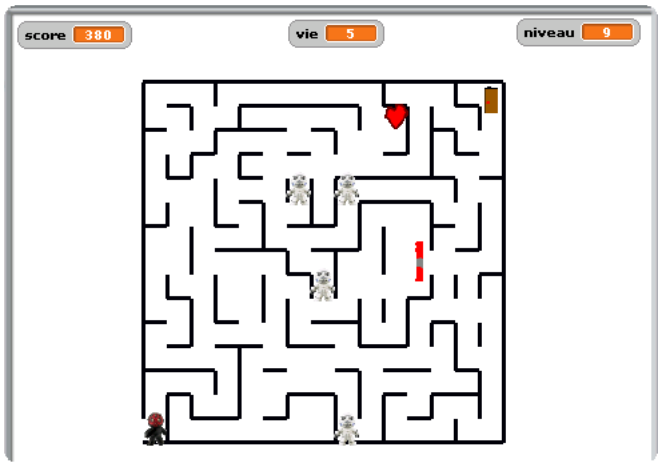
Faire des labyrinthes dans scratch

http://www.derepas.com/scratch

Nous allons voir comment faire des jeux nécessitant la création de labyrinthes. Voici un exemple de résultat final, un labyrinthe automatiquement généré avec personnages à l'intérieur :



Une version en ligne est également disponible :

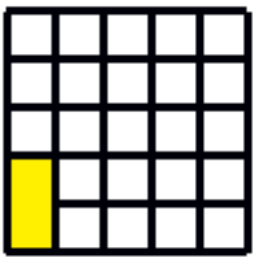
http://scratch.mit.edu/projects/fabricederepas/2387249

1 Comment faire des labyrinthes

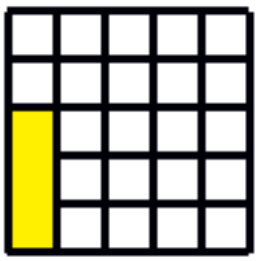
1.1 Le principe général

Voici le principe (on appelle ça un algorithme) que j'ai utilisé pour générer des labyrinthes au hasard.

L'idée simple qui consiste à mettre des murs au hasard ne marche pas, car parfois cela rend certaines parties du labyrinthe inaccessibles.



Étape 1



Étape 2

Étape 3

Étape 4

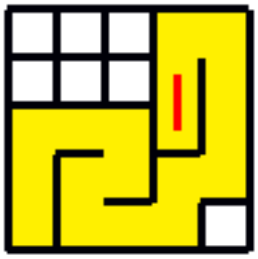
Étape 5

Étape 6

Étape 7

Étape 13

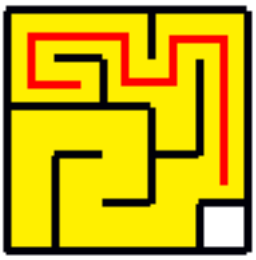
Étape 16



Étape 17

Étape 18

Étape 19



Étape 23

Étape 24

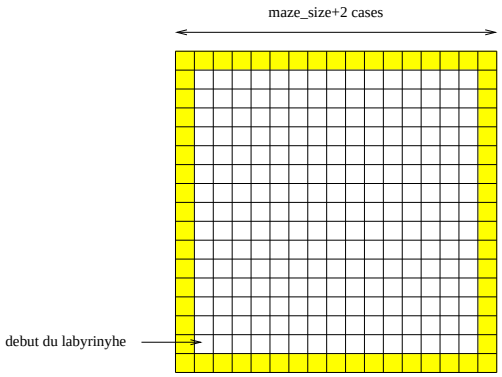
On commence avec des murs partouts (comme dans l'étape 1). Puis partant de la case en bas à gauche on choisit au hasard une case qui n'est pas encore coloriée en jaune on va dedans en faisant sauter le mur. On continue comme cela jusque dans l'étape 17 où on se retrouve dans une situation où tous les voisins immédiats sont déjà coloriés en jaune. Alors on revient en arrière jusqu'à trouver une case avec un voisin blanc (ici on recule d'une case suivant le trait rouge) et on recommence comme avant. Cela nous mène dans l'étape 23 où à nouveau il n'y a plus de cases blanches autour, on revient donc en arrière (suivant le trait rouge) et du coup on peut finir de colorier le labyrinthe comme à l'étape 24.

Ainsi nous disposons d'un labyrinthe où nous sommes certains que toutes les cases sont accessibles depuis n'importe quelle autre case, ce qui aurait été dur en mettant des murs au hasard.

1.2 Adaptation dans Scratch

Maintenant nous allons coder dans Scratch le principe que nous venons de voir pour faire des labyrinthes.

Si on essaye avec directement le principe proposé dans la section précédente alors il y a plein de cas particulier à gérer : par exemple si on est sur le bord supérieur alors il n'y a pas de voisins au dessus, si on est dans un coin il n'y a que deux voisins. Pour faire simple et éviter ces cas particulier on agrandit le labyrinthe d'une case dans toute les diretions et on les colories en jaune pour être sur de ne pas aller dedans. Si `maze_size` est la taille du labyrinthe on obtient donc un carré de (`maze_size` + 2) cases de côté :

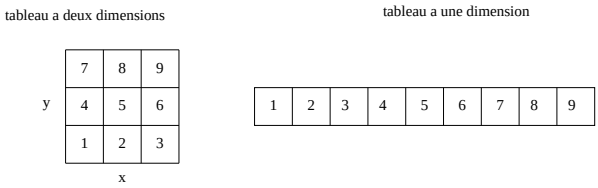


En effet il n'y a plus de cases particulières car pour toutes les cases blanches il y a toujours des voisins au dessus, en dessous, à droite et à gauche.

Voici les tableaux que nous allons faire pour conserver en mémoire le labyrinthe :

- `cell` (cela veut dire cellule ou case en français) : ce tableau va conserver la couleur des cases (jaune ou blanche sur le dessin).
- `north` (cela veut dire "nord" en français) : ce tableau va dire si on peut aller dans la case au dessus (au nord).
- `south` (cela veut dire "sud" en français) : ce tableau va dire si on peut aller dans la case en dessous (au sud).
- `east` (cela veut dire "est" en français) :ce tableau va dire si on peut aller dans la case à droite (à l'est).
- `west` (cela veut dire "ouest" en français) :ce tableau va dire si on peut aller dans la case à gauche (à l'ouest).

Avant de voir comment manipuler les tableaux `north`, `south`, `east` et `west`, voyons tout d'abord comment rentrer le labyrinthe dans un tableau. Le labyrinthe est une structure à deux dimensions : pour nous repérer il faut une coordonnée horizontale (notée x en général) et une coordonnée verticale (notée y en général) :



Ainsi dans la figure ci-dessus la case (1,1) a le chiffre 1, la case (3,1) (x vaut 3 et y vaut 1) a le chiffre 3, la case (3,2) (x vaut 3 y vaut 2) a le chiffre 6. Pour passer cela dans un tableau ligne, il suffit de mettre toutes les lignes les unes au bout des autres. Mais du coup étant donné une case (x,y) dans le tableau d'origine pour avoir sa position dans le tableau à une dimension il faut regarder la case $3*y+x$. On va faire la même chose avec les tableaux `north`, `south`, `east` et `west`. Il faut juste remplacer 3 par `maze_size` $+2$ qui est la taille du labyrinthe.

Ainsi pour une case (x,y) dans le labyrinthe, on peut aller au dessus (c'est à dire il n'y a pas de mur vers le haut) si la case $y \times (\texttt{maze\_size}+2)+x$ de `north` vaut 1, sinon il y a un mur. Pareil pour `south`, `east` et `west` :

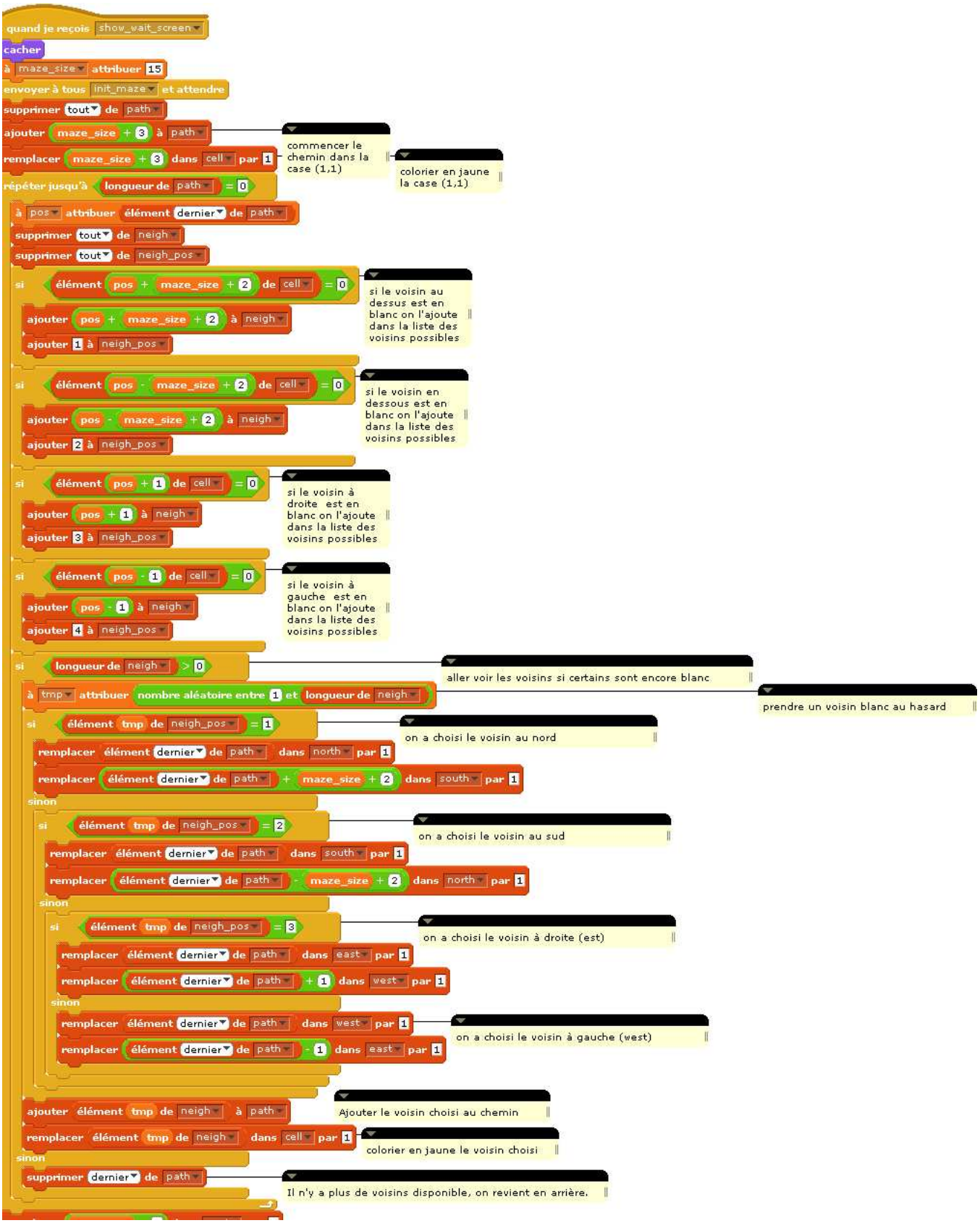
- la valeur de la case $y \times (\texttt{maze\_size}+2)+x$ de `south` est 1 si on peut aller en dessous de (x,y), et zéro sinon,
- la valeur de la case $y \times (\texttt{maze\_size}+2)+x$ de `east` est 1 si on peut aller à droite de (x,y), et zéro sinon,
- la valeur de la case $y \times (\texttt{maze\_size}+2)+x$ de `west` est 1 si on peut aller à gauche de (x,y), et zéro sinon,

Pour `cell`, la case de $y \times (\texttt{maze\_size}+2)+x$ de `cell` vaut 1 si la case est jaune et zéro sinon.

A partir de là on en déduit la phase d'initialisation du labyrinthe, c'est à dire des murs partout et les cases au bord en jaune :

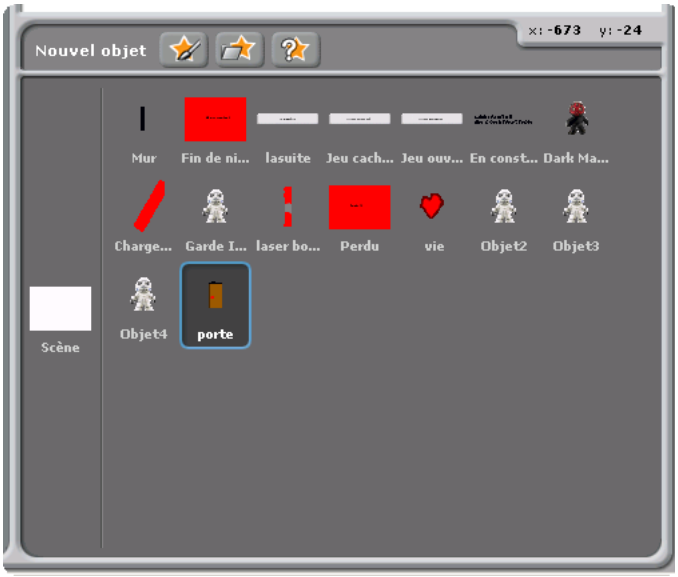

quand je reçois init_maze
initialise les tableaux
supprimer tout de cell
supprimer tout de south
supprimer tout de east
supprimer tout de north
supprimer tout de west
à i attribuer 0
répéter maze_size + 2 * maze_size + 2 fois
insérer 0 à i de cell
insérer 0 à i de north
insérer 0 à i de south
insérer 0 à i de east
insérer 0 à i de west
à i attribuer i + 1
à i attribuer 0
répéter maze_size + 2 fois
mettre les cases du bord en jaune
remplacer i dans cell par 1
remplacer i * maze_size + 2 dans cell par 1
remplacer i * maze_size + 2 - 1 dans cell par 1
remplacer maze_size + 2 * maze_size + 2 - i dans cell par 1
à i attribuer i + 1

À partir de là on peut mettre en oeuvre les principes décrits précédemment. Le chemin est gardé dans la list `path` (cela veut dire chemin en français).



2 Le jeu

2.1 Les objets



- mur: un mur du labyrinthe (c'est juste un trait droit),
- fin de niveau : un écran pour dire que l'on passe au niveau suivant,
- lasuite : un bouton pour passer à la suite,
- jeu caché : un bouton pour choisir le mode où on découvre le labyrinthe au fur et à mesure,
- jeu ouvert : un bouton pour choisir le mode où tout le labyrinthe est affiché au départ,
- en construction : affiche un texte pour faire patienter pendant la génération du labyrinthe,
- dark maul : le personnage que l'on déplace,
- chargement : la barre qui tourne pendant que le labyrinthe est en construction,
- garde impérial : un adversaire dans le labyrinthre,
- laser bonus : l'épée qui nous permet de tuer les méchants dans le labyrinthe
- perdu : l'écran qui s'affiche quand on a perdu,
- vie : permet de regagner une vie,
- objet2 à objet4 : d'autres méchants dans le labyrinthe,
- porte : la porte qui permet de passer au niveau d'après.

2.2 Évènements

Pour que les objets communiquent entre eux ils peuvent s'envoyer des sortes de signaux:

Les signaux s'envoient avec *"envoyer à tous"*. Pour qu'un objet reçoive un signal il suffit de mettre dans le script un *"quand je reçois"* comme illustré plus haut. Pour créer de nouveaux évènements, c'est facile, il suffit de cliquer sur *"nouveau"* dans le menu déroulant dans *"envoyer à tous"*.

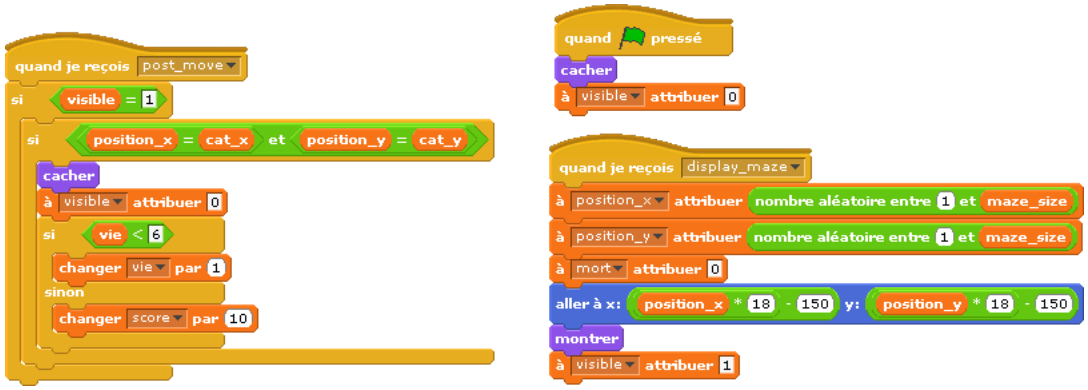
Voici les évèments qui sont présent dans le jeu :

- new_game (en français nouveau jeu) : le début du jeu. Quand on appuie sur le drapeau vert dans la *"Scène"* il y a une instruction *"envoyer à tous new_game"*. Cela affiche "jeu ouvert" et "jeu caché".
- show_wait_screen (en français montre l'écran d'attente) : cet évènement est déclenché quand on clique sur "jeu ouvert" ou "jeu caché". Il lance le cacul du labyrinthe et l'affichage de la barre qui tourne.
- display_maze (en français affiche le labyrinthe) : cet évèment est lancé à la fin du calcul du labyrinthe (dans l'objet mur). Il lance l'affichage du labyrinthe.
- post_move (en français après le mouvement) est un évènement qui est appelé à chaque fois que Dark Maul bouge. Cet évènement est récupéré dans les ennemis qui alors effectuent un mouvement.

Passons maintenant en revue quelques objets du jeu.

2.3 Vie

C'est un objet simple qui augmente notre vie quand on le touche dans le labyrinthe. Voici les instructions :



Cet objet dispose de trois variables locales :

- `visible` pour savoir si l'objet est visible ou non,
- `position_x` un nombre qui indique la case horizontale dans le labyrinthe,
- `position_y` un nombre qui indique la case verticale dans le labyrinthe.

L'objet est initialement caché quand on le jeu est lancé par un appui sur le drapeau vert. On a une variable :`visible` locale à l'objet `vie` pour conserver en mémoire si l'objet est visible ou non. On attribue zéro à `visible` initialement.

Au moment où on reçoit display_maze là on attribue à `position_x` et `position_y` un nombre aléatoire entre 1 et `maze_size` qui est la variable représentant la taille du labyrinthe (ici 15 puisque le labyrinthe est un carré de 15 sur 15 cases).

Pour afficher le coeur au bon endroite connaissant `position_x` et `position_y` il faut faire un petit calcul : chaque case du labyrinthe fait 18 pixels de large et le coin inférieur gauche est à $(-150, -150)$. Du coup l'endroit pour afficher le coeur sur l'écran de Scratch est:

$$(18 \times \texttt{position\_x} - 150 \quad , \quad 18 \times \texttt{position\_y} - 150)$$

d'où le "aller à x:position_x*18-150 y:position_y*18-150" en bleu. On peut ensuite rendre l'objet visible et mettre la valeur de la variable `visible` à 1.

Au moment où on reçoit post_move c'est que dark maul vient de bouger. La position de Dark Maul dans le labyrinthe est donnée par `cat_x` pour la position horizontale et `cat_y` pour la position verticale. Si on est visible et si notre position dans le labyrinthe est la même que Dark Maul (c'est à dire si `position_x`=`cat_x` et `position_y`=`cat_y`) alors c'est que Dark Maul vient de passer sur le coeur. Dans ce cas on cache le coeur on met la variable `visible` à zéro, si on a moins de 6 vies alors on augmente la variable `vie` de 1, sinon on augmente le score de 10.

2.4 Dark maul

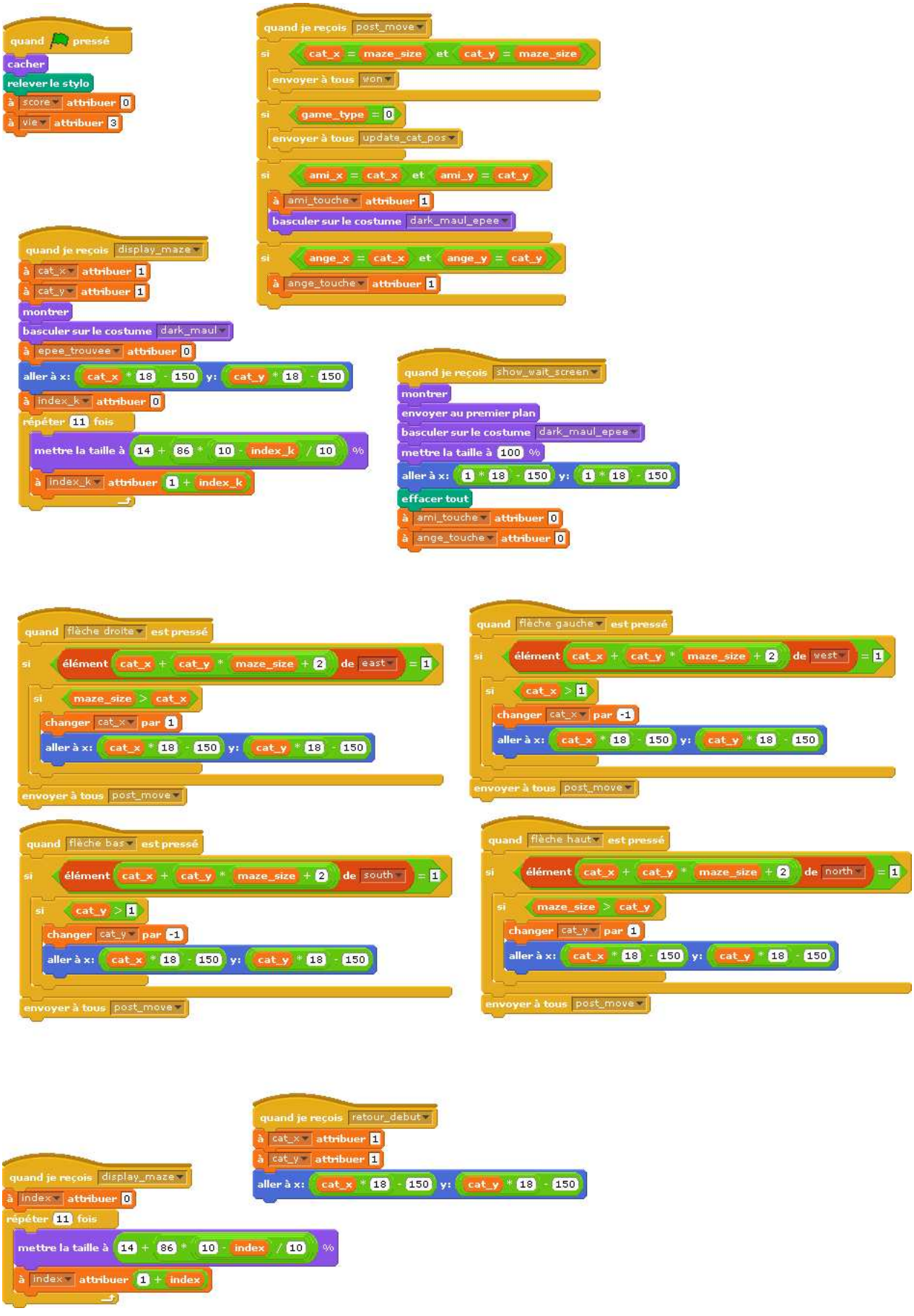
Voici à la page suivante les instructions pour Dark Maul, le personnage que l'on fait bouger dans le labyrinthe à l'aide des flèches.

Quand le drapeau vert est préssé, on cache l'objet, on relève le stylo (pour ne pas lasser une trace dans le labyrinthe), on initialise les variables `score` à 0 et `vie` à 3.

Quand le message display_maze est reçu on positionne Dark en bas à gauche, c'est à dire que l'on attribue 1 à `cat_x` et à `cat_y`, on initialise la variable `epee_trouvee` à zéro (pour dire que Dark Maul n'a pas encore trouvé l'épée). Puis on va à la posision :

$$(18 \times \texttt{cat\_x} - 150 \quad , \quad 18 \times \texttt{cat\_y} - 150)$$

exactement comme pour le coeur. Ensuite la séquence d'instructions "à index_k attribuer zéro" et le "répète 11 fois ..." sert à donner un effet de zoom arrière pour faire rentrer Dark Maul dans le labyrinthe.


quand pressé
cacher
relever le stylo
à score attribuer 0
à vie attribuer 3
quand je reçois post_move
si cat_x = maze_size et cat_y = maze_size
envoyer à tous won
si game_type = 0
envoyer à tous update_cat_pos
si ami_x = cat_x et ami_y = cat_y
à ami_touche attribuer 1
basculer sur le costume dark_maul_epee
si ange_x = cat_x et ange_y = cat_y
à ange_touche attribuer 1
quand je reçois display_maze
à cat_x attribuer 1
à cat_y attribuer 1
montrer
basculer sur le costume dark_maul
à epee_trouvee attribuer 0
aller à x: cat_x * 18 - 150 y: cat_y * 18 - 150
à index_k attribuer 0
répéter 11 fois
mettre la taille à 14 + 86 * 10 - index_k / 10 %
à index_k attribuer 1 + index_k
quand je reçois show_wait_screen
montrer
envoyer au premier plan
basculer sur le costume dark_maul_epee
mettre la taille à 100 %
aller à x: 1 * 18 - 150 y: 1 * 18 - 150
effacer tout
à ami_touche attribuer 0
à ange_touche attribuer 0
quand flèche droite est pressé
si élément cat_x + cat_y * maze_size + 2 de east = 1
si maze_size > cat_x
changer cat_x par 1
aller à x: cat_x * 18 - 150 y: cat_y * 18 - 150
envoyer à tous post_move
quand flèche gauche est pressé
si élément cat_x + cat_y * maze_size + 2 de west = 1
si cat_x > 1
changer cat_x par -1
aller à x: cat_x * 18 - 150 y: cat_y * 18 - 150
envoyer à tous post_move
quand flèche bas est pressé
si élément cat_x + cat_y * maze_size + 2 de south = 1
si cat_y > 1
changer cat_y par -1
aller à x: cat_x * 18 - 150 y: cat_y * 18 - 150
envoyer à tous post_move
quand flèche haut est pressé
si élément cat_x + cat_y * maze_size + 2 de north = 1
si maze_size > cat_y
changer cat_y par 1
aller à x: cat_x * 18 - 150 y: cat_y * 18 - 150
envoyer à tous post_move
quand je reçois retour_debut
à cat_x attribuer 1
à cat_y attribuer 1
aller à x: cat_x * 18 - 150 y: cat_y * 18 - 150
quand je reçois display_maze
à index attribuer 0
répéter 11 fois
mettre la taille à 14 + 86 * 10 - index / 10 %
à index attribuer 1 + index