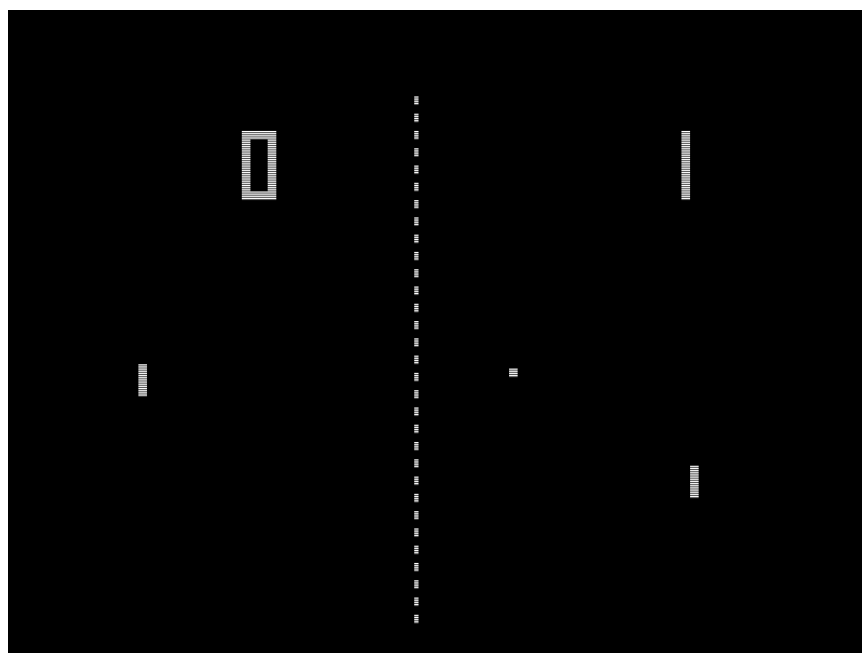


Tutoriel

CONSTRUCT 3

Série 'Classiques d'arcade'

PONG



ÉTAPE 0

UNE ANALYSE DU JEU

La plupart d'entre vous connaissez sûrement déjà le jeu Pong. En effet, il s'agit selon plusieurs du premier 'vrai' jeu vidéo tel que nous les connaissons aujourd'hui.

La version originale du jeu, sortie en 1972, se jouait uniquement à 2 joueurs, avec un système de contrôle plutôt primitif : une roulette!



La version de Pong que nous allons 'cloner' dans ce tutoriel ressemble beaucoup à la version originale, à quelques exceptions près :

- Il y aura un joueur humain qui affrontera l'ordinateur.
- Nous utiliserons le clavier pour contrôler notre joueur.
- Nous sommes libres d'utiliser plus de couleurs et de sons que la version originale.

Cependant, il sera relativement facile de modifier le jeu pour y inclure vos idées lorsque vous serez plus confortable avec le fonctionnement de Construct 3. Des idées d'améliorations et des défis sont proposés à la fin du document pour ceux qui voudraient aller plus loin.

Pong est, croyez-le ou non, un jeu de sport! En effet, comme son nom l'indique, il s'agit d'une simulation de tennis de table, sport aussi connu sous le nom de *ping-pong*.

Les joueurs sont représentés par les rectangles, la balle est un petit carré, et le score est affiché en haut de l'écran. Une ligne pointillée sépare le terrain en son milieu.

Au début d'un '*round*' (c'est-à-dire quand la balle est remise en jeu), la balle choisit aléatoirement une direction sur l'axe horizontal. Les joueurs se renvoient la balle jusqu'à ce qu'un but soit marqué. La balle est ensuite remise en jeu, jusqu'à temps qu'un des joueurs atteigne un certain nombre de points. Aussi, afin d'ajouter un peu d'action, la balle accélère légèrement lorsqu'un joueur réussit à renvoyer la balle vers son adversaire.

Il faut finalement noter que la direction que la balle prend lorsqu'elle entre en collision avec un joueur dépend de son point de contact sur la 'raquette'.

Si les mécaniques ne sont pas claires, vous pouvez ouvrir le projet Construct 3 qui est fourni et y jouer afin de vous familiariser avec le concept du jeu (et pour le plaisir, bien sûr!).

Allons-y!

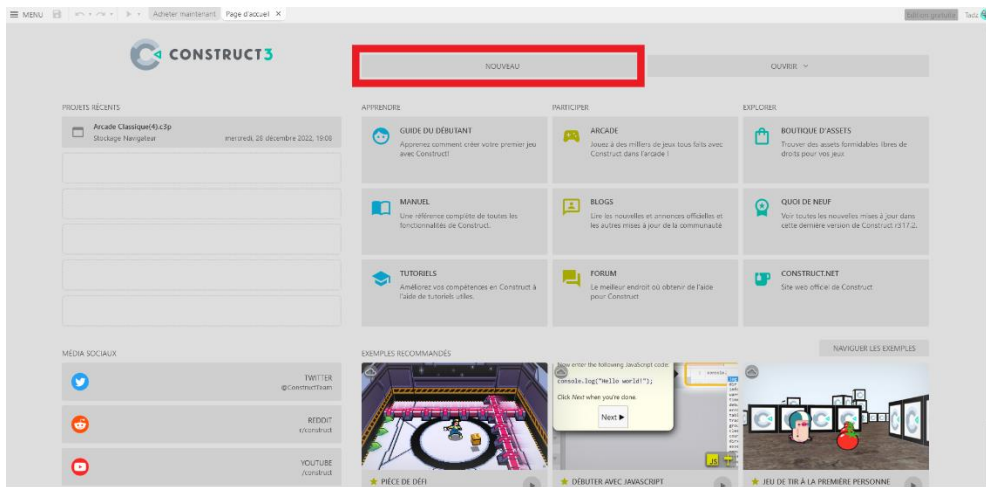
ÉTAPE 1

UN NOUVEAU PROJET CONSTRUCT 3

Premièrement, si ce n'est pas déjà fait, ouvrez un navigateur internet (comme Google Chrome ou Firefox), et tapez dans la barre d'adresse :

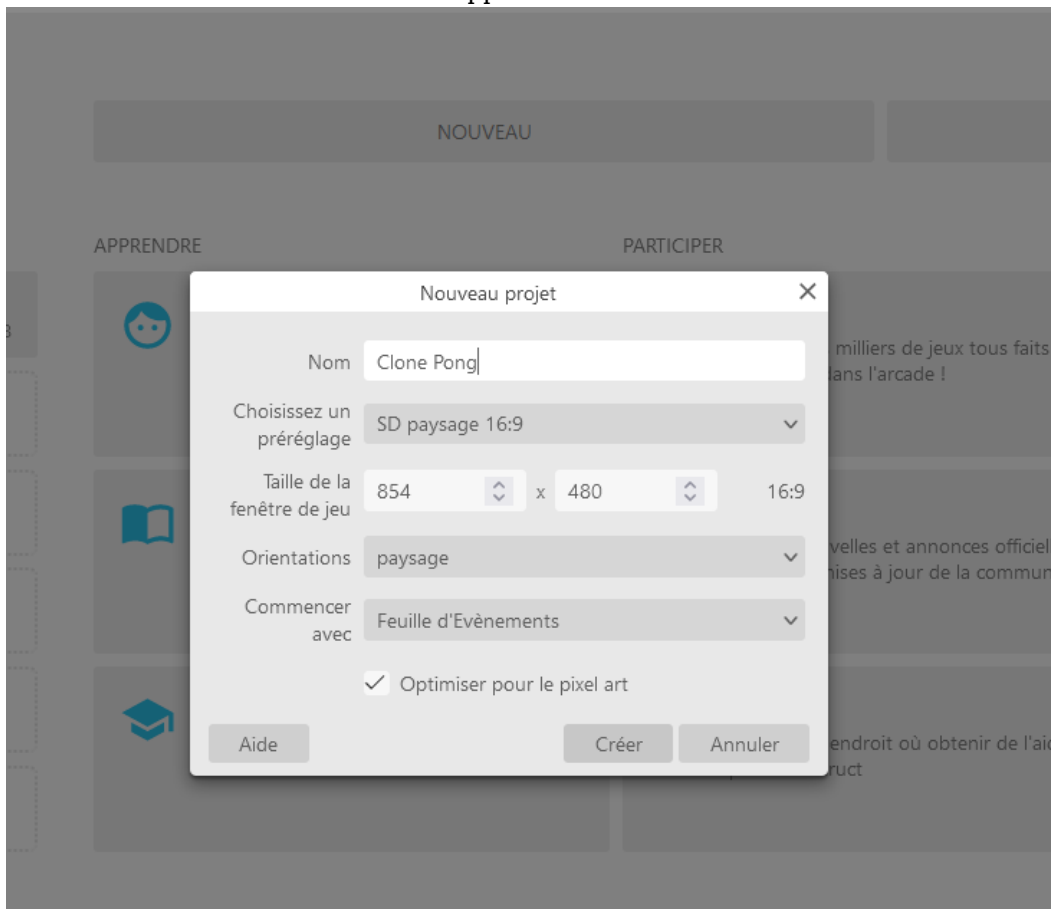
editor.construct.net

Ceci vous amènera à la page d'accueil de l'éditeur que nous utiliserons.



Nous pouvons ignorer les fenêtres pop-up qui peuvent apparaître. Il nous faut cliquer sur 'NOUVEAU'.

Une fenêtre comme celle-ci devrait apparaître.



➔ Donnons un nom à notre projet, comme 'Clone Pong', par exemple.

➔ Les autres paramètres peuvent être laissés à leur valeur par défaut (il est toujours possible de modifier ces valeurs une fois notre projet créé). Une fois ceci fait, cliquez sur 'Créer'.

Une fois le projet créé, vous devriez voir la fenêtre suivante :



La création du projet s'est bien déroulée, nous sommes maintenant prêts à commencer notre jeu!

******* TRUC DE PRO *******

Tenez la touche 'Ctrl' enfoncée et utilisez la molette de la souris pour modifier le 'zoom' dans l'éditeur.

ÉTAPE 2

L'OBJET JOUEUR

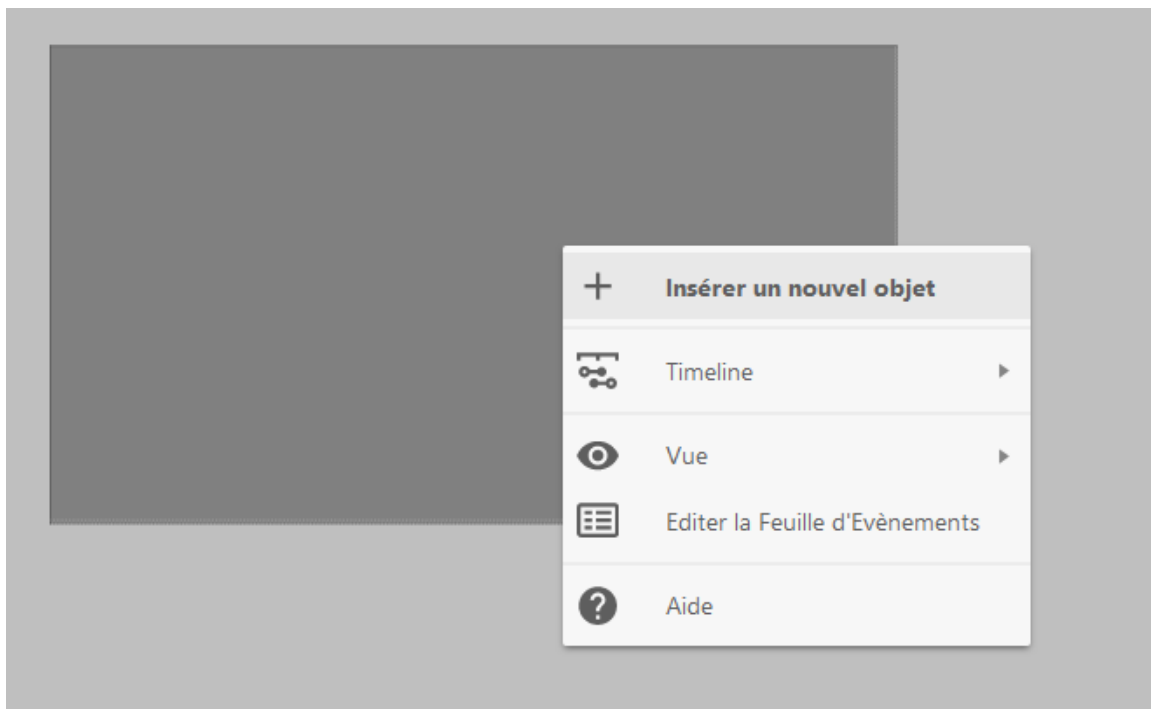
Comme dans tous les jeux, notre joueur doit pouvoir avoir le contrôle sur un objet. Dans le cas de Pong, le joueur est représenté par sa raquette de Ping-Pong, qui consiste, rappelons-nous, en un très artistique rectangle blanc.

Presque tout ce qui existe dans Construct 3 est considéré comme un *Objet*. Un *sprite*, un arrière-plan, un ennemi, une manette de jeu, la souris et le clavier, etc.

Comme vous allez bientôt le voir, il existe une multitude de types d'*Objets* disponibles avec Construct 3, mais nous allons généralement n'en utiliser que quelques-uns.

Il existe plusieurs façons de créer un nouvel objet, mais la plus simple est d'effectuer un clic-droit à l'intérieur de notre scène, et de cliquer sur 'Insérer un nouvel objet'.

Il est important de se familiariser avec la procédure de création d'objet : cette opération reviendra souvent!

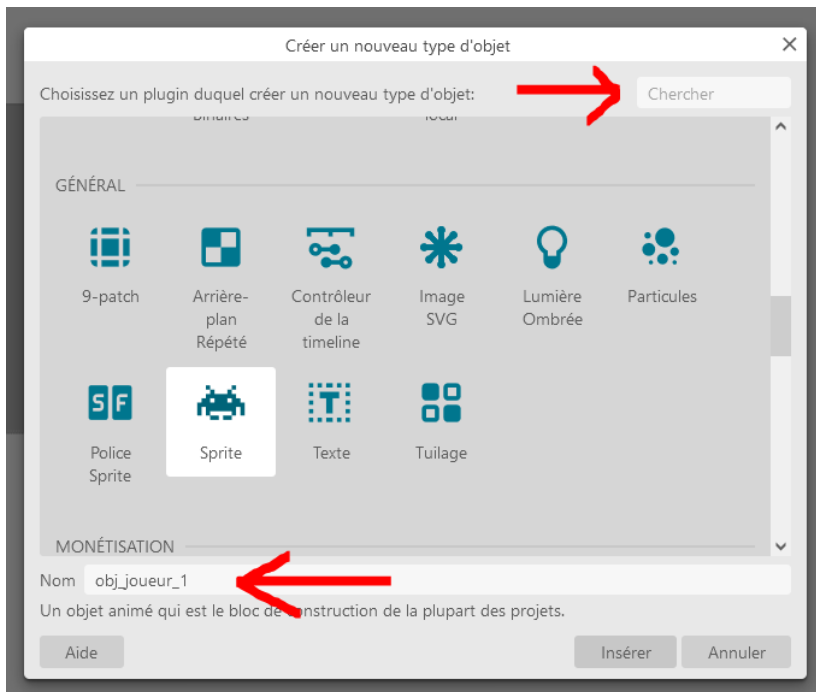


Une fenêtre apparaît automatiquement, avec tous les types d'objets disponibles. Avec la barre du menu déroulant, trouvez la catégorie '*Général*' et choisissez '*Sprite*'.

***** N'ALLEZ PAS TROP VITE! *****

Pour avoir une bonne expérience quand on crée un jeu, il est important de bien organiser notre projet afin de ne pas se perdre à travers toutes les ressources différentes. Il nous faut bien nommer chacun de nos objets afin de savoir son rôle. Personnellement, j'utiliserai le préfixe **obj_*** pour définir le nom de chacun de mes objets. Les sons seront, par exemple, préfixés par **snd_***, mais nous y reviendrons.

Ici, notre objet est (poétiquement) nommé `obj_joueur_1`



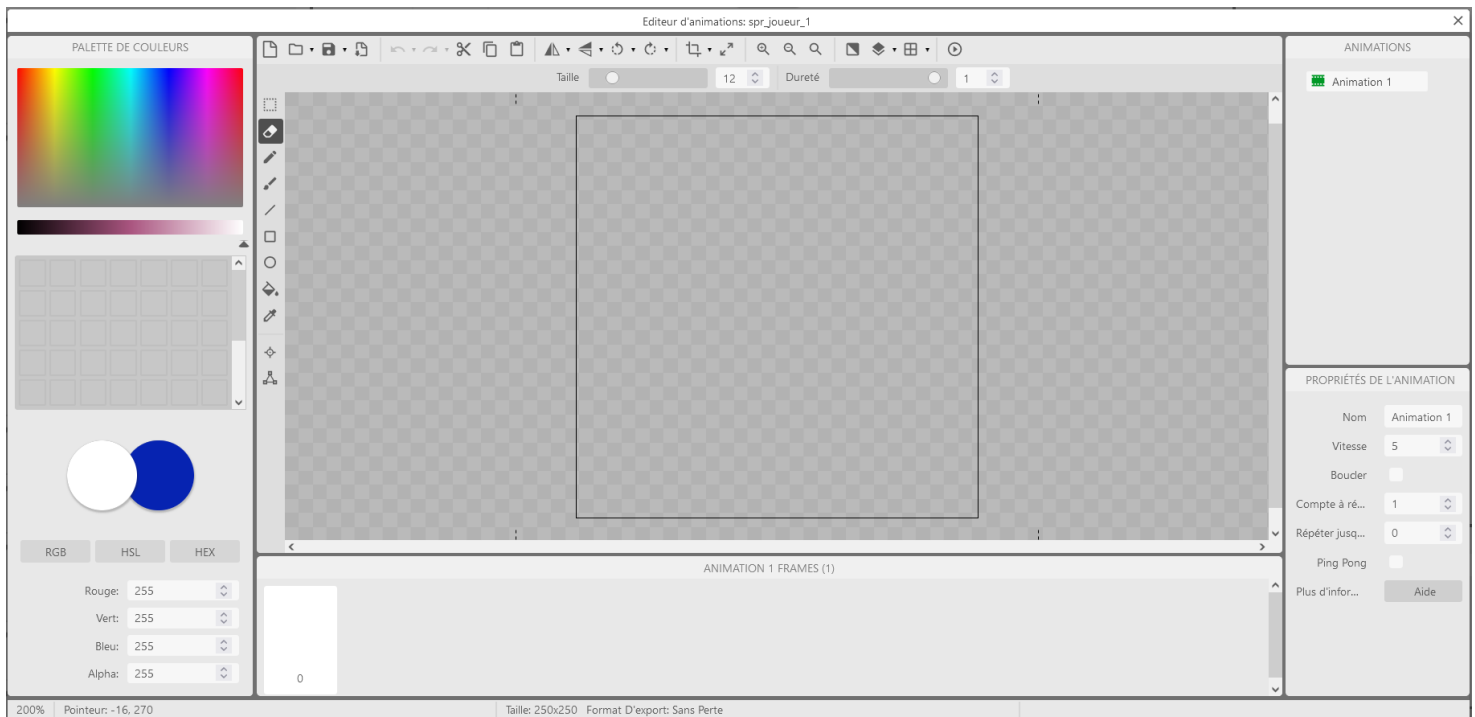
Notez aussi la barre de recherche dans le coin supérieur droit. Très utile pour retrouver rapidement un type d'objet particulier lors de la création!

Nous pouvons maintenant cliquer sur 'Insérer' pour créer notre joueur.

***** QUE SE PASSE-T-IL ?! ? *****

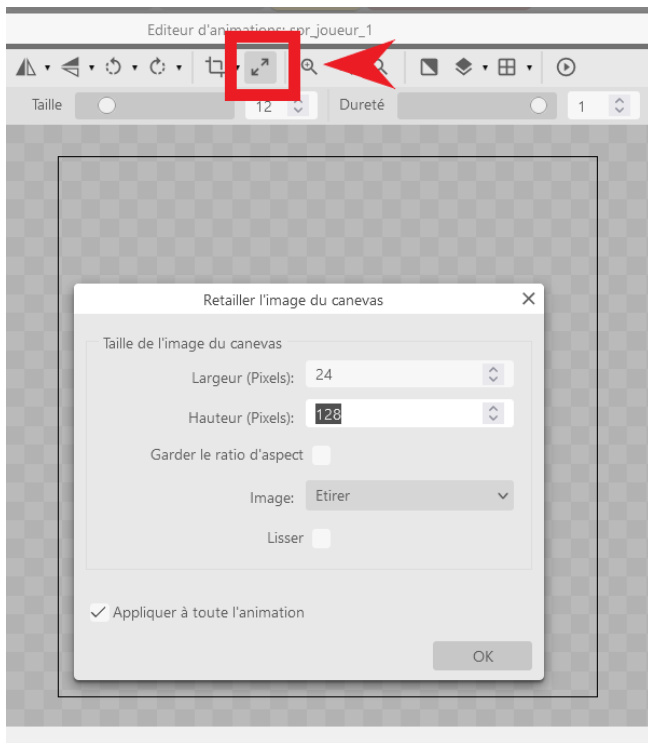
Nous sommes revenus sur l'écran de notre scène, et rien ne semble avoir changé! En fait, si vous regardez attentivement, le curseur de votre souris s'est transformé en 'croix'. Construct veut que nous placions notre objet dans la scène. Nous n'avons qu'à cliquer n'importe où sur notre scène (l'endroit n'a pas d'importance).

Après avoir cliqué à l'endroit où créer votre objet, cet écran devrait alors apparaître :



Nous voyons apparaître un carré transparent. La barre d'information au bas de la page indique que la taille de notre *sprite* est de 250 x 250 pixels. C'est beaucoup trop gros! Rappelez-vous que notre scène ne fait que 480 pixels de hauteur !

Pour modifier la taille de notre *sprite*, il suffit de cliquer sur l'icône des deux flèches diagonales qui vont en direction opposées, sur la barre de menu, en haut, au centre.

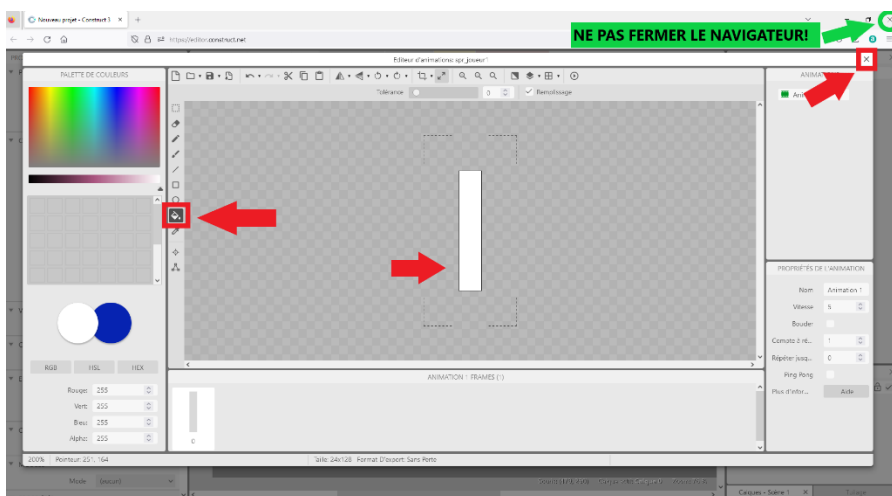


Dans notre cas, une taille de 24 x 128 pixels me semble bonne, mais vous pouvez expérimenter pour trouver des valeurs qui vous semblent meilleures. Ceci-dit, je suggère de faire ce genre de modification une fois les mécaniques fonctionnelles.

Une fois les dimensions entrées, cliquez 'OK'.

****Note, les autres paramètres n'ont pas d'importance ici, leur effet ne sera pas visible sur des *sprites* de forme carrée qui n'ont qu'une seule couleur, comme c'est notre cas.**

Une fois les dimensions ajustées, sélectionnons le mode '*Remplissage*', soit en appuyant sur son icône, ou avec le raccourci-clavier 'F'. Une fois l'outil de remplissage sélectionné, peignez tout votre sprite en cliquant à l'intérieur de celui-ci. Une fois terminé, fermez simplement la fenêtre du sprite avec le 'X'.



******* ATTENTION! *****
NE FERMEZ PAS VOTRE NAVIGATEUR
WEB PAR ERREUR!**

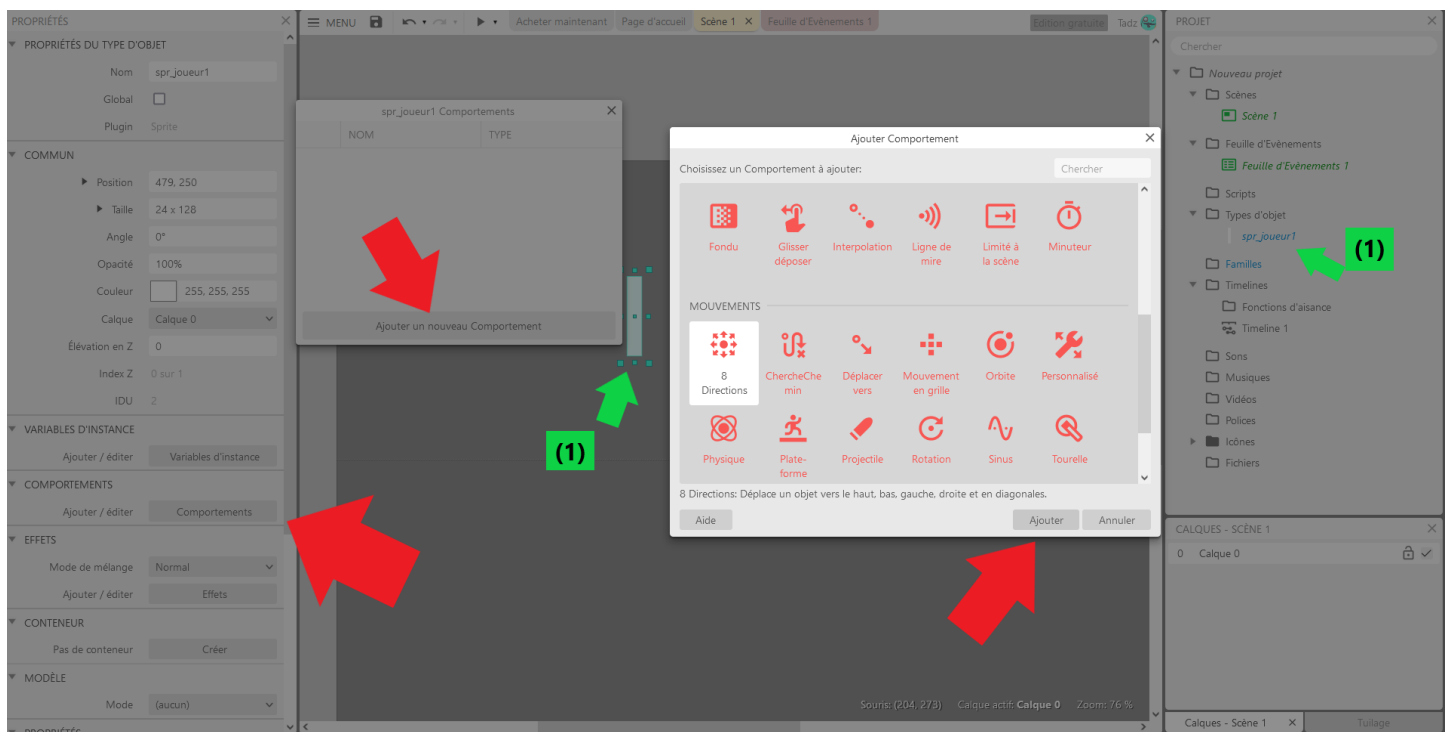
Bien! Le sprite de notre joueur devrait maintenant être visible dans notre scène!
Il nous faut maintenant lui ajouter la possibilité de se déplacer de haut en bas.

Sélectionnez votre objet joueur en cliquant dessus, soit dans la scène, ou bien dans le panneau 'Projet', à droite (sous le fichier 'Types d'objets' (voir les flèches vertes sur l'image suivante).

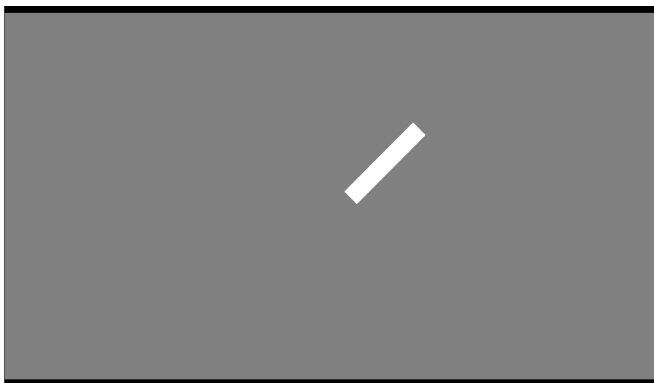
Avec votre obj_joueur_1 sélectionné, trouvez la section 'Comportements' dans le panneau 'Propriétés' (à gauche de l'écran). Cliquez ensuite sur le bouton 'Comportements'.

Un popup apparaît avec une liste vide. Au bas du popup, cliquez sur 'Ajouter un nouveau Comportement'.

Un nouveau popup apparaît avec une liste d'icônes orange. Trouvez le comportement '8 Directions', qui se trouve sous la catégorie 'Mouvements'. Vous pouvez aussi utiliser la barre de recherche. Une fois le comportement '8 Directions' sélectionné, cliquez sur le bouton 'Ajouter' et fermez le popup qui reste.



Construct a maintenant assigné le comportement '8 Directions' à notre joueur. Pour s'assurer que tout a bien fonctionné, vous pouvez appuyer sur le bouton 'Play' de la barre de menu, ou bien utiliser le raccourci-clavier 'F4' pour tester. Utilisez les flèches et observez le comportement de notre raquette!



Nous pouvons voir que notre joueur peut se déplacer, comme l'indique le nom du comportement, dans 8 directions différentes!

Bien amusant, mais ce n'est pas tout à fait ce que l'on veut pour notre clone de *Pong*.

Heureusement, il existe un moyen très simple de restreindre le mouvement du joueur.

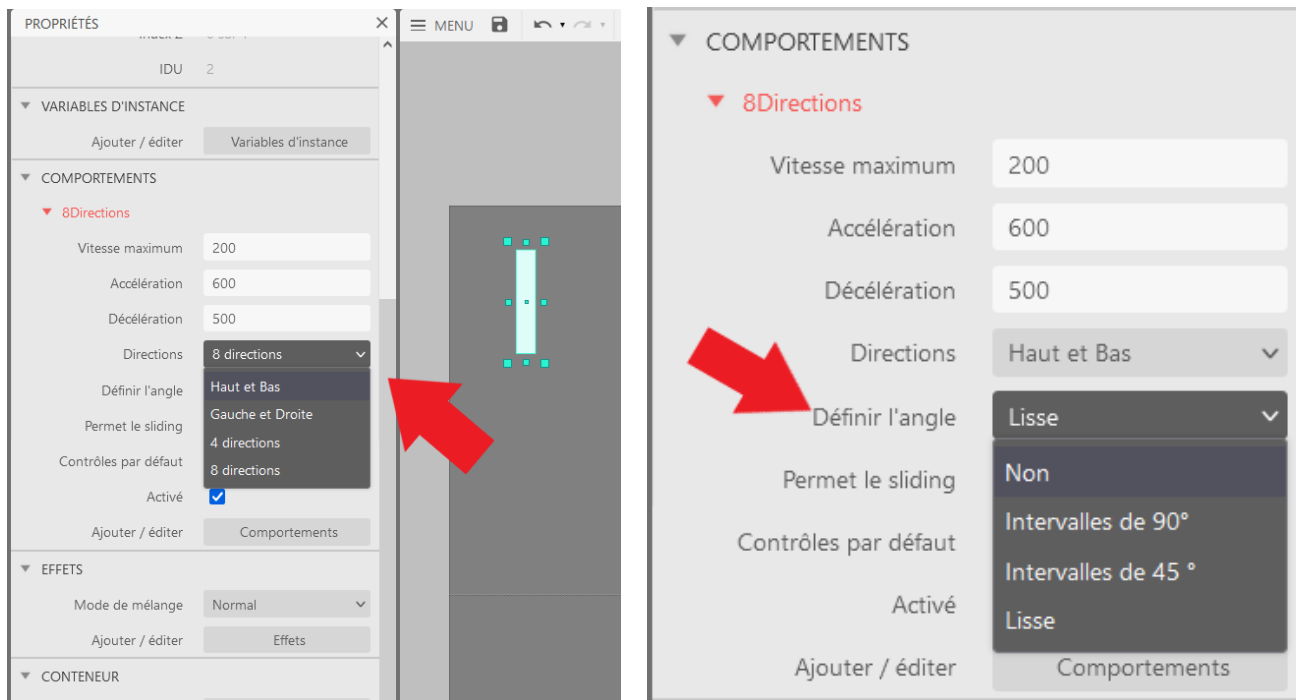
Fermons la fenêtre de jeu, et allons inspecter le panneau 'Propriétés' de notre joueur. Nous voyons qu'il y a de nouvelles options disponibles sous la section 'Comportements'. C'est ici que l'on pourra modifier les comportements ajoutés.

- 1) Trouvez l'option '*Directions*' et ouvrez son menu déroulant.
- 2) Sélectionnez '*Haut et Bas*'.
- 3) Appuyez sur F4 (ou le bouton '*Play*'), et observez la réaction de l'objet joueur!

Oh...

Le joueur se déplace de haut en bas, mais son orientation est décalée de 90 degrés. La raison est que, par défaut, Construct tourne automatiquement notre sprite dans la direction dans laquelle se déplace l'objet. Dans notre cas, ce n'est pas ce que l'on veut, car les raquettes doivent rester à la verticale. On ne veut pas 'regarder en haut' lorsqu'on se déplace vers le haut, et vice-versa.

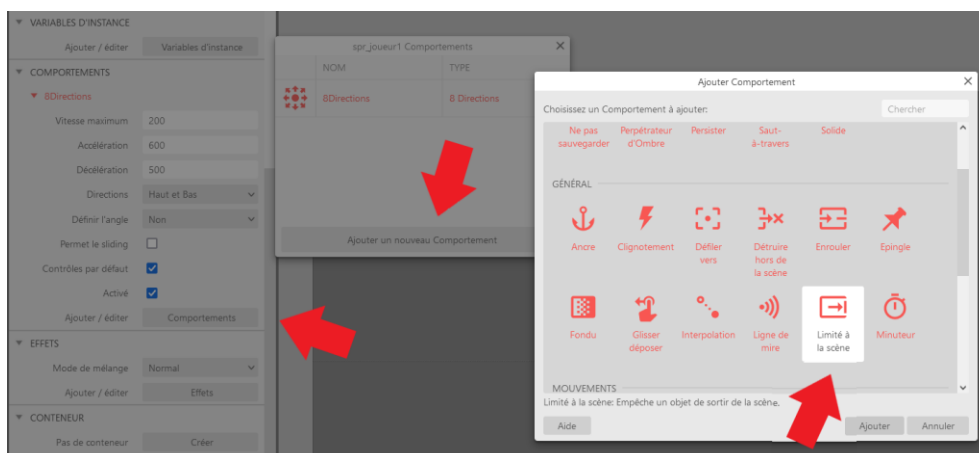
Nous pouvons facilement modifier cet aspect du comportement en mettant l'option '**Définir l'angle**' à 'Non'



Nous pouvons tester encore une fois, avec '*F4*' ou le bouton '*Play*'. Cette fois, ça y est!

Enfin...presque, car il y a encore un problème majeur : le joueur peut sortir de la scène et de déplacer à l'infini (et plus loin encore) vers le haut et vers le bas. Mais ne paniquons pas car, comme vous commencez peut-être déjà à douter, Construct nous offre une solution clé-en-main extrêmement simple.

Ajoutons un nouveau comportement avec la même procédure que nous avons fait pour ajouter '*8 Directions*'. Cette fois,



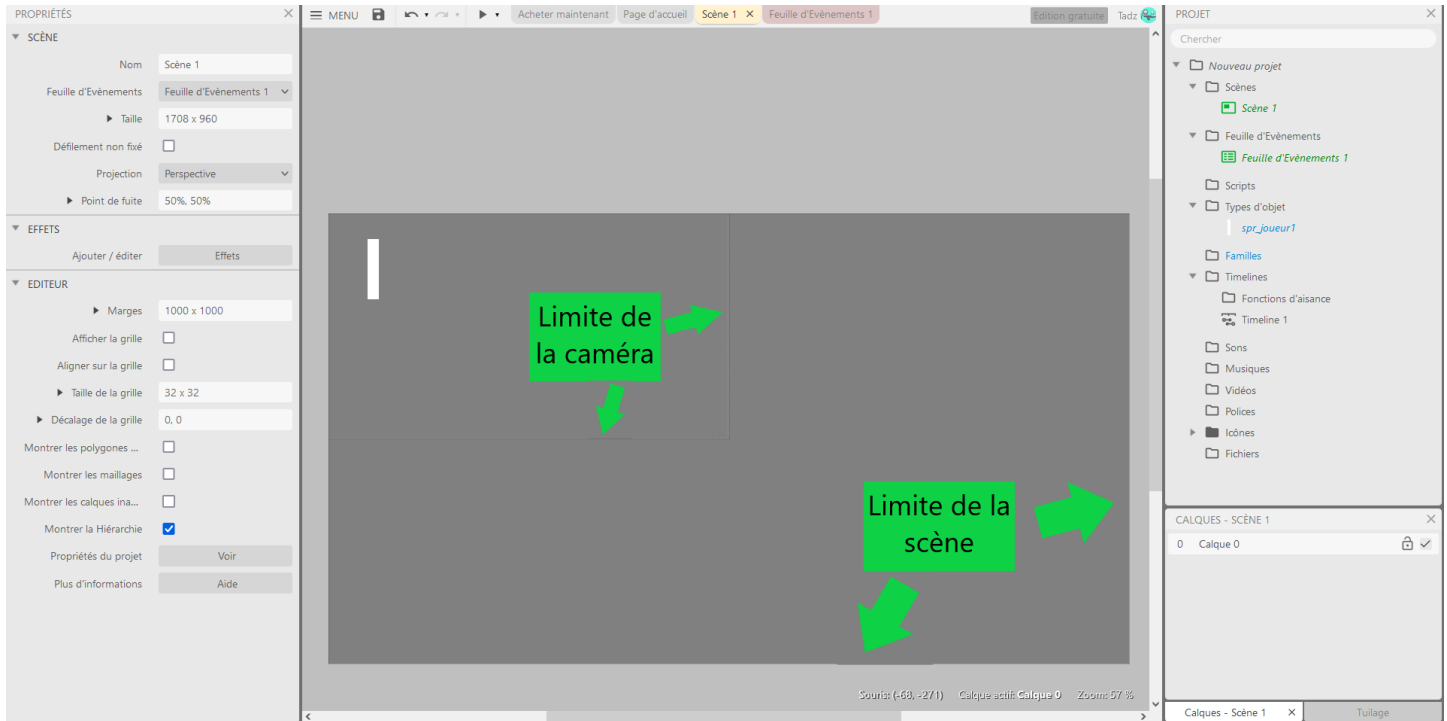
trouvons le comportement nommé '*Limité à la scène*' et cliquez sur '*Ajouter*'. Les options par défaut du comportement sont celles que l'on veut pour notre jeu, nous n'avons donc pas besoin de les modifier. Testons notre jeu encore une fois...
...Oh! Zut!

Notre joueur ne peut plus se déplacer hors de la scène vers le haut, mais il disparaît toujours vers le bas !!! La raison est bien simple...

ÉTAPE 3

LA CAMÉRA ET LA SCÈNE

Techniquement, notre comportement '*Limité à la scène*' fonctionne comme il devrait. La source de notre bug se trouve en fait dans la caméra. En effet, en ce moment notre caméra ne 'regarde' qu'une petite section de notre scène. Nous pouvons voir ceci dans l'éditeur : le camp de vision de la caméra est représenté par les lignes pointillées.



Dans le cas de *Pong*, le champ de vision de la caméra est identique aux dimensions de la scène. Cependant, rien n'oblige un jeu à avoir une caméra aux mêmes dimensions que la scène. Par exemple, le niveau 1-1 de *Super Mario Bros.* fait environ 3860 pixels de large, mais la caméra ne voit que de 256x240 pixels à la fois! Si ce jeu était fait dans Construct 3, la scène ressemblerait à ceci. Remarquez à droite la section souterraine du niveau 1-1 pour vous donner une référence de la largeur 'd'un écran' par rapport à la largeur de la scène.



Laissons les *Goombas* de côté pour revenir à nos pongistes (oui, c'est le nom donné aux joueurs de ping-pong, et se prononce comme 'éponge', et non comme le jeu *Pong*).

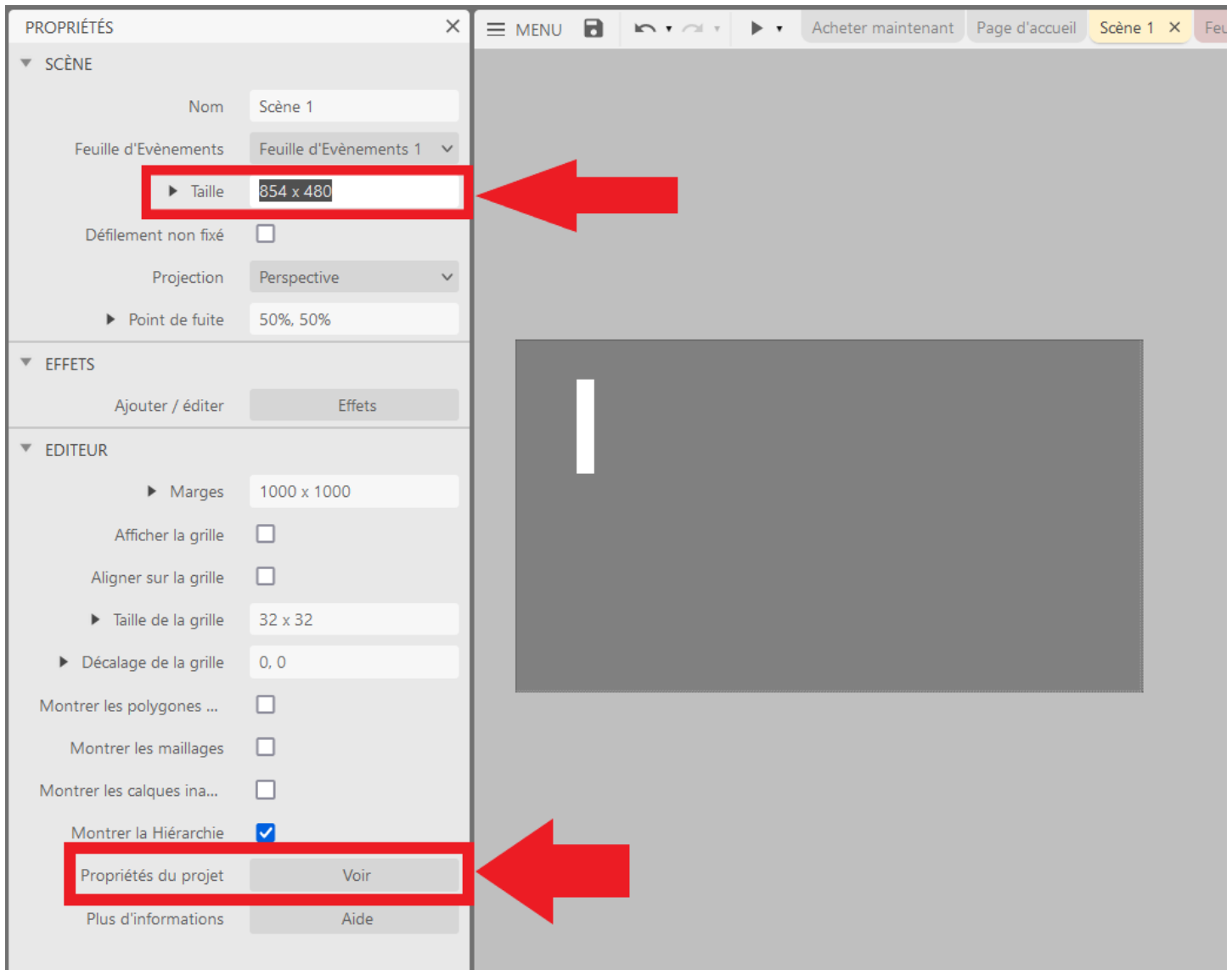
Le plus logique dans notre cas est de redimensionner la scène aux mêmes dimensions que le champ de vision de la caméra. Rappelez-vous à la création du projet, les valeurs par défaut étaient de 854x480 pixels.

Puisque les graphiques de ce jeu sont très primitifs, nous n'avons pas besoin d'une grande résolution. Il est important de comprendre que l'utilisateur peut agrandir, rapetisser ou mettre le jeu en mode plein-écran, s'il le désire. Dans ce cas, notre scène sera automatiquement 'étirée' pour s'adapter à l'écran du joueur. Il est donc totalement possible de jouer en plein écran sur un moniteur de haute résolution (1920 x 1080 pixels, et plus) un jeu dont la résolution originale est (960 x 540), par exemple.

Cliquons sur un endroit vide sur notre scène pour ne pas avoir notre joueur sélectionné. Le panneau de propriétés nous montre maintenant les propriétés de la scène elle-même! Trouvez à gauche, dans le panneau '*Propriétés*', sous

la catégorie 'Scène' (la première de la liste) le champ 'Taille' et entrez comme valeur 854x480, et appuyez sur la touche 'Entrée'.

Si vous avez utilisé une autre valeur pour la taille de fenêtre que la taille par défaut, vous pouvez cliquer sur le bouton 'Voir' de 'Propriétés du projet' de la section 'Éditeur'. Cliquez ensuite sur la scène pour ajuster sa taille aux bonnes dimensions.



Testons encore, et cette fois ça y est! Notre joueur se déplace uniquement de haut en bas sans 'sortir' de l'écran!
Enfin !!!

C'est tout ce que nous avons à faire pour le moment avec notre objet joueur!

******* ATTENTION *******

C'est le moment parfait pour faire une sauvegarde de votre projet! Utilisez le raccourci *Ctrl+F* ou bien cliquez sur l'icône de disquette dans la barre de menu. Il existe plusieurs options, mais l'option 'Fichier local' est la plus simple : elle télécharge une copie de votre projet sur votre ordinateur.

Passons maintenant à la prochaine étape : la balle!

ÉTAPE 4

LA BALLE

Il est important de comprendre que lorsque l'on programme un jeu, ou n'importe quel logiciel, il existe la plupart du temps plusieurs manières d'obtenir un résultat similaire. Il est peut-être ambitieux de dire que *tous* les chemins mènent à Rome, mais nous pouvons très certainement affirmer que *plusieurs* chemins se rendent à Rome!

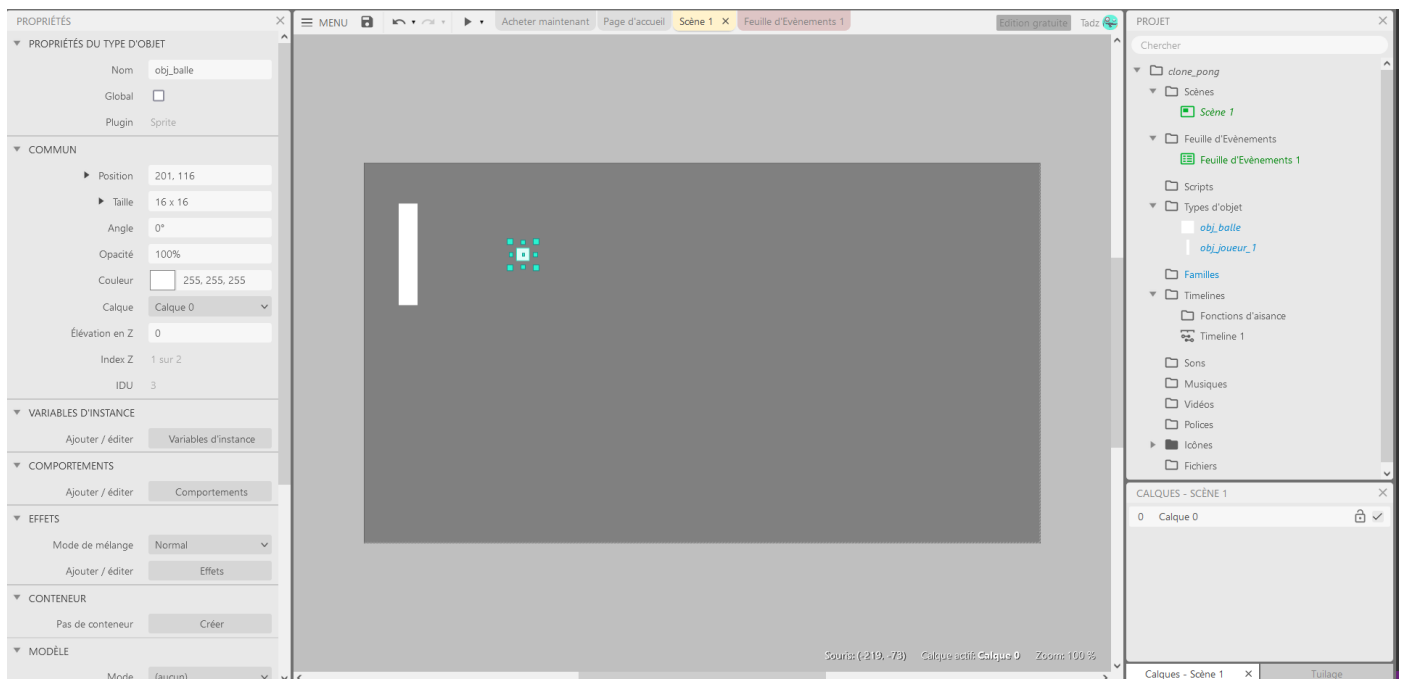
Nous allons donc adopter une stratégie différente pour notre balle : nous allons utiliser la *Feuille d'évènements* afin de la programmer.

Nous avons quand même besoin d'un objet, car rappelez-vous que tout dans Construct est un objet.

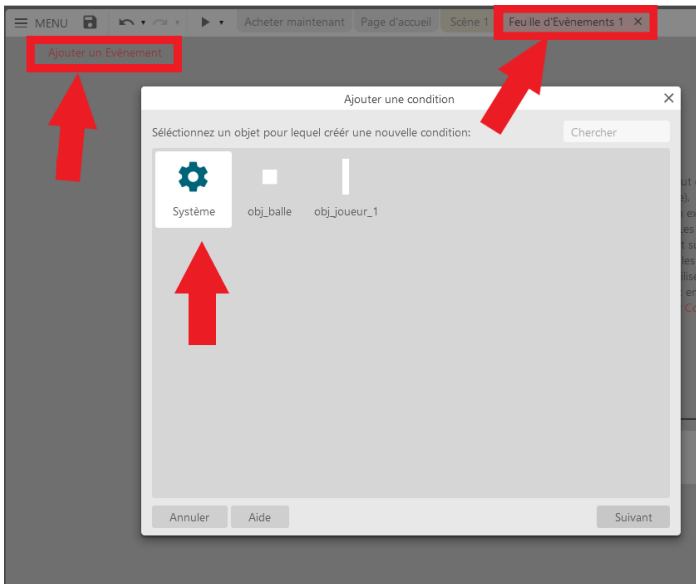
Même procédure que lorsque nous avons créé notre joueur :

1. Clic-droit sur un endroit vide de la scène
2. Sélectionnez '*Insérer un nouvel objet*'
3. Choisissez l'objet '*Sprite*'
4. N'oubliez pas de donner un nom significatif à notre balle. Ici, nous suivrons la même convention que pour notre joueur et la nommerons **obj_balle**
5. Cliquez n'importe où sur la scène pour placer la balle (l'endroit n'a pas d'importance, vous verrez pourquoi bientôt)
6. Redimensionnez la taille du sprite. Une bonne valeur de départ est 16x16.
7. Peinturez votre sprite en blanc (ou de la couleur de votre choix), et fermez l'éditeur de sprite.

À ce point, vous devriez avoir un écran semblable à celui-ci.



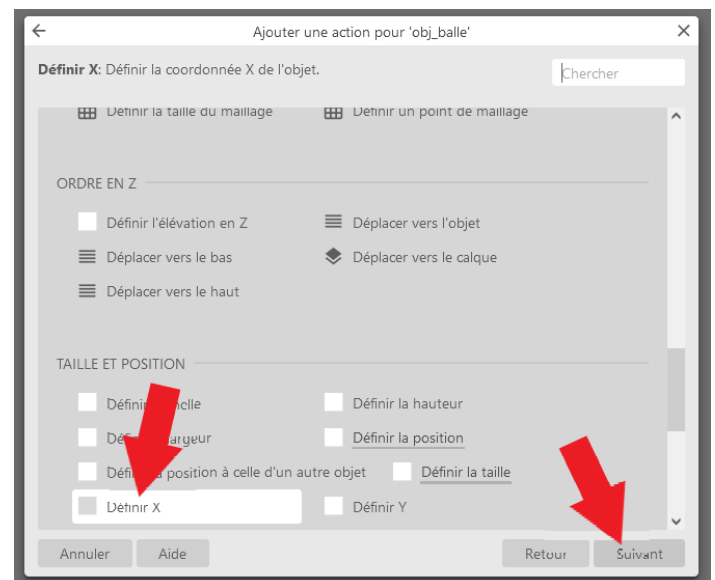
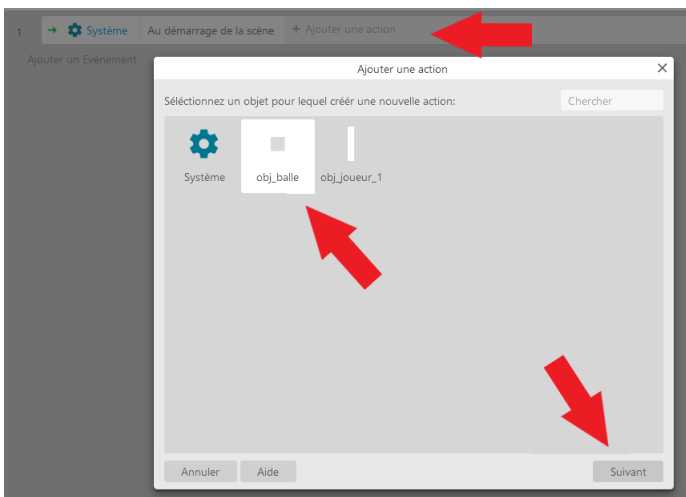
Notre objet créé, allons maintenant :



- 1) Cliquer sur l'onglet '**Feuille d'Événements 1**' dans la barre de menu, en haut de l'éditeur (l'onglet rose).
- 2) Sous l'icône de sauvegarde, cliquez sur '**Ajouter un événement**'.
- 3) Choisissez '**Système**' (et non pas la balle!)
- 4) Faites '**Suivant**'
- 5) Dans le menu, trouvez et sélectionnez l'option '**Au démarrage de la scène**'.
- 6) Cliquez sur '**Suivant**'.

Nous venons de définir une condition, ce qui veut dire que lorsque cette condition sera 'vraie', toutes les actions dans son bloc seront exécutées. Ici, notre condition est '**Au démarrage de la scène**'. Alors, que voulons-nous faire lorsque notre scène démarre? Eh bien, la réponse est : plusieurs choses! Mais commençons doucement, et tentons de placer notre balle au centre de l'écran quand notre jeu démarre.

- 1) Dans notre événement Système → Au démarrage de la scène que nous venons de créer, nous allons cliquer sur '**Ajouter une action**'.
- 2) Sélectionnez **obj_balle**, et cliquez '**Suivant**'
- 3) Trouvez la catégorie '**Taille et Position**'
- 4) Sélectionnez Définir X
- 5) Cliquez sur '**Suivant**'



Un *popup* apparait, ou il faut définir la valeur que nous voulons donner à X, qui est l'axe horizontal.

Bien entendu, pour centrer la balle, nous pourrions simplement entrer la valeur 427 (car notre scène fait 854 pixels de large), mais il existe une meilleure façon. Nous allons demander à Construct de nous donner la largeur de la scène automatiquement. Ce faisant, nous pourrions aussi modifier la taille de la scène plus tard sans 'briser' notre code.

Alors, au lieu d'entrer une valeur numérique dans le champ X, nous allons :

1) Cliquer sur '**Trouver une expression**'.

2) Sélectionnez '**Système**'.

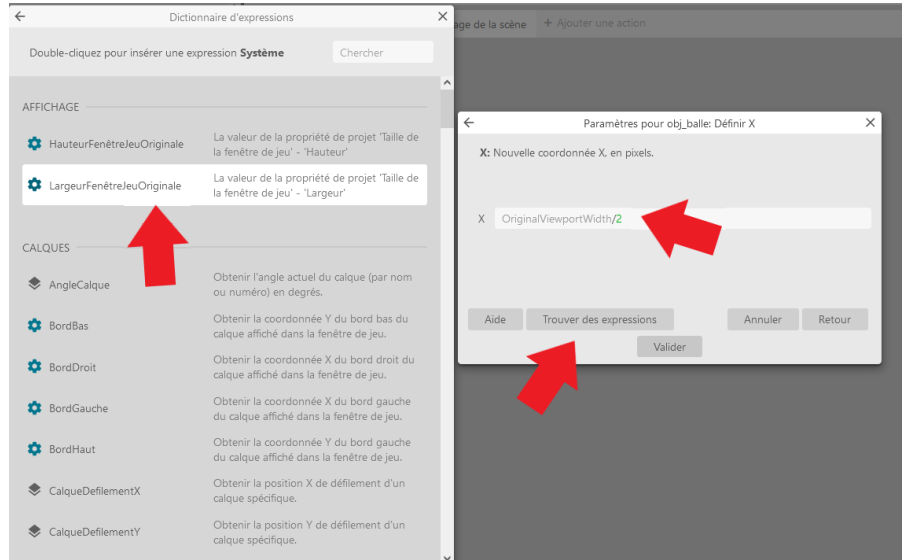
3) Sélectionnez '**LargeurFenêtreJeuOriginale**' en double-cliquant.

4) Le texte *OriginalViewportWidth* apparait dans le champ de la valeur X.

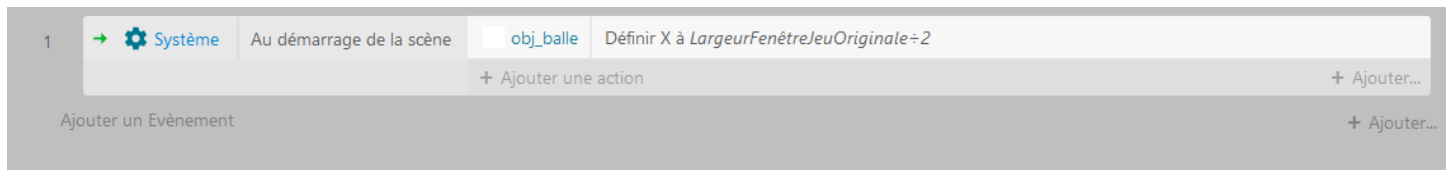
5) Nous pouvons simplement diviser la valeur par 2 en ajoutant /2 à la fin de la variable.

6) Le résultat devrait être celui-ci -->

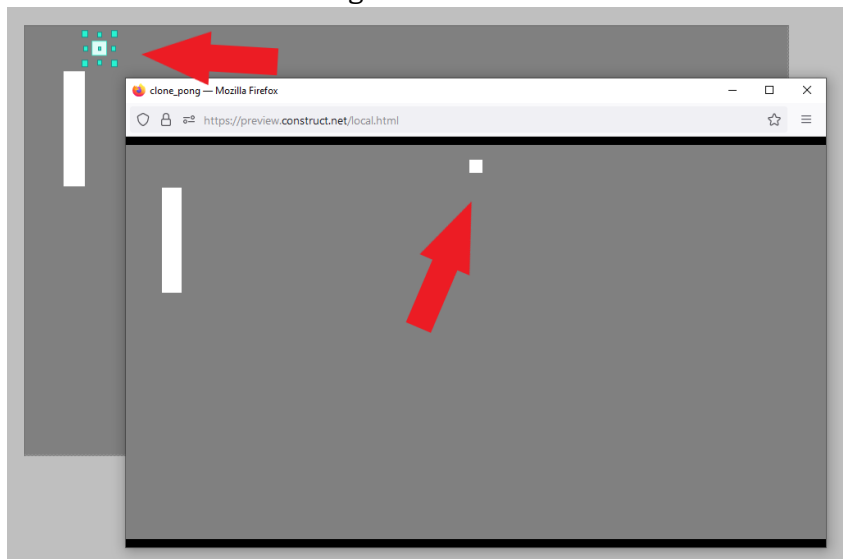
7) Cliquez sur 'Valider'



Votre évènement devrait ressembler à ceci. Si ce n'est pas le cas, revenez en arrière et essayez de repérer votre erreur.



Faisons le test. Retournons dans notre scène en cliquant sur l'onglet 'Scène 1' et plaçons notre balle à droite ou à gauche de la scène. Appuyons sur F4 pour voir que la balle se place automatiquement au centre de la scène sur l'axe horizontal au démarrage.



Maintenant, c'est à votre tour de travailler! Notre balle est centrée sur l'axe des X, mais elle n'est pas centrée verticalement sur l'axe des Y. Sous la même condition 'Système -> Au démarrage de la scène' que nous avons créé, ajoutez une seconde action qui placera la balle au centre de la scène sur l'axe des Y.

Pour rappel, les étapes sont les suivantes :

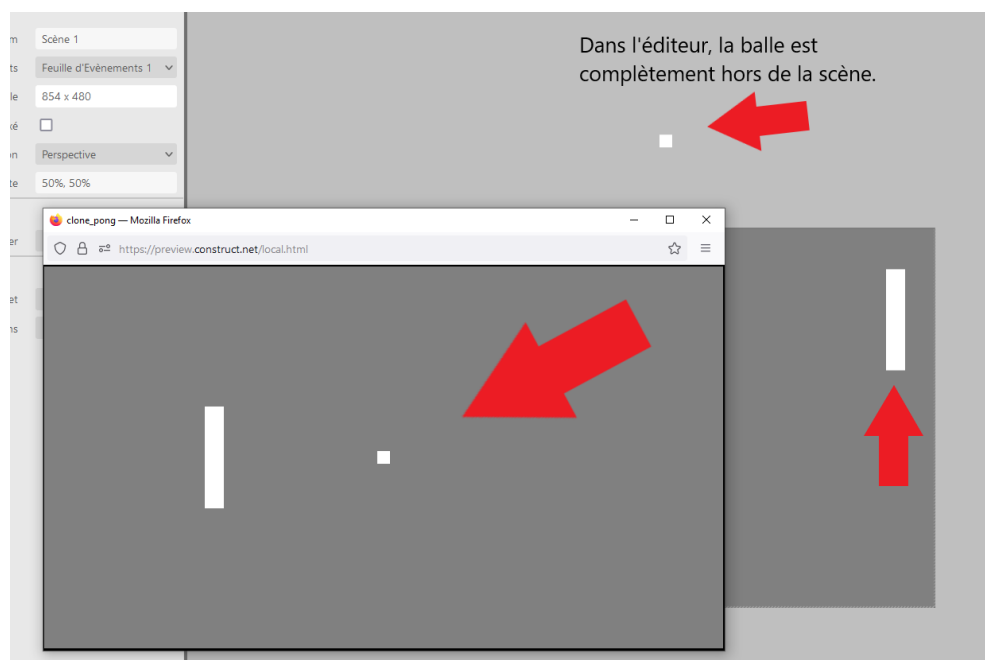
- 1) Cliquez sur '**Ajouter une action**'.
- 2) Sélectionnez **obj_balle**, et cliquez 'Suivant'
- 3) Trouvez la catégorie '*Taille et Position*'
- 4) Sélectionnez Définir Y
- 5) Cliquez sur 'Suivant'
- 6) Cliquez sur '**Trouver une expression**'.
- 7) Sélectionnez '**Système**'.
- 8) Sélectionnez 'HauteurFenêtreJeuOriginale' en double-cliquant.
- 9) Divisez par 2, puis validez.

Tant qu'à être dans le bain, faisons aussi la même chose pour placer notre **obj_joueur_1**. La position de départ du joueur sur l'axe Y (vertical) devrait être au centre de la scène, mais sa position horizontale ne l'est pas. Nous allons la placer au **quart** de la largeur de la scène, en divisant la valeur par 4 au lieu de 2. Utilisez le même principe que précédemment, mais remplacez l'étape 2) par **obj_joueur_1** au lieu de **obj_balle**, puisque c'est la position de joueur que nous voulons assigner cette fois.

Après ceci, votre événement 'Système -> Au démarrage de la scène' devrait être comme ceci :



Nous pouvons tester de la même façon que précédemment : retournons dans l'onglet '**Scène 1**' et plaçons notre balle et notre joueur à droite de la scène. Vous pouvez même les placer complètement hors des limites de la scène! Appuyez sur **Play** (ou la touche F4), et comme par magie, notre joueur et notre balle sont automatiquement positionnés au bon endroit! Tout cela nous donne un code beaucoup plus robuste et flexible que simplement positionner manuellement nos objets.



*** ATTENTION ***

C'est un autre excellent moment pour sauvegarder notre travail et prendre une pause si vous en sentez le besoin!

Maintenant que la balle et le joueur sont positionnés, il nous faut déterminer (et programmer!) la 'logique' que la balle devrait avoir, c'est-à-dire :

1. Elle doit 'rebondir' lorsqu'elle frappe la raquette d'un joueur.
2. Elle doit 'rebondir' lorsqu'elle frappe un mur.
3. Elle doit se replacer au centre de l'écran lorsqu'un but est marqué.

Il y a du pain sur la planche!

Premièrement, il faut créer des **variables** pour notre objet **obj_balle**.

*** Variables ***

Une variable, c'est simplement un mot qui est associé à une valeur dans le code. Par exemple, dans un jeu d'action, notre joueur pourrait avoir une variable 'points_de_vie' dans laquelle se trouverait les points de vie qui lui reste. De cette manière, nous pouvons avoir du code plus lisible et facile à modifier. Avec l'exemple précédent, nous pourrions par exemple dire à Construct :

- ➔ Si la variable points_de_vie est égale ou plus petite que zéro, montrer l'écran 'Game Over'
- ➔ Si un ennemi frappe notre joueur, soustraire les dommages à points_de_vie
- ➔ Etc.

Construct nous offre 3 types de variables :

- ➔ Nombre
- ➔ Texte
- ➔ Booléen

*** Booléen ***

Une variable booléenne, plus connue sous le nom de **bool**, est une variable qui peut seulement avoir 2 valeurs :

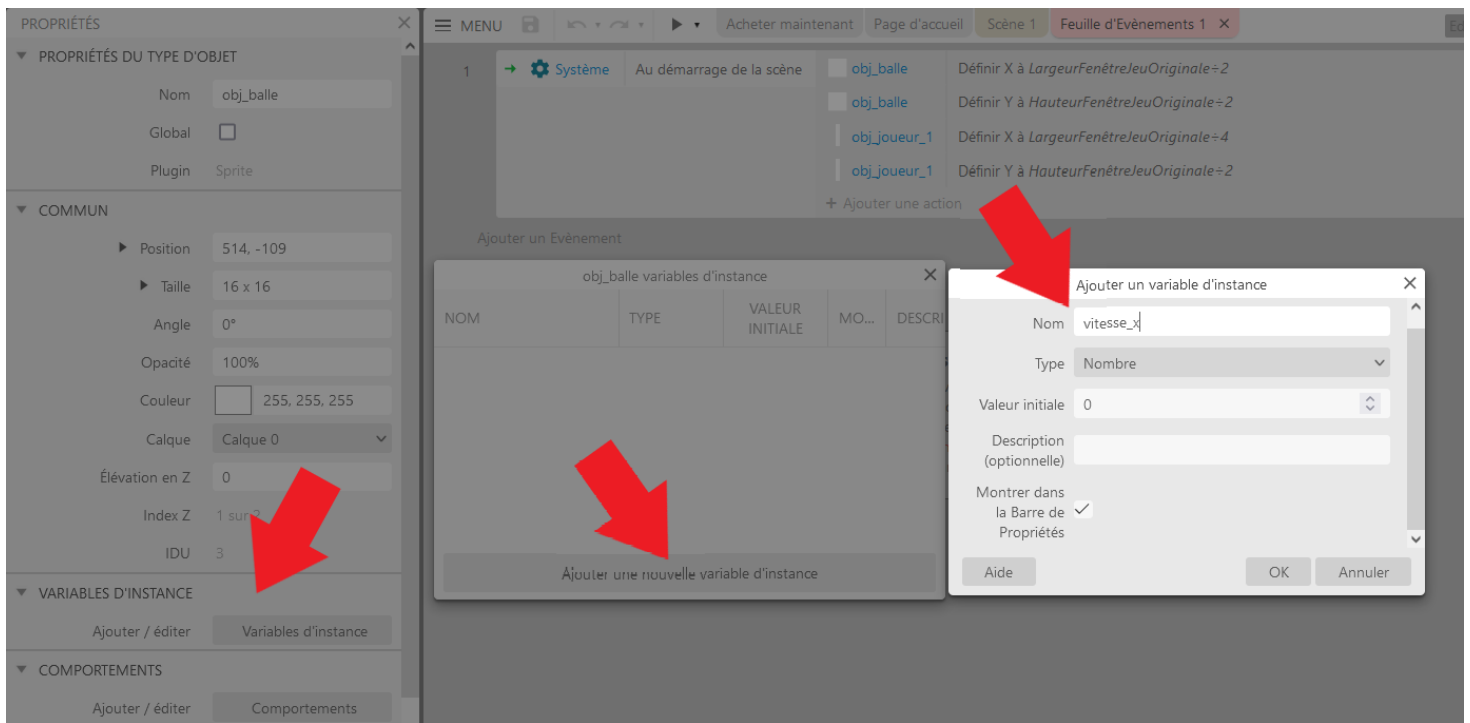
- ➔ Vrai
- ➔ Faux

Dans le cas de notre balle, nous allons utiliser des variables pour la **vitesse** et la **direction** dans laquelle elle se dirige. Également, si l'on se souvient de l'analyse du jeu au chapitre 1, la balle accélère lorsque qu'un joueur la renvoie, il nous faudra donc une autre variable, **accélération**.

Pour ajouter une variable à un objet, il suffit de le sélectionner en cliquant dessus, soit sur la scène, ou bien dans le panneau 'Projet' à droite de l'écran. Une fois l'objet sélectionné, dans le panneau 'Propriétés' à gauche, trouvez la catégorie 'Variables d'Instance', et cliquez sur le bouton 'Ajouter/Éditer'.

Un popup apparaîtra, cliquez sur le bouton 'Ajouter une nouvelle variable d'instance' dans le bas de la fenêtre.

Un deuxième popup s'affiche. Tout comme lorsque nous créons nos objets, il est primordial de donner des noms significatifs à nos variables. Nous allons commencer par créer la variable **vitesse_x**, qui représentera la vitesse horizontale de notre balle. Cette variable sera de type **nombre**, et nous pouvons laisser sa valeur initiale à zéro (spoiler : nous allons la déterminer au démarrage de la scène dans la feuille d'évènement plus tard). Laissez la case '**Montrer dans la Barre de Propriétés**' cochée.



Cliquez sur 'OK'. Notre variable est maintenant créée.

Le second popup devrait disparaître, et notre variable est maintenant visible dans la première fenêtre.



Tant qu'à être ici, faisons une pierre 3(!) coups, et créons tout de suite les autres variables dont nous avons besoin.

- ➔ vitesse_y
- ➔ acceleration

Les deux sont également du type *nombre*.

Une fois vos trois variables créés, votre écran devrait montrer ceci :



*** ATTENTION ***

Un nom de variable

NE DOIT PAS :

- ➔ Contenir de caractères spéciaux, incluant les accents

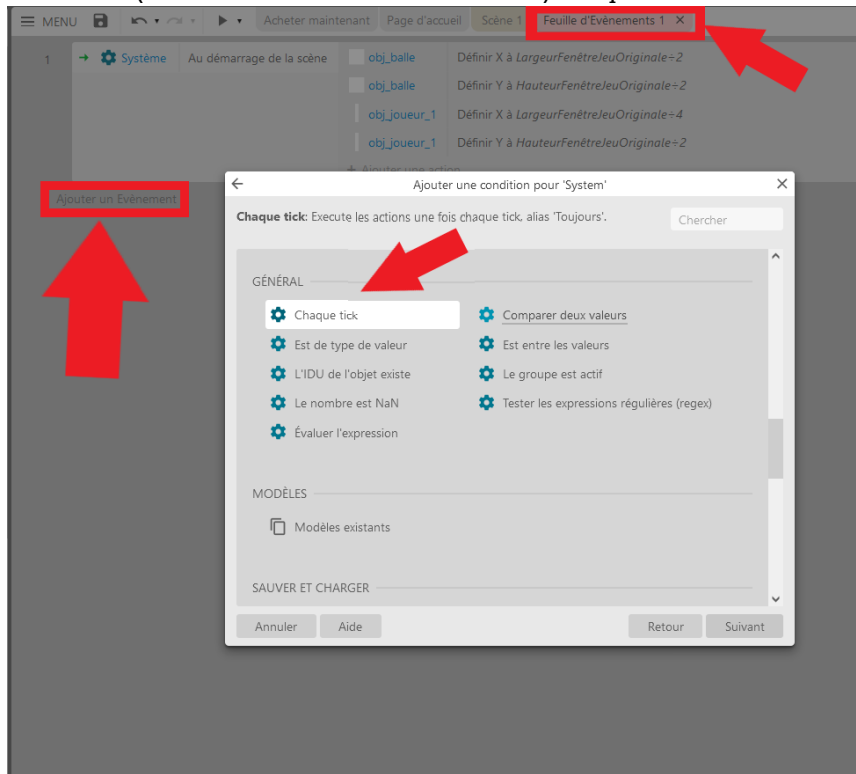


DOIT ABSOLUMENT :

- ➔ Commencer par une lettre ou un *underscore*, aussi connue sous '*la_petite_barre_en_dessous*'

Fermez la fenêtre des variables, et retournons sur notre '*Feuille d'Évènements 1*', en cliquant sur l'onglet rose en haut de l'écran, ou encore dans la barre 'Projet' à droite.

Ajoutons un nouvel *Évènement pour le Système*, mais cette fois, nous allons sélectionner '*Chaque Tick*' dans le menu déroulant (ou utilisez la barre de recherche). Cliquez 'Suivant'.



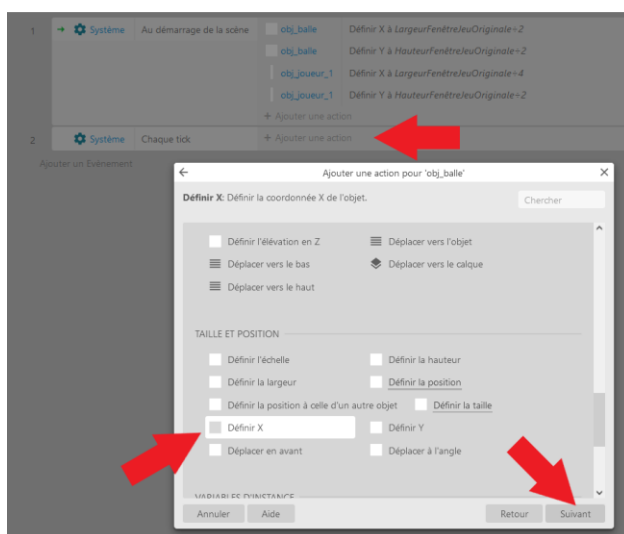
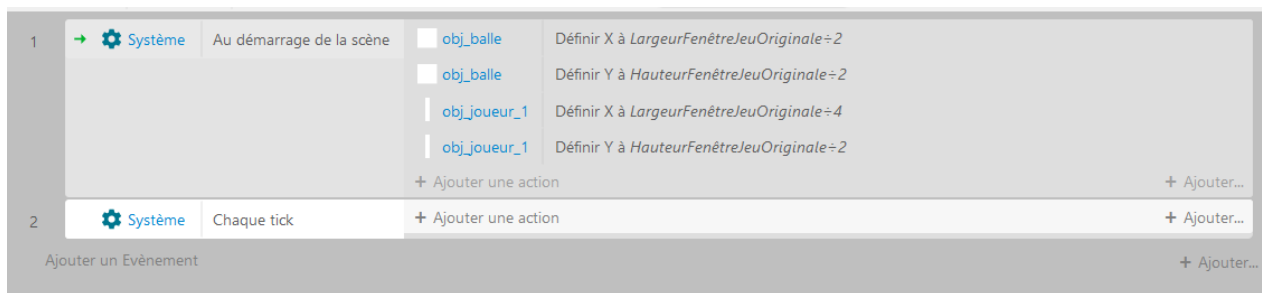
*** Boucle de Jeu ***

L'évènement 'Chaque Tick' représente ce que l'on appelle la *boucle de jeu*. Simplement, c'est le code qui se répète en boucle pour toute la durée de la scène.

Son exécution est très rapide, plusieurs dizaines, voire plus d'une centaine d'exécutions *chaque seconde*! Le standard est aujourd'hui environ 60 exécutions par seconde.

C'est donc l'évènement idéal pour mettre à jour les variables que l'on veut continuellement ajuster, comme la position d'un joueur (ou d'une balle!), par exemple.

Un nouvel évènement devrait apparaître au bas de la liste.

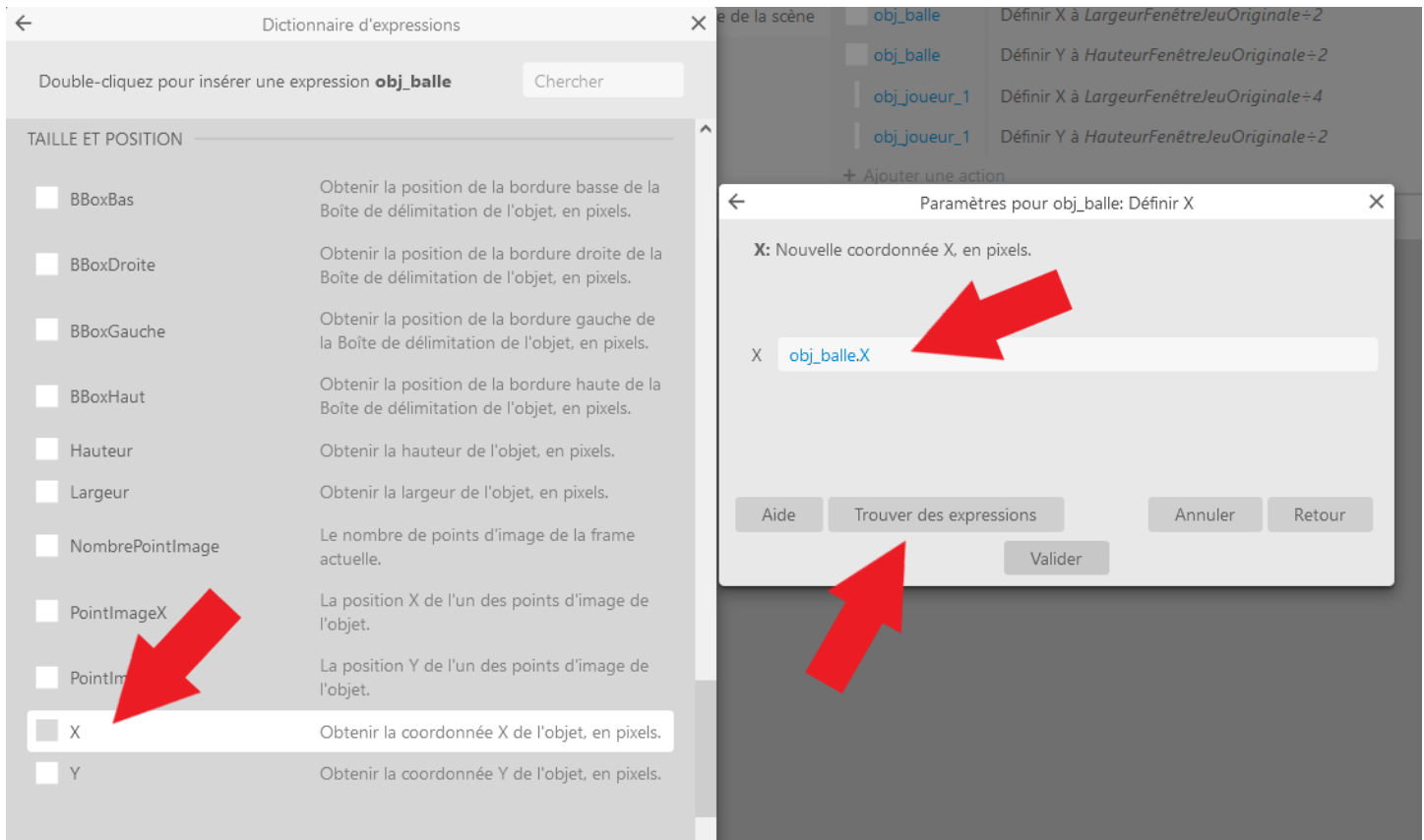


Cliquez sur '*Ajouter une action*' et sélectionnez **obj_balle**.

Dans la liste d'actions, trouvez '*Définir X*', dans la catégorie 'Taille et Position'. Cliquez sur suivant.

Définissons encore une fois la variable avec une *Expression*.

1. Cliquez sur 'Trouver des expressions'
2. Sélectionnez **obj_balle**
3. Sélectionnez '**X**' dans la section '*Taille et Position*'
4. Vous devriez voir '**obj_balle.X**' apparaître dans le popup



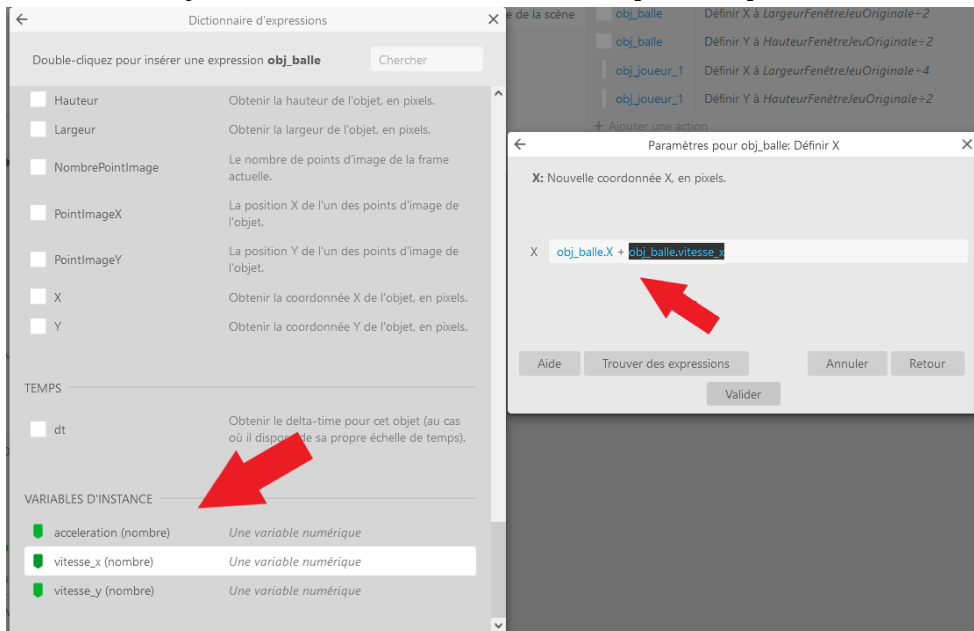
En revanche, ce code ne peut être laissé tel quel, puisqu'il ferait simplement, à chaque 'tick', remplacer sa position X à la même valeur que sa position X actuelle... *duh!*

L'idée ici, c'est **d'ADDITIONNER NOTRE VITESSE** à notre position actuelle à chaque *tick*.

Le champ '*Expression*', comme son nom l'indique, peut contenir des opérations mathématiques qui seront évaluées lorsque le jeu est en cours.

À la fin de la ligne **obj_balle.X**, ajoutons un symbole d'addition **+**

Une fois le **+** ajouté, allez dans le dictionnaire d'expression pour trouver, tout en bas de la liste, la variable



vitesse_x que nous avons créée, dans la section '*Variables d'Instances*'.

Votre expression devrait être identique à celle-ci.

➔ Cliquez sur '**Valider**' pour revenir automatiquement à la feuille d'événements.

➔ Répétez le processus pour définir Y. Ajoutez simplement une nouvelle action pour *Définir Y* à la *valeur Y actuelle + vitesse_y*

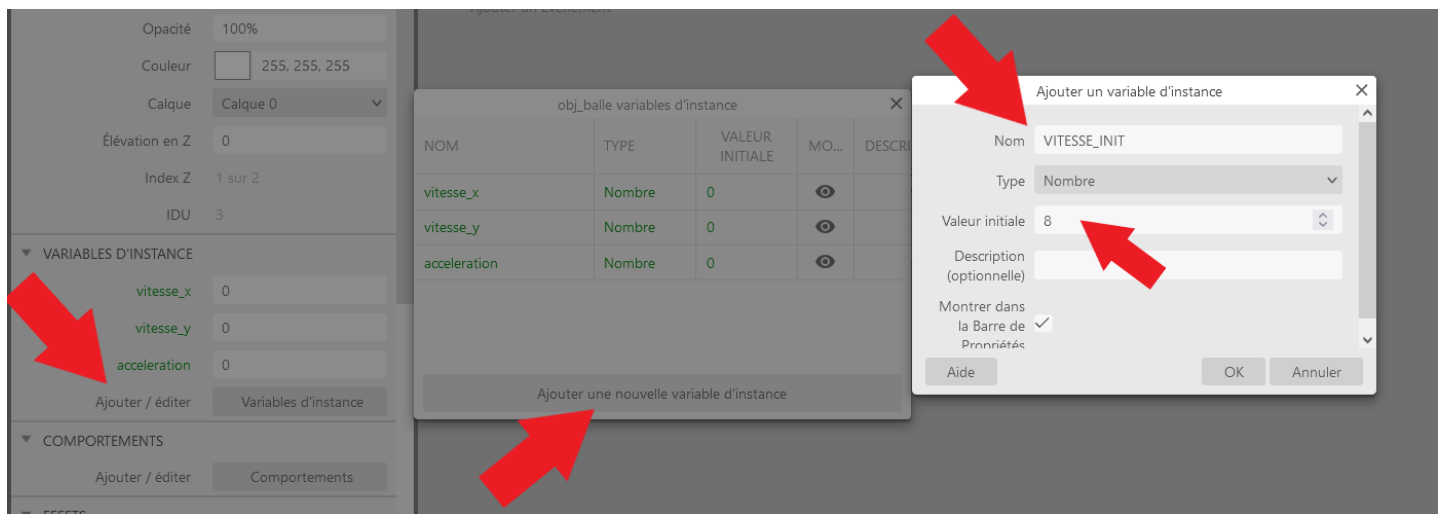
Votre Feuille d'évènements devrait maintenant ressembler à ceci :

1	→ Système	Au démarrage de la scène	 obj_balle	Définir X à $\text{LargeurFen\^etreJeuOriginale} \div 2$
			 obj_balle	Définir Y à $\text{HauteurFen\^etreJeuOriginale} \div 2$
			 obj_joueur_1	Définir X à $\text{LargeurFen\^etreJeuOriginale} \div 4$
			 obj_joueur_1	Définir Y à $\text{HauteurFen\^etreJeuOriginale} \div 2$
			+ Ajouter une action	
			+ Ajouter...	
2	→ Système	Chaque tick	 obj_balle	Définir X à $\text{obj_balle.X} + \text{obj_balle.vitesse_x}$
			 obj_balle	Définir Y à $\text{obj_balle.Y} + \text{obj_balle.vitesse_y}$
			+ Ajouter une action	
			+ Ajouter...	
Ajouter un Evènement			+ Ajouter...	

Si nous pouvons testons avec *F4*, nous pouvons voir que la balle reste au centre de l'écran. C'est tout à fait normal, puisque nous avons défini la valeur de nos variables à 0, et additionner 0 à un nombre ne le déplacera bien entendu jamais.

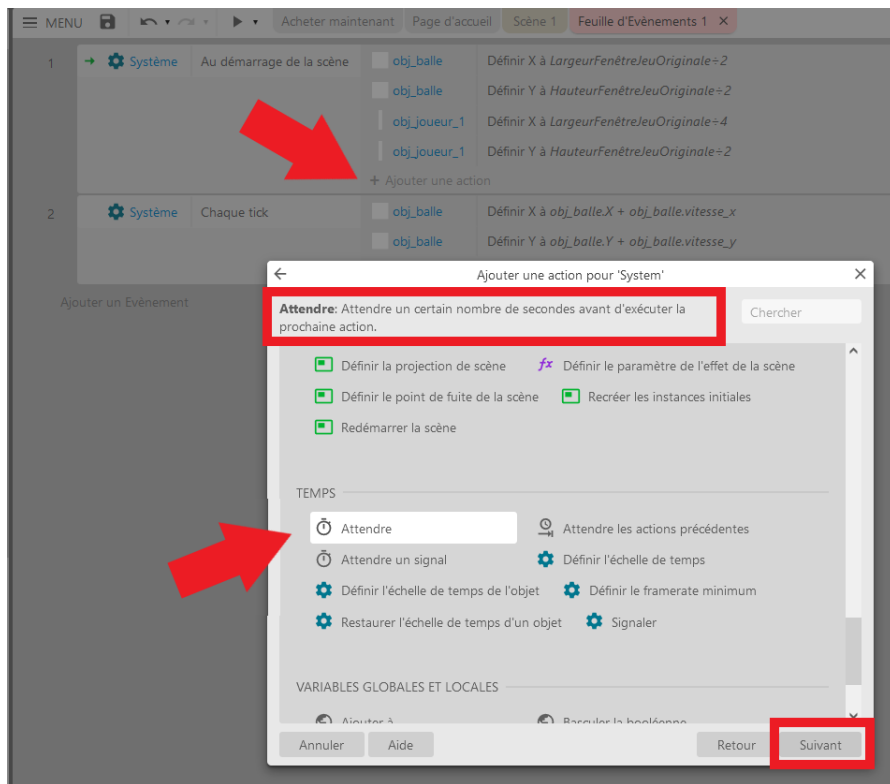
Créons une nouvelle variable pour notre balle, et baptisons la **VITESSE_INIT**. La raison pour laquelle je mets cette variable en majuscule est simplement un aide-mémoire : c'est la syntaxe que j'utiliserai lorsque les variables ne devraient pas changer durant le jeu. Nous pouvons les voir ici comme des 'constantes'. Vous pourrez expérimenter avec différentes valeurs d départ pour la balle très facilement en ajustant simplement la valeur de VITESSE_INIT.

Cette fois, il ne faut pas laisser sa valeur initiale à 0. Une bonne valeur de départ ici selon moi est 8, c'est ce que je vous conseille d'utiliser pour le moment, afin d'obtenir des résultats identiques aux miens. Vous pourrez modifier sa valeur sans problème à tout moment dans l'éditeur.



Bon! Récapitulons!

Ce qu'il faut maintenant, c'est que notre balle commence à se déplacer à la vitesse initiale que nous avons dans notre nouvelle variable. Modifions notre premier évènement 'Système' → 'Au démarrage de la Scène', et ajoutons une nouvelle *action*. Nous allons choisir ici, non pas la balle, mais bien **Système**.



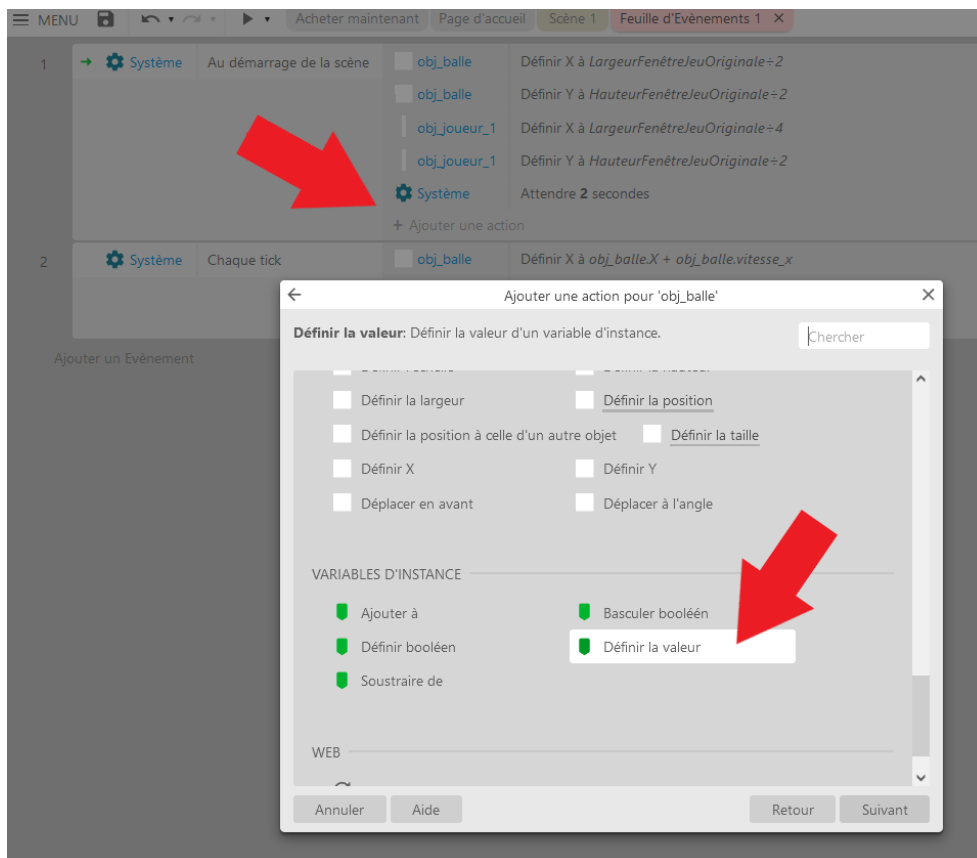
Nous aurions pu utiliser immédiatement **obj_balle**, mais l'objet *Système* nous offre une action très intéressante : **Attendre**.

Cette action se trouve sous la catégorie 'Temps', dans le bas du menu déroulant.

Cette valeur représentera le temps avant que la balle soit mise en jeu au début de la partie.

Vous devriez voir la nouvelle action **attendre** apparaître dans votre évènement Démarrage de la Scène. Parfait!

Ajoutons tout de suite une nouvelle action, mais cette fois, choisissons l'**obj_balle**.



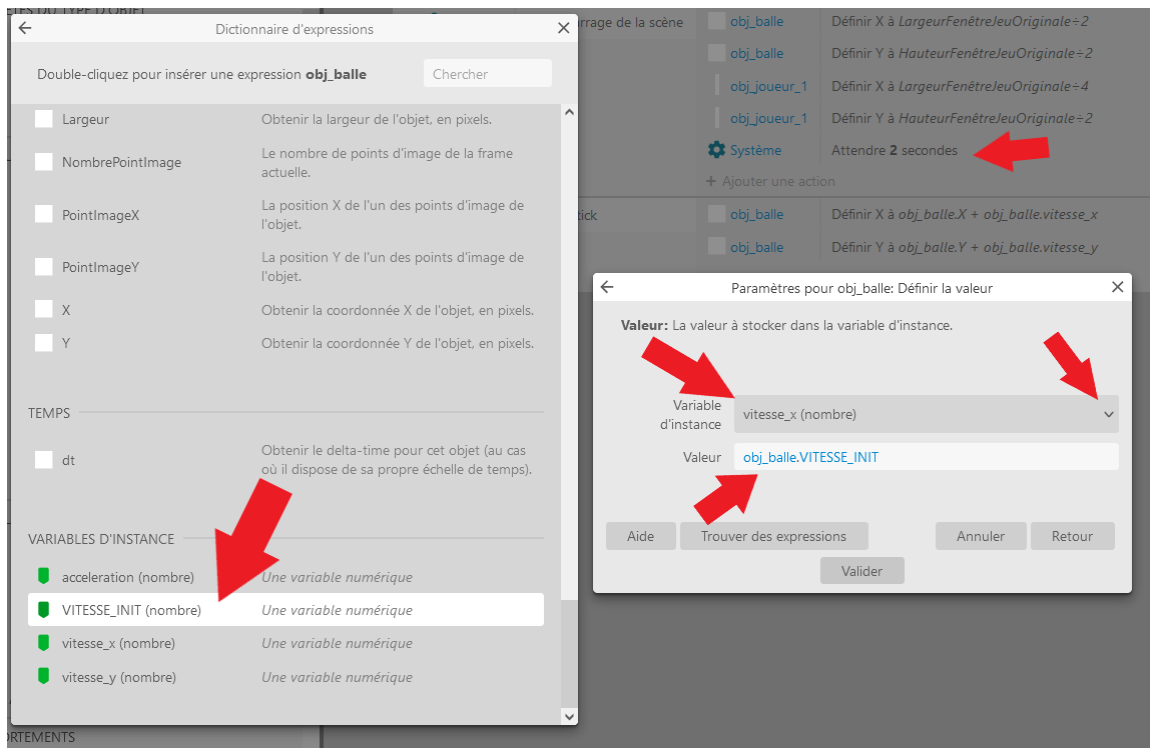
→ Sélectionnez **Définir la Valeur** dans la catégorie *Variables d'Instances* (ce sont les variables que nous avons créées).

→ Une fenêtre s'affiche. Dans le menu déroulant 'Variable d'instance', sélectionnez **vitesse_x**.

→ Dans le champ 'Valeur', effacez le chiffre '0', puis cliquez sur 'Trouver une expression'.

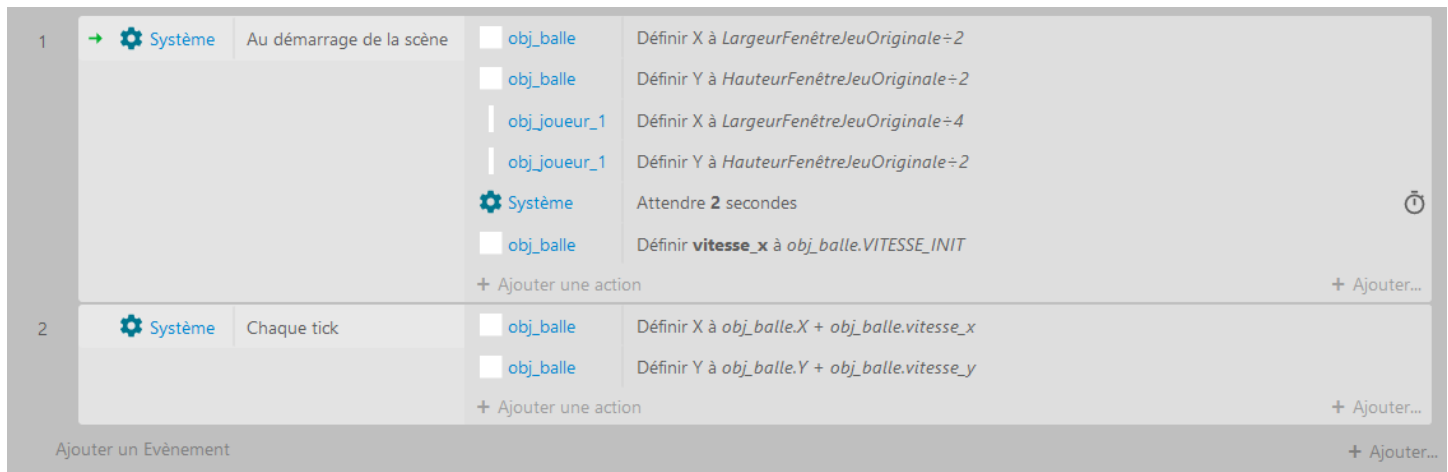
→ Sélectionnez **obj_balle**, et allez chercher au bas du menu déroulant notre valeur **VITESSE_INIT**.

→ Cliquez sur 'Valider'.



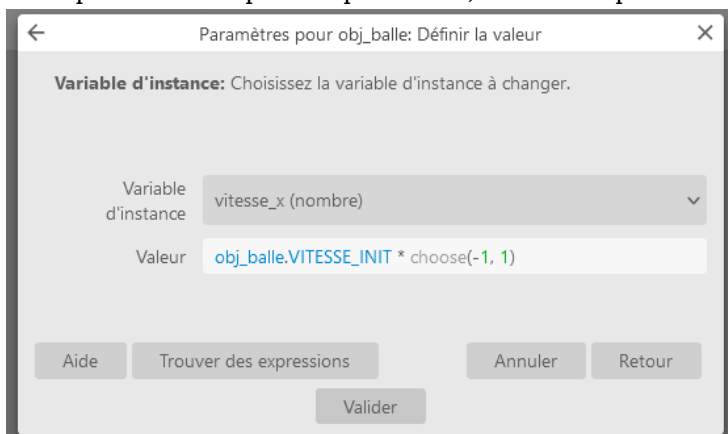
Validez votre choix.

Votre feuille d'évènements est semblable à ceci :



En testant le jeu, nous pouvons observer qu'après 2 secondes, la balle commence à se déplacer! C'est parfait!

Cette prochaine étape est optionnelle, mais nous pouvons donner une direction aléatoire à notre balle au départ du jeu. Il suffit de modifier légèrement notre expression qui définit la vitesse_x de notre balle en ajoutant *** choose(-1, 1)** à la suite de obj_balle.VITESSE_INIT. Il y aura donc une chance sur deux que la balle prenne la direction du joueur lors de la mise en jeu.



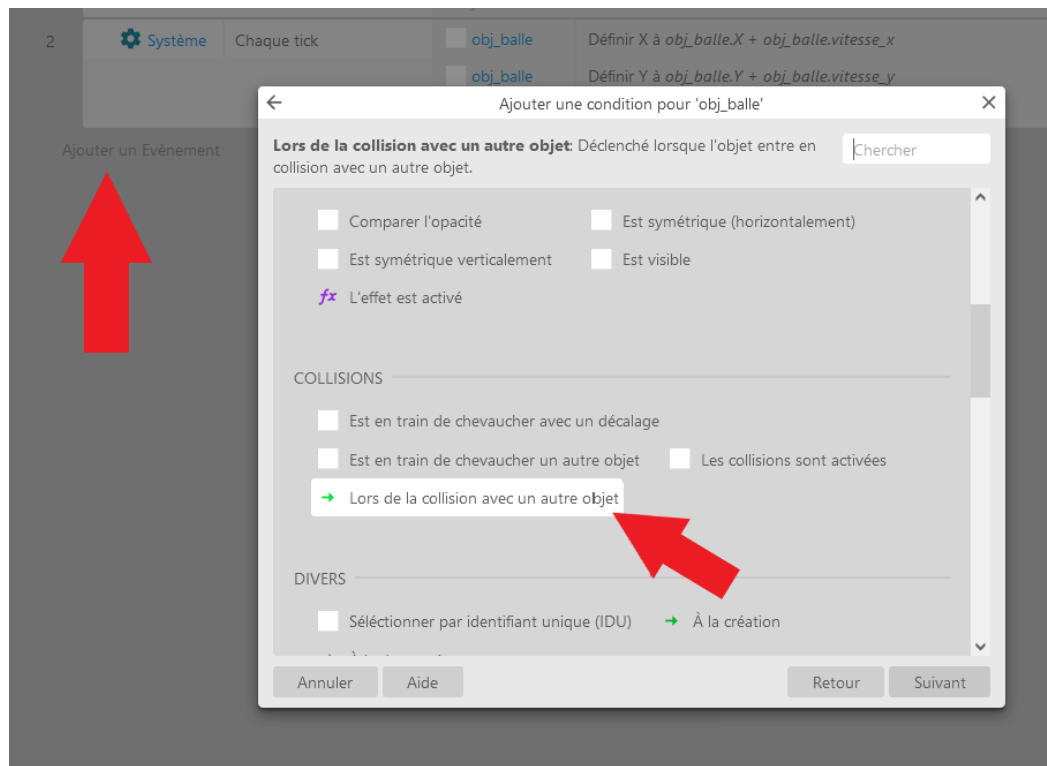
***** ALERTE SAUVEGARDE *****
C'est un excellent moment pour sauvegarder
notre travail et prendre une pause si vous en
sentez le besoin!

ÉTAPE 5 LE REBOND

Si vous avez implémenté la fonction optionnelle de donner une direction aléatoire à la balle lors de la mise en jeu, vous avez probablement remarqué un problème majeur : la balle passe au travers de la raquette du joueur!

Nous devons donc ajouter un troisième évènement à notre feuille d'évènements qui définira ce que fera notre balle quand elle frappe un joueur.

Cliquez sur '*Ajouter un Évènement*', choisissez **obj_balle**



➔ Pour condition, nous allons choisir '**Lors de la collision avec un autre objet**'

➔ Cliquez sur '*Suivant*'

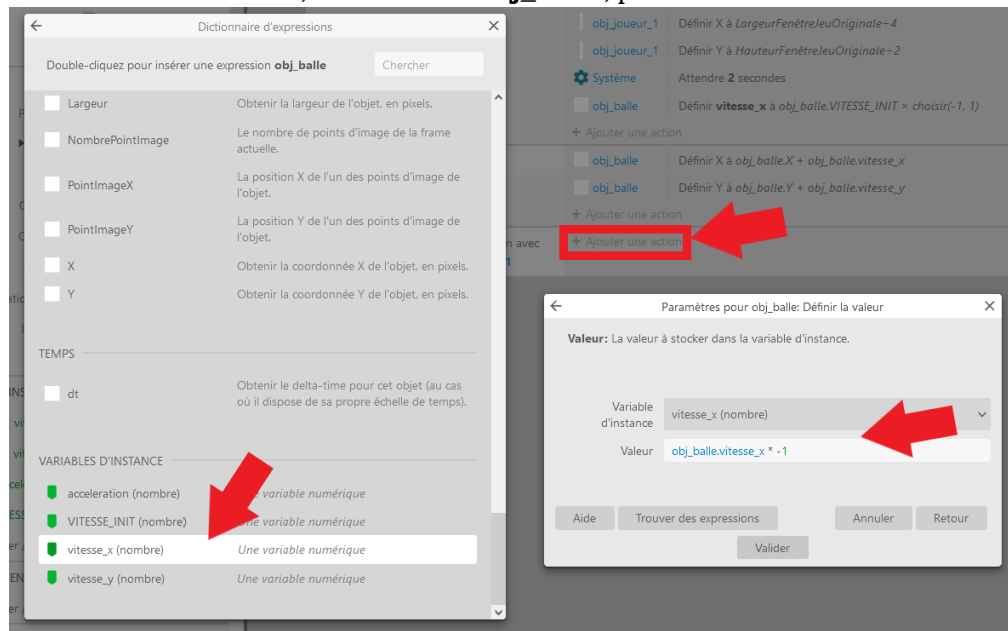
➔ Dans le menu déroulant, choisissez **obj_joueur_1**, puisque nous voulons définir ce que la balle fait lorsqu'elle frappe le joueur.

➔ Cliquez sur '*Valider*'

➔ Le nouvel évènement devrait apparaître dans la feuille d'évènement.

➔ Cliquez sur '*Ajouter une action*' sur notre nouvel évènement.

Dans la nouvelle action, sélectionnez **obj_balle**, puis sélectionnez '**Définir la valeur**'.



➔ Choisissez **vitesse_x** dans le menu déroulant

➔ Dans le champ '*Valeur*', trouvez une expression, puis sélectionnez **vitesse_x**

➔ Modifiez le champ '*Valeur*' en multipliant la variable **obj_balle.vitesse_x** par -1

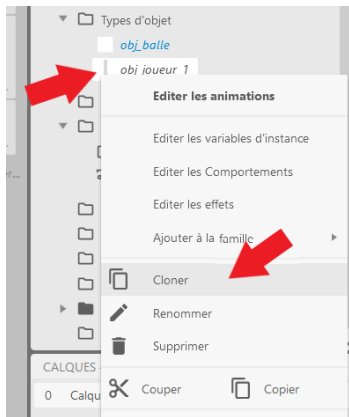
➔ L'expression finale du champ '*Valeur*' devrait être : **obj_balle.vitesse_x * -1**

➔ Validez

L'évènement devrait être identique à ceci :



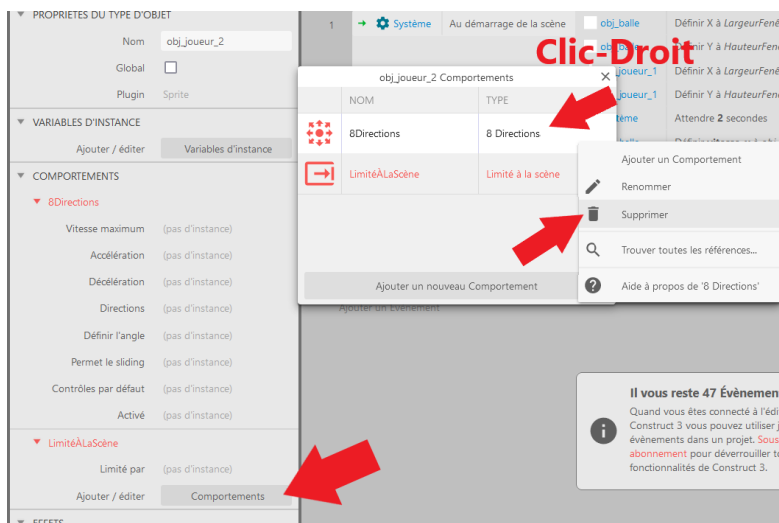
Maintenant que la balle rebondit sur la raquette du joueur, il serait intéressant d'avoir un adversaire à qui la renvoyer, n'est-ce pas? Heureusement, puisque les comportements de l'adversaire contrôlé par l'ordinateur seront très semblables à ceux du joueur humain, le plus simple est simplement de dupliquer notre **obj_joueur_1**.



Pour ce faire, il suffit de faire

- ➔ Clic-droit sur notre **obj_joueur_1** dans le panneau de *Projet* (celui à la droite de l'écran).
- ➔ Dans le menu, cliquez sur '**Cloner**'.
- ➔ Nous voyons un **obj_joueur_2** apparaître dans notre hiérarchie de projet, c'est exactement ce que nous voulons!

Avec le second **obj_joueur_2** sélectionné, allez dans son panneau de propriétés.

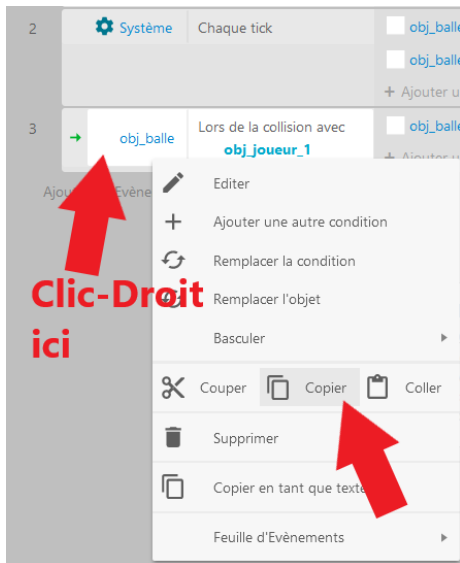


- ➔ Trouvez la section '*Comportements*'
- ➔ Cliquez '*Ajouter / Éditer*'
- ➔ Faites un clic-droit sur le comportement '*8 Directions*'
- ➔ Cliquez sur '*Supprimer*'

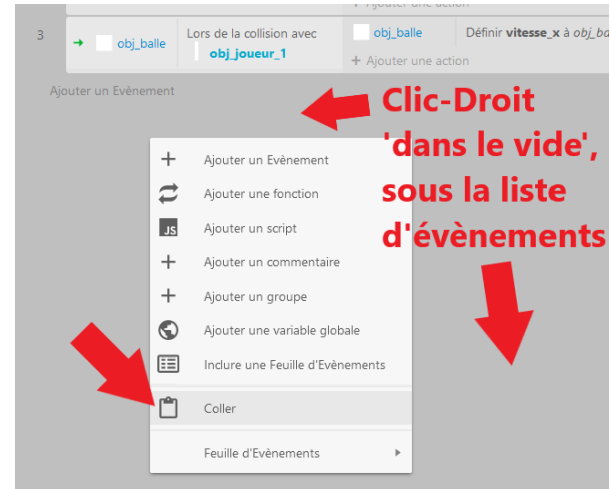
Note : La raison pour laquelle nous supprimons ce comportement est que, si nous le laissions en place, les deux joueurs se déplaceraient de manière identique, puisqu'ils seraient tous deux contrôlés par les mêmes touches.

Maintenant, nous devons dire à notre balle quel comportement adopter lorsqu'elle entre en collision avec notre nouvel objet **obj_joueur_2**. En fait, nous voulons exactement la même chose que pour la collision avec notre joueur 1.

Faisons un simple copier-coller de notre évènement 'Collision de la balle avec le joueur 1'.



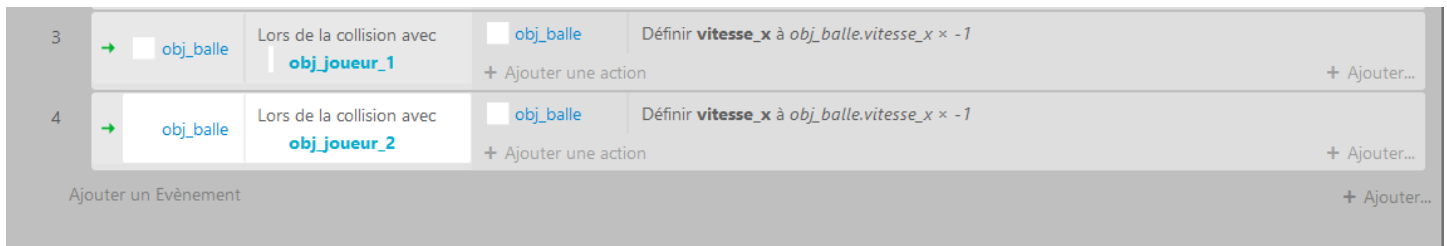
- ➔ Clic-droit sur l'Évènement (pas la condition !!)
- ➔ Copier
- ➔ Clic-droit dans une zone 'vide' de la feuille d'évènements
- ➔ Coller



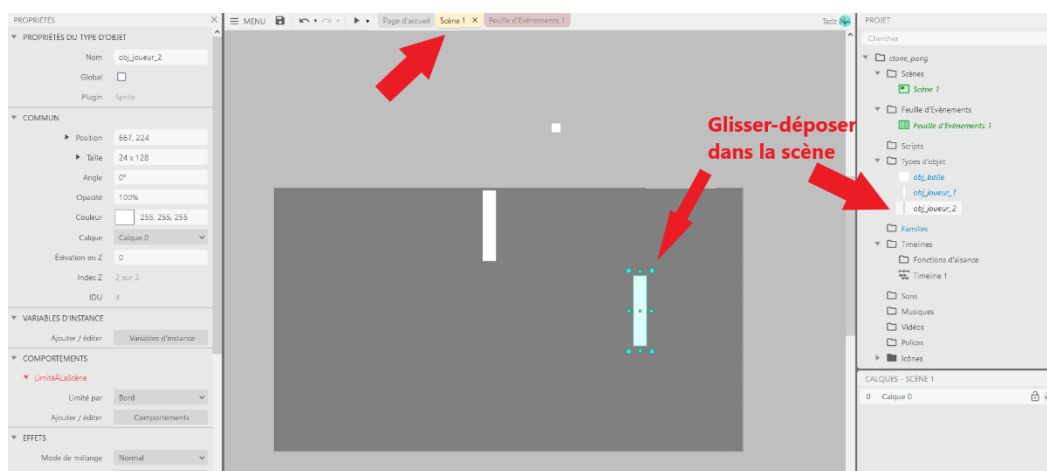
Vous devriez avoir 2 évènements identiques.

Sur l'évènement que vous venez de dupliquer, double-cliquez sur la condition 'Lors de la collision avec obj_joueur_1', puis remplacez-la simplement par obj_joueur_2.

Voici le résultat :



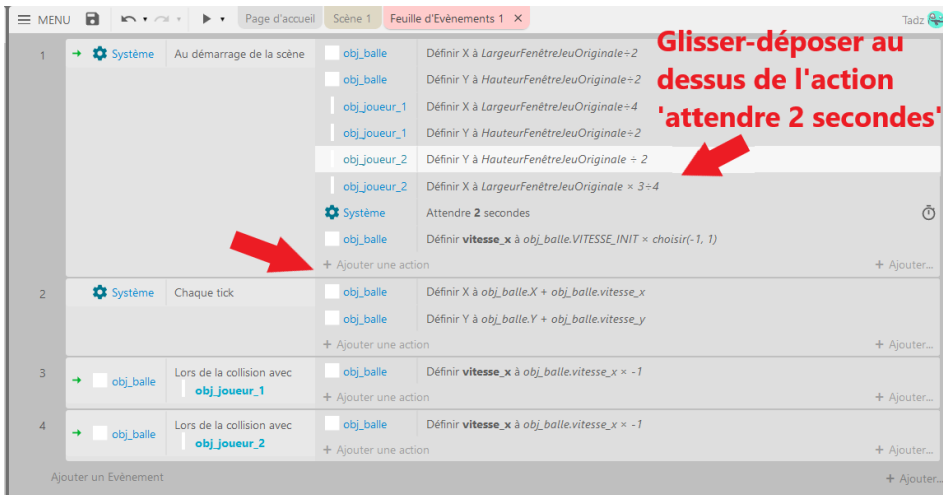
Vous pouvez maintenant glisser-déposer une instance de **obj_joueur_2** dans la scène à partir de l'onglet *Projet* (celui à droite). Rappel : pour sortir de la *feuille d'évènements* et revenir à la *scène*, il suffit de cliquer sur l'onglet *Scène 1*, en haut de l'écran.



➔ Placez votre second joueur vers la droite, environ à la moitié de l'écran (en hauteur).

➔ Testez et observez ce qui se passe : après 2 secondes, la balle devrait commencer à se déplacer, et les deux joueurs devraient se renvoyer la balle! Yayy!

Tout comme pour le premier joueur, faisons en sorte que notre adversaire se positionne automatiquement sur le terrain de jeu au départ de la scène. Nous utiliserons le même procédé que pour le joueur 1, à l'exception qu'au lieu de le positionner au $\frac{1}{4}$ de l'écran, nous le positionnerons au $\frac{3}{4}$ de l'écran, afin d'avoir une symétrie parfaite.



➔ La position initiale en Y reste toujours égale à la moitié de la hauteur de l'écran.

➔ Nous allons donc ajouter 2 nouvelles actions, **Définir X** et **Définir Y**

➔ Les nouvelles actions apparaîtront en-dessous de l'action 'Attendre 2 secondes'. Ce n'est pas ce que l'on veut, sinon le joueur 2 ne sera en position lorsque la balle se met en mouvement!

➔ On peut simplement glisser-déposer les actions au-dessus de l'action 'Attendre'

On peut tester le jeu, et voir que les deux joueurs se mettent automatiquement en position au début de la partie et que la balle rebondit lorsqu'elle touche l'un d'eux.

On peut cependant observer trois choses :

1. La balle n'accélère pas
2. La balle se déplace toujours à l'horizontale
3. Rien ne remet la balle en jeu si elle 'dépasse' un joueur.

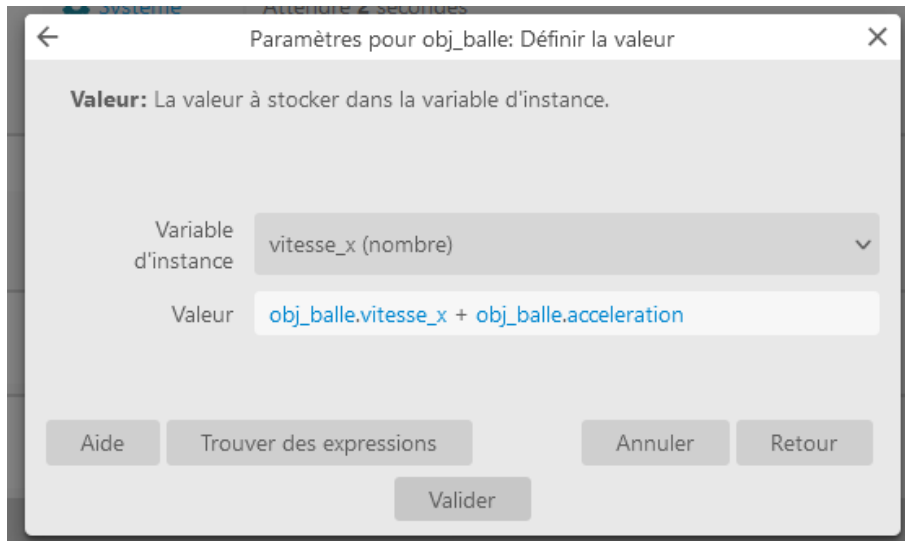
Prenons un moment pour préparer notre plan. Il nous faudra attaquer chacun de ces problèmes séparément. Voici le plan d'action pour chacun des 3 points :

1. Nous allons ajouter la valeur de **accélération** à la balle à chaque fois qu'elle entre en collision avec un joueur.
2. Nous allons ajuster la vitesse en Y de la balle par rapport à l'endroit où elle a frappé la raquette.
3. Nous allons créer un nouvel objet invisible qui représentera une 'zone de but' (en fait nous allons en créer deux, soit une pour chacun des joueurs).
4. Il nous faudra également créer un objet 'mur', afin que la balle ne sorte pas de l'écran vers le haut ou vers le bas.
5. Mais avant...

***** ALERTE SAUVEGARDE *****

C'est un excellent moment pour sauvegarder notre travail et prendre une pause si vous en sentez le besoin!

Alors, ce qu'il faut faire est d'ajouter une nouvelle action lors de la collision de la balle et des joueurs.



➔ Cliquez sur 'Ajouter une action', puis sélectionnez **obj_balle**

➔ Choisissez **Définir la valeur**

➔ Sélectionnez dans le menu déroulant **vitesse_x**, puisque c'est la variable que l'on veut modifier.

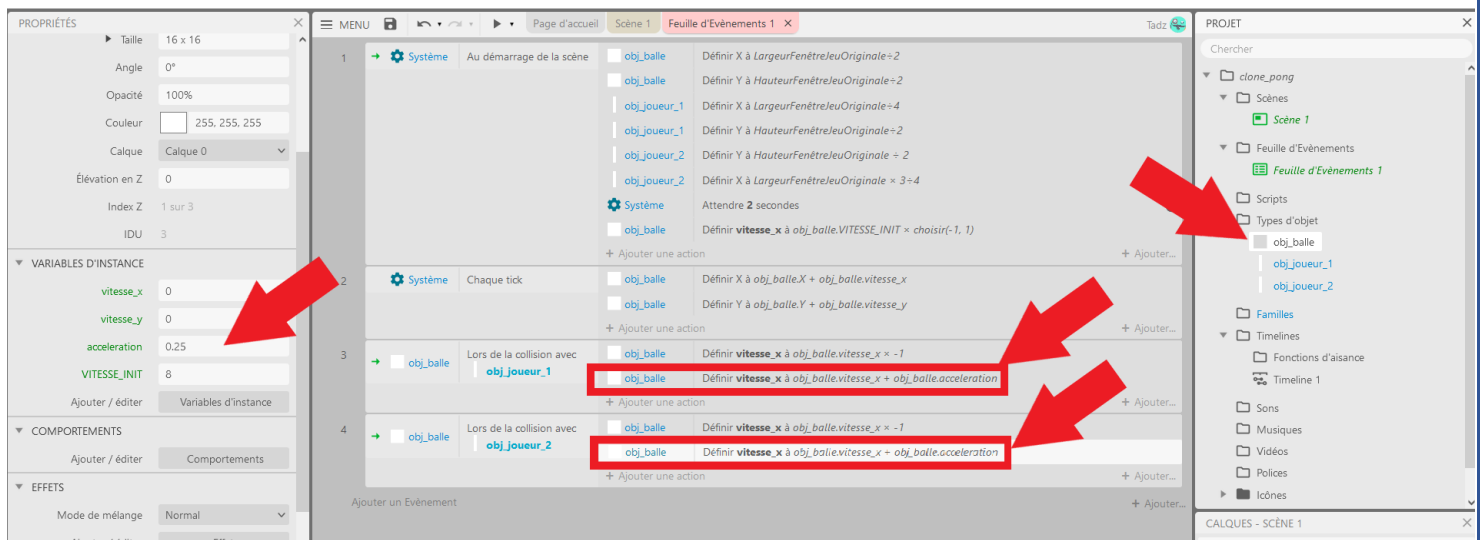
➔ Dans le champ valeur, additionnez la vitesse de la balle à son accélération.

➔ Vos paramètres devraient être identiques à la capture d'écran ci-contre.

Répétez les mêmes étapes pour l'évènement de collision avec le second joueur.

Dans le panneau 'Projet', sélectionnez **obj_balle** et ajustez sa variable '**accélération**' à 0.25. Vous pouvez mettre la valeur que vous voulez, mais attention à ne pas mettre une trop grande accélération, nous y reviendrons.

À ce point, projet Construct ressemble maintenant à ceci :



Un simple test confirme que tout est fonctionnel.

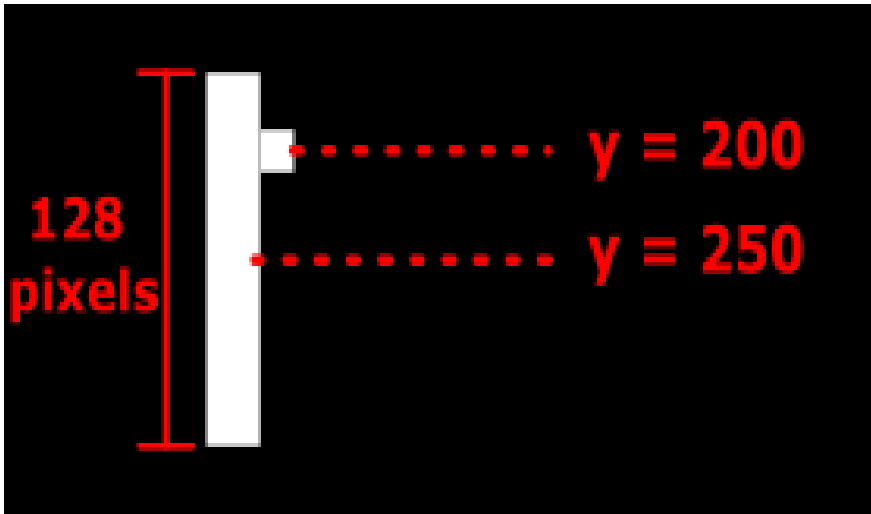
Mais ce n'est pas encore au point. Notre balle se déplace toujours à l'horizontal peu importe l'endroit où elle frappe la raquette du joueur. L'angle de notre balle sera déterminé par notre variable **vitesse_y**, c'est donc celle-ci qu'il nous faudra modifier lors d'une collision entre la balle et le joueur.

Mais comment faire? Eh bien, il est possible, grâce aux mathématiques, de calculer ceci en comparant la position de la balle et de la raquette sur l'axe Y lors de la collision. Ensuite, il nous faudra comparer cet écart avec la hauteur de notre raquette. Finalement, il faudra multiplier ce résultat par la vitesse maximale que pourra prendre notre balle sur l'axe Y.

Ouf! Je crois que tout cela mérite un dessin et un exemple pour illustrer le concept.

Et ne soyez pas inquiet, la formule mathématique finale vous est fournie à la page suivante. Il est toutefois important de comprendre son fonctionnement et comment nous l'avons conçue.

Prenons en exemple la situation de collision suivante. Comment allons-nous déterminer la vitesse en Y que prendra notre balle? Appliquons notre formule!



→ La position de la balle sur l'axe Y est 200 pixels.

→ La position de la raquette est 250 pixels.

→ Si nous soustrayons la position de la raquette à celle de la balle nous obtenons :

$$200 - 250 = -50$$

→ Divisons le résultat -50 avec la hauteur de la raquette de notre joueur, qui est dans notre cas 128 pixels. Donc

→ $-50 / 128 = -0.39$ (environ)

Imaginons que nous avons décidé que la vitesse maximale en Y serait de 10. Nous pouvons simplement faire la multiplication :

$$-0.39 * 10 = -3.9$$

Donc, à la suite de cette collision, la vitesse en Y de la balle sera de -3,9.

Fiou!

Il nous faut donc maintenant utiliser notre formule mathématique pour assigner la valeur **vitesse_y** de notre balle lors de la collision avec les 2 joueurs.

Il faut donc ajouter une action à la balle qui va **définir la variable vitesse_y** à :

Lors de la collision avec le joueur 1 :

$$((\text{Self.Y} - \text{obj_joueur_1.Y}) / \text{obj_joueur_1.Height}) * 10$$

Lors de la collision avec le joueur 2 :

$$((\text{Self.Y} - \text{obj_joueur_2.Y}) / \text{obj_joueur_2.Height}) * 10$$

Votre feuille d'évènements devrait contenir ceci :

3	→ obj_balle	Lors de la collision avec obj_joueur_1	obj_balle	Définir vitesse_x à $\text{obj_balle.vitesse_x} \times -1$
			obj_balle	Définir vitesse_x à $\text{obj_balle.vitesse_x} + \text{obj_balle.acceleration}$
			obj_balle	Définir vitesse_y à $((\text{Self.Y} - \text{obj_joueur_1.Y}) / \text{obj_joueur_1.Height}) \times 10$
+ Ajouter une action				
4	→ obj_balle	Lors de la collision avec obj_joueur_2	obj_balle	Définir vitesse_x à $\text{obj_balle.vitesse_x} \times -1$
			obj_balle	Définir vitesse_x à $\text{obj_balle.vitesse_x} + \text{obj_balle.acceleration}$
			obj_balle	Définir vitesse_y à $((\text{Self.Y} - \text{obj_joueur_2.Y}) / \text{obj_joueur_2.Height}) \times 10$
+ Ajouter une action				

Ajouter un Evènement

***** IMPORTANT *****

Une vitesse négative signifie simplement 'vers le haut de l'écran'.

Le point (0,0) de notre scène se trouve dans le coin supérieur gauche.

En exécutant le jeu, nous pouvons déplacer légèrement notre joueur vers le haut ou vers le bas et voir que le rebond s'exécute correctement!

Oulala!

Pour vous rassurer, c'était l'étape la plus difficile, si vous avez suivi jusqu'ici le reste sera un jeu d'enfant. Dans le cas contraire, sauvegardez votre travail, prenez une pause et revenez à tête reposée un peu plus tard. Il s'agit souvent d'une simple erreur de frappe ou d'inattention.

Si l'on voulait améliorer notre code, nous pourrions modifier la portion '**multiplié par 10**' de notre formule par une variable. Rappelez-vous que cette valeur 10 représente la vitesse maximale de la balle en Y, il est fortement probable que vous vouliez ajuster cette valeur plus tard. Ce sera beaucoup plus facile si vous utilisez une variable. Sinon, il vous faudra changer chacune des formules individuellement pour faire des modifications.

Cela dit, notre mission n'est toujours pas terminée. Nous n'avons toujours pas de 'murs' ni de zone de but.

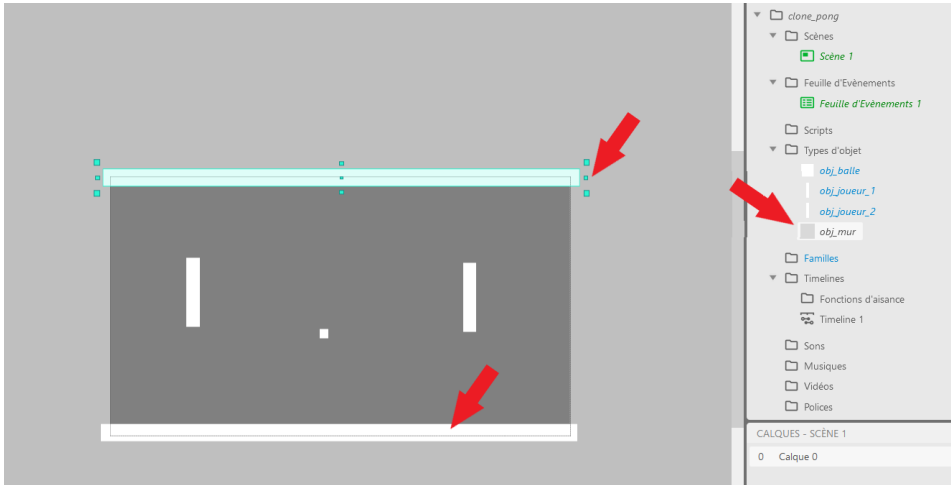
Ce sont tous deux des objets très élémentaires qui seront très facile à implémenter.

***** ALERTE SAUVEGARDE *****

C'est un excellent moment pour sauvegarder notre travail et prendre une pause si vous en sentez le besoin!

ÉTAPE 6 LES MURS

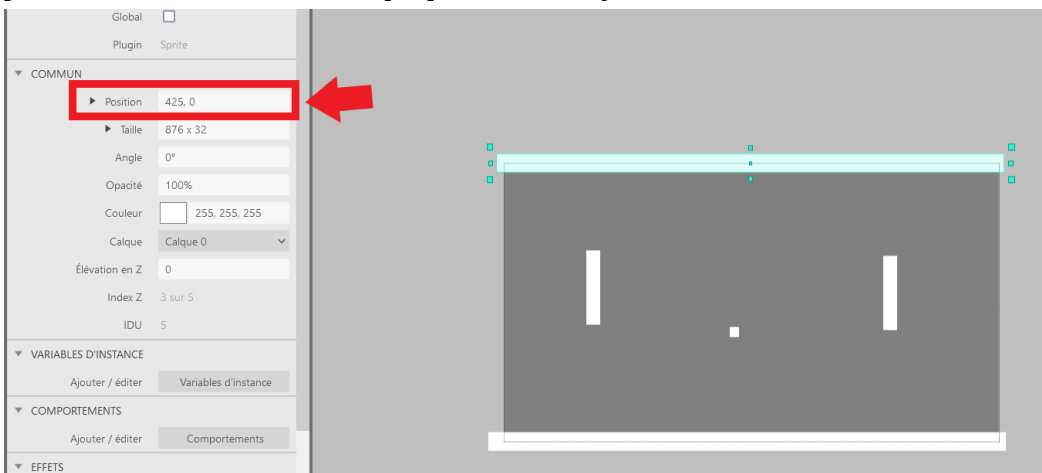
Retournons dans l'onglet *Scène*, et créons un nouvel objet de type *Sprite* que nous nommerons **obj_mur**. Tout comme le sprite du joueur et de la balle, il serait judicieux d'ajuster sa taille dans l'éditeur de sprite, car la valeur par défaut de 250x250 est beaucoup trop grande. Allons-y pour une valeur de **32x32**. Remplissez votre sprite de couleur blanche.



Une fois placé dans votre scène, vous pouvez étirer votre mur en le sélectionnant et en utilisant les points qui apparaissent autour de votre objet.

Ce n'est pas grave si vos murs sont plus larges que votre scène, il n'y aura aucun effet sur le jeu.

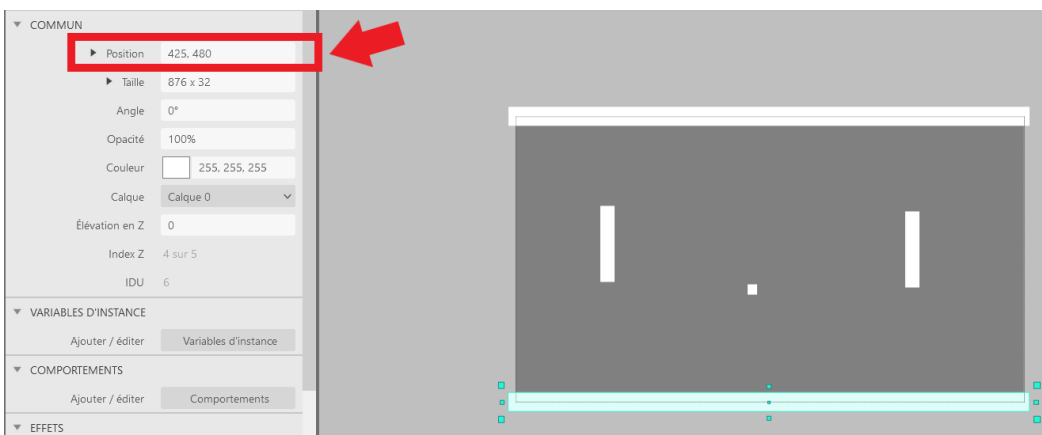
Pour que nos murs soient alignés parfaitement à la verticale, nous pouvons simplement manuellement entrer leur position en Y dans la barre de propriétés de l'objet.



➔ Le mur du haut sera à la position Y= 0

➔ Le mur du bas sera à la position Y= 480

➔ 480 représente la hauteur de notre scène.



➔ La position pour le mur du bas

Une fois vos murs placés, retournons dans la *Feuille d'Évènements* afin de coder la collision balle-mur.

La logique que nous implémenterons ici est extrêmement simple : nous allons simplement inverser la **vitesse_y** lors de chaque collision avec un mur. Une simple **multiplication par -1** sera suffisante.

Dans la *Feuille d'Évènement*, cliquez sur *Ajouter un Évènement*

- ➔ Choisissez **obj_balle**
- ➔ Sélectionnez 'Lors de la collision avec un autre objet'
- ➔ Sélectionnez **obj_mur**
- ➔ Validez

- ➔ Dans notre nouvel évènement, ajoutons une action pour notre **obj_balle**
- ➔ Sélectionnez *Définir la valeur*
- ➔ Dans le menu déroulant, sélectionnez **vitesse_y**
- ➔ Dans le champ valeur, *Trouvez une expression* et sélectionnez **vitesse_y**, ou bien tapez **obj_balle.vitesse_y**
- ➔ Multipliez par -1, pour obtenir l'expression **obj_balle.vitesse_y * -1** dans le champ 'Valeur'



Il est possible de tester rapidement pour voir que les murs fonctionnent comme il faut, mais nous n'avons toujours pas de mécanisme de 'but', ni de remise en jeu de la balle.

Sauvegardons notre projet, et retournons à l'onglet Scène pour créer l'objet qui servira de zone de but.

ÉTAPE 7

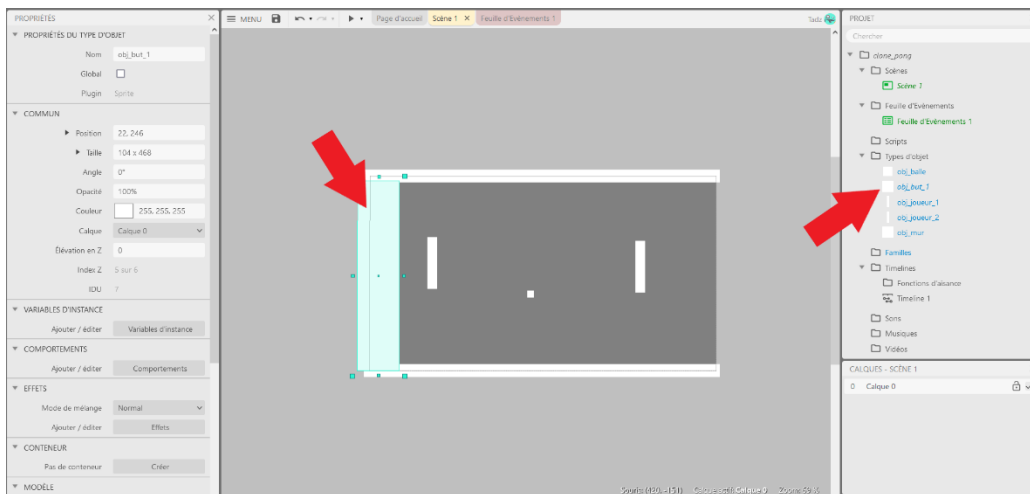
LA ZONE DES BUTS

Encore une fois, nous pourrions adopter plusieurs stratégies différentes pour coder l'objet de la zone des buts. En fait, nous pourrions le faire *sans créer d'objet*, simplement en vérifiant la position de la balle lorsqu'elle sort de l'écran.

La raison pour laquelle j'ai opté pour créer des objets est pour démontrer l'utilité des objets invisibles dans les jeux vidéo en général. Ce concept est très souvent utilisé et pour plusieurs raisons : des obstacles et plateformes invisibles, des récompenses cachées, mais aussi des objets invisibles qui pourraient, par exemple, être responsable de faire apparaître une nouvelle vague d'ennemis chaque minute.

Donc, avec cette approche, nous aurons besoin de deux nouveaux objets différents, mais comme la logique sera quasiment identique pour les deux objets, il est aussi possible de seulement coder le premier objet, et de simplement le dupliquer et faire les modifications.

Donc, à partir de l'onglet *Scène* :



➔ Clic-droit, 'Insérer un nouvel Objet'

➔ Choisissez *Sprite*, nommez-le **obj_but_1**

➔ Insérez, puis placez-le à un endroit quelconque sur votre scène, ce qui ouvrira l'éditeur de *Sprite*

➔ Peinturons temporairement le sprite pour avoir un meilleur aperçu de son positionnement

Placez ensuite l'objet derrière le joueur 1, et étirez-le pour qu'il couvre toute la hauteur de la scène. Ce n'est pas grave si la zone de but chevauche les murs.

Retournons maintenant à notre *Feuille d'événements* pour coder la logique de collision.

Comme nous n'avons pas encore introduit la notion du *score* des joueurs, nous allons pour l'instant se concentrer sur la remise en jeu de la balle. Notre logique devrait ressembler à celle-ci :

1. Réinitialiser la vitesse de la balle à 0.
2. Repositionner la balle au centre de la scène.
3. Attendre un moment.
4. Remettre la balle en jeu dans une direction aléatoire.

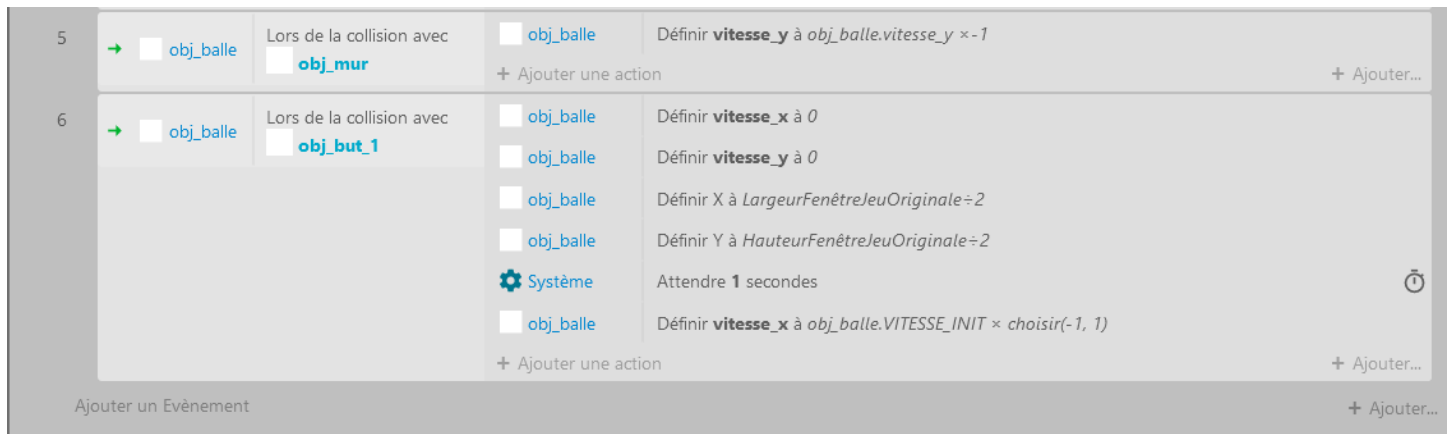
La bonne nouvelle, c'est que nous avons déjà codé chacune de ces actions précédemment, donc le procédé devrait vous être familier.

N'oubliez pas que la position et la vitesse ont chacune **deux variables** qui leur sont associées, l'une en X, et l'autre en Y. Il faut penser à réinitialiser les deux.

Donc, lors de la *collision* **obj_balle** avec **obj_but_1** :

- ➔ Réinitialiser **vitesse_x** et **vitesse_y** à 0
- ➔ Réinitialiser **X** et **Y** au centre de l'écran
- ➔ Demander au **Système** d'*attendre* un moment (vous pouvez utiliser une variable ou entrer une valeur en secondes)
- ➔ Définir vitesse_x à sa valeur initiale VITESSE_INIT, dans une direction aléatoire
 - Truc : Nous avons la même action *Au Démarrage de la Scène*, vous pouvez simplement faire un copier-coller!

Voici l'évènement une fois complété. Le vôtre devrait être identique.



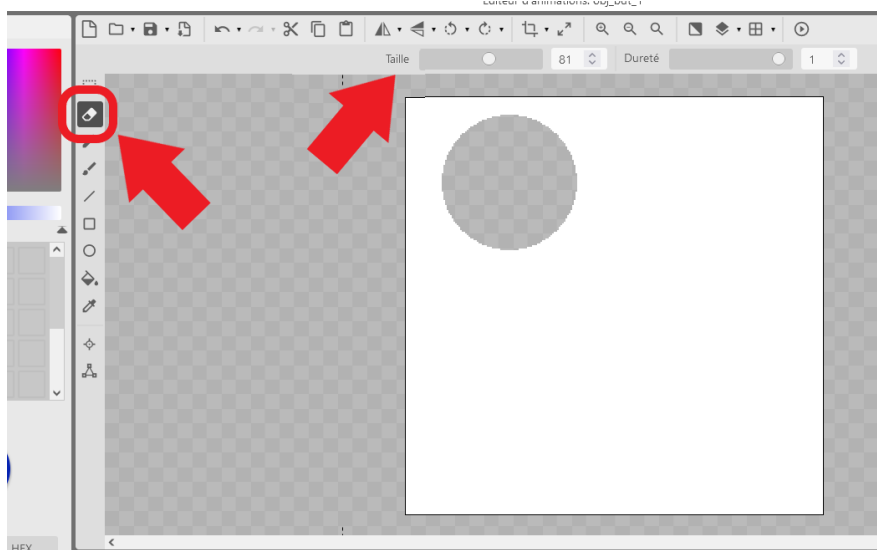
Plaçons temporairement un second **obj_but_1** derrière le joueur 2 pour tester que tout fonctionne correctement.

Et comme par magie, la balle se remet en jeu automatiquement lorsqu'elle entre en collision avec un but, et elle rebondit sur les murs, en plus de prendre un angle de rebond lorsque le joueur la renvoie!

Nous avons fait beaucoup de chemin, et nous l'objectif final se rapproche dangereusement!

Nous pouvons donc simplement dupliquer notre **obj_but_1** en faisant un clic-droit dans le panneau de projet à droite, et sélectionner '*Cloner*'. Un objet sera automatiquement créé, **obj_but_2**. Selon moi, le nom est parfait!

Comme pour la première zone de buts, nous allons glisser-déposer un **obj_but_2** sur la scène derrière le joueur 2, puis l'étirer pour qu'il prenne toute la hauteur de la scène. Ensuite :



➔ Double-cliquez sur l'objet **obj_but_1** dans l'onglet Projet, ce qui ouvrira l'éditeur de *Sprite*

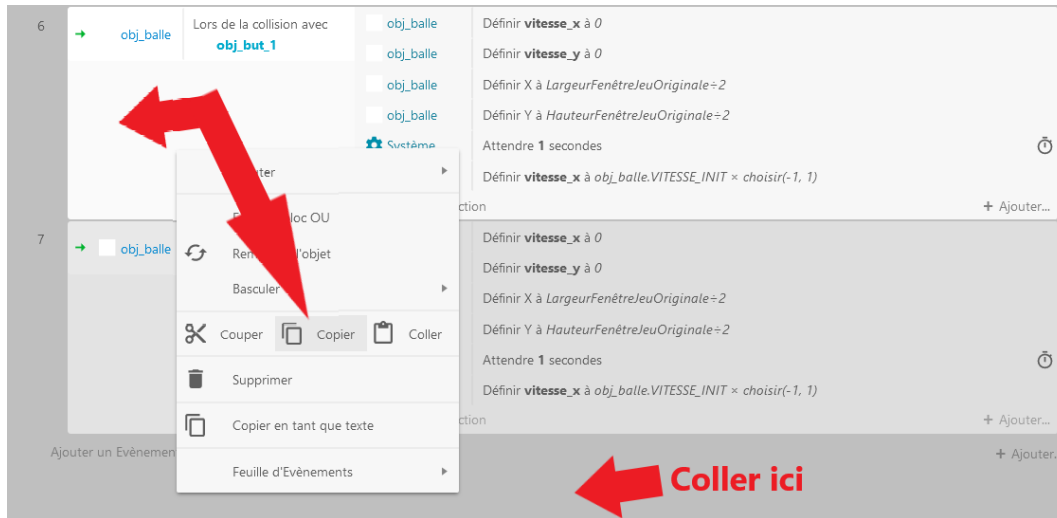
➔ Sélectionnez l'outil *Effacer*

➔ Agrandissez la taille de l'efface, puis effacez toute la couleur pour obtenir un sprite totalement transparent.

➔ Fermez l'éditeur de *Sprite*

➔ Faites la même chose avec **obj_but_2**

Les buts semblent avoir disparu, mais ils sont toujours bien dans la scène, ne vous inquiètes pas. Ils sont encore sélectionnables en cliquant sur leur position. Retournons à l'onglet *Feuille d'Évènement*. Il s'agit ici d'une formalité :



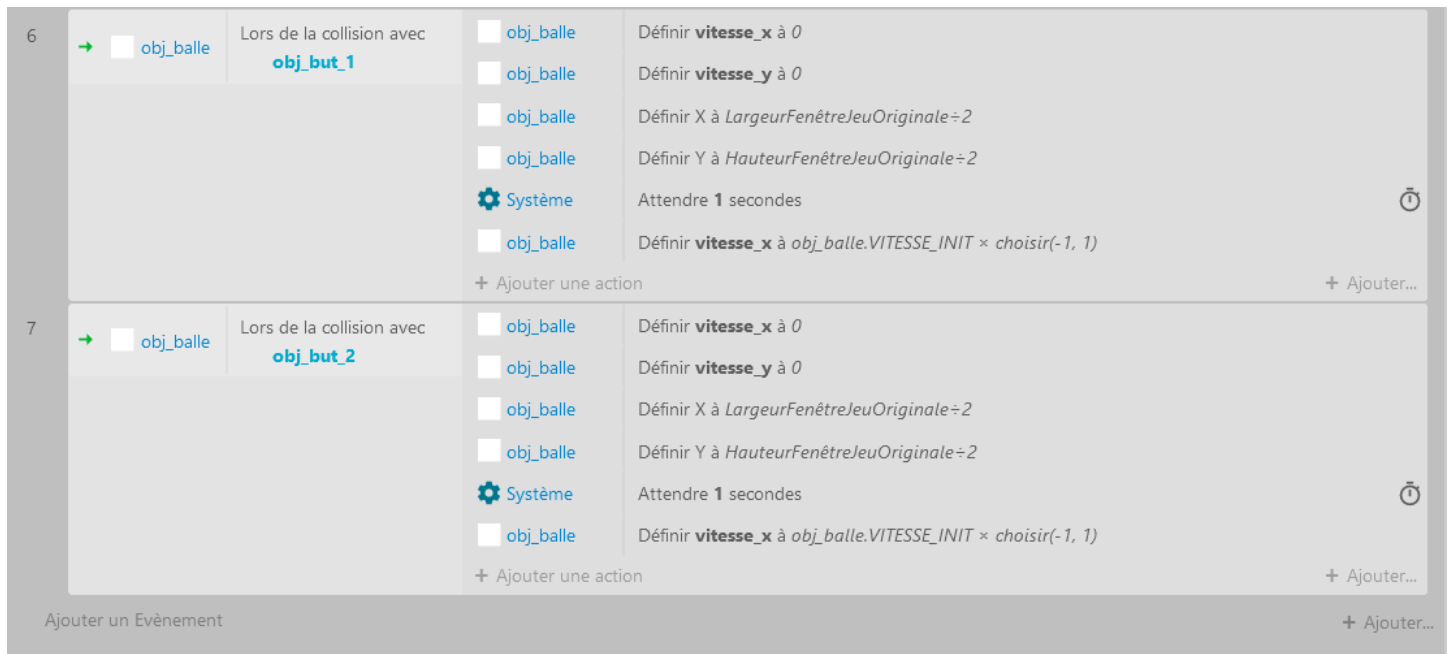
➔ Cliquez-droit sur l'évènement de collision entre notre **obj_balle** et notre **obj_but_1**. Copier.

➔ Clic-droit sous la liste d'évènements. Coller

➔ Double-cliquer sur '*Lors de la collision avec obj_but_1*' sur le nouvel évènement

➔ Remplacez **obj_but_1** par **obj_but_2**, puis validez

Voici le résultat :



Un test confirme que tout fonctionne comme prévu.

Si ce n'est pas le cas, vérifiez bien toutes les étapes depuis votre dernier test.

C'est aussi un moment parfait pour sauvegarder notre projet.

Au retour, nous allons 'donner vie' à notre adversaire en lui ajoutant du mouvement.

***** ALERTE SAUVEGARDE *****

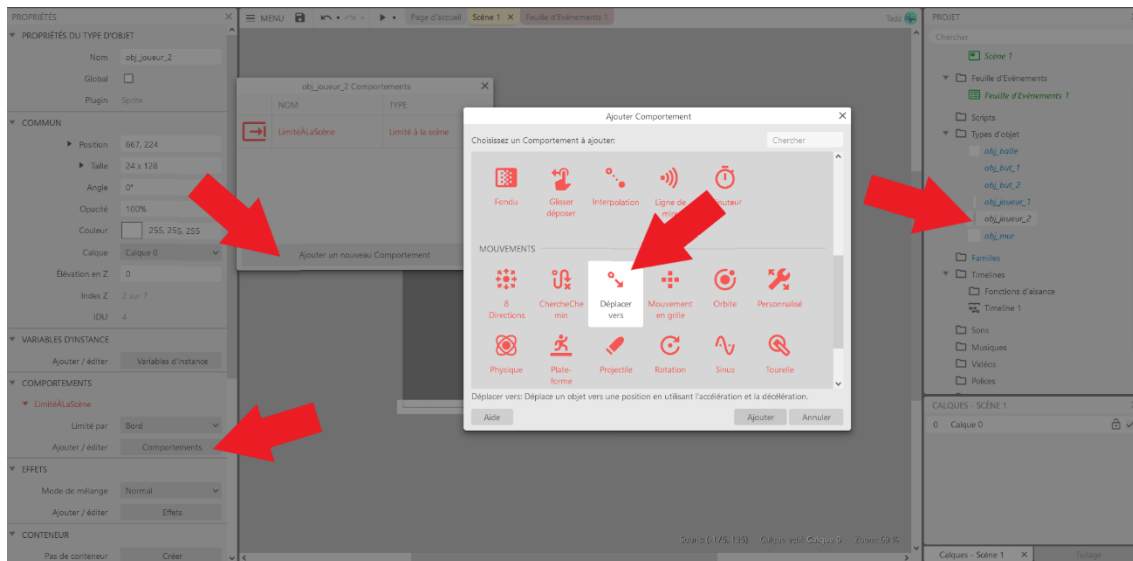
C'est un excellent moment pour sauvegarder notre travail et prendre une pause si vous en sentez le besoin!

ÉTAPE 8

L'ADVERSAIRE

Il va sans dire qu'en ce moment, notre adversaire se montre très peu enthousiaste. Il faut trouver un moyen afin qu'il suive la trajectoire de la balle. Cependant, nous ne pouvons pas simplement assigner la valeur Y de la balle à l'opposant à chaque 'tick', puisqu'il serait impossible pour nous de vaincre l'adversaire!

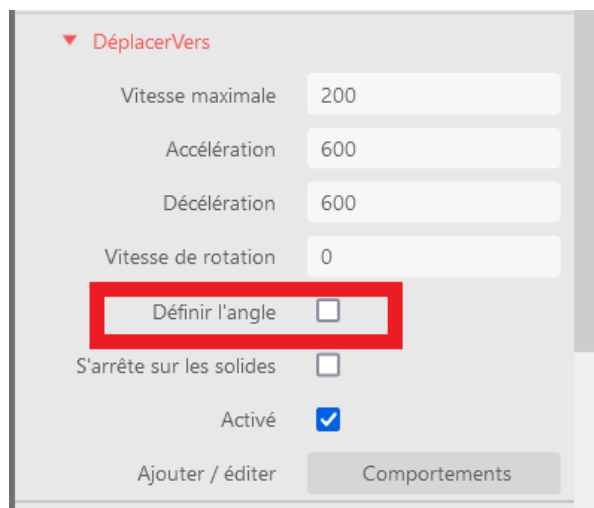
Construct offre un comportement qui permet à un objet de se déplacer vers une position avec une accélération et une décélération variable, c'est ce que nous allons utiliser.



➔ Sélectionnez **obj_joueur_2** et cliquez sur 'Comportements'

➔ Cliquez sur 'Ajouter un nouveau comportement'

➔ Sélectionnez 'Déplacer vers' dans la catégorie Mouvement. Validez votre choix.



Un nouveau comportement apparaîtra dans le panneau de Propriétés. Il faut décocher l'option **Définir l'angle**, sinon l'opposant ne restera pas 'à la verticale'.

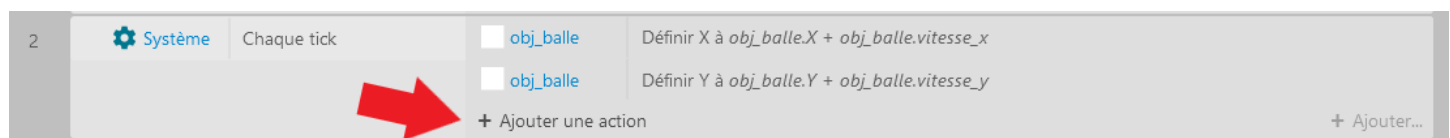
Les champs 'Accélération' et 'Décélération' s'expliquent d'eux même, il est à vous de trouver des valeurs qui vous conviennent.

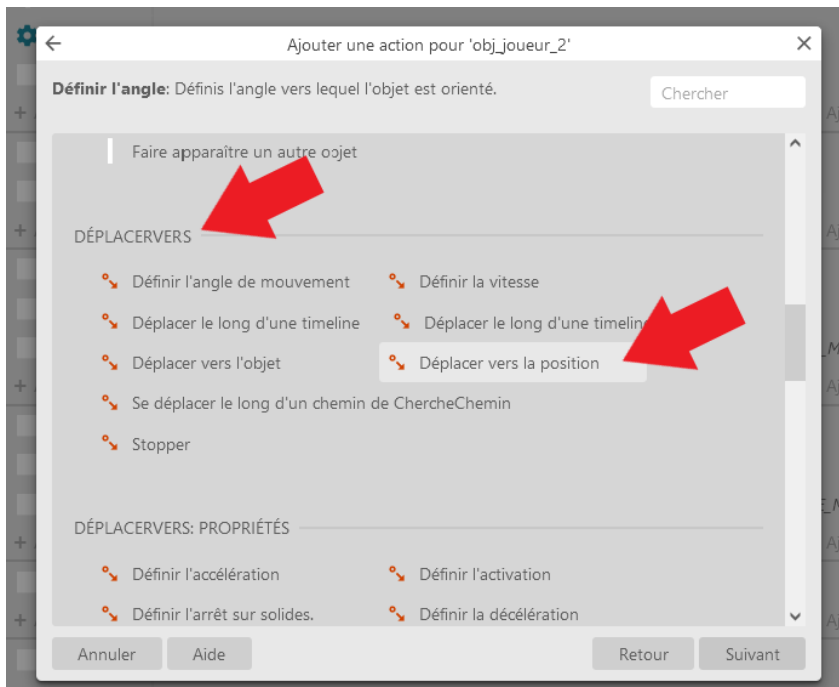
Ce genre de modification devrait toutefois attendre que toutes les mécaniques soient fonctionnelles.

En effet, rien dans ce panneau de comportement ne nous permet de dire 'déplace toi à la verticale vers la position Y de la balle'.

C'est dans la *Feuille d'Événements* que nous pourrons programmer ces actions.

Comme nous voulons que notre opposant se déplace continuellement, c'est dans l'évènement **Système – Chaque Tick** que nous devons ajouter l'action de déplacement.





➔ Une nouvelle catégorie se trouve sans la liste d'actions disponible : *Déplacer Vers*

➔ Choisissez '*Déplacer vers la position*'.

➔ Cliquez sur *Suivant*.

➔ Un nouveau panneau de paramètres apparaît. Il nous demande de lui fournir les coordonnées vers lesquelles notre **obj_joueur_2** devrait se déplacer

➔ Pour la position **X**, nous voulons toujours qu'elle soit la même. En revanche, nous sommes obligés d'entrer un paramètre.

➔ Assignons donc la valeur X vers laquelle nous voulons nous déplacer à la valeur X actuelle de **obj_joueur_2**

➔ Vous pouvez *trouver l'expression*, ou taper **obj_joueur_2.X**

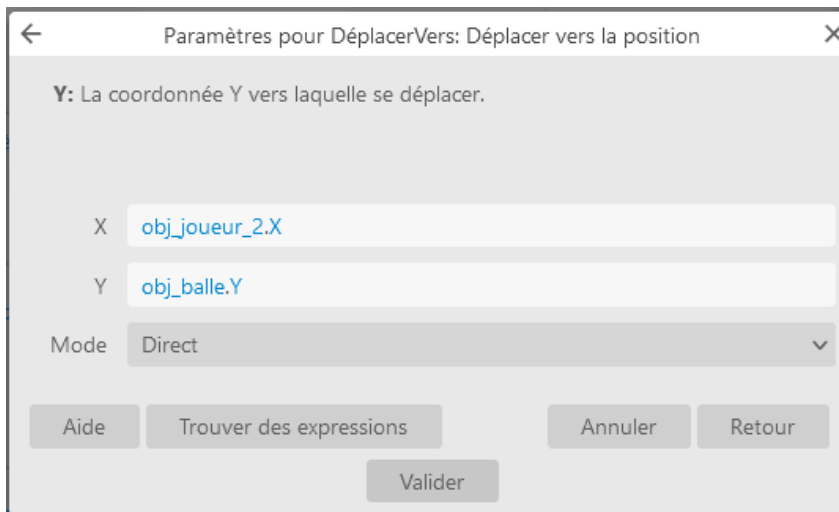
➔ Pour la valeur **Y**, nous voulons que le joueur se déplace vers la valeur y *de la balle*.

➔ Écrivez dans le champ Y **obj_balle.Y**


➔ Le mode doit rester à '*Direct*'

➔ Votre panneau de paramètres devrait être identique à celui-ci-contre.

➔ Validez.



Voici l'évènement Système – Chaque Tick :

2	 Système	Chaque tick	<input type="checkbox"/> obj_balle	Définir X à $obj_balle.X + obj_balle.vitesse_x$
			<input type="checkbox"/> obj_balle	Définir Y à $obj_balle.Y + obj_balle.vitesse_y$
			<input checked="" type="checkbox"/> obj_joueur_2	DéplacerVers: Déplacer vers ($obj_joueur_2.X$, $obj_balle.Y$) (Direct)
			+ Ajouter une action	
			+ Ajouter...	

On fait un test, et puis *oulala!*

Ça commence à ressembler à un vrai jeu!

Sauvegardez votre projet.

Nous approchons de la fin! Il ne nous manque qu'à ajouter des effets sonores et un système de score!

***** ALERTE SAUVEGARDE *****

C'est un excellent moment pour sauvegarder notre travail et prendre une pause si vous en sentez le besoin!

ÉTAPE 9

LES 'BIP BIP'

Le fait que notre adversaire ait l'air un peu plus 'vivant' est un grand pas en avant!

Bien qu'elle soit souvent sous-estimée, la prochaine étape contribuera presque autant à 'mettre de la vie' dans notre jeu que l'ajout du mouvement de l'adversaire.

Croyez-le ou non, j'ai nommé : les effets sonores!

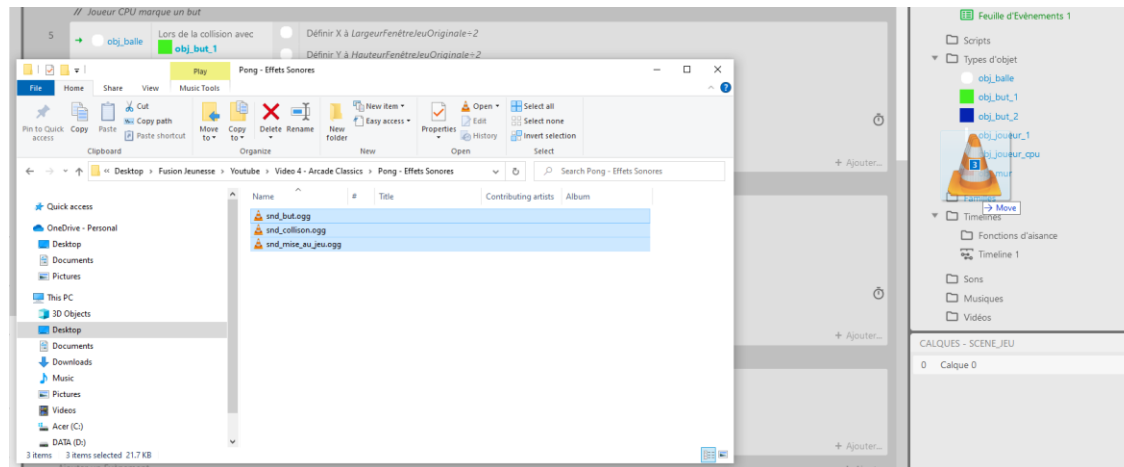
Les effets sonores que j'utilise sont disponibles dans le dossier **Pong – Effets Sonores** à l'adresse

<https://github.com/STadz/FusionJeunesse>

J'utiliserai 3 effets sonores :

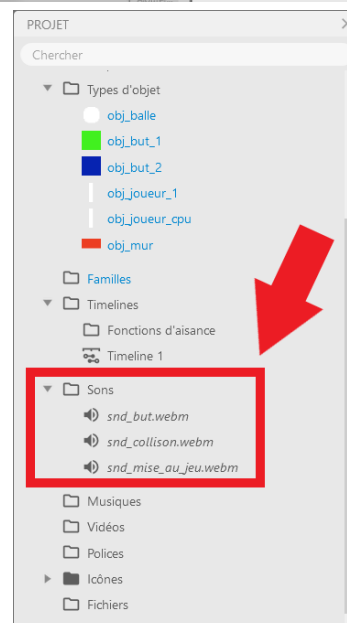
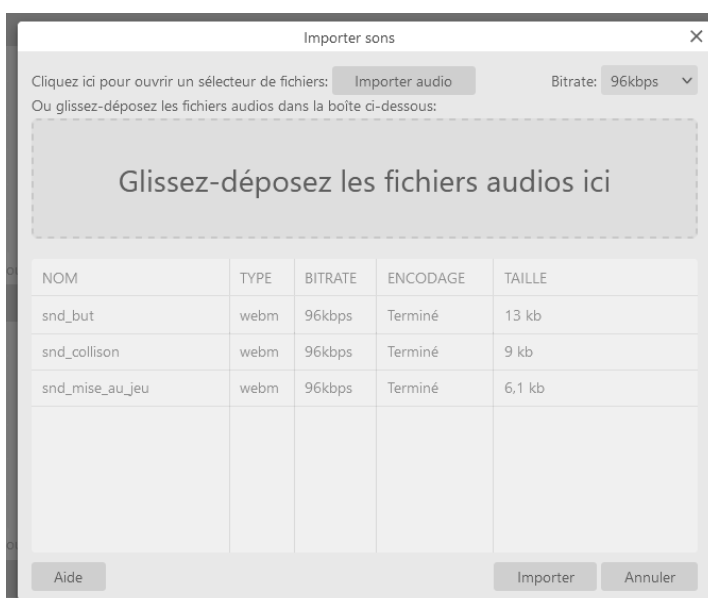
1. Quand la balle frappe la raquette d'un joueur ou un mur
2. Quand un joueur marque un point
3. Quand la balle est mise en jeu

Téléchargez les sons fournis (ou utilisez les vôtres!), et puis glisser-déposer les trois fichiers audios dans la barre de projet, à droite.



➔ Cliquez sur 'Importer' dans la fenêtre qui s'affiche.

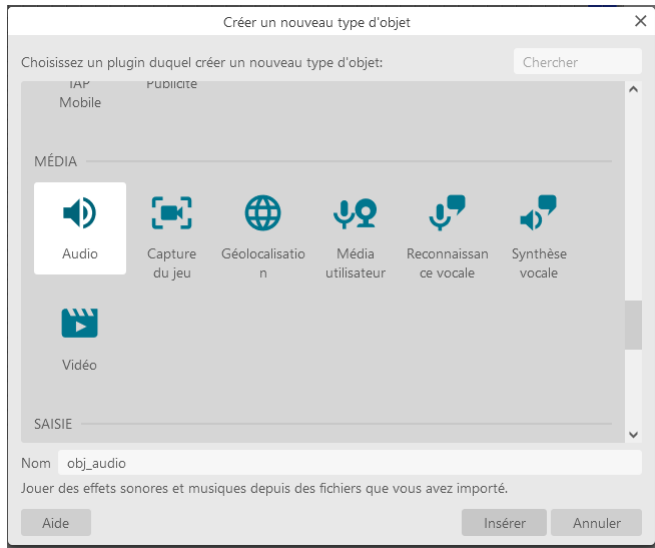
➔ Les paramètres par défaut nous conviennent.



➔ Les nouveaux sons devraient apparaître dans le panneau 'Projet'

Maintenant, on pourrait penser que jouer un fichier audio est un évènement géré par Système, mais ce n'est pas le cas.

Il nous faut créer un nouvel objet, mais cette fois il ne sera **pas** du type *Sprite*.



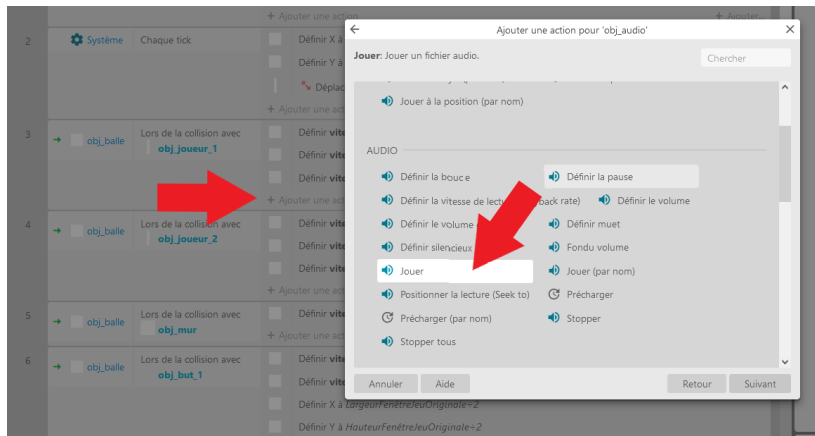
➔ Dans la section 'Média', nous allons sélectionner le type **Audio**.

➔ Nommons notre nouvel objet **obj_audio**.

➔ Nous n'avons besoin que *d'un seul* objet de ce type pour l'ensemble de nos sons.

➔ C'est tout ce que nous avons besoin de faire pour intégrer de l'audio pour toutes nos *Scènes*.

Dans la *Feuille d'évènements*, trouvez l'évènement de collision entre la balle et le joueur 1.



➔ Ajoutez une action.

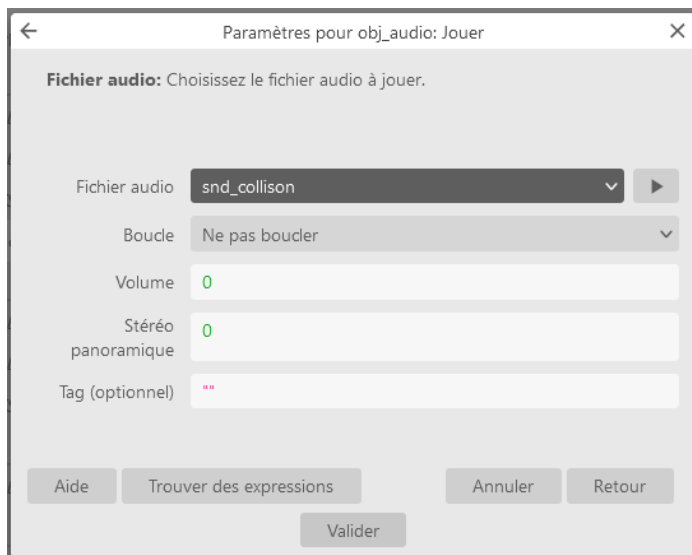
➔ Sélectionnez le nouvel **obj_audio** que nous venons de créer.

➔ Défilez jusqu'à la section 'Audio'

➔ Sélectionnez 'Jouer'

➔ Faites 'Suivant'

Voici le panneau qui s'affiche,



➔ Sélectionnez **snd_collision** comme fichier audio

➔ Assurez-vous que l'option 'Boucle' soit à 'Ne pas boucler'

➔ Assurez-vous que le volume soit à 0

➔ **Validez**



➔ Répétez la même chose avec la collision de la balle avec **obj_joueur_2**

***** NOTEZ BIEN *****

Dans le champ 'volume, 'zéro' représente le volume original. La valeur dans le champ représente le nombre de décibels que l'on ajoute ou enlève par rapport à l'audio original.

Faisons également la même chose pour la collision de la balle avec le mur, mais cette fois, afin de donner une illusion de variété et de briser la répétition, donnons la valeur **-6** au champ '*Volume*', afin que le son soit un peu moins fort lorsqu'il frappe un mur que lorsqu'il frappe une raquette.





Une fois ces opérations effectuées, la feuille d'évènement pour les collisions devrait être la suivante :

3	→ 	Lors de la collision avec 	<input type="checkbox"/> Définir vitesse_x à $obj_balle.vitesse_x \times -1$ <input type="checkbox"/> Définir vitesse_x à $obj_balle.vitesse_x + obj_balle.acceleration$ <input type="checkbox"/> Définir vitesse_y à $((SelfY - obj_joueur_1.Y) \div obj_joueur_1.Hauteur) \times obj_balle.VITESSE_MAX_Y$ <input checked="" type="checkbox"/> [Play snd_collison Ne pas boudier at volume 0 dB (stereo pan 0, tag "")] + Ajouter une action + Ajouter...
4	→ 	Lors de la collision avec 	<input type="checkbox"/> Définir vitesse_x à $obj_balle.vitesse_x \times -1$ <input type="checkbox"/> Définir vitesse_x à $obj_balle.vitesse_x + obj_balle.acceleration$ <input type="checkbox"/> Définir vitesse_y à $((SelfY - obj_joueur_2.Y) \div obj_joueur_2.Hauteur) \times obj_balle.VITESSE_MAX_Y$ <input checked="" type="checkbox"/> [Play snd_collison Ne pas boudier at volume 0 dB (stereo pan 0, tag "")] + Ajouter une action + Ajouter...
5	→ 	Lors de la collision avec 	<input type="checkbox"/> Définir vitesse_y à $obj_balle.vitesse_y \times -1$ <input checked="" type="checkbox"/> [Play snd_collison Ne pas boudier at volume -6 dB (stereo pan 0, tag "")] + Ajouter une action + Ajouter...

Nous pouvons maintenant tester notre jeu pour s'assurer que tout est fonctionnel.

Il faut noter que si vous n'entendez rien, votre fenêtre de jeu n'est peut-être pas en *focus*. Il suffit de cliquer dessus avec votre curseur pour régler le problème.

Nous pouvons maintenant répéter l'opération pour les sons **snd_but** et **snd_mise_au_jeu**.

6	→ 	Lors de la collision avec 	<input checked="" type="checkbox"/> [Play snd_but Ne pas boudier at volume 0 dB (stereo pan 0, tag "")] <input type="checkbox"/> Définir vitesse_x à 0 <input type="checkbox"/> Définir vitesse_y à 0 <input type="checkbox"/> Définir X à LargeurFenêtreJeuOriginale÷2 <input type="checkbox"/> Définir Y à HauteurFenêtreJeuOriginale÷2 <input checked="" type="checkbox"/> Attendre 1 secondes <input type="checkbox"/> Définir vitesse_x à $obj_balle.VITESSE_INIT \times choisir(-1, 1)$ <input checked="" type="checkbox"/> [Play snd_mise_au_jeu Ne pas boudier at volume 0 dB (stereo pan 0, tag "")] + Ajouter une action + Ajouter...
7	→ 	Lors de la collision avec 	<input checked="" type="checkbox"/> [Play snd_but Ne pas boudier at volume 0 dB (stereo pan 0, tag "")] <input type="checkbox"/> Définir vitesse_x à 0 <input type="checkbox"/> Définir vitesse_y à 0 <input type="checkbox"/> Définir X à LargeurFenêtreJeuOriginale÷2 <input type="checkbox"/> Définir Y à HauteurFenêtreJeuOriginale÷2 <input checked="" type="checkbox"/> Attendre 1 secondes <input type="checkbox"/> Définir vitesse_x à $obj_balle.VITESSE_INIT \times choisir(-1, 1)$ <input checked="" type="checkbox"/> [Play snd_mise_au_jeu Ne pas boudier at volume 0 dB (stereo pan 0, tag "")] + Ajouter une action + Ajouter...

➔ Pour les collisions avec les **obj_but_1** et **2**, il faut ajouter 2 sons différents : celui du but et celui de la mise au jeu

➔ Attention! Le son **snd_but** doit être au-dessus de l'action 'Attendre'

➔ Attention! Le son **snd_mise_au_jeu** doit être en-dessous de l'action 'Attendre'

➔ Vous pouvez utiliser le glisser-déposer pour réorganiser l'ordre des actions

Finalement, faites jouer le son **snd_mise_au_jeu** lors de la mise au jeu initiale, à la fin de l'évènement Système '*Au démarrage de la scène*'. Et voilà, vous pouvez maintenant tester votre jeu! Il e reste qu'une étape : le tableau des scores.

***** ALERTE SAUVEGARDE *****

C'est un excellent moment pour sauvegarder notre travail et prendre une pause si vous en sentez le besoin!

ÉTAPE 10

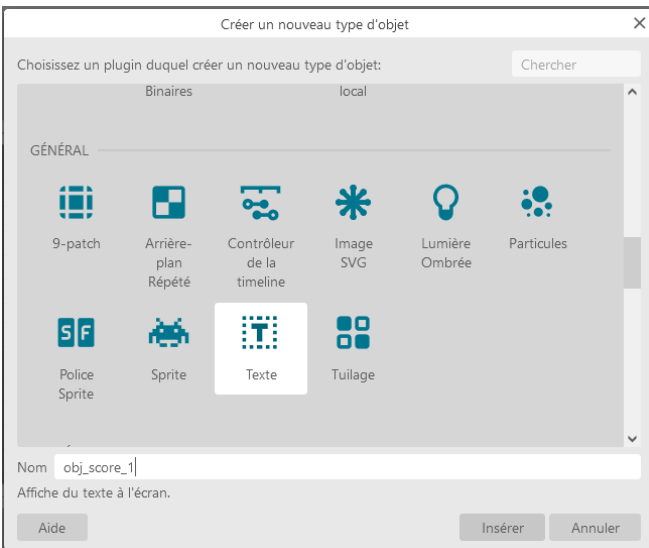
LE TABLEAU DES SCORES

Nous en sommes presque à la fin. Notre jeu est fonctionnel, mais nous n'avons pas encore défini de conditions de victoire, et rien ne comptabilise ni n'affiche le score des joueurs. Heureusement, l'*interface utilisateur* dans Pong est extrêmement simple : le score de chaque joueur est affiché en haut de l'écran, au-dessus de sa raquette.

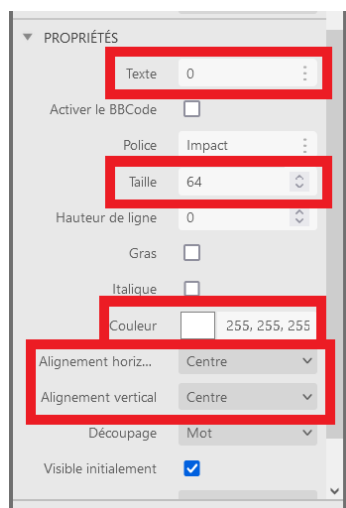
Nous allons créer un nouvel objet pour afficher le score, mais cette fois nous ne sélectionnerons *pas* le type *Sprite*. Construct 3 nous offre un type d'objet mieux adapté à la situation : l'objet *Texte*.

➔ Créons un nouvel objet de type *Texte* et nommons le **obj_score_1**. Cet objet aura comme fonction d'afficher le score du joueur 1.

➔ Positionnez le nouvel objet au-dessus du joueur 1 dans la scène.

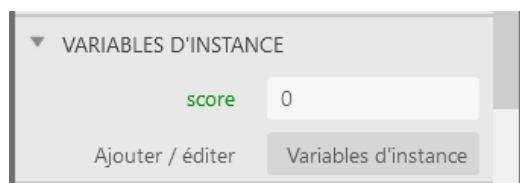


Avec votre **obj_texte_1** sélectionné, ouvrez ses *Propriétés*.



- ➔ Inscrivez le chiffre 'zéro' dans le champ 'Texte'
- ➔ Choisissez une police de caractère de votre choix. Ici, j'ai sélectionné 'Impact'
- ➔ Agrandissez la taille de police. La taille par défaut est beaucoup trop petite pour notre jeu. Une valeur de 64 est un bon point de départ.
- ➔ Modifiez la couleur du texte pour le mettre en blanc
- ➔ Modifiez l'alignement vertical et horizontal pour 'Centre', afin que le texte soit au milieu de notre objet.
- ➔ Assurez-vous que la case 'Visible initialement' est bien cochée.

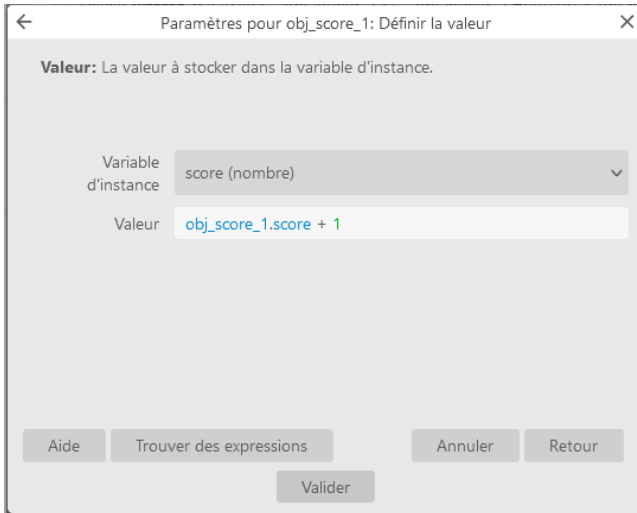
Toujours dans le panneau de Propriétés, trouvez la section '*Variables d'Instance*'



➔ Il faut ajouter une variable à notre objet texte qui représentera le **score**.

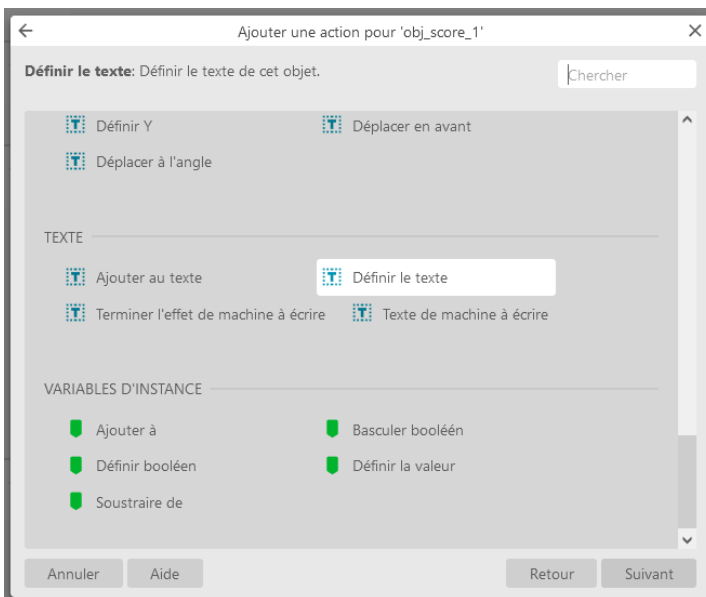
Une fois votre police de caractère ajustée et votre variable **score** créée, rendez-vous sur la Feuille d'Événements afin de programmer la logique de notre nouvel objet.

Puisqu'il s'agit de l'objet représentant le score du joueur 1, modifions l'évènement de *Collision* entre **obj_balle** et **obj_but_2**.

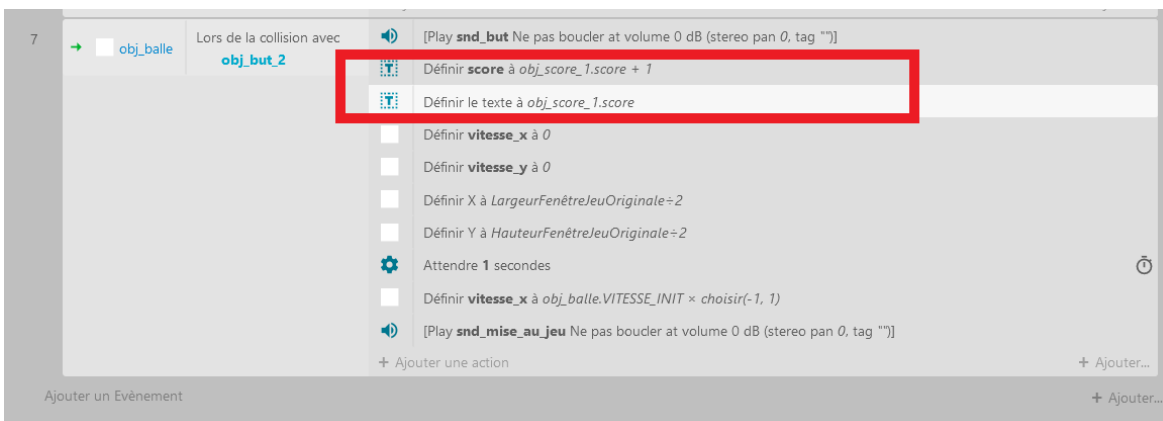


- ➔ Cliquez sur 'Ajouter une action'
- ➔ Choisissez **obj_score_1**
- ➔ Dans la section 'Variables', choisissez 'Définir la valeur'
- ➔ Sélectionnez la variable **score** que nous avons créée dans le menu déroulant
- ➔ Additionnons 1 à la valeur du score actuelle, soit en tapant **obj_score_1.score + 1**
- ➔ Validez

Notre variable **score** contient maintenant la valeur actuelle du score du joueur 1, mais nous n'avons pas demandé à Construct de changer le texte de notre objet afin de refléter le nouveau score. Toujours dans le même évènement :

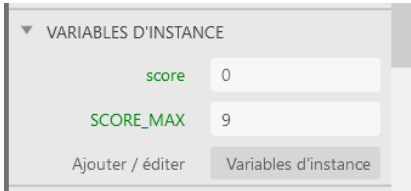


- ➔ Cliquez sur 'Ajouter une action'
- ➔ Choisissez **obj_score_1**
- ➔ Dans la section 'Variables', choisissez 'Définir le Texte'
- ➔ Cliquez sur 'Suivant'
- ➔ Dans le champ 'Texte', trouvez la variable **score** de l'objet **obj_score_1**, ou écrivez simplement **obj_score_1.score**
- ➔ Validez
- ➔ Glissez-déposer les deux nouvelles actions *au-dessus* de l'action 'Attendre', sinon le score ne se mettra à jour que lorsque la balle est remise en jeu.

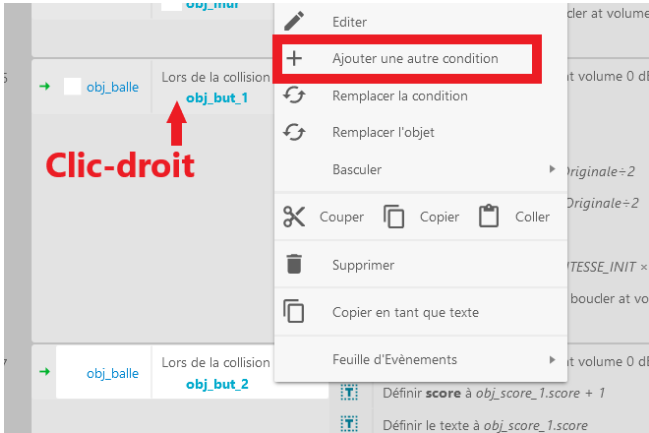


Testez le jeu et observez ce qui se passe lorsque vous marquez un point.
Parfait!

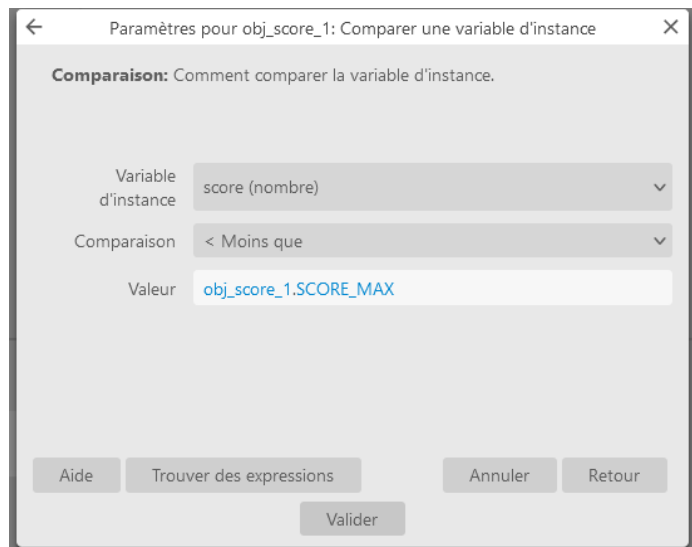
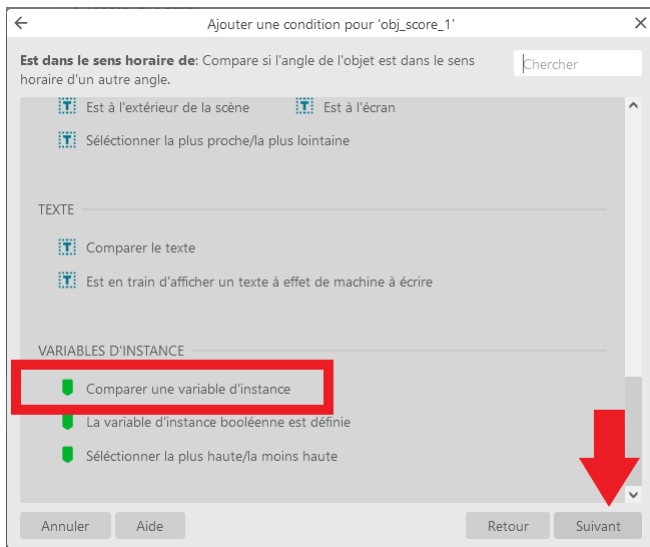
C'est bien, mais la partie ne se terminera jamais, même si nous marquons un million de points. Créons une nouvelle variable **MAX_SCORE**, qui représentera le nombre de points nécessaires pour gagner la partie.



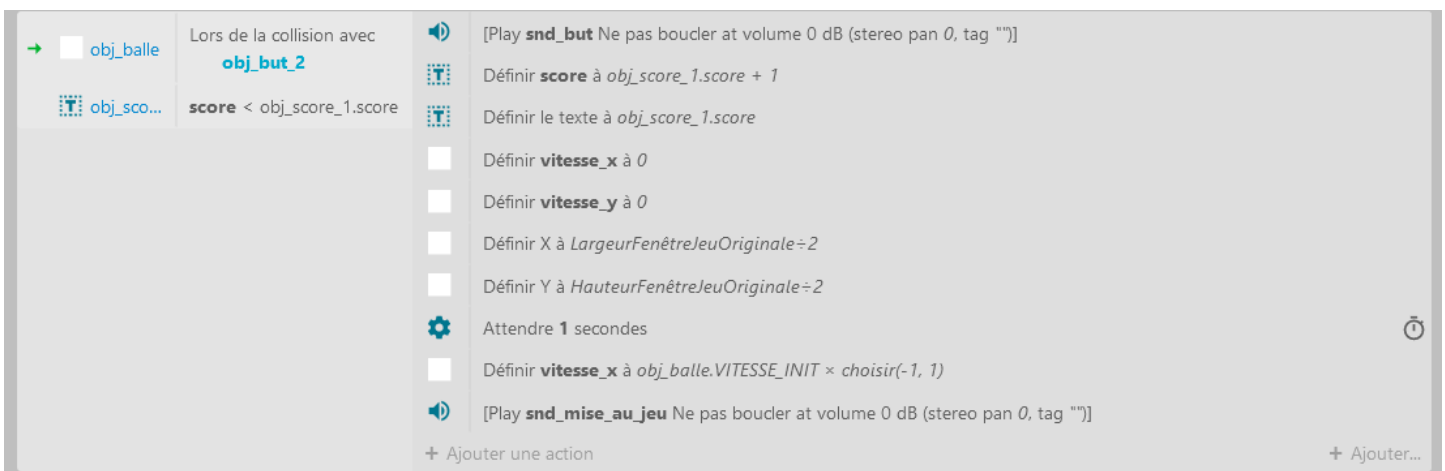
Dans la Feuille d'Évènements



- ➔ Faites un clic-droit sur l'action '*Lors de la collision avec obj_but_2*'
- ➔ Sélectionnez '*Ajouter une autre condition*'
- ➔ Pour **obj_score_1**, choisissez '*Comparer une variable d'instance*'
- ➔ Choisissez la variable **score** dans le menu déroulant
- ➔ Choisissez l'option '*Moins que*'
- ➔ Dans le champ valeur, entrez notre nouvelle variable que nous avons créée : **obj_score_1.SCORE_MAX**

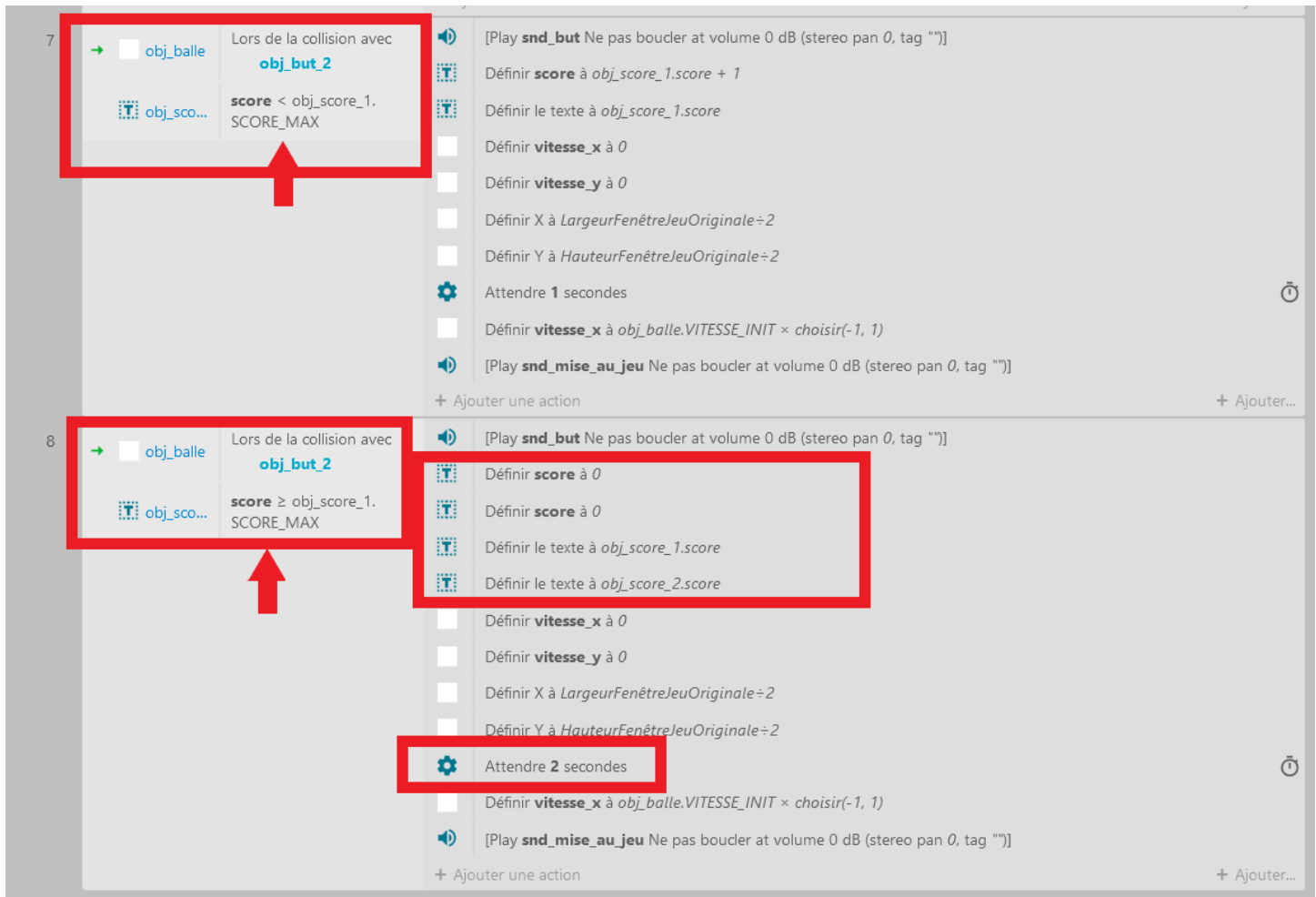


Votre évènement ressemble alors à ceci :



Nous venons de programmer ce qui se passe lorsque le joueur 1 marque un point, mais ne gagne pas la partie. Dupliquons l'évènement de collision, et changeons simplement la seconde condition pour vérifier si le score est '*Plus grand ou égal*' à notre variable **MAX_SCORE**.

Il nous faut aussi remettre le score à zéro dans cet évènement dupliqué, puisque c'est la fin de la partie. J'ai également mis une attente légèrement plus longue à la fin de la partie, afin de donner un moment de répit pour savourer la victoire. Une fois cela fait, votre Feuille d'Évènements devrait se trouver dans l'état suivant :



*** ALERTE SAUVEGARDE ***














C'est un excellent moment pour sauvegarder notre travail et prendre une pause si vous en sentez le besoin!

Maintenant, c'est à vous de créer un **obj_score_2** pour représenter le score de l'adversaire. Vous pouvez répéter les mêmes étapes, ou bien simplement *cloner* votre **obj_score_1** et copier-coller les évènements que nous venons de programmer.
















Faites bien attention aux détails, il est facile de se tromper lorsque l'on choisit nos variables!

La prochaine page montre les 4 évènements de collision Balle-But que nous avons.














// Le joueur 2 marque

6	 obj_balle	Lors de la collision avec obj_but_1	 [Play snd_but Ne pas boucler at volume 0 dB (stereo pan 0, tag "")]
	 obj_sco...	score < obj_score_2. SCORE_MAX	 Définir score à obj_score_2.score + 1
			 Définir le texte à obj_score_2.score
			 Définir vitesse_x à 0
			 Définir vitesse_y à 0
			 Définir X à LargeurFenêtreJeuOriginale ÷2
			 Définir Y à HauteurFenêtreJeuOriginale ÷2
			 Attendre 1 secondes 
			 Définir vitesse_x à obj_balle.VITESSE_INIT × choisir(-1, 1)
			 [Play snd_mise_au_jeu Ne pas boucler at volume 0 dB (stereo pan 0, tag "")]
			+ Ajouter une action + Ajouter...
















// Le joueur 2 gagne

7	 obj_balle	Lors de la collision avec obj_but_1	 [Play snd_but Ne pas boucler at volume 0 dB (stereo pan 0, tag "")]
	 obj_sco...	score ≥ obj_score_2. SCORE_MAX	 Définir score à 0
			 Définir score à 0
			 Définir le texte à obj_score_1.score
			 Définir le texte à obj_score_2.score
			 Définir vitesse_x à 0
			 Définir vitesse_y à 0
			 Définir X à LargeurFenêtreJeuOriginale ÷2
			 Définir Y à HauteurFenêtreJeuOriginale ÷2
			 Attendre 2 secondes 
			 Définir vitesse_x à obj_balle.VITESSE_INIT × choisir(-1, 1)
			 [Play snd_mise_au_jeu Ne pas boucler at volume 0 dB (stereo pan 0, tag "")]
			+ Ajouter une action + Ajouter...

// Le joueur 1 marque

8	 obj_balle	Lors de la collision avec obj_but_2	 [Play snd_but Ne pas boucler at volume 0 dB (stereo pan 0, tag "")]
	 obj_sco...	score < obj_score_1.score	 Définir score à obj_score_1.score + 1
			 Définir le texte à obj_score_1.score
			 Définir vitesse_x à 0
			 Définir vitesse_y à 0
			 Définir X à LargeurFenêtreJeuOriginale ÷2
			 Définir Y à HauteurFenêtreJeuOriginale ÷2
			 Attendre 1 secondes 
			 Définir vitesse_x à obj_balle.VITESSE_INIT × choisir(-1, 1)
			 [Play snd_mise_au_jeu Ne pas boucler at volume 0 dB (stereo pan 0, tag "")]
			+ Ajouter une action + Ajouter...

// Le joueur 1 gagne

9	 obj_balle	Lors de la collision avec obj_but_2	 [Play snd_but Ne pas boucler at volume 0 dB (stereo pan 0, tag "")]
	 obj_sco...	score ≥ obj_score_1. SCORE_MAX	 Définir score à 0
			 Définir score à 0
			 Définir le texte à obj_score_1.score
			 Définir le texte à obj_score_2.score
			 Définir vitesse_x à 0
			 Définir vitesse_y à 0
			 Définir X à LargeurFenêtreJeuOriginale ÷2
			 Définir Y à HauteurFenêtreJeuOriginale ÷2
			 Attendre 2 secondes 
			 Définir vitesse_x à obj_balle.VITESSE_INIT × choisir(-1, 1)
			 [Play snd_mise_au_jeu Ne pas boucler at volume 0 dB (stereo pan 0, tag "")]
			+ Ajouter une action + Ajouter...

Ajouter un Evènement

+ Ajouter...

Notre jeu est presque terminé!

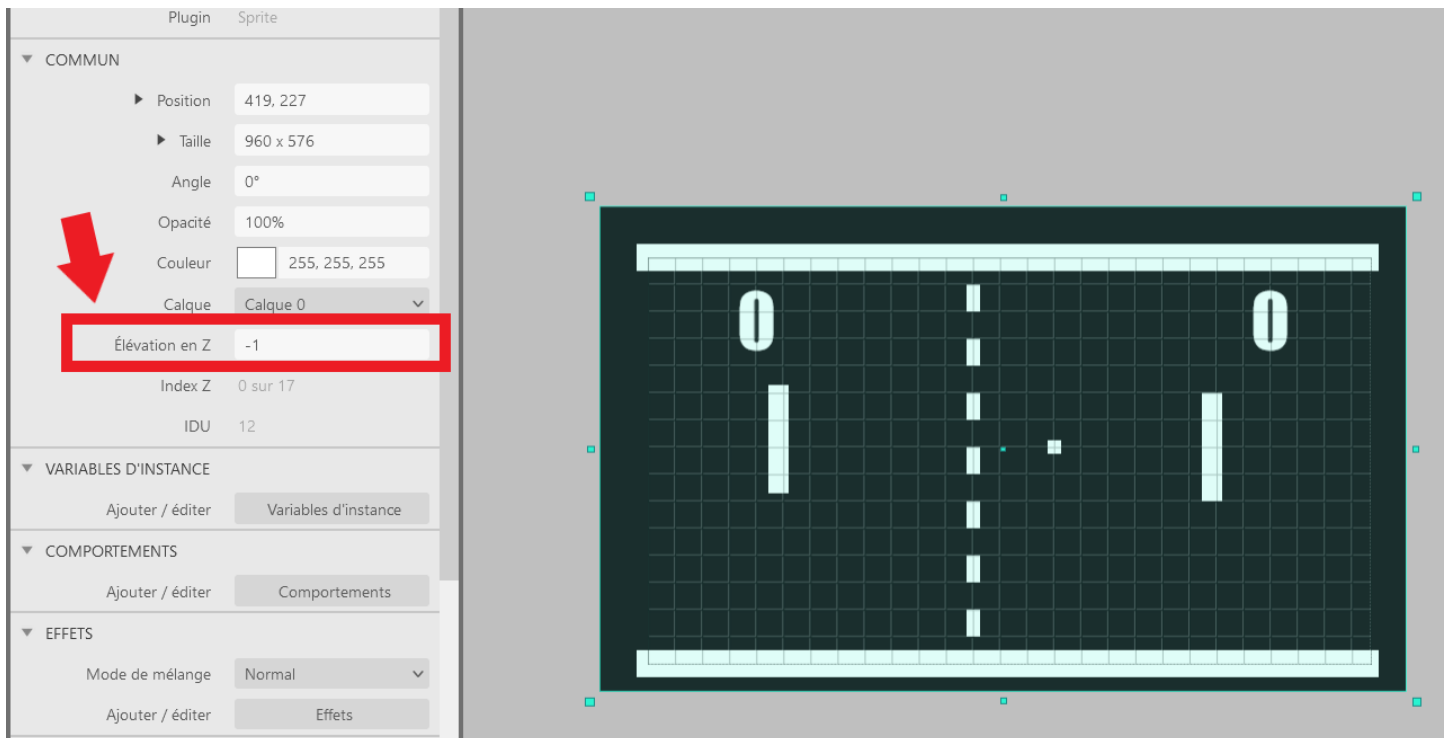
En fait, techniquement parlant, il l'est déjà, car nous n'allons pas ajouter de nouvelles mécaniques ou comportements. Cependant, pour rester fidèle à la version originale, nous pourrions mettre un dernier effort afin de mettre un arrière-plan noir, ainsi qu'une ligne centrale.

Cette étape est triviale, puisqu'il ne s'agit que de créer de nouveaux *Sprites*, sans leur ajouter de comportement.

Pour s'assurer que l'arrière-plan ne cache pas nos joueurs, il faut *baisser l'Élévation en Z* de notre arrière-plan. Vous trouvez cette option dans le panneau de propriétés.

Comme tous nos autres *sprites* ont une élévation de 0, nous pouvons simplement la mettre à -1.

Et voilà!



Le fichier du projet est disponible, ainsi que les sons utilisés sur Github.

<https://github.com/STadz/FusionJeunesse>

Notes

Cette démonstration visait à vous initier à la logique de programmation et aux concepts généraux de Construct 3. En aucun cas ce code est-il optimal ni ne prétend l'être. Il a été créé de cette manière afin d'expliquer la logique du code de manière, je l'espère, plus claire.

En prenant de l'expérience (si Pong ne vous a pas découragé!), vous apprendrez petit à petit des techniques qui vous rendront la vie beaucoup plus simple et qui vous permettront de créer du code robuste et efficace. Il ne faut pas vouloir aller trop vite!

Défis

Je propose ici quelques défis pour ceux qui voudraient aller plus loin avec leur version de Pong :

1. Créez un menu principal permettant de choisir entre les modes 'Humain vs. Humain' ou 'Humain vs. CPU'
2. Donnez un léger angle (au hasard) à la balle lors de la mise en jeu.
3. Corrigez le bug potentiel qui fait que la balle pourrait sortir de l'écran et ne plus jamais revenir. Truc : il s'agit ici de donner une vitesse maximale à la balle et de ne lui ajouter de l'accélération que si sa vitesse est sous la vitesse maximale
4. Mettez deux balles en jeu
5. Ajoutez des débris flottants qui feront dévier la balle si elle le frappe, avant de se détruire (pensez au jeu *Astéroid*)
6. Donnez un temps maximum à une partie : le joueur avec le plus de points à la fin de la partie gagne.

Crédits, notes et droits d'utilisation

Ce document, le code Construct 3 et JavaScript ainsi que la vidéo de présentation s'y rapportant ont été créés par Simon Taddio pour l'utilisation des élèves participants au programme Fusion Jeunesse.

Ces éléments NE SONT PAS monétisés, et sont distribués sous la licence MIT. Si vous avez payé pour ce contenu, veuillez le rapporter à l'adresse courriel suivante :

staddio@fusionjeunesse.org

Il est permis de consulter, modifier, briser, réparer, re-briser, expérimenter et, en un mot, faire ce que bon vous semble avec le code Construct3 associé à ce tutoriel.

Tous les sons et images qui n'ont pas été créés spécifiquement pour ce tutoriel proviennent du site :

www.kenney.nl

et leur licence VOUS PERMET d'utiliser et de modifier ces ressources pour vos projets personnels, commerciaux, éducatif et travaux de recherche. En un mot, vous ne pouvez simplement pas les revendre tel-quel.