

## Feuille d'activité :

Titre : Créer un jeu de labyrinthe avec le BBC Micro:bit

### Objectif :

- Comprendre le concept de perception en intelligence artificielle en utilisant l'accéléromètre du Micro:bit pour contrôler un personnage de jeu dans un labyrinthe.
- Développer des compétences en codage pour créer un jeu interactif.

### Matériel :

- BBC Micro:bit avec câble USB.
- Ordinateur avec l'environnement de codage MakeCode installé.

### Instructions :

#### Étape 1 : Introduction

- Connectez le Micro:bit à votre ordinateur via USB.
- Ouvrez l'environnement de codage MakeCode dans un navigateur web.

#### Étape 2 : Créer un nouveau projet

- Démarrez un nouveau projet MakeCode pour votre jeu de labyrinthe.

#### Étape 3 : Comprendre la perception

- La perception en intelligence artificielle implique la détection et la compréhension de l'environnement. Dans cette activité, vous utiliserez l'accéléromètre du Micro:bit pour détecter les mouvements d'inclinaison, permettant au personnage du jeu de se déplacer dans le labyrinthe.

#### Étape 4 : Concevoir et personnaliser votre labyrinthe

Dans cette étape, vous concevrez le labyrinthe en utilisant la grille fournie dans l'environnement de codage MakeCode. Vous pouvez personnaliser le labyrinthe en ajoutant des murs et des passages ouverts pour créer un puzzle stimulant. Le labyrinthe doit avoir un point de départ clair et une destination, qui est le point final du jeu. Rendez-le Challengeant : Tenez compte du niveau de difficulté de votre labyrinthe. Le chemin du début à la fin doit présenter un défi pour le joueur. Le joueur doit naviguer à travers le labyrinthe en inclinant le Micro:bit tout **en** évitant les murs pour

atteindre la destination.

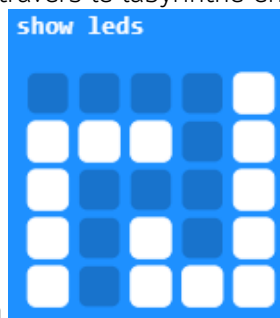


Figure 1 Maze layout

Créez la disposition du labyrinthe en utilisant la grille fournie dans MakeCode. Utilisez les formes de blocs pour représenter les murs, les chemins ouverts, le début ( $x = 0, y = 0$ ) et la fin ( $x = 1, y = 4$ ) du labyrinthe. Personnalisez la disposition pour correspondre au labyrinthe fourni ou créez votre propre conception de labyrinthe. Vous pouvez également créer plusieurs niveaux pour le joueur.

L'emplacement du joueur sur l'écran du Micro:bit sera indiqué par une LED rouge clignotante. Les LED rouges fixes symboliseront les murs, tandis que les LED éteintes représenteront les chemins du labyrinthe.

Les coordonnées sont utilisées pour manipuler efficacement les LED du Micro:bit ! Les coordonnées  $x$  vont de 0 à gauche à 4 à droite, tandis que les coordonnées  $y$  vont de 0 en haut à 4 en bas. Par conséquent, la LED en haut à gauche est désignée par  $x=0, y=0$ , et de même, la LED en bas à droite est représentée par  $x=4, y=4$ .

### Étape 5 : Programmer le jeu

Utilisez les blocs suivants pour programmer le comportement du jeu :

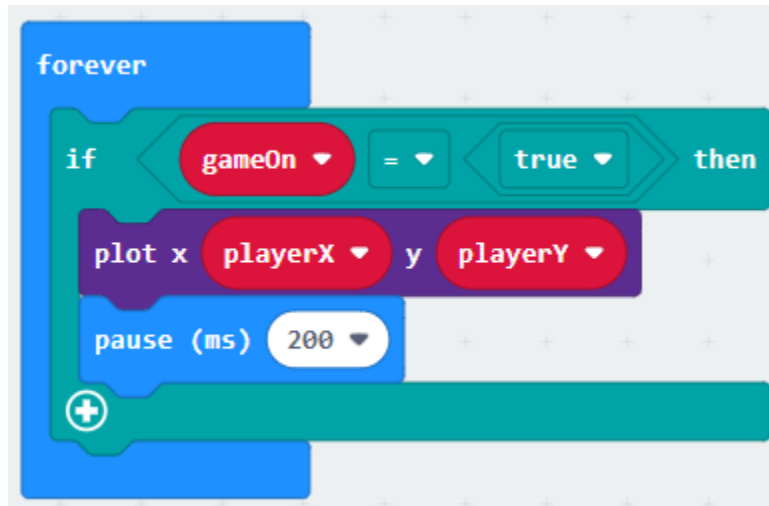


Tout d'abord, nous devons créer quelques variables. Rappelons que les variables fonctionnent comme des conteneurs qui stockent des informations. Dans ce cas, deux variables sont nécessaires pour surveiller l'emplacement du joueur. L'une est désignée pour enregistrer la position  $x$  du joueur, tandis que l'autre est dédiée au suivi de la position  $y$  du joueur.

De plus, nous avons besoin d'une variable pour surveiller le niveau du labyrinthe, permettant la possibilité de plusieurs niveaux. Une autre variable est nécessaire pour suivre le statut du jeu, indiquant s'il est actif ou s'il est terminé.

Les valeurs initiales sont définies pour commencer au niveau 1, et `gameOn` est initialisé comme `True`. Cela est dû au fait que, lors de l'allumage du Micro:bit, l'intention est de démarrer le jeu immédiatement. Bien que le point de départ pour l'emplacement du joueur puisse être choisi arbitrairement, il doit être rappelé ultérieurement lors de la configuration du niveau du labyrinthe pour

garantir que le joueur ne commence pas à l'intérieur d'un mur. Pour cet exemple, le joueur est initié à  $x=0$  et  $y=0$ .



Maintenant que les variables initiales sont en place, assurons-nous que notre joueur est affiché sur l'écran du Micro:bit !

Pour obtenir un effet de clignotement distinctif pour le joueur, nous utiliserons le bloc 'plot x y' alterné avec le bloc 'pause' à l'intérieur d'une boucle sans fin ('forever loop'). L'intention est que le joueur clignote continuellement. Lorsque les murs du labyrinthe sont introduits, le Micro:bit écrasera le joueur à chaque fois qu'il dessinera les murs. En incorporant un bloc pause ici, nous nous assurons que le joueur ne sera pas immédiatement remplacé, ce qui produira l'effet de clignotement désiré.

L'utilisation des variables `playerX` et `playerY` créées précédemment est cruciale. Pourquoi ? Si des valeurs numériques étaient directement entrées ici, cela limiterait la flexibilité pour déplacer le joueur. L'utilisation de variables nous permet de modifier les valeurs de `playerX` et `playerY`, permettant à la boucle sans fin de tracer la nouvelle position du joueur.

Il est essentiel de noter que le bloc pause fonctionne en millisecondes (par exemple, 200 ms = 0,2 secondes), et la vitesse de clignotement peut être personnalisée en ajustant la durée de la pause.



Maintenant, nous devons configurer les mouvements du joueur (gauche, droite, haut et bas). Nous utiliserons les deux boutons intégrés et la fonction de balayage du logo.

Nous définirons le geste de balayage vers le haut pour se déplacer vers le haut, le geste de balayage vers le bas pour se déplacer vers le bas, le bouton A pour se déplacer vers la gauche, et le bouton B pour se déplacer vers la droite.

Pour cela, nous utilisons des instructions if. Ces instructions évaluent si une condition est vraie ; si c'est le cas, tous les blocs à l'intérieur du bloc if sont exécutés. Lorsque nous intégrons une instruction if à l'intérieur d'une boucle sans fin, nous vérifions continuellement si la condition est vraie.

Pour le mouvement du joueur, nous modifions les variables playerX ou playerY. Il est crucial de se rappeler que diminuer ou augmenter playerX entraîne respectivement un déplacement vers la gauche ou la droite, tandis que diminuer ou augmenter playerY entraîne respectivement un déplacement vers le haut ou vers le bas. Étant donné que nous traçons constamment l'emplacement du joueur en utilisant ces variables, tout changement reflète automatiquement la nouvelle position du joueur.

Il convient de noter qu'une brève pause de 300ms est ajoutée après chaque pression de bouton. Cela empêche le Micro:bit de déplacer le joueur à travers plusieurs espaces rapidement à chaque pression de bouton, car le code s'exécute rapidement sans la pause.



5x5.

Nous poursuivons avec la création du niveau du labyrinthe. Plusieurs tâches requièrent de l'attention : premièrement, afficher les murs du labyrinthe sur l'écran LED ; deuxièmement, vérifier en permanence si le joueur entre en collision avec un mur (ce qui indique la fin du jeu) ; et troisièmement, évaluer en permanence si le joueur réussit à terminer le niveau du labyrinthe.

Une boucle sans fin est utilisée. À l'intérieur de cette boucle, une instruction 'if' est utilisée pour vérifier si la variable de niveau équivaut à 1. Par conséquent, ce segment de code ne s'exécutera que lorsque la variable de niveau équivaut à 1. Si nous voulons ajouter plus de niveaux, nous devons nous assurer que cette variable change en conséquence.

À l'intérieur de l'instruction 'if', les murs du labyrinthe sont affichés à l'aide du bloc 'show leds'. Les LEDs sont illuminées pour représenter les murs, tandis que les LEDs éteintes représentent les chemins du labyrinthe. Il faut faire attention à ce que la position de départ du joueur, définie précédemment à  $x=0, y=0$ , ne coïncide pas avec un mur du labyrinthe.

La tâche suivante consiste à vérifier si le joueur entre en collision avec un mur. Cela est réalisé grâce à des instructions 'if' supplémentaires, vérifiant si les variables playerX et playerY correspondent aux coordonnées d'un mur dans la grille LED

Enfin, le code vérifie si le joueur réussit à naviguer à travers le labyrinthe. Dans cet exemple, la fin du labyrinthe se trouve à  $x=1, y=4$ . Si ces conditions sont remplies, une mélodie de réussite est jouée, la position du joueur est réinitialisée au début du labyrinthe, et un visage souriant apparaît sur le Micro:bit. Si nous avons ajouté des niveaux supplémentaires, nous devons également incrémenter la variable de niveau de 1.



En cas de fin de partie, nous devons mettre en œuvre une action déclenchée par la variable 'gameOn' indiquant une collision avec un mur.

À l'intérieur d'une boucle sans fin, une instruction 'if' est utilisée pour évaluer la valeur de la variable 'gameOn'. Si elle équivaut à 'false', le code de fin de partie est exécuté.

Dans ce cas, une mélodie triste est jouée en arrière-plan, le 'niveau' est réinitialisé, la LED du joueur est éteinte, un visage triste est affiché, et le jeu recommence depuis le début.

Étape 6 : Testez votre jeu

- Testez votre jeu en guidant le personnage à travers le labyrinthe. Tout fonctionne-t-il correctement ?

Étape 7 : Jouez et partagez

- Partagez votre jeu de labyrinthe avec d'autres. Chargez-le sur votre Micro:bit et lancez un défi à vos amis pour qu'ils réussissent à compléter le labyrinthe.

Ce projet permet aux élèves de découvrir le concept de perception en intelligence artificielle en utilisant l'accéléromètre du Micro:bit pour contrôler un personnage dans un labyrinthe. Les élèves peuvent



concevoir

leurs propres labyrinthes, créer différents niveaux de difficulté et partager leurs jeux avec leurs pairs pour plus de plaisir et d'apprentissage.



**Cofinancé par  
l'Union européenne**

Financé par l'Union européenne. Les points de vue et avis exprimés n'engagent toutefois que leur(s) auteur(s) et ne reflètent pas nécessairement ceux de l'Union européenne ou de l'Agence exécutive européenne pour l'éducation et la culture (EACEA). Ni l'Union européenne ni l'EACEA ne sauraient en être tenues pour responsables..