Programmation Objet - Projet

1 Présentation générale

Pour ce projet de Programmation Objet, nous vous proposons l'implémentation d'un "Tower Defense". Un Tower Defense est un type de jeu consistant à protéger une base en affrontant une succession de vagues d'ennemis.

Pour les affronter, le joueur place des "Tours" alliées qui vont venir éliminer les ennemis. Si le joueur réussit à éliminer toutes les vagues d'ennemis, il gagne. Si les ennemis réussissent à prendre sa base, il perd.

Vous trouverez ci-dessous deux exemples de Tower Defense.

FIGURE 1 – Random Dice PVP TD

FIGURE 2 – Plants VS Zombies

2 Vue générale du projet

Ce jeu, en temps réel, utilisera une interface graphique simple (*cf.* Figure 3) avec laquelle le Joueur pourra interagir grâce à sa souris.

Pour réaliser ce projet, vous devrez donc réaliser différentes parties :
- La gestion de la Carte,
- La gestion du Joueur,
- L'achat de Tours,
- La gestion des Ennemis,
- Les combats entre les différentes entités (Ennemis et Tours),
- L'enchaînement des différentes vagues et niveaux.

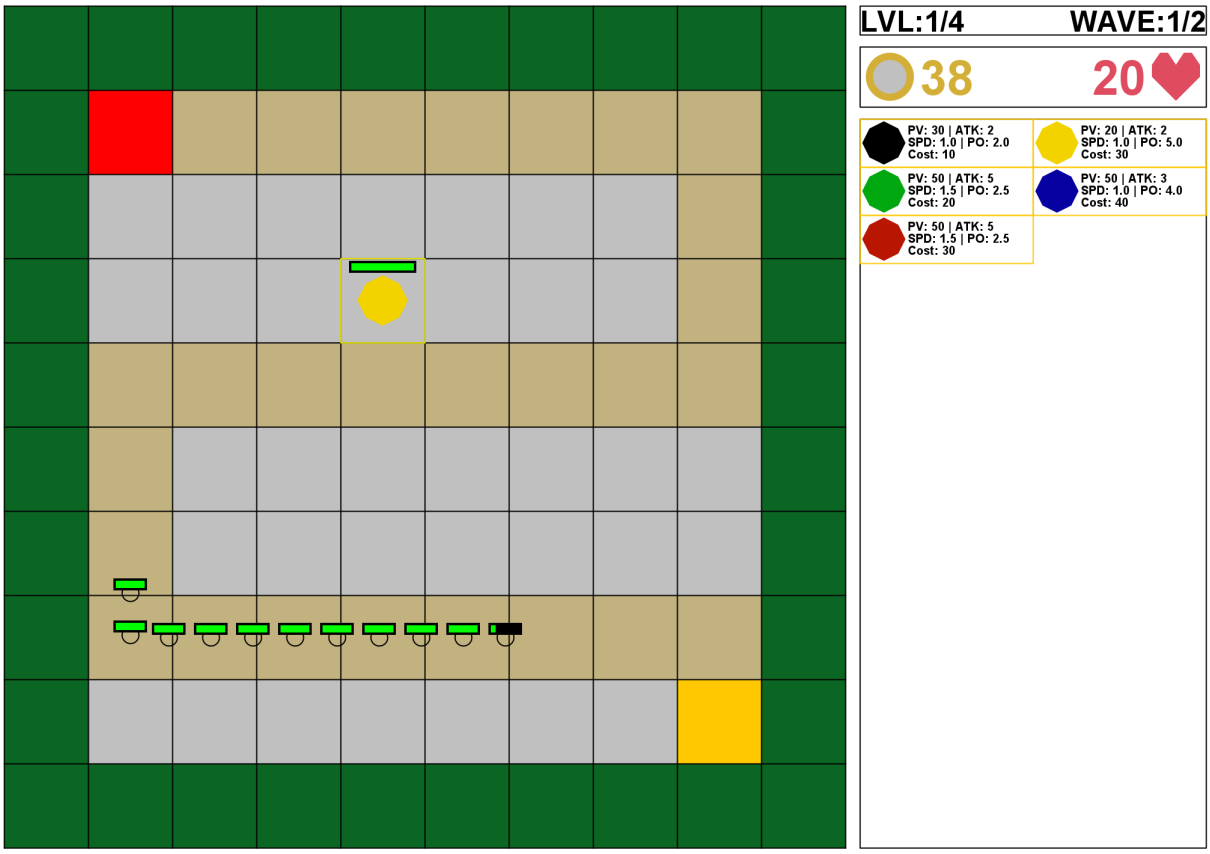

FIGURE 3 – Vue générale du jeu

2.1 Interface Graphique

L'**Interface Graphique** se décompose en 4 zones (*cf.* Figure 4) :

- Dans la zone Violette, la carte.
  - Un quadrillage permet de délimiter visuellement les cases.
  - Chaque type de case possède sa propre couleur.
  - Les différentes entités (Tours et Ennemis) apparaissent à leurs positions sur la carte accompagné d'une barre de points de vie.
- Dans la zone Rouge, les informations relatives à la progression dans le jeu :
  - Le niveau et la vague que le joueur est en train de jouer.
  - Le nombre total de niveau à jouer.
  - Le nombre de vague à jouer dans le niveau.
- Dans la zone Verte, les informations relatives au Joueur :
  - Son nombre de pièces pour acheter de nouvelles Tours.
  - Ses points de vie restants.
- Dans la zone Bleue, le magasin.
  - Chaque Tour disponible est affichée avec ses différentes caractéristiques.

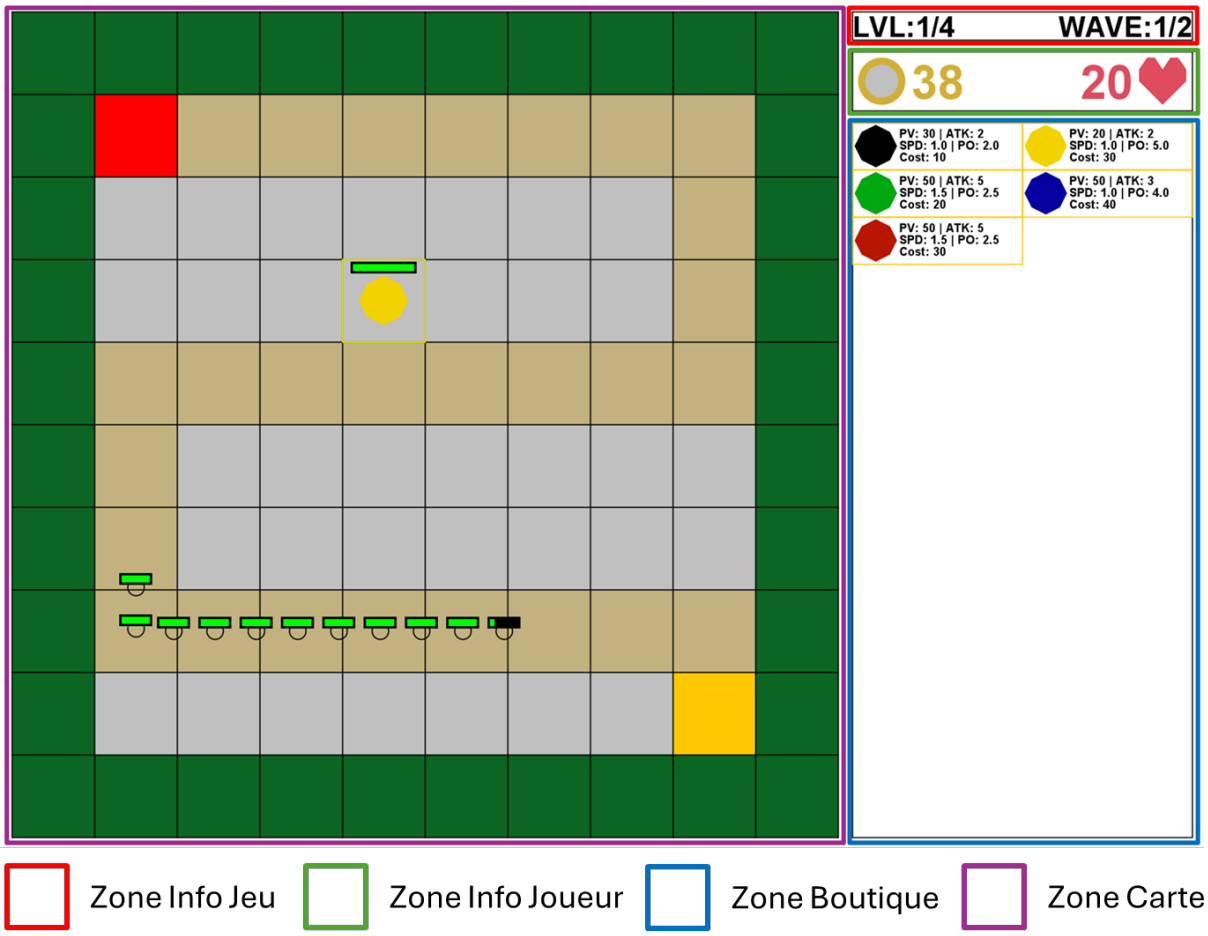

FIGURE 4 – Zones du jeu

2.2 Structure de votre projet

Vous utiliserez Visual Studio Code.
Pour créer un nouveau projet Java, allez dans `File` > `New Files ...` > `New Java Project` > `No build Tools`.

A la racine de votre projet, à côté de `src`, vous placerez le dossier `resources` fourni sur Moodle dans l'activité lié au sujet du projet (lien). Ce dossier est structuré comme suit :

- `resources` : dossier contenant l'ensemble des ressources :
  - `games` : dossier contenant l'ensemble des parties possibles,
  - `levels` : dossier contenant l'ensemble des niveaux possibles,
  - `maps` : dossier contenant l'ensemble des cartes disponibles,
  - `waves` : dossier contenant l'ensemble des vagues disponibles,
  - `saved` : dossier contenant les parties enregistrées si vous effectuez le défi,
  - `graphisms` : dossier contenant les éventuelles images que vous utilisez si vous souhaitez enrichir l'affichage proposé.

Les contenus des fichiers seront détaillés plus tard dans les parties adéquates.

3 Structure d'une partie

Votre jeu consiste à affronter des vagues successives d'ennemis. Le joueur les affronte les unes après les autres s'il a triomphé de la précédente.

Les vagues d'ennemis (`Wave`) seront réparties en niveaux (`Level`). Par exemple, le joueur pourra affronter 3 niveaux, contenant respectivement 2, 3 et 5 vagues.

Au cours d'une vague, les `Tours` du joueur et les `Ennemis` vont s'affronter selon des schémas d'attaque qui seront propres à chacun.

Le `Joueur` peut, à n'importe quel moment du jeu, décider de construire une nouvelle `Tour` sur un emplacement disponible.

Votre jeu utilisera la structure définie ci-après.

3.1 Map (/resources/maps)

Une carte décrit le terrain avec lequel le joueur va pouvoir intéragir et où les Ennemis et Tours vont combattre.

Pour gérer une carte, vous chargerez un fichier de jeu (extension .mtp) dont le contenu décrira la matrice des cases. Par exemple, pour 10-10.mtp (dont vous retrouvez l'affichage dans la figure 3) :

```
XXXXXXXXXX
XSRRRRRRRX
XCCCCCCCRX
XCCCCCCCRX
XRRRRRRRRX
XRCCCCCCCX
XRCCCCCCCX
XRRRRRRRRX
XCCCCCCCBX
XXXXXXXXXX
```

Chaque ligne décrit une ligne de la carte de haut en bas, et chaque colonne une colonne de gauche à droite.

Chaque lettre correspond à une case avec la signification suivante :

- S : case de spawn de des ennemis
  - Couleur utilisée : Color.RED
- B : case de base du joueur
  - Couleur utilisée : Color.ORANGE
- R : case de route sur laquelle les ennemis peuvent marcher.
  - Couleur utilisée : Color(194, 178, 128)
- C : case constructible où le joueur peut placer une tour.
  - Couleur utilisée : Color.LIGHT_GRAY
- X : case non constructible de décor.
  - Couleur utilisée : Color (11, 102, 35)

Les ennemis devront progresser sur cette carte le long de la route qui est définie. Vous allez donc devoir la modéliser pour savoir comment actualiser la position des Ennemis.

Pour cela, nous vous proposons d'utiliser dans un premier temps un algorithme simple pour trouver ce chemin. Attention cependant à ne pas utiliser une carte mal formée, car il ne gère pas les cas d'erreur. Enregistrez chaque étape du Spawn à la Base pour connaître le chemin complet.

- Trouvez le Spawn Ennemi
- Pour chaque case autour, vérifiez si c'est une Route.
- Déplacez-vous sur la case Route suivante.
- Vérifiez à nouveau les Routes autour, en excluant la case de laquelle vous venez.
- Arrêtez-vous quand vous tombez sur une case Base.

3.2 Game (/resources/games/)

Votre jeu permettra au joueur de compléter une suite de niveaux (**un nombre ordonné de Level**).

Pour cela, vous chargerez un fichier de jeu (extension .g) dont le contenu sera la liste des niveaux à charger, qu'il faudra aller chercher dans le dossier associé. Par exemple game.g :

```
level1
level2
level3
level4
```

Cela signifie que vous devrez charger les niveaux situés dans /resources/levels : level1.lvl, level2.lvl, etc.
Entre chaque niveau **les ressources et Tours du joueur sont réinitialisées**.
Lorsqu'un niveau est gagné par le joueur, le jeu passe automatiquement au niveau suivant.
La carte peut donc changer entre les niveaux !

3.3 Level (/resources/levels)

Un niveau consiste à affronter des vagues successives d'ennemis (**un nombre ordonné de Wave**)
Pour cela, vous chargerez un fichier de niveau (extension .lvl) dont le contenu décrira la carte à utiliser (première ligne du fichier), ainsi que la liste des vagues à charger, qu'il faudra aller chercher dans le dossier associé. Par exemple level1.lvl :

```
10-10
wave1
wave2
```

Cela signifie qu'il faudra charger la carte /resources/maps/10-10.mtp avec les vagues de wave1.wve et wave2.wve situées dans /resources/waves.
Ces affrontements se font sur une même carte, le joueur conserve d'une vague à l'autre ses Ressources et ses Tours.
Lorsqu'une vague est terminée, le jeu passe automatiquement à la vague suivante.

3.4 Wave (/resources/waves)

Une vague est une définie par une liste de paires Temps / Ennemi, spécifiant à quel moment après le début de la vague un ennemi commence à parcourir le chemin jusqu'à la base du Joueur.
Pour cela, vous chargerez un fichier de vague (extension .wve) dont le contenu décrira la liste des paires Temps/Ennemis à charger. Par exemple wave1.wve :

```
5.0|Basic
5.5|Basic
6.0|Basic
6.5|Basic
7.0|Basic
```

5.0|Basic signifie alors "Apparition d'un ennemi 'Basic' à 5.0 secondes après le début de la vague".
Les ennemis apparaissent toujours au début du chemin (Spawn) et se dirigent vers la fin (Base du Joueur).

3.5 Conditions d'arrêt du jeu

Le jeu se termine "normalement" dans 2 cas précis :
— Défaite : Les ennemis ont vaincu le joueur en réduisant ses points de vie à 0.
— Victoire : Le joueur a triomphé de tous les niveaux et toutes les vagues d'ennemis.
Dans tous les autres cas, l'arrêt du programme constitue une erreur qu'il faut gérer et/ou signifier au Joueur (par exemple en texte dans la console).

4 Interface Utilisateur

Sur l'affichage graphique de votre projet (Figure 4), le joueur peut interagir avec deux zones, en cliquant grâce à la souris :

— La zone Violette : la carte, sur laquelle il peut sélectionner une case (**Tile**) si celle-ci est constructible et libre.
— La zone Bleue : le magasin, dans lequel il peut sélectionner la Tour à construire en cliquant.
  — S'il a au préalable sélectionné une case constructible et libre (ie. sans Tour déjà construite),
  — Et si son budget le lui permet,
  — Elle sera ajoutée sur la carte et son coût déduit de la bourse du Joueur.

4.1 Mécanique de la souris

Dans ce jeu, plusieurs éléments sont interactifs avec la souris. Vous avez le choix entre :

— Définir des éléments "cliquables", auxquels cas vous pourrez parcourir l'ensemble de vos éléments "cliquables" pour savoir si le Joueur a interagi avec l'un d'eux et lequel.
— Regarder où le Joueur a utilisé la souris : si le clic est utilisé dans une zone d'interaction (Map ou Store), vous pouvez alors convertir les coordonnées de la souris pour extraire l'élément associé.

5 Les combattants

Deux types d'unités vont s'affronter :

— Les Tours, fixes sur le terrain et gérées par le joueur.
— Les Ennemis, se déplaçant le long du chemin.

Ces unités sont caractérisées par :

— Un nombre de point de vie (**PV**). Tant que ce nombre est supérieur à 0, l'unité est encore en vie et agit.
— Une valeur d'attaque (**ATK**). Quand l'unité attaque, c'est la valeur de base utilisée pour infliger des dégâts.
— Une vitesse d'attaque (**ATKSpeed**). Exprimé en seconde, c'est l'intervalle de temps entre deux attaques de l'unité.
— Une portée (**Range**). Exprimé en nombre de case (de la carte), c'est la distance euclidienne à laquelle l'unité peut lancer une attaque. Les unités situées au-delà ne peuvent pas être visées.
— Un élément (**Element**). Pouvant avoir une des valeurs suivantes : "Neutre", "Feu", "Terre", "Air", "Eau". La Couleur utilisée pour afficher l'unité dépend de cet élément. Pour information, voici les codes couleurs utilisés dans les images, même si vous pouvez les changer :
  — Neutre : Color.BLACK
  — Feu : Color(184, 22, 1)
  — Terre : Color(0, 167, 15)
  — Air : Color(242, 211, 0)
  — Eau : Color(6, 0, 160)
— Une position (x,y). Elle permet de localiser les unités les unes par rapport aux autres, notamment pour le calcul de portée.
— Un mode d'attaque. C'est la manière dont l'unité va attaquer : une seule cible, plusieurs, aléatoirement, la plus proche, la plus forte (ATK), la plus forte (%PV), etc.

5.1 Calcul des dégâts

Les unités ne font des dégâts qu'aux unités opposées (ie. Les Tours attaquent uniquement les Ennemis et les Ennemis uniquement les Tours, sauf mention contraire.)

Un facteur de vulnérabilité (**Vulnérabilité**) est défini selon un cycle élémentaire (Feu < Eau < Air < Terre < Feu < ...) :

— 1.5 si l'unité est vulnérable à l'élément.
— 0.5 si l'unité est résistante à l'élément.

— 1 sinon.

Le nombre de points de vie perdu par la cible est calculé par : **ATK** x **Vulnérabilité**

5.2 Les Tours

Chaque type de Tour possède un coût (**Cost**) qui lui est propre et que le joueur doit payer pour pouvoir la construire.

5.2.1 Liste des Tours

Dans la version basique du jeu, vous devrez implémenter 5 Tours différentes, dont les caractéristiques sont résumés dans le Tableau 1.

Les modes d'attaques sont décrits individuellement par la suite.

Name	PV	ATK	ATKSpeed	Range	Element	Cost
Archer	30	5	1	2	NONE	20
Wind Caster	30	5	1.5	6	WIND	50
Water Caster	30	3	1	4	WATER	50
Earth Caster	50	7	0.5	2.5	EARTH	100
Fire Caster	30	10	0.5	2.5	FIRE	100

TABLE 1 – Statistiques des Tours

Archer Il vise l'ennemi qui se trouve à sa portée le plus avancé sur le chemin de la base.

Wind Caster Il vise l'ennemi qui se trouve à sa portée le plus proche de sa position.

Water Caster Il vise l'ennemi qui se trouve à sa portée le plus avancé sur le chemin de la base.

Earth Caster Il vise l'ennemi qui se trouve à sa portée ayant le plus grand nombre de PV. Les ennemis à moins de 1.0 case de la cible subissent les mêmes dommages que sa cible.

Fire Caster Il vise l'ennemi qui se trouve à sa portée le plus proche de sa position. Les ennemis à une distance (euclidienne entre les positions des deux éléments) inférieure à 0.75 case de la cible subissent les mêmes dommages que sa cible.

5.3 Les Ennemis

Un ennemi se déplace le long du chemin jusqu'à la base du joueur à une vitesse (**Speed**) qui lui est propre. Lorsque ses points de vie tombent à 0, le joueur engrange une prime (**Reward**) pour son élimination et il disparaît.

5.3.1 Progression

Le chemin parcouru par les ennemis est décrit dans la Figure 5 en bleu. Les ennemis apparaissent au **centre de leur Spawn**, et progressent jusqu'au **centre de la Base** du Joueur. Ils progressent toujours le long **des médiatrice de la case** (du milieu du segment d'entrée vers le centre de la case, puis du centre de la case vers le milieu du segment de sortie). Ainsi, chaque case prend le même temps pour être parcouru.

Afin de réaliser cette partie, il existe de nombreuses possibilités. Une manière de faire peut être d'utiliser le calcul du chemin réalisé à l'import de la carte. Plutôt que de traiter une position 2D (x,y), l'ennemi peut progresser sur une seule dimension (la distance parcourue). Le chemin que vous avez calculé peut alors faire la conversion de cette distance vers une position 2D.

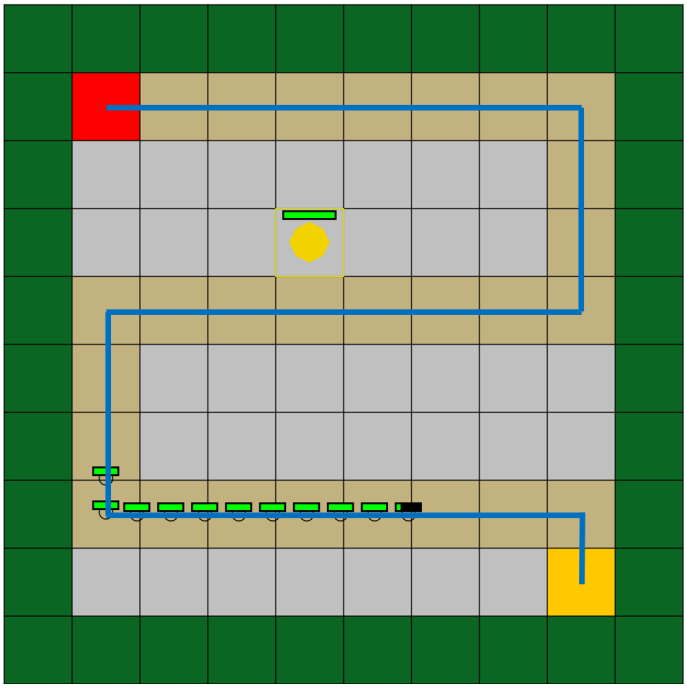

FIGURE 5 – Chemin des ennemis

5.3.2 Liste des Ennemis

Dans la version basique du jeu, vous devrez implémenter 6 Ennemis différents, dont les caractéristiques sont résumés dans le Tableau 2.

L'attribut **"Name"** correspond au nom qui sera utilisé dans les fichiers de Vague.

Les modes d'attaques sont décrits individuellement par la suite.

Name	PV	ATK	ATKSpeed	Range	Element	Speed	Reward
Minion	10	3	0	0	NONE	1	1
Wind Grognard	1	7	2	5	WIND	2	1
Fire Grognard	1	7	2	3	FIRE	2	1
Water Brute	30	5	1	3	WATER	1	3
Earth Brute	30	5	1	3	EARTH	1	3
Boss	150	100	10.0	2.0	FIRE	0.5	100

TABLE 2 – Statistiques des Ennemis

Minion Il ne peut pas attaquer.

Wind Grognard Il vise la Tour qui se trouve à sa portée ayant le moins de PV.

Fire Grognard Il vise la Tour qui se trouve à sa portée la plus proche de sa position. Les Tours à une distance (euclidienne entre les positions des deux éléments) inférieure à 1.5 case de la cible subissent les mêmes dommages que sa cible.

Water Brute Il vise la Tour qui se trouve à sa portée ayant le moins de PV. Les Tours à une distance (euclidienne entre les positions des deux éléments) inférieure à 1.5 case de la cible subissent les mêmes dommages que sa cible.

Earth Brute Il vise la Tour qui se trouve à sa portée le plus proche de sa position.

Boss Il vise la Tour qui se trouve à sa portée le plus proche de sa position.

6 Le Joueur

Le Joueur (**Player**) possède des points de vie (**PdV**).
— Il commence avec 100 point de vie.
— Sa vie diminue de la valeur d'attaque du monstre chaque fois que l'un d'eux atteint sa base.
— Une fois la vie du joueur à 0, il perd la partie.
Le joueur amasse de l'argent (**Money**).
— Il commence avec 50 pièces d'or.
— Chaque fois qu'il tue un ennemi, il gagne le montant de sa prime en pièces d'or.
— Chaque fois qu'il achète une tour, il perd son coût en pièces d'or.
— Il ne peut pas avoir un nombre de pièces d'or négatif
Le Joueur peut sélectionner une case constructible sur la carte. Si une case constructible est sélectionnée, il peut choisir une tour à y construire en échange de son coût.

7 Bonus

Pour aller plus loin, une fois la base de votre jeu réalisée (*cf.* Partie 2) et validée par l'enseignant, vous pouvez enrichir votre jeu avec les éléments suivants.

7.1 Nouvelles Unités

Pour augmenter la diversité des comportements, vous devrez implémenter de nouvelles unités avec de nouvelles mécaniques .

7.1.1 Nouvelles Tours

Name	PV	ATK	ATKSpeed	Range	Element	Cost
Ice Caster	40	1	2	5	WATER	70
Poison Caster	50	1	2	5	AIR	80
Gold Digger	20	1	2	10	EARTH	20
Railgun	20	1	-	-	FIRE	150

Table 3 – Statistiques des Tours

Ice Caster Il vise l'ennemi qui se trouve à sa portée le plus avancé sur le chemin de la base. Son attaque touche tous les ennemis à une distance (euclidienne entre les positions des deux éléments) inférieure à 1.0 case autour de sa cible et réduit la vitesse d'attaque et de déplacement des unités ennemies de 30%.

Poison Caster Il vise l'ennemi qui se trouve à sa portée le plus avancé sur le chemin de la base qui ne soit pas déjà empoisonné. Son attaque fait des dommages directs et inflige un poison qui réitère les mêmes dégâts avec la même vitesse d'attaque.

Gold Digger Il vise l'ennemi qui se trouve à sa portée le plus proche. Chaque fois que son attaque touche un ennemi, il génère 1 pièce pour le Joueur.

Railgun Il vise l'ennemi le plus proche de la position de la souris chaque fois que le Joueur clique.

7.1.2 Nouveaux Ennemis

Name	PV	ATK	ATKSpeed	Range	Element	Speed	Reward
Healer	10	1	1	2	NONE	0.5	3
Buffer	10	2	2	3	WIND	0.5	5
Bomb	1	5	2	3	FIRE	2	2
Merchant King	100	0	0	0	EARTH	2	-10
Termiernator	999	999	15	-	NONE	0.5	100

Table 4 – Statistiques des Ennemis

Healer Il vise la Tour qui se trouve à sa portée la plus proche de sa position. Toutes les 5 secondes, il soigne de 5 PV les ennemis à une distance (euclidienne entre les positions des deux éléments) inférieure à 1.5 case de lui.

Buffer Il vise une Tour qui se trouve à sa portée aléatoirement. Il augmente passivement de 50% la vitesse d'attaque et de déplacement des autres ennemis à une distance (euclidienne entre les positions des deux éléments) inférieure à 1.5 case de lui.

Bomb Il vise la Tour qui se trouve à sa portée ayant le moins de PV. Les Tours situées à une distance (euclidienne entre les positions des deux éléments) inférieure à 1.5 case de sa cible subissent les mêmes dommages. À sa mort, explose et inflige 10 fois son attaque aux Tours à une distance (euclidienne entre les positions des deux éléments) inférieure à 1.5 case autour de lui.

Merchant King Il n'attaque pas. Si le joueur le tue, il perdra des pièces. S'il parvient à rejoindre la Base, il proposera une récompense à prix réduit (selon le pourcentage de vie qu'il lui reste). Le joueur pourra alors sélectionner un bonus permanent parmi 4 proposition :

- +10% de puissance d'attaque sur toutes les Tours. Coût 200.
- -10% de vitesse de déplacement chez les ennemis. Coût 300.
- +10% de vitesse d'attaque sur les Tours. Coût 200.
- Aucun bonus. + 30 pièces.

Ils peuvent se cumuler jusqu'à 5 fois et reste malgré les changements de niveaux.

Termiernator Il élimine aléatoirement ~~un étudiant~~ une Tour toutes les 15 secondes. Il affiche également un message dans une info-bulle toutes les 10 secondes pour une durée de deux secondes : "L'examen final se rapproche !".

7.2 Exceptions

Votre jeu contient actuellement de nombreuses erreurs potentielles. Dans cette partie, nous vous demandons de gérer quelques unes d'entre elles.

Pour cela, la bonne méthode est d'utiliser les exceptions. Elles permettent de traiter proprement des problèmes d'intégrité grâce à la propagation d'erreurs (e.g. l'utilisateur est sur le point de mettre deux tours sur la même case).

Vous devrez implémenter et gérer les exceptions suivantes en définissant la hiérarchie des différentes exceptions tout comme le fait qu'elles soient abstraites ou concrètes.

GameException Exception regroupant les caractéristiques de toutes les exceptions du jeu.

InvalidMapPathException Exception symbolisant que le jeu essaye de charger une map dont le chemin est non valide (ie. on ne peut pas aller du Spawn des ennemis à la Base du joueur).
- Affichez une erreur pour informer l'utilisateur :
  - Du fichier niveau dans lequel la carte erronée est utilisée.
  - Du fichier carte dans lequel l'erreur s'est produite.
  - Indiquant que le path n'est pas valide et sa raison.
    - Pas de chemin (fichier Error_Path_Loop_1.mtp fourni)
    - Chemin multiple (fichier Error_Multiple_Path.mtp fourni)
    - Boucle (selon comment vous avez codé). (fichiers Error_Path_Loop_1.mtp et Error_Path_Loop_2.mtp fournis)
- Le jeu s'arrête.

MapException Exception symbolisant que le jeu essaye de charger une map non valide.
- Affichez une erreur pour informer l'utilisateur :
  - Du fichier niveau dans lequel la carte erronée est utilisée.
  - Du fichier carte dans lequel l'erreur s'est produite.
  - Indiquant que la Carte n'est pas valide et sa raison.
    - Tuile inconnue rencontrée (en ajoutant sa ligne et sa colonne) (fichier Error_UnknownTile.mtp fourni)
    - Largeur de la carte non constante (fichier Error_Not_Same_Width.mtp fourni).
    - Hauteur de la carte non constante (fichier Error_Not_Same_Height.mtp fourni).
- Le jeu s'arrête.

MultipleEnemySpawnException Exception symbolisant que le jeu essaye de charger une map qui possède plusieurs spawns d'ennemis.
- Affichez une erreur pour informer l'utilisateur :
  - Du fichier niveau dans lequel la carte erronée est utilisée.
  - Du fichier carte dans lequel l'erreur s'est produite.
  - Et les positions (lignes et colonnes) de chacun des Spawn.
- Le jeu s'arrête.
- Vous pouvez utiliser le fichier Error_Multiple_Spawn.mtp fourni.

MultiplePlayerBaseException Exception symbolisant que le jeu essaye de charger une map qui possède plusieurs bases.
- Affichez une erreur pour informer l'utilisateur :
  - Du fichier niveau dans lequel la carte erronée est utilisée.
  - Du fichier carte dans lequel l'erreur s'est produite.
  - Et les positions (lignes et colonnes) de chacune des Bases.
- Le jeu s'arrête.
- Vous pouvez utiliser le fichier Error_Multiple_Base.mtp fourni.

NoEnemySpawnException Exception symbolisant que le jeu essaye de charger une map dépourvue de spawn ennemi.
- Affichez une erreur pour informer l'utilisateur :
  - Du fichier niveau dans lequel la carte erronée est utilisée.
  - Du fichier carte dans lequel l'erreur s'est produite.
  - Et qu'il manque le Spawn des ennemis.
- Le jeu s'arrête.
- Vous pouvez utiliser le fichier Error_No_Spawn.mtp fourni.

NoPlayerBaseException Exception symbolisant que le jeu essaye de charger une map dépourvue de base.
- Affichez une erreur pour informer l'utilisateur :
  - Du fichier niveau dans lequel la carte erronée est utilisée.
  - Du fichier carte dans lequel l'erreur s'est produite.
  - Et qu'il manque la Base du joueur.
- Le jeu s'arrête.
- Vous pouvez utiliser le fichier Error_No_Base.mtp fourni.

NotEnoughMoneyException Exception qui survient lorsque le joueur tente d'effectuer une action nécessitant plus d'argent qu'il n'en a.
- Affichez un message pour informer l'utilisateur du type : "Not enough money to build Tower ! Please, kill ennemies to gain wealth or be cheap and buy something you can afford."
- Le jeu continue.

TileOccupiedException Exception symbolisant que le joueur tente de placer une tour sur une case déjà occupée.
- Affichez un message pour informer l'utilisateur du type : "Map tile already built ! Cannot place new Tower !"
- Le jeu continue.

UnknownEnemyException Exception symbolisant que la vague tente de placer un ennemi dont le type n'est pas reconnu par le jeu.
- Affichez une erreur pour informer l'utilisateur :
  - Du fichier niveau dans lequel la vague erronée est utilisée.
  - Du fichier vague dans lequel l'erreur s'est produite.
  - De la ligne associée : numéro de ligne + contenu de la ligne.
- Le jeu s'arrête.
- Vous pouvez utiliser le fichier Error_UnknownEnnemy.wve fourni.

7.3 Sauvegarde

En cas de défaite, on voudrait que le joueur puisse reprendre à l'endroit où il a perdu.
Vous implémenterez une sauvegarde automatique de l'état du jeu, qui prendra en compte :
- Le fichier jeu que vous le joueur est en train de jouer (lu du dossier "/resources/games")
- Le niveau auquel le joueur se trouve.
- La vague à laquelle il en est.

Pour cela, vous devrez **impérativement** :
- Créer le dossier "/resources/saved"
  - Y créer un fichier portant le même nom que le fichier jeu d'origine.
- Utiliser l'interface **Serializable** pour sauvegarder les données nécessaires.
  - Faire cette action à chaque changement de vague.
- Au lancement du programme, si un fichier porte le même nom que le jeu lancé dans le dossier "saved", vous importerez les paramètres.

8 Ressources fournies

Dans cette partie, nous vous donnons des indications, du code ou des explications sur les ressources mises à disposition afin de réaliser le projet dans de bonnes conditions.

8.1 Dossier “resources”

A la racine de votre projet, vous avez placé le dossier “resources” fourni. Son contenu est le suivant :

- resources (dossier contenant l'ensemble des ressources)
  - games (dossier contenant l'ensemble des parties possibles)
  - levels (dossier contenant l'ensemble des niveaux possibles)
  - maps (dossier contenant l'ensemble des cartes disponibles)
    - 5-8.mtp : Une map correcte de 5 par 8.
    - 10-3.mtp : Une map correcte de 10 par 3. Attention, il n'y a pas de bordure !
    - 10-10.mtp : Une map correcte de 10 par 10.
    - Error_UnknownTile.mtp : Une map contenant une tuile inconnue (caractère ‘A’)
    - Error_No_Base.mtp : Une map où il manque la base (caractère ‘B’)
    - Error_No_Spawn.mtp : Une map où il manque le Spawn (caractère ‘S’)
    - Error_Multiple_Base.mtp : Une map où il y a trop de bases (caractère ‘B’)
    - Error_Multiple_Spawn.mtp : Une map où il y a trop de Spawn (caractère ‘S’)
    - Error_No_Path.mtp : Une carte où le chemin est coupé.
    - Error_Multiple_Path.mtp : Une carte où plusieurs chemins mènent à la Base.
    - Error_Path_Loop_1.mtp : Une carte où la route boucle sur le Spawn et la Base (2 loops distinctes)
    - Error_Path_Loop_2.mtp : Une carte où le chemin boucle sur lui-même.
    - Error_Not_Same_Width.mtp : Une carte qui n'a pas une largeur fixe.
    - Error_Not_Same_Height.mtp : Une carte qui n'a pas une hauteur fixe.
  - waves (dossier contenant l'ensemble des vagues disponibles)
    - Error_UnknownEnnemy.wve : Vague faisant référence à un ennemi ‘Unknown’ n'existant pas.

8.2 StdDraw

La library StdDraw vous permettra de gérer un affichage et l'interface graphique associée. Placez simplement le fichier StdDraw.java dans le dossier src de votre projet.

Vous y trouverez notamment :

- Gestion du canvas
  - setCanvasSize(int width, int height)
  - clear()
- Une gestion de la souris :
  - isMousePressed()
  - mouseX()
  - mouseY()
- Afficher du texte :
  - text(double x, double y, String text)
- Différentes formes géométriques :
  - point(double x, double y)
  - line(double x1, double y1, double x2, double y2)
  - circle(double x, double y, double radius)
  - ellipse(double x, double y, double semiMajorAxis, double semiMinorAxis)
  - square(double x, double y, double radius)
  - rectangle(double x, double y, double halfWidth, double halfHeight)

8.3 Boucle de jeu

Une boucle de jeu (gameloop) vous permettra de gérer les éléments temporels de votre jeu.

Afin de bien démarrer le projet, nous vous fournissons un squelette pour l'application principale (**App**), qui vous permettra de lancer le jeu (**Game**).

Le code de ces classes est susceptible de changer légèrement selon les défis que vous réaliserez.

```
public class App
{
    public static void main(String[] args)
    {
        Game g = new Game();
        g.launch();
    }
}
```

```
public class Game
{
    ...

    public void launch()
    {
        init();
        long  previousTime = System.currentTimeMillis();
        while(isGameRunning())
        {
            long  currentTime = System.currentTimeMillis();
            double deltaTimeSec = (double)(currentTime - previousTime)/1000;
            previousTime = currentTime;

            update(deltaTimeSec);
        }
    }

    private boolean isGameRunning()
    {
        ...
    }

    private void init()
    {
        ...
    }

    private void update(double deltaTimeSec)
    {
        ...
    }
}
```

La fonction **launch** démarre le jeu et boucle tant qu'il est en cours d'exécution. A vous d'implémenter correctement les fonctions :

- **isGameRunning** pour arrêter le jeu quand une des conditions d'arrêt est rencontrée.
- **init** pour intialiser le jeu en fonction du fichier chargé.
- **update** pour mettre à jour tous **les objets** et **affichages** du jeu selon leur propre temporalité.
  - Pour cela, nous vous conseillons de définir une fonction *public void update (double deltaTime)*, pour tous les éléments du jeu nécessitant d'être actualisés.
  - Le "deltaTime" (exprimé en seconde ici) est utilisé pour gèrer la temporalité des actions : vous savez combien de temps s'est écoulé depuis la dernière boucle de jeu, vous pouvez donc actualiser les éléments en fonction de ce temps. Par exemple :
    - Déplacer les ennemis selon leurs vitesses
    - Attaquer à un rythme défini

8.4 Draw

L'affichage de votre interface graphique peut très rapidement virer au casse-tête. Pour cette raison, nous vous proposons quelques éléments à reprendre afin de partir sur de bonnes bases.

Pour la définition de la taille de la fenêtre, nous vous proposons une taille fixe de 1024 par 720, dont les coordonnées iront sur l'axe X (largeur) de -12 à 1012 et sur l'axe Y (hauteur) de -10 à 710.

```
StdDraw.setCanvasSize(1024, 720);
StdDraw.setXscale(-12, 1012);
StdDraw.setYscale(-10, 710);
StdDraw.enableDoubleBuffering();
```

Vous pourrez délimiter les zones avec le code suivant et les valeurs données dans les sections associées :

```
StdDraw.setPenColor(Color.BLACK);
StdDraw.rectangle(center.getX(), center.getY(), halfDist.getX(), halfDist.getY());
```

8.4.1 Zone Map

Pour la zone de la carte, elle est définie par :
— center : (350, 350)
— halfDist : (350, 350)

Pour les cases, nous souhaitons les afficher sous forme de carrés. Pour cela vous utiliserez le code suivant :

```
StdDraw.setPenColor(color);
StdDraw.filledSquare(centerX, centerY, halfLength);
```

Avec les paramètres suivant :
— color : la couleur de la case que vous affichez.
— (centerX, centerY) : les coordonnées du centre de la case.
— halfLength : la largeur du carré.

Afin de maximiser la taille de la carte, vous devrez regarder si elle est plus large que haute afin de déterminer la plus grande "halfLength" qui rentrera dans la zone de la carte. Ainsi, peu importe la taille de la carte, elle pourra toujours être affichée.

Par exemple, si vous avez une map de 10 par 10, chaque case aura une largeur de 70. Idem si vous avez des cartes de 5 par 10 ou 10 par 5. Mais une carte de 5 par 5 aura une largeur de 140.

Pour tracer la grille permettant de visualiser les cases correctement, vous pourrez utiliser le code suivant après avoir déterminé les coordonnées (x0,y0) et (x1,y1) des extrémités du segment :

```
StdDraw.setPenColor(Color.BLACK);
StdDraw.line(x0,y0,x1,y1)
```

8.4.2 Zone Level

Pour la zone du niveau, elle est définie par :
— center : (856, 688)
— halfDist : (144, 12)

8.4.3 Zone Player

Pour la zone du Joueur, elle est définie par :
— center : (856, 641)
— halfDist : (144,25)

Pour tracer une pièce comme dans l'affichage proposé, vous pourrez utiliser le code suivant après avoir déterminer son centre et son rayon :

```
    StdDraw.setPenColor(new Color(212, 175,55));
    StdDraw.filledCircle(center.getX(), center.getY(), radius);
    StdDraw.setPenColor(new Color(192, 192,192));
    StdDraw.filledCircle(center.getX(), center.getY(), 0.7 * radius);
```

Pour tracer un coeur, il vous faudra également son centre et sa demi-hauteur :

```
    // Draw a heart
    StdDraw.setPenColor(new Color(223, 75, 95));

    double[] listX = new double[]
        {
            center.getX(),
            center.getX() - halfHeight,
            center.getX() - halfHeight,
            center.getX() - 0.66 * halfHeight,
            center.getX() - 0.33 * halfHeight,
            center.getX(),
            center.getX() + 0.33 * halfHeight,
            center.getX() + 0.66 * halfHeight,
            center.getX() + halfHeight,
            center.getX() + halfHeight,
        };
    double[] listY = new double[]
        {
            center.getY() - halfHeight,
            center.getY(),
            center.getY() + 0.5 * halfHeight,
            center.getY() + halfHeight,
            center.getY() + halfHeight,
            center.getY() + 0.5 * halfHeight,
            center.getY() + halfHeight,
            center.getY() + halfHeight,
            center.getY() + 0.5 * halfHeight,
            center.getY(),
        };
    StdDraw.filledPolygon(listX, listY);
```

8.4.4 Zone Store

Pour la zone du Magasin, elle est définie par :
— center : (856, 303)
— halfDist : (144, 303)

9 Réalisation du projet

Pour ce projet, vous travaillerez en **binôme**.

Nous vous conseillons de réfléchir à votre hiérarchie de classe et la faire examiner régulièrement par votre encadrant.

Nous vous proposons de découper le projet en plusieurs étapes et de ne progresser à l'étape suivante qu'une fois la précédente réalisée.

Vous restez toutefois maître quant à votre manière de procéder et pouvez opter pour une autre méthode.

9.1 Etape 1 : La Carte

La Carte est au centre de votre jeu. L'implémenter dès le début vous permet de comprendre et débuguer votre jeu en temps voulu.

— Commencez par tracer les 4 zones de votre jeu comme décrite précédemment.
— Ouvrez un simple fichier dans resources/maps pour lire et afficher sa carte.

— Créez la structure de votre code pour les différentes parties de la carte.
  — La carte en elle-même
  — Chaque case indépendante
— Affichez vos cases et votre grille
— Interagissez avec la souris (changez l'affichage de la case sélectionnée, en lui affichant une bordure ou modifiant sa couleur).
— Enfin, vous pouvez également calculer le chemin du Spawn ennemi jusqu'à votre Base.

9.2 Etape 2 : Les ennemis

Etant donné que vous venez de commencer à lire les fichiers, autant continuer.
— Comme précédemment, lisez un fichier dans resources/waves.
— Créez votre structure de vague et d'ennemis.
— Initialisez votre ensemble d'ennemis à générer à partir de celle-ci.
— Faites votre boucle de jeu pour les faire apparaître au moment opportun.
— Faites avancer les ennemis sur le chemin que vous avez calculez.

9.3 Etape 3 : Le Joueur

Maintenant que vous avez des ennemis, votre joueur peut apparaître
— Créez votre joueur.
— Affichez sa vie et son argent dans l'interface dédiée.
— Faites lui perdre de la vie quand les ennemis arrive à sa base (vous ne pouvez pas encore les combattre à ce stade)
— Codez une première Tour simple.
— Quand le joueur clique sur un case disponible, construisez la Tour simple que vous avez codé (si son budget le lui permet).

9.4 Etape 4 : Les combats

Maintenant que vous avez des ennemis et pouvez placer au moins une Tour.
— Actualisez les entités du Jeu avec le deltaTime pour savoir s'ils doivent attaquer.
— Affichez les points de vie des Tours et des Ennemis.
— Pour chaque Tour, faites les attaquer les Ennemis.
— Pour chaque Ennemi, faites les attaquer les Tours.
— Supprimez les éléments qui ont été vaincus.
— Faites gagner de l'argent au joueur le cas échéant.

9.5 Etape 5 : La boutique

Diversifions les différentes Tours que le joueur peut placer.
— Implémentez toutes les Tours de la structure
— Affichez-les dans la zone de la boutique.
— Détectez quand le Joueur clique sur une Tour de la zone de la boutique.
— Générez-la sur la carte si le Joueur dispose du montant suffisant.

9.6 Etape 6 : Diversité des Ennemis

De même que pour les Tours, vous pouvez terminer la diversité de vos Ennemis. Utilisez d'autres fichiers de resources/waves pour les tester. Créez-en si nécessaire.
— Implémentez toutes les Ennemis de la structure

— Testez-les avec un fichier Wave que vous définirez.

9.7 Etape 7 : Chargez le jeu entier

Maintenant que vous tous les ennemis avec une vague, chargez le jeu complet.
— Commencez par lire un fichier de /resources/levels.
— Créez la structures pour que votre jeu contienne votre niveau, cartes et vagues.
— Enchaînez les vagues si le joueur gagne.
— Gérez les conditions de victoire/défaite par le Joueur.
— Lisez un fichier de /resources/games.
— Initialisez les niveaux avec leurs vagues et cartes.
— Enchaînez les niveaux.
— Affichez l'état des niveaux / vagues dans votre interface graphique.

9.8 Etape 8 : Les Bonus

Si vous en arrivez là, vous pouvez implémenter des fonctionnalités supplémentaires proposées (Section Bonus) ou de votre propre cru pour rendre le jeu plus intéressant (équilibrage, graphisme, animation d'attaque et de dégâts reçus, plus de cases différentes avec des effets associés, etc.)

10 Livrables

Référez vous à Moodle pour la date de rendu. Attention de bien enregistrer votre binôme dès que possible, il ne sera pas possible de rendre votre projet si vous n'êtes pas dans un binôme enregistré.

Le projet sera à déposer sur moodle. Il faudra rendre une archive compressée (zip), contenant :
— Un **zip** de projet Visual Studio Code contenant l'essentiel pour exécuter votre projet :
  — Vos **sources**
  — Les éventuelles **ressources** complémentaires (images).
  — **Attention :**
    — Le projet rendu doit **obligatoirement** être un projet Visual Studio Code, nous n'acceptons pas de projets d'autres éditeurs (note de 0 au projet si cette consigne n'est pas respectée)
    — Le projet ne doit s'appuyer sur aucune librairie externe autre que StdDraw
    — Vérifiez que le projet que vous rendez compile et s'exécute avant de le rendre !
— Un fichier **readme.txt** de présentation votre projet :
  — **Noms** des étudiants.
  — Explications des **fonctionnalités réalisées**.
  — Eventuelles **fonctionnalités supplémentaires**.
  — **Guide** pour exécuter votre projet.
  — Description de l'interface pour jouer avec votre projet.
— Un **diagramme de classe** de votre projet **à jour** sous forme d'image (format .png par exemple).
— La **Javadoc** générée.
  — Pour générer la documentation dans un dossier "docs" à côté de votre dossier "src" :
    — Ouvrez un terminal et placez-vous dans votre dossier "src"
    — Utilisez la commande : javadoc -d ../docs ./*
    — javadoc lira automatiquement tous vos fichiers .java dans tous les sous-dossiers. La commande vous affichera également de nombreuses erreurs / warning qu'il faudra lire et corriger pour générer correctement la documentation.
    — La commande javadoc est inclue dans le Java Development Kit (jdk).

11 Notation

Le "jeu de base" (c'est à dire toutes les parties sauf "Défis") vous permettra d'obtenir la note de 14/20 si elles sont correctement réalisées :

— Le jeu doit être fonctionnel et respecter le cahier des charges ci-dessus.
— Le code doit présenter une conception objet cohérente.
— Le code doit être lisible, propre et commenté, avec un nommage des fonctions et variables compréhensible pour faciliter la lecture.
— Vous devez générer la Javadoc associée à votre projet.
— L'implémentation doit utiliser intelligemment les Collections, Exceptions, Héritage, Interfaces, etc.

Les défis proposés en Partie 7 vous permettent de gagner des points supplémentaires. Ci-dessous une idée des points que peuvent vous rapporter chaque défi.

— Nouvelles unités (toutes) : 3 points.
— Exception : 1 point.
— Sauvegarde : 1 point.

Vous pouvez aussi rajouter d'autres éléments de votre invention, que nous évaluerons et qui pourront donner lieu à des points bonus. Expliquez bien ces éléments dans votre fichier readme ! Et rappelez vous que nous sommes en cours de programmation, pas de graphisme : nous donnerons des points pour des améliorations demandant un travail de programmation et/ou d'algorithmique. Si vous mettez juste des jolies images, ce n'est pas ce qui nous intéressera...