FONDATION La main à la pâte
POUR L'ÉDUCATION À LA SCIENCE

Cycle 4

Programmer un jeu d'arcade sur Scratch : algorithmique branchée

Une séquence du projet *1,2,3... CODEZ !*

Résumé

Cette séquence a pour objectif d'initier les élèves à la programmation, à travers la conception et la réalisation d'un jeu vidéo simple (type « jeu d'arcade »). Au cours de cette séquence, ils découvrent et s'approprient de nombreux concepts propres à l'informatique : algorithme, variable, test, boucle, langage, bug, événement… Cette séquence s'adresse en priorité à des élèves débutants en programmation Scratch.

Projet « Programmation d'un jeu d'arcade » : initiation à *Scratch*

Ce projet s'adresse en priorité à des élèves débutants ou ayant une première expérience en programmation *Scratch*. Pour des élèves confirmés, nous suggérons de mener le projet « Programmation d'un jeu de plateforme », page 351.

Afin d'éviter des répétitions dans les différentes séquences, nous n'avons pas intégré, ici, de séance de découverte de *Scratch* : nous renvoyons pour cela au chapitre « Introduction générale à *Scratch* » (page 71). Ce chapitre propose, outre cette séance de prise en main (familiarisation avec l'interface, exercices d'initiation, etc.), une discussion générale (pourquoi le choix de *Scratch*, quels sont ses avantages et ses limites, comment l'installer, etc.) et quelques conseils pour un usage en classe.

Disciplines concernées et liens avec les programmes

Les programmes de 2016 introduisent des notions d'informatique aussi bien en mathématiques qu'en technologie. Nous conseillons de mener ce projet dans le cadre du cours de mathématiques afin de cibler l'enseignement de technologie vers des projets qui mettent davantage en avant l'aspect matériel, comme la robotique (cf. pages 66 et 67).

Les programmes 2016 de mathématiques comportent un chapitre « algorithmique et programmation » qui justifie parfaitement un tel projet :

> Au cycle 4, les élèves s'initient à la programmation, en développant dans une démarche de projet quelques programmes simples, sans viser une connaissance experte et exhaustive d'un langage ou d'un logiciel particulier. En créant un programme, ils développent des méthodes de programmation, revisitent les notions de variables et de fonctions sous une forme différente, et s'entraînent au raisonnement.
> – Décomposer un problème en sous-problèmes afin de structurer un programme ; reconnaître des schémas.
> – Écrire, mettre au point (tester, corriger) et exécuter un programme en réponse à un problème donné.
> – Écrire un programme dans lequel des actions sont déclenchées par des événements extérieurs.
> – Programmer des scripts se déroulant en parallèle.
> • Notions d'algorithme et de programme.
> • Notion de variable informatique.
> • Déclenchement d'une action par un évènement, séquences d'instructions, boucles, instructions conditionnelles.
> • Notion de message échangé entre objets.

Le professeur de mathématiques peut mener le projet seul, ou en collaboration avec d'autres professeurs (notamment d'arts plastiques pour le design des différents éléments du jeu).

Objectifs

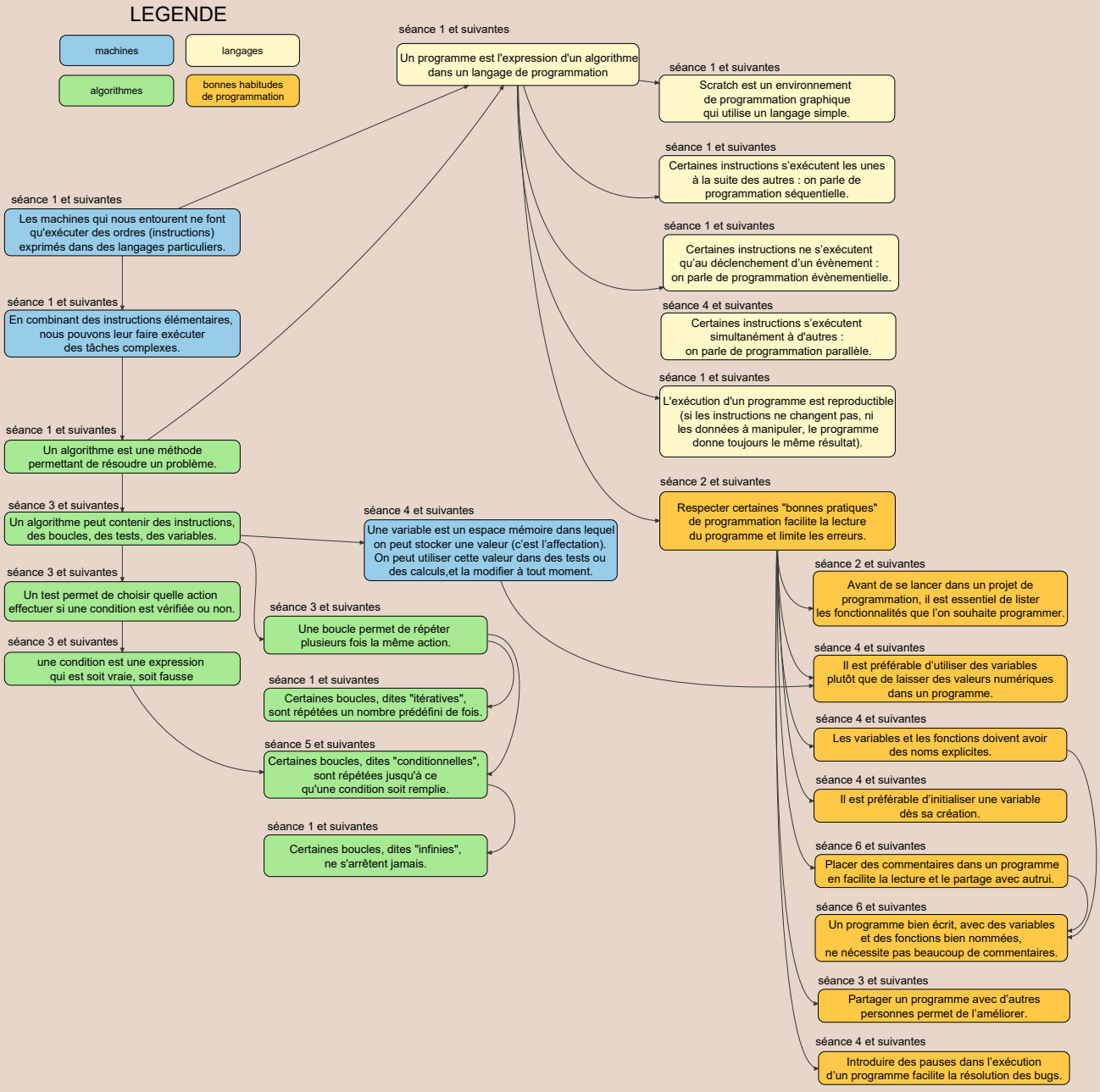
L'objectif du projet est d'initier les élèves à la programmation, à travers la conception et la réalisation d'un jeu vidéo simple (type « jeu d'arcade »). Au cours de ce projet, ils découvrent et s'approprient de nombreux concepts propres à l'informatique : algorithme, variable, test, boucle, langage, bug, événement, etc. (*cf.* scénario conceptuel, page suivante).

Résumé des séances

Avec des élèves débutants, réaliser l'ensemble du projet nécessitera 7 séances d'1 heure. Certains binômes auront produit un jeu plus complet et complexe que d'autres, mais tout le monde aura produit un jeu abouti, satisfaisant et valorisant.

	Séance	Titre	Page	Résumé
	Séance 1	Découvrir *Scratch*	82	Les élèves découvrent l'environnement de programmation *Scratch*. Ils apprennent à créer un programme enchaînant quelques instructions simples.
	Séance 2	Définir les mécanismes du jeu	83	Les élèves constituent un listing des fonctionnalités d'un jeu d'arcade. Ils organisent ces fonctionnalités sous la forme d'une carte mentale (mind-map) qui servira de feuille de route pour le projet.
	Séance 3	Premier programme : piloter l'avatar (utilisation de structures de contrôle SI… ALORS)	87	Les élèves créent leur premier programme permettant de piloter l'avatar du joueur. Ils apprennent également à personnaliser la scène dans *Scratch* (lutin et arrière-plan), ainsi qu'à enregistrer leur travail pour le réutiliser plus tard.
	Séance 4	Programmer la chute des pièges : utilisation de variables	92	Les élèves utilisent des variables pour programmer la chute des pièges (vitesse de chute, et nombre de vies restantes). Ils réinvestissent également des structures conditionnelles SI… ALORS vues précédemment.
	Séance 5	Programmer la chute des pièges (suite) : utilisation de clones	98	Les élèves utilisent une fonctionnalité propre à *Scratch* permettant de cloner des lutins pour programmer la chute d'un nombre aléatoire de pièges.
	Séance 6	Programmer la chute des récompenses : révision générale	102	Programmer la chute des récompenses permet aux élèves de revenir sur les variables, les clones, les boucles, les tests et les nombres aléatoires.
	Séance 7	Améliorer le rendu graphique du jeu : costumes et messages	106	Les élèves améliorent le rendu graphique du jeu en créant et en manipulant des « costumes » pour chaque lutin. Ils créent également un lutin appelé « game over » qui apparait à la fin du jeu. Pour faire communiquer les différents programmes entre eux, ils utilisent des « messages ».

Scénario conceptuel



Séance 1 – Découvrir *Scratch*

Discipline dominante	Mathématiques ou technologie
Résumé	Les élèves découvrent l'environnement de programmation *Scratch*. Ils apprennent à créer un programme enchaînant quelques instructions simples.
Notions nouvelles *(cf. scénario conceptuel, page 81)*	Machines : • Les machines qui nous entourent ne font qu'exécuter des ordres (instructions) exprimés dans des langages particuliers. • En combinant des instructions élémentaires, nous pouvons leur faire exécuter des tâches complexes. Algorithmes : • Un algorithme est une méthode permettant de résoudre un problème. Langages • Un programme est l'expression d'un algorithme dans un langage de programmation. • *Scratch* est un environnement de programmation graphique, qui utilise un langage simple. • Certaines instructions s'exécutent à la suite les unes des autres : on parle de « programmation séquentielle ». • Certaines instructions ne s'exécutent qu'au déclenchement d'un évènement : on parle de « programmation évènementielle ». • L'exécution d'un programme est reproductible (si les instructions ne changent pas, ni les données à manipuler, le programme donne toujours le même résultat).
Modalités d'investigation	Programmation
Matériel	Pour chaque binôme : • Un ordinateur possédant une connexion Internet (pour utiliser la version en ligne de *Scratch*) ou ayant *Scratch* préinstallé Pour l'enseignant : • Un ordinateur possédant une connexion Internet (pour utiliser la version en ligne de *Scratch*) ou ayant *Scratch* préinstallé • Un vidéoprojecteur, utile lors des mises en commun

Cette séance est décrite pas à pas dans le chapitre « Introduction générale à *Scratch* », page 71.

Séance 2 – Définir les mécanismes du jeu

Discipline dominante	Mathématiques
Résumé	Les élèves constituent un listing des fonctionnalités d'un jeu d'arcade. Ils organisent ces fonctionnalités sous la forme d'une carte mentale (*mind-map*) qui servira de feuille de route pour le projet.
Notions nouvelles *(cf. scénario conceptuel, page 81)*	Bonnes habitudes de programmation : • Avant de se lancer dans un projet de programmation, il est essentiel de lister les fonctionnalités que l'on souhaite programmer.
Matériel	Pour chaque binôme : • (facultatif) Un ordinateur possédant une connexion Internet • Des post-it® de deux couleurs différentes et une affiche au format A2 Pour l'enseignant : • Un ordinateur possédant une connexion Internet (pour utiliser la version en ligne de *Scratch*) ou ayant *Scratch* préinstallé • Un vidéoprojecteur • (facultatif) Une plateforme de mind-mapping collectif : – Par exemple, un compte sur https://coggle.it au nom de l'enseignant – La liste, au format numérique, des e-mails des élèves

Avant-propos

Dans le descriptif de ce projet (c'est-à-dire à partir du paragraphe suivant), on suppose que l'enseignant dispose d'un ordinateur connecté à internet avec système de vidéo-projection, et que les élèves ont un poste informatique par binôme, avec accès internet également. Si ce n'est pas le cas, il est tout à fait possible de mener le projet hors-ligne, après installation de *Scratch* sur chaque poste. Dans ce cas, ne pas tenir compte de tout ce qui concerne les comptes *Scratch* ou la plateforme de *mind-mapping* collectif.

Situation déclenchante

L'enseignant demande à la classe quels sont les différents types de jeux vidéo. Attention, il ne faut pas citer des titres de jeux, mais une typologie : jeux de stratégie, jeux d'arcade, jeux de combat, jeux de tir, jeux d'aventure, jeux de rôle, jeux de réflexion, jeux de simulation, etc.

Il explique que tous les groupes d'élèves vont programmer un « jeu d'arcade », ce qui permettra d'échanger sur les difficultés rencontrées et de s'entraider plus efficacement que si chacun se lance dans un type de jeu totalement différent. Au-delà de ce choix commun d'arcade, il y aura des possibilités de personnalisation, donc autant de jeux produits que de groupes d'élèves. Les premières séances seront guidées, mais les dernières seront très libres. L'aboutissement du projet va nécessiter environ 5 séances de programmation… c'est donc un travail de longue haleine.

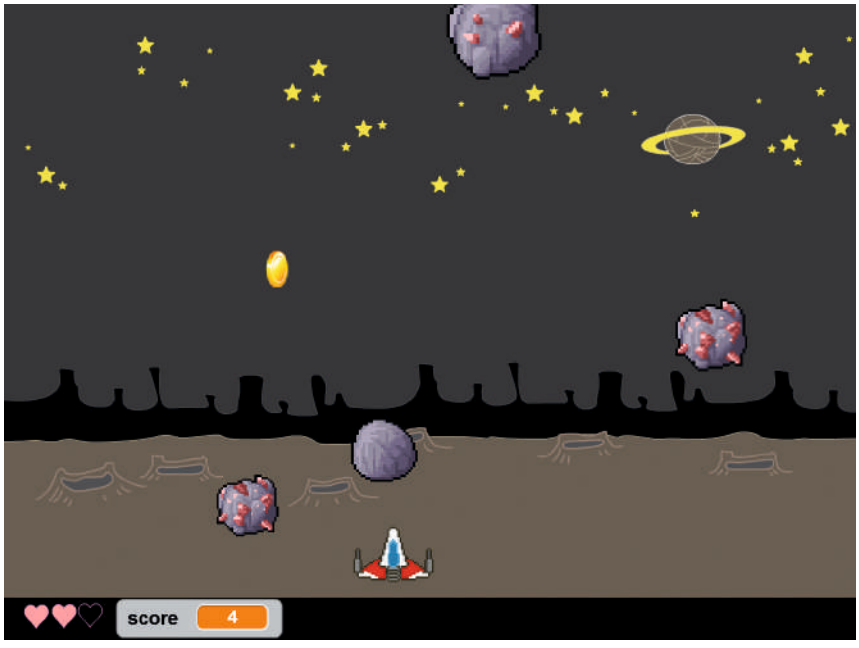
L'enseignant précise – si c'est le cas – qu'il y aura aussi des séances de travail sur l'esthétique du jeu (dessin des décors et des personnages ou objets), et que ces séances se feront en arts plastiques.

L'enseignant demande, toujours en classe entière, ce qui caractérise les jeux d'arcade : niveaux très courts, mécanismes très simples, difficulté qui augmente rapidement, impossibilité de « gagner » (il n'y a pas d'autre but à atteindre que rester « en vie » le plus longtemps possible, éventuellement en gagnant le plus de points possible).

Depuis son poste informatique, l'enseignant montre un jeu d'arcade existant sous *Scratch*, par exemple celui qu'il a lui-même réalisé à l'avance[1] et qui est l'objectif à atteindre par les élèves. Cette démonstration permet de faciliter l'étape suivante (les élèves auront plus de facilité à préciser les fonctionnalités d'un jeu qu'ils ont pu voir).
Avec des élèves déjà expérimentés en programmation, on peut préférer faire une démonstration à l'aide d'un jeu similaire mais non identique, comme celui-ci : https://scratch.mit.edu/projects/80147818 (ce dernier est un peu plus simple que le programme que nous proposons de réaliser car il ne s'agit que d'éviter des astéroïdes… nous voulons, en plus, récolter des ressources, et gérer un nombre de « vies »).

Notes pédagogiques

- Cette démonstration est très importante car elle motive les élèves (ils vont vraiment programmer un « vrai » jeu vidéo) !
- Il est important d'expliquer aux élèves que programmer un tel jeu ne se fait pas en une séance mais nécessitera plusieurs séances (typiquement, 5 ou 6 séances en fonction de leur niveau et de leurs exigences).



Capture d'écran : version finale du jeu vidéo qui sera réalisé au cours de ce projet. Il s'agit de récolter les pièces en évitant les astéroïdes. (N.B. : un programme corrigé est disponible sur le site web du projet, *cf.* page 404)

La classe vérifie bien que les mécanismes de base évoqués ci-dessus sont bien ceux présents sur notre jeu : il s'agit bien d'un jeu d'arcade.

Lister les fonctionnalités du jeu (par groupes)

Avant de se lancer dans la programmation proprement dite, il est nécessaire de lister les fonctionnalités à programmer, afin de définir un plan d'action.

1. Rappel : il est absolument indispensable que l'enseignant fasse lui-même le projet avant de le proposer aux élèves ! Il suffit de suivre les étapes décrites dans la séquence. Un enseignant débutant en *Scratch* prendra entre 2 et 3 heures pour faire l'intégralité du projet.

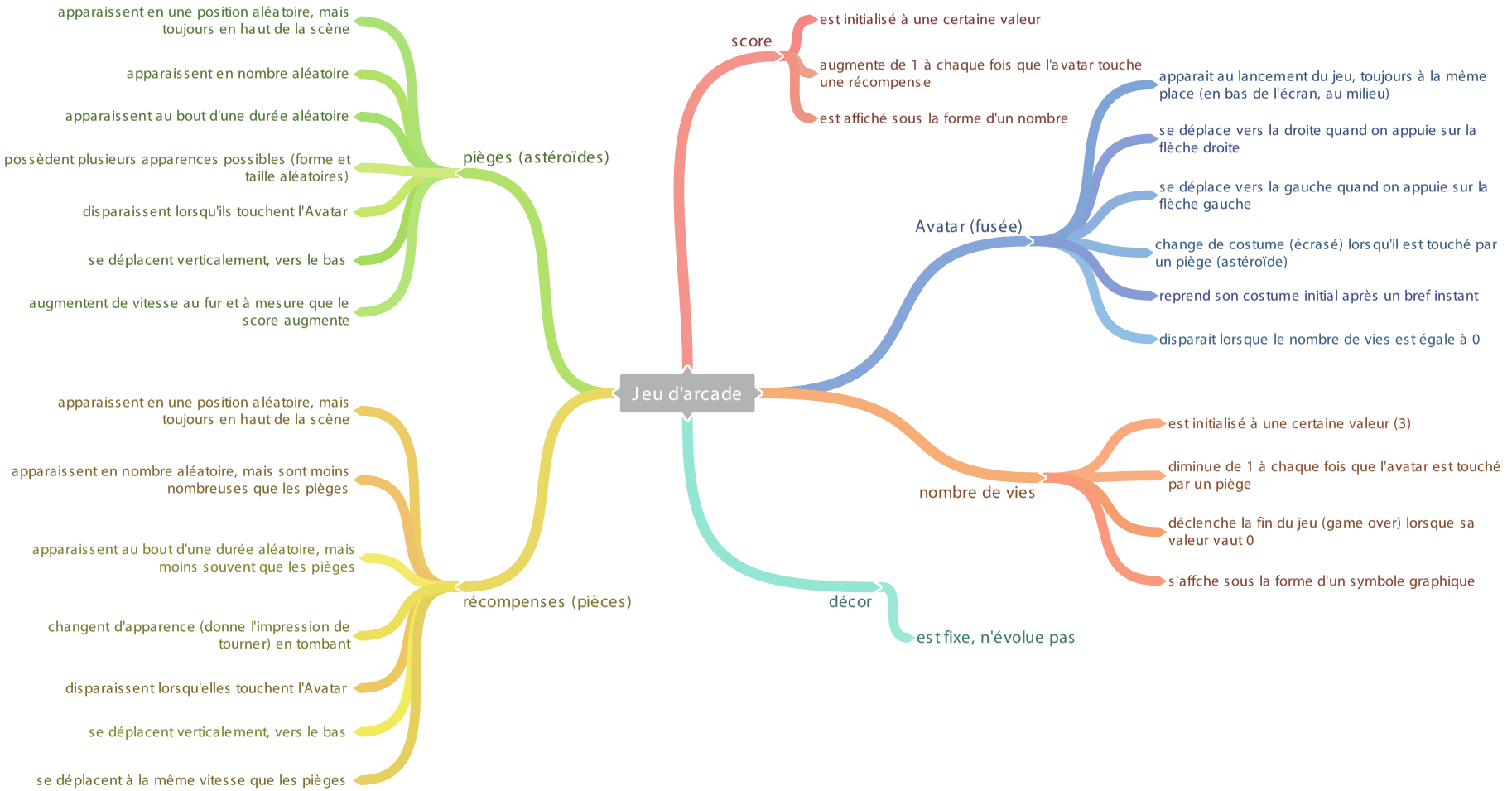
Chaque groupe (constitué de 2 ou 3 binômes) doit réfléchir à ce que va nécessiter l'accomplissement du projet. Le groupe dispose de plusieurs post-it de 2 couleurs : une couleur pour noter les éléments de décor et les personnages ou objets nécessaires (post-it « Qui ? »), une couleur pour noter ce que font ces éléments de décor et personnages (post-it « Quoi ? »).
L'enseignant donne un exemple : sur un post-it « Qui ? » il note « Avatar du joueur ». Sur un post-it « Quoi ? » il note « L'avatar se déplace à droite quand on appuie sur la flèche droite du clavier ». Il colle les deux post-it au tableau et les associe par une flèche, en précisant qu'un post-it « Qui ? » peut tout à fait être associé à plusieurs post-it « Quoi ? ».
L'enseignant invite alors les groupes à créer leurs propres post-it et à les agencer sous forme d'affiche pour lister les fonctionnalités du jeu. Le jeu d'arcade utilisé comme situation déclenchante reste accessible – et les élèves peuvent demander à mieux l'observer – pendant que les groupes créent leur affiche. Les affiches sont ensuite accrochées pour constituer une galerie. Les élèves visitent cette galerie en binômes et notent sur leur cahier de projet les nouvelles idées qu'ils y trouvent ainsi que les questions qu'ils se posent.
En cas de difficulté, l'enseignant peut guider la réflexion des élèves, jeu de démonstration à l'appui, en posant des questions telles que : où apparaissent les astéroïdes ? Que font-ils ? Que se passe-t-il lorsqu'ils touchent l'avatar en bas de l'écran ?

Synthèse : réalisation d'une carte mentale (collectivement)

Note pédagogique

Si possible, utiliser une plateforme de *mind-mapping* collaborative. En cas d'impossibilité d'accès à Internet, réaliser cette carte collectivement sous la forme d'une affiche, qui restera dans la classe (car elle guidera les prochaines séances de programmation).

L'enseignant explique que la classe va, collectivement, produire une carte mentale répertoriant les éléments de décor et les personnages nécessaires au projet, ainsi que les actions de ces personnages et éléments de décor. Depuis son poste informatique, il accède à la plateforme collaborative de *mind-mapping* et donne un exemple : il nomme la racine de la carte mentale « Jeu d'arcade » puis crée une boîte « Avatar du joueur » qui pointe vers une action – et commence par un verbe : « se déplace à droite quand on appuie sur la flèche droite ».
L'enseignant, ou un élève, continue la carte mentale, sous la dictée du reste de la classe, en reliant directement à la racine les boîtes qui concernent des éléments de décor ou personnages, et en reliant à ces dernières boîtes les actions qui les concernent.
Voici un résultat auquel la classe peut aboutir à l'aide du logiciel de *mind-mapping Coggle* :


Jeu d'arcade
score
est initialisé à une certaine valeur
augmente de 1 à chaque fois que l'avatar touche une récompense
est affiché sous la forme d'un nombre
Avatar (fusée)
apparait au lancement du jeu, toujours à la même place (en bas de l'écran, au milieu)
se déplace vers la droite quand on appuie sur la flèche droite
se déplace vers la gauche quand on appuie sur la flèche gauche
change de costume (écrasé) lorsqu'il est touché par un piège (astéroïde)
reprend son costume initial après un bref instant
disparait lorsque le nombre de vies est égale à 0
nombre de vies
est initialisé à une certaine valeur (3)
diminue de 1 à chaque fois que l'avatar est touché par un piège
déclenche la fin du jeu (game over) lorsque sa valeur vaut 0
s'affche sous la forme d'un symbole graphique
décor
est fixe, n'évolue pas
pièges (astéroïdes)
apparaissent en une position aléatoire, mais toujours en haut de la scène
apparaissent en nombre aléatoire
apparaissent au bout d'une durée aléatoire
possèdent plusieurs apparences possibles (forme et taille aléatoires)
disparaissent lorsqu'ils touchent l'Avatar
se déplacent verticalement, vers le bas
augmentent de vitesse au fur et à mesure que le score augmente
récompenses (pièces)
apparaissent en une position aléatoire, mais toujours en haut de la scène
apparaissent en nombre aléatoire, mais sont moins nombreuses que les pièges
apparaissent au bout d'une durée aléatoire, mais moins souvent que les pièges
changent d'apparence (donne l'impression de tourner) en tombant
disparaissent lorsqu'elles touchent l'Avatar
se déplacent verticalement, vers le bas
se déplacent à la même vitesse que les pièges

Séance 3 – Premier programme : piloter l'avatar (utilisation de structures de contrôle SI... ALORS)

Discipline dominante	Mathématiques ou technologie
Résumé	Les élèves créent leur premier programme permettant de piloter l'avatar du joueur. Ils apprennent également à personnaliser la scène dans *Scratch* (lutin et arrière-plan), ainsi qu'à enregistrer leur travail pour le réutiliser plus tard.
Notions nouvelles *(cf. scénario conceptuel, page 81)*	Algorithmes • Un algorithme peut contenir des instructions, des boucles, des tests, des variables. • Une boucle permet de répéter plusieurs fois la même action. • Certaines boucles, dites « infinies », ne s'arrêtent jamais. • Certaines boucles, dites « itératives », sont répétées un nombre prédéfini de fois. • Un test permet de choisir quelle action effectuer si une condition est vérifiée ou non. • Une condition est une expression qui est soit vraie, soit fausse. Langages • En *Scratch*, la programmation est « événementielle » : des événements déclenchent l'exécution de séquences d'instructions. Bonnes habitudes de programmation • Partager un programme avec d'autres personnes permet de l'améliorer.
Matériel	Pour chaque binôme • Un ordinateur possédant une connexion Internet (pour utiliser la version en ligne de *Scratch*) ou ayant *Scratch* préinstallé • Un fichier *Scratch* accessible aux élèves (accessible sur un répertoire réseau par exemple) : Arcade_V01 • Un cahier de projet Pour l'enseignant • Un ordinateur possédant une connexion Internet (pour utiliser la version en ligne de *Scratch*) ou ayant *Scratch* préinstallé • Un vidéoprojecteur

Notes pédagogiques

• Tous les fichiers utiles pour ce projet sont téléchargeables sur le site du projet[2]. L'enseignant doit charger ces fichiers sur son compte *Scratch* pour les utiliser lors de démonstrations ou lors des mises en commun. Les fichiers destinés aux élèves doivent être placés dans un répertoire accessible à tous (sur le réseau du collège), au fur et à mesure des besoins.

• L'objectif de ce guide pédagogique est l'apprentissage de l'informatique (ici, plus particulièrement, la programmation) : nous ne décrivons donc pas les éventuelles séances d'arts plastiques permettant de dessiner les différents éléments graphiques du jeu (scène, personnages et objets).

• Nous conseillons de former des binômes relativement homogènes plutôt que des binômes associant un élève en difficulté et un élève plus avancé dans la programmation (car dans ce cas, l'expérience montre que l'élève en difficulté est passif et laisse faire l'autre).

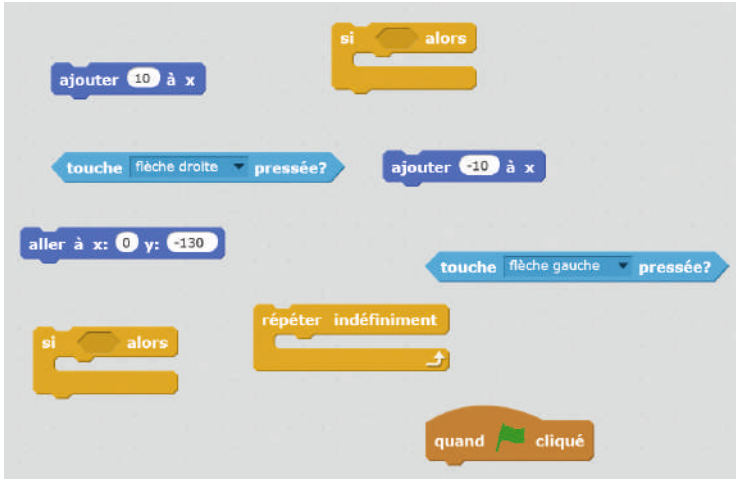
2. *Cf.* page 404.

L'enseignant annonce que dès aujourd'hui, les groupes vont commencer à programmer leur jeu d'arcade : il s'agira d'apprendre à piloter l'avatar et de définir les éléments graphiques (arrière-plan et avatar du joueur).

Tâche 1 : créer et enregistrer son fichier (5 minutes)

Les élèves ouvrent (ou importent, s'ils utilisent *Scratch* en ligne) le fichier Arcade_V01 mis à disposition sur le répertoire partagé et utilisent la commande « enregistrer sous » (dans le menu « fichier »), en ajoutant, au nom actuel du fichier, le nom de leur binôme.

Tâche 2 : piloter l'avatar (15 minutes)

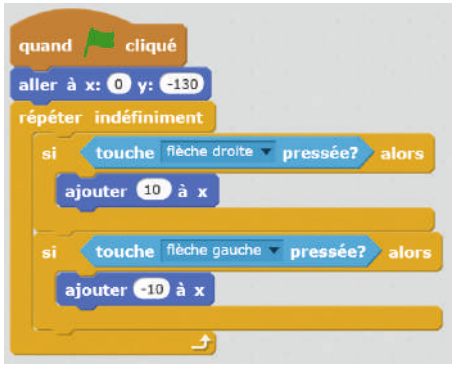


Instructions fournies « en vrac » dans le programme Arcade_V01.

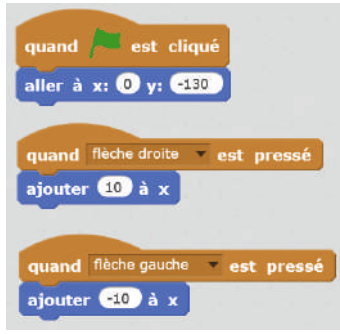
Notes pédagogiques

• Pour ce premier travail de programmation, nous proposons de guider les élèves en mettant à leur disposition des instructions utiles. Ils doivent les combiner entre elles de façon que le programme fasse ce que l'on souhaite, à savoir :
– au lancement du programme (drapeau vert), le chat se place en bas de l'écran, au centre.
– Chaque fois que l'on clique sur la flèche droite (ou gauche) du clavier, il se déplace à droite (ou à gauche).
• Des élèves ayant déjà utilisé *Scratch* n'auront pas besoin d'être guidés de la sorte, et pourront directement programmer cette fonctionnalité, qui est très simple.
• En cas de besoin, l'enseignant peut revenir sur la manière de repérer la position d'un point sur un plan, à l'aide de 2 axes (abscisses, ordonnées), et préciser que, par convention, « x » désigne l'abscisse et « y » l'ordonnée du lutin. La fiche documentaire distribuée lors de la séance de prise en main de *Scratch* peut être utile !

Voici deux solutions possibles :



ou



Notes scientifiques

• Il y a 2 manières, dans *Scratch*, de déclencher une action si une touche est pressée :
– celle utilisée ci-dessous (une boucle « répéter indéfiniment » comportant un test « si touche… pressée »)
– ou l'instruction « Quand… est cliqué » disponible dans l'onglet « événement ».

• En théorie, les 2 méthodes sont équivalentes… pourtant, nous avons remarqué que la seconde donnait lieu à des mouvements plus saccadés, c'est pourquoi nous préférons la première, que nous appliquerons par la suite.
• Il peut arriver que le lutin se retrouve « la tête en bas » lorsqu'on lui demande de changer de direction (cela dépend des lutins !). Dans ce cas, on peut changer le « style de rotation » en cliquant sur le « i » bleu qui apparait quand on sélectionne le lutin. Cela ouvre la fenêtre suivante :

Sélectionner le style de rotation ⟷ permet d'éviter que le lutin se retrouve la tête en bas !

L'enseignant invite les élèves à enregistrer leur programme, à le partager en cliquant sur l'icône « Partager » (uniquement pour l'utilisation en ligne) et à noter son numéro sur leur cahier de projet.
Lors de la mise en commun, l'enseignant sélectionne un groupe d'élèves. Ce groupe vient au tableau et se rend sur la page de son programme pour présenter sa démarche. Cette présentation sert de base de discussion à l'ensemble de la classe, et l'enseignant veille à ce que les points suivants soient abordés :

• Les instructions 

ou 

permettent de déclencher les instructions qui y sont accolées (et qui forment une séquence d'instructions). Il y a d'autres instructions de ce type dans la catégorie « Événements ».
• L'instruction « aller à x : 0 y : -130 » est une initialisation : elle impose un état initial.
Les instructions de la catégorie « Contrôle » que les élèves viennent d'utiliser sont une instruction conditionnelle (« Si… alors… ») et une boucle infinie « Répéter indéfiniment ».
• Les instructions conditionnelles permettent de faire en sorte que certaines actions ne soient effectuées que lorsqu'une certaine condition est vérifiée. Ici, par exemple, c'est seulement si le joueur appuie sur la flèche droite que le lutin se déplace vers la droite.
• Les boucles permettent de répéter plusieurs fois une même séquence d'instructions.

Tâche 3 : personnaliser le lutin (10 minutes)

Cette tâche peut être réalisée de 2 façons différentes, au choix de l'enseignant :

• De façon astucieuse (avant de supprimer le chat, on importe un autre lutin, notre avatar, et on copie le programme réalisé auparavant)

• De façon volontairement inefficace : on supprime d'abord le chat, puis on importe un nouveau lutin qui sera notre avatar. Cependant, puisqu'on a supprimé le chat, on a également perdu le programme réalisé auparavant : il va falloir le ré-écrire (mais sans modèle, cette fois !). Cela peut constituer une bonne évaluation formative. C'est cette façon qui est décrite ci-dessous.

L'enseignant explique qu'il est possible de supprimer le lutin « chat » et d'en créer un autre à la place, plus en phase avec le thème du jeu. Nous proposons, dans les illustrations qui suivent, un thème « spatial » (dans le jeu, un vaisseau doit éviter des astéroïdes), mais l'on pourrait imaginer bien d'autres thèmes. Par exemple :

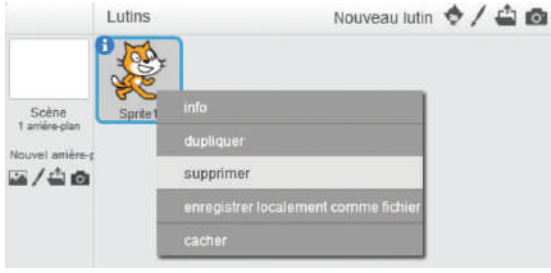
• un maraîcher qui doit récolter dans son panier des fruits qui tombent, en évitant les fruits pourris ou des grêlons ;

• un passant qui doit traverser une route en évitant les voitures ;

• une souris qui doit récolter des morceaux de fromage en évitant les chats…

Les élèves doivent donc prendre quelques minutes pour réfléchir au thème de leur jeu (en revanche, la mécanique du jeu, elle, reste la même pour tout le monde : elle correspond à la carte mentale réalisée précédemment).

• Pour **supprimer le chat**, il faut cliquer (bouton droit) sur son icone, dans la zone des lutins, et choisir « supprimer ».



• Il y a 4 façons différentes de créer un nouveau lutin, accessibles depuis la barre d'outils « nouveau lutin » en bas à droite de la scène (nous mettons en gras la méthode que nous préconisons ici).

	***Choisir un lutin dans la bibliothèque*** *Scratch* est livré avec une centaine de lutins prédéfinis dans la bibliothèque. Ces lutins peuvent être pratiques pour un projet d'élève, mais sont de styles très hétérogènes. On y trouve des personnages, des animaux réels ou imaginaires, des objets, etc.
	***Dessiner un nouveau lutin*** *Scratch* possède un outil intégré de dessin permettant aux élèves de créer « à la main » leur propre lutin. A ne choisir que pour des dessins très simples (les dessins plus élaborés devront être réalisés dans un logiciel spécialisé, puis importés grâce à la fonctionnalité ci-dessous).
	***Importer le lutin depuis un fichier*** C'est l'option qui sera choisie si les élèves ont créé un avatar dans un autre logiciel, ou s'ils veulent utiliser une image téléchargée sur le Web, ou encore s'ils veulent utiliser celle que nous mettons à leur disposition (le vaisseau spatial utilisé dans nos exemples).
	***Nouveau lutin depuis une webcam*** Cet outil peut être très pratique pour des projets personnels (on peut ajouter son propre visage comme nouveau lutin) mais n'a pas d'intérêt dans le cadre d'un jeu comme celui-ci.

Notes pédagogiques

• Le site http://opengameart.org propose de très nombreux graphismes adaptés aux jeux vidéo (décors, personnages, objets…), tous libres de droits : une vraie mine d'or !

• Nous avons remarqué que, sur certaines machines, l'importation du lutin depuis un fichier fonctionne mal. Si l'importation échoue, il existe une façon très simple de remédier au problème : sauvegarder le travail en cours, quitter *Scratch*, relancer *Scratch*… et recommencer l'import du fichier. Après cette petite manœuvre, ça marche !

Tâche 4 : changer l'arrière-plan (5 minutes)

De la même façon que précédemment, il est possible de changer l'arrière-plan de la scène, soit à partir d'une image issue de la bibliothèque, soit à partir d'un fichier fourni par l'utilisateur, soit encore en le dessinant soi-même.

Les élèves choisissent l'arrière-plan qu'ils préfèrent en fonction du thème de leur jeu (celui utilisé dans les captures d'écran est disponible dans la bibliothèque de *Scratch* et s'intitule « space ».).

Conclusion

Outre les éléments de conclusion lors des mises en commun, l'enseignant guide les élèves vers la formulation de bonnes habitudes de programmation (*cf.* scénario conceptuel, page 81) :

• avant de se lancer dans un projet de programmation, il est essentiel de lister les fonctionnalités que l'on souhaite programmer ;

• l'environnement de programmation *Scratch* en ligne permet de prendre connaissance d'un programme d'une autre personne, de le remixer et de partager ses propres programmes. Le partage avec d'autres est une bonne habitude de programmation.

Séance 4 – Programmer la chute des pièges : utilisation de variables

Discipline dominante	Mathématiques ou technologie
Résumé	Les élèves utilisent des variables pour programmer la chute des pièges (vitesse de chute, et nombre de vies restantes). Ils réinvestissent également des structures conditionnelles SI… ALORS vues précédemment.
Notions nouvelles *(cf. scénario conceptuel, page 81)*	Machines : • Une variable est un espace mémoire dans lequel on peut stocker une valeur (c'est l'affectation). On peut utiliser cette valeur dans des tests ou des calculs, et la modifier à tout moment. Langages : • Certaines instructions s'exécutent simultanément à d'autres : on parle de « programmation parallèle ». Bonnes habitudes de programmation : • Il est préférable d'utiliser des variables plutôt que de laisser des valeurs numériques dans un programme. • Les variables doivent avoir un nom explicite. • Il est préférable d'initialiser une variable dès sa création. • Introduire des pauses dans l'exécution d'un programme facilite la résolution des bugs.
Matériel	Identique à la séance précédente.

Notes pédagogiques

• Cette séance permet aux élèves de se familiariser avec une notion très importante : les variables (que l'on retrouve dans tous les langages de programmation).

• Pour cette séance, l'enseignant a procédé au préalable au découpage en tâches simples des fonctionnalités à programmer (*cf.* ci-dessous), et propose directement une feuille de route. Lors de séances ultérieures, les élèves auront davantage d'expérience et pourront effectuer par eux-mêmes le découpage en tâches simples, d'abord collectivement, puis par binômes.

• À partir de maintenant, il est illusoire de chercher à ce que tous les élèves avancent au même rythme (sinon, les groupes ayant le plus de facilité vont très vite s'ennuyer et se dissiper). Le découpage en tâches simples permet de faciliter la gestion de classe par l'enseignant : chacun, où qu'il en soit, a quelque chose à faire.

La classe reprend la carte mentale élaborée lors de la Séance 2 et s'intéresse à un des mécanismes essentiel du jeu : la chute des astéroïdes (ou autres types de piège selon le thème choisi par les élèves). Nous conseillons le cheminement suivant (idéalement, à discuter avec les élèves) :

Difficulté (cf. page 69)	Nom de la tâche	Travail à effectuer
	**Tâche 1 :** faire chuter 1 astéroïde depuis une position définie	• Créer le lutin • Le positionner en haut de l'écran • Programmer sa chute vers le bas de l'écran
	**Tâche 2 :** faire chuter 1 astéroïde depuis une position aléatoire	• Faire en sorte que l'astéroïde ne parte jamais du même endroit lorsqu'on relance le programme
	**Tâche 3 :** faire perdre une vie à l'avatar lorsqu'il touche un astéroïde	• Créer une variable indiquant le nombre de vies restantes • Lui donner une valeur initiale • La diminuer de 1 à chaque fois que l'avatar est touché par un astéroïde
	**Tâche 4 :** utiliser des clones pour faire chuter plusieurs astéroïdes	• Créer plusieurs clones de l'astéroïde • Faire en sorte que ces clones tombent avec des temps d'attente aléatoires • Faire en sorte que ces clones soient en nombre aléatoire • Faire disparaître les clones quand ils arrivent en bas de l'écran

N.B. : la tâche 3, ainsi que le début de la tâche 4 sont à faire collectivement. La tâche 4 est décrite dans la séance suivante (inutile d'introduire trop de nouveautés dans une seule séance !)

 Tâche 1 : faire chuter 1 astéroïde depuis une position définie (10 minutes)

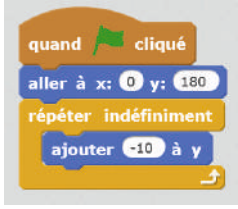
Cette étape ne présente pas de difficulté particulière, puisque les élèves ont déjà utilisé les fonctionnalités qui permettent de la programmer.

• Il faut dans un premier temps créer un nouveau lutin pour l'astéroïde (comme précédemment, cela peut se faire en dessinant ce lutin, ou en important une image prise dans la bibliothèque *Scratch* ou dans un fichier quelconque).

• Le positionnement de ce lutin en haut de l'écran se fait à l'aide de la commande « aller à x… y… » (dans l'onglet « Mouvement », en bleu)

• La chute de l'astéroïde vers le bas de l'écran se fait en modifiant la valeur de son ordonnée.

Un programme possible est donc :



N.B. : ce programme sera modifié plusieurs fois par la suite

L'enseignant organise une rapide mise en commun pour s'assurer que cette tâche a été bien remplie, car les élèves sont encore peu expérimentés.

Il fait notamment remarquer que, dans *Scratch*, chaque lutin possède son propre programme. Les programmes des différents lutins s'exécutent en parallèle, étant chacun déclenchés par des événements

(pour l'instant, le seul événement que nous avons rencontré est le clic sur le drapeau vert… mais nous en verrons d'autres bientôt !).

Tâche 2 : faire chuter 1 astéroïde depuis une position aléatoire (15 minutes)

Revenant sur la carte mentale élaborée précédemment, l'enseignant rappelle que l'astéroïde est censé partir d'une position aléatoire (mais toujours en haut de l'écran). Il demande aux élèves d'explorer les différents onglets de *Scratch* afin de chercher quelle instruction peut être utilisée pour créer un nombre aléatoire, puis comment modifier le programme pour en tenir compte.
La solution se trouve dans l'onglet vert « opérateurs ». Le programme est facilement modifié :

Tâche 3 : faire perdre une vie à l'avatar lorsqu'il touche un astéroïde (20 minutes)

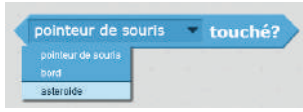
Puisque les élèves n'ont jamais manipulé de variables, il peut être utile de faire cette étape collectivement. L'enseignant rappelle aux élèves que l'avatar ne dispose que d'un nombre limité de « vies » (par exemple, 3, en début de partie). Il faut que ce nombre soit géré par une « variable » : une variable est simplement une « boîte » située dans la mémoire de l'ordinateur, boîte qui peut contenir une valeur (cette valeur peut, bien sûr, changer).
Il montre aux élèves comment créer une variable (cliquer sur « créer une variable » depuis l'onglet « données », orange).

Notes scientifiques et pédagogiques

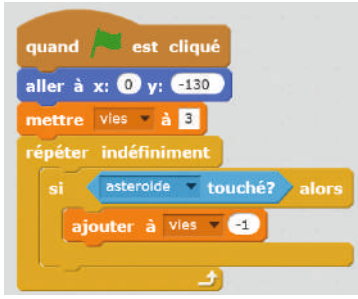
• La variable ainsi créée peut être accessible par un seul lutin (celui dans le programme duquel elle est créée) ou par tous. On parle respectivement de variable locale ou de variable globale dans d'autres langages de programmation. Puisqu'on ne sait pas encore si d'autres lutins auront besoin d'accéder à cette variable, le plus sage est de la rendre accessible à tous les lutins.
• Pour qu'un programme soit facile à comprendre, il est important de donner des noms explicites aux variables que l'on crée. Cette bonne habitude limite également les bugs de programmation. Le nom de la variable peut donc être, tout simplement : « vies ».

• Plusieurs variables seront nécessaires à notre programme (le nombre de vies, le score, la vitesse de chute des pièges et récompenses…). Selon les préférences de chacun, il peut paraître plus logique de regrouper ces variables dans un même programme, considéré comme le programme principal (celui de l'avatar, par exemple), ou au contraire de les créer et initialiser dans les programmes concernés. Ici, la variable « vies » concerne davantage le lutin « vaisseau »… il peut donc sembler logique de la créer et de l'initialiser dans le programme du lutin « vaisseau ».
• On remarque que lorsque la variable est créée, elle est affichée à l'écran, ainsi que sa valeur. Pour faire disparaître cet affichage, il suffit de décocher la case à gauche du nom de la variable, dans la palette « données ».
• L'enseignant fait remarquer aux élèves qu'ils ont déjà manipulé des variables dans les séances précédentes (l'abscisse X et l'ordonnée Y, qui donnent la position d'un lutin à l'écran). Ces variables étaient déjà disponibles et les élèves ont pu les manipuler (faire des tests, leur affecter des valeurs…) sans avoir besoin de les créer.

Les élèves créent cette variable, qu'ils nomment par exemple « vies » et qu'ils initialisent (par exemple à 3). Ils réutilisent la structure de contrôle SI… ALORS déjà vue à la Séance 3 (disponible dans l'onglet jaune « contrôle »), ainsi qu'un « capteur » (onglet bleu clair) indiquant si les lutins « asteroide » et « vaisseau » se touchent. Si l'on ajoute cette fonctionnalité dans le programme du vaisseau, alors le capteur prend cette forme.



Le programme principal du vaisseau devient alors :



Solution provisoire trouvée pour la gestion du nombre de vies.

Notes pédagogiques

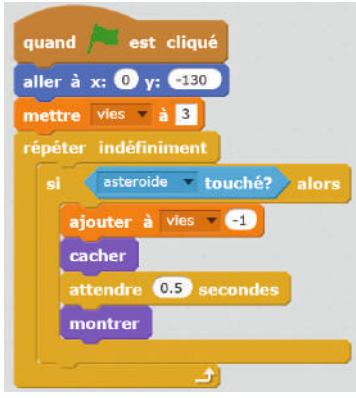
• On commence à voir ici qu'il est utile de nommer correctement, non seulement les variables, mais aussi les lutins. Lorsque les lutins se multiplient, cela devient un impératif. L'instruction « si astéroïde touché » est beaucoup plus explicite que « si lutin2 touché » !
• Pour renommer un lutin, il faut le sélectionner, puis cliquer sur le « i » qui apparait en bleu sur son icône.
• Ici, on a choisi de gérer la collision dans le programme du vaisseau. C'est ce qui semble le plus logique, à cet instant (car il s'agit de gérer le nombre de vies du vaisseau). Nous verrons plus tard (tâche 2 de la Séance 7, page 107) que, si l'on souhaite améliorer le rendu graphique du jeu (changer l'apparence du vaisseau lors de l'impact et faire disparaitre l'astéroïde), la seule solution consiste à placer ce test dans le programme de l'astéroïde. Nous conseillons au professeur de ne pas imposer, dès maintenant, ce qui s'avère être la bonne solution *in fine*, mais de suivre la logique des élèves, quitte à modifier le programme ensuite.

On s'aperçoit très vite d'un problème : lorsque les 2 lutins « astéroïdes » et « vaisseau » se touchent, ils le font pendant une certaine durée, qui n'est pas un simple instant. Pendant tout ce temps, la variable « vie » est diminuée. On perd donc de nombreuses vies à chaque impact, ce qui ne correspond pas à ce que l'on souhaite.

Pour identifier la cause de ce problème, on peut par exemple introduire une pause d'1 seconde à l'intérieur de la boucle « répéter indéfiniment ». Ainsi, on peut plus facilement suivre l'exécution du programme « pas à pas » et voir comment évolue la valeur de la variable. D'une façon générale, **introduire des pauses dans un programme est une bonne habitude à prendre pour le débugger**.

Pour résoudre ce problème, il suffit de rompre le contact entre les 2 lutins. Par exemple, faire disparaître le vaisseau pendant ½ seconde en utilisant l'instruction « cacher » (onglet « apparence », en violet). Attention, puisqu'on lui demande de se cacher à un moment donné, il faut, dans ce cas, l'obliger à se montrer au lancement du programme (sinon, il se cachera une fois, puis restera caché en permanence).

Le programme du vaisseau devient donc :



Amélioration de la gestion du nombre de vies. En cachant le vaisseau lors du contact, on empêche le nombre de vies de diminuer de plus de une unité. Le vaisseau réapparaît ensuite (une fois que l'astéroïde a passé son chemin).

Note pédagogique

Ici, le vaisseau disparaît pendant un instant, lors d'un impact. Le programme sera amélioré plus tard (Séance 7) : l'utilisation de costumes et de messages permettra de changer l'apparence du vaisseau plutôt que de le faire disparaître ; c'est l'astéroïde qui disparaîtra lors de l'impact.

Mise en commun

La mise en commun sert à échanger sur les principales difficultés rencontrées par les groupes, et inclut un temps pendant lequel les élèves peuvent modifier leur programme à la lumière des différents échanges. Les élèves envisagent également les fonctionnalités qui restent à programmer.

Conclusion

L'enseignant guide les élèves vers la formulation d'une conclusion :

• n'importe quelle tâche complexe, que l'on peut résoudre automatiquement, peut se décomposer en un ensemble de tâches simples.

• les tâches simples que nous avons programmées aujourd'hui comportaient :

– des affectations de variables : une variable est un espace mémoire dans lequel on peut stocker une valeur (c'est l'affectation). On peut utiliser cette valeur dans des tests ou des calculs ultérieurs, et la modifier à tout moment ;

– des boucles : ces instructions permettent de répéter plusieurs fois un bloc d'instructions ;
– des tests, qui permettent de vérifier si une condition est vraie ou non.

• Ces instructions étaient organisées en séquences d'instructions, déclenchées chacune par un événement (clic sur le drapeau vert).

Par ailleurs, les élèves notent les bonnes habitudes de programmation évoquées lors de cette séance dans leur cahier de projet.

Séance 5 – Programmer la chute des pièges (suite) : utilisation de clones

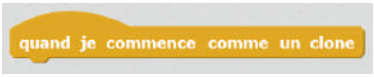
Discipline dominante	Mathématiques ou technologie
Résumé	Les élèves utilisent une fonctionnalité propre à *Scratch* permettant de cloner des lutins pour programmer la chute d'un nombre aléatoire de pièges.
Notions nouvelles *(cf. scénario conceptuel, page 81)*	Algorithmes : • Certaines boucles, dites « conditionnelles », sont répétées jusqu'à ce qu'une condition soit remplie.
Matériel	Identique à la séance précédente. En plus, pour chaque binôme (facultatif : uniquement pour les élèves en difficulté) : • Un fichier *Scratch* accessible aux élèves : Arcade_V02 (fichier disponible sur le site web du projet)

L'enseignant, après avoir fait le point sur l'avancement des différents groupes, rappelle que le jeu nécessite d'avoir plusieurs astéroïdes, et non pas un seul. Il demande aux élèves comment faire. Certains peuvent proposer de créer autant de lutins que d'astéroïdes possibles. C'est une solution, mais elle est peu élégante :
• D'une part, cela peut devenir très fastidieux si le nombre d'astéroïdes devient grand (on ne va pas créer 50 lutins identiques !)
• D'autre part, cela oblige à dupliquer le même programme de nombreuses fois. Si l'on souhaite modifier ces programmes (pour corriger un bug ou ajouter une fonctionnalité), il faudra le faire partout, ce qui risque d'être pénible et sera assurément source d'erreurs.
L'enseignant explique que *Scratch* possède une fonctionnalité qui peut être très utile dans ce cas : on peut « cloner » les lutins. Un lutin peut exister en un nombre de clones aussi grand qu'on le souhaite, chacun étant piloté par le même programme.
Pour une meilleure lisibilité, on peut découper la tâche 4 en différentes « sous-tâches » (que l'on va nommer 4.1, 4.2, etc.).

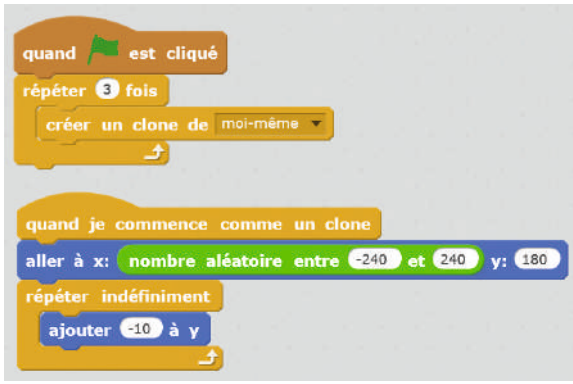
Difficulté (cf. page 69)	Nom de la tâche
	**Tâche 4.1 :** créer 3 clones de l'astéroïde
	**Tâche 4.2 :** faire disparaître les clones quand ils arrivent en bas de l'écran
	**Tâche 4.3 :** créer un nombre aléatoire de clones
	**Tâche 4.4 :** faire en sorte que ces clones tombent avec des temps d'attente aléatoires

Tâche 4.1 : créer plusieurs clones de l'astéroïde (20 minutes)

La création d'un clone se fait via la commande 

disponible dans l'onglet « contrôle ».
Cette instruction peut être exécutée autant de fois que nécessaire (dans une boucle par exemple), ce qui peut donc donner lieu à un nombre de clones aussi grand que souhaité.
Outre la création de clones, on souhaite que chacun se comporte comme l'astéroïde unique programmée à la séance précédente : position initiale aléatoire, déplacement vers le bas.
Pour déclencher toute cette séquence d'instructions dès qu'un clone est créé, il faut utiliser l'événement 

(également situé dans l'onglet « contrôle »).
Une façon possible de programmer la création de 3 astéroïdes, avec le comportement souhaité pour chacun est par exemple :



Notes pédagogiques

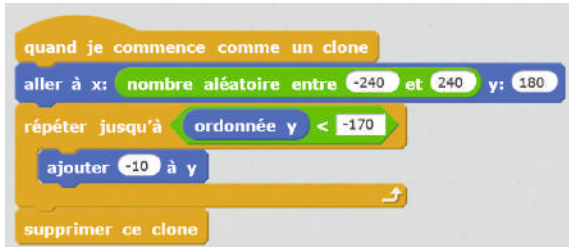
- On remarque, sur la capture d'écran ci-dessus, que notre programme est désormais composé de plusieurs « branches » (qu'on appellera « sous-programmes »). Chaque sous-programme est déclenché par un événement particulier (ici, le clic sur le drapeau vert ou la création d'un clone).
- Avec certains élèves en difficulté, on peut procéder comme lors de la Séance 3 : leur donner un programme qui contient déjà toutes les instructions, dans le désordre. C'est l'objet du programme Arcade_V02

Tâche 4.2 : faire disparaître les clones quand ils arrivent en bas de l'écran (10 minutes)

Cette tâche ne présente en général pas de difficulté. Si certains élèves ont du mal à manipuler les abscisses et ordonnées, ne pas hésiter à faire une activité débranchée. Des questions simples comme *A quel mouvement correspond l'instruction « ajouter 10 à x », « ajouter -10 à y » ?* ou *Quelles sont les coordonnées d'un lutin lorsqu'il est sur les bords de l'écran ?* les aideront à se familiariser avec ces notions.
On peut imaginer 2 méthodes différentes pour la réaliser la tâche souhaitée (faire disparaître les clones quand ils arrivent en bas de l'écran) :
- A chaque étape de la boucle « répéter indéfiniment » régissant la descente de l'astéroïde, on peut insérer un test : SI (y < -180) ALORS (supprimer ce clone) ;
- On peut aussi remplacer cette boucle indéfiniment par une boucle « répéter jusqu'à (y < -170) » et mettre l'instruction « supprimer ce clone » après la boucle.

Ces 2 possibilités sont illustrées ici :

ou



Note scientifique

Plutôt que de supprimer le clone, on pourrait choisir de simplement le cacher (instruction « masquer »). Mais pourquoi encombrer la mémoire de l'ordinateur par des objets qui ne servent plus à rien ?

Tâche 4.3 : créer un nombre aléatoire de clones (10 minutes)

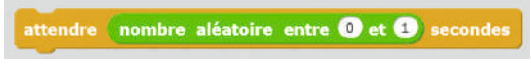
Aucune difficulté non plus pour cette tâche : il suffit de remplacer le nombre « 3 » par un nombre aléatoire (compris par exemple entre 1 et 10) dans la boucle du sous-programme déclenchant la création des clones. Ce sous-programme devient :

Tâche 4.4 : faire en sorte que ces clones tombent avec des temps d'attente aléatoires (10 minutes)

La tâche 4.3 crée un nombre aléatoire de clones, mais une seule fois : dès que ces clones disparaissent en bas de l'écran, il ne se passe plus rien. Si l'on souhaite que des clones apparaissent en permanence, de façon aléatoire, il faut modifier le programme précédent :

• Remplacer la boucle « répéter (un nombre aléatoire de fois) » par une boucle « répéter indéfiniment » ;

• Introduire une pause d'une durée aléatoire à l'intérieur de la boucle. Pour cela, une petite astuce est nécessaire. Imaginons que l'on souhaite une pause aléatoire durant entre 0 et 1 seconde.

– L'instruction 

semble répondre à notre besoin. En réalité, le résultat obtenu est une attente aléatoire dont la durée vaut soit 0, soit 1 seconde (mais rien entre ces 2 valeurs).

– Pour faire une attente aléatoire entre 0 et 1 seconde, on peut par exemple tirer un nombre entre 1 et 10, et diviser le résultat par 10 (ainsi, on tirera au hasard des valeurs parmi {0 ; 0,1 ; 0,2 ; 0,3… 0,9 ; 1}.

Le sous-programme devient donc, par exemple :

Conclusion

L'enseignant guide les élèves vers la formulation d'une conclusion : en *Scratch*, il est possible de dupliquer un lutin en autant de clones que souhaité. Chacun est « piloté » à l'aide du même programme.

Séance 6 – Programmer la chute des récompenses : révision générale

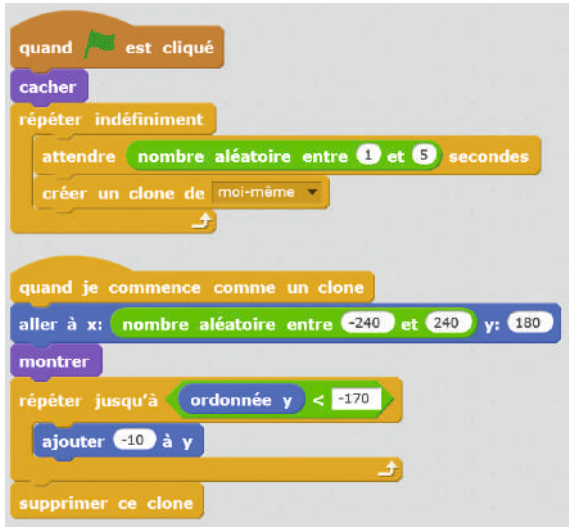
Discipline dominante	Mathématiques ou technologie
Résumé	Programmer la chute des récompenses permet aux élèves de revenir sur les variables, les clones, les boucles, les tests et les nombres aléatoires. Les élèves découvrent également la possibilité de commenter leur programme pour en faciliter la lecture.
Notions nouvelles *(cf. scénario conceptuel, page 81)*	**Bonnes habitudes de programmation :** • Placer des commentaires dans un programme en facilite la lecture et le partage avec autrui. • Un programme bien écrit, avec des variables et des fonctions bien nommées, ne nécessite pas beaucoup de commentaires.
Matériel	Identique à la séance précédente.

La classe reprend la carte mentale élaborée lors de la Séance 2 et s'intéresse à un autre mécanisme essentiel du jeu : la chute des pièces (ou autres types de récompenses selon le thème choisi). Les élèves remarquent que les récompenses se comportent à peu près comme les pièges, et peuvent donc facilement proposer un découpage en tâches « simples » pour cette fonctionnalité du jeu (puisqu'il s'agit d'un réinvestissement de notions connues, le problème peut être découpé moins finement que précédemment).
Voici par exemple un découpage en 3 étapes :

Difficulté (cf. page 69)	Nom de la tâche	Travail à effectuer
	**Tâche 1 :** faire chuter un nombre aléatoire de récompenses	• Créer le lutin • Attendre une durée aléatoire avant de le cloner • Pour chaque clone : le faire chuter vers le bas de l'écran (à la même vitesse que les astéroïdes)
	**Tâche 2 :** augmenter le score chaque fois que l'avatar touche une récompense	• Créer une variable indiquant le score • Lui donner une valeur initiale • L'augmenter de 1 lorsque l'avatar touche une récompense
	**Tâche 3 :** augmenter la vitesse de chute à chaque fois que le score augmente	• Créer une variable « vitesse_chute » • L'initialiser • Remplacer la valeur numérique de la vitesse des astéroïdes et des récompenses par cette variable • Augmenter la valeur de la vitesse à chaque fois que le score augmente
	**Tâche 4 :** commenter son programme	• Insérer des commentaires facilitant la lecture du programme

Tâche 1 : faire chuter un nombre aléatoire de récompenses (20 minutes)

Les élèves reproduisent en autonomie les différentes étapes des deux séances précédentes, appliquées cette-fois au nouveau lutin « récompense », en commençant par la création du lutin, jusqu'à la programmation de la chute de ses clones.
Le programme ressemble à la capture d'écran ci-contre.
L'enseignant organise une mise en commun afin de s'assurer que ces différentes notions (boucles, tests, clones, nombres aléatoires…) sont bien acquises par tout le monde.



Programme de la récompense (pièce)

Notes scientifiques

- Ici, on a fait le choix d'utiliser la même vitesse pour les astéroïdes et les récompenses.
- De même, on a fait un autre choix : faire apparaître les récompenses moins souvent que les pièges : cela n'a rien d'obligatoire !
- On utilise les commandes « cacher » et « montrer » pour corriger un dysfonctionnement : lorsqu'on supprime un clone, cela marche pour tous les clones, mais pas pour le lutin d'origine, qui reste à l'écran. Ici, on décide de cacher ce lutin, et de ne montrer que ses clones.

Tâche 2 : augmenter le score chaque fois que l'avatar touche une récompense (10 minutes)

Cette tâche reprend également des notions déjà vues auparavant, puisqu'il s'agit de créer, d'initialiser et d'utiliser une nouvelle variable : le score. La création et l'initialisation peuvent se faire dans le lutin de l'avatar (le vaisseau), tandis que la modification de cette variable peut se faire dans le lutin « récompense », en particulier dans le sous-programme qui gère chacun des clones.
Il suffit d'ajouter un test « SI le vaisseau est touché, ALORS supprimer le clone et augmenter le score ».

Sous-programme concernant la récompense

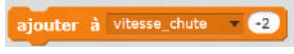
Tâche 3 : augmenter la vitesse de chute à chaque fois que le score augmente (15 minutes)

La difficulté du jeu va augmenter au fur et à mesure que le score progresse, ce qui se traduira par le fait que les astéroïdes (et les pièces) tombent de plus en plus vite. Pour modifier la vitesse de chute, il faut que celle-ci soit gérée par une « variable ». Une fois la variable créée, il ne faut pas oublier de lui donner une valeur (sinon, elle prend la valeur 0 par défaut).
Les élèves procèdent comme pour le score et le nombre de vies pour créer leur variable « vitesse_chute ».

Ils initialisent sa valeur (par exemple à « -10 »), par exemple dans le programme du vaisseau (où les autres variables ont été initialisées).
Il faut ensuite modifier le programme de l'astéroïde et celui de la récompense pour remplacer « ajouter -10 à y » par « ajouter vitesse_chute à y ».

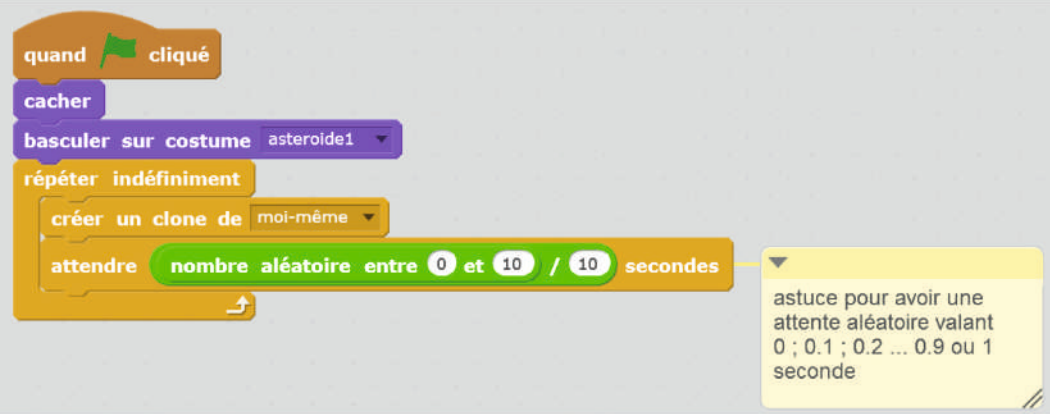
Modification du programme des récompenses en introduisant la variable vitesse_chute. Note : une modification similaire est nécessaire pour le programme de l'astéroïde.

Il faut désormais modifier la vitesse de chute lorsque le score augmente, de façon à corser la difficulté du jeu. Cette dernière tâche ne présente aucune difficulté d'un point de vue de la programmation. La seule difficulté est d'ordre conceptuel : par convention, nous avons défini une vitesse négative (car aller vers le bas signifie faire décroître l'ordonnée), donc « augmenter la vitesse » signifie ici « augmenter en valeur absolue » (on va en réalité diminuer sa valeur).
Il suffit d'ajouter l'instruction suivante juste en dessous l'augmentation du score, dans le programme de la récompense :



Tâche 4 : commenter son programme (10 minutes)

L'enseignant fait remarquer à la classe combien il peut être difficile pour une personne de lire le programme réalisé par une autre ou, pour une même personne, de comprendre son propre programme si l'on n'y a pas touché depuis longtemps.
Il explique que tous les langages de programmation (*Scratch* inclus) permettent d'ajouter des commentaires (ce sont des indications écrites en français – ou en anglais – qui ne sont pas interprétées par la machine et qui s'adressent aux lecteurs humains).
Pour créer un commentaire, il suffit de cliquer avec le bouton droit sur une instruction dans le programme.
Il est inutile de tout commenter : il faut aller à l'essentiel !



Exemple de commentaire dans le programme de l'astéroïde

Les élèves passent quelques minutes à commenter leur programme.

Conclusion

Les bonnes habitudes de programmation évoquées lors de cette séance sont recopiées dans le cahier de projet :

- *Placer des commentaires dans un programme en facilite la lecture et le partage avec autrui.*
- *Un programme bien écrit, avec des variables et des fonctions bien nommées, ne nécessite pas beaucoup de commentaires.*

Séance 7 – Améliorer le rendu graphique du jeu : costumes et messages

Discipline dominante	Mathématiques ou technologie
Résumé	Les élèves améliorent le rendu graphique du jeu en créant et en manipulant des « costumes » pour chaque lutin. Ils créent également un lutin appelé « game over » qui apparait à la fin du jeu. Pour faire communiquer les différents programmes entre eux, ils utilisent des « messages ».
Notions nouvelles *(cf. scénario conceptuel, page 81)*	
Matériel	Identique à la séance précédente.

Note pédagogique

Le début de cette séance se prête particulièrement bien à un prolongement en arts plastiques, puisque l'on peut choisir de faire dessiner les costumes des différents lutins par les élèves eux-mêmes.
On peut aussi, pour gagner du temps, se contenter de piocher dans des bibliothèques d'images (celle intégrée à *Scratch*, ou celle du site http://opengameart.org par exemple).

Les élèves reviennent sur la carte mentale élaborée lors de la Séance 2 et s'intéressent aux graphismes du jeu. Ils font la liste des fonctionnalités à programmer. Voici un découpage possible. Par nature, ce sont des raffinements non indispensables au jeu (mais qui rendront les élèves vraiment satisfaits de leur production) : ils peuvent s'étaler sur plusieurs séances.

Difficulté (cf. page 67)	Nom de la tâche	Travail à effectuer
●	Tâche 1 : créer un nouveau costume pour l'avatar	• Créer un nouveau costume pour l'avatar à partir du costume actuel • Personnaliser ce costume
●	Tâche 2 : changer de costume lorsque l'avatar est percuté par un astéroïde	• Initialiser le costume au lancement du programme • Changer de costume lorsque l'avatar est percuté par un astéroïde, puis revenir au costume initial après un bref instant (pour cela, il faut utiliser l'envoi de « messages »)
●	Tâche 3 : afficher « game over » à la fin du jeu	• Créer un lutin « game over » • Cacher ce lutin au démarrage du jeu • Lorsque le nombre de vies restantes vaut zéro, afficher ce lutin, et stopper tous les autres programmes (utiliser un « message »)
●	Tâche 4 : changer aléatoirement l'apparence de chaque nouvel astéroïde	• Créer plusieurs costumes pour le lutin « astéroïde » • Lorsqu'un nouveau clone est créé, basculer sur un costume aléatoire, et donner au clone une taille aléatoire

	Tâche 5 (optionnelle) : donner une illusion de mouvement en alternant des costumes	• Créer plusieurs costumes pour le lutin « récompense » • Alterner ces costumes de façon à créer une illusion de mouvement (par exemple, une rotation s'il s'agit d'une pièce)
	Tâche 6 (optionnelle) : afficher le nombre de vies de manière graphique	• Créer un lutin « vies » avec plusieurs costumes, chaque costume affichant un symbole correspondant au nombre de vies en cours • Changer de costume à chaque changement de nombre de vies

Tâche 1 : créer un nouveau costume pour l'avatar (15 minutes)

Comme cela a été vu dans l'exercice 4 de la Séance 1, *Scratch* permet de changer l'apparence de chaque lutin à travers l'utilisation de « costumes ». Un lutin peut posséder autant de costumes que l'on souhaite, et ainsi donner une illusion de mouvement (*cf.* le lutin « chat » qui semble marcher) ou tout simplement mettre en évidence un événement particulier. Ici, par exemple, on souhaite montrer que le vaisseau a été percuté par un astéroïde. Il faut donc créer un costume dans lequel son apparence a été modifiée. Pour cela :

• Sélectionner le lutin

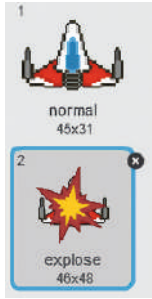
• Cliquer sur l'onglet « costumes » (en haut de la zone de programmation)

• Cliquer, avec le bouton droit de la souris, sur le costume actuel, et choisir « dupliquer » (ainsi, le nouveau costume est identique à l'ancien)

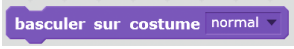
• Modifier le nouveau costume à l'aide des outils de dessin (ou en important une image…)

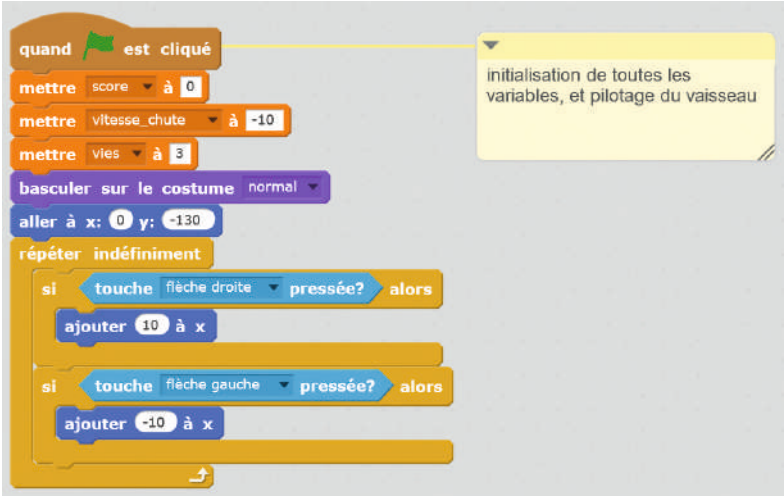
• Renommer chaque costume pour faciliter la programmation ultérieure. Dans notre exemple, les 2 costumes du lutin « vaisseau » ont été renommés « normal » et « explose ».

On vérifie que le lutin change bien d'apparence dans la scène si l'on clique sur l'un ou l'autre des deux costumes.



Tâche 2 : changer de costume lorsque l'avatar est percuté par un astéroïde (30 minutes)

Dire au lutin « vaisseau » de prendre le costume « normal » lorsque le programme se lance est très simple. Il suffit d'ajouter l'instruction  au tout début du programme de ce lutin. Ce programme (qui n'avait pas été modifié depuis la Séance 3) devient donc :



Programme de l'avatar (lutin « vaisseau ») comportant l'initialisation du costume. N.B. : on a également choisi de regrouper ici toutes les initialisations de variables (score, nombre de vies, vitesse de chute des astéroïdes), ce qui n'a rien d'obligatoire.

Il faut maintenant demander au lutin de basculer sur le costume « explose » quand il est touché par un astéroïde. Dans ce cas, puisque le vaisseau ne se cachera plus, il faut demander à l'astéroïde de se cacher (sinon, le contact est prolongé et la variable « vie » change continument, cf. Séance 4). Le problème, c'est que le test « est-ce que le vaisseau touche l'astéroïde » ne se trouve pas dans le programme de l'astéroïde, mais dans le programme du vaisseau. Or, dans le programme de du vaisseau, on peut changer l'apparence (costumes, montrer, cacher…) du vaisseau, mais pas l'apparence de l'astéroïde ou de ses clones. Comment faire ?

Heureusement, *Scratch* possède une fonctionnalité extrêmement pratique (et qui sera utilisée très souvent à l'avenir), permettant aux différents programmes de « communiquer » entre eux. Un programme (celui du vaisseau), peut envoyer un « message » aux autres programmes (celui de l'astéroïde, mais aussi celui de la récompense). Dans le programme de l'astéroïde, on veut déclencher la disparition du clone lorsque le message est reçu.

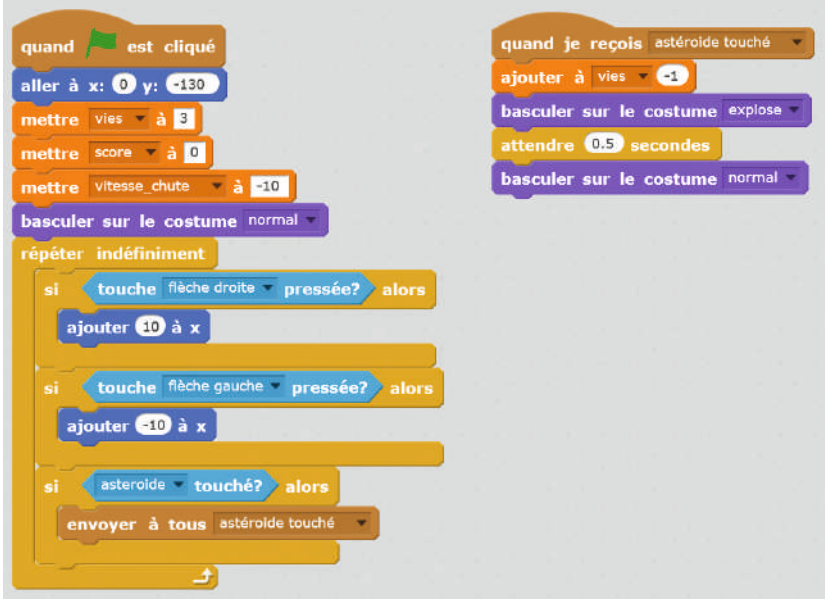
Note scientifique

Envoyer un message revient à créer un « événement » (tout comme le clic sur le drapeau vert était un événement, prédéfini, dans *Scratch*). *Scratch* est un langage de programmation dit « événementiel » : de nombreux sous-programmes peuvent cohabiter, chacun étant déclenché par un événement.

Dans le programme du vaisseau, on va ajouter l'instruction 

à chaque fois que l'astéroïde entre en contact avec le vaisseau. Dans le programme de l'astéroïde, on va ajouter un sous-programme qui se déclenche lorsque le message est reçu, grâce à l'instruction 

Ces 2 instructions se trouvent dans l'onglet « événement » (marron). Dans le programme du vaisseau, on ajoute un nouveau sous-programme, tandis-que dans le programme de l'astéroïde, on insère simplement l'envoi de message lors du test.



Programme du vaisseau modifié en utilisant la fonctionnalité « envoi et réception de messages » (solution provisoire).

Dans le programme de l'astéroïde, on ajoute simplement le bloc d'instructions suivant :

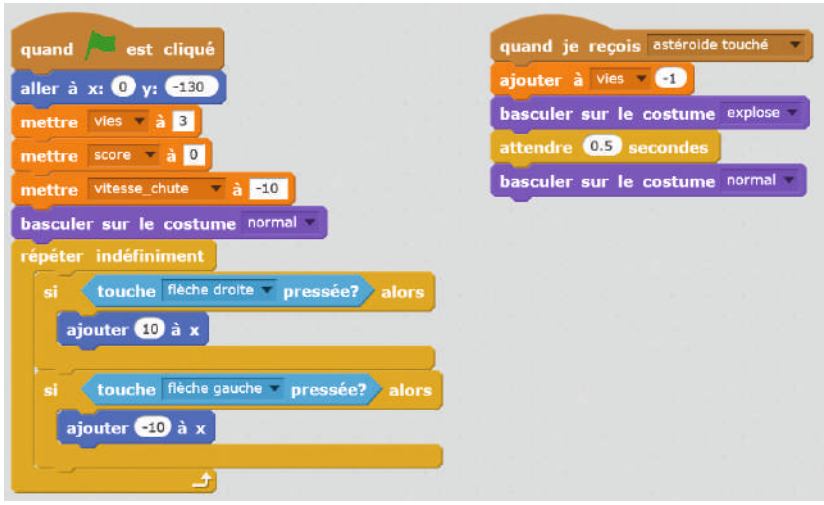
Ajout dans le programme de l'astéroïde (solution provisoire)

Lorsque l'on teste cette solution, on se rend compte qu'elle est fonctionnelle, mais pas vraiment satisfaisante : tous les clones de l'astéroïde sont supprimés d'un coup, lors d'un impact, alors qu'on aimerait ne supprimer que le clone qui a heurté le vaisseau.
Pour cela, la seule solution est de placer le test « si le vaisseau est touché par un astéroïde » dans le programme de l'astéroïde, et pas dans celui du vaisseau. Or, lorsque nous avons programmé cette fonctionnalité (cf. Séance 4), rien ne nous permettait d'imaginer cette contrainte. La solution la plus intuitive, pour gérer l'impact et le nombre de vies était de mettre, naturellement, ce test dans le programme du vaisseau. Il faut donc, désormais, modifier ces 2 programmes pour déplacer le test d'un lutin à l'autre.

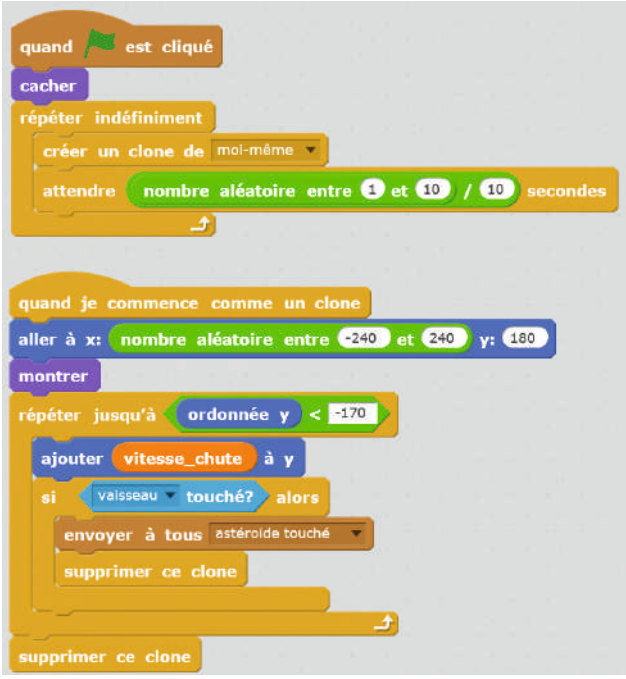
Note pédagogique

Nous aurions pu, dès la Séance 4, expliquer cette contrainte et ainsi placer le test au bon endroit dès le départ. Cependant, nous pensons qu'il est plus pertinent de laisser les élèves suivre leur intuition (d'autant plus qu'elle est justifiée) et de découvrir les contraintes au fur et à mesure, quitte à modifier le programme a posteriori.

Les élèves modifient donc le programme du vaisseau ainsi que celui de l'astéroïde de façon à résoudre ce problème. Finalement, les 2 programmes sont :



Programme du vaisseau (qui gère correctement le nombre de vies et le changement de costumes).



Programme de l'astéroïde (qui gère correctement la collision et la suppression du clone).

Tâche 3 : afficher « game over » à la fin du jeu (20 minutes)

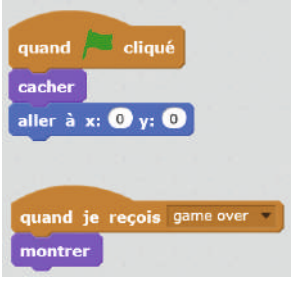
Cette nouvelle fonctionnalité permet aux élèves de revenir sur l'envoi de messages vu lors de la précédente tâche. On souhaite faire disparaitre tous les lutins lorsque le nombre de vies restantes vaut zéro, et faire apparaître le message « game over » à la place. Il faut donc :

• Créer un lutin dont l'apparence est le texte « game over » (on peut utiliser pour cela l'outil de dessin incluse dans *Scratch*, car cet outil donne la possibilité d'écrire du texte).

• Dans le programme qui gère le nombre de vies, ajouter un test : SI vies <1, ALORS envoyer le message « game over ».

• Pour tous les lutins (sauf « game over »), ajouter une instruction « stop tout » indiquant la fin des différents programmes lorsque le message « game over » est reçu.

• Pour le lutin « game over », ajouter l'instruction « montrer » lorsque le message est reçu.



Programme du lutin « game over »

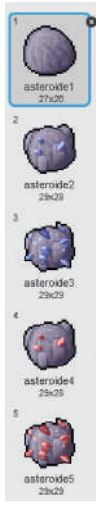
Sous-programme qui gère le nombre de vies.

À ajouter dans les programmes des autres lutins, pour marquer la fin du jeu (aucune action n'est désormais possible pour ce lutin).

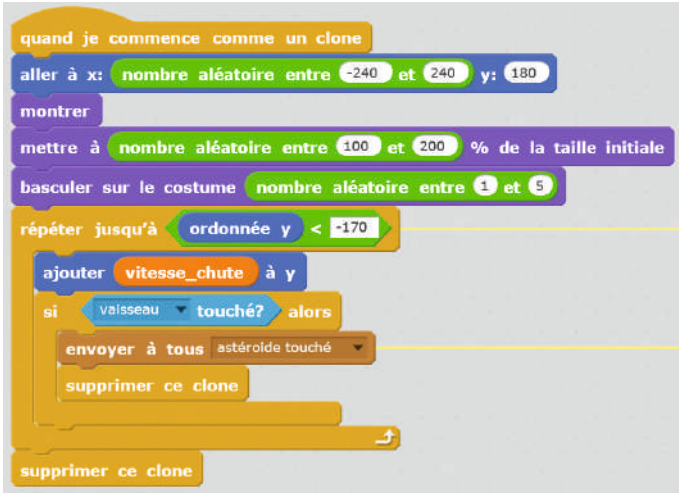
Tâche 4 : changer aléatoirement l'apparence de chaque nouvel astéroïde (15 minutes)

Désormais, les élèves savent créer un costume, et manipuler des nombres aléatoires. Cette tâche ne présente donc pas de difficulté (le plus long étant de créer les costumes).

Voici par exemple les 5 costumes du lutin « astéroïde », ainsi que l'ajout de l'instruction permettant de choisir un costume et une taille aléatoires



Les 5 costumes de l'astéroïde



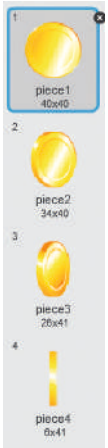
Choix d'un costume et d'une taille aléatoire chaque fois que l'on crée un clone de l'astéroïde.

Tâche 5 (optionnelle) : donner une illusion de mouvement en alternant des costumes (15 minutes)

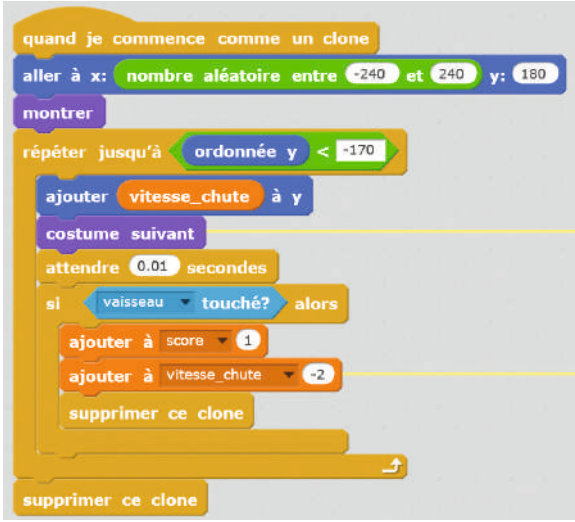
Cette nouvelle tâche n'apporte pas grand-chose d'un point de vue de la programmation, mais elle embellit le jeu. On peut la passer en cas de manque de temps.

Il s'agit de créer des costumes d'un même objet (la pièce) vu sous différents angles, et ainsi créer une illusion de mouvement en alternant les costumes. L'exemple suivant a été réalisé en utilisant l'image « spinning coin » disponible ici : http://opengameart.org/content/spinning-coin

Il suffit de découper les différentes parties de l'image et de les enregistrer comme images indépendantes, puis d'importer ces images dans les différents costumes du lutin « récompense ».



Les 4 costumes de la pièce



Sous-programme de la pièce. L'utilisation de l'instruction « costume suivant » à chaque pas vers le bas crée l'illusion d'une rotation de la pièce sur elle-même (NB : on peut ralentir cette rotation, si on le souhaite, en ajoutant une pause de quelques centièmes de secondes).

Tâche 6 (optionnelle) : afficher le nombre de vies de manière graphique (15 minutes)

Jusqu'à présent, le nombre de vies restantes est une variable dont la valeur est affichée à l'écran. C'est fonctionnel, mais peu esthétique. Dans la plupart des jeux vidéo, le nombre de vies est représenté par un symbole (en général, un cœur), que l'on répète autant de fois qu'il reste de vies.

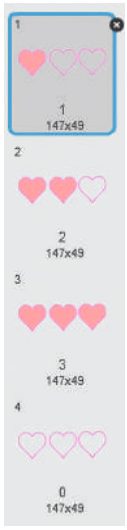
Pour réaliser cette tâche, il faut donc créer un nouveau lutin, que l'on peut appeler « vies », et qui contient plusieurs costumes. Dans notre exemple, puisqu'on limite le nombre de vies à 3, il y a 4 costumes correspondant aux différentes étapes du jeu (3 vies, 2 vies, 1 vie, 0 vie), comme sur la capture d'écran suivante.

À noter que nous avons modifié l'arrière-plan pour qu'il affiche une bande noire en bas, dans une zone dédiée à l'affichage du nombre de vies et du score. Il a également fallu s'assurer que le vaisseau est bien au-dessus de cette bande.

Puisqu'il existe désormais un lutin dédié à l'affichage du nombre de vies, il semble logique de déplacer le sous-programme qui gère l'évolution du nombre de vies dans ce lutin (astuce : pour déplacer un programme d'un lutin à un autre, il suffit de cliquer sur le programme et de le faire glisser vers le lutin cible).

Le programme du lutin « vies » est donc composé de 2 parties :

- Une initialisation (position, taille, costume)
- Le sous-programme qui gère le nombre de vies (déclenché quand le vaisseau est touché par l'astéroïde). Ce sous-programme est modifié pour que l'on bascule vers le costume adéquat.



Les 4 costumes du lutin « vies »

Le programme complet du lutin « vies »

Fin du projet

Le projet consistant à créer un jeu d'arcade est désormais terminé, au sens où le programme créé répond au cahier des charges qui avait été défini en début de projet. Cependant, un tel jeu peut toujours être amélioré, complété (on peut, par exemple, ajouter une fonction de tir pour détruire les astéroïdes)…

Ne pas hésiter à poursuivre ce travail pour découvrir de nouvelles fonctionnalités *Scratch* :

• le stylo (permet à un lutin de dessiner à l'écran)

• les interactions avec l'utilisateur :

– clavier (poser une question à l'utilisateur, qui tape la réponse au clavier, la réponse est enregistrée dans une variable que l'on peut manipuler)

– souris (on peut déclencher des actions en cliquant sur un lutin par exemple, ou suivre les mouvements de la souris)

• les fonctions (un concept fondamental en programmation, mais que nous avons laissé de côté pour ce projet d'initiation à *Scratch*).

Etc.

Avant de clore définitivement ce projet, ne pas oublier de rendre publics les jeux créés par les élèves ! Pour cela, il faut avoir créé un compte utilisateur sur le site https://scratch.mit.edu/ (ce qui est déjà le cas pour les classes ayant travaillé en ligne).

Bilan : a-t-on fait des maths au cours de ce projet ?

Une discussion collective en fin de projet aidera les élèves à réaliser que ce qu'ils ont pris comme une activité ludique comporte de nombreux apprentissages, notamment mathématiques. Les élèves font souvent ressortir l'aspect « logique » de ce travail, mais il est nécessaire de les guider pour faire expliciter toutes les notions abordées. Le scénario conceptuel, page 3, est très utile pour un tel bilan.

Cette ressource est issue du projet thématique *1,2,3... CODEZ !*, paru aux Éditions Le Pommier.

Retrouvez l'intégralité de ce projet sur : https://www.fondation-lamap.org/projets-thematiques.

Fondation *La main à la pâte*
43 rue de Rennes
75006 Paris
01 85 08 71 79
contact@fondation-lamap.org

Site : www.fondation-lamap.org

FONDATION La main à la pâte
POUR L'ÉDUCATION À LA SCIENCE