ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

1A info – 2018-2019

Projet Conception de site Web

Objectif :

L'objectif de ce projet est de concevoir en HTML et Javascript, une animation dans laquelle plusieurs robots se déplaceront.

1. *Canvas 2D Context* – Moteur de jeu

En se basant sur la balise <canvas> et le langage Javascript, l'idée est de créer un moteur de jeu (simple).

Le moteur de jeu que nous allons créer est basé sur une grille, constituée de nbX colonnes et nbY lignes, soit nbX*nbY cellules. La grille représente l'environnement. Elle sera représentée en Javascript par un objet **Grille**, dont voici la spécification :

propriété	*désignation*	*valeur par défaut*
nbX	nombre de cellules selon les x	constructeur
nbY	nombre de cellules selon les y	constructeur
canva	DOM canvas	constructeur
ctx	contexte 2D	constructeur
dX	largeur d'une cellule	automatique
dY	hauteur d'une cellule	automatique
tab	matrice nbX*nbY bouléens	chaque élément à false
obj	tableau contenant les objets ajoutez à l'environnement	vide
nbC	nombre de cellules	nbX*nbY
nbCur	nombre de cellules explorées	0
Grille(nbX,nbY,canvaID)	constructeur de la grille	
dessinGrille()	dessine la grille (ses bords)	
ajoutObjet(objet)	ajoute un nouvel objet au tableau obj, incrémente nbCur et positionne à explorée la cellule (tab[objet.posX][objet.posY]=true)	
dessin()	dessine les objets sur la grille, en appelant la méthode dessin() de chaque objet, ainsi que la grille elle-même	
mvtAdmis(objet,dx,dy)	retourne true si l'objet peut se déplacer d'un mouvement de (dx,dy) depuis sa position actuelle (objet.posX,objet.posY)	
majDessinObjet(objet,dx,dy)	met à jour l'affichage d'un objet suite à son déplacement sur la grille de (dx,dy)	
stopObjets()	annule les déplacement de tous les objets en les rendant endormis	

La grille (environnement) est initialement vide (constructeur). On peut

- la dessiner (méthode dessinGrille())
- lui ajouter des objets, chaque objet étant sur une cellule uniquement
- dessiner les objets et la grille
- vérifier si un objet peut se déplacer sur les 4 cellules voisines (horizontalement et verticalement), nous prendrons pour l'instant comme règle : toute cellule est admise si elle appartient à la grille
- dessiner le déplacement d'un objet en effaçant la case sur laquelle il est, puis en mettant à jour sa position, et en le dessinant sur la nouvelle case (tous les pixels du canvas ne sont ainsi pas mis à jour)
- les objets sont des agents autonomes qui se déplacent, il faut pouvoir les stopper (arrêter leur déplacement), nous les stopperons lorsque toutes les cases auront été parcourues (pour l'instant)

Bien que la grille permette d'accueillir différents objets, nous ne considèrerons pour l'instant qu'un seul type d'objet : **Robot** avec la spécification suivante.

propriété	*désignation*	*valeur par défaut*
posX	indice de la cellule selon x	constructeur
posY	indice de la cellule selon y	constructeur
temps	vitesse en ms	constructeur
couleur	couleur du fond	constructeur
timeID	id du dernier timeout	null
eveille	true si le robot est éveillé, false sinon	false
Robot(x,y,vitesse,couleur)	constructeur du robot	
dessin(ctx,dx,dy)	dessine le robot (un disque inclus dans la cellule) sur la grille sans se préoccuper de l'endroit où il se situera, dx et dy représentent les dimensions d'une cellule	
perception()	détermine les mouvements possibles du robot avec la méthode mvtAdmis de Grille, sous la forme d'un tableau, et appel la méthode reflexion après temps (avec setTimeout)	
reflexion(tabMvt)	détermine aléatoirement quel mouvement le robot va choisir parmi l'ensemble des mouvements possibles tabMvt déterminés lors de la phase de perception, puis appel la méthode action après temps (avec setTimeout)	
action(act)	effectue l'action déterminée lors de la phase de réflexion, *i.e.* le déplacement donné par act en appelant la méthode majDessinObj de Grille, puis la méthode perception est appelée pour assurer la boucle perception-reflexion-action	
reveille()	positionne à true eveille, et lance la boucle perception-reflexion-action	

Une fois créé, un robot peut être dessiné, puis être réveillé, dans quel cas il entre dans la boucle perception-reflexion-action, qui lui permettra de se déplacer dans son environnement.

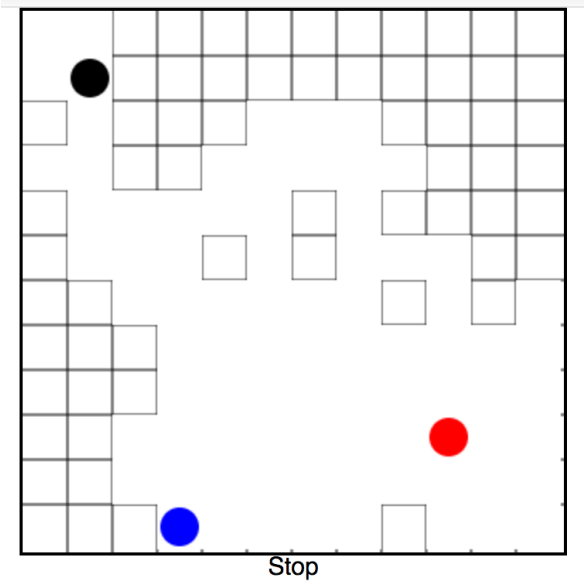
En vous basant sur le fichier projetRobot.html, codez ces 2 objets, sachant que les méthodes doivent être codées en utilisant prototype. Cela permettra de créer une grille (dimensions quelconques) sur un canvas (dimensions quelconques), de dessiner la grille, d'ajouter des robots et des les réveiller. Les robots se déplaceront jusqu'à ce que toutes les cellules de la grille aient été explorées par un des robots, à l'instar de ce qui est présenté dans l'exemple ci-dessous.

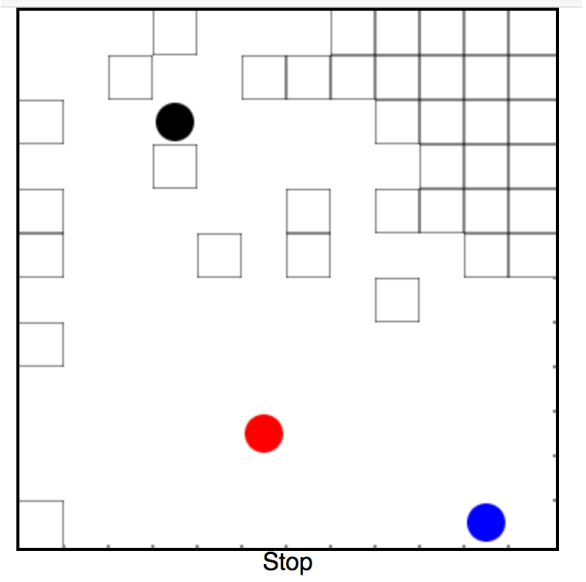
Code utilisé pour la scène globale :

```
var gr = new Grille(12,12,'mon_canvas');
var rbt1 = new Robot(4,4,0.1,'black');
var rbt2 = new Robot(11,11,0.03,'red');
var rbt3 = new Robot(5,10,0.05,'blue');
gr.ajoutObjet(rbt1);
gr.ajoutObjet(rbt2);
gr.ajoutObjet(rbt3);
gr.dessin();
rbt1.perception();
rbt2.perception();
rbt3.perception();
```

N'oubliez pas de prévoir l'arrêt des déplacements des robots lorque l'utilisateur clique sur le texte `STOP`.

Exemple de résultats de l'exécution du code à deux instants différents :





2. Simplification de l'exécution

L'exécution d'un code JS est séquentielle. Une portion de code s'exécute jusqu'à sa dernière instruction. Les fonctions mises en attente d'exécution avec setTimeout ne sont exécutées que lorsque le temps prévu est dépassé et si aucune autre portion de code ne s'exécute.

Modifier l'exécution de la boucle perception-reflexion-action pour qu'il n'y ait plus d'attente entre chaque exécution de fonction de cette boucle. Seule une attente subsiste entre chaque exécution de la boucle (dans la méthode action). Observez dans la console l'exécution du code. Si vous avez correctement réalisé la modification de la boucle perception-reflexion-action, un robot exécute sa boucle du début à la fin sans être interrompu par l'exécution d'une autre portion de code. Un robot s'arrête (sort de sa boucle) uniquement lorsque l'utilisateur a cliqué sur 'Stop', ou bien lorsque toute les cases de la grille ont été explorées.

3. Modification du comportement, ajout/suppression de robots

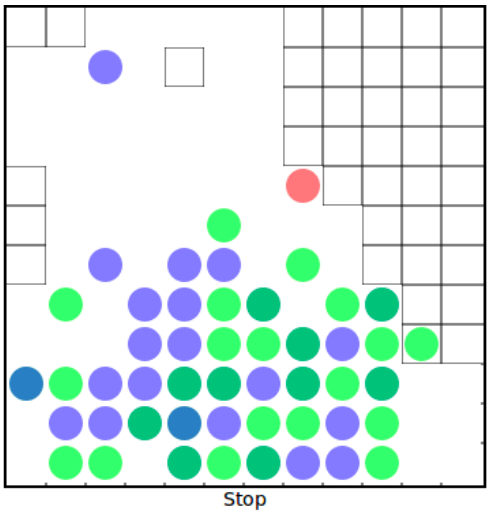
Vous allez maintenant complexifier le comportement des robots selon les règles suivantes :

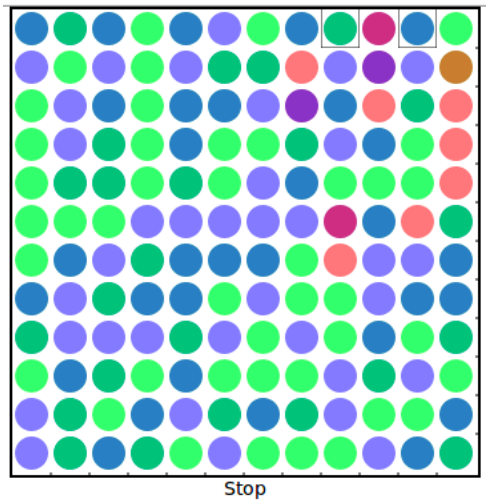
- un robot peut se déplacer sur une cellule que si la cellule comporte déjà 0 ou 1 robot
- lorsqu'un robot A quitte une cellule de la grille pour se déplacer sur une nouvelle :
    a. si la nouvelle cellule accueille déjà un robot B
    b. si la couleur du robot B est la même que celle de A alors A est arrêté, il sort de sa boucle perception-reflexion-action, sinon un nouveau robot C est créé, avec la couleur de A, et positionné sur la même cellule que celle de A et B
- un robot s'arrête lorsqu'il ne peut plus se déplacer (toutes ses cellules voisines sont déjà occupées par au moins 2 autres robots)
- tous les robots s'arrêtent lorsque l'U clique sur 'Stop' (ou bien implicitement lorsque tous les robots ne peuvent plus bouger).

Vérifier que ce comportement est correctement codé de manière à ce que

- les robots ne s'arrêtent plus lorsque la toutes les cellules de la grille ont été parcourues,
- chaque cellule est maintenant un tableau d'objets (ceux sur la cellule) à-travers la propriété tab de Grille.

Attention la création de nouveau robots peut entraîner un blocage du script si elle est mal gérée.





4. Amélioration du code

Améliorer le code pour permettre les tâches suivantes :

- création automatique de robots avec une nouvelle méthode de Grille à coder :

```
ajoutRobots(nb_pourcent,mintps,maxtps)
```

où `nb_pourcent` est le pourcentage de robots à créer relativement au nombre de cellules de la grille, la position de chaque est déterminée aléatoirement sur la grille, ainsi que le temps de pause entre chaque exécution de la boucle (déterminé aléatoirement entre `mintps` et `maxtps`). Pour la couleur du robot, cette dernière doit être déterminée avec un interpolateur de couleur. Pour cela vous pourvez utiliser la bibliothèque chroma (https://github.com/gka/chroma.js) déjà présente dans le dossier 'tools' en version minimale. Modifiez votre document pour avoir les fonctionnalités ci-dessus. La scène globale est alors céée de la façon suivante :

```
// scène globale
var gr = new Grille(50,50,'mon_canvas');
// stop
var oStopClic = document.getElementById('stop_clic');
if (oStopClic)
oStopClic.addEventListener('click',function(e){ gr.stopObjets();
});
// robots
gr.ajoutRobots(6,0.005,0.1);
```

Pour la création de l'interpolateur de couleur vous utiliser l'interpolation de bezier.

5. Ajout d'un marcheur

Modifiez le code de Grille pour ajouter un nouvel objet Robot qui possède une propriété supplémentaire par rapport aux autres robots : la propriété **marcheur** qui a pour valeur **true**. Un robot ayant cette propriété est appelé un marcheur. L'ajout de ce nouvel objet se fera avec la méthode **ajoutMarcheur(mintps)** de Grille que vous devez coder. Cette méthode ajoute un nouveau robot ayant la propriété marcheur=true, une couleur noire, et un temps de pause de mintps entre chaque appel de la boucle perception-reflexion-action. Le marcheur est placé au centre de la grille, puis il est réveillé. Contrairement aux autres robots, le marcheur ne peut se déplacer que sur une cellule vide de la grille. Le marcheur a gagné si il touche un bord de la grille (1ère ou dernière ligne ou colonne), et meurt si un robot non marcheur vient se déplacer sur sa cellule. Dans ces deux derniers cas tous les robots s'arrêtent et il est indiqué si le marcheur a gagné ou non (via la console).

Codez ce comportement dans un nouveau document 'tp4-5.html'.

6. Pour aller plus loin : thread en JS

Pour aller plus loin, si vous avez tout fait, il est possible de créer des threads en JS, ce qui permettrait de rendre complètement autonome tout objet. Ceci se réalise avec les Web Workers. Inspirer-vous de la documentation et modifier votre code en conséquence.

http://blogs.msdn.com/b/davrous/archive/2011/07/08/introduction-aux-web-workers-d-html5-le-multithreading-version-javascript.aspx

https://developer.mozilla.org/en-US/docs/Web/Guide/Performance/Using_web_workers