Étape 1 – Qu'est-ce qu'un jeu de plateforme ?

Discipline dominante	Mathématiques
Résumé	Après la démonstration d'un jeu de plateforme existant par l'enseignant, les élèves constituent un listing des fonctionnalités d'un tel jeu. Ils organisent ces fonctionnalités sous la forme d'une carte mentale qui servira de feuille de route pour le projet.
Notions nouvelles *(cf. scénario conceptuel, page 353)*	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. • 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 • (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 ou si la connexion est très lente, il est tout à fait possible de mener le projet hors-ligne, après installation de *Scratch* sur chaque poste[2]. Dans ce cas, ne pas tenir compte de tout ce qui concerne les « Comptes *Scratch* » et, pour cette séance, de la plateforme de *mind-mapping* collectif.

L'ensemble des fichiers *Scratch* du projet (Platformer_V01, V02, etc.) est téléchargeable sur le site Web du projet (*cf.* page 404). Les fichiers destinés à l'enseignant sont à utiliser lors de démonstrations ou lors des mises en commun. Les fichiers destinés aux élèves doivent leur être rendus accessibles au fur et à mesure de l'avancement du projet, soit sur un studio *Scratch* dédié à la classe, et créé par l'enseignant sur son compte *Scratch*, soit dans un dossier sur le réseau de l'établissement. Dans la suite, on considère que l'enseignant utilise un studio *Scratch*.

Situation déclenchante

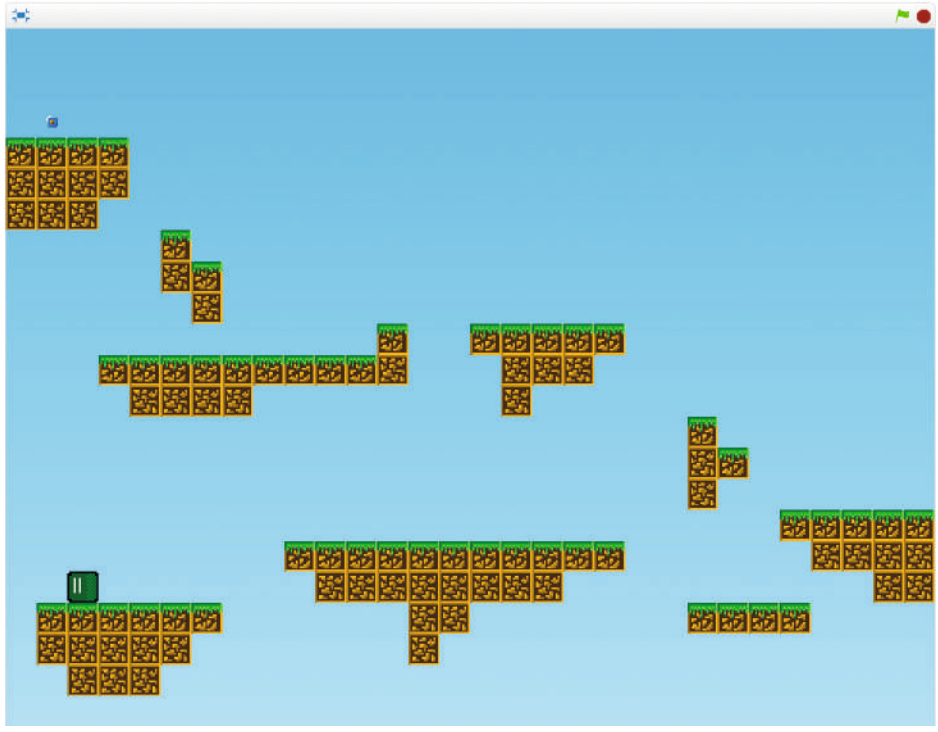
L'enseignant annonce que pour progresser en science informatique, il n'y a rien de tel que de se lancer dans des projets de création. Il propose aux élèves de programmer un jeu vidéo à l'aide de l'environnement de programmation *Scratch*, que les élèves connaissent déjà[3]. Il existe différents types de jeux vidéo : jeux

2. Voir page 70.

3. Voir l'« avertissement », page 351.

de combat, jeux de tir, jeux d'aventure, jeux de rôle, jeux de réflexion, jeux de simulation, etc. Tous les groupes d'élèves vont programmer un « jeu de plateforme », 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 du jeu de plateforme, 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 10 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), et que ces séances se feront en arts plastiques.

Depuis son poste informatique, l'enseignant lance un exemple de jeu de plateforme (nous suggérons d'utiliser l'exemple suivant « A bit of Luck » : https://Scratch.mit.edu/projects/49434122/). Il fait une démonstration du jeu, sans expliquer comment fonctionne le programme. Il demande aux élèves de préciser pourquoi on parle d'un jeu de plateforme : le joueur contrôle un personnage dont les déplacements mettent en jeu des sauts de plateforme en plateforme. Dans chaque « niveau » (ou « tableau »), les plateformes ont une certaine configuration et le personnage doit atteindre une sortie, ce qui lui donne accès au « niveau » suivant.



Capture d'écran du jeu « A bit of luck » utilisé pour cette démonstration.

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. C'est l'objectif de cette étape, pour laquelle nous proposons deux activités successives :

Lister les fonctionnalités du jeu (par groupes)

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 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

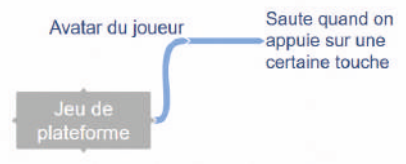
du joueur ». Sur un post-it « Quoi ? » il note « L'avatar saute quand on appuie sur une certaine touche ». Il colle les deux post-it au tableau et les associe par une flèche, en précisant qu'un post-it « Qui ? » peut tout à fait être associé à plusieurs post-it « Quoi ? ».
L'enseignant invite alors les groupes à créer leurs propres post-it et à les agencer sous forme d'affiche pour lister les fonctionnalités du jeu. Le jeu de plateforme 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. Puis chaque groupe récupère son affiche pour l'activité suivante.

Synthèse : organiser les fonctionnalités du jeu en une carte mentale (collectivement)

Note pédagogique

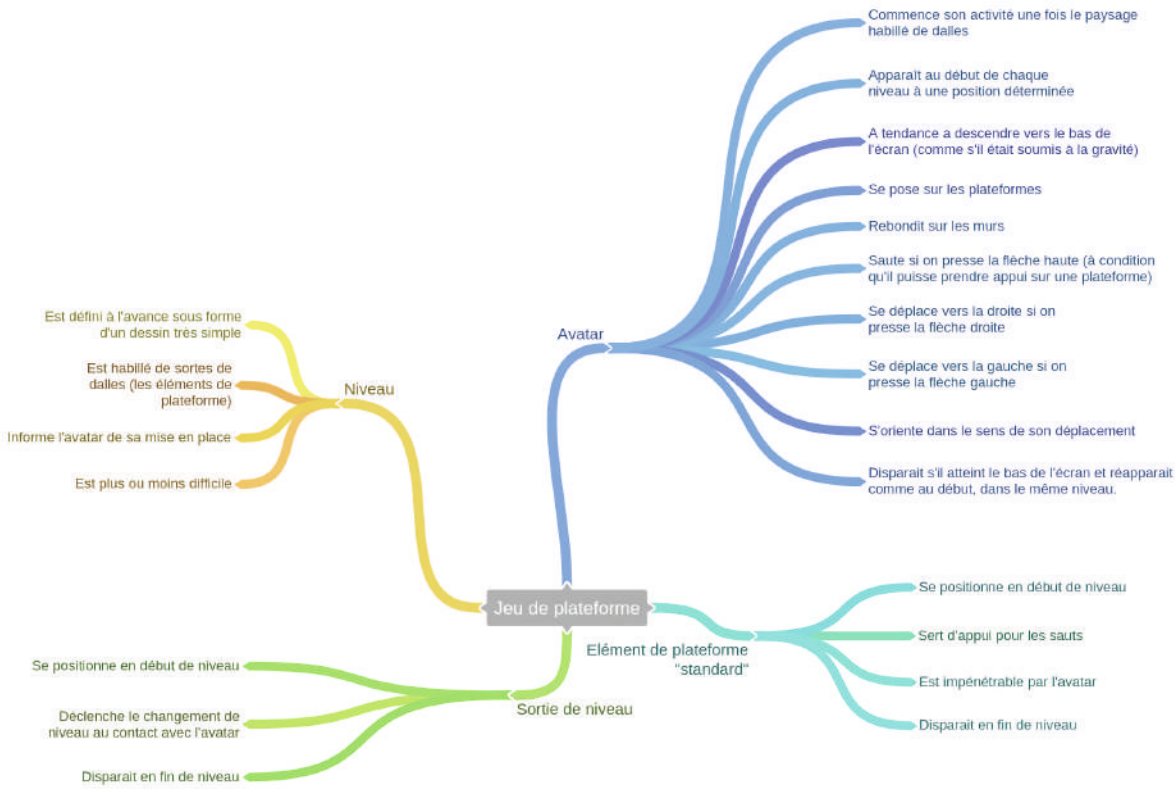
Si possible, utiliser une plateforme de *mind-mapping* collaborative. En cas d'impossibilité d'accès à Internet, réaliser cette carte collectivement sous la forme d'une affiche, qui restera dans la classe (car elle guidera les prochaines séances de programmation).

L'enseignant explique que la classe va mettre en commun le contenu des affiches des différents groupes pour produire une carte mentale collective. Comme les affiches, la carte mentale devra répertorier 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, l'enseignant accède à la plateforme de *mind-mapping* collectif et donne un exemple : il nomme la racine de la carte mentale « Jeu de plateforme » puis crée une boîte « Avatar du joueur » qui pointe vers une action : « Saute quand on appuie sur une certaine touche. ». Il obtient par exemple ceci, s'il utilise la plateforme *Coggle* :



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 de *Coggle*:



Notes pédagogiques

• Lors des deux activités, l'enseignant peut guider la réflexion des élèves, jeu de démonstration à l'appui, en posant des questions telles que: que se passe-t-il si l'avatar arrive en bas de l'écran ? L'avatar peut-il traverser les murs ? Que se passe-t-il si l'avatar se cogne à un plafond ? L'avatar peut-il sauter quand il est déjà en l'air ? Que se passe-t-il quand l'avatar atteint la sortie ? Est-ce que la trajectoire de l'avatar semble naturelle ?

• En particulier, il veille à ce que les élèves remarquent que le décor est constitué de pièces carrées juxtaposées, qui se positionnent sur un fond pastel en début de niveau. On voit le fond furtivement au début du jeu, quand on change de niveau, et quand on interrompt le jeu en cliquant sur l'icône rouge « stop ». Les questions suivantes peuvent guider les élèves sur ce point: de quoi le décor semble-t-il constitué ? Que voit-on apparaître en fond de scène lorsqu'on interrompt le jeu, et de façon furtive quand on change de niveau ?

Mise en commun

Carte mentale à l'appui, l'enseignant annonce que la programmation des différentes fonctionnalités va démarrer dès la prochaine séance. Il demande si l'on peut s'y attaquer dans n'importe quel ordre, ou non. Certains élèves peuvent proposer de commencer par ce qui est le plus facile, et de garder les tâches les plus difficiles pour plus tard. D'autres élèves peuvent remarquer qu'on ne peut pas programmer certaines fonctionnalités (ou en tout cas, pas les tester) tant que d'autres ne sont pas déjà programmées: par exemple, comment vérifier si le saut n'est possible qu'au départ d'une plateforme, dans le cas où il n'y a pas encore de plateforme ?

L'enseignant confirme que la classe va programmer les fonctionnalités en tenant compte à la fois de la difficulté (qui sera progressive) et de la dépendance de certaines fonctionnalités par rapport à d'autres.

Conclusion

L'enseignant guide les élèves vers la formulation de la conclusion suivante : *avant de se lancer dans un projet de programmation, il est essentiel de lister les fonctionnalités que l'on souhaite programmer.*

Étape 2 – Programmer les déplacements latéraux du lutin « Joueur »

Discipline dominante	Mathématiques
Résumé	Cette étape permet aux élèves de se remémorer *Scratch*, en programmant la fonctionnalité de déplacement du lutin « Joueur » sous l'action des flèches droite et gauche du clavier.
Notions nouvelles *(cf. scénario conceptuel, page 353)*	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. • 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 : • *Scratch* est un environnement de programmation très facile à prendre en main, qui utilise un langage de programmation graphique. • 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 la classe : • Le même matériel qu'à l'étape 1. Pour les élèves : • Le même matériel qu'à l'étape 1. • Un ordinateur par binôme permettant d'utiliser *Scratch*. • La Fiche 1 page 363 (seulement si les fichiers *Scratch* sont fournis aux élèves sur un studio *Scratch*), modifiée pour que l'adresse du Studio *Scratch* de la classe apparaisse. • La Fiche 2 page 364.

Situation déclenchante

En début de séance, l'enseignant projette la carte mentale obtenue collectivement par la classe et en distribue une photocopie à chaque élève, à coller dans le cahier de projet : chacun pourra ainsi suivre l'avancement du projet en cochant les fonctionnalités programmées avec succès.

Activité : une première mission avec *Scratch* (par binômes)

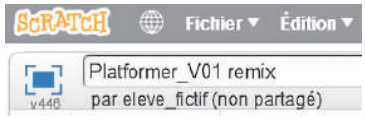
L'enseignant annonce que dès aujourd'hui, les groupes vont commencer à programmer une fonctionnalité toute simple du futur jeu de plateforme : le contrôle des déplacements de l'avatar du joueur vers la droite et vers la gauche, à l'aide des flèches (les sauts seront pour plus tard). Cela permettra à chacun de se remettre dans le bain de *Scratch*, en partant d'une même base. Le travail s'effectuera à l'aide de la Fiche 1 page 363 (si l'enseignant utilise un studio *Scratch* pour fournir les fichiers aux élèves, cas décrit ci-dessous) et de la Fiche 2 page 364.

Conformément aux instructions de la Fiche 1, les binômes s'installent à un ordinateur, se connectent sur le site *Scratch* avec leur compte de binôme et se rendent sur le studio *Scratch* dédié à la classe, dont ils notent

pour mémoire l'adresse dans leur cahier de programmation. Puis ils sélectionnent le projet « Platformer_V01 », ce qui donne accès à l'écran suivant :

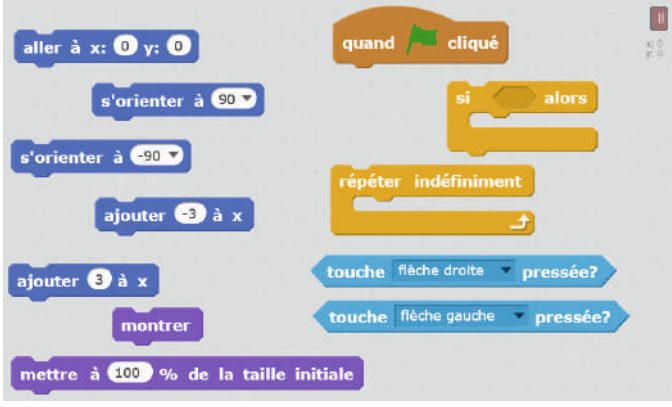
Ils demandent alors à « voir à l'intérieur » le projet (ce qui leur permet de visualiser les instructions de programmation) puis à le « remixer » (ce qui crée une copie modifiable du projet sur leur propre compte). Pour cela, ils cliquent successivement sur les deux icônes suivantes :

Ils renomment alors le projet en remplaçant le suffixe « remix » par leur nom de groupe :



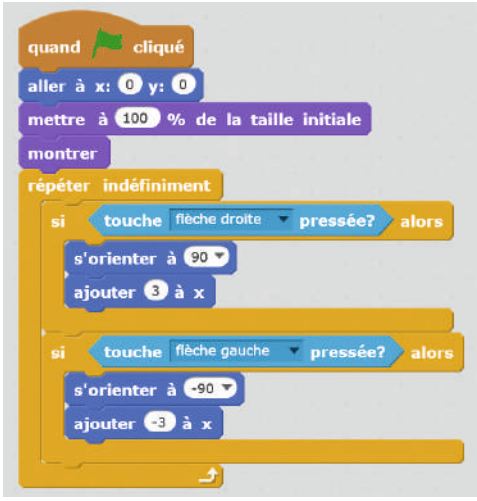
L'ensemble de cette procédure devra être utilisé à chaque fois que l'enseignant fournira un fichier *Scratch* sur le studio. Elle est notée pour mémoire sur la Fiche 1 page 13, que les élèves collent dans leur cahier de programmation.

Le moment est venu pour les élèves de combiner les instructions déjà présentes dans la zone de programmation pour aboutir au résultat recherché : le contrôle des déplacements du lutin « Joueur » par les flèches gauche et droite du clavier. Les instructions proposées sont les suivantes, et les élèves n'ont pas l'obligation de les utiliser toutes :

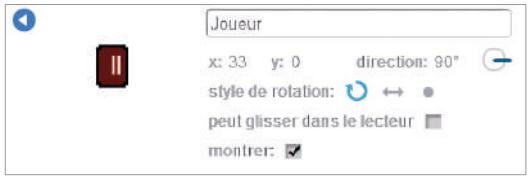


L'enseignant invite les élèves à enregistrer leur programme. En cas de besoin, il peut revenir sur la manière de repérer la position d'un point sur un plan, à l'aide de 2 axes (abscisses, ordonnées), et préciser que, par convention, « x » désigne l'abscisse et « y » l'ordonnée du lutin.

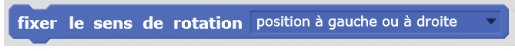
Voici un programme possible, utilisant deux fois l'instruction « Si… alors… », ce qui suppose de la dupliquer après un clic droit ou d'aller en chercher un second exemplaire dans la liste des instructions, catégorie « Contrôle » :



Toutefois, lorsqu'on teste le programme, on s'aperçoit que le lutin « joueur » se met la tête en bas lorsqu'il se dirige vers la gauche, ce qui n'est pas le but recherché ! Le coup de pouce de la Fiche 2 propose deux solutions :
– soit les élèves modifient manuellement ce style, après avoir cliqué sur l'icône « i » associée au lutin ;



– soit ils utilisent l'instruction « fixer le sens de rotation… », accessible dans la catégorie d'instructions « Mouvement » :



Note pédagogique

Les élèves seront amenés à modifier leur façon de programmer les déplacements droite/gauche, à l'Étape 8, page 392. La présente façon de faire s'avèrera en effet peu concluante, une fois que la fonctionnalité de saut aura été programmée.

Mise en commun

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 :

• l'instruction 

permet de déclencher les instructions qui y sont accolées (et qui forment une séquence d'instructions) par l'événement « Clic sur le drapeau vert ». Il y a d'autres instructions de ce type dans la catégorie « Événements ».
• les instructions « montrer », « mettre à 100 % de la taille initiale » et « aller à x : 0 y : 0 » sont des initialisations : elles imposent un état initial, ici pour le lutin « Joueur ».

• 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 choisir quoi faire dans différentes situations. Ici, par exemple, c'est seulement si le joueur appuie sur la flèche droite que le lutin « Joueur » se déplace vers la droite.
• les boucles permettent de répéter plusieurs fois une même séquence d'instructions.
• 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. Les variables « x » et « y » sont prédéfinies dans *Scratch*. Elles représentent l'abscisse et l'ordonnée d'un lutin sur la scène.
La classe termine la séance en cochant les fonctionnalités de la carte mentale qui ont été programmées à cette séance :
• Avatar : apparaît au lancement du programme à une position déterminée.
• Avatar : se déplace vers la droite si on presse la flèche de droite.
• Avatar : se déplace vers la gauche si on presse la flèche de gauche.
• Avatar : s'oriente dans le sens de son déplacement.

FICHE 1
Modifier un fichier *Scratch* fourni sur le Studio de la classe.

Consigne : Suis la procédure décrite ci-dessous pour accéder à une toute première version de ton futur jeu de plateforme. Tu devras répéter cette procédure à chaque fois qu'un fichier *Scratch* te sera fourni sur le studio dédié à la classe.

1. Connecte-toi sur le site https://Scratch.mit.edu/ à l'aide de ton nom d'utilisateur et de ton mot de passe.

2. Dans la fenêtre de saisie du navigateur, saisis l'URL du studio *Scratch* de la classe :
URL du studio : https://Scratch.mit.edu/studios/ /

3. Sélectionne le projet du jour, en l'occurrence : Platformer_V01.

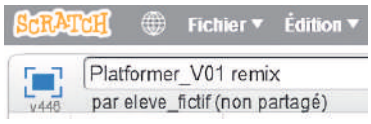
4. Clique sur l'icône 

pour importer le projet sur ton compte.

5. Clique sur l'icône 

pour créer une version personnelle du projet que tu pourras modifier.

6. Renomme le fichier en remplaçant le suffixe « remix » par le nom de ton groupe et clique sur « entrée » :



7. Tu es maintenant prêt(e) à modifier le programme fourni.

8. N'oublie pas de sauvegarder ton travail : 

Consigne : Suis la procédure décrite ci-dessous pour accéder à une toute première version de ton futur jeu de plateforme. Tu devras répéter cette procédure à chaque fois qu'un fichier *Scratch* te sera fourni sur le studio dédié à la classe.

1. Connecte-toi sur le site https://Scratch.mit.edu/ à l'aide de ton nom d'utilisateur et de ton mot de passe.

2. Dans la fenêtre de saisie du navigateur, saisis l'URL du studio *Scratch* de la classe :
URL du studio : https://Scratch.mit.edu/studios/ /

3. Sélectionne le projet du jour, en l'occurrence : Platformer_V01.

4. Clique sur l'icône 

pour importer le projet sur ton compte.

5. Clique sur l'icône 

pour créer une version personnelle du projet que tu pourras modifier.

6. Renomme le fichier en remplaçant le suffixe « remix » par le nom de ton groupe et clique sur « entrée » :

7. Tu es maintenant prêt(e) à modifier le programme fourni.

8. N'oublie pas de sauvegarder ton travail : 

FICHE 2
Programmer les déplacements latéraux du lutin « Joueur »

Consigne :
Combine les instructions déjà présentes dans la zone de programmation du lutin « Joueur » pour que celui-ci se déplace vers la gauche en cas de pression sur la flèche gauche, et vers la droite en cas de pression sur la flèche droite. Veille à ce que le lutin « Joueur » s'oriente dans le sens de son déplacement. Tu n'es pas obligé(e) d'utiliser toutes les instructions et une instruction peut servir plusieurs fois.

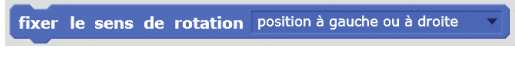
Consigne :
Combine les instructions déjà présentes dans la zone de programmation du lutin « Joueur » pour que celui-ci se déplace vers la gauche en cas de pression sur la flèche gauche, et vers la droite en cas de pression sur la flèche droite. Veille à ce que le lutin « Joueur » s'oriente dans le sens de son déplacement. Tu n'es pas obligé(e) d'utiliser toutes les instructions et une instruction peut servir plusieurs fois.

Coup de pouce :
Pense à choisir le style de rotation du lutin « Joueur » après avoir cliqué sur l'icône « information » et profites-en pour tester l'effet des 3 styles de rotation possibles :

Tu peux aussi choisir le style de rotation en utilisant l'instruction suivante dans ton programme :

Coup de pouce :
Pense à choisir le style de rotation du lutin « Joueur » après avoir cliqué sur l'icône « information » et profites-en pour tester l'effet des 3 styles de rotation possibles :

Tu peux aussi choisir le style de rotation en utilisant l'instruction suivante dans ton programme :



Étape 3 – Programmer la chute du lutin « Joueur »

Discipline dominante	Mathématiques
Résumé	La classe décide de programmer la chute du lutin « Joueur ». Cela suppose de créer un « moteur physique » qui simule l'effet de la gravité. Les élèves travaillent à partir d'une feuille de route de programmation fournie par l'enseignant, qui décompose le travail en tâches simples. Les élèves sont amenés à créer et à manipuler des variables.
Notions nouvelles *(cf. scénario conceptuel, page 353)*	Machines : • En combinant des instructions élémentaires, nous pouvons faire exécuter des tâches complexes à des 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.
Matériel	Pour la classe : • Le même matériel qu'à l'étape 2 Pour les élèves : • Le même matériel qu'à l'étape 2. • La Fiche 3 page 370.

Note pédagogique

Cette séance peut être préparée conjointement avec le professeur de physique-chimie, pour ce qui concerne le « moteur physique » (gravité…)

Situation déclenchante

En début de séance, l'enseignant projette la carte mentale de la classe et propose de s'attaquer aux deux fonctionnalités suivantes, qui sont liées :

• Avatar : a tendance à descendre vers le bas de l'écran (comme s'il était soumis à la gravité).

• Avatar : disparait s'il atteint le bas de l'écran et réapparait à la même position qu'au lancement du programme.

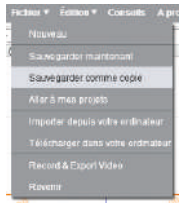
Activité : programmation de la chute du « Joueur » (par binômes)

L'enseignant propose la feuille de route suivante (projetée au tableau et collée dans les cahiers de programmation). Cette feuille de route associe plusieurs tâches simples à chaque fonctionnalité :

Fonctionnalité du programme	Nature des tâches à réaliser	Difficulté (cf. page 69)
1 – Avatar : a tendance à descendre vers le bas de l'écran (comme s'il était soumis à la gravité)	**Tâche 1 :** créer une variable « vitesse verticale » pour le « Joueur » et l'initialiser à une valeur au choix.	
	Tâche 2 : faire en sorte que le « Joueur » se déplace verticalement – et en continu – conformément à la valeur de la variable « vitesse verticale ». Tester différentes valeurs de cette variable (une valeur positive doit donner un déplacement vers le haut, une valeur négative doit donner un déplacement vers le bas).	
	Tâche 3 : créer une variable « gravité » pour le « Joueur » et l'initialiser à une valeur négative au choix.	
	Tâche 4 : faire en sorte que le « Joueur » aille de plus en plus vite vers le bas, en modifiant la valeur de la variable « vitesse verticale » à l'aide de la variable « gravité ». Tester différentes valeurs négatives de la variable « gravité » et différentes valeurs initiales de la variable « vitesse verticale ».	
2 – Avatar : disparait s'il atteint le bas de l'écran et réapparait à la même position qu'au lancement du programme.	**Tâche 5 :** faire disparaître le « Joueur » si la variable prédéfinie « ordonnée y » indique qu'il a atteint le bas de l'écran. Le faire réapparaître à sa position de départ, avec une vitesse verticale nulle.	

Les élèves :

- s'installent aux ordinateurs ;
- se connectent sur leur compte *Scratch* ;
- reprennent leur programme *Platformer_V01* (cliquer  sur puis sur Voir à l'intérieur) ;
- l'enregistrent aussitôt comme copie (voir illustration ci-contre) et modifient le numéro de version : V02 au lieu de V01 ;
- se lancent dans les tâches listées, tandis que l'enseignant passe de groupe en groupe.



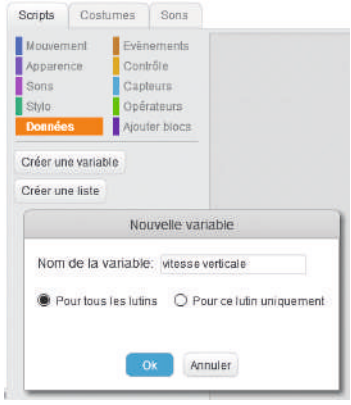
Note pédagogique

Pour cette séance, l'enseignant a procédé au préalable au découpage en tâches simples des fonctionnalités, et propose directement la 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.

Ci-après, quelques éléments de correction pour les tâches 1 à 5.

Tâche 1 : créer une variable « vitesse verticale » pour le « Joueur »

La création d'une variable s'effectue dans la catégorie « Données » de l'onglet « Scripts », comme illustré ci-dessous :



Notes scientifiques

• 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.

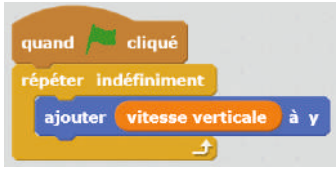
• Pour qu'un programme soit facile à comprendre, il est important de donner des noms explicites aux variables que l'on crée. Cette bonne habitude limite également les bugs de programmation. Le nom de la variable peut donc être, tout simplement : « vies ».

L'initialisation d'une variable s'effectue dans le corps du programme :

Tâche 2 : faire en sorte que le « Joueur » se déplace verticalement

Les élèves ont tendance à proposer ceci :

Mais dans cette proposition, l'instruction « ajouter vitesse verticale à y » n'est exécutée qu'une fois, au lancement du programme. Pour que le déplacement s'effectue pendant toute l'exécution du programme (en fait, tant que le lutin « Joueur » ne bute pas sur un bord de la scène), on insère l'instruction de mouvement dans une boucle infinie. Le déplacement se fait verticalement, et d'un nombre de pixels correspondant à la valeur de la variable « vitesse verticale », à chaque itération de la boucle :



L'enseignant amène les élèves à tester le programme avec une initialisation de la variable « vitesse verticale » à différentes valeurs : des valeurs positives, négatives, ou la valeur zéro. Les élèves confirment que les valeurs

positives donnent un déplacement du lutin vers le haut, les valeurs négatives un déplacement vers le bas. Si la valeur est nulle, le lutin reste sur place. Une valeur faiblement négative (par exemple -1) donne un déplacement lent vers le bas, une valeur plus fortement négative (par exemple, -5) donne un déplacement plus rapide vers le bas.

Note pédagogique

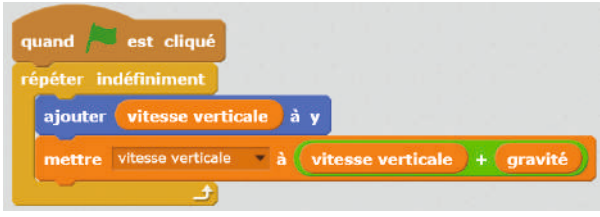
L'enseignant peut aborder la notion de valeur absolue à cette occasion.

Tâche 3 : créer une variable « gravité » pour le « Joueur »

Voir tâche 1, on obtient :

Tâche 4 : faire en sorte que le « Joueur » aille de plus en plus vite vers le bas, en modifiant la valeur de la variable « vitesse verticale » à l'aide de la variable « gravité »

Pour simuler l'effet de l'attraction gravitationnelle terrestre, on doit faire en sorte que le lutin ait une vitesse verticale de plus en plus petite. Si la vitesse verticale est nulle au départ, elle vaudra par exemple -1 après une itération de la boucle, puis -2, puis -3, etc. Si la vitesse verticale est positive au départ (par exemple, valeur 2), elle vaudra 1 après une itération de la boucle, puis 0, puis -1, etc. Si la vitesse verticale est négative au départ (par exemple, valeur -4), elle vaudra -5 après une itération de la boucle, puis -6, puis -7, etc.
Il suffit donc d'incrémenter la valeur de la variable « vitesse verticale » de la valeur de la variable « gravité », à chaque itération de la boucle. On initialise ensuite la variable « gravité » à une valeur qui rende le mouvement réaliste (forcément une valeur négative si l'on souhaite que le lutin chute vers le bas de l'écran).



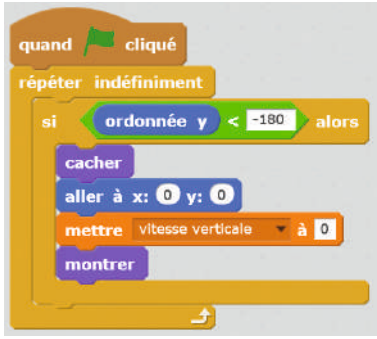
Notes pédagogiques

• La réflexion sur la façon « physiquement correcte » de programmer l'effet de la gravité peut être menée avec le professeur de sciences physiques. Le fait qu'une force (la gravité) soit assimilable à une accélération (on ne tient pas compte de la masse ici) va au-delà du programme du cycle 4, mais peut être abordé rapidement avec les élèves, sans faire l'objet d'une évaluation. En particulier, il faut que les élèves comprennent la différence entre une vitesse et une accélération.
– « vitesse verticale » est une vitesse. Elle affecte donc la position du joueur (d'où l'instruction « ajouter vitesse verticale à y ») ;
– « gravité » est une accélération (qu'on nomme « accélération de pesanteur »). Elle modifie, non pas la position du joueur, mais sa vitesse (d'où l'instruction « ajouter gravité à vitesse verticale »).

• Cette tâche est difficile pour la plupart des élèves. Ne pas hésiter, au besoin, à proposer une version déjà corrigée de ce programme. L'activité n'est plus « comment programmer cette fonctionnalité ? » mais « que signifie ce programme ? Pourquoi est-il écrit de cette façon ? »…

Tâche 5 : gérer la disparition du « Joueur » quand il arrive en bas de l'écran

Afin de détecter si le « Joueur » atteint le bas de l'écran, on teste la valeur de la variable prédéfinie « ordonnée y ». Si cette valeur est inférieure à -180 (qui correspond au bas de l'écran), on cache le « Joueur », on le repositionne comme souhaité et on réinitialise sa vitesse verticale, par exemple comme ceci :



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 la suite du travail : les fonctionnalités qui restent à programmer pour le lutin « Joueur » sont le repos sur les plateformes, les sauts depuis des plateformes et les rebonds sur les éléments de plateformes. Nous pourrions programmer ces fonctionnalités « dans le vide », mais faute de plateformes, nous ne pourrions pas les tester. Cette façon de faire serait inefficace et source d'erreurs (aussi appelées bugs). Il nous faut donc, la prochaine fois, programmer le positionnement des plateformes.

Conclusion

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

• 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.

• n'importe quelle tâche complexe 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).

FICHE 3
Une feuille de route de programmation

Consigne : mets en œuvre la feuille de route de programmation suivante :

Fonctionnalité du programme	Nature des tâches à réaliser	Difficulté (cf. page 69)
1 – Avatar : a tendance à descendre vers le bas de l'écran (comme s'il était soumis à la gravité)	**Tâche 1 :** créer une variable « vitesse verticale » pour le « Joueur » et l'initialiser à une valeur au choix.	
	Tâche 2 : faire en sorte que le « Joueur » se déplace verticalement – et en continu – conformément à la valeur de la variable « vitesse verticale ». Tester différentes valeurs de cette variable (une valeur positive doit donner un déplacement vers le haut, une valeur négative doit donner un déplacement vers le bas).	
	Tâche 3 : créer une variable « gravité » pour le « Joueur » et l'initialiser à une valeur négative au choix.	
	Tâche 4 : faire en sorte que le « Joueur » aille de plus en plus vite vers le bas, en modifiant la valeur de la variable « vitesse verticale » à l'aide de la variable « gravité ». Tester différentes valeurs négatives de la variable « gravité » et différentes valeurs initiales de la variable « vitesse verticale ».	
2 – Avatar : disparait s'il atteint le bas de l'écran et réapparait à la même position qu'au lancement du programme.	**Tâche 5 :** faire disparaître le « Joueur » si la variable prédéfinie « ordonnée y » indique qu'il a atteint le bas de l'écran. Le faire réapparaître à sa position de départ, avec une vitesse verticale nulle.	

Consigne : mets en œuvre la feuille de route de programmation suivante :

Fonctionnalité du programme	Nature des tâches à réaliser	Difficulté (cf. page 69)
1 – Avatar : a tendance à descendre vers le bas de l'écran (comme s'il était soumis à la gravité)	**Tâche 1 :** créer une variable « vitesse verticale » pour le « Joueur » et l'initialiser à une valeur au choix.	
	Tâche 2 : faire en sorte que le « Joueur » se déplace verticalement – et en continu – conformément à la valeur de la variable « vitesse verticale ». Tester différentes valeurs de cette variable (une valeur positive doit donner un déplacement vers le haut, une valeur négative doit donner un déplacement vers le bas).	
	Tâche 3 : créer une variable « gravité » pour le « Joueur » et l'initialiser à une valeur négative au choix.	
	Tâche 4 : faire en sorte que le « Joueur » aille de plus en plus vite vers le bas, en modifiant la valeur de la variable « vitesse verticale » à l'aide de la variable « gravité ». Tester différentes valeurs négatives de la variable « gravité » et différentes valeurs initiales de la variable « vitesse verticale ».	
2 – Avatar : disparait s'il atteint le bas de l'écran et réapparait à la même position qu'au lancement du programme.	**Tâche 5 :** faire disparaître le « Joueur » si la variable prédéfinie « ordonnée y » indique qu'il a atteint le bas de l'écran. Le faire réapparaître à sa position de départ, avec une vitesse verticale nulle.	

Étape 4 – Positionner des éléments de plateforme : les dalles

Discipline dominante	Mathématiques
Résumé	Afin que le lutin « Joueur » évolue dans un paysage contenant des plateformes, les élèves programment la fonctionnalité de positionnement des éléments de plateforme sur la scène. Ils démarrent d'un programme fourni par l'enseignant qu'ils modifient et complètent.
Notions nouvelles *(cf. scénario conceptuel, page 353)*	Bonnes habitudes de programmation : • Lorsqu'un même bloc d'instructions doit être utilisé plusieurs fois dans un programme, il est judicieux de l'intégrer dans une fonction. Algorithmes : • Certaines boucles, dites « itératives » sont répétées un nombre prédéfini de fois.
Matériel	Pour l'enseignant : • Fichier *Scratch* Platformer_V03_demo. Pour les élèves : • Si l'enseignant choisit de faire la première activité, fichier Platformer_V03. • Sinon, fichier Platformer_V03_dalles_24.

Notes pédagogiques

• **Cette séance est la plus difficile du projet.** Elle permet d'habiller le jeu en définissant un décor, puis en habillant ce décor à l'aide de dalles. Le résultat est gratifiant, et motivera les élèves. Cependant, devant la difficulté de la tâche, des élèves n'ayant pas un très bon niveau en *Scratch* peuvent se décourager.

• Avec des élèves « moyens » (rappel : ce projet n'est pas du tout adapté à des élèves débutants ou faux débutants, même après quelques séances d'initiation), nous conseillons d'adopter, au choix, l'une des 2 méthodes suivantes :

– **Choix 1** : fournir le programme final corrigé de cette séance, et analyser ce programme avec les élèves. Pourquoi y a-t-il des dalles ? Pourquoi ont-elles ces dimensions ? Quel est le rôle du lutin explorateur ?, etc.

– **Choix 2** : fournir un décor fixe (sous la forme d'un paysage que l'on importe) et passer aux séances suivantes en utilisant ce décor fixe. En fin de projet, pour les élèves les plus avancés, on pourra peaufiner le programme en définissant un décor personnalisable à l'aide de dalles, comme proposé dans cette séance.

Situation déclenchante

Notre « Joueur » est dorénavant soumis à une gravité virtuelle (il accélère vers le bas de l'écran). Nous pouvons le contrôler à l'aide des flèches et il disparaît lorsqu'il atteint le bas de l'écran, pour réapparaître à une position prédéfinie. Il faudrait maintenant qu'il ait des plateformes sur lesquelles se poser et se déplacer, comme discuté à la séance 2. Pour la séance d'aujourd'hui, les élèves vont travailler avec des éléments de plateforme (ou « dalles ») prédéfinis et un « paysage » fourni. Ce paysage est géré par le(s) costume(s) d'un nouveau lutin. Autrement dit, la fonctionnalité suivante a été programmée par l'enseignant :

• Niveau : est défini à l'avance sous forme d'un dessin très simple.

Le travail des élèves va consister à programmer deux nouvelles fonctionnalités :
• Élément de plateforme : se positionne en début de niveau.
• Niveau : est habillé de sortes de « dalles » (éléments de plateforme).
Ces deux fonctionnalités sont étroitement liées : c'est le positionnement de plusieurs dalles qui « habille » le niveau.
L'enseignant fait la démonstration du résultat attendu à l'aide du fichier Platformer_V03_demo. Les élèves pourront à tout moment demander à revoir ce programme s'exécuter.
Le travail va se dérouler en trois temps :
• définition de la dimension des dalles (l'enseignant montre que plusieurs formats sont possibles en relançant la démonstration avec les dimensions A, B, C et D).
• découpage de la fonctionnalité complexe recherchée en tâches de programmation simples.
• mise en œuvre de ces tâches de programmation.

Quelle dimension donner aux dalles ? (par groupes, optionnelle)

Note pédagogique

Si l'enseignant ne souhaite pas mener cette activité optionnelle, il impose un format de dalle, par exemple 24 x 24 pixels et peut simplement vérifier avec la classe que ce format permet de paver la scène entièrement. Dans ce cas, pour l'activité suivante, il fournit à la classe un fichier Platformer_V03_dalles_24 (au lieu du fichier Platformer_V03), qui a été nettoyé des scripts et costumes devenus inutiles.

L'enseignant demande aux élèves de s'installer en groupes (constitués de 2 ou 3 binômes de projet). Les binômes vont devoir décider d'une dimension à donner aux dalles qui constitueront les plateformes de leurs jeux. Il y a quelques contraintes, qui peuvent être discutées avec la classe ou présentées comme un cahier des charges :
• les dalles doivent être de forme rectangulaires (on imposera même qu'elles soient carrées) ;
• une fois juxtaposées en lignes et en colonnes, elles doivent pouvoir couvrir exactement toute la scène (sans déborder ni laisser de blancs) ;
• elles doivent être plus hautes et plus larges que le « Joueur » (pour que celui-ci puisse passer entre deux alignements de dalles, au niveau d'une ligne/colonne sans dalle), mais en même temps être assez petites pour que l'on puisse dessiner des paysages intéressants à l'aide des dalles ;
• la scène fait 480 pixels de largeur et 360 pixels de hauteur ;
• le « Joueur » fourni fait 12 pixels de largeur et 16 pixels de hauteur.
Les groupes disposent d'une dizaine de minutes pour réfléchir, après quoi l'enseignant demandera à chaque groupe de faire une (ou deux) proposition(s), en nombre de pixels pour le côté des dalles. Les groupes devront également indiquer combien de lignes et de colonnes de dalles permettent de couvrir entièrement la scène.

Notes pédagogiques

Si nécessaire, l'enseignant peut guider les élèves en leur suggérant :
• de schématiser la scène avec des 480 pixels de largeur et 360 pixels de hauteur pour faciliter leur raisonnement.
• de rechercher des nombres qui divisent à la fois 480 et 360, soit de façon un peu empirique, soit en décomposant 480 et 360 en produits de facteurs premiers (on

divise par 2 tant que c'est possible, puis par 3, etc. en égrenant tous les nombres premiers successifs) :

$$480 = 2 \times 2 \times 2 \times 2 \times 2 \times 3 \times 5$$
$$360 = 2 \times 2 \times 2 \times 3 \times 3 \times 5$$

Les diviseurs communs de 480 et de 360 (1 mis à part) sont des produits de termes communs aux deux décompositions : 2, 3, $2 \times 2 = 4$, 5, $2 \times 3 = 6$, $2 \times 2 \times 2 = 8$, $2 \times 5 = 10$, $2 \times 2 \times 3 = 12$, $3 \times 5 = 15$, $2 \times 2 \times 5 = 20$, $2 \times 2 \times 2 \times 3 = 24$, $2 \times 3 \times 5 = 30$, $2 \times 2 \times 2 \times 5 = 40$, $2 \times 2 \times 3 \times 5 = 60$, $2 \times 2 \times 2 \times 3 \times 5 = 120$.

Lors de la mise en commun, les propositions des élèves sont rassemblées dans un tableau de synthèse et évaluées au vu des contraintes :

Dimension du côté des dalles (en pixels)	Nombre de lignes de dalles sur la scène de 360 pixels de hauteur	Nombre de colonnes de dalles surla scène de 480 pixels de hauteur	Passage du « Joueur » possible au niveau d'une ligne ou d'une colonne	Possibilité de dessiner des formes intéressantes en dalles	Toutes les contraintes respectées
10	36	48	NON	1028 dalles pour dessiner	NON
15	24	32	NON	768 dalles pour dessiner	NON
16	Pas un nombre entier	Pas un nombre entier	OUI (sans aucune marge)	Pas un nombre entier de dalles	NON
20	18	24	OUI	432 dalles pour dessiner	OUI
24	15	20	OUI	300 dalles pour dessiner	OUI
30	12	16	OUI	192 dalles pour dessiner	OUI (mais nombre de dalles un peu faible)
36	10	Pas un nombre entier	OUI	Pas un nombre entier de dalles	NON
40	9	12	OUI	108 dalles pour dessiner	NON
60	6	8	OUI	48 dalles pour dessiner	NON

Trois dimensions de dalle permettent de respecter toutes les contraintes : 20 pixels, 24 pixels et 30 pixels. La suite du projet est décrite dans le cas où les dalles ont 24 pixels de côté, ce qui correspond à 15 lignes et à 20 colonnes de dalles pour couvrir toute la scène. En pratique, les groupes n'ont pas obligation de choisir tous la même dimension de dalle, mais une fois leur choix opéré, ils devront s'y tenir.

Créer la feuille de route de programmation (par binômes puis collectivement)

Pour ce travail, les élèves s'installent aux ordinateurs par binômes de projet. Ils récupèrent le fichier Platformer_V03 fourni par l'enseignant (rappel : en cas d'utilisation d'un studio *Scratch* par l'enseignant, la procédure que doivent suivre les élèves pour remixer le programme fourni est décrite sur la Fiche 1, page 363). Ils prennent un moment pour explorer le fichier fourni (onglet Scripts et onglet Costumes de chaque lutin) et lisent les commentaires intégrés au programme. Ces commentaires fournissent des informations importantes.

Les élèves peuvent constater que :

• les costumes du lutin « Paysages » sont minuscules. Ils sont affichés sur la scène avec un très fort grossissement et définissent alors le paysage à l'échelle du pixel : un pixel du costume correspond à une dalle sur la scène. Les angles du paysage sont repérés hors champ pour faciliter le repérage lors du dessin.

N.B. : pourquoi un lutin si petit ? Imaginons que l'on souhaite habiller notre paysage avec des dalles de 24 pixels de coté. Si l'on se contente de dessiner le paysage à l'échelle 1 dans le costume du lutin, il est très probable que certains éléments soient à cheval sur les futures dalles, à moins de se forcer à ne dessiner que des éléments de 24 pixels de coté.

Il parait plus simple de tout réduire : on dessine un paysage tout petit (les éléments font 1 pixel de coté), et ensuite on agrandit le tout d'un facteur 24 pour que cela occupe toute la scène. Ainsi, on évite d'avoir des éléments à cheval sur plusieurs dalles…

• le lutin « Dalles » et le lutin « Costumes » ont chacun 3 costumes adaptés aux dimensions 20, 24 et 30 pixels des dalles. Seul un de ces trois costumes est utile pour un binôme donné, selon la dimension choisie pour ses dalles. Le binôme peut éventuellement supprimer les deux autres costumes, et simplifier le script du lutin « Paysages » en conséquence.

• le lutin « Dalles » comporte un costume nommé « Explorateur », de dimension 2 x 2 pixels, en plus du costume « Dalle ». Lors de la démonstration de l'enseignant, on voyait le costume « Explorateur » parcourir la scène, et les dalles se positionner si l'explorateur passait sur un élément de paysage. Cela fournit une piste de programmation…

Les binômes ont maintenant une quinzaine de minutes pour proposer leur liste de tâches simples, en s'aidant de leurs observations précédentes.

Lors de la mise en commun, un groupe d'élèves présente sa liste de tâches. Puis celle-ci est enrichie ou modifiée sur suggestion d'autres groupes et de l'enseignant. La classe arrive alors à un découpage qui sera mis en œuvre par chaque groupe lors de l'activité suivante. Voici une proposition de découpage très détaillée, mais il n'est pas nécessaire que la classe atteigne ce niveau de détail avant de passer à l'activité suivante :

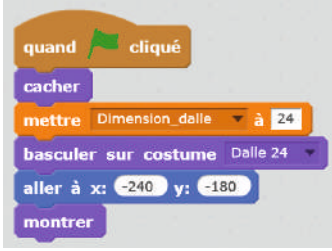
Fonctionnalité du programme	Nature des tâches à réaliser	Difficulté (cf. page 69)
1 – Avatar : a tendance à descendre vers le bas de l'écran (comme s'il était soumis à la gravité)	**Tâche 1 :** créer une variable « dimension dalle » visible par tous les lutins [c'est déjà fait dans le programme fourni]. Affecter une valeur à la variable dimension dalle (par exemple 24 pixels).	
	Tâche 2 : positionner le lutin « Dalles » tout à fait en bas à gauche de la scène et le faire apparaître avec son costume « Dalle ».	
	Tâche 3 : faire déplacer le lutin « Dalles » vers la droite de sa propre largeur.	
	Tâche 4 : répéter ce déplacement de façon à ce que le lutin termine à l'angle en bas à droite de la scène.	
	Tâche 5 : faire en sorte qu'une « copie » du lutin « Dalles » reste à chaque position explorée, pour remplir la ligne du bas de la scène.	
	Tâche 6 : créer une fonction « Remplissage ligne » qui prenne comme information l'ordonnée de la base de la ligne à remplir, et qui effectue l'équivalent de la tâche 5. Faire appeler cette fonction par le programme principal du lutin « Dalles ».	

	Tâche 7 : appeler la fonction « Remplissage ligne » autant de fois qu'il y a de lignes, avec des valeurs adéquates pour l'entrée Y.	
	Tâche 8 : ne faire poser un clone du lutin « Dalles » que s'il y a un élément de paysage à cet endroit.	
	Tâche 9 : faire effectuer la détection du paysage par le costume « Explorateur » du lutin « Dalles ».	

Les élèves se lancent, en autonomie, dans la réalisation des tâches listées précédemment, tandis que l'enseignant passe de groupe en groupe pour échanger avec les élèves sur l'avancement de leur travail. L'enseignant rappelle que les élèves doivent sauvegarder régulièrement leur travail.
Ci-dessous, des éléments de correction pour les tâches 1 à 9 :

Tâches 1 et 2 : créer une variable « dimension dalle » et initialiser la position du lutin « Dalles »

À l'issue des tâches 1 et 2, le programme du lutin « Dalles » ressemble à ceci :



Le lancement du programme par un clic sur le drapeau vert déclenche l'affectation de la valeur 24 (par exemple) à la variable Dimension_dalle. Le lutin prend son costume « Dalle 24 », puis il se rend à la position d'abscisse – 240 pixels et d'ordonnée – 180 pixels, conformément au repère de position du costume actif. Enfin, le lutin se montre. Les instructions « cacher » et « montrer » évitent que l'on voie furtivement une dalle mal positionnée ou de taille incorrecte, au lancement du programme.
Le repère de position des costumes est visible depuis l'onglet Costumes : il est indiqué par une croix grise (voir illustration ci-dessous). Dans le cas qui nous intéresse, ce repère a été positionné (en amont de la séance) à l'angle en bas à gauche de la dalle. Donc, lorsque le lutin est envoyé à une certaine position, c'est en fait l'angle en bas à gauche de la dalle qui se retrouve à cette position.
N.B. : la position du repère est configurable depuis l'onglet Costumes, grâce à l'icône 

Tâche 3 : faire déplacer le lutin « Dalles » vers la droite de sa propre largeur

Plusieurs instructions permettent de déplacer le lutin « Dalles » vers la droite de sa propre largeur :

La première produit l'effet recherché seulement si le lutin est déjà orienté vers la droite. La seconde est plus robuste : elle fonctionne quelle que soit l'orientation du lutin. Pour remplir la tâche 3, il suffit en pratique d'ajouter une de ces deux instructions à la fin du sous-programme du lutin.

Tâche 4 : répéter ce déplacement de façon à ce que le lutin termine à l'angle en bas à droite de la scène.

Pour que le lutin atteigne l'angle en bas à droite de la scène, il faut qu'il effectue 19 fois le déplacement demandé à la tâche 3. Il suffit pour cela d'ajouter une instruction de boucle prédéfinie, disponible dans la catégorie « Contrôle ». On obtient alors le programme suivant :

Notes pédagogiques

• On peut faire afficher sur la scène la valeur de la variable prédéfinie « abscisse x », en cochant la case prévue devant le nom de la variable, tout en bas de la catégorie « Mouvement » : 

• Afin que les élèves visualisent mieux ce qui se produit lorsque la boucle s'effectue, on peut ajouter une petite pause avant l'instruction de déplacement, à l'aide de l'instruction « attendre… secondes » disponible dans la catégorie « Contrôle ».

Tâche 5 : faire en sorte qu'une « copie » du lutin « Dalles » reste à chaque position explorée

Les élèves peuvent proposer différents moyens de positionner des « copies » de la dalle aux emplacements qu'elle occupe successivement, y compris la première et la dernière position. Pour vérifier le programme, on peut placer à la fin une instruction « cacher », qui fera disparaître la dalle mobile et révèlera si une « copie » a bien été positionnée.

Une première façon de faire est d'utiliser l'instruction « estampiller », disponible dans la catégorie d'instructions « Stylo » : 

Cette instruction doit être placée dans la boucle, si l'on souhaite que l'estampillage se produise à chacune des positions successives de la dalle qui se déplace. Elle doit être associée à une instruction « effacer tout »

qui, placée en début de programme, permet que la scène soit « propre » à chaque lancement du programme. Si l'on conserve 19 répétitions comme à la tâche 4, il faut procéder à un estampillage supplémentaire, soit juste avant, soit juste après la boucle (selon que l'estampillage est placé après ou avant le déplacement, dans la boucle). Si l'on souhaite éviter cela, on peut augmenter à 20 le nombre de répétitions de la boucle, mais dans ce cas, il faut positionner obligatoirement l'estampillage avant le déplacement, comme proposé ci-dessous :

Une seconde façon de faire est d'utiliser les instructions de clonage, qui sont propres à *Scratch*. Ces instructions sont disponibles dans la catégorie « Contrôles ». Elles sont au nombre de trois :

L'instruction « quand je commence comme un clone » est une instruction qui permet, tout comme l'instruction « quand drapeau vert cliqué », de faire en sorte qu'un sous-programme soit déclenché par un événement, ici la création du clone. Le sous-programme en question s'applique alors au clone.
Dans le cas qui nous concerne, il s'agit de créer un clone à chaque itération de la boucle. Les subtilités sur le nombre de répétitions (19 ou 20) décrites pour l'option d'estampillage restent valables, et on obtient par exemple, si l'on fait 20 répétitions :

Notes pédagogiques

• Si l'on fait 20 répétitions de la boucle, on aperçoit furtivement la dalle mobile légèrement hors champ, avant sa disparition. Si les élèves trouvent que cela ne fait

pas très « pro », on peut leur proposer de rendre la dalle mobile fantomatique à l'aide de l'instruction suivante placée vers le début du programme :

• Toutefois, si les élèves ont utilisé des clones, il est alors nécessaire d'annuler cet effet graphique à la création du clone, pour que celui-ci soit effectivement visible :

• Plutôt que de demander 20 répétitions de la boucle, ce qui n'est correct que lorsque les dalles font 24 pixels de côté, on pourrait demander un nombre de répétitions qui dépende de la dimension des dalles, en l'occurrence, la largeur de l'écran en nombre de pixels (480) divisée par la dimension des dalles :

• De façon générale, c'est une bonne habitude à prendre que d'éviter au maximum les valeurs numériques dans les programmes, en tout cas pour les « constantes » qui ne sont pas si constantes que cela ! Ici, on peut conserver la valeur numérique 480 car elle ne peut pas changer (toutes les scènes *Scratch* font 480 pixels de largeur). Mais mieux vaut utiliser la variable Dimension-dalles que la valeur numérique 24.
• À ce stade de l'avancement du projet, les deux options – estampillage et utilisation de clones – semblent équivalentes. Elles ne le sont en réalité pas du tout : les dalles estampillées ne peuvent ni être déplacées, ni changer de costume, ni détecter la présence du lutin « Joueur ». Elles sont dorénavant passives. Tout ce que l'on peut faire, c'est les effacer d'un bloc ou détecter leur couleur. Au contraire, les clones peuvent avoir leurs sous-programmes propres, et peuvent donc jouer un rôle actif sur la scène. L'inconvénient des clones étant l'espace mémoire qu'ils occupent, au risque de ralentir tout le programme.
• L'enseignant doit discuter avec les élèves de l'option qui leur paraît la meilleure : souhaite-t-on que les dalles puissent se déplacer ? Qu'elles puissent interagir avec le joueur, et notamment détecter sa présence ? Plus généralement, aimerait-on qu'elles aient leurs propres sous-programmes ? Si la réponse de la classe est positive, l'utilisation des clones s'impose. Sinon, la fonction d'estampillage suffit.

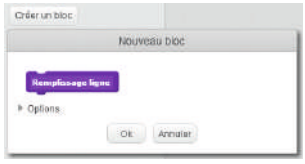
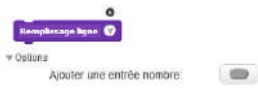
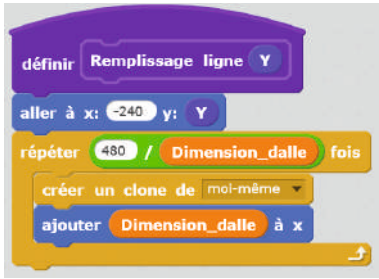
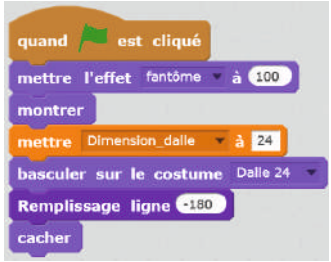
Tâche 6 : créer une fonction « Remplissage ligne »

Notes scientifiques

• Découper le travail de programmation en tâches élémentaires permet de tester les différentes parties du programme au fur et à mesure. C'est beaucoup plus facile, et plus sûr, que d'écrire un long programme et de le tester à la fin (dans ce cas, on obtient en général de nombreux bugs qu'il est très difficile de résoudre).
• Travailler à l'aide de fonctions permet, une fois que la fonction est écrite, de passer à autre chose sans avoir besoin de toucher ce morceau de programme.
• Attention : *Scratch* souffre d'une limitation importante : les fonctions ne peuvent pas retourner de valeur. Si l'on souhaite qu'une fonction renvoie un résultat, il faut avoir créé une variable *ad hoc* et faire en sorte que la fonction modifie cette valeur. En informatique, de tels sous-programmes (sans valeur de sortie) sont plutôt appelés

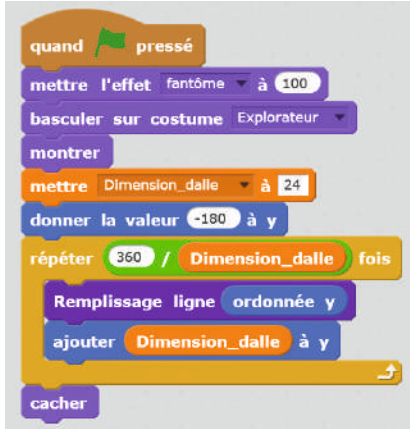
des « procédures » : nous employons le mot « fonction » (qui est un abus de langage) par souci de cohérence avec les programmes scolaires.

On souhaiterait que le remplissage d'une ligne par des copies de la dalle mobile puisse être répété pour chacune des lignes de la scène (15 lignes dans le cas de dalles de 24 pixels de côté). On se propose donc de placer les instructions permettant de remplir une ligne dans une fonction « remplissage ligne », qu'il suffira ensuite d'appeler 15 fois. La création de fonctions est disponible dans la catégorie « Ajouter blocs ». Concrètement, pour remplir la tâche 6, nous devons :

Créer un bloc que nous appelons par exemple « Remplissage ligne »	
Dans les options de création du bloc, ajouter une entrée numérique (l'ordonnée Y de la ligne à remplir) puis valider par « OK »	
Utiliser ce nouveau bloc pour créer la fonction de remplissage de ligne (en déplaçant certaines des instructions utilisées pour programmer la tâche 5)	
Appeler cette fonction dans le programme de la dalle mobile, avec une valeur au choix pour Y (on peut tester plusieurs valeurs, par exemple – 180 pour remplir la ligne du bas)	

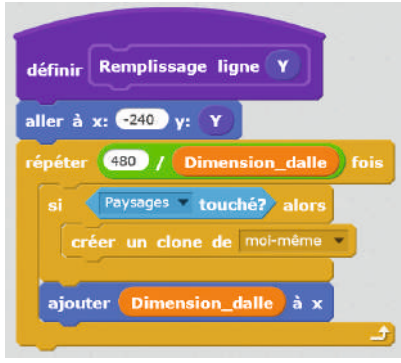
Tâche 7 : appeler la fonction « Remplissage ligne » autant de fois qu'il y a de lignes

Pour remplir cette tâche, il faut appeler la fonction « Remplissage ligne » 15 fois, avec les valeurs de Y suivantes : - 180, - 156, - 132… + 132, + 156. Autrement dit, avec des valeurs de Y espacées de 24. Pour cela, on initialise la position en ordonnées du lutin à la valeur – 180, et dans la boucle qui appelle la fonction « Remplissage ligne », on incrémente l'ordonnée du lutin de « Dimension_dalle ». On obtient alors, pour le programme principal du lutin « Dalles » :



Tâche 8 : ne faire poser un clone du lutin « Dalles » que s'il y a un élément de paysage à cet endroit.

Pour résoudre cette tâche, il suffit de placer l'instruction de création de clone dans une instruction conditionnelle : le clone ne doit être créé que si la dalle mobile touche un élément de paysage. La fonction « Remplissage ligne » devient alors :

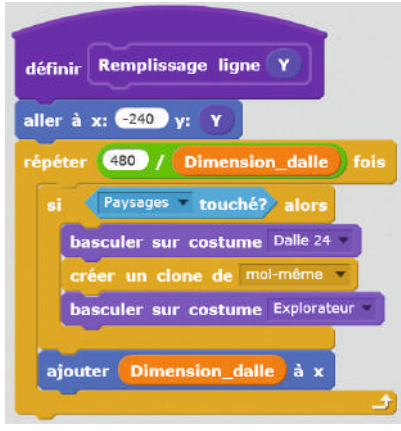


Tâche 9 : faire effectuer la détection du paysage par le costume « Explorateur » du lutin « Dalles ».

Cette tâche permet que la détection des éléments de paysage soit faite par le lutin « Dalles » ayant son costume « Explorateur ». Elle n'est pas absolument obligatoire, mais permet d'éviter des problèmes d'habillage du paysage dans le cas où le costume des dalles n'est pas tout à fait calé sur la dimension choisie. On obtient l'effet recherché avec une initialisation du costume et des changements de costume (avant et après la création du clone).

L'initialisation a lieu dans le programme principal à l'aide de l'instruction « Basculer sur costume… » :

Ces mêmes instructions encadrent la création du clone, dans la fonction « Remplissage ligne » :



Note pédagogique

Pour mieux visualiser ce qui se passe, on peut demander l'affichage du numéro du costume du lutin « Dalles », en cochant  en bas de liste de la catégorie

« Apparence ». Attention, pour que les changements de numéro soient visibles, il faut aussi insérer des instructions « attendre… secondes » après les instructions de changement de costume.

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.
Une démonstration montre :

1. au lancement du programme, le lutin « paysage » est affiché : chaque zone noire, non encore « habillée », correspond à un futur élément du décor.

2. le lutin « explorateur » parcourt l'écran, et « habille » les dalles noires à l'aide du costume prédéfini. La plateforme est ainsi créée.

Conclusion

L'enseignant revient tout d'abord sur la conclusion de la séance 2, qui se trouve renforcée par cette séance. Puis il guide la classe vers une conclusion concernant les fonctions en informatique :
Lorsqu'un même bloc d'instructions doit être utilisé plusieurs fois dans un programme, il est judicieux de l'intégrer dans une fonction. En *Scratch*, la création de fonctions se fait dans la catégorie « Ajouter blocs ».

Étape 5 – Faire reposer le lutin « Joueur » sur les plateformes

Discipline dominante	Mathématiques
Résumé	Les élèves créent leur propre feuille de route de programmation pour programmer la fonctionnalité de repos du lutin « Joueur » sur les plateformes. Différents algorithmes peuvent être envisagés, qui tous nécessitent d'utiliser des expressions conditionnelles. Certains algorithmes conduisent à des bugs. Pour résoudre les bugs, les élèves analysent leur programme pas à pas.
Notions nouvelles *(cf. scénario conceptuel, page 353)*	« Algorithmes » : • Un algorithme est une méthode permettant de résoudre un problème. Il se construit en combinant des instructions. • Plusieurs algorithmes peuvent permettre de résoudre un même problème. • Certaines boucles, dites « conditionnelles », sont répétées jusqu'à ce qu'une condition soit remplie. « Langages » : • Un bug est une erreur dans un programme. Un bug peut provenir d'une erreur de conception de l'algorithme. « Bonnes habitudes de programmation » : • Introduire des pauses dans l'exécution d'un programme et observer les valeurs des variables facilitent la résolution des bugs.
Matériel	Pour l'enseignant : • Fichier *Scratch* Platformer_V04_demo.

Situation déclenchante

L'enseignant se connecte sur son compte *Scratch* et ouvre le projet Platformer_V04_demo. C'est une copie conforme du programme achevé à la séance précédente, sauf que le paysage a été simplifié : il y a seulement une plateforme continue vers le bas de la scène. L'enseignant explique qu'il souhaiterait y intégrer le lutin « Joueur », qui se trouve dans la version V02. Pour cela, distinguons deux cas :

• Si l'enseignant et la classe travaillent avec *Scratch* en ligne, l'enseignant montre comment utiliser le « sac à dos » de *Scratch*. Il ouvre la version V02 du projet, puis il déploie le sac à dos, par un clic sur la petite flèche grise située sous la zone de programmation. Il y glisse le lutin « Joueur », qui sera dorénavant accessible à cet endroit. Puis il retourne sur la version V04 du projet et déploie le sac à dos. Le lutin « Joueur » s'y trouve effectivement, il suffit de le glisser dans la zone des lutins pour l'intégrer au projet, avec tous ses scripts et costumes. Les élèves s'installent aux ordinateurs et se connectent sur leur compte *Scratch*, puis effectuent immédiatement le même travail, y compris la simplification du paysage à une simple plateforme (un « trait » horizontal). Ils enregistrent le programme obtenu sous le nom Platformer_V04_nom_du_groupe.

• Si l'enseignant et la classe travaillent avec *Scratch* installé en local, l'enseignant montre comment exporter et importer un lutin. Il ouvre la version V02 du projet, fait un clic droit sur l'icône du lutin et choisit « enregistrer localement comme fichier ». Puis il ouvre la version V04 du projet et importe le lutin qu'il vient d'enregistrer localement, grâce à l'icône « Importer le lutin depuis un fichier ».

L'enseignant demande aux élèves d'exécuter le programme, et de se préparer à dire ce qu'ils en pensent. Les élèves soulèvent notamment le problème suivant : le lutin passe à travers les dalles !

L'objectif principal de cette séance est d'éviter ce problème, et donc de programmer cette fonctionnalité de la carte mentale :

• Avatar : se pose sur les plateformes.

• Elément de plateforme standard : est impénétrable par l'avatar (par le dessus)

Un autre problème peut être identifié par les élèves : le « Joueur » commence sa chute avant même que le paysage soit habillé de dalles. Si nécessaire, l'enseignant fait remarquer cela, et la classe ajoute les fonctionnalités correspondantes dans la carte mentale pour les programmer à la séance suivante :

• Niveau : informe l'avatar de sa mise en place.

• Avatar : apparaît et démarre une fois le paysage habillé de dalles.

Créer la feuille de route de programmation (par binômes)

L'enseignant propose aux élèves de créer eux-mêmes leur feuille de route. L'enseignant peut ponctuellement guider la réflexion d'un binôme, mais il n'intervient pas outre mesure. Les élèves passent à l'activité de programmation proprement dite au fur et à mesure qu'ils achèvent leur listing de tâches.

Mettre en œuvre la programmation (par binômes)

Les élèves se lancent, en autonomie, dans la réalisation des tâches qu'ils ont listées, tandis que l'enseignant passe de groupe en groupe pour échanger avec les élèves sur l'avancement de leur travail. L'enseignant rappelle que les élèves doivent sauvegarder régulièrement leur projet.

Notes pédagogiques

• La constitution du listing des tâches en autonomie par les groupes est une source de diversité dans les démarches, mais aussi une source de difficultés de programmation. C'est pourquoi cette séance se limite à la programmation d'une seule fonctionnalité d'apparence assez simple.

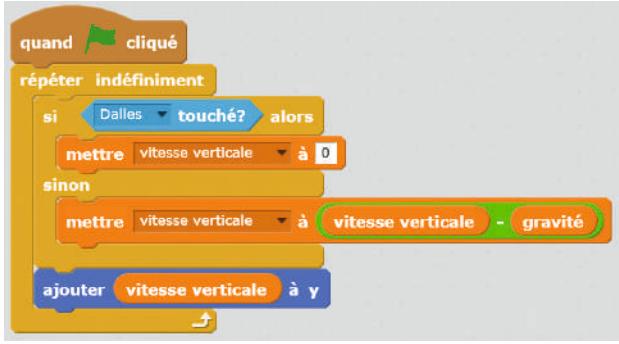
• L'enseignant peut suggérer aux élèves de s'inspirer des feuilles de route de programmation des séances précédentes.

• Il peut demander aux élèves de quelles variables et de quelles fonctions ils auront besoin.

Mise en commun

L'enseignant identifie un groupe qui se heurte à une difficulté courante et lui demande de partager son programme sur *Scratch*. Ce groupe se rend au tableau et montre le problème à la classe, pour la recherche collective d'une solution.

Prenons un exemple concret : un groupe d'élèves explique que le « Joueur », au lieu de stopper net sa chute lorsqu'il atteint une plateforme, s'enfonce de plusieurs pixels dans la plateforme et reste enfoncé ! Les élèves ont modifié le sous-programme qui contrôle la chute du joueur, en y intégrant une instruction conditionnelle, comme ceci :



Pourquoi ce problème ? Manifestement, le « Joueur » détecte bien la dalle, puisqu'il cesse sa

chute, mais au lieu de s'arrêter à la surface de la dalle, il s'y enfonce. Pour mieux voir ce qui se passe, on peut demander l'affichage sur la scène de la valeur de la variable « vitesse verticale » et/ou introduire une petite pause avant les déplacements verticaux, « attendre 0.1 secondes ». On constate alors que l'enfoncement dans la dalle est dû à la discrétisation des déplacements : le « Joueur » descend par exemple de 10 pixels d'un coup, et se retrouve enfoncé dans la dalle. Juste avant ce déplacement, il ne touchait pas la dalle, donc il est normal qu'il soit descendu de « vitesse verticale » pixels. Il touche dorénavant la dalle, donc sa vitesse verticale a pris une valeur nulle. Logiquement, il reste enfoncé. La première conclusion de ces observations est que le programme fait bien ce qu'on lui a demandé. Reste à trouver un moyen de faire remonter le « Joueur ».

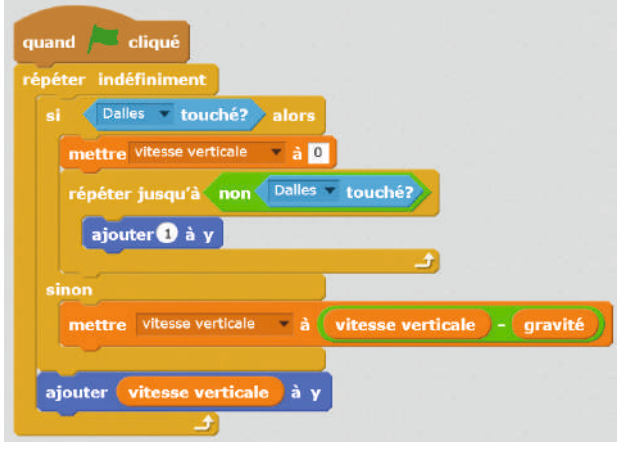
Note scientifique

Ce problème ne peut avoir lieu que si les élèves ont choisi de placer leurs « Dalles » par « clonage ». Dans le cas de la méthode par « estampillage », les collisions sont exclusivement gérées par détection de couleur. L'estampillage est la méthode la plus fluide pour gérer les collisions.

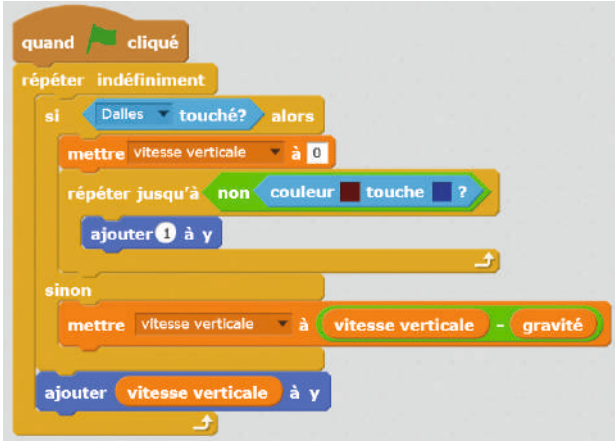
Plusieurs méthodes, ou algorithmes, peuvent être envisagés et testés collectivement, avec un guidage de l'enseignant si nécessaire :

• Lorsqu'une dalle est touchée, on peut mettre la vitesse verticale à une valeur positive (1 par exemple) au lieu de 0. Ainsi, le « Joueur » remonte. L'implémentation de cette proposition donne un caractère sautillant au « Joueur » : celui-ci effectue en permanence de petits bonds. On aime ou pas… c'est une affaire de goût. Mais en tout cas, cela posera problème pour programmer ultérieurement la fonctionnalité de saut depuis les plateformes, car le saut ne peut avoir lieu que lorsque le lutin touche le dessus d'une plateforme, ce qui n'est pas le cas lorsque le lutin vient de remonter.

• Lorsqu'une dalle est touchée, on peut mettre la vitesse verticale à 0 comme programmé initialement par les élèves, mais faire remonter le « Joueur » pixel par pixel en jouant sur la variable prédéfinie ordonnée Y, jusqu'à ce qu'il ne touche plus la dalle. Malheureusement, là encore, le rendu est sautillant… et on a comme pour l'algorithme précédent le problème des sauts.



• Un autre algorithme consiste à utiliser les capteurs de contacts entre couleurs. Il se trouve que le « Joueur » a une ligne de pixels noirs sur son pourtour, et des pixels rouge foncé au cœur. Si l'on reprend la proposition ci-dessus en conditionnant la remontée pixel par pixel au contact entre la couleur rouge foncé et la couleur de fond des dalles, le « Joueur » remontera jusqu'à avoir seulement la ligne de pixels noire enfoncée dans la dalle. Du coup, il touchera la dalle (donc sa vitesse verticale sera nulle), sans pour autant remonter davantage. Le rendu est cette fois-ci parfaitement stable.



Ce dernier algorithme constitue une correction possible de la tâche de programmation de la séance.
Si le temps le permet, un autre groupe vient présenter une difficulté à résoudre. L'approche pour la résolution collective est la même : analyse du programme, ajout de pauses, affichage des valeurs de certaines variables, propositions d'algorithmes à tester.
Un moment est prévu à l'issue de la mise en commun pour que les groupes puissent peaufiner leur programme et tenir compte de ce qui a été appris collectivement.
A l'écran, le jeu montre désormais un lutin capable de tomber puis se stabiliser s'il entre en contact avec un élément de la plateforme :

Conclusion

L'enseignant guide la classe vers la formulation de la conclusion suivante :

- *Un algorithme est une méthode permettant de résoudre un problème. Il se construit en combinant des instructions.*
- *Plusieurs algorithmes peuvent permettre de résoudre un même problème.*
- *Un bug est une erreur dans un programme. Un bug peut provenir d'une erreur de conception de l'algorithme.*
- *Introduire des pauses dans l'exécution d'un programme et observer les valeurs des variables facilitent la résolution des bugs.*

Étape 6 – Démarrer le jeu lorsque le paysage est prêt

Discipline dominante	Mathématiques
Résumé	Les élèves organisent dans le temps deux fonctionnalités (la mise en place des plateformes et le démarrage du jeu) en utilisant des événements de type « message » : le démarrage du jeu ne doit se faire qu'une fois le paysage installé.
Notions nouvelles *(cf. scénario conceptuel, page 353)*	« Langages » : • Le langage de programmation de *Scratch* est événementiel. La diffusion de messages est l'un des types d'événements utilisés. « 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	Pour l'enseignant : • Fichier *Scratch* Platformer_V04_correction. • Fichier *Scratch* Platformer_V05_correction.

Situation déclenchante

La classe se remémore l'état d'avancement du projet, correction de la séance précédente à l'appui. Le lutin « Joueur », soumis à une gravité virtuelle, est maintenant capable de se poser sur les plateformes, devenues « impénétrables » quand le lutin les aborde par le dessus. Le lutin se déplace vers la droite et vers la gauche en obéissant aux commandes des flèches du clavier.

L'enseignant propose de reprendre les programmes de la fois précédente et de régler le problème identifié : on voudrait que le « Joueur » commence sa chute une fois que le paysage est habillé de dalles. Il s'agit donc de programmer les fonctionnalités suivantes ajoutées la dernière fois à la carte mentale :

- Niveau : informe l'avatar de sa mise en place.
- Avatar : apparaît et démarre une fois le paysage habillé de dalles.

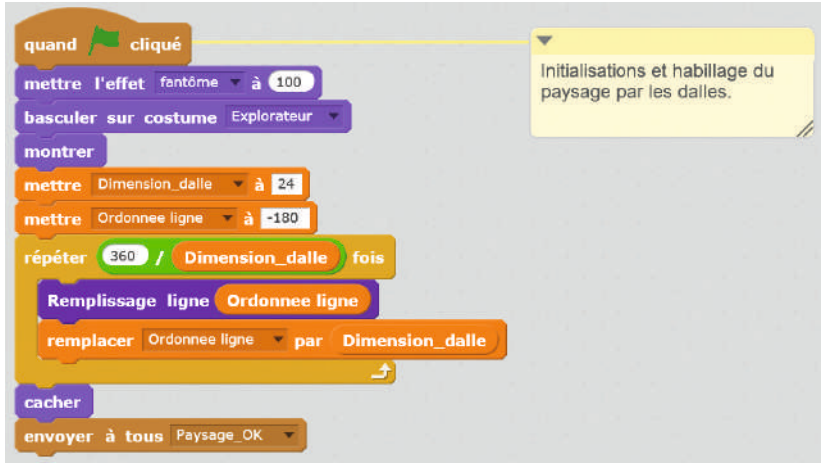
Faire en sorte que le jeu démarre une fois le paysage positionné (collectivement puis par binômes)

L'analyse collective du programme montre que de nombreux sous-programmes sont déclenchés par l'événement « Quand drapeau vert cliqué ». C'est notamment le cas du sous-programme du lutin « Dalles » qui habille le paysage par les dalles, et de tous les sous-programmes du lutin « Joueur ». Il est donc normal que le lutin « Joueur » apparaisse, réponde aux commandes de déplacement et tombe vers le bas de l'écran dès le clic sur le drapeau vert. Ce n'est pas conforme à ce que l'on souhaite, mais c'est ce que l'on a programmé. Il faudrait plutôt que les scripts du lutin « Joueur » (sauf celui qui gère les initialisations) soient déclenchés par un événement qui survient à la fin de l'habillage du paysage par les dalles. C'est tout l'intérêt du duo d'instructions suivant :

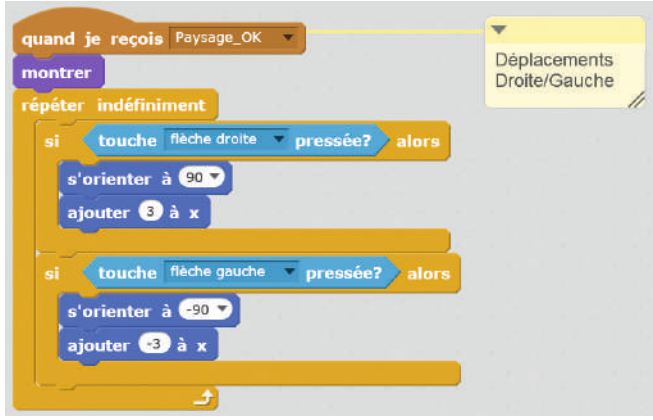
L'instruction « envoyer à tous… » permet de diffuser un message, ce qui constitue un événement. Ce message, puisqu'il est « envoyé à tous », sera reçu par tous les lutins, y compris celui dont un script commande la diffusion du message. La réception du message peut alors déclencher les sous-programmes que l'on souhaite lancer à ce moment-là.

Les élèves reprennent leur programme de la séance précédente et l'enregistrent sous un nouveau nom : Platformer_V05_nom_du_groupe. Ils cherchent, par binôme, comment utiliser les instructions d'envoi et de réception de message pour obtenir le résultat recherché dans le jeu de plateforme. (La mise en commun peut s'appuyer sur le fichier Platformer_V05_correction.)

On positionne l'instruction d'envoi de message à la fin du sous-programme d'habillage du paysage, comme ceci, en donnant un nom explicite au message, par exemple « Paysage_OK » :

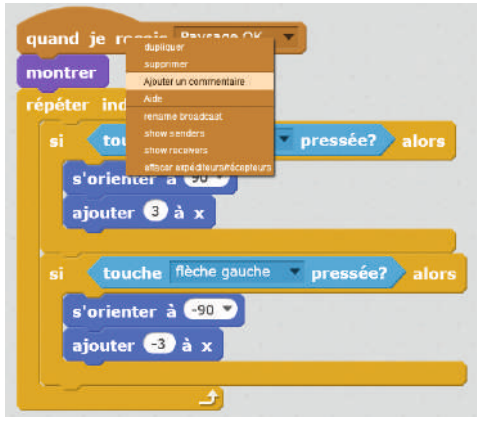


L'envoi de ce message signale ainsi que l'habillage du paysage est achevé. Sa réception peut alors déclencher les sous-programmes que l'on souhaite lancer à ce moment-là, c'est-à-dire tous les sous-programmes du lutin « Joueur » sauf celui qui gère les initialisations. Il suffit de remplacer l'instruction « Quand drapeau vert cliqué » par l'instruction « quand je reçois… », en tête des sous-programmes adéquats. Par exemple, le sous-programme qui contrôle les déplacements droite/gauche du lutin « Joueur » devient :



Activité : commenter son programme (par binômes)

L'enseignant explique que le programme commence à être complexe, et qu'il est temps d'y ajouter des commentaires pour mieux s'y retrouver. C'est aussi l'occasion de refaire le point sur l'avancement du projet. L'ajout de commentaire est accessible par un clic droit sur l'instruction à laquelle on souhaite ancrer le commentaire :



À ce stade, le lutin « Dalles » comporte :

• trois sous-programmes :

– un qui réalise les initialisations et l'habillage du paysage par les dalles.

– un qui assure l'habillage d'une ligne par les dalles.

– un qui gère l'apparence des dalles nouvellement posées (lesquelles sont des clones ou des estampillages de la dalle mobile initiale).

• deux costumes :

– un « explorateur » utilisé seulement pendant l'habillage

– un costume de dalle standard du paysage

Le lutin « Joueur » comporte :

• quatre sous-programmes :

– un qui réalise les initialisations.

– un qui gère les déplacements droite/gauche.

– un qui gère la chute du lutin et son repos sur les plateformes.

– un qui s'occupe de la disparition en bas de l'écran et de la réinitialisation

• un costume

Le lutin « Paysage » comporte :

• un sous-programme d'initialisation.

• un costume correspondant à une certaine disposition des dalles à placer.

Conclusion

L'enseignant échange avec la classe pour formuler une conclusion au sujet des bonnes habitudes de programmation :

• *Pour faciliter la maintenance d'un programme et son partage avec autrui, il peut être utile de placer des commentaires qui explicitent le rôle des différents sous-programmes.*

• *Pour la même raison, il faut veiller à donner des noms explicites aux variables et aux fonctions. Un programme bien écrit ne nécessite pas beaucoup de commentaires.*

Étape 7 – Faire sauter le lutin « Joueur » depuis les plateformes

Discipline dominante	Mathématiques
Résumé	Les élèves programment le comportement de saut du lutin « Joueur » depuis les plateformes. Plusieurs algorithmes peuvent être envisagés, mais certains sont plus commodes que d'autres à mettre en œuvre.
Notions nouvelles *(cf. scénario conceptuel, page 353)*	Idem séances précédentes.
Matériel	Pour l'enseignant : • Fichier *Scratch* Platformer_V06_demo. • Fichier *Scratch* Platformer_V06_correction.

Situation déclenchante

L'enseignant lance le programme Platformer_V06_demo, qui correspond à une correction de la version V05 avec, en plus de la plateforme continue, des plateformes vers lesquelles sauter. Il propose aux élèves de s'attaquer aujourd'hui à la fonctionnalité de saut du lutin « Joueur » :

• Avatar : saute si on appuie sur la flèche haute, à condition qu'il puisse prendre appui sur une plateforme.

• Elément de plateforme : sert d'appui pour les sauts.

L'organisation sera la même qu'à la séance 4 :

Créer la feuille de route de programmation (par binômes)

L'enseignant propose aux élèves de créer eux-mêmes leur feuille de route. Il passe de groupe en groupe et peut guider la réflexion, mais n'intervient pas outre mesure. Les élèves passent à l'activité de programmation proprement dite au fur et à mesure qu'ils achèvent leur listing de tâches.

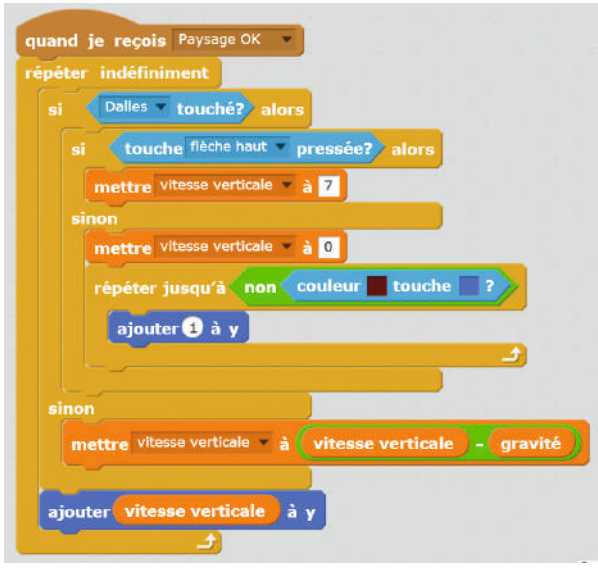
Mettre en œuvre la programmation (par binômes)

Les élèves reprennent leur programme de la séance précédente et l'enregistrent sous un nouveau nom : Platformer_V06_nom_du_groupe. Ils se lancent, en autonomie, dans la réalisation des tâches qu'ils ont listées, tandis que l'enseignant passe de groupe en groupe pour échanger avec les élèves sur l'avancement de leur travail. L'enseignant rappelle que les élèves doivent sauvegarder régulièrement leur projet.

Mise en commun

L'enseignant identifie un groupe qui se heurte à une difficulté typique et lui demande de montrer son programme sur *Scratch*. Ce groupe se rend au tableau et montre le problème à la classe, pour la recherche collective d'une solution.

Par exemple, un groupe a programmé la fonctionnalité recherchée comme illustré ci-dessous, en modifiant le sous-programme du lutin « Joueur » qui permettait précédemment de faire remonter le lutin à la surface des plateformes :



Le saut en lui-même fonctionne, et ce seulement si le lutin « Joueur » est effectivement en contact avec une plateforme (si on presse sur la flèche haute alors que le lutin est déjà en l'air, il ne saute pas de nouveau, comme on le souhaite). Un problème persiste toutefois : lorsque le « Joueur » saute en étant sous une plateforme, dès lors qu'il touche cette plateforme, il monte pixel par pixel et traverse en quelque sorte le plafond !

L'enseignant demande aux élèves de détailler ce qui se passe si le lutin touche une dalle par le dessous. La classe constate qu'en suivant rigoureusement le programme, on anticipe l'effet obtenu (et non souhaité) : cet effet correspond parfaitement à ce qui a été programmé. Simplement, le groupe n'avait pas anticipé que la condition « Dalles touché ? » est vraie que le lutin « Joueur » touche les dalles par le dessus ou par le dessous. Plusieurs algorithmes peuvent être proposés par les élèves pour distinguer les contacts au plafond et au plancher :

• une idée est de se baser sur la valeur de la vitesse du « Joueur » au moment du contact avec une dalle, pour savoir s'il heurte un plafond ou le dessus d'une plateforme. Certains groupes peuvent essayer de mettre en œuvre une solution de ce type.

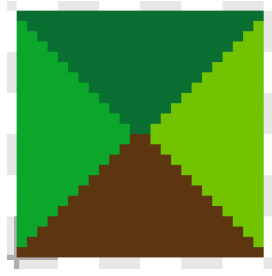
• une autre idée est de distinguer la face des dalles que le « Joueur » touche d'après leur couleur. Cette façon de faire contraint le dessin des dalles, mais elle semble très pratique. On peut d'ailleurs distinguer en même temps la face gauche et la face droite des dalles, ce qui facilitera la programmation ultérieure des rebonds sur les murs.

Notes pédagogiques

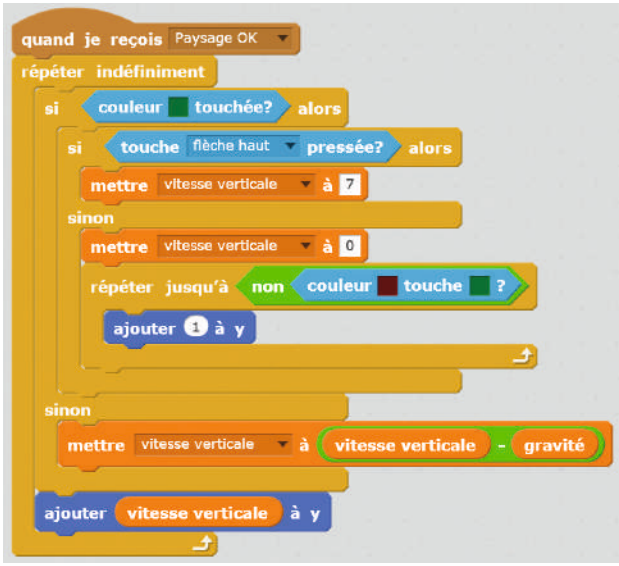
• Pour information, la première idée est difficile à mettre en œuvre de façon satisfaisante. Si des élèves se lancent dans l'aventure, ils rencontreront des difficultés qui renforceront la conclusion de séance sur le choix d'algorithme.

• Nous recommandons de leur permettre plus tard de récupérer le lutin « Dalle » et les éléments de programme d'autres groupes qui auront mis en œuvre la deuxième idée, dans un esprit de coopération à l'échelle de la classe.

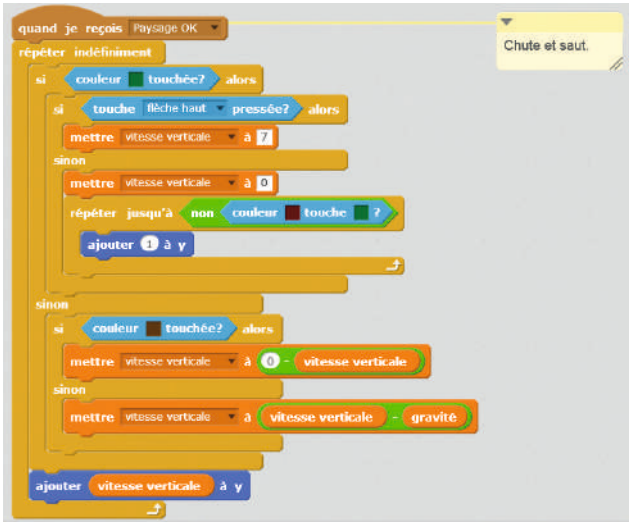
L'implémentation de la seconde idée nécessite de revoir le costume des dalles, par exemple comme ceci en format 24x24, avec des couleurs différentes sur chacune des 4 faces :



Le sous-programme du lutin « Joueur », qui contrôle maintenant sa chute et ses sauts, devient à une étape intermédiaire :



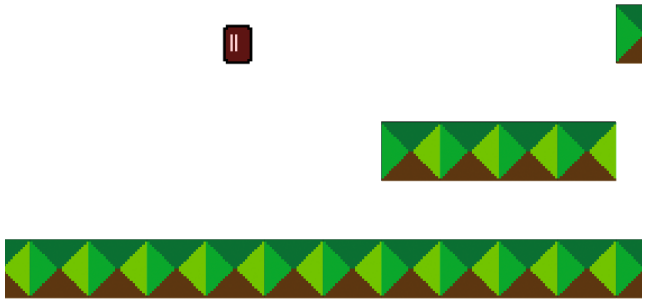
Pour que le rebond sur les plafonds soit effectif, il faut en plus faire en sorte que la vitesse verticale change de signe lorsque le lutin touche la couleur marron des plafonds :



Conclusion

La classe reprend la conclusion de la séance 4, qui s'applique tout à fait à la présente séance et s'en trouve renforcée.

A l'écran, le rendu du jeu ressemble désormais à cela (la capture d'écran a été réalisée pendant un saut) :



Étape 8 – Gérer le rebond sur les bords verticaux des plateformes

Discipline dominante	Mathématiques
Résumé	Les élèves programment le comportement de rebond du lutin « Joueur » sur les rebords verticaux des plateformes. Puis ils créent des fonctions pour chacun des comportements déjà programmés.
Notions nouvelles *(cf. scénario conceptuel, page 353)*	Idem séances précédentes.
Matériel	Pour l'enseignant : • Fichier *Scratch* Platformer_V07_demo.

Situation déclenchante

L'enseignant lance le programme Platformer_V07_demo, qui correspond au travail issu des séances 1 à 6, avec dans le paysage des éléments de plateforme formant un mur. L'enseignant met en évidence des dysfonctionnements, si ceux-ci n'ont pas déjà été signalés par les élèves :

• lorsque le lutin « Joueur » saute et touche une plateforme par un bord vertical, il rentre dans la plateforme et remonte jusqu'à être posé sur elle.

• lorsque le lutin « Joueur » rencontre un mur de dalles par le côté, il le traverse.

Il s'agit donc de programmer la fonctionnalité suivante de la carte mentale :

• Avatar : rebondit sur les murs.

• Elément de plateforme standard : est impénétrable par l'avatar (par les côtés)

Les élèves vont devoir régler ces problèmes, après la production d'une feuille de route de programmation collective.

Gérer le rebond sur le bord vertical des plateformes (collectivement puis par binômes)

L'enseignant demande aux élèves de réfléchir à la façon dont ils vont gérer le rebond du lutin « Joueur » sur les bordures verticales des plateformes. Pour cela, il leur conseille d'analyser comment ils ont programmé le repos du lutin « Joueur » sur les plateformes et son rebond sur les plafonds, et de se demander si, dans l'état actuel des choses, ils peuvent procéder de même pour les bordures verticales. Une mise en commun aura lieu après quelques minutes de réflexion des binômes.

Certains groupes d'élèves auront certainement remarqué que le repos sur les plateformes et le rebond sur les plafonds sont obtenus en manipulant la variable « vitesse verticale ». Si ce n'est pas le cas, l'enseignant guide la classe vers cette observation :

• pour sauter, on met la valeur de la variable « vitesse verticale » à 7 : 

• pour reposer sur une plateforme, on met cette valeur à 0.

• pour rebondir sur un plafond, on change le signe de cette valeur :

• pour accélérer vers le bas, on y ajoute la valeur de la variable « gravité » (le signe négatif vient de la convention choisie de prendre une valeur positive de la gravité alors qu'elle entraîne le « Joueur » vers le bas et doit donc diminuer, via « vitesse verticale », son ordonnée) :

Puis on met à jour la position selon l'axe des ordonnées par l'instruction :

Or, il n'y a pas pour le moment de variable « vitesse horizontale » : les déplacements droite/gauche sont obtenus en modifiant directement la valeur de l'abscisse « x » du lutin « Joueur ». C'est d'ailleurs pour cela que le lutin a des trajectoires peu réalistes lorsqu'il saute : si on cesse d'appuyer sur la flèche droite ou sur la flèche gauche, en cours de saut, le lutin cesse brutalement son déplacement droite/gauche. Il peut aussi changer de direction en plein saut… comme dans les dessins animés (et comme dans le jeu de plateforme montré à la séance 1), mais pas comme dans la réalité !

L'enseignant encourage les élèves à introduire une nouvelle variable « vitesse horizontale », à l'utiliser pour gérer les déplacements droite/gauche (ce qui supposera de modifier un sous-programme existant du lutin « Joueur ») puis à l'utiliser pour gérer les rebonds sur les bordures verticales. Cela correspond à la feuille de route de programmation suivante, construite collectivement (sauf la tâche 6 que les élèves n'anticiperont certainement pas) :

Fonctionnalité du programme	Nature des tâches à réaliser	Difficulté (cf. page 69)
1 – Avatar : se déplace vers la droite si on presse la flèche de droite et vers la gauche si on presse la flèche de gauche.	**Tâche 1 :** créer une variable « vitesse horizontale » visible par tous les lutins. Initialiser cette variable à la valeur 0.	
	Tâche 2 : modifier le sous-programme « déplacement droite/gauche » du lutin « Joueur » de façon à ce que les déplacements droite/gauche soient obtenus en fixant d'abord la valeur de la variable « vitesse horizontale », puis en ajoutant « variable horizontale » à la variable « x ».	
	Tâche 3 (facultative) : faire en sorte que les flèches droite/gauche n'aient d'effet que lorsque le lutin « Joueur » touche une plateforme par le dessus.	
	Tâche 4 (facultative) : pour prendre en compte les frottements sur le sol, faire en sorte que le lutin « Joueur » arrête progressivement son déplacement droite/gauche quand on cesse d'appuyer sur les touches flèche droite/gauche, s'il repose sur une plateforme.	
2 – Lutin « Joueur » : rebondit sur les bords verticaux des plateformes	**Tâche 5 :** changer le signe de la vitesse horizontale lorsque le lutin rencontre un bord vertical de plateforme, pour produire un effet de rebond.	
	Tâche 6 : empêcher l'entrée du lutin dans les plateformes lorsqu'il touche à la fois un dessus de plateforme et un côté de plateforme (cas de dalles superposées formant un mur).	

Après avoir créé une copie V07 du fichier issue de la séance précédente, les élèves se lancent, en autonomie, dans la réalisation des tâches de la feuille de route, tandis que l'enseignant passe de groupe en groupe pour échanger avec les élèves sur l'avancement de leur travail. L'enseignant rappelle que les élèves doivent sauvegarder régulièrement leur projet.

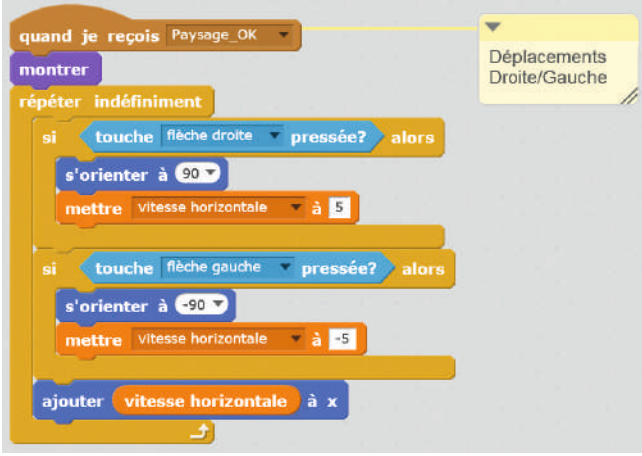
Ci-dessous, des éléments de correction pour les tâches 1 à 6 :

Tâche 1 : créer et initialiser une variable « vitesse horizontale »

La création et l'initialisation de variable a déjà été décrite à l'Étape 2 (correction des tâches 1 et 3). Il est judicieux de placer l'initialisation de la variable « vitesse horizontale » au même endroit que celle de la variable « vitesse verticale », c'est-à-dire dans le sous-programme des initialisations du lutin « Joueur ».

Tâche 2 : gérer les déplacements horizontaux du Joueur

On utilise la variable nouvellement créée « vitesse horizontale » pour commander les déplacements droite/gauche du lutin « Joueur ». Les modifications du sous-programme écrit à la séance 1 sont assez mineures :

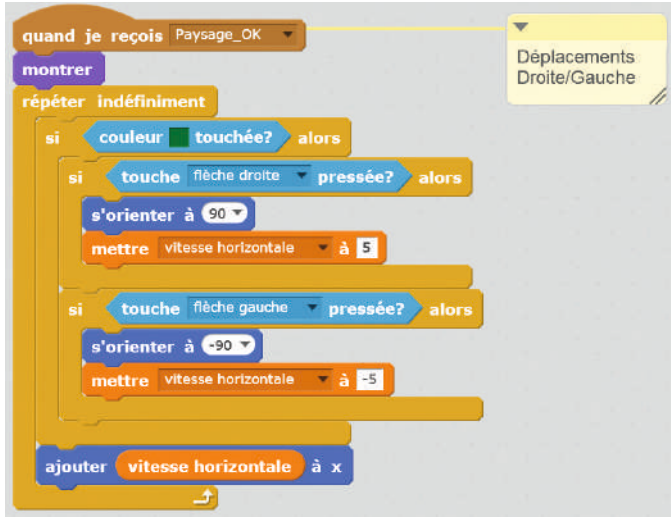


Pour le moment, les déplacements peuvent être commandés aussi bien lorsque le lutin est « en l'air » que lorsqu'il repose sur une plateforme. De plus, son déplacement ne cesse pas lorsqu'on arrête de presser les touches flèche droite/gauche.

Tâche 3 (facultative) : empêcher les changements de direction pendant les sauts

Lorsque le lutin « Joueur » repose sur le dessus d'une plateforme, il touche la couleur verte spécifique du dessus des plateformes. Pour résoudre la tâche 3, on peut donc ajouter cette condition de contact dans le sous-programme qui contrôle les déplacements droite/gauche du lutin « Joueur », comme ceci :

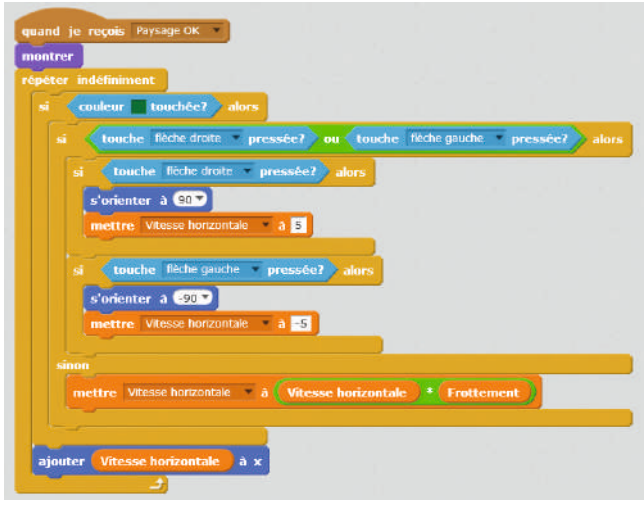
La mise à jour de la position du lutin selon l'axe des abscisses doit être effectuée que la condition de contact soit vraie ou fausse. L'instruction « ajouter vitesse horizontale à x » ne doit donc pas dépendre de cette condition : on la place hors des mâchoires du « si… alors ».



Tâche 4 (facultative) : prendre en compte les frottements sur le sol

La résolution de la tâche 4 suppose de rapprocher la valeur de la variable « vitesse horizontale » de zéro si le lutin « Joueur » touche le dessus de plateforme (donc la couleur verte correspondante) et si les flèches droite/gauche ne sont pas pressées. Pour obtenir ce résultat, on peut multiplier la valeur de la vitesse par un « coefficient de frottement » plus petit que 1, ce qui suppose au préalable de créer une variable « frottement » et de l'initialiser (par exemple à la valeur 0.5).

On doit également agencer des instructions conditionnelles « si… alors » et « si… alors… sinon », ce qui est conceptuellement difficile. La proposition suivante n'est pas la plus courte possible, mais elle est la plus simple à comprendre :



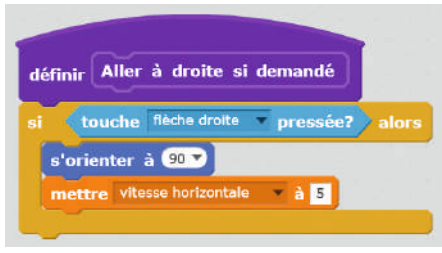
Si on affiche la valeur de la variable « Vitesse horizontale » sur la scène, on s'aperçoit qu'en phase de ralentissement du lutin, cette variable prend des valeurs très petites : 0,00000… Ce n'est pas très malin : autant assimiler cette valeur à 0 dès lors qu'elle est faible, par exemple comme ceci :

Tâche 5 : rebondir sur les murs

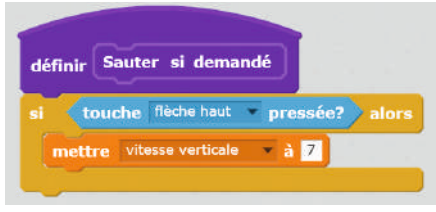
Pour programmer cette tâche, on est amené à tester le contact du lutin avec la couleur des bordures verticales des dalles et à changer le signe de la vitesse horizontale seulement si ce contact a lieu, par exemple ainsi :

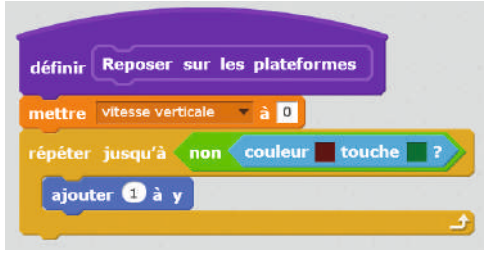
Mais où positionner ces instructions dans le programme du lutin « Joueur » ? La plupart des façons de faire que peuvent tester les élèves donnent des résultats peu satisfaisants, car il y a des conflits entre contacts sur/sous/par le côté des plateformes. De plus, ce n'est pas très commode de réorganiser les sous-programmes pour tester l'effet de différents agencements : on s'y perd ! Moralité : il est grand temps de créer des fonctions correspondant à chacun de ces blocs d'instructions, comme on l'a appris en Étape 4. On pourra ainsi, sans

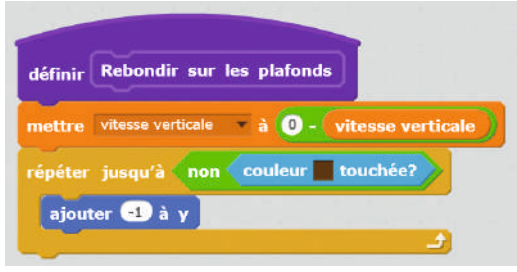
modifier les fonctions, changer la façon dont on les appelle, et choisir l'option qui correspond le mieux à ce que l'on recherche.
On peut ainsi créer deux fonctions pour les déplacements vers la droite et vers la gauche (seule la fonction permettant les déplacements vers la droite est illustrée ici, l'autre se construit de la même façon) :

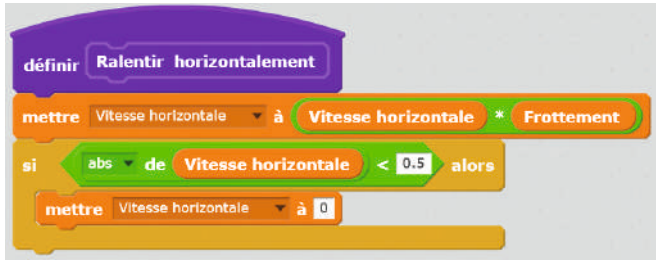


Et aussi créer une fonction pour le contrôle des sauts, une autre pour le repos sur les plateformes, une autre encore pour le rebond sur les plafonds et pour le ralentissement progressif du lutin :

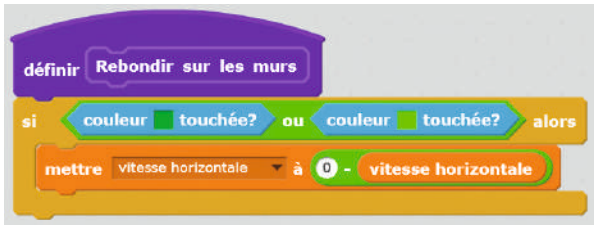




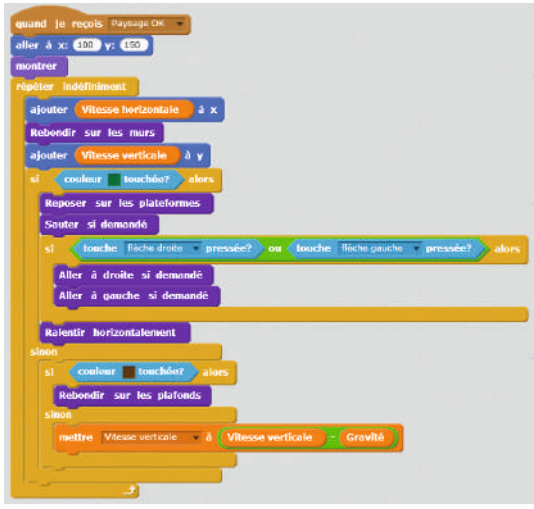




Enfin, une fonction s'impose pour les rebonds sur les murs :

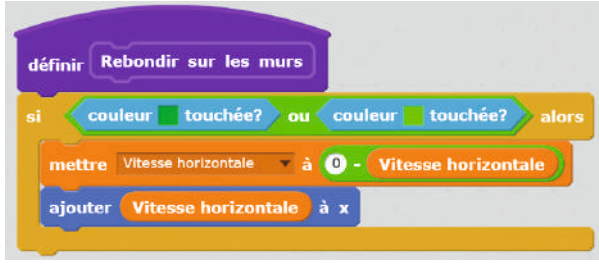


Reste à trouver comment agencer efficacement l'appel de ces fonctions. Mieux vaut certainement effacer les deux sous-programmes qui géraient jusqu'ici les déplacements horizontaux et verticaux, et repartir à zéro, « *from* Scratch » comme on dit en anglais !
Le programme suivant fonctionne plutôt bien, mais ce n'est certainement pas la seule possibilité :



Tâche 6 : améliorer la non-pénétration des murs

Un gros problème persiste toutefois : le lutin peut traverser les murs, s'il se déplace latéralement sur une plateforme. C'est normal si on analyse le programme dans le détail ! Comme d'ordinaire, le programme fait ce qui est demandé…
La modification suivante de la fonction de rebond sur les murs permet de rendre les murs réellement impénétrables :



Étape 9 – Créer deux niveaux et les enchaîner

Discipline dominante	Mathématiques
Résumé	Les élèves créent deux « niveaux » de jeu et les coordonnent. Cela suppose de créer un élément de plateforme correspondant à la sortie d'un niveau. C'est l'occasion de remplacer toutes les valeurs numériques du programme par des variables initialisées dans une fonction appropriée.
Notions nouvelles *(cf. scénario conceptuel, page 353)*	Idem séances précédentes.
Matériel	Pour l'enseignant : • Fichier *Scratch* Platformer_V08_correction.

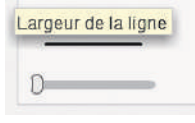
Situation déclenchante

L'enseignant annonce aux élèves qu'ils vont pouvoir, aujourd'hui, créer le paysage de deux « niveaux » du jeu. Ils devront ensuite faire en sorte que le lutin « Joueur » débute le jeu au niveau 1, puis passe au niveau 2 lorsqu'il aura atteint la sortie du niveau 1.

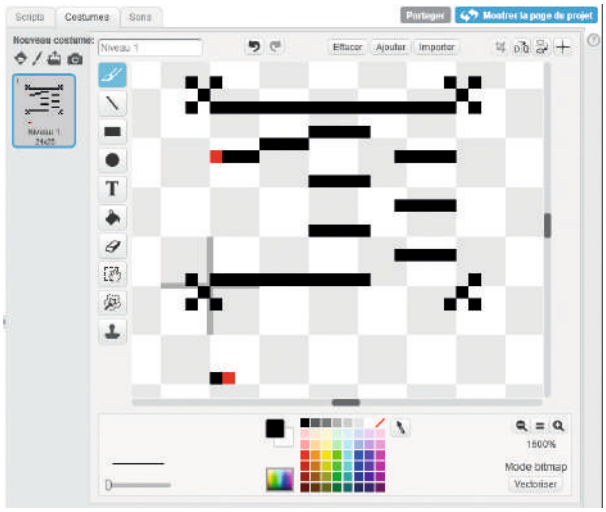
Les binômes ont toute autonomie pour gérer leur travail (l'enseignant ne demande même pas de faire une feuille de route de programmation, afin de tester si les élèves ont bien intégré l'importance de cette étape). L'enseignant passe de binôme en binôme au fur et à mesure de l'avancement du travail. Il y aura autant de propositions de paysages que de binômes, et plusieurs façons d'assurer le changement de niveau. Une solution possible a été programmée dans le fichier Platformer_V08_correction. Les éléments de description de cette solution ci-dessous permettent de discuter de certaines difficultés qui peuvent être rencontrées par les élèves et de signaler quelques bonnes habitudes de programmation.

Création de deux paysages comprenant une sortie

Se rendre dans l'onglet « Costumes » du lutin « Paysages » pour modifier le paysage existant. Supprimer tout d'abord les pixels déjà positionnés (sauf les repères d'angles qui sont hors champ), à l'aide de l'outil « Gomme ». Puis sélectionner l'outil « Pinceau » et réduire au maximum la largeur de la ligne de contour :



Positionner d'un clic un pixel noir à chaque endroit où devra se trouver une dalle standard. Choisir une couleur pour la position de la future dalle de sortie, par exemple rouge vif, et positionner un pixel de cette couleur là où devra se trouver la dalle de sortie. Renommer le niveau « Niveau 1 ».



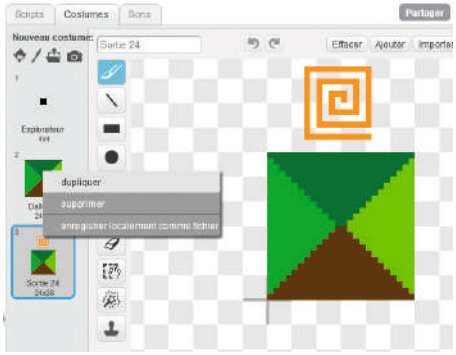
Note pédagogique

Les deux pixels hors champ (un noir et un rouge) dans le bas de la zone de dessin permettent de conserver la mémoire des couleurs utilisées, et de les sélectionner facilement à l'aide de la pipette « Choisir la couleur ».

Pour créer un second paysage, dupliquer le paysage existant après un clic droit sur lui. Les repères d'angle et les couleurs utilisables sont ainsi disponibles dans le nouveau paysage. Puis effacer ce qui doit l'être et dessiner le nouveau paysage. Le renommer « Niveau 2 ».

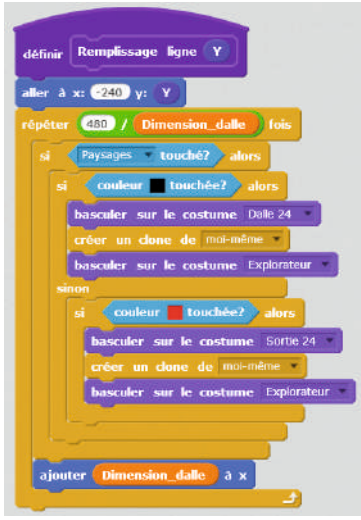
Création d'une dalle « sortie de niveau »

Se rendre dans l'onglet « Costumes » du lutin « Dalles » puis dupliquer la dalle standard après un clic droit sur cette dalle. Renommer la nouvelle dalle d'un nom explicite comme « Sortie », et ajouter un élément de dessin qui signale la sortie. Si cet élément de dessin est d'une nouvelle couleur, présente nulle part ailleurs dans le paysage, cela permettra que le changement de niveau soit déclenché par le contact entre le lutin et cette couleur.

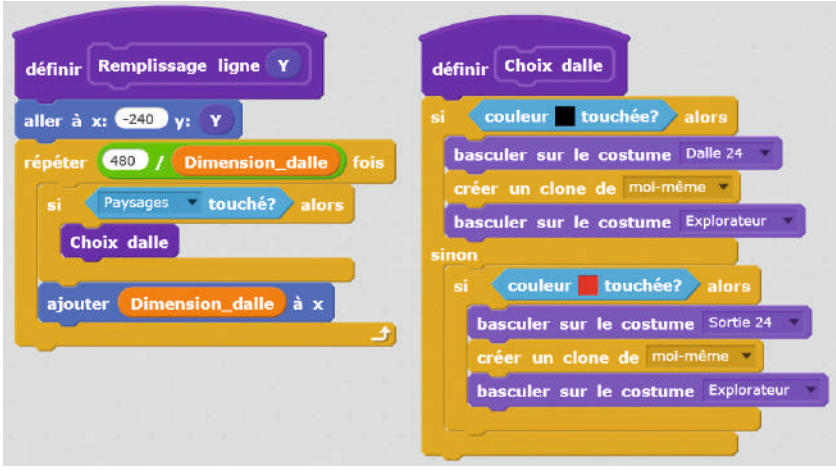


Habillage du paysage par les deux types de dalles

Lors de l'habillage du paysage, il faut maintenant programmer le choix de la dalle adéquate, selon la couleur du pixel rencontré dans le paysage : une dalle standard pour un pixel noir et une dalle de sortie pour un pixel rouge. La modification suivante de la fonction « Remplissage ligne » du lutin « Dalles » permet d'obtenir ce résultat :



Afin que le nom de chaque fonction corresponde bien à ce qu'elle fait effectivement, ce qui est une bonne pratique de programmation, on peut scinder la fonction en deux : la fonction « Remplissage de ligne » se limite à cela, et appelle la fonction « Choix dalle » :

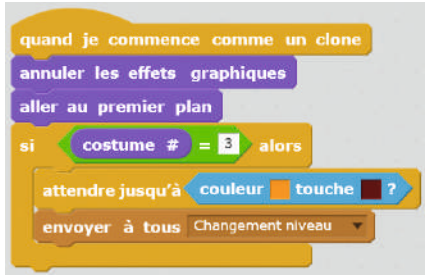


Détection du contact entre le lutin « Joueur » et la sortie

On peut programmer cette fonctionnalité soit dans le programme du lutin « Joueur », soit dans le programme du lutin « Dalles », ce qui correspond à deux algorithmes différents.

Première option, dans le programme du lutin « Joueur », il suffit de détecter le contact entre ce lutin et la couleur spécifique de la sortie (dans notre cas, une nuance orange), et d'envoyer un message lorsque ce contact a lieu :

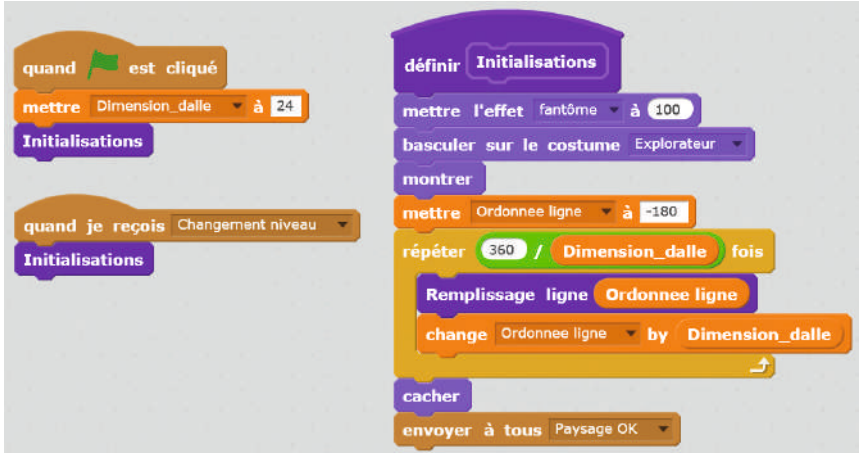
Deuxième option, dans le programme du lutin « Dalles », il faut détecter le contact entre la spirale orange de la dalle de sortie et le lutin « Joueur ». Pour éviter que les dizaines de dalles potentiellement présentes sur la scène n'aient ce même script qui tourne tout le temps du jeu, on peut conditionner l'attente du contact au fait que le clone corresponde à la dalle de sortie (costume numéro 3) :



La zone de programmation du lutin « Joueur » est beaucoup plus chargée que celle du lutin « Dalles », ce qui peut faire pencher la balance vers l'option 2.

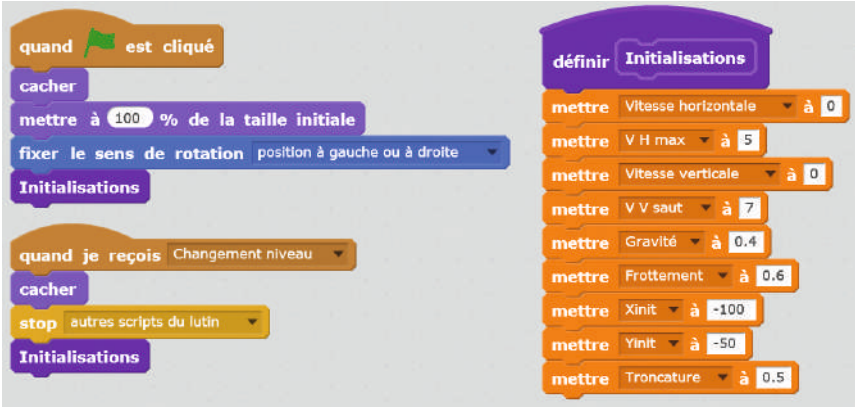
Initialisations au lancement du programme et aux changements de niveau

Nous avons déjà programmé des initialisations qui s'effectuent au clic sur le drapeau vert, mais il faut maintenant programmer des réinitialisations qui s'opèrent au changement de niveau. Afin d'éviter de recopier plusieurs fois les mêmes blocs d'instructions, on peut créer une fonction d'initialisation pour les lutins qui le nécessitent, et appeler cette fonction deux fois : au lancement du programme et à la réception du message « Changement de niveau ».
Par exemple, dans le cas du lutin « Dalles » :



En fait d'initialisation, la fonction assure l'habillage du paysage. On peut donc préférer la nommer par exemple « Habillage paysage ».
Il faut également penser à effacer les dalles lors du changement de niveau, que ce soit par « Effacer tout » dans le cas de l'estampillage ou « Supprimer ce clone » dans le cas des clones :

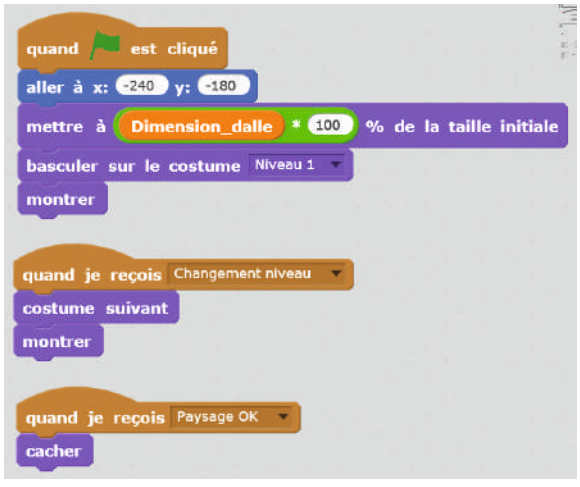
Pour le lutin « Joueur », la création d'une fonction d'initialisation est l'occasion de remplacer par des variables toutes les valeurs numériques du programme susceptibles de faire l'objet d'ajustements, ce qui est aussi une bonne habitude de programmation, comme nous l'avons régulièrement rappelé :



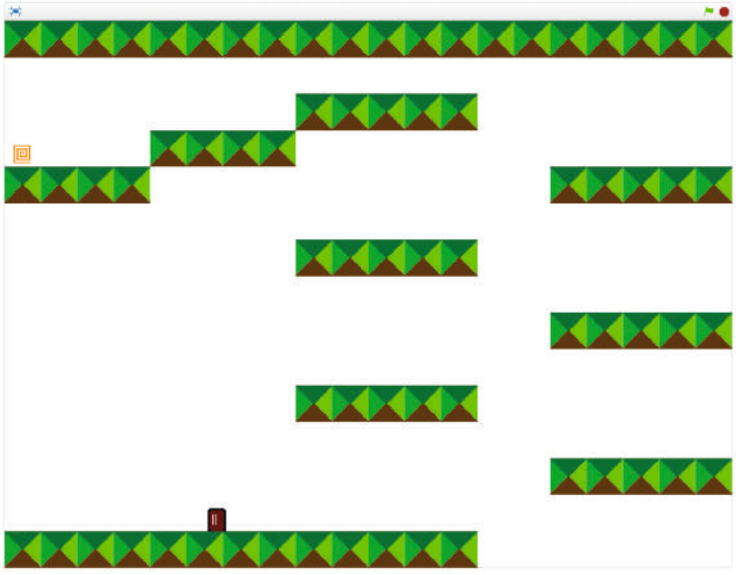
« Vitesse horizontale » et « Vitesse verticale » sont les deux composantes de vitesse du lutin « Joueur » à un instant donné, « V H max » est la vitesse horizontale que prend le lutin lorsqu'on appuie sur une flèche droite/gauche, « V V saut » est la vitesse verticale que prend le lutin en début de saut, « Gravité » indique de combien décrémenter la vitesse verticale à chaque itération lorsque le lutin est en chute, « Frottement » contrôle le ralentissement du lutin lorsqu'il se déplace latéralement sur une plateforme, « Xinit » et « Yinit » donnent la position du lutin au lancement du jeu et au changement de niveau, « Troncature » indique en dessous de quelle valeur on considère que la vitesse horizontale du lutin est nulle (utilisation dans la fonction « Ralentir horizontalement »).

Noter que jusque-là on avait des valeurs numériques « 0.5 » qui trainaient un peu partout dans le programme, et qui n'avaient absolument pas la même signification. C'est une source classique de bug.

Le programme du lutin « Paysage » nécessite également quelques ajustements : il doit prendre son costume « Niveau 1 » au lancement du programme puis passer au costume suivant à chaque changement de niveau :



Le jeu comporte désormais 2 niveaux :



Niveau 1

Niveau 2

Conclusion

La classe reprend la carte mentale du projet, et constate que toutes les fonctionnalités sont désormais en place dans leur jeu. Les élèves sont maintenant libres d'aller plus loin et peuvent donner libre cours à leur imagination : création de nouvelles dalles (raccourci, piège, catapulte…) et de niveaux supplémentaires, limitation du temps autorisé pour chaque niveau, cohabitation à l'écran de 2 joueurs concurrents, ajout de ressources à récolter avant de pouvoir quitter le niveau, ajout d'ennemis qui se déplacent… les possibilités sont infinies, dans le respect des bonnes pratiques de programmation :

- création d'une feuille de route de programmation
- création de fonctions et de variables autant que nécessaire
- tests réguliers du programme

La conception du dessin des niveaux (ou « *level design* ») pourra être réalisée en arts plastiques.