EXAMEN DE FIN D'ÉTUDES SECONDAIRES – Sessions 2024

QUESTIONNAIRE

Date :	20.09.24		*Horaire :*	08:15 - 11:15	*Durée :*	180 minutes
Discipline :	INFOR	*Type :*	écrit	*Section(s) :*	GIG	
				Numéro du candidat :		

Dans votre répertoire de travail (à définir par chaque lycée), vous trouverez un sous-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 sous-dossier, qui sera appelé **votre dossier** dans la suite !

Vous trouvez une version exécutable du jeu – **Demo.jar** – dans votre dossier.

Avant de commencer, il est recommandé de lancer et d'essayer ce jeu.

Préliminaire :

- dans le présent projet, ne créez pas d'autres attributs que ceux indiqués dans le questionnaire et illustrés dans le diagramme UML ;
- chaque classe fille doit, pour autant que possible, se servir des méthodes de sa classe mère ;
- ajoutez, tout en haut de chaque classe, votre code de l'examen en commentaire.

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.

Principe du jeu 1GIGInvaders — 60 points

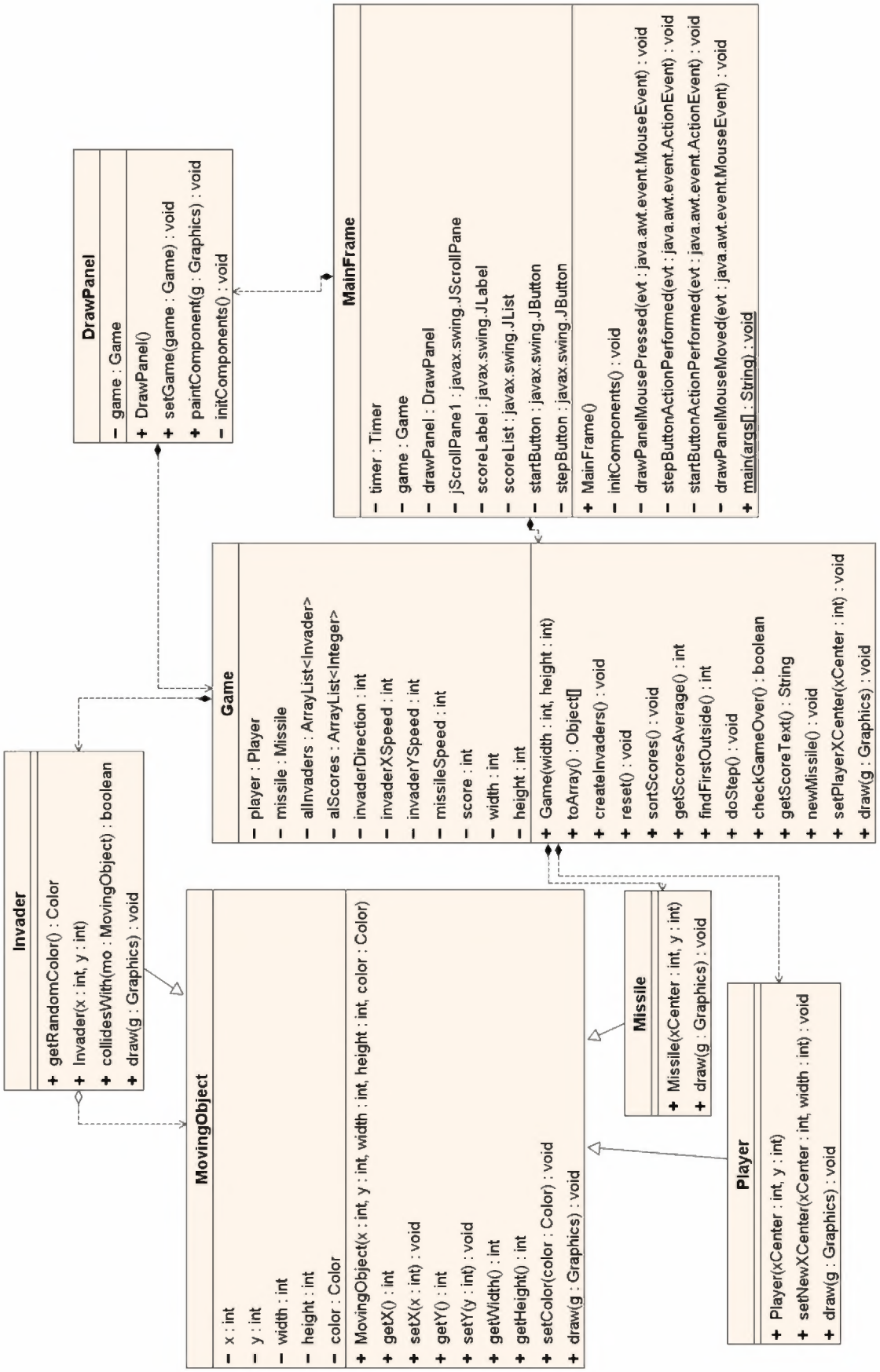
Dans la suite vous allez développer le jeu 1GIGInvaders (voir capture d'écran ci-dessous). Dans ce jeu il s'agit d'éliminer autant d'envahisseurs que possible avec des missiles. Le joueur peut bouger à gauche et à droite avec la souris et tirer avec le bouton gauche. Tant qu'un missile est en cours de route, il n'est pas possible d'en tirer un nouveau.

IMPORTANT :

Créez dans votre dossier un nouveau projet nommé **`1GIGInvaders`** et développez le projet en vous basant sur la version exécutable fournie, ainsi que sur le diagramme UML ci-dessous tout en respectant les instructions et précisions données dans la suite.

Toutes vos classes devront être sauvegardées à l'intérieur de ce projet.

Diagramme UML du jeu 1GIGInvaders



La classe `MovingObject` — 1 point

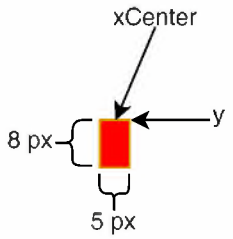
Implémentez la classe **`MovingObject`** en vous basant sur le diagramme UML donné et les indications supplémentaires ci-dessous.

Les coordonnées **`x`** et **`y`** représentent le coin supérieur gauche de l'objet, **`width`** et **`height`** sa largeur et hauteur, **`color`** sa couleur.

- Implémentez les attributs, le constructeur ainsi que les accesseurs et manipulateurs. 0,5p
- La méthode **`draw`** ne fixe que la couleur de dessin. 0,5p

La classe `Missile` — 1,5 points

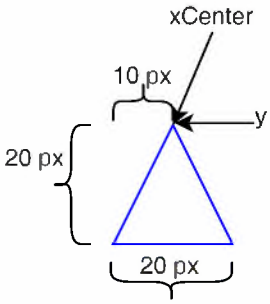
La classe **`Missile`** représente le missile tiré par le joueur. Implémentez la classe **`Missile`** en vous basant sur le diagramme UML donné et les indications supplémentaires ci-dessous.



- Le constructeur initialise le missile avec les valeurs indiquées sur l'image ci-contre. Le paramètre **`xCenter`** représente le centre horizontal du missile. Le missile est de couleur rouge. 0,5p
- La méthode **`draw`** dessine le missile. 1p

La classe `Player` — 3,5 points

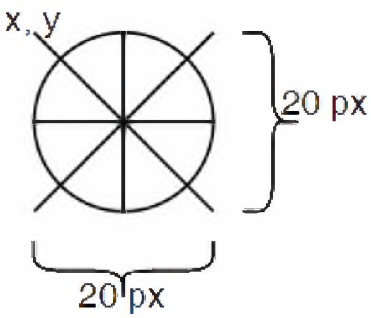
La classe **`Player`** représente le joueur. Implémentez la classe **`Player`** en vous basant sur le diagramme UML donné et les indications supplémentaires ci-dessous.



- Le constructeur initialise le joueur avec les valeurs indiquées sur l'image ci-contre. Sa couleur est bleue. 0,5p
- La méthode **`setNewXCenter`** place le joueur à la nouvelle position, mais seulement si celle-ci maintient le joueur entièrement à l'intérieur de l'aire de jeu dont la largeur est passée en paramètre. 1p
- La méthode **`draw`** dessine le joueur. 2p

La classe `Invader` — 6 points

La classe **`Invader`** représente un envahisseur. Implémentez la classe **`Invader`** en vous basant sur le diagramme UML donné et les indications supplémentaires ci-dessous.



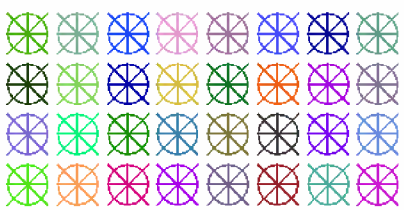
- La méthode **`getRandomColor`** retourne une couleur aléatoire. 1p
- Le constructeur initialise un envahisseur avec les valeurs indiquées sur l'image ci-contre et de couleur noire. Ensuite il change la couleur en couleur aléatoire. 1p

- La méthode **`collidesWith`** détermine si l'envahisseur touche un objet donné, c.-à-d. si les rectangles englobants se touchent. Dans ce cas elle retourne **`true`**, sinon **`false`**. 1,5p
- La méthode **`draw`** dessine l'envahisseur comme sur l'image ci-dessus. 2,5p

La classe Game — 35 points

La classe **`Game`** gère le jeu. Implémentez la classe **`Game`** en vous basant sur le diagramme UML donné et les indications supplémentaires ci-dessous.

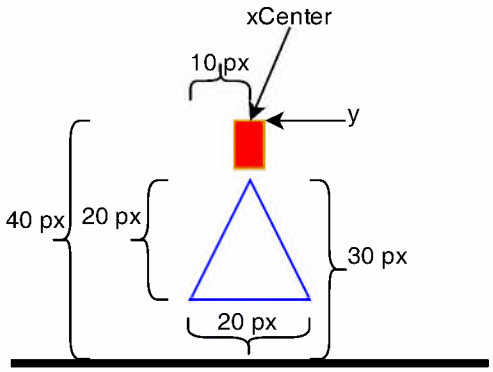
- Implémentez les attributs et le constructeur sachant que :
  1. **`player`** représente le joueur ;
  2. **`missile`** représente un missile, s'il y en a un ;
  3. **`alInvaders`** contient les envahisseurs qui existent encore ;
  4. **`alScores`** contient les scores ;
  5. **`invaderDirection`** prend initialement la valeur 1 pour indiquer que les envahisseurs se déplacent de la gauche vers la droite. Cet attribut prend la valeur -1 quand les envahisseurs se déplacent de la droite vers la gauche ;
  6. **`invaderXSpeed`** prend initialement la valeur 5 et représente la vitesse horizontale des envahisseurs, c'est-à-dire le nombre de pixels que chaque envahisseur bouge vers la droite ou la gauche ;
  7. **`invaderYSpeed`** a une valeur de 10 et représente le nombre de pixels que tous les envahisseurs se déplacent vers le bas lorsqu'au moins un a atteint le bord droit ou gauche ;
  8. **`missileSpeed`** a une valeur de 10 et représente le nombre de pixels qu'un missile se déplace vers le haut lorsqu'il est en cours de route ;
  9. **`score`** est initialisé à 0 et représente le score actuel du joueur ;
  10. **`width`** et **`height`** représentent les dimensions de l'aire de jeu. 0,5p
- La méthode **`createInvaders`** vide la liste et ajoute 32 envahisseurs. Ils sont placés en 4 rangées de 8 avec 5 pixels de distance horizontale entre les envahisseurs et 5 pixels de distance verticale entre les 4 rangées (voir le graphique à droite). Le premier se trouve à 10 pixels du bord gauche et à 10 pixels du bord supérieur de l'aire de jeu. 2p



- La méthode **`reset`** redonne les valeurs initiales aux attributs **`invaderXSpeed`**, **`score`** et **`missile`**. Elle crée un nouveau joueur **`player`** situé au milieu de l'aire de jeu et à 30 pixels du bas (voir image de la page suivante), ainsi que les envahisseurs. 2,5p
- La méthode **`sortScores`** trie les scores par ordre croissant par la méthode de sélection directe. 4,5p
- La méthode **`getScoresAverage`** retourne la moyenne des scores ou 0 si la liste est vide. 3p
- La méthode **`findFirstOutside`** retourne l'index du premier envahisseur qui, lors de son prochain mouvement, toucherait le bord droit ou gauche. S'il n'y en a pas elle retourne -1. 3p
- La méthode **`doStep`** est responsable de la gestion d'une étape du jeu.

1. Si un missile est en cours de route, la méthode déplace le missile. Ensuite la méthode vérifie si le missile a quitté complètement l'aire de jeu. Si c'est le cas, le missile est enlevé, sinon la méthode efface le premier envahisseur qui a été touché. Dans ce cas, le score est incrémenté et le missile enlevé. S'il ne reste plus d'envahisseurs, la vitesse horizontale des envahisseurs est augmentée de 50 % (la partie décimale est à ignorer) et de nouveaux envahisseurs sont créés.
2. Si un envahisseur toucherait le bord droit ou gauche lors de son prochain mouvement, la direction de déplacement est inversée.
3. Tous les envahisseurs sont déplacés horizontalement. Si la direction de déplacement a changé, ils sont aussi déplacés verticalement. Chaque envahisseur qui quitte ainsi entièrement le canevas vers le bas est enlevé. 12p

- La méthode **`checkGameOver`** vérifie si un envahisseur touche le joueur. Si c'est le cas, la liste des scores est actualisée et triée et la méthode retourne **`true`**. Sinon la méthode retourne **`false`**. 3p
- La méthode **`getScoreText`** retourne le score précédé du texte « Score: ». Si le score actuel est supérieur à la moyenne des scores existants, elle retourne le score précédé du texte « ! Score: ». 1,5p
- La méthode **`newMissile`** vérifie si un joueur existe et s'il n'y a pas de missile en cours de route. Si tel est le cas elle crée un nouveau missile situé au-dessus du joueur (voir image ci-contre). 1p
- La méthode **`setPlayerXCenter`** place le joueur horizontalement à la position passée en paramètre. 0,5p
- La méthode **`draw`** dessine le joueur, le missile et les envahisseurs s'ils existent. 1,5p



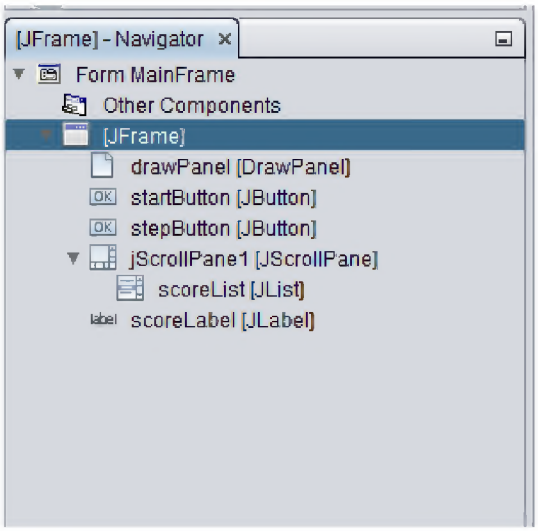
La classe `DrawPanel` — 2 points

- Implémentez la classe **`DrawPanel`** – avec son attribut et manipulateur requis – en vous basant sur le diagramme UML donné. 0,5p
- La méthode **`paintComponent`** dessine un rectangle blanc sur toute la surface du canevas et, si possible, le jeu. 1,5p

La classe `MainFrame` — 11 points

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

- Implémentez l'interface graphique (voir le graphique ci-dessous). L'attribut **`timer`** représente le chronomètre qui a une périodicité de 50 millisecondes et est lié au bouton caché **`stepButton`**. L'attribut **`game`** représente le jeu. Le titre de l'application est « 1GIGInvaders ». 2,5p



- Complétez le constructeur qui crée un nouveau chronomètre et un nouveau jeu. 2p
- La méthode **`startButtonActionPerformed`** désactive le bouton de démarrage, fait une mise à zéro du jeu et lance le chronomètre. 1,5p
- La méthode **`stepButtonActionPerformed`** exécute une étape du jeu. Elle affiche le score actuel et vérifie si le jeu est terminé. Si c'est le cas elle arrête le chronomètre, actualise l'affichage des scores et rend le bouton de démarrage disponible. 3,5p
- La méthode **`drawPanelMouseMoved`** bouge le joueur si un jeu est en cours. La position horizontale de la souris correspond au milieu du joueur. 1p
- La méthode **`drawPanelMousePressed`** crée un nouveau missile. 0,5p

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()	