**Fichiers de condiguration .csv** (à ouvrir avec Excel ou Libre Office, il faut bien configurer le séparateur comme étant le “;” dans les paramètres régionaux de Windows).

Situé dans :***Unstable\StarFighter\StarFighter\Assets\Scripts\...***

***Remarque: toutes les configurations prennent comme référence le jeu joué en vertical scrolling de bas en haut. Lorsque le jeu est orienté différemment, le moteur fera les calcules de transformations nécessaires.***

* **Weapons.csv** : l’arme permet de tirer des munitions qui infligeront des dégâts. Un objet qui tire (joueur ou ennemi) doit donc se voir donner une Weapon, et un Ammo sur lequel brancher la Weapon.
  + **weapon\_name** : le nom qui sera affiché pour l’arme (*display\_name*)
  + **image** : asset 2D représentant l’arme dans l’interface et quand l’arme est affichée dans le jeu sous forme de loot.
  + **width** : largeur de l’image en pixels
  + **hieght** : hauteur de l’image en pixels
  + **frames** : nombre de frames si l’image est animée. Si l’image possède plusieurs frames, la largeur et hauteur donnée précédemment doit être celle d’une frame, mais l’asset 2D utilisé doit inclure toutes les frames les unes à la suite des autres, de gauche à droite
  + **ammo\_name**: nom de la munition tirée, configurée dans ***Ammo.csv***
  + **damage** : dommages infligés par la munition sur une cible ennemie
  + **vspeed**: vitesse de la munition tirée (en fonction de la direction du scrolling de la scène)
  + **range**: portée de la munition tirée avant de disparaitre
  + **rate\_of\_fire** : cadence de tir en secondes. Accept les flottants (ex: 0.5)
  + **rafale** : nombre de bullets à partir duquel l’arme se voit imposer un délai de rechargement avant de pouvoir retirer (donné par rafale\_cooldown). Si rafale < 0, alors cela active le mode « beam » : tir en faisceau continu. Dans ce cas, mettre une speed à 0. Le *rate\_of\_fire* devient le temps de recharge entre deux faisceaux et *rafale\_cooldown*, la durée du faisceau avant rechargement (0 pour infini).
  + **rafale\_cooldown**: délai de rechargement entre deux rafales.

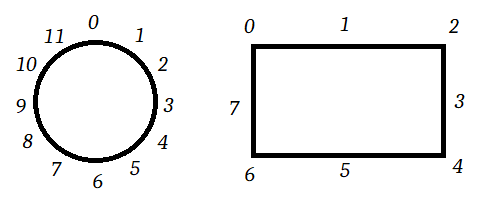
Note: le délai de rechargement n’est pas comptabilisé dans la cadence de tir de l’arme. Sa valeur “rate of fire” est appliquée entre chaque tir au sein d’une rafale.

* + **multishot** : nombre de munitions partant en même temps. Si multishot est supérieur à 1, l’arme aura des options de tirs supplémentaires dans les colonnes qui suivent.
  + **xspread**: (si multishot > 1) espace horizontal entre chaque munition tirée, dans le cas d’un tir multiple.
  + **dispersion**: (si multishot > 1) donne l’amplitude des tirs du multishot en degrés. Les tirs seront donc orientées de manière à former un arc de cercle et chaque tir est orienté de manière à former des angles égaux dont la somme fait la valeur demandée (exemple: 3 tirs réglés avec une dispersion de 120° donne 3 tirs formant deux tranches de 60°).
  + **angle\_offset**: permet de décaler le tir d’un angle spécifié en degrés. Cela fera aussi tourner l’arme autour du sprite du tireur.
  + **alternate**: (si multishot > 1) options de tir permettant de tirer toutes les munitions d’un tir multiple une par une dans un certain ordre.
    - “***0***” pour ne pas utiliser cette fonction.
    - “**ascending**” pour les tirer dans l’ordre de la plus à gauche à la plus à droite.
    - “**descending**” de droite à gauche.
    - “**ascending2**” alterne les motifs “ascending” et “descending” en commençant par “ascending”.
    - “**descending2**” alterne les motifs “ascending” et “descending” en commençant par “descending”.
    - “**alternate**” part du centre puis tire la bullet la plus proche à gauche, puis à droite, puis à gauche etc… en alternant vers les bullets les plus extérieurs au fur et à mesure.

Note: la cadence de tir de chaque bullet tirée individuellement est calculée à partir de la cadence de tir de l’arme divisée par le nombre de multishot, ceci afin que la totalité des bullets du tir soit toujours tirée dans la cadence de tir prévue pour le multishot.

* **target\_homing**: active le mode “tête chercheuse”, permettant à chaque tir de partir automatiquement dans la direction de l’ennemi le plus proche au moment du tir (ne tient pas compte de la vitesse de la cible). Pour les tirs ennemis, ce sera le joueur le plus proche.
  + “***0***” pour ne pas utiliser cette fonction.
  + “**homing**” pour que le tir vise automatiquement la position de l’ennemi au moment du tir
  + “**semi\_homing**” pour que la visée de la cible ne soit calculée qu’une fois par rafale (la rafale partira vers la position calculée au moment du 1er tir)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **No homing** | **Semi-homing** | **Homing** |
| **Ennemi sans face\_target** | Tire dans la direction actuelle. | Tire vers le joueur, position calculée en début de salve uniquement. | Tire vers le joueur, position calculée à chaque tir. |
| **Ennemi sans face\_target avec un laserbeam** | Tire dans la direction actuelle. | Tire dans la direction actuelle, puis fige son orientation durant toute la salve. | Tire vers le joueur, position actualisée toute la salve. |
| **Ennemi avec « face\_target »** | Se tourne vers le joueur et tire uniquement lorsqu’aligné. | Se tourner vers le joueur et tire vers lui, même si pas aligné, puis fige son orientation durant toute la salve. | Se tourne vers le joueur et tire vers lui, même s’il n’est pas aligné, position calculée à chaque tir. |
| **Ennemi avec « face target » et un laserbeam (rafale = -1)** | Tire dans la direction actuelle. | Tire vers le joueur, position calculée en début de salve uniquement. | Tire vers le joueur, position actualisée pendant toute la salve. |
| **Ennemi avec « face\_target » et un laserbeam** | Se tourne vers le joueur et tire uniquement lorsqu’aligné (avec une tolérance plus grande), et continue de se tourner vers lui pendant la salve. | Se tourne vers le joueur et tire uniquement lorsqu’aligné (avec une tolérance plus grande), puis fige son orientation durant toute la salve. | Se tourne vers le joueur et tire avec son orientation actuelle, même s’il n’est pas aligné, et continue de se tourner vers lui pendant la salve. |
| **Joueur** | Tire droit devant. | Tire vers l’ennemi le plus proche, position calculée en début de salve uniquement. | Tire vers l’ennemi le plus proche, position calculée à chaque tir. |
| **Joueur avec un laserbeam** | Tire droit devant. | Tire vers l’ennemi le plus proche, position calculée en début de salve uniquement. | Tire vers l’ennemi le plus proche, position actualisée pendant toute la salve. |

* **pattern**: type de mouvement, sachant que le référentiel est la position du joueur (le bot suit toujours la position du vaisseau du joueur).
  + “***0***” pour ne pas utiliser de mouvement supplémentaire.
  + “***circle***” pour faire des cercles ou des oscillations (si height = 0).
  + “***rectangle***” pour faire des rectangles ou des lignes allers-retour (si height = 0)
* **speed**: vitesse du pattern, si la colonne “pattern” a été réglée avec une valeur autre que “0”.
* ***cw (clockwise)*** : sens de circulation du pattern.
  + “***1***” = sens des aiguilles d’une montre
  + “***-1***” sens inverse des aiguilles d’une montre
  + “***0***” = l’un ou l’autre choisi aléatoirement au début du pattern, sauf si le pattern précédent étant un pattern de même nature, dans ce cas on continue dans le même sens que précédemment.
* **starting\_point** : point de départ dans le pattern
* 
* **width** : amplitude du pattern sur l’axe des X.
* **height**: amplitude du pattern sur l’axe des Y.

**Ammo.csv:** les Weapons doivent être branchées sur des munitions, qui définissent les bullets qui apparaitront au moment du tir, tandis que l’arme gère la cadence de tir, l’orientation, la cible etc...

* **ammo\_name**: nom de la munition, tel qu’elle sera utilisée dans les fichiers de configuration.
* **image**: asset 2D du projectile
* **width**: largeur en pixels de l’image
* **height**: hauteur en pixel de l’image
* **frames**: nombre de frames si l’image est animée. Si l’image possède plusieurs frames, la largeur et hauteur donnée précédemment doit être celle d’une frame, mais l’asset 2D utilisé doit inclure toutes les frames les unes à la suite des autres, de gauche à droite.
* **FX\_explosion**: nom du FX d’explosion du projectile à l’impact, configuré dans ***FX.csv***
* **sound\_name** : asset sonore utilisé au moment du tir (pour l’instant pas branché dans le jeu)
* **rotation\_speed** : vitesse angulaire de rotation du projectile sur lui-même (en degrés)

**Scenes.csv:** (a.k.a. “Map stellaire”, elle permet d’organiser l’enchainement des niveaux et donc le monde ouvert dans lequel on évolue)

* **scene\_name**: nom de la scène tel qu’il sera utilisé dans le fichiers de config
* **filename**: adresse du fichier de configuration de la scène
* **display\_name**: nom de la scène tel qu’il sera affiché dans le jeu
* **link\_UP**: nom de la scène vers laquelle on peut transiter par la bordure supérieure de la scène actuelle. Elle sera accolée à la bordure supérieure du background de la scène, et centrée. Entrer “0” pour n’avoir aucune scène vers laquelle aller vers le haut.
* **link\_DOWN**: idem pour une scène liée par le bord inférieur du background de la scène actuelle.
* **link\_RIGHT**: idem pour une scène liée par le bord droit du background de la scène actuelle.
* **link\_LEFT** : idem pour une scène liée par le bord gauche de la scène actuelle.

**Scenes/**MyVerticalScene1**.csv:** il faut créer un fichier par scène et les placer dans Assets/Scripts/Scenes/. afin de les utiliser dans Scenes.csv

Ce fichier permet de configurer à la fois le background de la scène et les ennemis qui seront spawnés dans le niveau. Selon la valeur entrée dans la première colonne, les colonnes suivantes serviront donc à des choses différentes.

* “**bg**” : à entrer dans la première colonne pour indiquer que les colonnes suivantes de cette ligne servent à configurer le background de la scène.
  + **filename**: adresse de l’asset 2D correspondant au background
  + **width**: largeur de l’asset
  + **height**: hauteur de l’asset
  + **vspeed**: vitesse de scrolling (en fonction de la direction du scrolling de la scène)
  + **direction**: entrer “H” pour que la scène soit jouée en scrolling horizontal, et “V” pour un scrolling vertical. Le sens (haut/bas et droite/gauche) dépend dans quel sens le joueur est arrivée dans la scène, les scènes étant jouables dans un sens comme dans un autre.
* “**enemy**”: à entrer dans la première colonne pour indiquer que les colonnes suivantes de cette ligne servent à configurer un type d’ennemi de la scène
  + **enemy\_name**: nom de l’ennemi, faisant référence à un ennemi existant dans Enemy.csv
  + **proba**: pondération de la probabilité d’apparition de cet ennemi dans le niveau. Le jeu fera la somme des valeurs de tous les ennemis du niveau et en déduira des probabilités d’apparition (*exemple: si ennemi1 a une valeur de “1”, ennemi2 a une valeur de “3”, alors l’ennemi1 a 25% de chances d’être spawné au détriment de ennemi2, qui a 75% de chances d’être spawné*).

Un jet aléatoire est fait au moment où l’on spawn un ennemi : il sera choisi au hasard parmi les types d’ennemis possibles dans cette scène, pour une classe d’ennemi donnée).

* + **pool\_size**: nombre max d’ennemis de ce type pouvant être spawné dans un niveau (désactivé actuellement)
  + **enemy\_class**: classe d’ennemi. Chaque classe d’ennemi a des probabilités de loot différentes. De plus, les classes servent à construire des groupes d’ennemis appelés “enemy pool” a.k.a cluster d’ennemis (voir spec: ***StarFighter/Documentation/EnemyPools.docx***).

Note: il est prévu d’ajouter à ce fichier une liste de cluster d’ennmis parmi laquelle on fera un jet aléatoire pour décider lequel spawner. Ensuite le cluster s’occupera de sélectionner des ennemis pour chaque classe d’ennemi utilisée par le cluster, parmi les ennemis réglés pour cette scène. Actuellement le cluster n’est pas configurable mais hardcodé.

* “**boss**” : spawn un boss (ou plusieurs) boss de fin de niveau, qui va bloquer l’accès au portail de fin de zone (le portail disparaitra jusqu’à ce que le boss soit vaincu).
  + **name**: nom de l’ennemi, tel qu’il est référencé dans Enemies.csv
  + **enemy\_class**: classe d’ennemi, qui définit la qualité de ses loots
  + **spawnX**: coordonnées d’apparition (axe X). Scripté, contrairement aux ennemis aléatoires pendant le niveau)
  + **spawnY**: coordonnées d’apparition (axe Y).
  + **rotation**: rotation par rapport à l’orientation standard d’un ennemi dans cette direction. Permet d’ajouter par exemple “180” degrés et faire apparaitre un boss à l’envers.

**Ship.csv:** permet de configurer le vaisseau de départ du joueur

* **shipconfig\_name**: nom de la configuration (si on veut pouvoir passer d’une configuration de départ à l’autre)
* **airbrake**: nom de l’équipement de type Airbrake utilisé par la config. Doit faire référence à un nom d’un équipement configuré dans ***Equipement.csv***
* **armor**: nom de l’équipement de type Armor
* **shield**: nom de l’équipement de type Shield
* **module**: nom de l’équipement de type Module
* **weapon**: nom de l’arme utilisée. Doit faire référence à un nom d’arme configuré dans ***Weapons.csv***
* **ammo**: nom de la munition utilisée par l’arme. Doit faire référence à un nom de munition configuré dans Ammo.csv
* **shipmodel**: nom du modèle de vaisseau de la configuration. Doit faire référence à un nom de Shipmodel configuré dans ***Equipement.csv***
* **deathFX**: nom du FX à jouer à la mort du vaisseau. Doit faire référence à un nom de FX configuré dans ***FX.csv***

**Equipment.csv:**

Une configuration de vaisseau (ship\_config) est définie par un modèle de vaisseau (shipmodel) et une liste d’équipements de type unique : Airbrake, Engine, Armor, Shield, Module, ainsi que d’une arme branchée sur des munitions à tirer. La configuration ne peut donc avoir qu’un équipement de chaque type, et il est obligatoire d’avoir un shipmodel. La puissance des différentes stats de la configuration du vaisseau sont obtenues en additionnant les stats du shipmodel et de tous les équipements utilisés.

Quand un loot est ramassé, il est automatiquement équipé si la configuration n’avait aucun équipement de ce type. Sinon, il est placé dans le coffre du vaisseau (“stash”) et pourra être équipé ou revendu via l’interface du jeu (inexistante actuellement).

* **type**: définie le type de l’objet, parmi les valeurs “engine”, “armor”, “shield”, “module”, et “shipmodel”
* **item\_name**: nom de l’équipement, tel qu’il sera utilisé dans les fichiers de configuration
* **texture**: adresse de l’asset 2D représentant l’objet dans l’interface et quand il est affiché dans le jeu sous forme de loot. L’asset du shipmodel donne la boite de collision du vaisseau. Ce n’est cependant pas celle qu’on voit visuellement (voir la colonne **fake\_texture** plus loin).
* **width** : largeur de l’image en pixels
* **hieght** : hauteur de l’image en pixels
* **frames** : nombre de frames si l’image est animée. Si l’image possède plusieurs frames, la largeur et hauteur donnée précédemment doit être celle d’une frame, mais l’asset 2D utilisé doit inclure toutes les frames les unes à la suite des autres, de gauche à droite
* **max\_speed**.x: statistique de vitesse de déplacement horizontal maximum donné par l’objet.
* **max\_speed**.y: idem en vertical
* **decceleratior**: bonus au coefficient de freinage donné par l’objet. Il définit à quelle vitesse le vaisseau arrivera à réduire sa vitesse jusqu’à arrêt complet.
* **acceleration**.x: bonus à l’accélération horizontale donné par l’objet.
* **acceleration**.y: idem en vertical
* **armor**: bonus aux points de vie
* **shield**: bonus au bouclier. Les dommages encaissés sont infligés en priorité au bouclier, puis quand le bouclier est réduit à 0, aux points de vie. Le bouclier a une régénération par tick, mais il ne se régénère plus une fois détruit.
* **shield\_regen**: bonus à la vitesse de régénération du bouclier. Lorsque le bouclier est détruit (valeur réduite à 0), il ne se régénère plus.
* **damage**: bonus aux dommages infligés aux ennemis lorsque le vaisseau est en contact avec eux
* **bot**: permet à l’objet de spawner un “bot” qui assistera le joueur. Doit faire référence au nom d’unbot configuré dans le fichier Bot.csv
* **fake\_texture**: nom de l’asset 2D à afficher par-dessus l’asset défini précédemment. Cet asset n’a aucune hitbox, il est un habillage visuel du vaisseau, tandis que la boite de collision est donnée par le véritable asset du shipmodel.

Note: quand le joueur utilise les aérofreins (fonction Braking, avec la touche L-MAJ), la vitesse du vaisseau est réduite par un coefficient, et le véritable asset du vaisseau se retrouve affiché par-dessus la fake texture.

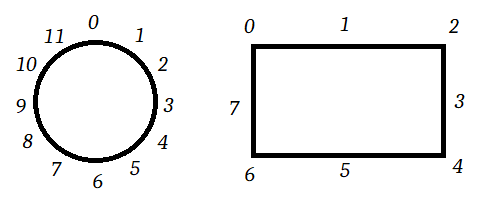
* **fake\_width** : largeur de la fake texture en pixels
* **fake\_hieght** : hauteur de la fake texture en pixels
* **fake\_frames** : nombre de frames si la fake texture est animée. Si l’image possède plusieurs frames, la largeur et hauteur donnée précédemment doit être celle d’une frame, mais l’asset 2D utilisé doit inclure toutes les frames les unes à la suite des autres, de gauche à droite

**Bot.csv:** les bots (parfois appelés “adds” ou “options” dans différents shmups) sont des petits vaisseaux qui accompagnent le joueur, le plus souvent pour lui fournir un appui-feu. Ils ne collisionnent pas avec les tirs ennemis ni avec les ennemis. Ils suivent automatiquement le mouvement du joueur.

* **bot\_name** : nom du bot, tel qu’il sera utilisé dans les fichiers de configuration.
* **armor** : barre de vie du bot (pour l’instant pas utilisé car le bot ne prend pas de dommages)
* **shield** : idem
* **shield\_regen** : idem
* **damage** : dommages que le bot inflige aux contact d’ennemis (pour l’instant inactif car les bots ne collisionnent pas avec les ennemis.

*Remarque: On pourra imaginer cepenendant des bots qui collisionnent avec les ennemis pour leur infliger des dommages de contact).*

* **weapon\_name** : nom de l’arme du bot configurée dans Weapons.csv
* **image**: asset 2D du bot
* **width**: largeur en pixels de l’image
* **height**: hauteur en pixel de l’image
* **frames**: nombre de frames si l’image est animée. Si l’image possède plusieurs frames, la largeur et hauteur donnée précédemment doit être celle d’une frame, mais l’asset 2D utilisé doit inclure toutes les frames les unes à la suite des autres, de gauche à droite.
* **explosion**: nom du FX de mort du bot, configuré dans FX.csv
* **pattern**: type de mouvement, sachant que le référentiel est la position du joueur (le bot suit toujours la position du vaisseau du joueur).
  + “***0***” pour ne pas utiliser de mouvement supplémentaire.
  + “***circle***” pour faire des cercles ou des oscillations (si height = 0).
  + “***rectangle***” pour faire des rectangles ou des lignes allers-retour (si height = 0)
* **speed**: vitesse du pattern, si la colonne “pattern” a été réglée avec une valeur autre que “0”.
* ***cw (clockwise)*** : sens de circulation du pattern.
  + “***1***” = sens des aiguilles d’une montre
  + “***-1***” sens inverse des aiguilles d’une montre
  + “***0***” = l’un ou l’autre choisi aléatoirement au début du pattern, sauf si le pattern précédent étant un pattern de même nature, dans ce cas on continue dans le même sens que précédemment.
* **starting\_point** : point de départ dans le pattern



* **width** : amplitude du pattern sur l’axe des X
* **height**: amplitude du pattern sur l’axe des Y
* **xspread**: décalage du bot par rapport au vaisseau du joueur sur l’axe horizontal, en pixels
* **yspread**: idem pour le décalage vertical

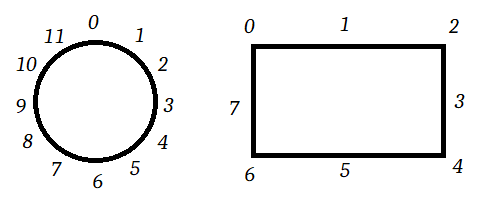
**FX.csv:** pour créer des explosions d’impact ou de mort

* **explosion**: nom du FX tel qu’il sera utilisé dans les fichiers de configuration
* **asset**: adresse de l’asset 2D du FX
* **width**: largeur de l’image en pixels
* **height**: hauteur de l’image en pixels
* **frames**: nombre de frames si l’image est animée. Si l’image possède plusieurs frames, la largeur et hauteur donnée précédemment doit être celle d’une frame, mais l’asset 2D utilisé doit inclure toutes les frames les unes à la suite des autres, de gauche à droite.
* **duration**: durée de vie du FX en secondes avant de ne plus être affiché. Une frame est réglée à 200ms actuellement, donc pour une animation de 3 frames il faut prévoir une durée de 0.6 secondes par exemple.

**Enemies.csv:**

* **enemy\_name**: nom de l’ennemi tel qu’il sera utilisé dans les fichiers de configuration
* **armor**: points de vie
* **shield**: bouclier. Les dommages encaissés sont infligés en priorité au bouclier, puis quand le bouclier est réduit à 0, aux points de vie. Le bouclier a une régénération par tick, mais il ne se régénère plus une fois détruit.
* **damage**: dommages infligés au contact du joueur
* **weapon\_name**: nom de l’arme utilisée. Doit faire référence à un nom d’arme configuré dans ***Weapons.csv***
* **weapon\_name2**: idem pour une 2ème arme. L’ennemi peut actuellement porter 3 armes, mais ça pourrait être plus si on branchait des colonnes supplémentaires. Toutes les armes de l’ennemi tirent automatiquement en fonction de leurs caractéristiques.
* **weapon\_name3**: idem pour une 3ème arme.
* **image**: adresse de l’asset 2D de l’ennemi
* **width**: largeur de l’image en pixels
* **height**: hauteur de l’image en pixels
* **frames**: nombre de frames si l’image est animée. Si l’image possède plusieurs frames, la largeur et hauteur donnée précédemment doit être celle d’une frame, mais l’asset 2D utilisé doit inclure toutes les frames les unes à la suite des autres, de gauche à droite.
* **vspeed**: vitesse de scrolling (en fonction de la direction du scrolling de la scène)
* **explosion\_name**: nom du FX joué à la mort de l’ennemi, et configuré dans FX.csv
* **value**: valeur de l’ennemi. Sert au cacul du niveau du loot généré par sa mort, et du nombre de points de Hazard qu’il donne (cf. Documentation/HazardLevel.docs).
* **phase**: si l’ennemi doit changer d’arme ou de pattern de mouvement au cours de son existence, il doit passer par le système de phase.
  + “***0***” pour ne pas utiliser.
  + Sinon, entrer le nom d’une phase configurée dans ***EnemyPhases.csv*** : ce sera la phase initiale de l’ennemi quand il est spawné. Si l’ennemi se voit configuré avec une phase, les colonnes de Weapon1, 2, 3 seront ignorées au profit des valeurs configurées dans la phase initiale.

**EnemyPhases.csv:**

* **phase\_name**: nom de la phase d’ennemi tel qu’il sera utilisé dans les fichiers de configuration.
* **vspeed**: vitesse de scrolling (en fonction de la direction du scrolling de la scène)
* **weapon\_name**: nom de l’arme utilisée. Doit faire référence à un nom d’arme configuré dans ***Weapons.csv***
* **weapon\_nameN**: idem pour une Nème arme. Une phase peut comporter jusqu’à 6 armes simultannées. Cela peut se changer rapidement dans le code.
* **pattern**: type de mouvement, sachant que le référentiel est la position du joueur (le bot suit toujours la position du vaisseau du joueur).
  + “***0***” pour ne pas utiliser de mouvement supplémentaire.
  + “***circle***” pour faire des cercles ou des oscillations (si height = 0).
  + “***rectangle***” pour faire des rectangles ou des lignes allers-retour (si height = 0)
* **speed**: vitesse du pattern, si la colonne “pattern” a été réglée avec une valeur autre que “0”.
* ***cw (clockwise)*** : sens de circulation du pattern.
  + “***1***” = sens des aiguilles d’une montre
  + “***-1***” sens inverse des aiguilles d’une montre
  + “***0***” = l’un ou l’autre choisi aléatoirement au début du pattern, sauf si le pattern précédent étant un pattern de même nature, dans ce cas on continue dans le même sens que précédemment.
* **starting\_point** : point de départ dans le pattern
* 
* **width** : amplitude du pattern sur l’axe des X.
* **height**: amplitude du pattern sur l’axe des Y.
* **rotation\_speed**: vitesse de pivot. Utilisé pour les modifiers ***face\_target*** et ***reset\_facing***, ou sinon par défaut pour tourner sur soi-même à l’infini.
* **modifier** : si différent de “0”, applique des modifications en fonction de la valeur :
  + ***“immune”***: immunisé contre les dégats.
  + ***“ghost”***: imminusé contre les collisions (et masque semi-transparent appliqué).
  + ***“death”***: auto-destruction instantanée.
  + “**face\_target**” se tourne vers la cible avant de tirer, à la vitesse donnée par ***rotation\_speed.*** Ne tire que si aligné, à moins d’avoir du Homing.
  + “**reset\_facing**” se tourne dans le sens du scrolling de la scène, à la vitesse donnée par ***rotation\_speed***
  + “***bouncing***” rebondit contre les bords d’écran
  + “***bouncingH***” rebondit contre les bords d’écran horizontaux
  + “***bouncingV***” rebondit contre les bords d’écran verticaux
  + “**freeze**” bloque les mouvements du joueur jusqu’à ce que l’ennemi soit détruit ou qu’il démarre une nouvelle phase qui n’a pas ce modifier
  + “**kill\_bullets**” détruit tous les tirs ennemis présents à l’écran
* ***modifier\_2*** : idem, pour ajouter un 2ème modifier.
* ***modifier\_3*** : idem, pour ajouter un 3ème modifier.
* **welcome\_weapon**: permet de configurer une arme qui tirera une fois au moment où la phase entre en vigueur. Permet à un ennemi de faire une attaque post-mortem (exemple: une mine qui explose et projette des débris).
* **condition**: type de condition à vérifier pour changer de phase. Si une valeur autre que “0” est entrée, il faut régler les 3 colonnes suivantes.
  + “***0***” pour que la phase soit immuable.
  + “***positionV***” calcule la coordonnée verticale de l’ennemi (dans le référentiel d’une scène orientée en scrolling vertical vers le haut toujours).
  + “***positionH***” calcule la coordonnée horizontale de l’ennemi.
  + “***playerPosY*** calcule la coordonnée verticale du joueur (dans le référentiel d’une scène orientée en scrolling vertical vers le haut toujours).
  + “***playerPosX***” calcule la coordonnée horizontale du joueur.
  + “***life***” vérifie le pourcentage de vie restante de l’ennemi. “shield” vérifie son pourcentage de bouclier restant.
  + “***phaseClock***” permet de passer une condition sur le temps écoulé depuis le début de la phase.
  + “***enemyClock***” permet de passer une condition sur le temps écoulé depuis le spawn de l’ennemi, ou de la dernière transition survenue en utilisant une condition enemyClock (le chronomètre est remis à zéro quand la condition est vérifiée).
  + “***proximity***” permet de passer une condition sur la distance avec le joueur le plus proche.
  + “***shotsFired***” permet de passer une condition sur le nombre de tirs effectués par toutes les armes additionnées de cet ennemi (mais un tir de plusieurs bullets instantanément ne compte que pour 1 tir, pour une arme donnée).
  + “***rotation***” permet de passer une condition sur le degré de rotation de l’ennemi
* **op**: type d’opérateur logique à uiliser pour vérifier la condition :
  + “***greater***”
  + “***equal***”
  + “***lesser***”.
* **value**: la valeur du test. Par exemple on veut déclencher la prochaine phase si la “positionV” de l’ennemi est supérieure à 100pixels (op=greater, value=100).

Note: Dans le cas des conditions de type “***life***” ou “***shield***” pour tester si l’ennemi à 0 points de vie ou 0 de bouclier, on considère qu’il n’a pas une valeur “supérieure à 0”, mais qu’il a une valeur “égale” ou “inférieure” à 0. Dans le doute, utiliser “égal” à 0 c’est le plus logique.

* **transition**: nom de la phase qui sera lancée si la condition réglée dans les colonnes suivantes est vérifiée.
* **condition2**: idem. Ceci est une 2ème condition possible. La première des 2 conditions qui est vérifiée emporte la priorité. Si une valeur autre que “0” est entrée, il faut régler les 3 colonnes suivantes.
* **op2**: opérateur logique de la condition 2.
* **value2**: valeur à comparer du test de la condition 2.
* **transition2**: nom de la phase lancée si la condition 2 est vérifiée2.