TP Labyrinthe : partie 1

Rédigé par : Jimmy Paquereau

1. Cahier des charges

Un logiciel de labyrinthes a été conçu par votre professeur préféré. Et vous allez devoir réécrire quelques-uns des algorithmes du programme. Cette application* possède le cahier des charges ci-après décrit.

** application = logiciel*

1.1. Fonctionnement général

o le joueur accède au mini-jeu via internet ;
o le mini-jeu de labyrinthe y est accessible en libre-service ;
o lorsque l'utilisateur est parvenu sur la page web du jeu (page de démarrage), il a le choix entre deux modes d'utilisation : le mode « solo » et le mode « duel » ;
o le mode « solo » est sélectionné par défaut ;
o le mode « duel » n'est accessible qu'aux utilisateurs connectés ;
o la page de démarrage comporte un bouton « Jouer » permettant de démarrer une nouvelle partie en mode « solo » ou « duel » selon le choix de l'utilisateur ;
o si l'utilisateur tente de lancer une partie en mode « duel » alors qu'il n'est pas connecté, un formulaire d'authentification/inscription s'affiche. S'il est déjà connecté, une nouvelle partie est lancée en mode « duel » ;
o si l'utilisateur clique sur le mode « solo », il peut alors sélectionner un niveau de difficulté dans une liste déroulante (facile, moyen, difficile), puis cliquer sur le bouton « Jouer ». Le niveau « facile » est sélectionné par défaut ;
o la page de démarrage comporte également un bouton d'aide. Lorsque l'utilisateur clique sur celui-ci, les règles du jeu lui sont présentées. Il a alors accès à un bouton retour lui permettant de revenir à la page de démarrage.

1.1.2. Mode « solo »

Une partie en mode « solo » se déroule comme suit :
o un labyrinthe est tout d'abord généré dynamiquement de manière pseudo-aléatoire ;
o la complexité du labyrinthe est fonction du niveau de difficulté sélectionné par l'utilisateur :
- au niveau « facile », le labyrinthe est généré sur une grille de 10x10 cases ;
- au niveau « moyen », le labyrinthe est généré sur une grille de 15x15 cases ;
- au niveau « difficile », le labyrinthe est généré sur une grille de 20x20 cases.
o le chronomètre est démarré et commence son décompte. Le temps imparti est fonction du niveau de difficulté et de la longueur du chemin menant à la sortie (on note N la longueur de ce chemin) :
- au niveau « facile », le temps imparti est : N x 700 (millisecondes) ;
- au niveau « moyen », le temps imparti est : N x 500 (millisecondes) ;
- au niveau « difficile », le temps imparti est : N x 400 (millisecondes) ;
o le chronomètre est mis à jour tous les $1/10^{ème}$ de seconde (soit toutes les 100 millisecondes) ;
o le joueur voit le labyrinthe. Deux flèches (→) apparaissent respectivement au niveau des cases de départ et d'arrivée ;

o une coccinelle apparaît sur la case départ. Le joueur doit déplacer la coccinelle jusqu'à la case d'arrivée dans le délai imparti (à l'aide des flèches directionnelles de son clavier) ;
o si le joueur n'est pas parvenu à sortir du labyrinthe dans le délai imparti, la solution est affichée (chemin allant de l'entrée à la sortie du labyrinthe), puis la vue « perdu » s'affiche. L'utilisateur a alors la possibilité soit de cliquer sur « rejouer » pour lancer une nouvelle partie, soit de cliquer sur « quitter » pour revenir à la page de démarrage ;
o si le joueur, au contraire, est parvenu à sortir du labyrinthe dans le délai imparti, la vue « gagné » s'affiche. L'utilisateur a alors une fois encore la possibilité soit de « rejouer » soit de « quitter ».

1.1.3. Mode « duel »

Dans son fonctionnement (point de vue utilisateur), le mode « duel » est similaire au mode « solo » à ceci près que la notion de mode multijoueur fait son entrée. En particulier, on constate les différences suivantes :
o à tout moment pendant la partie, le joueur voit son pseudonyme apparaître à côté d'une coccinelle rouge. Il voit également apparaître le pseudonyme de son adversaire à côté d'une coccinelle verte. ;
o à tout moment de la partie encore, le joueur voit la distance qui sépare son adversaire de la sortie du labyrinthe ;
o il n'y a pas de chronomètre ;
o le premier parvenu à la sortie remporte la partie.

Bien entendu, comme nous le disions initialement, l'utilisateur doit être connecté pour accéder au mode duel, ce qui nécessite de stocker *a minima* des utilisateurs dans une base de données. Egalement, le choix du joueur et de son adversaire est choisi comme suit :
o lorsqu'un utilisateur lance une partie en mode duel, un message « En attente d'adversaire » apparaît ;
o dès qu'un autre utilisateur lance une partie en mode duel, ou si un utilisateur est déjà « En attente d'adversaire », alors les deux utilisateurs sont « mis en relation », à savoir que le premier utilisateur rejoint la partie du second ;
o la partie commence et les deux joueurs ne sont dès lors plus en attente d'adversaire.

Une fois encore, il est nécessaire de stocker et partager ces informations.

Aussi, le mode « duel » nécessite inévitablement un intermédiaire afin de réaliser les traitements propres au mode « duel » et afin de faire transiter des informations entre joueurs : authentification, recherche d'adversaire, fin de la partie.

Soyons plus explicite. En mode « solo », le serveur web se contente de fournir le jeu vidéo. Le jeu vidéo s'exécute ensuite chez l'utilisateur, sur son navigateur, et le serveur web n'a plus besoin d'intervenir. Au contraire, en mode « solo », le serveur web intervient à plusieurs reprises au cours du cycle de vie du jeu vidéo. Plus encore, le mode « duel » requière la manipulation de données stockées (dans une base de données typiquement) : les utilisateurs et les parties en cours.

2. Intérêt

2.1. Compétences

Le présent TP porte sur plusieurs parties du programme de BTS CGO et va même au-delà. Mais, après tout, qui peut le plus, peut le moins. Aussi, il va vous falloir mettre en œuvre diverses compétences informatiques. En effet, nous allons manipuler concrètement et de manière assez réaliste des concepts autour des thèmes suivants :
o l'architecture client/serveur au travers de la mise en œuvre d'une application* web client/serveur. Dans cette 1ère partie, nous nous concentrerons sur la partie client, à savoir la partie du logiciel qui peut être

exécutée par un navigateur, autrement dit sans avoir besoin d'un quelconque serveur web. Dans la 2nde partie du TP, nous nous attellerons à compléter la partie serveur ;
o l'internet, les extranets et intranets au travers de l'utilisation de technologies répandues sur le web ;
o l'algorithmique au travers de l'utilisation élémentaire de deux langages de programmation (JavaScript et PHP), d'autres langages (HTML, CSS) et de quelques utilitaires (jQuery, PDO) ;
o le SQL et les bases de données au travers de l'utilisation de la mise en œuvre en PHP d'une base de données MySQL.

2.2. Objectif

Votre objectif n'est pas ici de tout retenir mais de retenir ce que vous pouvez. Il importe en revanche de comprendre ce TP afin de mieux ancrer dans vos mémoires les concepts informatiques étudiés au cours de votre formation. En somme, ce TP en deux parties est un travail de synthèse.

3. Travail à réaliser

3.1. Travail préliminaire

Avant toute chose, nous allons créer ensemble une page web afin que vous manipuliez concrètement un peu de HTML, de CSS et de JavaScript. Je vous recommande tout simplement de prendre des notes et de vous envoyer à vous-même le résultat par mail.

3.2. Le labyrinthe

A présent, préoccupons-nous du labyrinthe. En vous aidant de l'annexe 1, rédigez entre autre les procédures/fonctions suivantes :

Procédure/fonction	Description
`hasWun() : boolean`	Fonction retournant oui si la partie est gagnée, non sinon.
`moveTop()`	Fonction appelée lorsque l'utilisateur clique sur la flèche directionnel « haut », déplace la coccinelle selon les règles du jeu.
`moveBottom()`	Fonction appelée lorsque l'utilisateur clique sur la flèche directionnel « bas », déplace la coccinelle selon les règles du jeu.
`moveLeft()`	Fonction appelée lorsque l'utilisateur clique sur la flèche directionnel « gauche », déplace la coccinelle selon les règles du jeu.
`moveRight()`	Fonction appelée lorsque l'utilisateur clique sur la flèche directionnel « droite », déplace la coccinelle selon les règles du jeu.
`drawSolution2()`	Procédure colorant le chemin allant de l'entrée à la sortie du labyrinthe.
`drawSolution()`	Procédure traçant proprement le chemin allant de l'entrée à la sortie du labyrinthe.

Nous allons utiliser vos algorithmes, discuter ensemble afin de comprendre pour votre solution fonction ou ne fonctionne pas, et tester ces solutions afin de nous assurer de leur bon ou mauvais fonctionnement.

ANNEXE 1 - Fonctions disponibles

Le labyrinthe est à tout moment accessible au moyen de la variable globale **labyrinth**. Cette variable est plus qu'une simple variable, c'est ce qu'on appelle une instance d'objet. Une telle variable possède ses propres fonctions (appelées méthodes) et ses propres variables (appelés attributs).

Quelques-unes des méthodes de l'objet **labyrinth** :

Méthode et description	Exemple d'utilisation
`labyrinth.length():integer` *Description : retourne la largeur (=hauteur) du labyrinthe, à savoir le nombre de cases en largeur comme en hauteur. Le labyrinthe est un carré.*	`// Récupère la largeur du labyrinthe` `var taille = labyrinth.length();`
`labyrinth.position():Point` *Description : retourne la position actuelle de la coccinelle. La variable retournée possède deux attributs, x et y, déterminant la colonne (la 1ère colonne est à l'index 0) et la ligne (la 1ère ligne est à l'index 0).*	`// Récupère la position actuelle de la coccinelle` `var pos = labyrinth.position();` `// Affiche (message d'alerte) la position de la coccinelle :` `// colonne et ligne` `alert('Colonne : ' + pos.x);` `alert('Ligne : ' + pos.y);`
`labyrinth.startPoint():Point` **`labyrinth.endPoint():Point`** *Description : retourne respectivement les points (position) de départ et d'arrivée du labyrinthe. La variable retournée possède deux attributs, x et y, déterminant la colonne (la 1ère colonne est à l'index 0) et la ligne (la 1ère ligne est à l'index 0).*	`// Récupère le point de départ du labyrinthe` `var start = labyrinth.startPoint();` `// Affiche (message d'alerte) les coordonnées du point de` `// départ du labyrinthe` `alert('Colonne : ' + start.x);` `alert('Ligne : ' + start.y);`
`labyrinth.timeout():boolean` *Description : teste si le temps imparti pour atteindre la sortie du labyrinthe est écoulé.*	`// Récupère si oui ou non (true/false) le temps est écoulé` `var out = labyrinth.timeout();`
`labyrinth.hasWallLeft(i,i):boolean` **`labyrinth.hasWallRight(i,j):boolean`** **`labyrinth.hasWallTop(i,j):boolean`** **`labyrinth.hasWallBottom(i,j):boolean`** *Description : teste si la case (i,j) du labyrinthe possède respectivement :* *o un mur à gauche ;* *o un mur à droite ;* *o un mur en haut ;* *o un mur en bas.* *Le premier index, pour i comme pour j, est 0.*	`// Teste s'il existe un mur à gauche de la case (0,5) :` `// 1ère colonne, 6ème ligne :` `var murGauche = labyrinth.hasWallLeft(0,5);` `if(murGauche){` `  alert('Il y a un mur à gauche');` `}else{` `  alert('Il n\'y a pas de mur à gauche');` `}`
`labyrinth.move(i,j)` *Description : déplace la coccinelle et la positionne au niveau de la case (i,j). Une fois de plus, le 1er index pour i comme pour j est 0.*	`// Déplace la coccinelle sur la case (2,3) :` `// 3ème colonne, 4ème ligne :` `labyrinth.move(2,3);`
`labyrinth.orientation(orientation)` *Description : oriente la coccinelle dans une direction. Les directions possibles sont : top, left, right et bottom.*	`// Oriente la coccinelle vers le haut` `labyrinth.orientation('top');`

labyrinth.color(i,j) *Description : permet de colorer la case (i,j), en orange.*	// Colore la case (2,2) du labyrinthe // 3ème colonne, 3ème ligne `labyrinth.color(2,2);`
labyrinth.straightTop(i,j) **labyrinth.straightBottom(i,j)** **labyrinth.straightLeft(i,j)** **labyrinth.straightRight(i,j)** *Description : permet d'afficher un petit trait qui va du centre de la case (i,j) vers respectivement le haut, le bas, la gauche et la droite.*	// Trace deux traits dans la case (4,5) du labyrinthe // 5ème colonne, 6ème ligne `labyrinth.straightTop(5,6);` `labyrinth.straightRight(5,6);` Résultat : 
labyrinth.solution():Array *Description : retourne le chemin permettant d'aller de l'entrée à la sortie du labyrinthe. Ce chemin est un tableau de points. Le 1er point (index 0) est le point de départ, le dernier est le point d'arrivé.*	// récupère la solution (tableau) `var chemin = labyrinth.solution();` // Affiche (message d'alerte) la longueur du chemin `alert(chemin.length-1);` // parcourt le tableau `for(var i=0 ; i<chemin.length ; i++){` // récupère le i-ème point du chemin puis en affiche les // coordonnées `var point = chemin[i];` `alert(point.x);` `alert(point.y);` `}`
labyrinth.chrono(active) *Description : méthode permettant d'activer/désactiver (true/false) l'utilisation du chronomètre.*	// désactive le chronomètre. `labyrinth.chrono(false);`
labyrinth.play() *Description : méthode à appeler pour débuter une nouvelle partie.*	
labyrinth.win() *Description : méthode à appeler pour terminer une partie gagnée.*	
labyrinth.loose() *Description : méthode à appeler pour terminer une partie perdue.*	