David Weinachter

Cahier d'activités PYTHON pour les KIDS

Dès 10 ans

EYROLLES

Cahier d'activités PYTHON pour les KIDS

Apprends à coder en Python en t'amusant !

Avec le livre que tu tiens entre les mains, tu vas apprendre à coder ton propre jeu vidéo en Python, l'un des langages de développement les plus populaires du monde. Pas d'inquiétude : aucun savoir-faire préalable n'est demandé. Si c'est ta première expérience au sein du monde magique des développeurs, pas de panique : tout te sera expliqué de façon détaillée et, en cas de besoin, les différentes solutions te seront présentées au fur et à mesure. Tu trouveras dans cet ouvrage six chapitres de difficulté croissante pour créer ton premier jeu vidéo. Et tu vas même pouvoir inventer et cacher tes propres codes de triche pour le personnaliser !

Caractéristiques du jeu

- Deux joueurs : le premier qui tire un obus sur le tank de l'adversaire gagne
- Véritable moteur physique (gravité, météo, collisions)
- Codes de triche (gagner directement, modifier la direction de l'obus...)

L'auteur

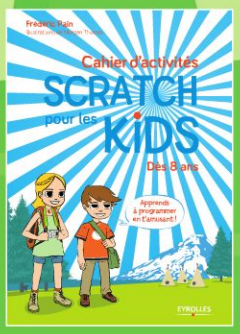
David Weinachter est un ingénieur informatique qui a toujours été passionné par la programmation : c'est pour transmettre sa passion à ses enfants qu'il a créé Kidscod.in, la première méthode en ligne pour apprendre aux enfants à coder en toute autonomie, dès qu'ils commencent à savoir lire. Convaincu qu'on apprend mieux en s'amusant, David s'attache à faire découvrir le code aux enfants via des histoires interactives ou des jeux vidéo à créer.

À qui s'adresse ce livre ?

Aux enfants (dès 10 ans), parents et enseignants !

Sur www.kidscod.in/app/tank

Crée, personnalise et partage ton propre jeu de tank en suivant les indications de ce livre.



www.editions-eyrolles.com

Dans la même collection

David Weinachter

Cahier d'activités PYTHON pour les KIDS

EYROLLES

Groupe Eyrolles
61, bd Saint-Germain
75240 Paris Cedex 05

www.editions-eyrolles.com

Pour Delhia, Ethan et Matthéo, merci pour votre enthousiasme
et vos séances de tests acharnées !

Pour Angy, qui a toujours su que j'écrirai un livre un jour
et sans qui rien n'aurait été possible.

Je vous aime

ISBN : 978-2-212-14332-4

Sommaire

Avant de te lancer dans la création de ton jeu vidéo

De nos jours, les créateurs de jeux vidéo sont souvent considérés comme des magiciens : avec juste un peu de code informatique en guise de baguette magique, ils arrivent à animer des petits pixels, à raconter des histoires fantastiques et même à créer des émotions. Quand on sait que les ordinateurs (tout comme les consoles, tablettes ou smartphones) n'utilisent que des 0 et des 1 pour coder l'information, il y a de quoi trouver cela magique !

Si tu fais partie de ceux qui aimeraient devenir un de ces sorciers du code, j'ai une très bonne nouvelle : le livre que tu tiens entre les mains est un véritable grimoire magique pour débutants. Divisé en six chapitres qui correspondent à six exercices, ce cahier d'activités va t'apprendre à créer ton premier jeu vidéo, pas à pas.

Il est donc destiné à tous les petits curieux qui ont envie d'apprendre à créer un jeu vidéo. Et pas d'inquiétude : aucun savoir-faire préalable n'est demandé. Bien sûr, si tu as déjà commencé à programmer, certaines choses te sembleront faciles et évidentes. Mais si c'est ta première expérience au sein du monde magique des développeurs, pas de panique : tout te sera expliqué de façon détaillée et, en cas de besoin, les différentes solutions te seront présentées au fur et à mesure.

Je ne t'ai pas encore tout dit concernant ce cahier : il ne révélera son entière puissance que s'il est accompagné d'un site Internet (http://www.kidscod.in/app/tank). Celui-ci t'accompagnera pendant tout ton apprentissage : c'est sur ce site que tu pourras créer ton jeu vidéo, en six étapes de difficulté croissante. Tu peux lire ce livre sans jamais aller sur le site, et tu apprendras sûrement des choses. Mais le mieux est d'avoir accès au site Internet parallèlement à ta lecture. Ainsi, tu pourras mettre en exécution immédiatement ce que tu lis et apprends. Et dans la mesure où le but est ici de t'apprendre à créer un jeu vidéo, il serait dommage de ne pas

pouvoir tester ton jeu vidéo au fur et à mesure de sa construction. Dans l'immédiat, nul besoin de se connecter à Internet : nous verrons tout cela au premier chapitre.

Comme tous les grimoires, celui que tu tiens entre les mains est rempli de caractères étranges : des caractères spéciaux qui te sembleront à la fois familiers et complètement bizarres. C'est tout simplement parce que tu vas apprendre à coder ton jeu en langage *Python*. Ce langage est utilisé aujourd'hui dans de nombreux domaines, y compris pour la création de jeux vidéo. Là encore pas d'inquiétude, chaque morceau de code en Python te sera expliqué en détail. Par exemple, un morceau de code Python te sera présenté de cette façon :

```
Ceci est un code Python (1)
Qui ne fait rien et qui n'est
Même pas formaté correctement.(2)
C'est une honte d'écrire n'importe quoi
Dans un exemple de code. Toi le jeune
Apprenti, ne t'amuse pas à faire de même (3)
Sous peine de te retrouver avec un jeu vidéo
Qui ne fonctionnera jamais… (4)
```

Si tu regardes attentivement ce texte, tu verras qu'il contient des numéros (1), (2), (3) et (4) : ils te serviront à comprendre de quelle partie du code on te parle quand on t'expliquera des extraits de code.

Il est bientôt temps de te lancer dans ton apprentissage magique du code, mais auparavant, laisse-moi te donner deux derniers conseils.

- Lis bien chaque début de chapitre : il y sera clairement expliqué ce qu'on attend de toi. N'hésite pas à relire les introductions de chapitre si jamais tu te sens perdu pendant ton apprentissage.
- Si jamais tu devais te perdre dans le monde merveilleux des codeurs, tu pourras toujours utiliser un super pouvoir que je vais t'enseigner : en cas de besoin, envoie un e-mail à l'adresse david@kidscod.in, en précisant le chapitre où tu t'es perdu et la dernière chose que tu as réussi à coder. Je t'aiderai personnellement à te sortir de n'importe quelle situation délicate. Moi-même je suis passé par là, et je me ferai une joie de t'aider.

Voilà, je crois que tu as en mains tous les atouts pour te lancer sereinement dans ton apprentissage : il ne me reste plus qu'à te donner rendez-vous au chapitre 1.

À très bientôt, jeune apprenti codeur !

Pssst

Comme pour toute quête, tu seras peut-être rassuré si tu es accompagné. Alors pourquoi ne pas convaincre ton (ou ta) meilleure ami(e) de se joindre à toi ? En plus, le jeu vidéo que tu vas créer est spécialement conçu pour deux joueurs. Alors tu sais ce qu'il te reste à faire...

Un jeu vidéo que tu pourras bientôt créer toi-même

Qu'est-ce que Kidscod.in ?

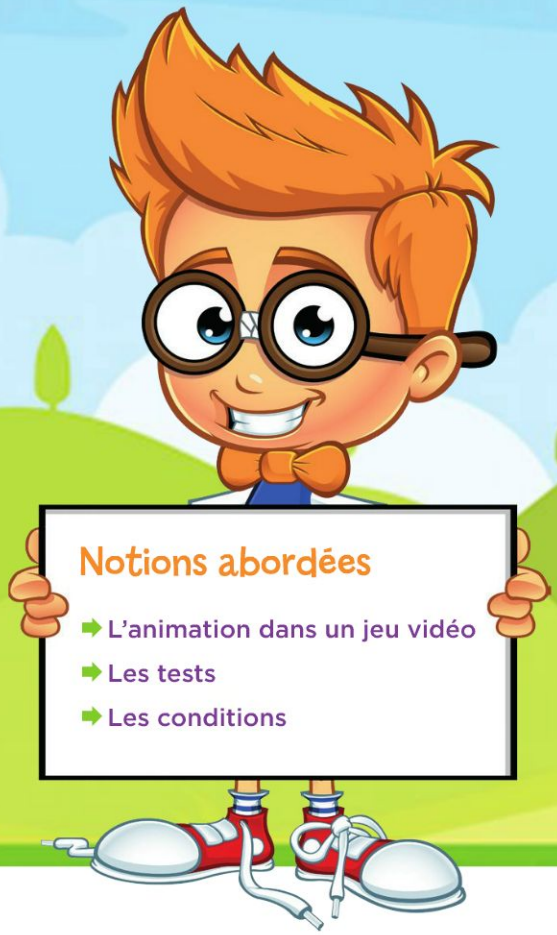
Kidscod.in est une méthode en ligne (http://www.kidscod.in) pour apprendre aux enfants de 7 à 77 ans les fondements de la programmation.

- L'apprentissage de la programmation s'y fait grâce à un langage de programmation visuel, reposant sur des blocs logiques à imbriquer.

- Chacun peut apprendre à son rythme, en toute autonomie, grâce à une alternance de séquences animées, d'exercices guidés et d'exercices libres.
- Kidscod.in est le tremplin parfait pour s'orienter ensuite vers des langages de programmation modernes tels que Python ou JavaScript.

Ce cahier d'activités s'appuie sur Kidscod.in pour aborder le langage Python de façon novatrice : chaque brique de code déposée sur l'espace de travail génère automatiquement le code Python correspondant, afin de favoriser la découverte de la syntaxe et de la structure du code Python, de façon visuelle et ludique.

N'hésite pas à visiter le site http://www.kidscod.in pour en savoir plus et rejoindre la communauté grandissante des codeurs en herbe !

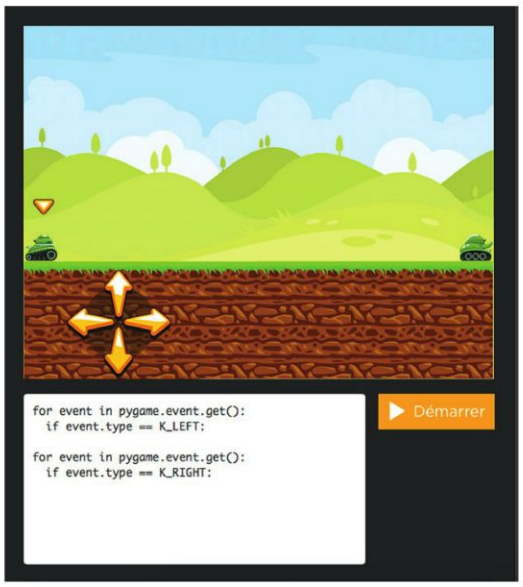


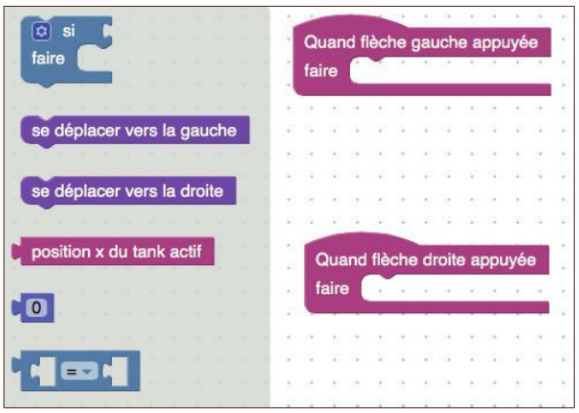
Chapitre 1

Apprendre à faire bouger un tank

Notions abordées

- L'animation dans un jeu vidéo
- Les tests
- Les conditions





Tout reste à faire...

Ce premier exercice va te permettre de te familiariser avec ton environnement de travail en réalisant les premières briques de ton jeu vidéo.

Bienvenue dans ton « Cahier d'activités Python ». Dans ce premier chapitre, tu découvriras ce qu'on appelle un « environnement de développement », et tu commenceras à créer ton jeu vidéo.

Création de ton compte

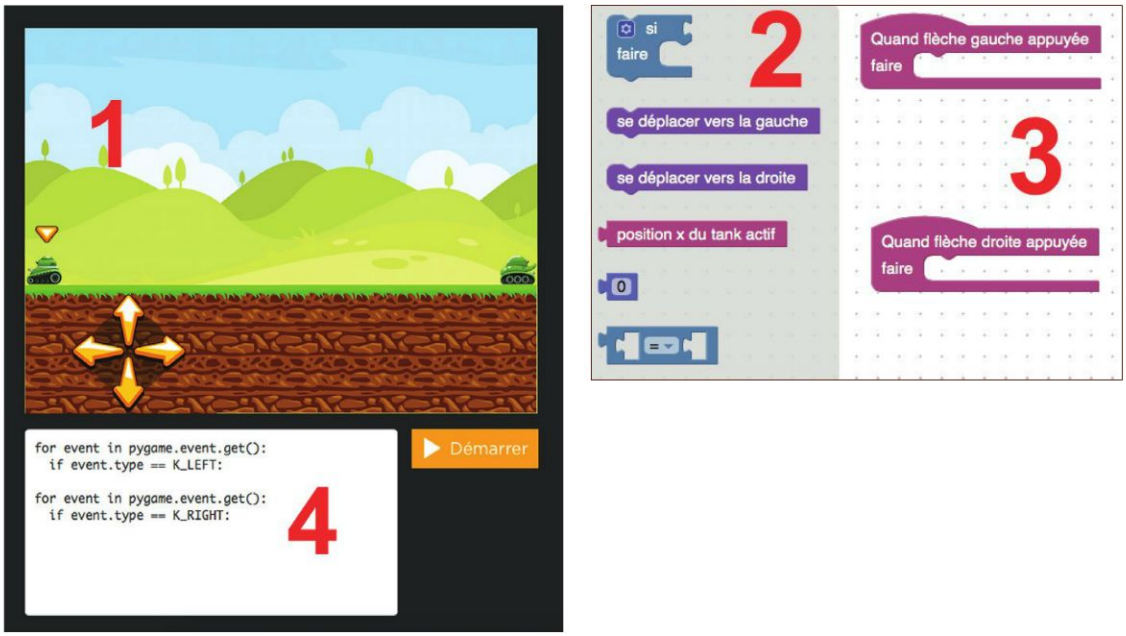
Tout d'abord, rends-toi à l'adresse http://www.kidscod.in/app/tank. Crée-toi un compte en te faisant aider de tes parents si nécessaire. Ce compte te servira pour accéder au contenu interactif qui accompagne ce cahier d'activités. De plus, il te permet de sauvegarder ta progression et de partager ton jeu vidéo avec le monde entier.

Tu vas recevoir un e-mail contenant un lien de validation sur lequel il faudra cliquer pour valider ton inscription. Ensuite, il te suffira de retourner à l'adresse http://www.kidscod.in/app/tank et de te connecter.

Environnement de développement

Une fois authentifié, tu arriveras sur ce qu'on appelle « l'environnement de développement » dans le monde des développeurs. C'est dans cet environnement que tu vas créer, tester et corriger ton jeu vidéo. Tu peux déjà remarquer les chiffres 1 à 6 tout en haut : ils correspondent aux six chapitres de ce livre. Je te conseille de suivre la progression proposée, même si tu es libre de faire les exercices dans l'ordre que tu veux.

Passons maintenant à la découverte de ton environnement de développement. Il est composé de quatre zones principales.



Ton environnement de développement

- La **scène** 1 correspond au jeu vidéo que tu crées.
- Les **blocs de code** utilisables sont **disponibles** dans la zone 2.
- C'est dans l'**espace de travail** 3 que tu vas assembler les blocs de code.
- Le **code généré** est automatiquement mis à jour dans la zone 4 en fonction des briques de code présentes dans l'espace de travail 3.

Tu peux déjà t'entraîner à faire les manipulations que tu vas répéter tout au long de ce cahier : commence par prendre un bloc de code de la zone 2, puis dépose-le dans la zone 3. Tu peux voir que la zone 4 sera automatiquement mise à jour en fonction du bloc que tu auras déposé. Enfin, en cliquant sur le bouton ▶ Démarrer (**Démarrer**), tu pourras tester ton programme qui s'exécutera dans la zone 1. Ce n'est pas plus compliqué que ça !

Bien évidemment, quand tu joues à un jeu vidéo, tu ne vois que la zone 1, la scène. Les zones 2, 3 et 4 ne sont accessibles qu'au créateur du jeu vidéo. Nous verrons par la suite comment tu pourras partager ton jeu avec tes amis ou ta famille, en ne leur montrant que la zone 1. Hors de question qu'ils aient accès à ton espace de travail : un magicien ne révèle jamais ses secrets !

Et le langage Python dans tout ça ?

Sans doute te demandes-tu pourquoi on va manipuler des blocs de code, semblables à des briques de Lego ?

C'est très simple : il est beaucoup plus facile d'apprendre à programmer de façon visuelle, avec des blocs de code à imbriquer les uns dans les autres, plutôt que d'écrire des lignes de code. En revanche, une fois que tu auras bien compris comment créer un jeu vidéo, tu pourras apprendre à programmer à l'aide d'un langage de programmation textuel, comme Python. On pense souvent à Python pour apprendre un langage de programmation car il a justement été pensé pour les débutants. Tout le texte que tu verras s'écrire tout seul dans la zone 4 sera la traduction en langage Python de ce que tu créeras de façon visuelle dans la zone de travail 3. Mais je sens bien que tu es impatient de commencer, alors c'est parti !

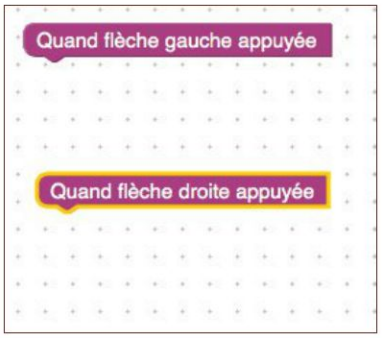
Pourquoi mon tank ne bouge-t-il pas ?

Dans un jeu vidéo, tu as l'habitude de faire bouger ton personnage (ou des objets) à l'aide d'un joypad, d'un clavier ou encore en inclinant ta tablette. Mais il n'y a rien de magique derrière tout ça : ce sont uniquement des lignes de code !

Dans ton jeu de tank, tu peux voir qu'il y a un pavé directionnel dans la zone 1, juste en dessous du tank n°1. **Le but de ce premier chapitre est de faire en sorte que si on appuie sur la flèche pointant vers la gauche, le tank se dirige vers la gauche, et inversement, que si on appuie sur la flèche pointant vers la droite, le tank avance vers la droite.**

Si tu testes maintenant, il ne se passera rien. Même si tu cliques sur **Démarrer**, cela ne change rien, impossible de faire bouger le tank. C'est tout à fait normal, et nous allons voir pourquoi.

Si tu observes l'espace de travail, tu verras deux blocs de code isolés : `Quand flèche gauche appuyée` et `Quand flèche droite appuyée`.



Blocs de code concernant les flèches gauche et droite

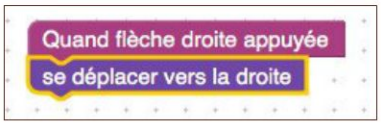
Comme il n'y a rien en dessous de ces blocs, ton programme ne sait pas ce qu'il faut faire quand on appuie sur les flèches **Gauche** ou **Droite** : donc il ne fait rien. Ce qui est pour toi évident (faire avancer un personnage vers la droite si on appuie sur une flèche allant vers la droite) n'a absolument aucun sens pour un ordinateur, qui n'a pas l'habitude de jouer à des jeux vidéo.

Il faut donc expliquer en détail tout ce qui caractérise ton jeu vidéo : c'est ce qu'on appelle « écrire un programme », et nous allons commencer dès maintenant.

Ton premier morceau de code

N'attendons plus et jetons-nous dans le feu de l'action ! Dans un jeu vidéo, s'il n'y a pas d'obstacles aux alentours du personnage principal, le joueur s'attend à pouvoir le déplacer dans toutes les directions permises par le jeu. Ici, nous pouvons uniquement déplacer un tank vers la droite ou vers la gauche.

Occupons-nous d'abord du déplacement vers la droite : il suffit de prendre le bloc `se déplacer vers la droite` dans la zone 2 puis de le déposer dans la zone 3, sous le bloc `Quand flèche droite appuyée`. Attention, il faut que les deux blocs soient accrochés l'un à l'autre : aide-toi des encoches présentes sur les blocs. Quand les deux encoches sont suffisamment proches, tu peux lâcher ton bloc : il ira se coller de lui-même au bon endroit. Si tu déposes ton bloc trop loin, il restera isolé et ne sera pas compris par ton jeu vidéo. Tu devrais obtenir ceci :



Deux blocs correctement positionnés

Tu as créé ton premier morceau de code ! Il est temps de le tester en cliquant sur **Démarrer**. Essaye ensuite de cliquer sur la flèche pointant vers la droite : youpi, ton tank avance vers la droite ! Ne t'arrête pas en si bon chemin, et procède de la même façon pour pouvoir déplacer ton tank vers la gauche.

Attention
Tes modifications ne sont pas prises en compte par ton jeu vidéo tant que tu ne cliques pas sur **Réinitialiser**, puis sur **Démarrer**.

Des murs invisibles

Si tu testes un peu ton jeu, tu remarqueras qu'il y a un petit problème : ton tank peut sortir de la scène, par la gauche ou par la droite. Nous allons devoir l'empêcher de sortir du cadre. Pour cela, nous inventerons des « murs virtuels » que le tank n'aura pas le droit de franchir.

Quand on crée un jeu vidéo, on repère généralement la position des sprites (tout ce qui bouge à l'écran) grâce à une grille, comme dans une bataille navale. Comme nous nous intéressons (pour l'instant) uniquement aux déplacements horizontaux, le long du sol, nous ne considérerons que la position horizontale du tank. On a l'habitude de parler de l'axe x pour désigner les graduations qui nous permettent

de repérer horizontalement des objets. Dans ton jeu, l'axe des x commence à la valeur 0, tout à gauche de l'écran, et finit à 550, tout à droite de l'écran.

Ces chiffres n'ont pas été choisis au hasard : il y a exactement 550 pixels entre les deux extrémités de l'image. Or, les pixels sont les points les plus petits que ton écran peut afficher.

Axe des x

On peut donc dire que quelle que soit la position du tank à l'écran, sa position « x » sera supérieure (ou égale) à 0 et forcément inférieure (ou égale) à 550. Je suis sûr que tu as déjà compris que pour empêcher le tank de sortir de l'écran, il suffit de s'assurer que sa position x reste entre 0 et 550. Avant chaque déplacement, nous vérifierons donc qu'elle est comprise entre 0 et 550, pour empêcher le tank de bouger au-delà, même si le joueur appuie sur une des deux flèches.

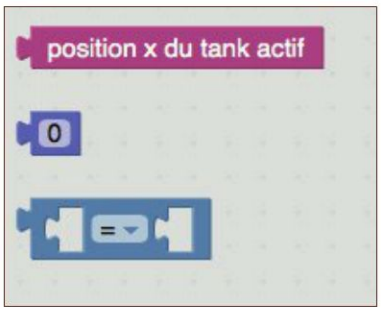
Allons-y !

Intéressons-nous d'abord au déplacement vers la gauche qui peut se résumer par : « **si** la position x du tank est plus grande que 0, **alors** il peut se déplacer vers la gauche ». Voyons comment traduire cela dans notre espace de travail.

Dans la phrase précédente, deux mots sont très importants : « si » et « alors ». On peut les matérialiser grâce au bloc de code `si… faire` que tu trouveras dans la liste des blocs disponibles.

Bloc « si ... alors faire ... »

Il suffit de placer ce bloc directement sous le bloc **Quand flèche gauche appuyée**. Il va prendre la place du bloc **se déplacer vers la gauche**, qui va se détacher. Rattache-le directement dans le bloc **si**, juste en face du verbe **faire**. Nous avons un nouveau bloc, mais tu as certainement remarqué qu'il n'est pas complet : il n'y a rien à côté du **si**, à part une encoche vide. C'est tout simplement parce que nous n'avons pas formulé la condition, à savoir « si la position x du tank est plus grande que 0 ». Pour matérialiser cette condition, il faut se servir de trois blocs : le comparateur, la position x du tank, et le bloc « nombre » .



Les trois blocs à utiliser pour ta première condition

Il est important que tu comprennes la nature du bloc **position x du tank actif** : c'est ce qu'on appelle une « variable » dans le monde des programmeurs. Une variable peut être assimilée à une boîte contenant une information : quand on y fait appel, c'est l'information qu'elle contient qui nous intéresse. Dans notre jeu de tank, la variable **position x du tank actif** contiendra toujours la position du tank actif selon l'axe horizontal. Dès que le tank se déplace, elle est automatiquement mise à jour avec la nouvelle valeur de la position du tank.

Je te laisse déposer ces trois blocs dans l'espace de travail (n'importe où pour l'instant) afin de créer la condition « position x du tank supérieure ou égale à 0 ». Pour cela, n'oublie pas de rechercher le symbole **≥** dans la liste déroulante du bloc de comparaison (qui par défaut vaut **=**). Une fois que tu as pris en compte cette condition, il te suffit de la « clipser » à droite du bloc **si…faire**, dans l'encoche prévue pour cela. Voici à quoi ton code devrait ressembler :

Comment définir un mur virtuel

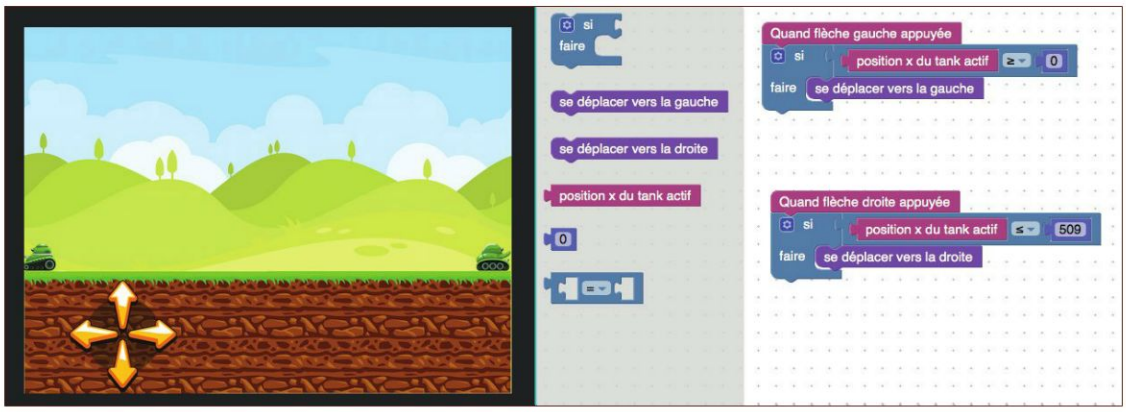
Génial, notre tank ne peut plus sortir de l'écran par la gauche !

As-tu bien compris pourquoi ? C'est tout simplement parce que si la condition n'est pas réalisée (si le tank est à la position `x = 0`), alors le programme ne fait rien : donc même si le joueur appuie sur la flèche **Gauche**, il ne se passera rien. En revanche, si la condition est réalisée (par exemple si le tank se trouve à la position `x = 100`), alors le programme a une instruction à réaliser, à savoir « se déplacer vers la gauche ».

Je te laisse procéder de la même manière pour l'empêcher de sortir par la droite. Attention, la nouvelle condition est « si la position x du tank est **inférieure ou égale** à **550** ».

Note

Dans la correction que nous avons utilisée, la valeur est de 509 au lieu de 550 : c'est tout simplement parce que le tank a une longueur égale à 41 pixels.



Fin du 1er chapitre

Félicitations !

Tu as terminé le premier chapitre ! Tu as appris à :

- déclencher une action suite à l'appui d'une touche ;
- tester la position du sprite pour vérifier s'il a « le droit » de se déplacer ou pas ;
- effectuer un test conditionnel.

Le code Python généré dans la zone 4 devrait ressembler à ça :

```
for event in pygame.event.get():1
      if event.type == K_LEFT:2
            if getActiveTankX() >= 0:3
                  avancerGauche()4

for event in pygame.event.get():
      if event.type == K_RIGHT:
            if getActiveTankX() <= 509:
                  avancerDroite()
```

Tu peux retrouver exactement ce que tu as fait de façon graphique :

1 : ton programme Python attend qu'il se passe quelque chose ;

2 : si l'utilisateur appuie sur la touche flèche Gauche ;

3 : il faut vérifier si le tank peut bouger vers la gauche (tu reconnaîtras la même condition) ;

4 : si la condition est vérifiée, alors on peut avancer vers la gauche : on donne l'instruction Python qui fera déplacer le tank vers la gauche, à savoir `avancerGauche()`.

Et la suite ?

Nous avons donc créé quelque chose d'extraordinaire : un tank qui bouge ! Mais pour que le jeu soit réellement intéressant, il va falloir lui apprendre d'autres choses, comme tirer un obus par exemple... Rendez-vous au chapitre 2 pour cette nouvelle étape !

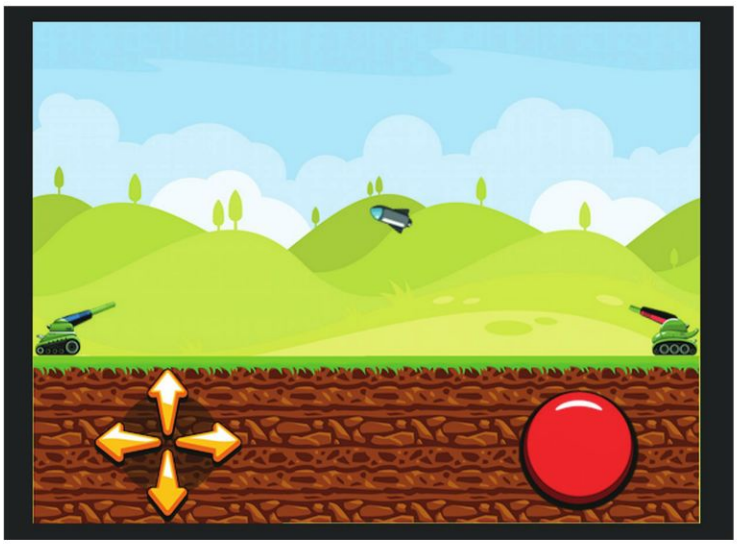
Psssst
As-tu essayé de cliquer sur le deuxième tank ?

Chapitre 2

Comment tirer un obus ?

Notions abordées

- Affectation de valeurs
- Apparition/disparition de sprites



Attention au bouton rouge...

Dans ce deuxième chapitre, tu vas apprendre à faire apparaître un obus, puis à le faire avancer en ligne droite.

Maintenant que tu es familiarisé avec l'environnement de développement, nous allons mettre en pratique ce que tu as appris au premier chapitre, en ajoutant des canons aux tanks. Nous allons également voir comment faire en sorte que, dans ton jeu, les tanks puissent tirer des obus.

Quoi de neuf ?

Dans cet exercice, tu peux remarquer quelques différences par rapport au chapitre précédent :

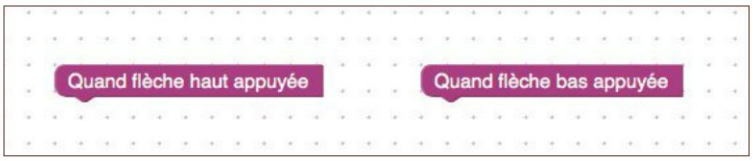
- les tanks ont dorénavant des canons (c'est plus pratique pour tirer un obus) ;
- un nouveau bouton rouge a été ajouté sur l'interface, à droite du pavé directionnel ;
- il y a 5 blocs de code sur l'espace de travail (au lieu de 3 avant) ;
- les blocs de code disponibles sont classés par catégories : **Contrôles**, **Actions**, **Affectation**, **Variables**, **Maths**, **Outils** et **Fonctions**.

Ce que nous allons faire dans ce chapitre

Tout d'abord, nous allons voir comment permettre au joueur d'orienter les canons des tanks. Ensuite, tu apprendras à faire tirer un obus à un tank. Mais commençons par nous intéresser aux canons !

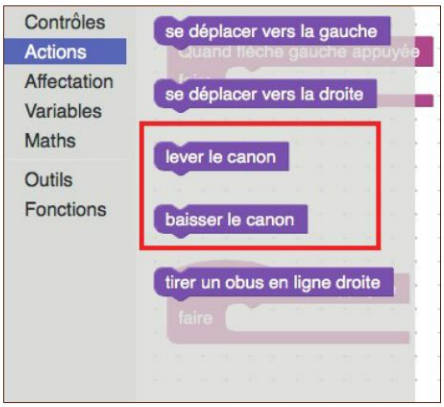
Vers le haut et vers le bas

Parmi les nouveaux blocs de code de l'espace de travail, tu peux remarquer que deux d'entre eux concernent les flèches **Haut** et **Bas**.



Blocs de code pour la gestion des flèches Haut et Bas

Tout comme nous avons vu comment faire se déplacer un tank grâce aux flèches **Gauche** et **Droite**, nous allons faire en sorte que lorsque l'utilisateur clique sur la flèche pointant vers le haut, le canon du tank actif s'oriente un peu plus vers le haut. Inversement, il faut que le canon s'oriente vers le bas lorsque c'est la flèche pointant vers le bas qui est appuyée. Pour cela, tu as à ta disposition deux nouvelles actions.



Les deux nouvelles actions disponibles

Code ces nouvelles interactions avec l'utilisateur en t'inspirant de ce que tu as appris au chapitre 1 : je suis certain que cela ne te posera aucun problème !

Te rappelles-tu que pour les déplacements horizontaux du tank, nous avons fait en sorte que le tank ne puisse pas sortir de l'écran ? Nous allons imposer des limites similaires pour les canons : il ne faut pas qu'un canon puisse descendre plus bas que la position horizontale déterminée, ni aller au-delà de la position verticale définie. Voici les deux positions maximales, pour que tu visualises ce qu'il faut coder :

Position la plus basse du canon

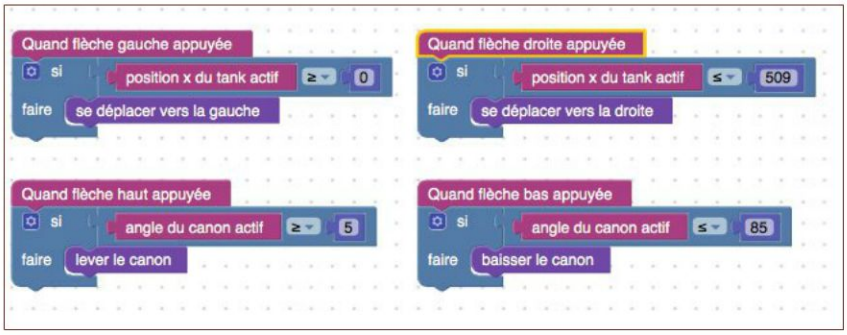
Position la plus haute du canon

Tout comme on a vérifié la position du tank grâce à la variable `position x du tank actif`, nous allons repérer l'inclinaison du canon grâce à la variable `angle du canon actif`. Cette dernière variable devra être comprise entre 5° (position presque verticale) et 85° (position presque horizontale). N'hésite pas à essayer plusieurs solutions et à toujours tester ton code en cliquant sur le bouton **Démarrer**.

Note

Les blocs dont tu auras besoin se trouvent respectivement dans les sous-menus **Contrôles**, **Variables** et **Actions**.

Voici la solution de cette partie du chapitre :



Code complet gérant les mouvements du tank

Les obus ! Les obus !

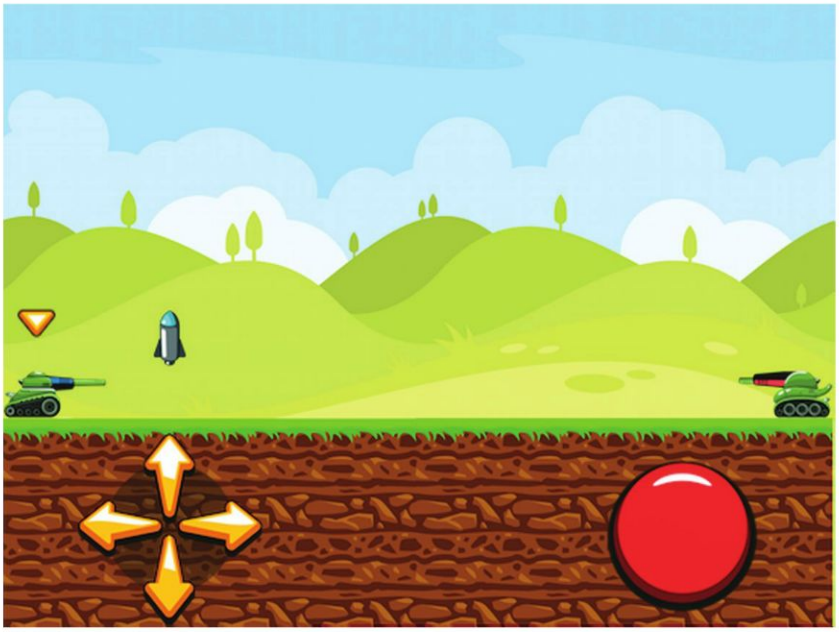
Il est temps que ton jeu permette de tirer des obus. Dans les jeux vidéo, une des astuces courantes consiste à dessiner tous les sprites du jeu lors de son lancement, même ceux qui ne sont pas affichés dès le début. Ces derniers sont simplement dessinés en dehors de la zone affichée à l'écran. Ainsi, lorsqu'on a besoin d'afficher un sprite qui était caché jusqu'ici, il suffit de changer ses coordonnées x et y pour qu'il apparaisse à l'écran.

Voici un exemple concret pour bien comprendre : au lancement du jeu, une image d'un obus a été chargée. Elle est affichée en dehors de l'écran pour qu'elle ne soit pas visible, mais elle est bien là ! Si tu veux la voir, il suffit de modifier sa position sur l'axe des x. En fixant la coordonnée x de l'obus à une valeur comprise entre 0 et 550 (les bornes définies au chapitre 1), on verra apparaître l'obus comme par magie !

Pour y arriver, nous allons nous servir d'un nouveau bloc qui se situe dans le sous-menu **Affectation**. Les blocs de type **Affectation** permettent de modifier les valeurs qui sont stockées dans les variables. Nous allons donc utiliser le bloc `Initialiser la position x de l'obus à`.

Affectation de la position x d'un obus

Positionne ce bloc sous le bloc `Quand bouton feu appuyé` de l'espace de travail. Ainsi tu pourras tester ton code en cliquant sur le bouton rouge (enfin !). Je te conseille d'initialiser la valeur x de l'obus à 100 mais tu peux essayer toutes les valeurs que tu souhaites comprises entre 0 et 550, et même au-delà : si l'obus n'est plus visible, c'est simplement parce que sa position x n'est pas comprise dans l'intervalle affiché à l'écran. Voici ce que donne le code exécuté pour une valeur de x affectée à 100 :



Un obus sorti de nulle part...

Donc nous avons un obus immobile, en plein milieu de l'écran...

Mais nous n'avons pas encore atteint notre but ! Pour rappel, nous voulons que l'obus parte du canon du tank, puis avance en ligne droite. Il nous reste donc deux problèmes à résoudre : faire bouger l'obus et le faire partir de l'extrémité du canon.

Jetons un coup d'œil aux blocs à notre disposition dans le menu **Variables** : il y en a un qui peut contribuer à résoudre notre problème, il s'agit de la variable `position x du canon actif`. En effet, si nous faisons en sorte que la position x de l'obus soit la même que la position x du canon, alors l'obus semblera partir du canon !

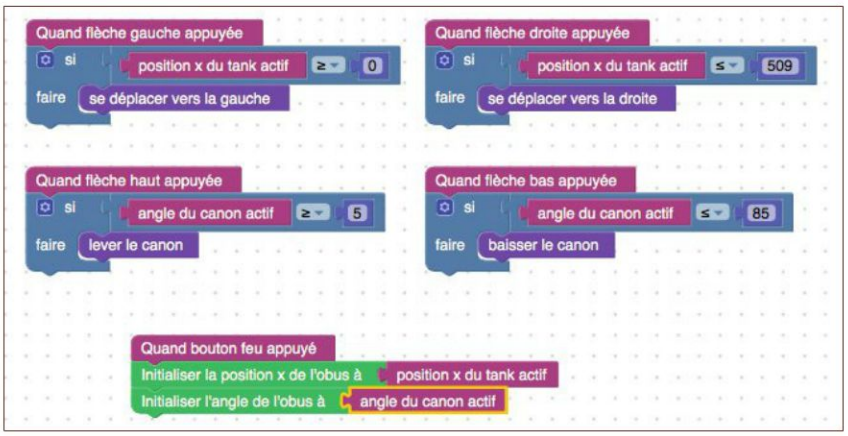
Essaye pour voir ce que ça donne : il te suffit choisir la variable `position x du tank actif` comme entrée de la fonction `Initialiser la position x de l'obus à`. Tu peux t'amuser à tester en bougeant les deux tanks : à chaque fois que tu appuieras sur le bouton rouge, l'obus devrait se positionner près du canon du tank

actif, quelle que soit la position du tank. Puis, pour que l'obus semble sortir du canon, il faut l'aligner avec l'axe du canon. Sinon, l'obus pointe vers le ciel quelle que soit l'orientation du canon, ce qui n'est pas très réaliste !

Je suis sûr que tu auras trouvé qu'il faut utiliser pour cela la fonction `Initialiser l'angle de l'obus à` avec en entrée la variable `angle du canon actif`. Essaye pour voir, et n'oublie pas de tester avec plusieurs positions des canons pour vérifier que ton code est correct.

Attention
Il est toujours plus facile de tester et corriger à chaque petite modification que de tester tout à la fin et de ne pas savoir d'où peut provenir ton erreur...

Voici à quoi devrait ressembler ton code à cet instant :

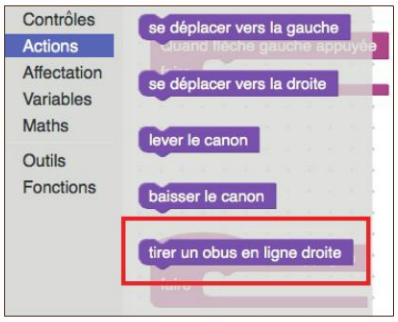


Solution intermédiaire concernant le tir d'un obus

Feu !

Il nous reste à résoudre un dernier problème : notre obus ne bouge pas ! Si tu regardes du côté du sous-menu **Actions**, tu devrais trouver le bloc qui va nous aider : `tirer un obus en ligne droite`.

Si tu positionnes ce nouveau bloc juste en dessous des deux blocs que tu as déjà placé en dessous de `Quand bouton feu appuyé`, que penses-tu qu'il va arriver ? Le mieux est encore d'essayer !



Comment tirer un obus en ligne droite

Victoire !

Je suis certain que tu as été ravi de découvrir le résultat ! Tu as réussi à coder un déplacement d'obus avec les caractéristiques suivantes :

- l'obus part du canon (grâce à l'affectation de la `position x de l'obus` à la `position x du canon actif`) ;
- il se dirige selon la direction du canon (grâce à l'affectation de la valeur `angle de l'obus` à la valeur `angle du canon`) ;
- il avance en ligne droite (grâce à l'action `tirer un obus en ligne droite`).

Tout au long de ce chapitre, tu as pu découvrir les notions de : variable, affectation de variable, appel de fonction.

Il est temps de jeter un œil au code Python généré pour bien comprendre comment tout ce que tu as fait aurait pu être codé de façon textuelle :

```
for event in pygame.event.get():
    if event.type == K_UP:1
        if getActiveBarrelsRotation() >= 5:2
            leverCanon()3

for event in pygame.event.get():
    if event.type == K_DOWN:4
        if getActiveBarrelsRotation() <= 85:5
            baisserCanon()6
```

```
for event in pygame.event.get():
      if event.type == K_SPACE:7
         setWeapon1X((getActiveTankX()))8
         setWeapon1Rotation((getActiveBarrelsRotation()))9
         fireWeapon()10
```

Analysons ce qui a été généré :

1 : si l'utilisateur appuie sur la touche flèche Haut ;

2 : il faut vérifier si le canon peut bouger vers le haut (tu reconnaîtras la condition introduite par if) ;

3 : si la condition est vérifiée, alors on peut lever le canon : on appelle l'instruction Python qui fera se lever un peu le canon, à savoir `leverCanon()` ;

4 : si l'utilisateur appuie sur la touche flèche Bas ;

5 : il faut vérifier si le canon peut bouger vers le bas (tu reconnaîtras la condition introduite par if) ;

6 : si la condition est vérifiée, alors on peut baisser le canon : on appelle l'instruction Python qui fera se baisser un peu le canon, à savoir `baisserCanon()` ;

7 : si l'utilisateur appuie sur la touche Espace (équivalent du bouton rouge) ;

8 : on utilise la fonction `setWeapon1X()` avec en entrée la position x du tank, obtenue grâce à la fonction `getActiveTankX()` ;

9 : on utilise la fonction `setWeapon1Rotation()` avec en entrée l'angle du canon, obtenue grâce à la fonction `getActiveBarrelsRotation()` ;

10 : BOUM ! On appelle la fonction `fireWeapon()` qui fait bouger un obus en ligne droite.

Et la suite ?

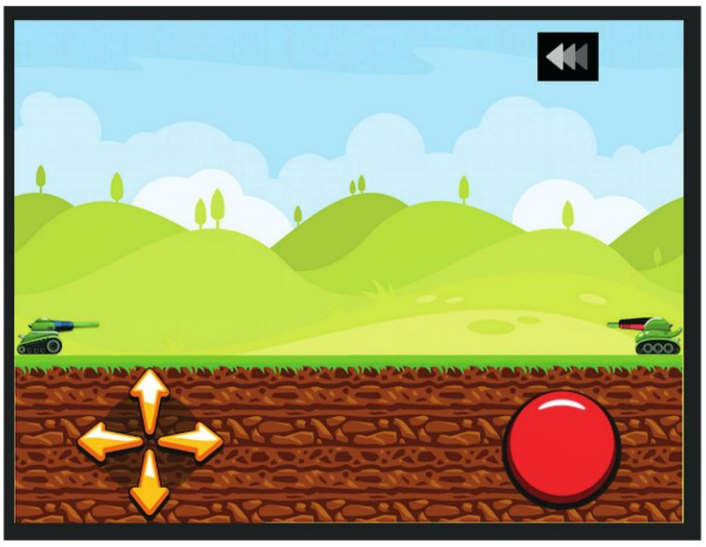
Pour que ton jeu vidéo devienne un peu plus amusant, nous allons introduire la notion de gravité : en effet, un obus qui avance en ligne droite, sans jamais retomber, ce n'est pas très réaliste. Pour que notre obus soit attiré par la Terre, nous allons intégrer un « moteur physique de gravité » dans ton jeu vidéo : c'est l'objet du chapitre 3 !

Chapitre 3

Simuler les lois de la physique

Notions abordées

- Création d'un moteur physique
- Influence de la gravité et des forces externes (comme le vent) dans un moteur physique



Quand l'environnement s'en mêle

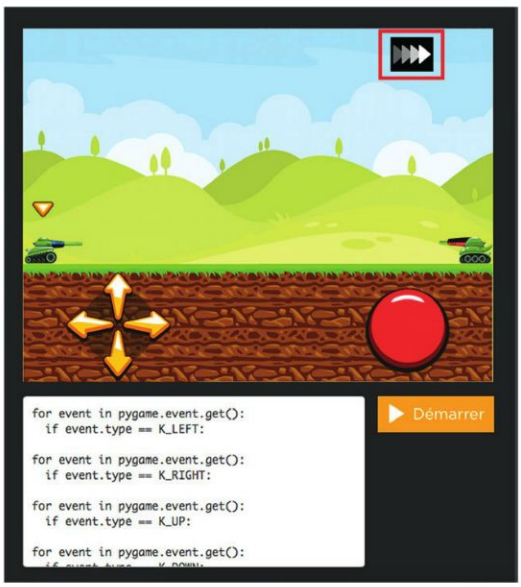
Dans ce chapitre, tu vas apprendre à simuler les lois physiques du monde qui nous entoure dans un jeu vidéo.

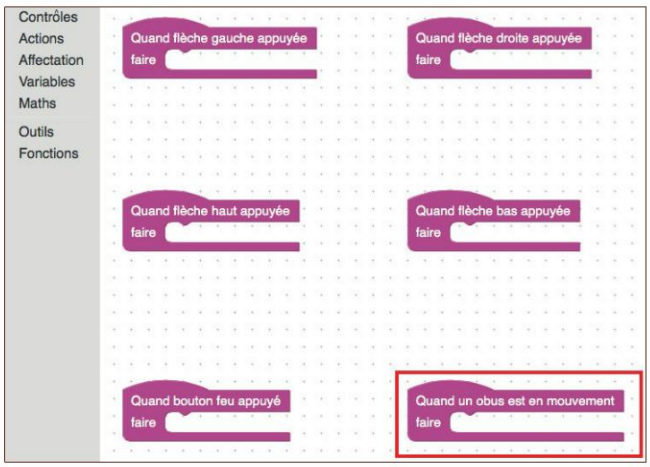
Dans le chapitre 2, nous avons vu comment tirer un obus en ligne droite. Cette solution n'est pas très satisfaisante pour ton jeu car ce n'est ni très amusant, ni conforme à ce qu'on peut observer au quotidien.

Quoi de neuf ?

Deux nouveautés sont visibles à l'écran :

- une nouvelle icône représentant la force du vent ;
- un nouveau bloc dans l'espace de travail.





Nouveaux blocs

L'icône représentant la force du vent peut prendre différentes formes qui sont récapitulées dans le tableau suivant.

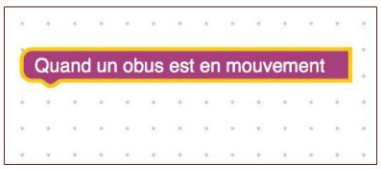
Différents états du vent

Image	Force du vent	Image	Force du vent
	Vent très fort soufflant vers la droite		Vent fort soufflant vers la gauche
	Vent fort soufflant vers la droite		Vent modéré soufflant vers la gauche
	Vent modéré soufflant vers la droite		Vent faible soufflant vers la gauche
	Vent faible soufflant vers la droite		Absence de vent
	Vent très fort soufflant vers la gauche		

La force du vent est déterminée au hasard au chargement de l'exercice 3. Elle ne peut pas être changée et reste constante tant que la page n'est pas rechargée.

Note
Nous verrons au chapitre 5 comment tu pourras générer une force de vent aléatoire par toi-même.

Le bloc `Quand un obus est en mouvement` sert à modifier le comportement d'un obus lorsqu'il est en mouvement.



Bloc pour agir pendant le déplacement d'un obus

Ce que nous allons faire dans ce chapitre

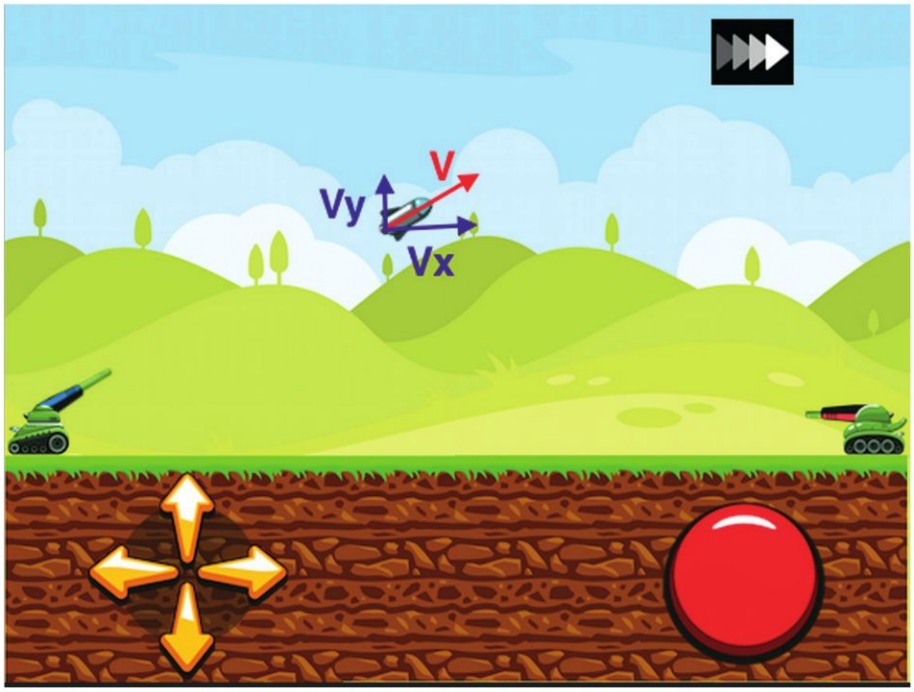
Le but de ce chapitre est de te faire comprendre comment il est possible de simuler les lois de la physique classique dans ton jeu vidéo. Pour cela, nous allons coder deux nouvelles caractéristiques dans ton jeu :

- la force du vent doit accélérer ou ralentir l'obus (cela dépend du sens du vent par rapport au déplacement de l'obus) ;
- la gravité doit attirer l'obus vers le bas (la gravité est constante, elle ne dépend pas du vent : l'effet sera donc toujours le même, à savoir attirer l'obus vers le bas, de plus en plus vite).

Déclenchons une tempête !

Avant de déclencher une tempête, intéressons-nous à la vitesse d'un obus en mouvement. Sur le schéma qui suit, tu peux voir qu'on a l'habitude de décomposer la vitesse d'un objet en mouvement en deux vitesses :

- une vitesse horizontale, qu'on nomme Vx ;
- une vitesse verticale, qu'on nomme Vy.



Décomposition des vitesses

Le mouvement de l'obus va directement dépendre des valeurs des vitesses horizontales et verticales. Par exemple, si la vitesse horizontale de l'obus est nulle, cela veut dire que l'obus ne se déplacera pas horizontalement. Il va donc se déplacer uniquement verticalement, en ligne droite. Au contraire, si la vitesse verticale de l'obus est nulle, cela signifie que l'obus ne va se déplacer qu'horizontalement, en ligne droite.

Si les vitesses horizontales et verticales ne sont pas nulles, alors l'obus se déplace horizontalement et verticalement, comme dans l'exemple ci-dessus.

Influence du vent sur la vitesse

Dans notre jeu vidéo, nous allons décider que le vent souffle uniquement horizontalement. Ce n'est donc pas une tempête que nous allons simuler, mais ce sera plus facile ainsi !

Le vent ne soufflant qu'horizontalement, il n'aura donc aucune influence sur la vitesse verticale Vy de l'obus. En revanche, si le vent souffle dans le même sens que la vitesse horizontale Vx, alors l'obus sera accéléré par le vent : le vent va augmenter la vitesse Vx de l'obus. Mais si le vent souffle dans le sens inverse par rapport à la vitesse Vx, alors l'obus sera ralenti par le vent : la vitesse Vx de l'obus sera diminuée par le vent.

Pour simuler l'influence du vent sur la vitesse de l'obus, nous allons simplement additionner la vitesse horizontale Vx de l'obus avec la vitesse du vent.

Note

Tu peux facilement vérifier qu'additionner les deux vitesses est la bonne solution pour simuler l'influence du vent : dans le cas où le vent souffle dans la direction opposée à celle de l'obus, les vitesses du vent et de l'obus n'ont pas le même signe, et donc additionner ces deux vitesses revient à ôter la force du vent à la vitesse de l'obus. Inversement, quand le vent souffle dans la direction de propagation de l'obus, les deux vitesses ont le même signe : en les additionnant, on ajoute bien la vitesse du vent à celle de l'obus. Et voilà !

Assez de physique, place au code !

Nous allons pour cela utiliser les blocs de code `Initialiser la vitesse horizontale de l'obus à`, `vitesse du vent` et `+`. Contrairement au chapitre 2 où on fixait les variables de l'obus au *départ*, ici nous nous intéressons à l'influence du vent *pendant* le déplacement de l'obus. Il faut donc se servir du bloc de code `Quand un obus est en mouvement` de l'espace de travail : tous les sous-blocs rattachés à ce bloc seront exécutés en boucle pendant toute la durée du mouvement.

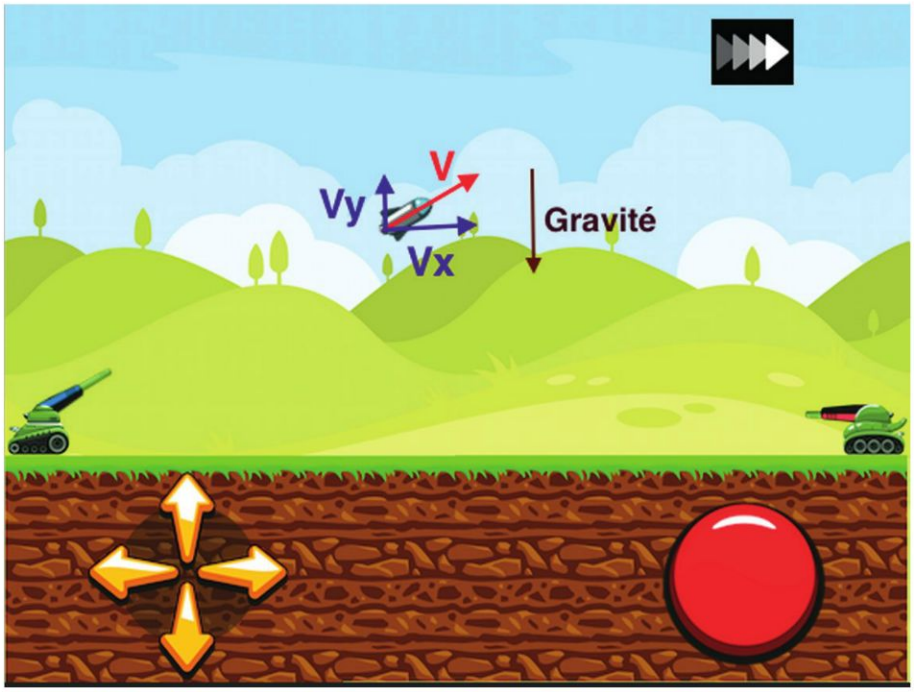
Place ce bout de code et teste-le dans différentes situations. Si tout va bien, tu devrais constater que l'obus est accéléré ou ralenti, selon la direction du vent et de l'obus. Il arrivera même que l'obus fasse demi-tour à cause du vent trop fort !

Note

Nous avons choisi ici de nous servir de la variable contenant la valeur de la vitesse du vent. Libre à toi d'expérimenter d'autres valeurs, en utilisant par exemple le bloc de code te permettant de choisir n'importe quel nombre. Pourquoi ne pas tester avec un vent très fort ?

Les obus tombent comme des pommes

Newton a eu de la chance de découvrir la loi de la gravité quand une pomme lui est tombée sur la tête. Il aurait tout aussi bien pu recevoir un obus, car un obus obéit à la même loi physique de la gravité : tout corps suffisamment proche de la Terre est attiré par le centre de la Terre.



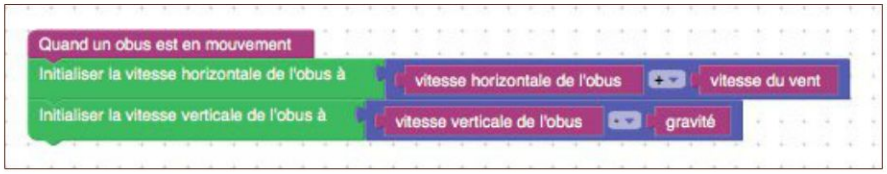
Influence de la gravité

Si tu fais le rapprochement avec la vitesse du vent, tu verras qu'ici la gravité a toujours la même valeur et sera toujours dirigée vers le bas. La gravité n'a donc aucune influence sur la vitesse horizontale de l'obus : elle n'influera que la vitesse verticale Vy.

Comme la gravité est orientée vers le bas, nous allons devoir ôter sa valeur de la vitesse Vy, tout au long du trajet de l'obus. Ainsi, un obus qui est en train de monter aura tendance à ralentir son ascension, s'arrêter puis à descendre en flèche vers le bas !

À toi de jouer avec la gravité

Tu commences à avoir l'habitude, mais au cas où, voici les blocs de code à utiliser pour créer notre modèle physique de la gravité : `Initialiser la vitesse verticale de l'obus à`, `gravité` et `-`. Finalement, ton moteur physique devrait ressembler à ça :



Code final du moteur physique

Vive la physique !

Toutes mes félicitations pour ce joli moteur physique : ton jeu vidéo est maintenant capable de simuler l'effet du vent, tout comme l'influence de la gravité. Ainsi, les obus ont une trajectoire un peu plus réaliste qu'une simple ligne droite ! Et pour cela, tu n'as pas eu besoin de faire des choses très compliquées : la simple mise en œuvre du concept d'affectation de variables a suffi...

Regardons un peu le code Python que tu as généré :

```
for event in pygame.event.get():
    if event.type == "ObusEnMouvement":(1)
        setWeapon1VX((getWeapon1VX() + getWindVX()))(2)
        setWeapon1VY((getWeapon1VY() - getGravity()))(3)
```

Si tu as du mal à faire le rapprochement avec ton code graphique, voici de quoi t'aider :

1. : le code suivant s'applique pendant tout le temps où l'événement `ObusEnMouvement` est reçu, soit pendant tout le temps où un obus se déplace à l'écran ;
2. : on modifie la valeur de la vitesse Vx de l'obus en lui ajoutant la valeur de la vitesse du vent ;
3. : on modifie la valeur Vy de l'obus en lui ôtant la valeur de la gravité.

Promenons-nous dans le système solaire

L'avantage de coder soi-même son jeu vidéo et son propre moteur physique, c'est qu'on peut décider si le jeu vidéo se passe sur Terre ou ailleurs dans l'espace : pour cela, ajuste la valeur de la gravité pour observer comment se comporteraient les obus sur d'autres planètes.

Par exemple, la gravité sur Mars vaut le tiers de la gravité sur Terre. Il te suffit donc de remplacer le bloc `gravité` par l'expression suivante :

Sur Mars, on a l'impression d'être trois fois plus léger que sur Terre.

À toi maintenant d'explorer d'autres valeurs pour la gravité, en t'inspirant du tableau suivant.

Ordre de grandeur des gravités de plusieurs astres rapportées à celle de la Terre

Planète	Ordre de grandeur de la gravité
Lune	égale à 0,1 fois celle de la Terre
Mercure	égale à 0,4 fois celle de la Terre
Vénus	égale à 0,9 fois celle de la Terre
Jupiter	égale à 2,5 fois celle de la Terre

Note

Il est techniquement **impossible** de poser un tank sur Jupiter car c'est une **planète gazeuse**. Mais sa gravité très élevée justifie à elle seule qu'on s'y intéresse !

La suite ! La suite !

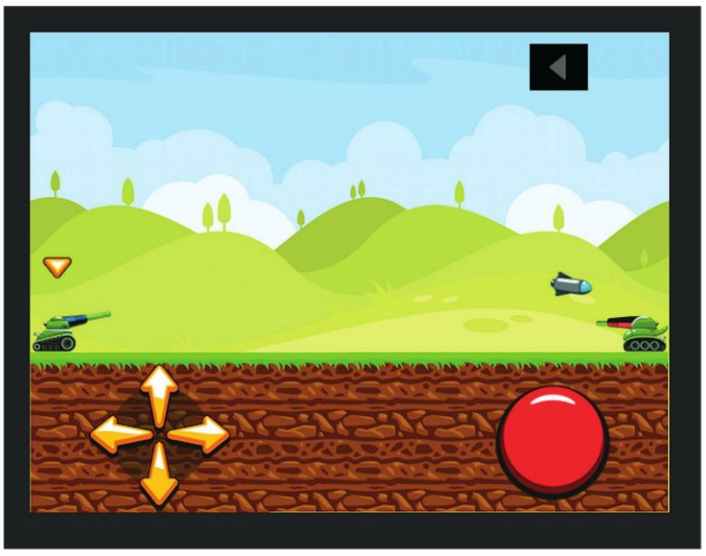
Tu as déjà appris beaucoup de choses sur la création d'un jeu vidéo. Mais il reste encore quelques secrets à découvrir : par exemple, tu as dû remarquer que les obus tirés passent au travers des obstacles, ce qui n'est encore une fois pas très réaliste. Rendez-vous au chapitre 4 pour apprendre à coder la détection de collisions.

Chapitre 4

Collisions et explosions

Notions abordées

- Détection de collisions
- Faire apparaître un sprite d'explosion
- Création et manipulation de fonctions



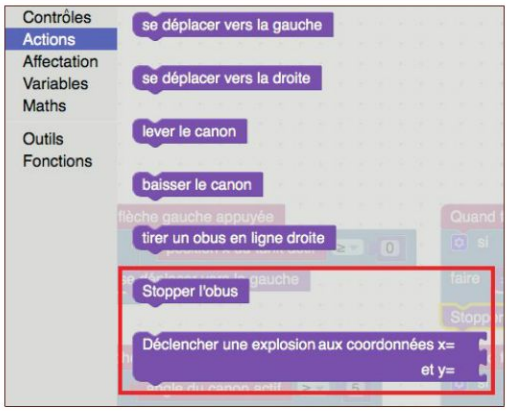
Jusqu'ici, tout va bien

Dans ce chapitre, tu vas apprendre comment détecter les collisions entre les différents sprites.

Jusqu'ici, tu as appris à coder de façon assez réaliste la trajectoire d'un obus tiré depuis un canon. Mais, tu as dû remarquer que ton obus se comportait assez bizarrement vis-à-vis des autres éléments du jeu vidéo : il les ignore complètement et passe au travers ! Nous allons donc coder la détection de collisions.

Quoi de neuf ?

L'état initial de l'exercice 4 est presque le même que l'état final de l'exercice 3. Il y a tout de même deux nouvelles actions qui te seront utiles dans ce chapitre, à savoir `Stopper l'obus` et `Déclencher une explosion aux coordonnées x = et y =`.



Nouveautés du chapitre

Ce que nous allons faire dans ce chapitre

Les obus que tirent nos tanks sont rigolos mais ils ne sont pas très dangereux : ils passent au travers de tous les éléments, y compris du sol !

C'est pareil pour nos tanks : même si nous les avons empêchés de sortir de l'écran, ils passent au travers l'un de l'autre sans souci. Nous allons donc remédier à cela en créant des fonctions qu'on appelle « détection de collisions ».

Ce sont des fonctions qui ne peuvent renvoyer que deux valeurs : « vrai » ou « faux ». Créons par exemple une fonction qui s'appellera `les deux tanks se touchent` ; celle-ci renverra `vrai` si effectivement les deux tanks se touchent, et `faux` si les deux tanks ne se touchent pas. Pour détecter une collision, nous allons appeler cette fonction tout au long de l'exécution du jeu vidéo. Quand la fonction renverra `vrai` à la place de `faux`, on pourra en déduire que les deux tanks se touchent.

Créons notre première fonction

Grande nouvelle, tu vas pouvoir réaliser ton propre bloc de code visuel ! En effet, le bloc `les deux tanks se touchent` n'existe pas encore, c'est à toi de le créer.

Pour cela, utilise le générateur de bloc que tu trouveras dans le sous-menu **Fonctions** : choisis le bloc qui s'appelle **pour faire quelque chose**, et qui possède une petite encoche `retour` en bas à droite.

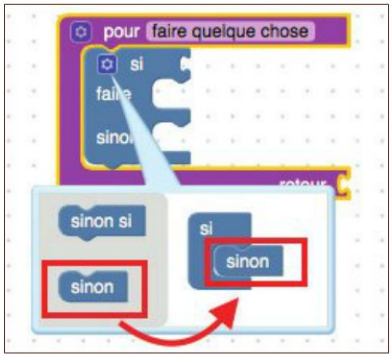
Bloc permettant de créer de nouveaux blocs de code

Dépose ce bloc sur l'espace de travail, puis renomme-le : remplace `pour faire quelque chose` par `les deux tanks se touchent`.

Attention

Pourquoi ne pas utiliser un verbe à l'infinitif mais un verbe conjugué pour nommer ce bloc ? La raison est toute simple : comme nous sommes en train de créer une fonction qui renvoie vrai ou faux, il faut trouver un nom qui ne symbolise pas une action mais plutôt un état, qui sera soit vrai, soit faux.

Le premier bloc à utiliser dans notre fonction est `si…faire` que tu trouveras dans le sous-menu **Contrôles**. Nous allons le transformer légèrement en un bloc `si…faire…sinon`, en cliquant sur le petit engrenage en haut à gauche du bloc : prendre le bloc `sinon` et le déposer sous le `si`, comme sur l'image suivante.



Transformation du bloc si … faire… en si … faire… sinon…

Valide ta modification de forme de bloc en cliquant de nouveau sur le petit engrenage.

Il faut ensuite que tu modélises la condition `si les deux tanks se touchent`. Pour cela, tu peux partir du principe que les deux tanks se touchent si leurs positions horizontales (sur l'axe des x) sont relativement proches. Nous allons donc mesurer la différence entre la position x du tank actif et la position x du tank adverse. Si cette distance est suffisamment petite (et donc les tanks suffisamment proches pour sembler se toucher), alors notre fonction devra renvoyer `vrai`. Sinon elle devra renvoyer `faux`. On va choisir une distance limite égale à 100 : si les deux tanks sont à une distance l'un de l'autre inférieure à 100, alors la fonction renverra `vrai` (et `faux` dans le cas contraire).

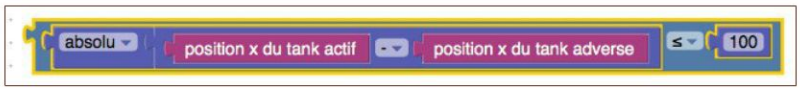
Un petit peu de maths

Suivant la position du tank actif par rapport au tank adverse, la différence entre les deux positions x sera soit positive, soit négative. Ce qui t'intéresse c'est la valeur de la différence, pas son signe. Nous allons donc utiliser une fonction mathématique appelée « valeur absolue » qui convertie une valeur négative en valeur positive.

Je t'invite donc à créer ta condition à l'aide des blocs suivants :

- via le sous-menu **Maths**, sélectionner le bloc `racine carrée` et choisir `absolu` via le menu déroulant du bloc ;
- `position x du tank actif` via le sous-menu **Variables** ;
- `position x du tank adverse` via le sous-menu **Variables** ;
- via le sous-menu **Contrôles**, choisir le bloc `=` et le modifier via le menu déroulant pour obtenir le bloc `≤`.

Voici à quoi devrait ressembler ta condition :



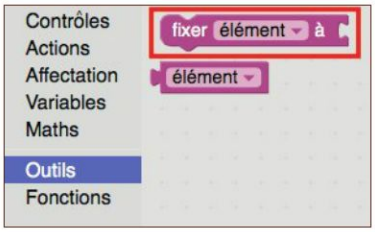
Code final de la condition

Il ne te reste plus qu'à clipser ta condition dans l'encoche du `si` de ta nouvelle fonction et tu auras fait le plus gros du travail !

Mais il va manquer deux points essentiels à ta fonction : que faire si le test est vrai, et que faire s'il est faux ?

Si le test est vrai, alors les deux tanks se touchent et nous devons renvoyer `vrai`. À l'inverse, s'il est faux, alors les deux tanks ne se touchent pas et nous devons renvoyer `faux`. Pour cela, nous allons créer une variable que nous appellerons `retour_fonction` et lui assigner la valeur `vrai` ou `faux`, selon le résultat du test.

Pour créer une nouvelle variable, choisis le bloc `fixer élément à` du sous-menu **Outils**.

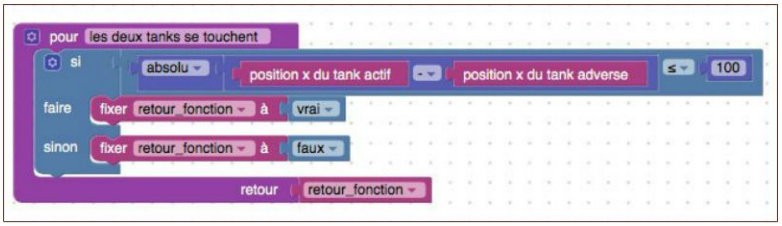


Utilisation des variables que tu crées toi-même

Dans le menu déroulant sous `élément`, choisis `Renommer la variable...` et donne-lui son nouveau nom (`retour_fonction`, par exemple). Puis, clipse ce bloc `fixer retour_fonction à` dans le bloc `faire`. Nous avons besoin d'initialiser cette variable à `vrai` : pour cela, il existe un bloc spécial dans le sous-menu **Contrôles** qui s'appelle `vrai`. Utilise-le pour fixer la variable à `vrai`.

Nous procéderons de la même façon pour le cas où la variable doit être initialisée à `faux`, mais cette fois le bloc `fixer retour_fonction à` existe déjà dans le sous-menu **Outils**. La valeur `faux` s'obtient avec le bloc `vrai`, mais en changeant sa valeur via le menu déroulant.

Bravo, ta fonction est presque terminée ! Pour finir, il faut lui demander de retourner la variable `retour_fonction` en clipsant la variable `retour_fonction` (disponible dans le sous-menu **Outils**) en bas à droite de ta fonction. Voici à quoi devrait ressembler ta fonction :



Code final de la détection de collision entre tanks

Une fonction qui n'est pas appelée ne sert à rien

Mais ta fonction ne sert à rien pour l'instant, car elle n'est jamais utilisée ! Or, le but de cette fonction est de pouvoir détecter une collision avec le tank adverse, il faut donc coder ce que ton jeu vidéo doit faire quand cette collision est détectée. Tu as le choix, mais une possibilité souvent utilisée est de faire rebondir en arrière un objet qui en percute un autre.

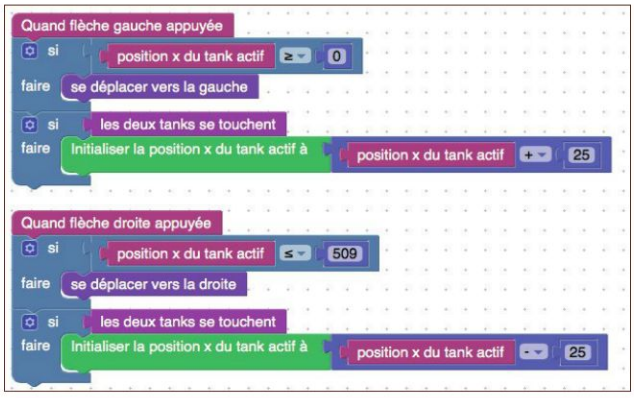
Pour coder ce rebond, il suffit de modifier les groupes de blocs `Quand flèche gauche appuyée` et `Quand flèche droite appuyée`, en ajoutant un test de collision à la fin : s'il y a collision, alors on modifie la position horizontale du tank actif pour le faire reculer brutalement, comme s'il rebondissait.

Change la position horizontale du tank actif pour le faire reculer brutalement, comme s'il rebondissait.

Effectue ces modifications sur les deux blocs, sachant que pour le bloc `Quand flèche gauche appuyée`, il faut augmenter la position x du tank actif de `25`, alors que pour le bloc `Quand flèche droite appuyée`, il faut diminuer la position x du tank actif de `25`. Tu auras besoin des blocs suivants, et n'oublie pas qu'il faut modifier deux groupes de blocs :

- `si…faire` ;
- `Initialiser la position x du tank actif à` ;
- `Position x du tank actif` ;
- `+` pour le groupe de blocs `Quand flèche gauche appuyée` et `-` pour le groupe de blocs `Quand flèche droite appuyée` ;
- `25`.

Une fois tes modifications terminées et testées, tu devrais obtenir quelque chose qui ressemble à cela :



Prise en compte de la détection de collision des tanks

Les obus n'aiment pas tomber par terre

La vie n'est pas toujours facile pour un obus : à peine touche-t-il le sol qu'il explose ! Mais pour l'instant, ce n'est pas le cas, et ta mission consiste à créer une nouvelle fonction qui détectera la collision entre un obus et le sol. Là encore, il s'agit d'une fonction qui renvoie `vrai` si la position verticale de l'obus dépasse le sol, ou `faux` si l'obus n'a pas encore atteint le sol.

Deuxième fonction

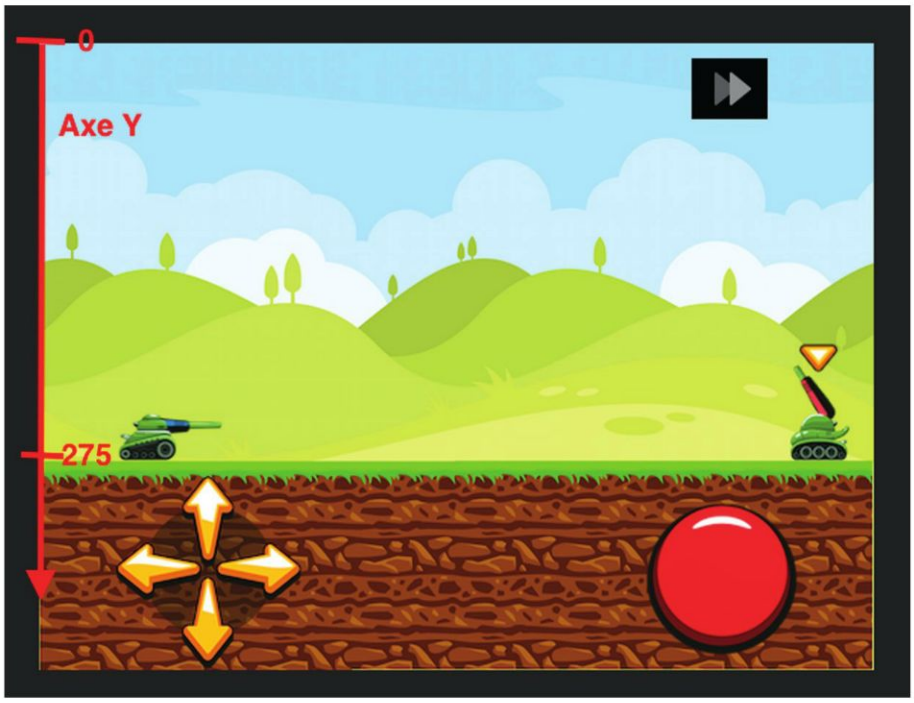
Pour créer cette fonction, tu peux t'inspirer de la fonction `les deux tanks se touchent`, et appliquer la même méthode en y apportant les modifications suivantes :

- nom de la fonction `l'obus touche le sol` ;
- variable à retourner : `retour_fonction` ;
- valeur de `retour_fonction` :
  - `Vrai` si la condition est réalisée ;
  - `Faux` si la condition n'est pas réalisée.

Pour écrire la condition, tu as besoin de connaître la position verticale du sol, car c'est elle qui permet de savoir si l'obus est au-dessus ou au-dessous du sol. Cette position vaut `275`.

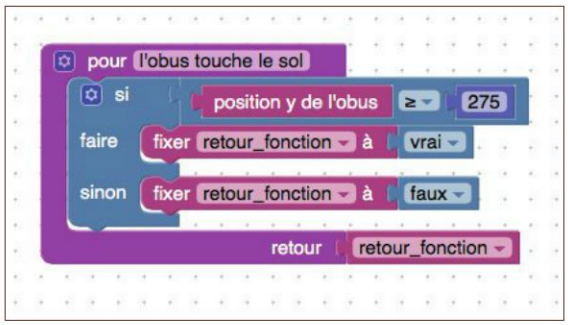
Attention

L'axe des y est orienté du haut vers le bas. Cela veut dire que le point le plus haut de l'écran a une coordonnée y égale à 0. La condition de ta fonction doit donc être vraie si la position y de l'obus est supérieure à 275. En effet, pendant sa course normale, la position y de l'obus est comprise entre 0 et 275. Aide-toi du schéma suivant pour mieux comprendre.



Axe y

Si tu as terminé ta fonction, elle devrait ressembler à ça :



Détection de collision entre le sol et l'obus

Une fonction qui n'est pas appelée ne sert décidément à rien !

Félicitations pour ta jolie fonction ! Mais comme tu l'as déjà compris, il faut l'appeler dans le code de ton jeu pour qu'elle serve réellement à quelque chose. L'endroit le plus adapté est le groupe de blocs `Quand un obus est en mouvement` : c'est grâce à ce groupe de blocs qu'on a codé l'influence de la gravité et du vent. Nous allons ajouter à ce

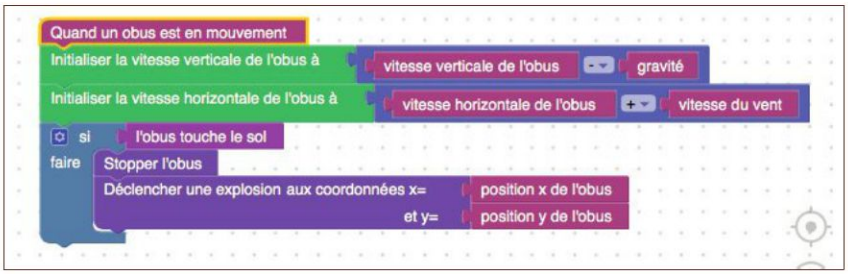
groupe la détection de l'impact avec le sol, et coder pour prévoir la réaction de ton jeu en cas d'impact.

Enfin une explosion !

L'idéal serait que l'obus stoppe sa course quand la collision avec le sol est détectée, puis qu'il disparaisse et soit remplacé par une animation d'explosion. Pour cela, tu auras besoin des blocs suivants :

- `si…faire` ;
- `l'obus touche le sol` ;
- `Stopper l'obus` ;
- `Déclencher une explosion aux coordonnées x = et y =` ;
- `position x de l'obus` et `position y de l'obus`.

Avec tout ce que tu as réussi à faire jusqu'ici, cela ne devrait pas te poser de problème. Mais, si tu as besoin d'aide, voici à quoi devrait ressembler le bloc de code `Quand un obus est en mouvement` :



Prise en compte de la détection de collision avec le sol

Jetons un coup d'œil au code généré automatiquement

À la fin de ce chapitre 4, ton jeu vidéo est quand même bien plus abouti qu'au début du cahier d'activités : les tanks ne passent plus au travers l'un de l'autre et les obus explosent en touchant le sol.

Si tu regardes le code généré, tu retrouveras facilement ce que tu as codé :

```
import math 1

retour_fonction = None 2
```

```
def l_obus_touche_le_sol(): ❸
  global retour_fonction ❹
  if getWeapon1Y() >= 275: ❺
    retour_fonction = True ❻
  else:
    retour_fonction = False ❼
  return retour_fonction ❽

def les_deux_tanks_se_touchent(): ❾
  global retour_fonction ❿
  if math.fabs(getActiveTankX() - getOponantTankX()) <= 100: ⓫
    retour_fonction = True ⓬
  else:
    retour_fonction = False ⓭
  return retour_fonction ⓮

for event in pygame.event.get():
      if event.type == "ObusMoving":
         setWeapon1VY((getWeapon1VY() - getGravity()))
         setWeapon1VX((getWeapon1VX() + getWindVX()))
      if l_obus_touche_le_sol(): ⓯
         stopperObus() ⓰
         doExplode((getWeapon1X()),(getWeapon1Y())) ⓱
```

Récapitulons :

❶ : on utilise une bibliothèque externe, appelée `math` (pour le calcul de la valeur absolue) ;

❷ : définition de la variable en tant que variable globale, c'est-à-dire utilisable partout dans le code ;

❸ : définition de notre fonction de détection de collision avec le sol ;

❹ : utilisation de la variable globale ;

❺ : condition (position y de l'obus plus grande que la position y du sol, donc l'obus est en dessous du sol à cause de l'orientation de l'axe des y) ;

❻ : initialisation de la variable retour à `vrai` ;

❼ : initialisation de la variable retour à `faux` ;

❽ : retour de la variable, fin de la fonction ;

❾ : définition de notre fonction de détection entre tanks ;

❿ : utilisation de la variable globale ;

⓫ : condition (la valeur absolue de la différence des positions x des deux tanks est inférieure à 100) ;

⓬ : initialisation de la variable retour à `vrai` ;

⓭ : initialisation de la variable retour à `faux` ;

⓮ : retour de la variable, fin de la fonction ;

⓯ : test de la détection de la collision avec le sol ;

⓰ : stopper la progression de l'obus ☆

⓱ : remplacer le sprite de l'obus par une animation d'explosion aux coordonnées x et y de l'obus au moment de l'impact.

Et maintenant ?

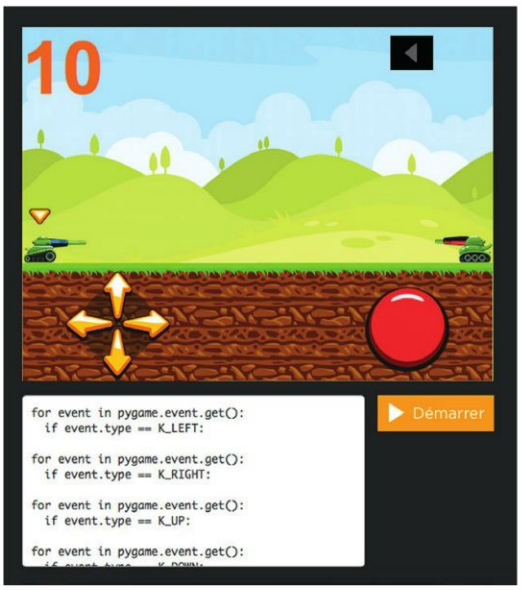
Le prochain chapitre va te permettre de finaliser ton jeu vidéo, en y apportant les derniers détails importants : mise en place d'un compteur de temps, changement automatique du tour de jeu, changement aléatoire du vent à chaque tour de jeu, et enfin, détection de la fin du jeu.

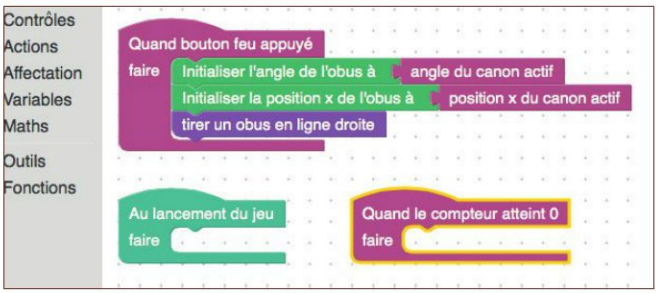
Chapitre 5

Finalisation de ton jeu vidéo

Notions abordées

- Mise en place des règles du jeu
- Boucles
- Nombres aléatoires





Bientôt la fin !

Dans ce chapitre, tu vas finaliser ton jeu vidéo, en apportant toutes les petites touches finales dont il a besoin.

Tu as énormément appris jusqu'ici, et ton jeu est presque terminé. Mais il manque quelques petites choses pour qu'il soit réellement « jouable » : comme c'est un jeu pour deux joueurs, il faudrait que chacun des joueurs puisse jouer l'un après l'autre.

De plus, qui dit jeu vidéo dit « but à atteindre » : ici, nous allons décider que pour gagner, il faut toucher le tank adverse avec un obus (ce qui est assez logique, tu ne trouves pas ?)

Quoi de neuf ?

Tu verras en haut à gauche un gros chiffre jaune : il s'agit du compteur de temps pour chaque tour. Pour l'instant il vaut `10`, et si tu lances ton programme, tu verras qu'il ne change pas. Nous allons y remédier !

Au niveau des blocs, nous avons ajouté :

- trois blocs Actions (`Lancer compteur`, `changer de tank actif` et `Afficher l'écran de victoire pour le tank n°`) ;
- deux blocs Affectation (`Initialiser le compteur à` et `Initialiser le vent à`) ;
- un bloc Variables (`numéro du tank adverse`) ;
- un bloc Maths (`entier aléatoire entre … et …`) ;
- deux blocs sur l'espace de travail : `Au lancement du jeu` et `Quand le compteur atteint 0`.

Ce que nous allons faire dans ce chapitre

Il ne manque pas grand-chose pour que ton jeu vidéo devienne une référence du genre, il faut juste lui apporter encore quelques petites améliorations.

- Nous allons **mettre en place des tours de jeu** : ainsi, deux joueurs pourront s'affronter, en jouant chacun leur tour. Pour cela, il suffit de créer un compteur et, à chaque fois qu'il arrivera à zéro, il faudra changer de tank actif. Avec ce système, tu ne pourras plus changer le tank actif en cliquant simplement sur les tanks, comme tu as pu le faire dans les exercices précédents.

Note
N'oublie pas de réinitialiser à 10 le compteur lorsque celui-ci atteint zéro !

- La **météo changera à chaque tour de jeu** : le vent changera pour une valeur aléatoire chaque fois qu'un nouveau tour de jeu commencera.
- Il faudra également mettre en place la **détection de collision entre un obus et un tank** : sans cette fonction de détection, impossible pour un joueur de détruire le tank adverse, et donc impossible de gagner...
- Enfin, il faudra **informer les joueurs lorsque la partie est terminée** : quand un tank aura été détruit, il faudra afficher un message précisant qui est le vainqueur.

Ce qui reste à faire te semble peut-être énorme, mais avec tout ce que tu as appris, tu vas y arriver très rapidement. C'est parti !

Mise en place du compteur et des tours de jeu

Pour lancer le compteur en début de partie, il faut faire deux choses à chaque lancement de partie.

- **Choisir la valeur de départ du compteur** : cette valeur est en secondes, si tu choisis 20 pour initialiser le compteur, cela signifie que chaque tour de jeu durera 20 secondes. Tu peux garder 10 si tu veux, mais c'est ton jeu : c'est toi qui choisis !
- **Puis lancer le compteur :** sinon, il ne va jamais descendre jusqu'à zéro.

Pour cela, je te laisse travailler. Petit indice, il faudra que tu utilises les blocs suivants :

À exécuter lors du lancement du programme

Initialiser le compteur

Nombre

Démarrer le décompte du compteur

N'oublie pas de tester ton compteur (en cliquant sur **Démarrer**) et observe ce qu'il se passe quand le compteur atteint zéro. Le résultat te surprend-il ? Je suis sûr que non : maintenant que tu es devenu un expert en conception de jeu vidéo, tu as compris que rien ne se fait tout seul. Si tu veux que quelque chose arrive quand le compteur arrive à zéro, c'est toi qui dois le coder.

Ne jamais rester à zéro

La première chose à faire quand le compteur atteint zéro, c'est tout simplement de lui réaffecter sa valeur d'origine, pour qu'il recommence à compter. Pour que les joueurs ne soient pas perdus, il vaut mieux lui affecter la même valeur que lors du lancement du jeu, mais c'est à toi de décider !

La seconde chose à faire, c'est d'inverser le tank actif et le tank au repos : n'oublie pas que le but de ce compteur est de mettre en place des tours de jeu alternés.

Pour réinitialiser le compteur et inverser les tanks actifs, tu vas avoir besoin de ces blocs :

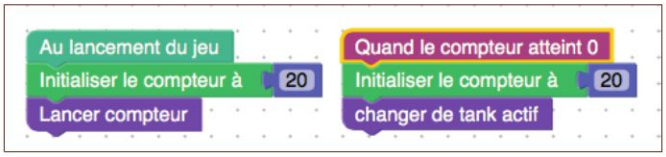
À exécuter quand le compteur arrive à zéro

Initialiser le compteur

Nombre

Changer de tank actif

N'hésite pas à tester, et tu verras que ton jeu se rapproche de plus en plus d'un « vrai » jeu vidéo ! Si tu as besoin d'aide, voici à quoi devrait ressembler la mise en place des tours de jeu :



Mise en place des tours de jeu

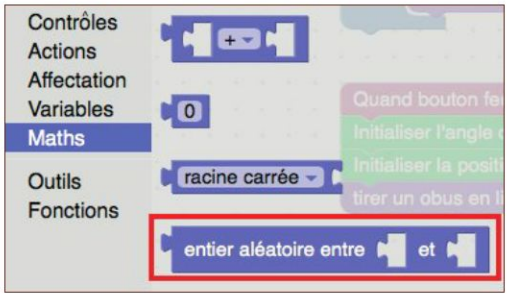
« Mais enfin, Dieu ne joue pas aux dés ! » (Albert Einstein)

Toi aussi tu t'es déjà fait tremper par une averse alors que la météo de la veille annonçait un soleil radieux ? Alors n'hésite pas à introduire un peu de hasard dans ton jeu vidéo : il n'en sera que plus amusant à jouer.

Nous allons changer aléatoirement la vitesse du vent, à chaque nouveau tour de jeu. Pour cela, tu vas devoir utiliser le nouveau bloc Affectation `Initialiser le vent à`. Tu peux t'amuser à tester ce nouveau bloc, en lui passant en entrée une valeur comprise entre 0 et 8. L'effet sur le vent sera celui décrit dans le tableau suivant.

Valeur en entrée de la fonction `Initialiser le vent à`	Effet sur le vent
0	Vent très fort vers la gauche
1	Vent fort vers la gauche
2	Vent modéré vers la gauche
3	Vent faible vers la gauche
4	Vent faible vers la droite
5	Vent modéré vers la droite
6	Vent fort vers la droite
7	Vent très fort vers la droite
8	Absence de vent

Ce qui nous intéresse ici, ce n'est pas de choisir la force du vent, mais plutôt de lui appliquer une valeur au hasard. Ce dernier peut être déterminé par le bloc **Maths** `entier aléatoire entre … et …`.



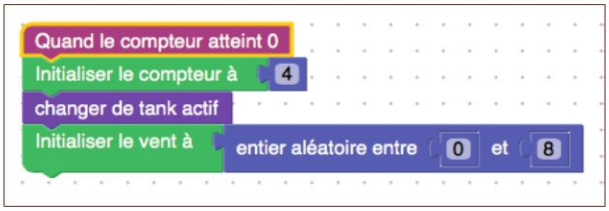
Bloc de génération d'un nombre aléatoire

Les deux valeurs en entrée de ce bloc doivent absolument être `0` et `8`, pour que seules les valeurs du tableau précédent puissent être choisies au hasard.

Note

Si tu choisis 0 et 3 comme valeurs limites, tu limites le choix du hasard aux seules valeurs du vent vers la gauche. De même si tu choisis 4 et 7, le vent soufflera au hasard, mais uniquement vers la droite…

Si tu as besoin d'un coup de pouce, voici à quoi devrait maintenant ressembler ton groupe de blocs :



Mise en place du hasard

Quand un obus rencontre un tank…

Il te reste une dernière fonction à coder : il s'agit de la détection de collision entre un obus et le tank adverse. Pour cela, procède de la même façon que pour les fonctions de détection de collision vues au chapitre 4 (collision entre un obus et le sol ou collision entre deux tanks). Tu pourras appeler cette fonction `l'obus touche un tank`.

Souviens-toi, le plus compliqué à coder est la condition qui permet de détecter une collision ou non. Dans notre cas, il faut vérifier deux choses :

- si la position x de l'obus est comprise entre les deux extrémités du tank (`position x du tank -20` pour l'extrémité gauche et `position x du tank +20` pour l'extrémité droite) ;
- si la position y de l'obus est comprise entre le sol et le sommet du tank (soit entre `240` pour le sommet et `275` pour le sol).

Pour t'aider à coder cette condition qui prend beaucoup de place sur ton écran, la voici en langage courant :

```
(
(position x de l'obus < (position x du tank adverse + 20))
et
(position x de l'obus > (position x du tank adverse - 20))
                )
et
(
(position y de l'obus < 275)
et
(position x de l'obus > (240)
                )
```

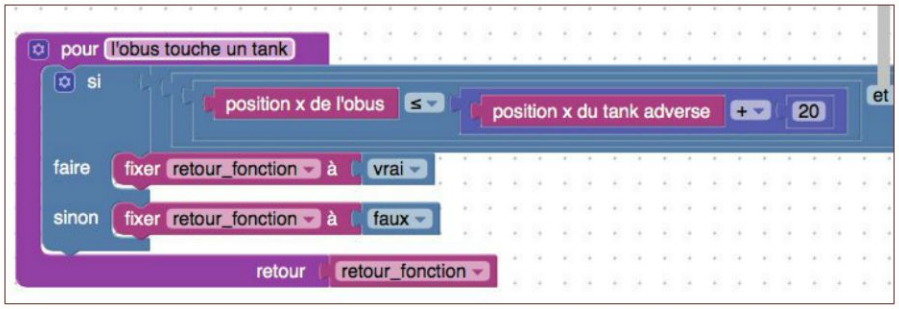
Tu vois qu'il faut utiliser trois blocs `et` : celui du milieu (le deuxième `et` listé ci-dessus) prendra en paramètres les deux autres blocs `et`. Le bloc `et` de gauche (le premier listé ci-dessus) permet de vérifier les deux conditions sur la position x de l'obus et celui de droite, les deux conditions sur la position y de l'obus.

Au final, voici à quoi devrait ressembler les deux parties de cette condition, reliées par un `et` :

Partie gauche de la condition

Partie droite de la condition

Le reste de la fonction ne devrait pas te poser de problème majeur :



Détection de la collision entre un tank et un obus

Et voilà ta dernière détection de collision prête à être utilisée ! Tout comme pour la détection de collision entre le sol et un obus, nous appelons cette fonction tout au long du mouvement d'un obus, donc à la suite du groupe de blocs `Quand un obus est en mouvement`. Le code sera le même que lorsqu'un obus touche le sol : la seule différence consiste à déclencher une explosion aux coordonnées du tank au lieu de l'obus (pour que cela soit plus spectaculaire).

Note

Pense bien à choisir les coordonnées x et y du tank adverse pour générer ton explosion. Si tu utilises les coordonnées du tank actif (qui a tiré l'obus), ton joueur risque de ne pas être content !

Rappel des blocs à utiliser pour effectuer un test de détection de collision

Bloc si ... faire...

Test : est-ce que l'obus touche un tank ?

Stopper le mouvement de l'obus

Déclencher une explosion

N'oublie pas de tester : si tout va bien, tu devrais voir une explosion apparaître quand un obus touche le tank adverse !

Game over

Il ne te reste plus qu'à coder l'écran informant les joueurs que la partie est finie et donnant le nom du vainqueur.

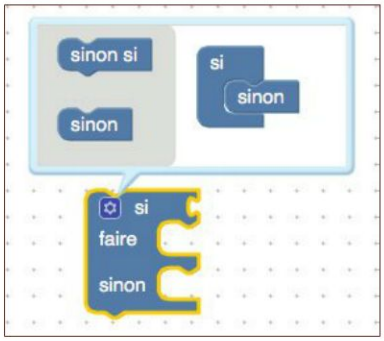
Nous sommes déjà capables de détecter quand un tank est touché par une explosion : il suffit d'ajouter à ce bloc les instructions permettant d'afficher l'écran d'information de fin de partie.

Dernière subtilité à prendre en compte : le bloc `Afficher l'écran de victoire pour le tank n°` a besoin qu'on lui passe en paramètre le numéro du tank qui a gagné. Il faut donc récupérer le numéro du tank adverse (c'est forcément le tank

adverse qui reçoit l'obus tiré par le tank actif). Si le tank adverse est le tank n°2, c'est le tank n°1 qui a gagné, et vice versa.

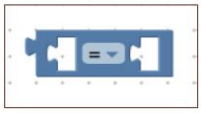
À toi de jouer : assemble les derniers blocs de code te permettant d'afficher cet écran de fin de partie.

Tu vas avoir besoin des blocs suivants :



Bloc si... faire... sinon...

Numéro du tank adverse



Bloc =

Afficher un écran de victoire pour tank

N'oublie pas de tester ! Si jamais tu as des soucis avec cette dernière modification, voici à quoi devrait ressembler ton code :

Mise en place de la détection de fin de partie

Enfin !

Tu peux souffler car tu viens de terminer ton jeu vidéo !

Ton jeu est enfin terminé !

Et le code dans tout ça ?

Voici le code généré en Python avec les modifications que tu viens d'apporter :

```
import math
import random

retour_fonction = None

def l_obus_touche_un_tank():
  global retour_fonction
  if (getWeapon1X() <= getOponantTankX() + 20 and getWeapon1X() >=
getOponantTankX() - 20) and (getWeapon1Y() <= 275 and getWeapon1Y() >= 240):1
    retour_fonction = True
  else:
    retour_fonction = False
  return retour_fonction
```

```
for event in pygame.event.get():
      if event.type == "ObusMoving":
         setWeapon1VY((getWeapon1VY() - getGravity()))
         setWeapon1VX((getWeapon1VX() + getWindVX()))
      if l_obus_touche_le_sol():
         stopperObus()
         doExplode((getWeapon1X()),(getWeapon1Y()))
      if l_obus_touche_un_tank(): (2)
         stopperObus()
         doExplode((getOponantTankX()),(getOponantTankY()))
         if getOponantTank() == 2: (3)
            showVictory(1) (4)
         else:
            showVictory(2)

for event in pygame.event.get():
      if event.type == "Start":
         compteur=10
         LancerCompteur()

for event in pygame.event.get():
      if event.type == "CompteurVautZero": (5)
         compteur=10 (6)
         changeActiveTank() (7)
         setWind((random.randint(0, 8))) (8)
```

(1) : cette condition est la plus longue que tu auras codée dans ce cahier d'activités ! Avec trois `et` imbriqués, pas facile de ne pas se tromper. Si tu es suffisamment attentif et à l'aise avec les priorités mathématiques, tu verras que Python utilise un système de priorité pour faire l'économie de quelques parenthèses ;

(2) : utilisation de ta nouvelle fonction de détection de collision ;

(3) : récupération du numéro du tank adverse ;

(4) : affichage de la bannière de victoire pour le tank vainqueur ;

(5) : quand le compteur arrive à zéro, il est temps de gérer le changement de tour ;

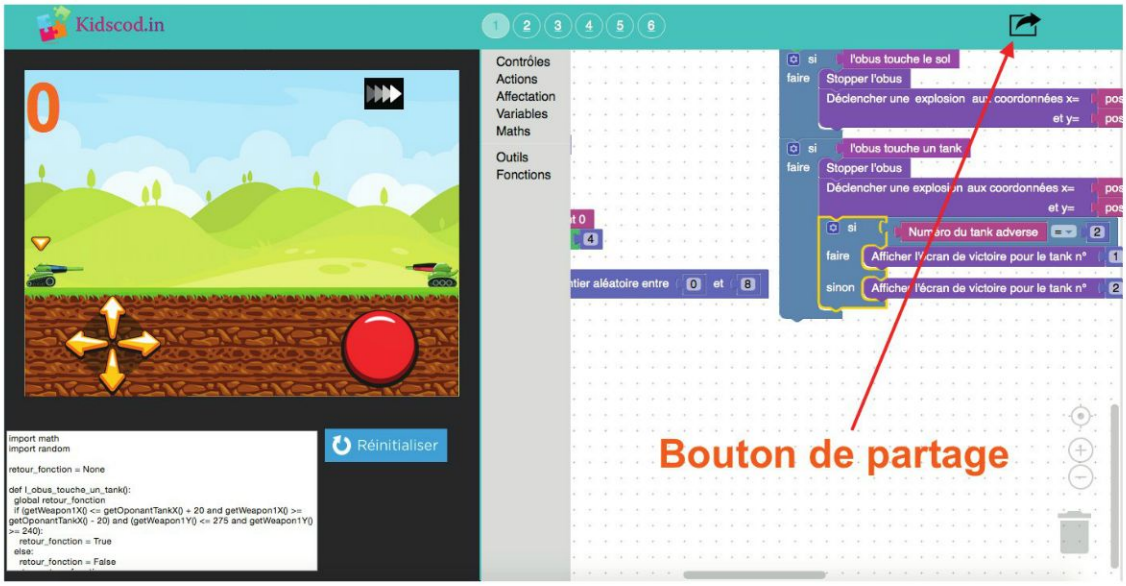
(6) : réinitialisation du compteur ;

❼ : changement du tank actif ;

❽ : changement de la force du vent, avec un nombre aléatoire tiré au sort entre 0 et 8.

Et maintenant ?

Tu peux te reposer un peu et profiter de ton jeu vidéo ! Et pourquoi pas le partager sur Internet avec tes amis ou ta famille ? Pour cela, clique sur l'icône de partage en haut à droite de la page.



Comment partager ta création ?

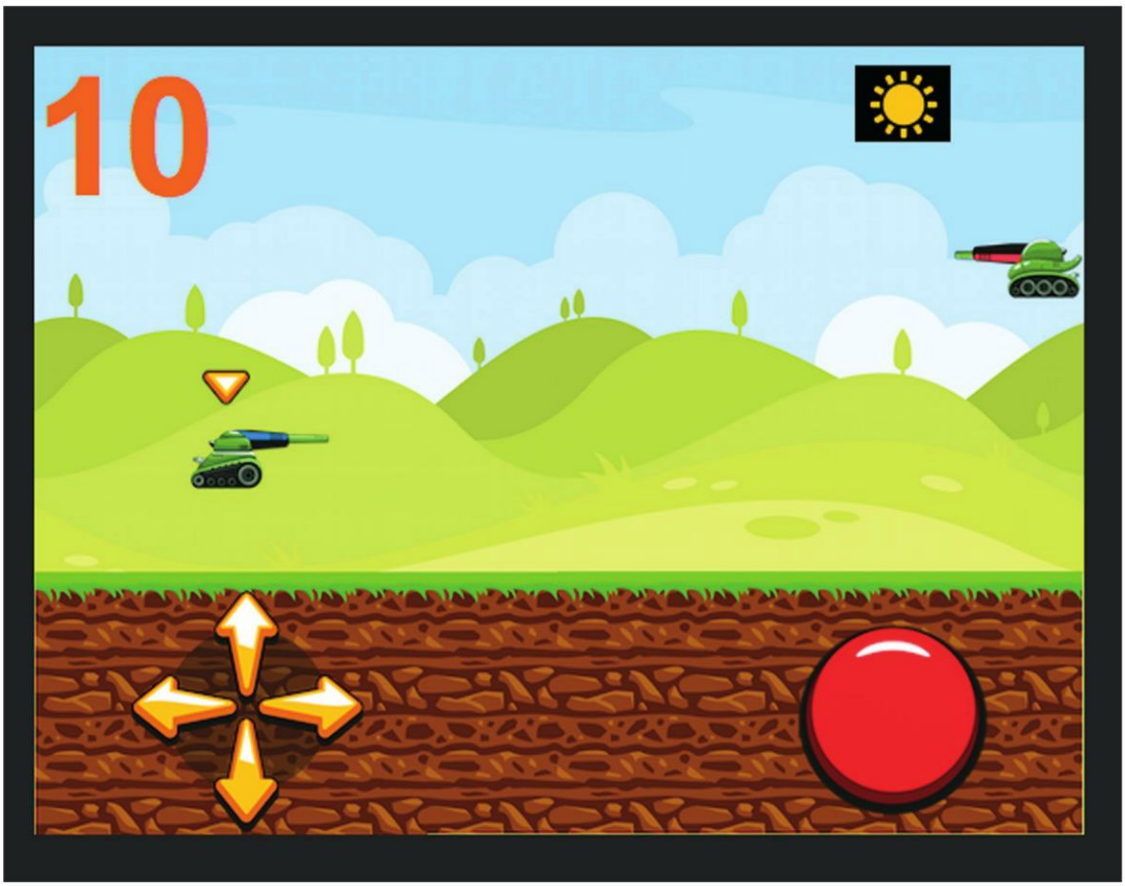
Tu obtiendras une URL de la forme http://www.kidscod.in/app/tank/player.php?level=XXX&hash=YYY qu'il te suffit d'envoyer par e-mail (ou autre) aux personnes de ton choix. En cliquant sur cette URL, tes amis pourront tester ton jeu vidéo, sans le modifier. À toi de leur montrer ce que tu es capable de faire !

Chapitre 6

Cacher des codes de triche

Notion abordée

- Mise en place de codes de triche



Qui a dit que des tanks ne pouvaient pas voler ?

Dans tous les jeux vidéo, il existe des codes permettant de tricher. Ceux-ci ne sont en général pas documentés car ils ne rendent pas le jeu plus agréable. Ils sont surtout utiles aux développeurs d'un jeu pour faire des tests plus rapidement.

Si un développeur veut tester son écran **Jeu terminé**, penses-tu qu'il ait envie de finir le jeu à chaque fois ? En général, pour tester son écran de fin sans attendre, il préférera cacher un code dans le jeu qui lui permet de terminer le jeu instantanément.

Dans cet ultime chapitre, nous allons voir quels codes de triche tu pourrais mettre en place, soit pour t'aider à modifier ton jeu, soit pour faire des blagues à tes adversaires.

Quoi de neuf ?

Ce chapitre 6 démarre exactement là où le chapitre 5 se termine : le jeu est tout à fait fonctionnel. Nous avons simplement ajouté un bloc `Quand touche Entrée appuyée` sur l'espace de travail : il va falloir te servir de ce bloc pour coder tes codes de triche.

Un premier exemple : la téléportation

Quoi de plus pratique que de pouvoir se téléporter n'importe où instantanément ? C'est ce que nous allons faire pour nos tanks favoris ! Tu peux essayer de mettre en place ce premier code de triche :

« si touche entrée appuyée, alors déplacer instantanément le tank actif à la position x = 200 ».

Pour cela, il suffira d'utiliser les blocs `Quand touche Entrée appuyée` (c'est un nouveau bloc spécial codes de triche) et le bloc d'affectation `Initialiser la position x du tank actif à`.

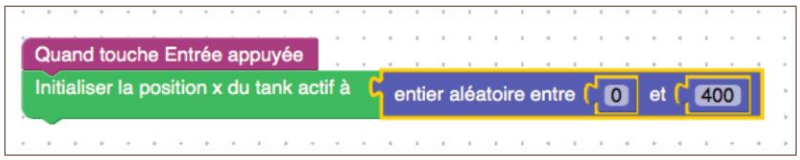
Téléportation instantanée !

N'hésite pas à tester ton code de triche : il te permettra soit de te placer derrière le tank adverse (c'est plus sûr), soit de téléporter ton tank derrière un obus en plein mouvement (assez pratique si on veut tester les tirs d'obus sans forcément mourir à chaque fois).

On ne joue jamais assez avec le hasard

Maintenant que tu maîtrises la téléportation, pourquoi ne pas ajouter une part de hasard dans le déplacement instantané de ton tank ? Tu pourrais, par exemple, utiliser le bloc `entier aléatoire entre … et …` en choisissant les valeurs `0` et `450`.

Rappel
0 correspond à l'extrémité gauche de l'écran, et 550 correspond à l'extrémité droite de l'écran.



Téléportation instantanée, mais au hasard…

Je t'encourage à tester ton nouveau code : il peut être très utile pour rendre fou ton adversaire !

Il est temps de jouer à tricher

Tu trouveras dans le tableau suivant une liste de quelques codes de triche que tu pourrais tester. La méthode sera toujours la même : le code de triche sera déclenché par l'appui sur la touche **Entrée**. Libre à toi de choisir ton code de triche, sachant que rien n'interdit d'en cumuler plusieurs à la suite…

Action du code de triche	Blocs à utiliser	Utilité du code de triche
Stopper l'obus instantanément	`Stopper l'obus`	Pouvoir tester le tir d'obus sans attendre que l'obus sorte de l'écran ou rencontre un obstacle. Éviter qu'un tank se fasse exploser par un obus.
Déclencher une explosion à un endroit aléatoire	`Déclencher une explosion aux coordonnées x= et y=` `entier aléatoire entre … et …`	Tester l'animation d'explosion. Faire peur à ton adversaire.
Changer de tank actif	`changer de tank actif`	Ne pas attendre la fin du tour adverse pour tester. Faire sauter le tour de ton adversaire.
Afficher l'écran de victoire instantanément	`Afficher l'écran de victoire pour le tank n°`	Tester l'affichage de l'écran de victoire. Énerver ton adversaire.
Téléporter un obus	`Initialiser la position x de l'obus à`	Tester le comportement d'un obus aux différents endroits du décor sans avoir à viser précisément. Rendre ton adversaire fou.
Accélérer ou ralentir la vitesse de l'obus	`Initialiser la vitesse verticale de l'obus à` `Initialiser la vitesse horizontale de l'obus à`	Ajuster les paramètres physiques du moteur du jeu vidéo.
Réinitialiser le compteur instantanément	`Initialiser le compteur à`	Permettre d'allonger le temps du tour de jeu, en cas de besoin. Permettre également d'apprendre la patience à ton adversaire.
Changer la vitesse du vent	`Initialiser la vitesse du vent à`	Tester l'influence du vent sur la course de l'obus. Faire se retourner un obus contre ton adversaire à cause du vent.

À toi de « jouer » !

Fais travailler ton imagination, n'oublie pas que tu peux créer des fonctions pour enchaîner des actions élémentaires. Tu peux donc imaginer des codes de triche beaucoup plus évolués que ceux décrits ici. Alors, n'hésite pas !

Pour partager ton jeu vidéo modifié, pense à cliquer sur l'icône de partage en haut à droite.

C'est fini ?

En ce qui concerne ce cahier d'activités, oui c'est malheureusement terminé. J'espère qu'il t'aura donné envie de continuer à créer tes propres jeux vidéo !

Tu as acquis les bases qui te permettront d'aller plus loin dans l'apprentissage de la programmation et tu peux t'attaquer à la conception d'un jeu plus évolué si tu le souhaites.

Le langage Python est un outil formidable pour continuer sur ta lancée : en t'aidant du module pygame (http://pygame.org/) et de ce que tu as appris, tu as en mains tout ce qu'il te faut pour programmer des merveilles !

Index