Projet "Jeu de Golf"
Programmation en JAVA 1.7 et en binôme (obligatoirement)

1 Objectifs

Le but du projet est d'implémenter un jeu de golf, en utilisant obligatoirement une structure de quadtree que vous définirez de sorte qu'elle soit la mieux adaptée pour effectuer de manière très efficace l'opération suivante du jeu de golf : identifier l'endroit (position, mais aussi environnement) où la balle envoyée par le golfeur tombe. Les principes de construction de cette structure vous sont donnés dans le sujet.

2 Description du terrain de jeu

Nous supposerons que le terrain de jeu est un carré de [0..10]×[0..10] (avec le coin (0,0) en bas à gauche, l'axe horizontal étant l'axe des abscisses - coordonnée x d'un point (x, y) - et l'axe vertical étant l'axe des ordonnées - coordonnée y d'un point (x, y)). Tout point (x, y) du terrain a des coordonnées réelles. Toutes les surfaces qui interviennent dans le jeu sont définies par des polygones simples (convexes ou concaves). Un exemple de terrain de jeu vous est donné dans la Figure 1, mais votre programme doit fonctionner sur n'importe quel autre terrain de jeu respectant les règles de construction décrites plus bas.

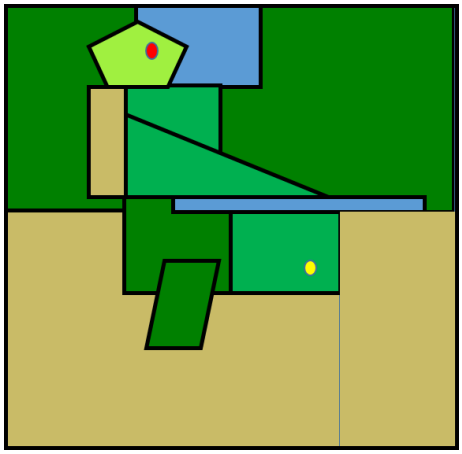

FIGURE 1 – Note : la surface beige en bas est définie par un SEUL polygone (malgré la ligne verticale sur le dessin)

Le terrain de jeu est partitionné en surfaces (définies par des polygones simples, donc) qui sont de cinq couleurs : vert (représentant l'herbe), vert clair (représentant le gazon bien tondu), beige (représentant le sable), bleu (représentant une surface d'eau), vert sapin (représentant une végétation haute, genre forêt).

Les surfaces de couleur verte et vert clair sont combinées pour former p surfaces de jeu qui s'appellent normalement des *trous*, mais que nous appellerons (les golfeurs nous pardonneront) des *tracés*. Tout tracé est une union de :

- $k \in \{1, 2, 3\}$ surfaces vertes, qui sont numérotées de 1 à k telles que l'on peut passer successivement de 1 à 2 (si $k \geq 2$) et de 2 à 3 (si $k = 3$) en traversant uniquement des surfaces beiges, bleues, ou vert sapin. Voir la figure, dans laquelle les trois zones vertes sont numérotées de 1 à 3 depuis le bas vers le haut.
- Une surface de couleur vert clair, voisine par au moins un côté (ou une partie d'un côté) à la k-ème surface verte. Voir la figure.

Chaque surface verte ou de couleur vert clair appartient à un tracé et un seul. Les surfaces vertes forment ce qu'on appelle le *fairway* du tracé (son parcours, en quelque sorte), et la surface verte clair est le *green* du tracé (sa zone d'arrivée). Chaque tracé contient en sa surface no. 1 un point D nommé départ et en son green un point T nommé trou (ce qui explique qu'on ait eu besoin de changer de terminologie ci-dessus). Le green est noté G.

3 Règles du jeu (simplifiées et librement interprétées)

Les tracés sont numérotés de 1 à p. A chaque tracé R_i on associe un nombre théorique $c_i \in \{3, 4, 5\}$ de coups pour que la balle partie du départ D_i du tracé R_i arrive dans le trou T_i (on dit alors que le tracé est un *par* c_i). Le *par* est une sorte de repère pour que le joueur évalue son propre score sur chaque tracé, c'est-à-dire le nombre réel de coups qu'il lui a fallu pour emmener la balle depuis D_i et jusqu'à T_i en suivant les règles, par rapport au *par* du tracé. Le score total du joueur est la somme de ses scores sur chaque tracé, en parcourant chaque tracé exactement une fois. A la fin, plus le score est bas, meilleur est le joueur.

3.1 Déroulement du jeu.

Le joueur joue sur tous les tracés dans l'ordre de 1 à p. A chaque fois qu'il tape ou qu'il fait rouler la balle (c'est-à-dire qu'il joue un coup), il ajoute 1 à son score. Des points de pénalité existent, décrits plus bas. Supposons que le joueur est au tracé R_i. Il tape la balle à partir du point de départ D_i, dont il connait les coordonnées, avec un *angle* en l'envoyant à une *distance*. Une fois que sa balle a atterri, en fonction de l'endroit (comme décrit plus bas), il tape à nouveau la balle avec un angle et à une distance. Et ainsi de suite jusqu'à ce que la balle atterrisse dans le green G_i ou dans le trou T_i. Dans le green, la balle est poussée pour rouler, de nouveau avec un angle et une distance, jusqu'à ce qu'elle arrive dans le trou ou qu'elle ressorte du green (dans ce deuxième cas, on la tape à nouveau).

3.2 Règles générales sur le déplacement d'une balle renvoyée par le joueur.

Quand il doit jouer la balle à partir d'un point (x, y) du terrain de jeu, le joueur choisit dans le terrain de jeu, où il veut, un *point cible* (z, t) qui est à une distance (euclidienne) d du point duquel il tape s'il est en position de taper, ou du point duquel il pousse la balle s'il est dans le green. La droite qui passe par (x, y) et (z, t) forme avec l'axe des abscisses un angle α (exprimé en degrés), calculé à partir de l'axe des abscisses en tournant dans le sens opposé des aiguilles d'une montre. Le joueur n'étant pas une machine et le vent étant, lui aussi, joueur, la balle ne va pas arriver forcément dans le point (z, t). En effet, l'angle α va être modifié (augmenté ou diminué) d'un nombre de degrés $a \in \{0, 1, \ldots, 10, 40\}$ tiré au hasard, et indiquant que le joueur a raté son angle un peu (entre 0 et 10 degrés) ou beaucoup (40 degrés). La distance d sera elle-même modifiée (augmentée ou diminuée) d'un pourcentage $b \in \{0\%, 1\%, \ldots, 10\%, 40\%\}$, qui est également tiré au hasard. La balle va donc atterrir dans le point (z', t') tel que l'angle α' formé par la droite reliant (x, y) à (z', t') et l'axe des abscisses est obtenu comme décrit ci-dessus, et la distance d' entre (x, y) et (z', t') est obtenue comme décrit ci-dessus.

3.3 Règles générales concernant le coup suivant un coup déjà joué.

En fonction du point (z', t') d'aterrissage de la balle, plusieurs situations peuvent apparaître définissant les règles sur le coup suivant à jouer :

- La balle atterrit dans une surface beige (sable). Elle doit alors être tapée de l'endroit où elle se trouve mais, exceptionnellement, elle est beaucoup ralentie. La distance d' calculée comme ci-dessus est réduite de moitié.
- La balle atterrit en dehors du terrain de jeu, ou dans une surface vert sapin ; elle est alors *hors-limites* et le joueur doit rejouer à partir de l'endroit où il a tapé/fait rouler la balle la dernière fois, en ajoutant 1 à son score pour ce tracé (il s'agit d'un point de pénalité).
- La balle atterrit dans le green du trou visé, à au plus 1 mètre du trou. Exceptionnellement, dans ce cas on suppose que le joueur maîtrise son coup, et au coup suivant, si le joueur vise le trou, la balle arrive bien dans le trou (ni l'angle α ni la distance d ne sont modifiés).
- La balle atterrit dans une surface bleue (dans l'eau). Elle doit alors être tapée depuis l'endroit E où elle a franchi la ligne délimitant la surface d'eau (juste avant de survoler l'eau). Un point de pénalité est ajouté au score. Si jamais le point E est hors-limites, alors le joueur doit rejouer à partir de l'endroit où il a tapé/fait rouler la balle la dernière fois.
- La balle atterrit dans le green du trou visé, à plus d'1 mètre du trou ou dans le fairway (celui du tracé visé ou d'un autre tracé). Alors le coup suivant est joué depuis l'endroit où la balle est tombée.

4 Travail demandé

Le programme que vous allez écrire doit d'abord représenter le terrain de jeu (avec ses surfaces colorées) à l'aide d'une structure de données de type quadtree. Pour cela, les surfaces sont d'abord découpées en triangles disjoints (qui ne peuvent avoir en commun que des points situés sur leurs côtés) dont les sommets sont des sommets des polygones (comme vu dans le cours sur les quadtrees, diapo 6, mais SANS introduire de nouveaux points). Ensuite, on découpe récursivement le terrain selon les règles de construction d'un quadtree, en quartiers. On arrête le découpage d'une région représentée dans le quadtree dès que la région intersecte au plus t triangles différents, où t est un entier fourni par l'utilisateur au début du programme. Pour cela, on considère qu'une région intersecte un triangle si et seulement si il y a un point interne (c'est à dire non sur les côtés) de la région qui est sur le triangle (cela équivaut à dire que la région et le triangle ont au moins deux points internes en commun, comme vu dans le TP2). Etant donné un point avec ses coordonnées, représentant la position où la balle est tombée, vous devez déduire de la manière la plus efficace possible, utilisant le quadtree mais aussi des structures supplémentaires que vous définirez, dans quelle surface est tombée la balle et à partir de quel point elle doit être jouée au coup suivant.

Votre programme doit implémenter obligatoirement les méthodes suivantes (mais aussi d'autres, selon vos besoins) :

- *TriangulationTerrain* qui réalise la découpe des surfaces en triangles disjoints, selon la méthode décrite dans les exercices 9 et 10 du TP 2. *Note.* Une variante simplifiée, et moins bien notée, consiste à considérer que les surfaces sont toutes convexes, selon l'exercice 8 du TP2. Vous pouvez vous contenter de cette version si vous n'arrivez pas à résoudre celle où les surfaces peuvent être concaves.
- *ConstructionQT* qui construit le quadtree à partir du terrain donné et des triangles obtenus par l'opération de triangulation.
- *CalculCoefficientsDroite* qui calcule les coefficients a, b, c de la droite $ax + by + c = 0$ qui passe par deux points donnés (x, y) et (x', y').
- *TestIntersectionDeuxSegments* qui, étant donnés quatre points $(x, y), (x', y'), (z, t), (z', t')$ teste si le segment de droite fermé compris entre (x, y) et (x', y') sur la droite définie par ces deux points et le segment de droite fermé compris entre (z, t) et (z', t') sur la droite définie par ces deux points ont une intersection non-vide et, si oui, si elle est identique à l'un des points $(x, y), (x', y'), (z, t), (z', t')$ ou non.

- *TestRegionIntersecteTriangle* qui teste si une région carrée donnée représentée par un noeud interne du quadtree intersecte un triangle donné.
- *TestTriangleContientPoint* qui teste si un triangle donné contient un point donné.
- *RecherchePointQT* qui cherche la région la plus petite réprésentée par le quadtree à laquelle appartient un point donné (si une telle région existe).
- *RecherchePointTriangle* qui, étant donné un point et la région retournée par RecherchePointQT, cherche le triangle auquel appartient le point parmi les triangles que cette région intersecte.
- *CalculePointAtterrissageBalle* qui, pour une balle tapée au point donné (x, y) alors que le joueur a choisi comme point cible le point donné (z, t), calcule l'endroit (z', t') où la balle doit atterrir selon les règles générales de déplacement de la balle.
- *CalculePointDepartBalle* qui, pour une balle tapée au point donné (x, y) et en fonction de l'endroit où elle a atterri, calcule le point (u, v) d'où la balle doit être renvoyée au coup suivant.
- *CalculeScore* qui calcule le score total courant à chaque fois que la balle a été jouée, en fonction du point d'atterrissage de la balle, et compare le score courant au *par* total des tracés parcourus, si on est arrivé à la fin d'un tracé.

Le programme doit être *aussi efficace que possible*. Il prendra en entrée les surfaces du terrain de jeu sous la forme d'un fichier avec le format suivant.

- La 1ère ligne contient le nombre h de surfaces.
- Les lignes de 2 à $h+1$ décrivent une surface chacune en énumérant ses points selon un parcours du polygone en suivant ses côtés, dans le sens contraire des aiguilles d'une montre, et en fournissant sa couleur avec le codage : V=vert, C=vert clair, B=bleu, S=vert sapin, J=beige. Ainsi

  $(3.5, 4.7), (2.5, 5.2), (1, 4), (0, 0), V$

  (sans aucun espace sur la ligne) indique un quadrilatère de couleur verte dont les côtés sont définis par (1) les points $(3.5, 4.7)$ et $(2.5, 5.2)$; (2) les points $(2.5, 5.2)$ et $(1, 4)$; (3) les points $(1, 4)$ et $(0, 0)$; (4) les points $(0, 0)$ et $(3.5, 4.7)$.
- La ligne $h + 2$ indique le nombre de tracés (c'est-à-dire la valeur de p).
- Les lignes de $h + 3$ à $h + p + 2$ indiquent un tracé chacune sous la forme :

  $l_1, l_2, l_3, l_4, (x_1, y_1), (x_2, y_2), par$

  (sans aucun espace sur la ligne) où l_1 est la ligne du fichier à laquelle apparait la surface numérotée 1 du tracé décrit ; la ligne l_2 est la ligne du fichier à laquelle apparaît la surface numérotée 2 du tracé décrit (si le tracé a une surface 2 ; sinon, $l_2 = 0$) ; la ligne l_3 est la ligne du fichier à laquelle apparaît la surface numérotée 3 du tracé décrit (si le tracé a une surface 3 ; sinon $l_3 = 0$) ; l_4 est la ligne du fichier à laquelle apparait le green du tracé décrit ; (x_1, y_1) et (x_2, y_2) sont respectivement les coordonnées du point de départ et du point d'arrivée du tracé ; par est un entier qui indique le *par* du tracé.

Nous supposerons que les informations fournies sont correctes, selon les règles de constitution d'un terrain de jeu décrites plus haut.

La description du terrain [1] présenté dans la Figure 1 est donnée dans le fichier joint Description-FigureGolf.txt, et ci-dessous (la notation décimale utilise le point ; la virgule est pour séparer les deux coordonnées) :

1. légèrement déformé, mais sans aucun impact sur le traitement qu'on vous demande

12
(2,8.5),(2.2, 8),(3.5,8),(4,8.5),(3,9),C
(5.5,10),(3,10),(3,9),(4,8.5),(3.5,8),(5.5,8),B
(4.5,8),(2.5,8),(2.5,7),(4.5,6.5),V
(2,8),(2,5.5),(2.5,5.5),(2.5,8),J
(2.5,7),(2.5,5.5),(7,5.5),V
(2.5,5.5),(2.5,3),(3,3),(3.5,4),(4.5,4),(4,3),(5,3),(5,5),(3.5,5),(3.5,5.5),S
(2.7,2),(3.7,2),(4.5,4),(3.5,4),S
(3.5,5.5),(3.5,5),(9,5),(9,5.5),B
(5,5),(5,3),(7,3),(7,5),V
(0,10),(0,5),(2.5,5),(2.5,5.5),(2,5.5),(2,8),(2.2,8),(2,8.5),(3,9),(3,10),S
(5.5,10),(5.5,8),(4.5,8),(4.5,6.5),(7,5.5),(9,5.5),(9,5),(10,5),(10,10),S
(0,5),(0,0),(10,0),(10,5),(7,5),(7,3),(4,3),(3.7,2),(2.7,2),(3,3),(2.5,3),(2.5,5),J
1
10,6,4,2,(6.5,4),(3.2,8.5),4

Avec ces informations, à partir du clavier et/ou de la souris l'utilisateur doit pouvoir :

1. jouer, en choisissant tous les paramètres indiqués dans la description du jeu *sans avoir à entrer dans le code*. En particulier, l'utilisateur doit pouvoir choisir le point de destination (z, t) où il veut envoyer la balle. Il ne vous est pas demandé de faire un affichage graphique, mais quel que soit l'affichage choisi il doit permettre de jouer coup après coup et de suivre pas à pas l'évolution de la balle dans le jeu (d'où elle part, quel est le point visé, de combien l'angle et la distance sont modifiés, dans quel point la balle attérrit, quel est le score à chaque moment etc.).
2. tester les méthodes que l'on vous a demandé d'implémenter, encore une fois sans avoir à entrer dans le code. Pour cela un menu sera fourni, permettant de tester en début de partie, avant de jouer, toutes les méthodes indiquées, sur les données fournies par l'utilisateur.

Le programme sera lancé en ligne de commande et pourra être utilisé indépendemment de Eclipse ou autre IDE.

5 Rendu de TP

Les programmes doivent être implémentés par vous-mêmes en Java 1.7 et doivent fonctionner parfaitement sur les machines du CIE, sous Linux. Aucun appel à une méthode mise à disposition dans les bibliothèques Java (par exemple un tri d'une liste) n'est autorisé si vous ne connaissez pas le fonctionnement de la méthode et sa complexité.

Un rapport d'au maximum 12 pages sera fourni, comprennant (entre autres) :

- les commandes de compilation et exécution en mode console
- une vue globale de votre approche, sous la forme d'un algorithme (c'est-à-dire en pseudo-code) dont les instructions sont des appels à des procédures/fonctions ; la spécification et les paramètres de chaque procédure/fonction sont précisément décrits
- pour chaque méthode parmi celles de la liste demandée :
  * le détail de la procédure/fonction, sous forme algorithmique (c'est-à-dire en pseudo-code) ;
  * l'explication de son fonctionnement ;
  * les raisons pour lesquelles l'algorithme proposé est correct ;
  * sa complexité ;
  * les raisons pour lesquelles vous considérez cet algorithme aussi efficace que possible
- des jeux de données commentés

Important

1. Les fichiers du programme, ainsi que les jeux de données, seront mis dans un répertoire Nom1Nom2 (où Nom1 et Nom2 sont les noms des deux étudiants du binôme). Ce répertoire sera ensuite archivé sous le nom Nom1Nom2.zip. L'archive sera déposée sur Madoc, au plus tard le **vendredi 8/12/2017 à 23h59** (Madoc n'acceptera pas de retard).
2. La dernière séance est consacrée à une démo de 10 minutes par projet, pour laquelle les fichiers en entrée vous seront fournis (respecter donc scrupuleusement le format indiqué). Cette dernière séance **n'est donc pas une séance de travail sur le projet**.
3. *Tout* est important pour la notation. En particulier, il sera accordé beaucoup d'attention au respect des consignes et à la recherche d'une complexité minimum - garantie d'une efficacité maximum - pour votre méthode, via la mise en place des structures de données les plus adaptées.