EXAMEN DE FIN D'ÉTUDES SECONDAIRES GÉNÉRALES **Sessions 2023 – QUESTIONNAIRE ÉCRIT**					
Date :	21.09.23	*Durée :*	08:15 - 11:15	*Numéro candidat :*	
Discipline :	Informatique	*Section(s) :*	GIG		

Dans votre répertoire de travail (à définir par chaque lycée), vous trouverez un dossier nommé **EXAMEN_GIG**. Renommez ce dossier en remplaçant le nom par votre numéro de candidat (exemple de notation : **LXY_GIG2_07**). Tous vos fichiers devront être sauvegardés à l'intérieur de ce dossier, qui sera appelé **votre dossier** dans la suite !

Vous trouvez une version exécutable du programme – ***Demo.jar*** – dans votre dossier.
Avant de commencer, il est recommandé de lancer et d'essayer ce programme.

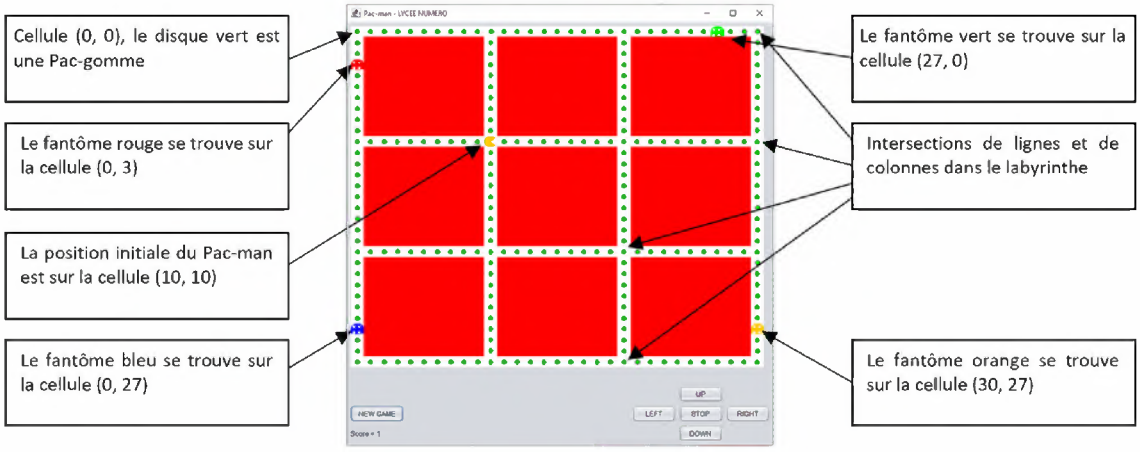
En cas de non-respect de la nomenclature, des conventions du cours, des indentations, des indications de l'UML, etc., jusqu'à 3 points pourront être retranchés de la note finale.

Le jeu du Pac-man — 2 + 4 + 16 + 11 + 6 + 12 + 1 + 8 = 60 p.

Il s'agit de développer le projet **Pacman** qui est une version simplifiée du jeu vidéo Pac-man. Dans ce jeu, le Pac-man se déplace à l'intérieur d'un labyrinthe pour manger des Pac-gommes. Quatre fantômes lui rendent la vie plus difficile en essayant de l'attraper. Le jeu se termine lorsque le Pac-man a mangé toutes les Pac-gommes ou lorsqu'il a été touché par un des fantômes.

Pac-man

Le labyrinthe (**maze**) est organisé sous forme d'une liste de cellules (**cells**) qui représentent des éléments de route, contenant initialement des Pac-gommes. Chaque cellule se trouve à une position (**colPos**, **rowPos**). **colPos** décrit la position horizontale et **rowPos** décrit la position verticale. Chaque cellule a une largeur en *pixels* (**width**) et une hauteur en *pixels* (**height**). En se déplaçant dans le labyrinthe, le Pac-man mange chaque Pac-gomme qu'il trouve en cours de route. Le Pac-man et les fantômes ne peuvent se déplacer que sur les éléments de route. Il en résulte qu'un déplacement ne peut être que horizontal ou vertical.



1 Génération automatique de code 2 p.

2 La classe Cell 1 + 1 + 2 = 4 p.

Implémentez la classe **Cell** – avec ses attributs, accesseurs et manipulateurs requis – en vous basant sur le diagramme UML donné (voir page 7) et les indications supplémentaires ci-dessous.

2.1 Attributs

- `colPos` indique la position horizontale et `rowPos` la position verticale de la cellule,
- `cellWidth` et `cellHeight` indiquent les dimensions de la cellule en *pixels*,
- `food` indique s'il y a une Pac-gomme sur cette cellule (`food=1`) ou pas (`food=0`).

2.2 Méthodes

- Le constructeur initialise les attributs `colPos` et `rowPos` à la valeur du paramètre respectif. Ensuite `cellWidth` et `cellHeight` sont initialisés à 10 et `food` à 1. (1 p.)
- `updateDimensions` met la valeur des attributs `cellWidth` et `cellHeight` à la valeur du paramètre respectif. (1 p.)
- `draw` dessine la cellule sous forme d'un rectangle blanc, rempli, de dimensions `cellWidth` et `cellHeight` à la position correspondante aux attributs `colPos` et `rowPos`. Si `food` est supérieur à 0, un disque vert, de bord noir et de rayon 3 *pixels* est dessiné au centre de la cellule. (2 p.)

3 La classe Maze 3 + 6 + 4 + 1 + 1 + 1 = 16 p.

Implémentez la classe **Maze** – avec ses attributs et accesseurs requis – en vous basant sur diagramme UML donné (voir page 7) et les indications supplémentaires ci-dessous.

3.1 Attributs

- `rowCount` représente le nombre de lignes du labyrinthe,
- `colCount` représente le nombre de colonnes du labyrinthe,
- `alCells` représente le labyrinthe qui est une liste de cellules.

3.2 Méthodes

- Le constructeur
  - initialise les attributs `colCount` et `rowCount` à la valeur du paramètre respectif ;
  - remplit la liste `alCells` avec les cellules du jeu, en observant que `colPos` $\in$ [0, `colCount-1`], `rowPos` $\in$ [0, `rowCount-1`] et qu'au moins une des deux coordonnées doit être un multiple de 10. (3 p.)
- `get` recherche et retourne la première cellule de `alCells` avec les coordonnées indiquées ou `null` si aucune cellule n'a été trouvée. (6 p.)
- `getRemainingFood` calcule et retourne la somme de `food` de toutes les cellules. (4 p.)
- `isValidCell` vérifie si `alCells` contient une cellule aux coordonnées indiquées. (1 p.)
- `updateDimensions` met à jour les dimensions de toutes les cellules de `alCells`. (1 p.)
- `draw` dessine toutes les cellules. (1 p.)

4 La classe Ghost 2 + 1 + 5 + 3 = 11 p.

Implémentez la classe **Ghost**, héritée de **Cell** – avec ses attributs requis – en vous basant sur le diagramme UML donné (voir page 7) et les indications supplémentaires ci-dessous.

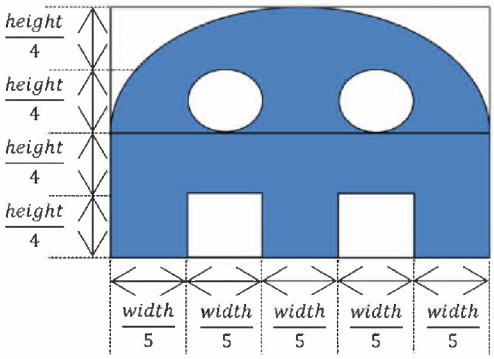
4.1 Attributs

- `colStep` : dans le labyrinthe, déplacement d'une position vers la gauche (`colStep = -1`), vers la droite (`colStep = 1`) ou aucun déplacement horizontal (`colStep = 0`),
- `rowStep` : dans le labyrinthe, déplacement d'une position vers le haut (`rowStep = -1`), vers le bas (`rowStep = 1`) ou aucun déplacement vertical (`rowStep = 0`),
- `color` : la couleur du fantôme.

4.2 Méthodes

- `setRandomSteps` recalcule des valeurs aléatoires pour `colStep` et `rowStep`, sachant que le déplacement est toujours de 1 pas et ne peut être qu'horizontal ou vertical, mais ni diagonal, ni nul. (2 p.)
- Le constructeur initialise les attributs `colPos`, `rowPos` et `color` à la valeur du paramètre respectif, puis fait attribuer des valeurs aléatoires valides à `colStep` et `rowStep`. (1 p.)
- `move` vérifie si le fantôme se trouve actuellement sur une intersection (voir graphique page 1) d'une ligne et d'une colonne du labyrinthe. Dans ce cas, des nouveaux pas de déplacement aléatoires sont calculés. Tant que la position à laquelle le fantôme se trouverait après le déplacement n'est pas valide, on continue à changer aléatoirement les pas de déplacement. Finalement, on attribue la nouvelle position au fantôme. (5 p.)
- `draw` dessine le fantôme en couleur `color` à la position correspondant aux attributs `colPos` et `rowPos` (voir figure ci-contre).

  Aucun bord ou trait ne doit être dessiné. Ils servent uniquement à illustrer la construction du fantôme. Il convient de dessiner les différentes parties dans l'ordre suivant :
  1. l'ellipse qui représente la tête,
  2. le rectangle qui sert à dessiner la partie inférieure,
  3. les carrés et ellipses de couleur blanche. (3 p.)



5 La classe Pacman 1 + 1 + 2 + 1 + 1 = 6 p.

Implémentez la classe **Pacman**, héritée de **Cell** – avec ses attributs, son accesseur et son manipulateur requis – en vous basant sur le diagramme UML donné (voir page 7) et les indications supplémentaires ci-dessous.

5.1 Attributs

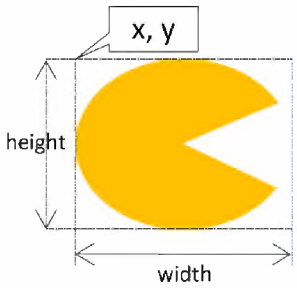
- `colStep` : dans le labyrinthe, déplacement d'une position vers la gauche (`colStep = -1`), vers la droite (`colStep = 1`) ou aucun déplacement horizontal (`colStep = 0`) ;
- `rowStep` : dans le labyrinthe, déplacement d'une position vers le haut (`colStep = -1`), vers le bas (`colStep = 1`) ou aucun déplacement vertical (`colStep = 0`) ;
- `score` : nombre de Pac-gommes que le Pac-man a mangées.

5.2 Méthodes

- Le constructeur fait appel au constructeur hérité pour initialiser la taille du Pac-man, sa position dans le labyrinthe et lui attribue la couleur orange. Au début, le Pac-man est immobile et n'a rien mangé. (1 p.)
- `setHeading` redéfinit `colStep` et `rowStep` à partir du paramètre respectif. (1 p.)
- `move`
  - vérifie si le Pac-man se trouve sur une cellule avec une Pac-gomme. Dans ce cas, il mange la Pac-gomme et gagne un point de score. La cellule concernée ne contient alors plus de Pac-gomme ;
  - calcule la nouvelle position du Pac-man en fonction de `colStep` et `rowStep`. Si celle-ci se trouve sur une cellule valide, alors elle est enregistrée. (2 p.)
- `collidesWith` retourne `true` si le Pac-man se trouve à la même position que le fantôme (passé comme paramètre) et `false` sinon. (1 p.)
- `draw` dessine le Pac-man en couleur **color** à la position correspondant aux attributs `colPos` et `rowPos` (voir figure ci-contre). (1 p.)

  En supposant que les entiers `x` et `y` contiennent les coordonnées (en *pixels*) en haut à gauche de l'ellipse, l'arc de cercle orange ci-contre est tracé avec l'instruction suivante :

  ```
  g.fillArc(x, y, width, height, 40, 280);
  ```



6 La classe Game — 2 + 2 + 2 + 1 + 2 + 3 = 12 p.

Implémentez la classe **Game** – avec ses attributs et accesseurs requis – en vous basant sur le diagramme UML donné (voir page 7) et les indications supplémentaires ci-dessous.

6.1 Attributs

- `alGhosts` : la liste des fantômes,
- `maze` : le labyrinthe,
- `pacman` : le Pac-man,
- `colCount` : nombre de cellules en direction horizontale,
- `rowCount` : nombre de cellules en direction verticale,
- `gameOver` : l'état du jeu (`true` si le jeu est terminé),
- `message` : chaîne de caractères indiquant le score.

6.2 Méthodes

- Le constructeur
  - initialise `colCount` et `rowCount` à la valeur du paramètre respectif ;
  - crée un nouveau labyrinthe aux dimensions indiquées ;
  - crée et ajoute à la liste des fantômes les 4 fantômes, leurs positions étant les quatre coins du labyrinthe et leurs couleurs étant le rouge, le vert, le bleu et l'orange respectivement ;
  - crée le Pac-man à la position (10,10) ;
  - le jeu n'est pas terminé et le contenu de `message` est « `Score = 0` ». (2 p.)

- `checkGameState` réalise les étapes suivantes :
  - si le labyrinthe ne contient plus de Pac-gommes, le jeu est terminé et le contenu de `message` est « `Game over, you won. Score =` », suivi du score ;
  - sinon, si le Pac-man a touché un des fantômes, le jeu est terminé, la couleur du Pac-man devient rouge et le contenu de `message` est « `Game over, you lost. Score =` », suivi du score. (2 p.)
- `move` réalise les étapes suivantes :
  - fait bouger tous les fantômes ;
  - vérifie l'état du jeu ;
  - si le jeu n'est pas terminé le Pac-man bouge et le contenu de `message` est « `Score =` », suivi du score ;
  - revérifie l'état du jeu une deuxième fois. (2 p.)
- `setPacmanHeading` redéfinit les pas de déplacement du Pac-man. (1 p.)
- `updateDimensions` recalcule les dimensions du labyrinthe, des fantômes et du Pac-man en fonction des dimensions du canevas passées par paramètre. (2 p.)
- `draw` effectue les étapes suivantes :
  - recalcule les dimensions de tous les éléments du jeu en fonction des dimensions du canevas passées par paramètre ;
  - dessine la surface du jeu uniquement (et non pas du canevas entier) de couleur rouge si le jeu est en cours et de couleur bleue sinon ;
  - dessine le labyrinthe, tous les fantômes et le Pac-man. (3 p.)

7 La classe DrawPanel 1 p.

Implémentez la classe **DrawPanel** – avec son attribut et manipulateur requis – en vous basant sur le diagramme UML donné et l'indication supplémentaire ci-dessous.

7.1 Méthode

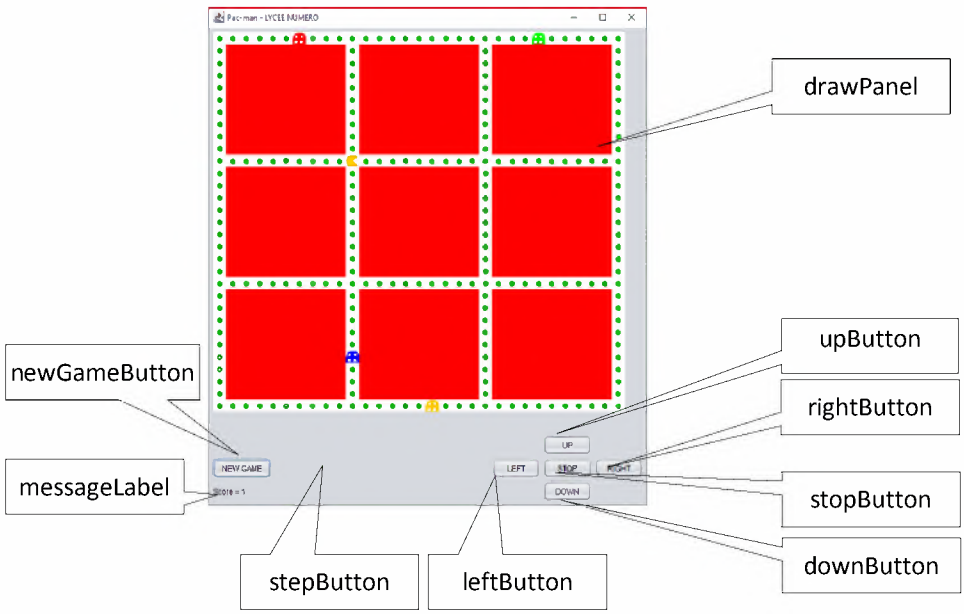
- La méthode `paintComponent` dessine, si possible, le jeu entier. (1 p.)

8 La classe MainFrame 2 + 1 + 1 + 1 + 1 + 1 + 1 = 8 p.

Implémentez la classe **MainFrame** – avec ses attributs requis – en vous basant sur le diagramme UML donné et les indications supplémentaires ci-dessous.

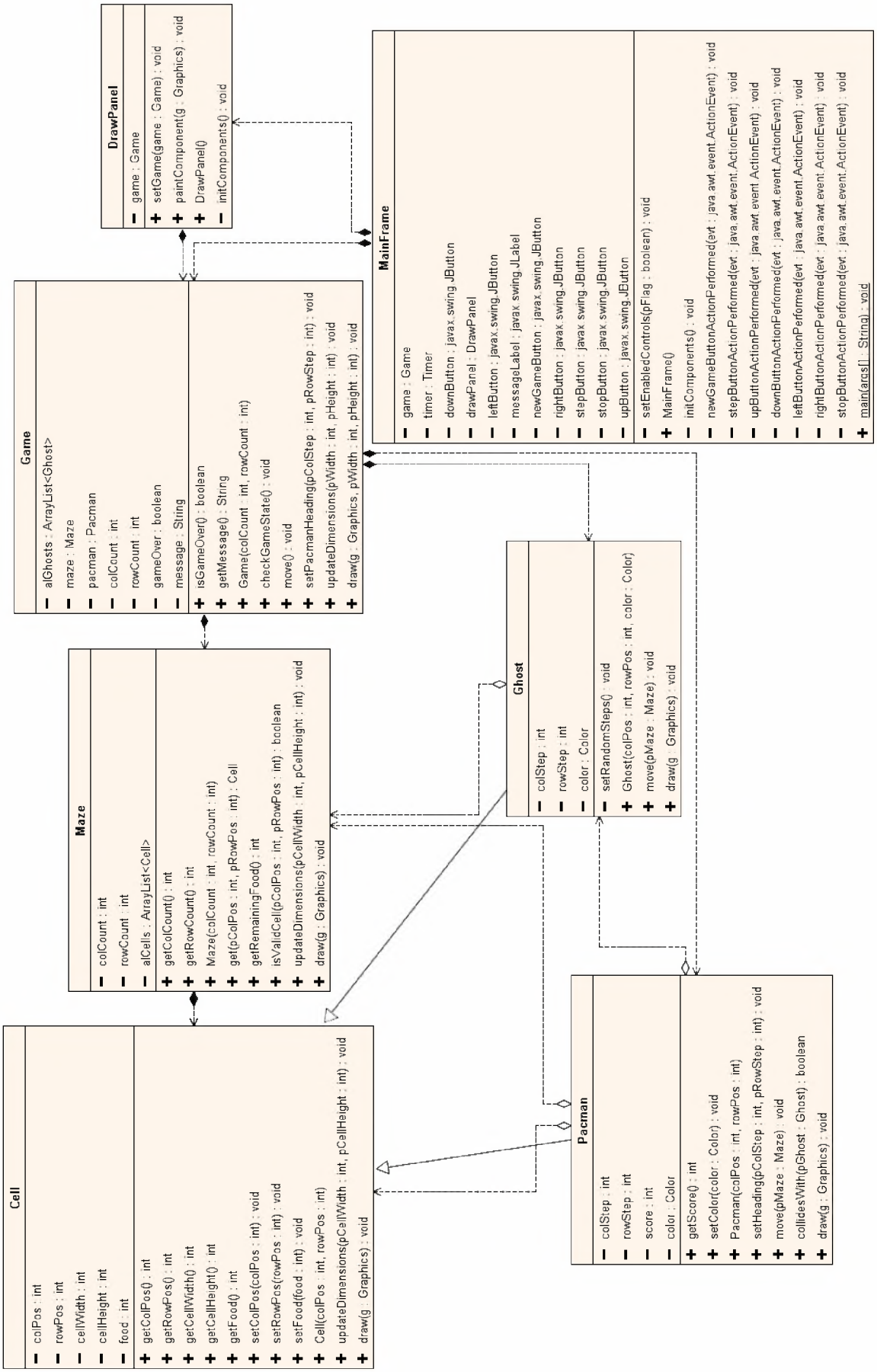
8.1 Graphical User Interface

Reproduisez fidèlement l'interface de la classe **MainFrame**, illustrée ci-dessous et respectez la nomenclature indiquée. Ajoutez le titre « Pacman » au programme suivi de votre numéro de candidat. (2 p.)



8.2 Méthodes

- `setEnabledControls` rend actifs / inactifs les cinq boutons qui contrôlent le Pac-man selon que la valeur du paramètre `pFlag` est vrai / faux. (1 p.)
- Le constructeur crée une instance du chronomètre, associé au bouton caché **stepButton** et à raison de 5 appels par seconde. (1 p.)
- Pour le bouton intitulé **New Game**, implémentez les actions suivantes :
  - arrêtez le chronomètre,
  - créez un nouveau jeu avec un labyrinthe de dimensions 31 x 31,
  - activez les cinq boutons qui contrôlent le mouvement du Pac-man,
  - lancez le chronomètre. (1 p.)
- Pour le bouton **stepButton**, implémentez les instructions suivantes :
  - faites bouger le jeu,
  - si le jeu est terminé alors arrêtez le chronomètre et désactivez les cinq boutons qui contrôlent le mouvement du Pac-man,
  - mettez à jour toute l'interface. (2 p.)
- Les cinq **boutons de contrôle** du Pac-man modifient sa direction de mouvement. Seulement un pas de -1, 0 ou 1 est permis à l'horizontale ou à la verticale. (1 p.)


DrawPanel
- game : Game
+ setGame(game : Game) : void
+ paintComponent(g : Graphics) : void
+ DrawPanel()
- initComponents() : void
MainFrame
- game : Game
- timer : Timer
- downButton : javax.swing.JButton
- drawPanel : DrawPanel
- leftButton : javax.swing.JButton
- messageLabel : javax.swing.JLabel
- newGameButton : javax.swing.JButton
- rightButton : javax.swing.JButton
- stepButton : javax.swing.JButton
- stopButton : javax.swing.JButton
- upButton : javax.swing.JButton
- setEnabledControls(pFlag : boolean) : void
+ MainFrame()
- initComponents() : void
- newGameButtonActionPerformed(evt : java.awt.event.ActionEvent) : void
- stepButtonActionPerformed(evt : java.awt.event.ActionEvent) : void
- upButtonActionPerformed(evt : java.awt.event.ActionEvent) : void
- downButtonActionPerformed(evt : java.awt.event.ActionEvent) : void
- leftButtonActionPerformed(evt : java.awt.event.ActionEvent) : void
- rightButtonActionPerformed(evt : java.awt.event.ActionEvent) : void
- stopButtonActionPerformed(evt : java.awt.event.ActionEvent) : void
+ main(args[] : String) : void
Game
- alGhosts : ArrayList<Ghost>
- maze : Maze
- pacman : Pacman
- colCount : int
- rowCount : int
- gameOver : boolean
- message : String
+ isGameOver() : boolean
+ getMessage() : String
+ Game(colCount : int, rowCount : int)
+ checkGameState() : void
+ move() : void
+ setPacmanHeading(pColStep : int, pRowStep : int) : void
+ updateDimensions(pWidth : int, pHeight : int) : void
+ draw(g : Graphics, pWidth : int, pHeight : int) : void
Maze
- colCount : int
- rowCount : int
- alCells : ArrayList<Cell>
+ getColCount() : int
+ getRowCount() : int
+ Maze(colCount : int, rowCount : int)
+ get(pColPos : int, pRowPos : int) : Cell
+ getRemainingFood() : int
+ isValidCell(pColPos : int, pRowPos : int) : boolean
+ updateDimensions(pCellWidth : int, pCellHeight : int) : void
+ draw(g : Graphics) : void
Ghost
- colStep : int
- rowStep : int
- color : Color
- setRandomSteps() : void
+ Ghost(colPos : int, rowPos : int, color : Color)
+ move(pMaze : Maze) : void
+ draw(g : Graphics) : void
Cell
- colPos : int
- rowPos : int
- cellWidth : int
- cellHeight : int
- food : int
+ getColPos() : int
+ getRowPos() : int
+ getCellWidth() : int
+ getCellHeight() : int
+ getFood() : int
+ setColPos(colPos : int) : void
+ setRowPos(rowPos : int) : void
+ setFood(food : int) : void
+ Cell(colPos : int, rowPos : int)
+ updateDimensions(pCellWidth : int, pCellHeight : int) : void
+ draw(g : Graphics) : void
Pacman
- colStep : int
- rowStep : int
- score : int
- color : Color
+ getScore() : int
+ setColor(color : Color) : void
+ Pacman(colPos : int, rowPos : int)
+ setHeading(pColStep : int, pRowStep : int) : void
+ move(pMaze : Maze) : void
+ collidesWith(pGhost : Ghost) : boolean
+ draw(g : Graphics) : void

Enseignement secondaire général
Division technique générale – Section technique générale
Examen 1GIG

Liste des composants et classes connus

Liste des composants (propriétés, événements et méthodes) et classes à connaître pour l'épreuve en informatique à l'examen de fin d'études secondaires générales - division technique générale.

Package	Classe	Details	Remarques / Constantes
javax.swing	JFrame	Méthodes - setTitle(...) / getTitle() NetBeans Object Inspector Property - title	
	JButton JLabel JTextField	Méthodes - setText(...) / getText() - setVisible(...) - setEnabled(...) Événement - actionPerformed NetBeans Object Inspector Property - icon	- le libellé *JLabel* peut aussi être utilisé pour visualiser des images via la propriété « icon » de l'inspecteur objet de NetBeans (le composant *JTextField* ne possède pas de propriété « icon »).
	JSlider	Méthodes - setMinimum(...) / getMinimum() - setMaximum(...) / getMaximum() - setValue(...) / getValue() Événement - stateChanged	
	JPanel	Méthodes - setVisible(...) - setBackground(...) / getBackground() - getWidth() / getHeight() - paintComponent(Graphics g) - repaint() Événements - MousePressed / MouseReleased - MouseDragged / MouseMoved	- *JPanel* est utilisé pour regrouper d'autres composants visuels et pour réaliser des dessins. - Lors de la réalisation de dessins, la méthode **`public void paintComponent (Graphics g)`** est à surcharger.
javax.swing	JList	Méthodes - setListData(...) - getSelectedIndex() / setSelectedIndex(...) Événement - valueChanged NetBeans Object Inspector - *Properties* - model - selectionMode (SINGLE) NetBeans Object Inspector - *Code* - Type Parameters : - *(vide)*	- *JList* est utilisé surtout pour afficher le contenu d'une liste *ArrayList*.
java.awt.event	ActionEvent	- Ce type d'objet est uniquement utilisé dans les méthodes de réaction ajoutées de manière automatique à l'aide de NetBeans.	
	MouseEvent	Méthodes - getX() / getY() - getPoint() - getButton()	Constantes - BUTTON1 - BUTTON2 - BUTTON3

Package	Classe	Details	Remarques
javax.swing	Timer	Constructeur - Timer(int,ActionListener) Méthodes - start() - stop() - setDelay(...) - isRunning()	
		- Comme *ActionListener* on utilisera de préférence celui d'un bouton. Exemple : `timer = new Timer(1000,stepButton.getActionListeners()[0]);`	
java.awt	Graphics	Méthodes - drawLine(...) - drawOval(...) / fillOval(...) - drawRect(...) / fillRect(...) - drawString(...) - setColor(...) / getColor()	
	Color	Constructeurs - Color(...)	
	Point	Constructeurs - Point(...) Attributs (publics) - x et y Méthodes - getLocation() / setLocation(...)	
java.util	ArrayList	Méthodes - add(...) - clear() - contains(...) - get(...) - indexOf(...) - remove(...) - set(...) - size() - isEmpty() - toArray()	- `Object[] toArray()` est employé uniquement pour passer les contenus d'une liste à la méthode *setListData(...)* d'une *JList*.
java.lang	String	Méthodes - equals(...) / compareTo(...) - contains(...) - valueOf(...)	
	Integer Double	Méthodes - equals(...) / compareTo(...) - valueOf(...)	
	Math	Méthodes - abs(...) - round(...) - random() - sqrt(...) - pow(...) - sin(...), cos(...), tan(...)	Constante: - PI
	System	Méthode - out.print() - out.println()	