rechercher l'objet « tumeur » responsable du problème. En l'occurrence, trail.

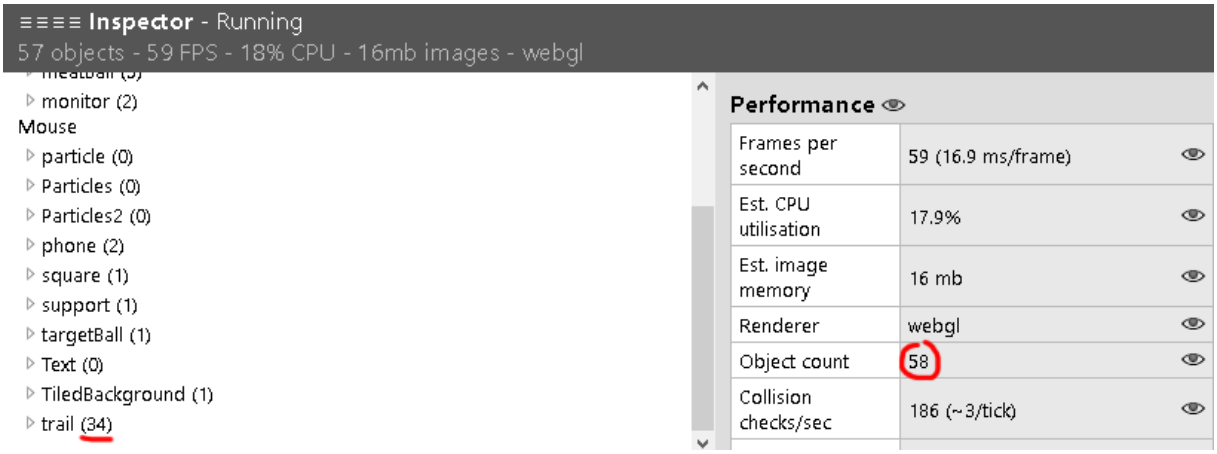
Optimiser notre projet :

Notre objet « trail » a été configuré pour devenir de plus en plus transparent, et ce dès sa création. Au bout d'un moment, il n'est donc plus visible du tout sur le canvas. Mais ce n'est pas pour autant qu'il n'existe plus. Il est toujours dans l'environnement, mais avec sa propriété Opacity à 0.

Du coup à chaque fois que vous tirez un boulet, vous noyez progressivement le canvas d'objets invisibles. En plus, les objets physiques sont très lourds pour le CPU. En avoir un grand nombre n'arrange pas la situation. Grosso modo, entre votre trainée et la physique des boulets et des écrans, le joueur peut faire cracher votre projet s'il tir en continue.



Une boucle foreach résoudra le souci de la trainée. Pour chaque objet trail, si l'opacité est inférieure ou égale à 10, on détruit définitivement l'objet du canvas. De cette façon, on évite de saturer l'environnement de sprites transparents.



Ici, notre boucle utilise un sous événement (lequel permet de vérifier l'opacité du trail). Pour faire des sous événements, il vous suffit de sélectionner n'importe quel événement de votre feuille, puis d'appuyer sur « S » (créer un Sub event).

A présent, voyons comment limiter le tir de notre canon.

Améliorations 3 - Limite de tir du canon :

Petite introduction à la création et l'utilisation des variables sur Construct. Nous voulons éviter que l'utilisateur puisse saturer l'environnement d'objets physiques.

Construct distingue deux types de variables :

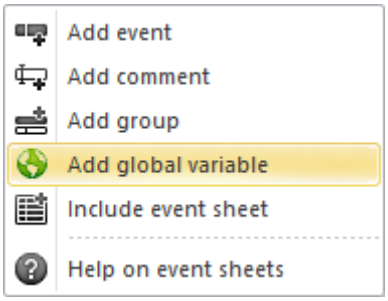
Variables globales – S'affichent tout en haut de la feuille d'événements et sont accessibles partout au sein de celle-ci. Elles peuvent être utilisées dans n'importe quelle situation.

Variables locales – S'affichent dans le groupe d'évènements ou sous événement ou elles ont été créées, et ne sont accessibles que par ces événements ou sous événements. Pas ailleurs. Leurs valeurs sont reset à chaque tick. Elles sont principalement utilisées pour stocker des valeurs temporaires.

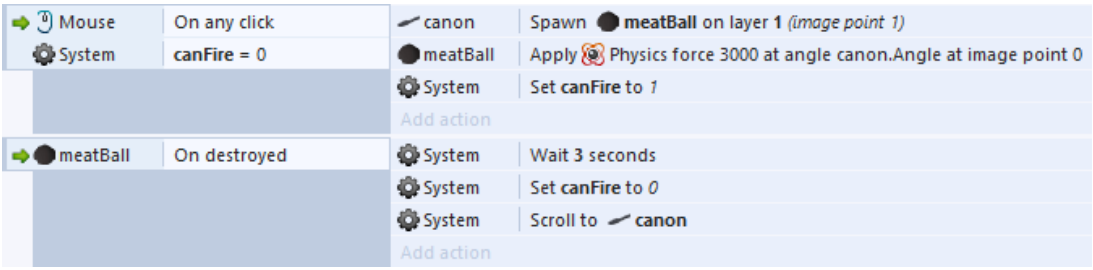
Une variable peut être textuelle, ou numérique. A noter qu'ils existent également des variables d'instances, c'est-à-dire des variables définies sur un objet précis au moment de sa création (disons par exemple les points de vie d'un sprite). Elles peuvent être booléennes, textuelles ou numériques.

Alternative 1 – Passer par la variable globale :

Sur la feuille d'événement, faites un clic droit et sélectionnez « Ajouter une variable globale »



Donnez-lui un nom, un type « number » et une valeur initiale égale à 0. Pour le reste, c'est tout simple :



On rajoute une condition dans l'événement ayant charge de faire tirer notre canon.

Il signifie désormais « Si on clique ET canFire == 0 ALORS on peut tirer » sachant qu'un tir initialise la variable canFire à 1. En l'état, le joueur ne peut jouer qu'une fois.

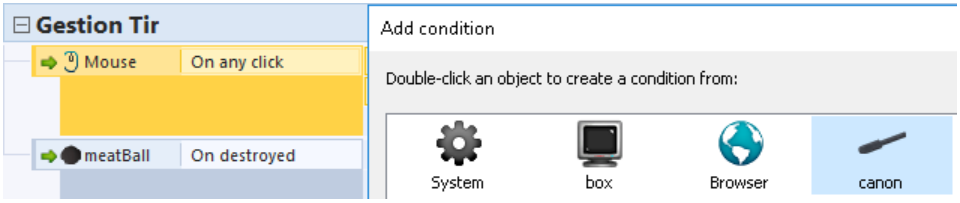
Après un tir, c'est terminé. Nous voulons lui redonner le contrôle du canon dès que son premier essai est terminé. Nous rajoutons donc un événement pour gérer ce qu'il se passe lorsque le boulet est détruit. Lorsque le boulet est détruit, on demande système d'attendre trois secondes avant de reset la variable canFire à 0 puis faire revenir la caméra au canon.

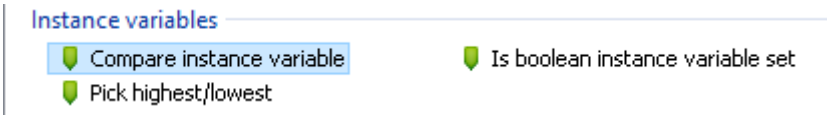
Alternative 2 – Passer par la variable d'instance :

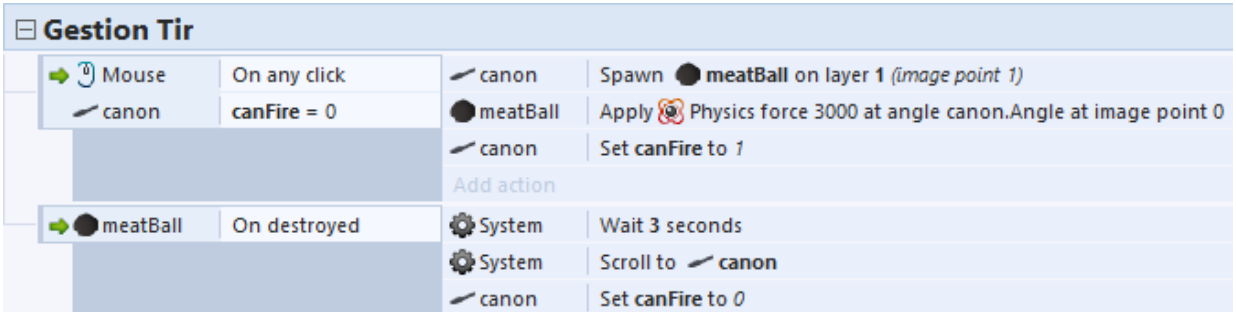
Dans notre histoire de canon, l'alternative la plus logique pour limiter le tir reste la variable d'instance. Il va simplement s'agir d'ajouter un attribut « canFire » à notre objet canon, pour savoir s'il peut tirer ou non.

Cliquez sur le canon, puis sur le volet de gauche, cliquez sur « Instance variables » pour en créer une toute nouvelle.

Le reste est identique à ce que nous avons fait avec la variable globale. Dans le groupe d'événements permettant de gérer le tir, vérifiez la valeur de l'attribut canFire.

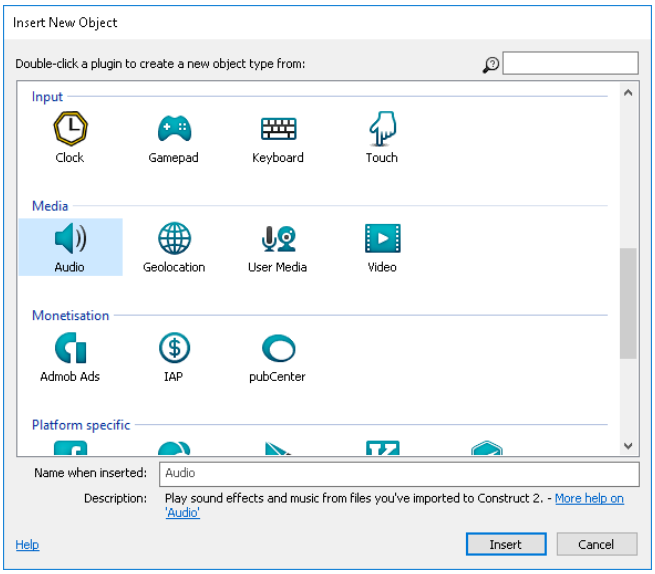




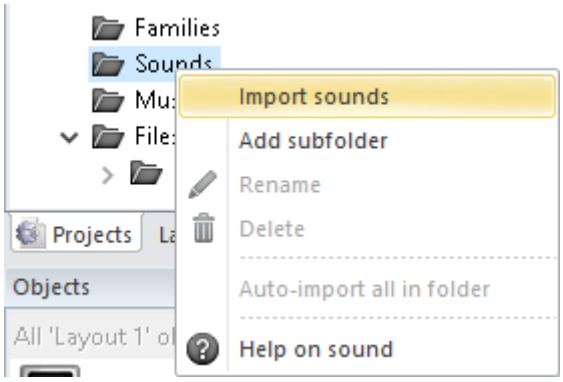


Améliorations 4 – Rajouter du son :

Dernière partie de ce petit cours. On va rajouter quelques sons au sein du projet. Ajoutez un objet Audio au projet.



Vous trouverez dans l'archive que je vous ai envoyé quelques exemples de sons utilisables. Ils doivent être en format WAV pour fonctionner convenablement. Faites importer ces sons.

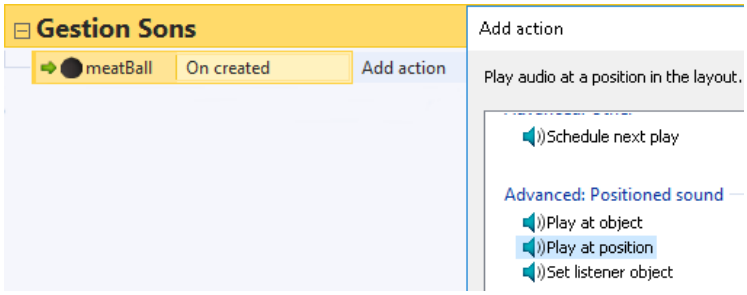


A noter que le dossier Sounds téléchargement complétement les sons avant de les jouer, alors que le dossier Music fonctionne en streaming. Ceci rejoint ce que nous avions vu en amphi.

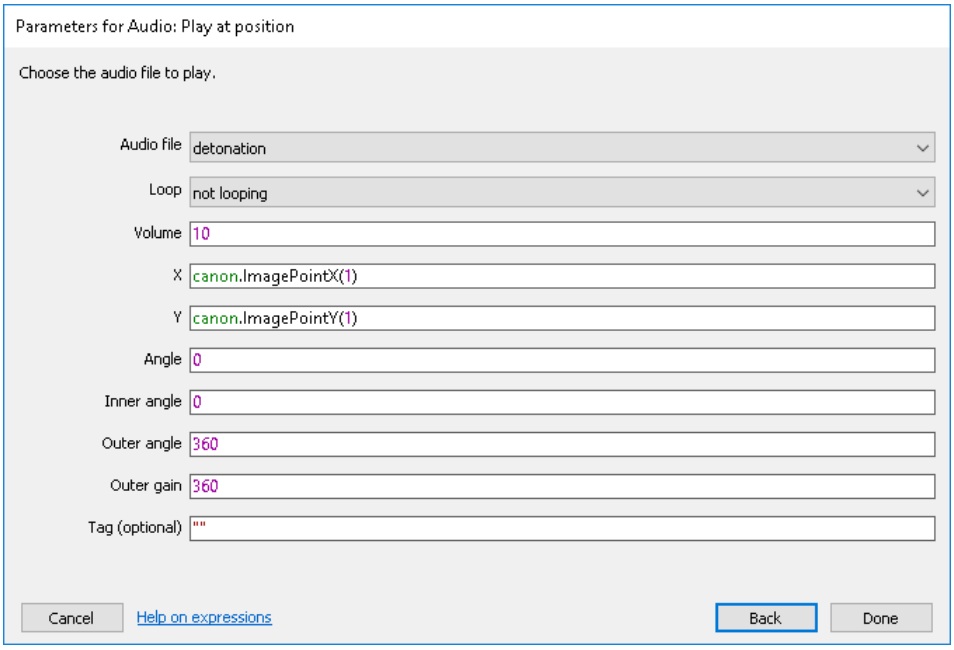
Il ne nous reste plus qu'à gérer nos sons depuis la feuille d'événement. Nous allons rajouter une détonation au canon et une musique d'ambiance. Si vous le souhaitez, vous pourriez aussi rajouter un ou plusieurs sons joués de façon aléatoire lorsque les écrans entrent en collision avec quelque chose.

Tutoriel – Physics

Ajoutez un nouveau groupe « Gestion sons »



Premièrement, quand un boulet est créé, on veut jouer le son « détonation » **une fois** à la position du canon. Plus précisément, sur l'image point correspondant à la zone de tir.



Pour la musique d'ambiance, nous n'avons pas à définir de position, puisque par définition, elle est ambiante. Elle doit juste se lancer dès le lancement du niveau. On obtient ceci :

Pour aller plus loin / Bibliographie :

Page du manuel sur les objets :

https://www.scirra.com/manual/68/objects

Page du manuel sur les événements :

https://www.scirra.com/manual/75/how-events-work

Page du manuel sur la feuille d'événements :

https://www.scirra.com/manual/121/event-sheets

Page du manuel sur les sous événements :

https://www.scirra.com/manual/128/sub-events

Page du manuel sur les commentaires :

https://www.scirra.com/manual/81/comments

Page du manuel sur les variables :

https://www.scirra.com/manual/83/variables

Page du manuel sur les groupes :

https://www.scirra.com/manual/80/groups

Meilleurs pratiques à adapter sur Construct :

https://www.scirra.com/manual/34/best-practices

Astuces générales d'optimisation :

https://www.scirra.com/manual/134/performance-tips

Page du manuel sur la physique :

https://www.scirra.com/manual/98/physics

Page du manuel sur le son :

https://www.scirra.com/manual/109/audio

Gestion de la lumière dynamique :

https://www.scirra.com/tutorials/9401/enlighten-your-games-with-a-dynamic-lighting/page-1