Interrogation d'informatique
2ème année - Janvier 2017-2018

Durée totale : *2h*
Documents autorisés : *Toutes notes personnelles ou du cours.*

— Le barème est indicatif et est sur 24 points.
— Le sujet est sur 6 pages.

On souhaite programmer un jeu simple qui consiste à faire sortir un personnage (le *randonneur*) d'un labyrinthe. Ce jeu est illustré dans la figure 1 (à gauche). Le randonneur est représenté par l'ellipse bleue. Il démarre du carré rouge (le départ, en bas à gauche sur la figure) et doit arriver au carré vert (l'arrivée, en haut à droite sur la figure). Le joueur le déplace en utilisant les boutons du bas. Les murs du labyrinthe sont représentés en noir. Le premier exercice traitera de l'interface graphique et le deuxième des aspects programmation orientée objet. Les exercices ne sont pas indépendants mais peuvent être fait dans n'importe quel ordre ; il est toutefois conseillé de bien lire tout l'énoncé avant de démarrer.

Exercice 1 Interface Graphique et événements (14.5 pts)

L'interface graphique est modélisée par 3 `JPanels` : *pMain*, *pDir* et *pTerrain* et 4 `JButtons` : *bHaut*, *bBas*, *bGauche*, *bDroite*. *pTerrain* et *pDir* sont contenus dans *pMain* et les 4 `JButtons` sont contenus dans *pDir*. Les coordonnées de ces éléments sont illustrés dans la figure 1 (à droite). Les couleurs des `JPanels` sont respectivement jaune, vert et blanc pour *pMain*, *pDir* et *pTerrain*.

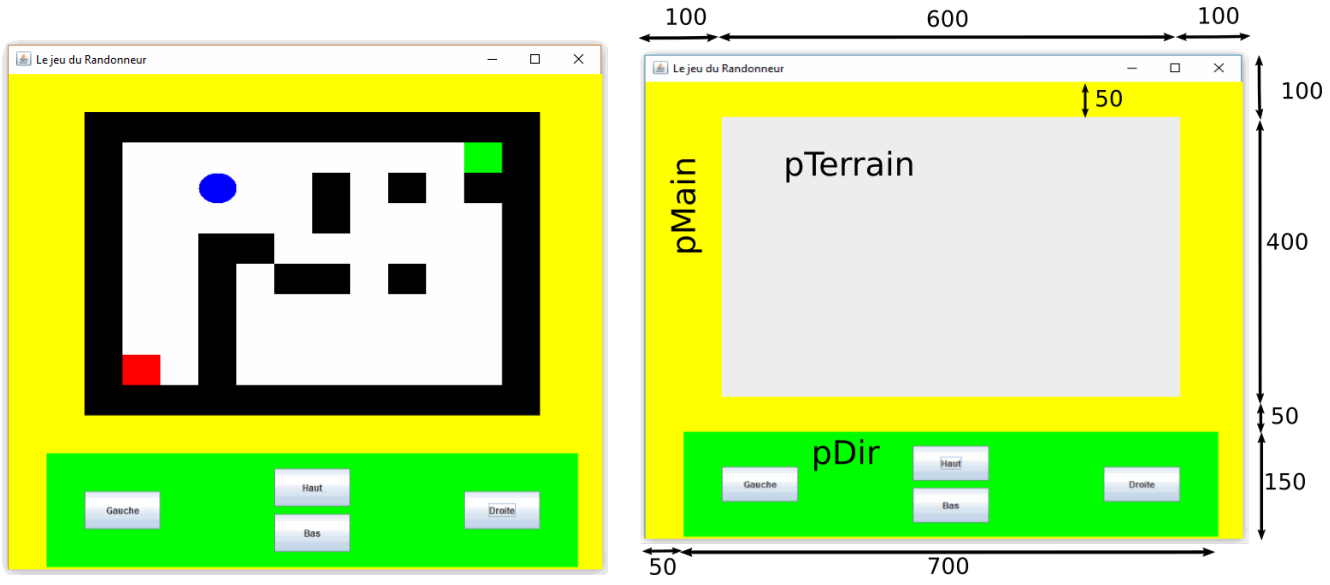

FIGURE 1 – Gauche : illustration de l'IHM du jeu. Droite : illustration de la structure et du placement des composants graphiques.

(Q1.1) Mise en place des composants graphiques - Quelques questions (3 pts)

La classe *Fenetre* détaillée en partie ci-dessous modélise cette interface graphique.

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Fenetre extends JFrame implements ActionListener{

    private JButton bHaut;
    private JButton bBas;
    private JButton bGauche;
    private JButton bDroite;

    private JPanel pTerrain;

    public Fenetre() {
        super("Le jeu du Randonneur");
        this.setSize(800,700);
        pTerrain = new Terrain();
        pTerrain.setLayout(null);
        JPanel pDir=new JPanel();
        pDir.setLayout(null);
        JPanel pMain = new JPanel();
        pMain.setLayout(null);
        bHaut   = new JButton("Haut");
        bBas    = new JButton("Bas");
        bGauche = new JButton("Gauche");
        bDroite = new JButton("Droite");
        bHaut  .setBounds(300,20,100,50);
        bBas   .setBounds(300,80,100,50);
        bGauche.setBounds(50,50,100,50);
        bDroite.setBounds(550,50,100,50);

        ...A COMPLETER (Q1.2)...

    }


    public static void main(String[] a){
        new Fenetre();
    }

     public void actionPerformed(ActionEvent e){
                ...A COMPLETER (Q1.8)...
    }
}
```

Répondez de manière précise et en 3 lignes max aux questions suivantes :

(a) A quoi servent les instructions `import` ?
(b) Pourquoi les `JButtons` et le `JPanel` *pTerrain* sont ils déclarés en attributs de la classe ?
(c) A quoi sert la méthode main ? que se passe-t-il si on l'enlève
(d) Quelle est la signification de `implements ActionListener` ?
(e) Que fait l'instruction `super("Le jeu du Randonneur")` ?
(f) Pourquoi y a t'il une méthode `actionPerformed` ? A quoi sert elle ?

Réponse : (3 pt)

(a) A importer les bibliothèques qui permettent de se servir des composants graphiques 0.5pt
(b) Afin que l'on puisse les utiliser dans les autres méthodes de la classe, notamment dans le ActionPerformed 0.5pt
(c) Elle appelle le constructeur de Fenêtre ce qui déclenche l'affichage de la fenêtre principale à l'écran. Si on l'enleve, le programme ne s'éxécute pas. 0.5pt
(d) On indique que notre fenêtre va « écouter » les événements de type ActionEvent 0.5pt
(e) Elle appelle le constructeur de la classe mère (JFrame) et définit le titre de la fenêtre 0.5pts
(f) Cette méthode permet de réagir au clics sur les boutons (événements de type action event)

0.5pts

(Q1.2) Mise en place des composants graphiques - A vous de jouer (2.5 pts)

Complétez le constructeur de la classe *Fenetre* de manière à obtenir l'IHM présentée à la figure figure 1. Les tailles sont données en pixels. L'utilisateur doit pouvoir interagir avec les boutons.

Rappel : pour mettre une couleur de fond à un `JPanel p`, en blanc par exemple, il faut appeler `p.setBackground(Color.WHITE);` .

Réponse : (2.5 pt)

```
pTerrain.setBounds(100,50,600,400);              //0.5pts pour les setBounds
pDir.setBounds(50,500,700,150);

pDir.setBackground(Color.GREEN);          //0.25pts pour les setBackground
pMain.setBackground(Color.YELLOW);

pDir.add(bGauche); //0.75pts pour les add
pDir.add(bDroite);
pDir.add(bHaut);
pDir.add(bBas);
pMain.add(pTerrain);
pMain.add(pDir);

this.setContentPane(pMain);    //0.5pts pour le setContentPane
this.setVisible(true);  //pas grave si ils oublient le set visible

bHaut  .addActionListener(this);  //0.5pts pour les Listener
bBas   .addActionListener(this);
bGauche.addActionListener(this);
bDroite.addActionListener(this);
```

(Q1.3) Dessin dans un JPanel - Quelques questions (1 pts)

Pour simplifier les interactions avec le plateau de jeu *pTerrain*, nous allons remplacer son type `JPanel` par le type `Terrain` défini ci-dessous. La ligne `private JPanel pTerrain` de la classe *Fenetre* est donc remplacée par `private Terrain pTerrain`. Le plateau de jeux à afficher est modélisé par une matrice d'entier de 0 à 3. La signification des entiers est donnée par les constantes $CASE\_VIDE$, $CASE\_MUR$, etc. La classe `Terrain` possède un attribut r de type `Randonneur` qui représente le personnage à déplacer. Cette classe `Randonneur` sera étudiée plus en détails dans la question suivante.

```
public class Terrain extends JPanel {
    final public int CASE_VIDE = 0;
    final public int CASE_MUR = 1;
    final public int CASE_DEBUT = 2;
    final public int CASE_FIN = 3;

    // Creation du plateau de jeu
    final private int[][] CARTE = {
        {1,1,1,1,1,1,1,1,1,1,1},
        {1,0,0,0,0,0,0,0,0,3,1},
        {1,0,0,0,0,0,1,1,0,1,1},
        {1,0,0,0,0,0,1,0,0,0,1},
        {1,0,0,1,1,0,0,0,0,0,1},
        {1,0,0,1,0,1,1,1,0,0,1},
        {1,0,0,1,0,0,0,0,0,0,1},
        {1,2,0,1,0,0,0,0,0,0,1},
        {1,1,1,1,1,1,1,1,1,1,1}
    };

    public Randonneur r;

    public Terrain() {
          r=new Randonneur(CARTE);
    }

    public void paint(Graphics g) {
          int haut = getHeight();
          int larg = getWidth();

          ...A COMPLETER (Q1.4)...

          r.dessiner(g,larg,haut);
     }
}
```

Répondez de manière précise et en 3 lignes max aux questions suivantes :

(a) D'après les tailles illustrées dans la figure 1, quelles sont les valeurs des variables `haut` et `larg` à la sortie de la méthode `paint` ?
(b) A quoi sert la méthode `paint` et quand est-elle appelée ?

Réponse : (1 pt)
(a) `haut`=400 et `larg`=600 0.5pt
(b) Elle gère l'affichage sur le JPanel et est appelée automatiquement quand le JPanel a besoin d'être ré-affiché 0.5pt

(Q1.4) Dessin dans un JPanel - A vous de jouer (2,5 pts)

Complétez la méthode `paint` de la classe `Terrain`. Cette méthode doit afficher un rectangle par cellule du plateau de jeu. La couleur de la cellule dépend de ce qu'elle contient : rouge pour la case de début (Color.RED), vert pour la case de fin (Color.GREEN), noir pour les murs (Color.BLACK) et blanc pour les cases vides (Color.WHITE).
Réponse : (2.5 pt)

```
int haut_carre = haut/CARTE.length;
int larg_carre = larg/CARTE[0].length;

for (int i=0;i<CARTE.length;i++){
  for (int j=0;j<CARTE[0].length;j++){
          if(CARTE[i][j]==CASE_VIDE){
                g.setColor(Color.WHITE);
          }else if(CARTE[i][j]==CASE_MUR){
                g.setColor(Color.BLACK);
          }else if(CARTE[i][j]==CASE_DEBUT){
                g.setColor(Color.RED);
          }else if(CARTE[i][j]==CASE_FIN){
                g.setColor(Color.GREEN);
          }
          int y1 = i*haut_carre;
          int x1 = j*larg_carre;
          g.fillRect(x1, y1, larg_carre,haut_carre);
  }
}
//0.5pts pour la double boucle
//0.5pts pour le calcul correct de la taille des rectangles
//0.5pts pour le calcul correct des coordonnees top-left
//0.5pts pour la couleur correcte suivant les cas
//0.5pts pour l'appel    fillRect
```

(Q1.5) La classe randonneur - Quelques questions (1,5 pts)

Une partie de la classe randonneur est détaillée ci-dessous. Elle représente le personnage qui va être déplacé par le joueur à l'aide des boutons et met à jour l'affichage. Ses attributs x et y représentent respectivement son numéro de colonne et de ligne (en partant du haut) dans le plateau (dans l'exemple de la figure 1, x=3 et y=2).

```java
public class Randonneur {
  private int x;
  private int y;
  private int[][] maCarte;
  private int ptVie;

  public Randonneur(int[][] carte) {
    // Recherche des coordonnees de depart sur la carte pour initialiser x et y
    ...A COMPLETER (Q1.6)...

    // initialisation des autres attributs
    maCarte=carte;
    ptVie=10;
  }

  public void dessiner(Graphics g, int w, int h) {
    int nbLignes = maCarte.length;
    int nbCols = maCarte[0].length;

    ...A COMPLETER (Q1.7)...
  }

  public void marcher(int deltaX, int deltaY) {
    int nbLignes = maCarte.length;
    int nbCols = maCarte[0].length;
    int newx=x+deltaX;
    int newy=y+deltaY;

    if(maCarte[newy][newx] == 1) {
      System.out.println("Obstacle. On ne bouge pas.");
    } else {
        x=newx;
        y=newy;
        if(maCarte[y][x] == 3) {
          System.out.println("Bravo, le randonneur est sorti!");
        }
      }
  }

  public int getX() {return x;}
  public int getY() {return y;}
    ...
}
```

Répondez de manière précise et en 3 lignes max aux questions suivantes :

(a) Quelle est l'utilité des méthodes `getX` et `getY` ?
(b) Que fait la méthode `marcher` ?
(c) Avec les données présentées jusque-là, quelles sont les valeurs des variables *nbLignes* et *nbCols* de la méthode `marcher` ?

Réponse : (1,5 pt)
(a) Ces méthodes sont des accesseurs qui permettent de lire la valeur des attributs protégés (ici avec private) depuis une classe exterieure. 0.5 pts
(b) Elle fait bouger le randonneur d'un deltaX et deltaY et vérifie s'il y a un obstacle ou s'il est arrivé. 0.5 pts
(c) nbLignes=9 et nbCols=11 selon la matrice d'entier 0.5pts
On accepte aussi nbLignes=10 et nbCols=12 si on regarde la figure de la 1ere page

(Q1.6) La classe randonneur - A vous de jouer 1/2 (1 pts)

Complétez le constructeur de la classe `randonneur`, afin d'initialiser les coordonnées x et y en recherchant la case de début (c-à-d la case égale à 2) dans la carte. Attention cette méthode doit fonctionner pour n'importe quelle carte est pas seulement celle présentée en exemple.

Réponse : (1 pt)

```
for(int i=0;i<carte.length;i++)
        for(int j=0;j<carte[i].length;j++)
                if ( carte[i][j]==2) {
                  x=j;
                  y=i;
                }
//on accepte n'importe quelle type de boucle
```

(Q1.7) La classe randonneur - A vous de jouer 2/2 (1 pts)

Donnez l'implémentation de la méthode `dessiner` de la classe `Randonneur`. On rappelle que cette méthode dessine une ellipse bleue représentant le randonneur et est appelée par la méthode `paint` de la classe `Terrain` (voir le code de la classe `Terrain` plus haut).

Réponse : (1 pt)

```
int haut_ellipse = h/nbLignes;
int larg_ellipse = w/nbCols;

int x1 = x*larg_ellipse;
int y1 = y*haut_ellipse;

g.setColor(Color.BLUE);
g.fillOval(x1,y1,larg_ellipse,haut_ellipse);

//0.25 pour le setColor
//0.75 pour le fillOval avec les bonnes valeurs
```

(Q1.8) La méthode ActionPerformed (2 pts)

Donnez l'implémentation de la méthode `ActionPerformed` de la classe `Fenetre`. Cette méthode déplace le randonneur d'une case (vers le haut, le bas, la droite ou la gauche) grâce à la méthode `marcher`, en fonction du clic sur l'un des boutons.

Réponse : (2 pt)

```
public void actionPerformed(ActionEvent e){
        int deltaX=0;
        int deltaY=0;

        if(e.getSource()==bHaut) {deltaX =  0; deltaY = -1;}
        if(e.getSource()==bBas) {deltaX =  0; deltaY =  1;}
        if(e.getSource()==bGauche) {deltaX = -1; deltaY =  0;}
        if(e.getSource()==bDroite) {deltaX =  1; deltaY =  0;}

        pTerrain.r.marcher(deltaX, deltaY);

        pTerrain.repaint();
}
//0.5 pt pour Les 4 conditions avec le getsource
//0.5 pt pour les deltas corrects
//0.5 pour le repaint
//0.5 pour l'appel correct a marcher
```

Exercice 2 Programmation objet et polymorphisme (9.5 pts)

On souhaite maintenant enrichir le jeu en ajoutant des objets spéciaux dans le labyrinthe. Il y en a 3 sortes :

- Les champignons (`Mushroom`) ajoutent 2 points de vie au randonneur. Ils sont représentés par des rectangles roses.
- Les sangliers (`WildBoar`) enlèvent 3 points de vie au randonneur. Ils sont représentés par des rectangles gris.
- Le puit magique (`MagicHole`) téléporte le randonneur sur une case libre. Il est représenté par un rectangle Jaune.

Ils sont organisés en hiérarchie de classes telle qu'illustrée sur la figure 2. Cette figure (à gauche) illustre également leur affichage.

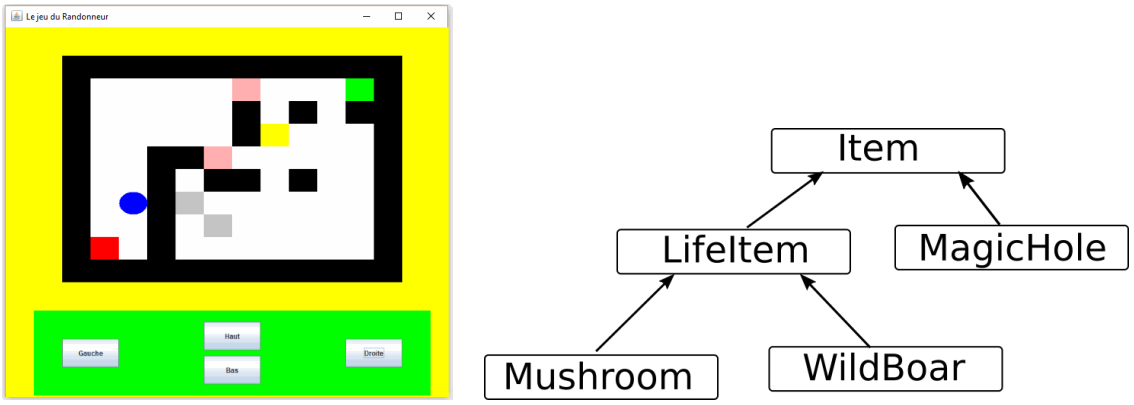

FIGURE 2 – Gauche : illustration de l'IHM du jeu avec les objets spéciaux. Droite : illustration de la hiérarchie de classe.

(Q2.1) Item et MagicHole (4 pts)

La classe *Item* est détaillée en partie ci-dessous.

```
public ... class Item {

  protected Color maCouleur;
  protected int x,y;
  private int nbLignes;
  private int nbCols;
  public Item(int[][] carte){
    nbLignes = carte.length;
    nbCols = carte[0].length;
    ...A COMPLETER (Q2.1)...
  }

  public abstract void action(Randonneur r);

  public void dessiner(Graphics g,int w, int h){
        ...dessin d'un rectangle avec maCouleur (code non demande)...

  }
}
```

(a) La classe `Item` est-elle abstraite ? Justifiez votre réponse.
(b) Complétez le constructeur de la classe `Item`. Ce constructeur détermine une position (x,y) en choisissant aléatoirement une case sur la plateau de jeu donné en paramètre. Attention cette case devra être une case vide.

Rappel : `(int)(x * Math.random())` renvoie un entier aléatoire dans [0, x-1].

(c) Donnez l'implémentation du constructeur de la classe `MagicHole` qui hérite de `Item`.
(d) Est-ce que la classe `MagicHole` contient une implémentation de la méthode action ? Si oui, donnez son entête (son implémentation n'est pas demandée).

Réponse : (4 pts)

```
a) Oui la classe est abstraite car elle contient une methode abstraite  0.5pts

b) 2 pts
public Item(int[][] carte){
  nbLignes = carte.length;
  nbCols = carte[0].length;
  int newx;
  int newy;
  do{
    newx = (int)(carte[0].length * Math.random());
    newy = (int)(carte.length * Math.random());
  }while(carte[newy][newx]!=0);

  x=newx;
  y=newy;
}

//Boucle while ou do..while avec le bon critere d'arret 0.75pts
//Generation correct des position aleatoire 0.75pt
//Remplissage correct de x et y 0.5pts

c) 1 pts
public MagicHole(int [][]carte){  //entete correcte, c-a-d avec carte 0.25pts
        super(carte);     //appel de Super correctement 0.5pts
        maCouleur=Color.YELLOW; //affectaton de la couleur 0.25 pts

}

d) Oui la classe MagicHole contient une implementation de la methode Action.
Son entete est la meme sans abstract. 0.5pts
```

(Q2.2) LifeItem et Mushroom (2 pts)

La classe `LifeItem` rassemble les classes qui modifient les points de vie du randonneur (`Mushroom` et `WildBoar`). Une partie de son implémentation est donnée ci-dessous.

```
public abstract class  LifeItem extends Item{

  protected int lifeModification;

  public LifeItem(int[][] carte, int alife){
    ...Initialisation des attributs x,y et lifeModification (code non demande)...
  }

  public void Action(Randonneur r){
    if(r.getX()==x && r.getY()==y){
          r.modifPtVie(lifeModification);
    }
  }
}
```

(a) Donnez l'implémentation de la classe `Mushroom`
(b) A quelle classe appartient la méthode *modifPtVie* appelée dans la méthode *action* de

`LifeItem` ? Donnez son entête et son implémentation.

Réponse : (2 pt)

```
a) 1 pt

public class Mushroom extends LifeItem{ //0.25pt pour le extends

  public Mushroom(int[][] carte){    //0.25pt pour l'entete correcte (sans parametre supplementaire)
    super(carte,2);                                    //0.25pt pour l'appel a super
    maCouleur=Color.PINK;                          //0.25pt pour a couleur
  }
}
//penalisez si l'etudiant rajoute la methode action.


b) modifPtVie appartient a la classe randonneur 0.5pts

c) 0.5pts
public void modifPtVie(int modif) {
  ptVie=ptVie+modif;

}
```

(Q2.3) Le tableau d'Items (3,5 pts)

Les objets spéciaux présents dans le jeu sont représentés par un tableau d'objets de la classe `Item` en attribut de la classe `Terrain` : *Item [] listItem ;*
(a) Que faut-il ajouter dans la méthode *paint* de `Terrain` pour afficher ces objets spéciaux ? Pour cette question (a), on suppose que la méthode dessiner de la classe `Item` existe déjà.
(b) Dans quelle classe et quelle méthode appelleriez vous les méthodes *action* des objets spéciaux ? Quel serait le code à ajouter ?

Réponse : (3,5 pt)

```
a) 1 pts

for(int i=0;i<listItem.length;i++) {     //0.5 pts
  listItem[i].dessiner(g,larg,haut);    // 0.5 pts
}

b)
Reponse a la question 1pt:

Il faut appeler action dans le actionPerformed de la fenetre 0.5pts
Il faut bien l'appeler *apres* l'appel de marcher 0.5pts
Il n'y a pas vraiment d'autres solutions sans modifier le code fourni
Comptez faux si action est apellee dans le paint de Terrain

code a ajouter: 1,5pt

for(int i=0;i<pTerrain.listItem.length;i++)  //0.5 pts
        pTerrain.listItem[i].Action(pTerrain.r);  //1 pts
```