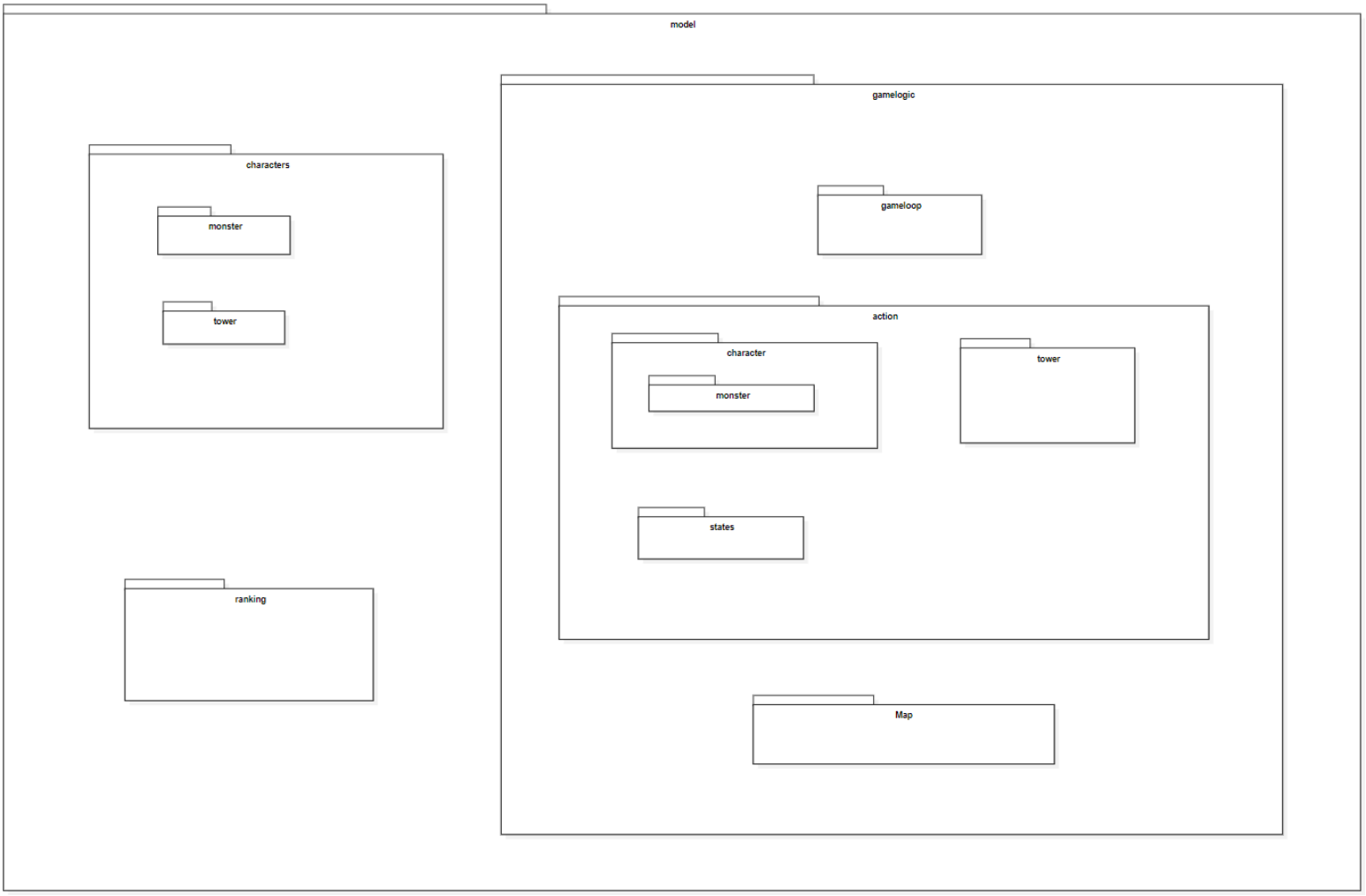
VIGNON UGO
MALEA BASTIEN

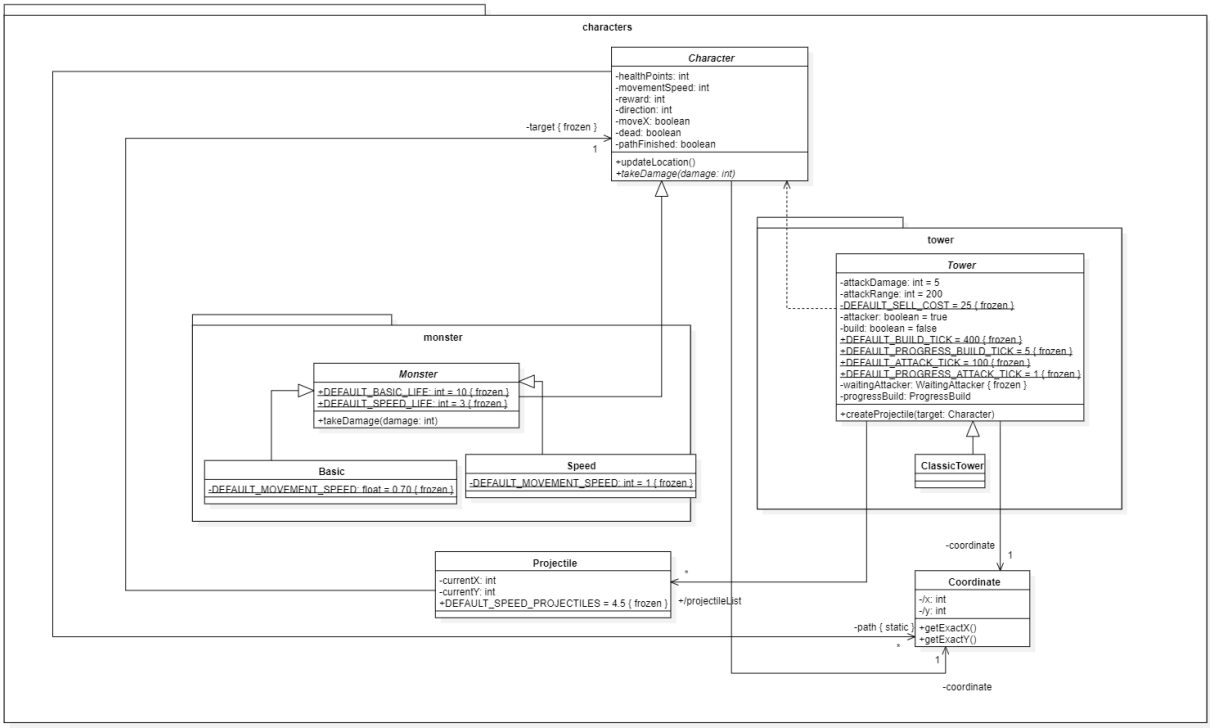
EXPLICATIONS DIAGRAMMES DE CLASSES

1. Package model



Structuration des packages pour le model

1.1 Package characters



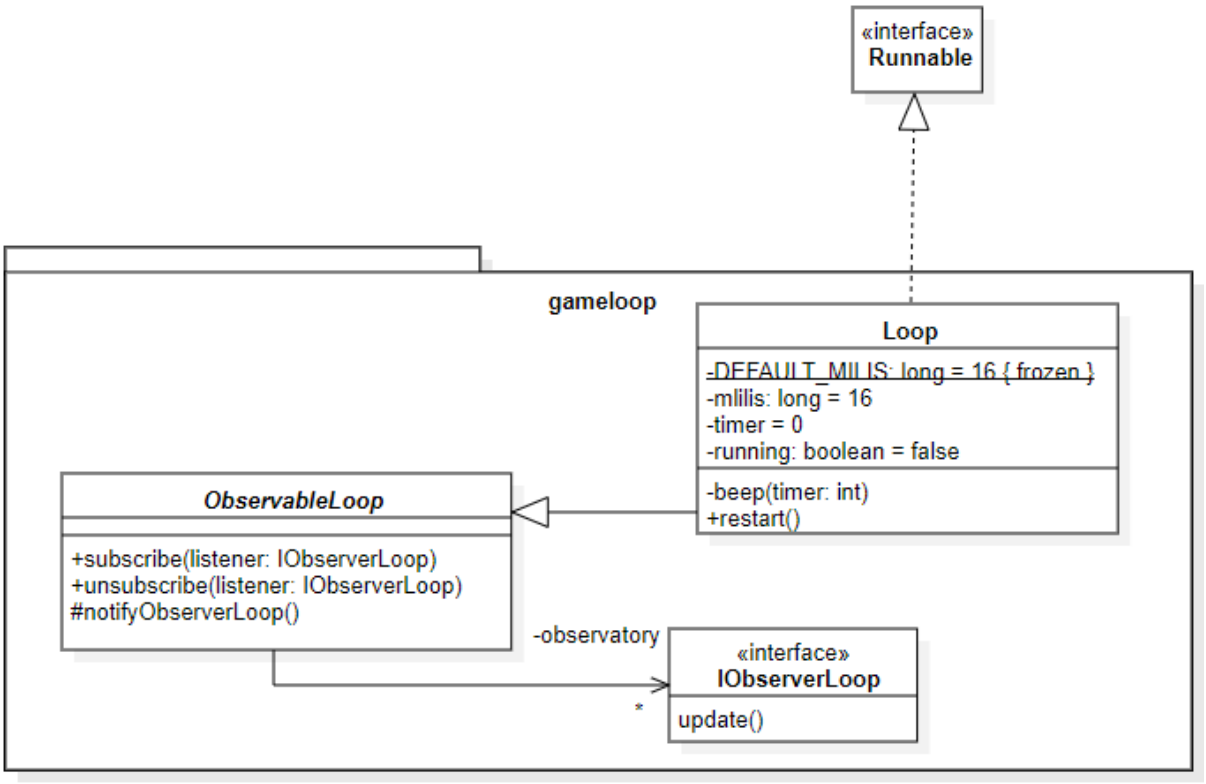
La classe Character est un point d'extensibilité et représente les entités de notre jeu. Il existe deux types de monstres les *Basic* ayant beaucoup de vie avec une vitesse faible et les *Speed* ayant une grande vitesse avec une faible vie.

La classe Coordinate représente la position d'un Character. Les méthodes *getExactX* et *getExactY* renvoient la position selon la taille des tuiles graphiques utilisées pour dessiner notre jeu.

Notre classe Tower est également un point d'extensibilité représente une Tour de défense. Le principe est qu'elle va tirer sur les monstres (1 à la fois) s'ils sont sa zone d'attaque. Cependant avant qu'une Tour puisse attaquer il faut qu'elle se construise. La construction d'une Tour est représentée avec la classe ProgressBuild. Sa cadence d'attaque est représentée avec la classe WaitingAttacker. Ces deux classes vous seront expliquées en détails plus tard.

La classe Projectile représente les projectiles tirés par les Tours.

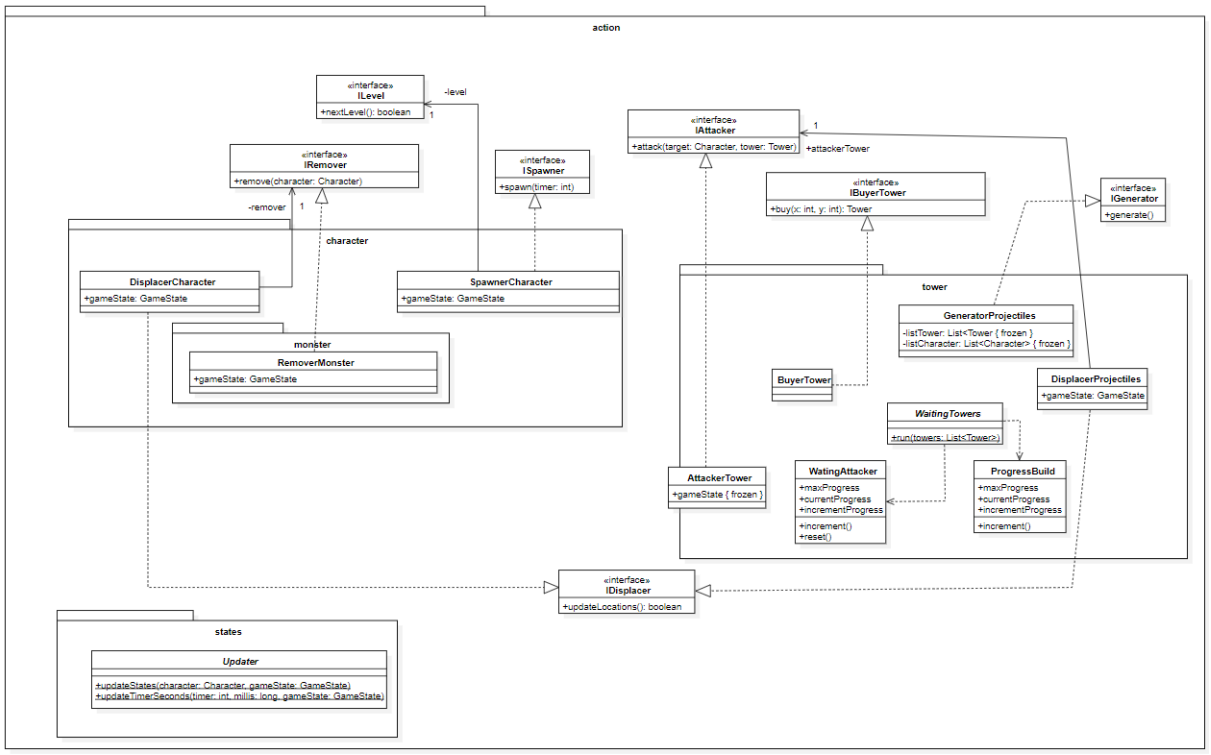
1.2 Package gameloop



L'idée générale est de faire tourner une boucle de jeu rythmant notre jeu. Cette boucle de jeu tourne dans un autre Thread qui n'est pas le Thread principal pour éviter que la fonction *sleep* endorme l'entièreté de notre Jeu. Pour endormir le Thread nous passons l'attribut *running* à false qui est ensuite perçu par notre boucle et l'appel à la méthode *wait* est déclenché. La méthode *restart* nous sert ensuite à réveiller notre Thread en passant *running* à true ainsi qu'en appelant la méthode *notify*. Le timer est utile pour avoir une gestion du temps et sera utilisé par le fonctionnement de notre jeu.

Notre classe Loop implémente la classe abstraite ObservableLoop, ce système nous permet de ne pas faire de l'attente active (patron de conception Observateur). Chaque tour de boucle appel la méthode *beep* qui va permettre de notifier tous les observateurs de notre boucle.

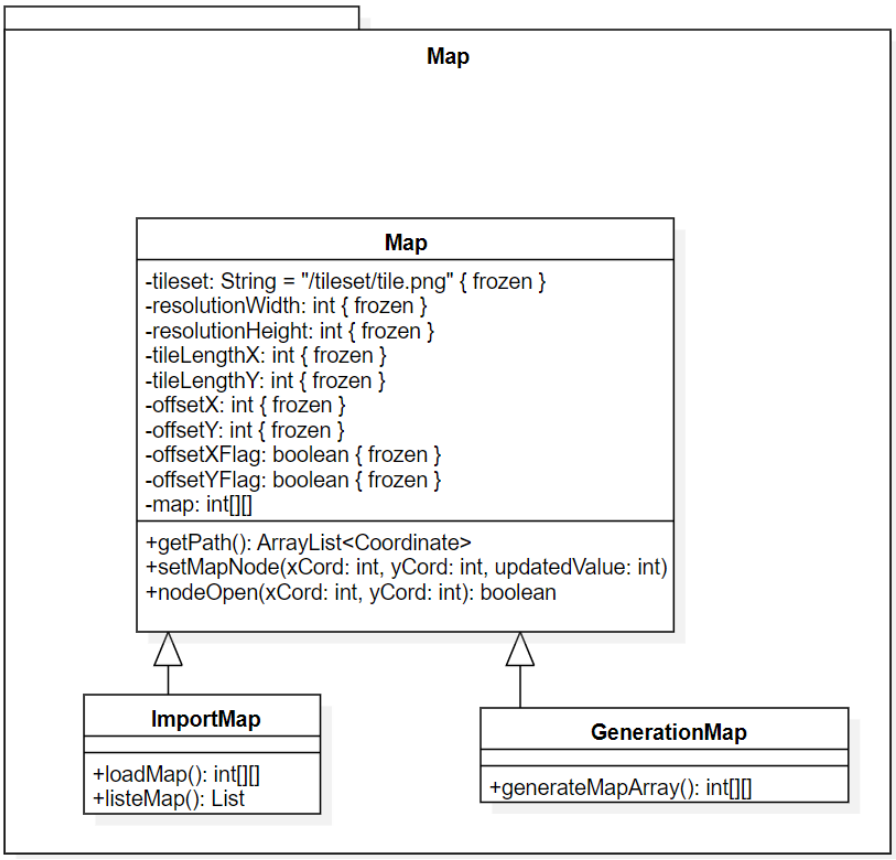
1.3 Package action



Le package action contient les actions relatives à la logique du jeu. La classe SpawnerCharacter utilise la classe AdministratorLevel qui vous sera expliqué plus tard. Ce qui est important c'est que nos niveaux dans le jeu sont scriptés. Les fichiers des niveaux sont lus et en fonction de ce qui est retourné, la bonne instance de Character est faite. Les différentes interfaces nous servent à créer plusieurs points d'extensibilités. La classe BuyerTower à comme responsabilité l'achat de Tower si différents critères sont remplis. WaitingAttacker représente l'attente entre chaque attaque d'une Tower et ProgressBuild représente la progression de la construction d'une Tower. La classe WaitingTowers parcourera la liste des Towers en jeu et déclenchera ou mettra à jour l'attaque ou la construction. La classe GeneratorProjectiles va générer des projectiles si les Character sont dans la zone d'attaque des Tower.

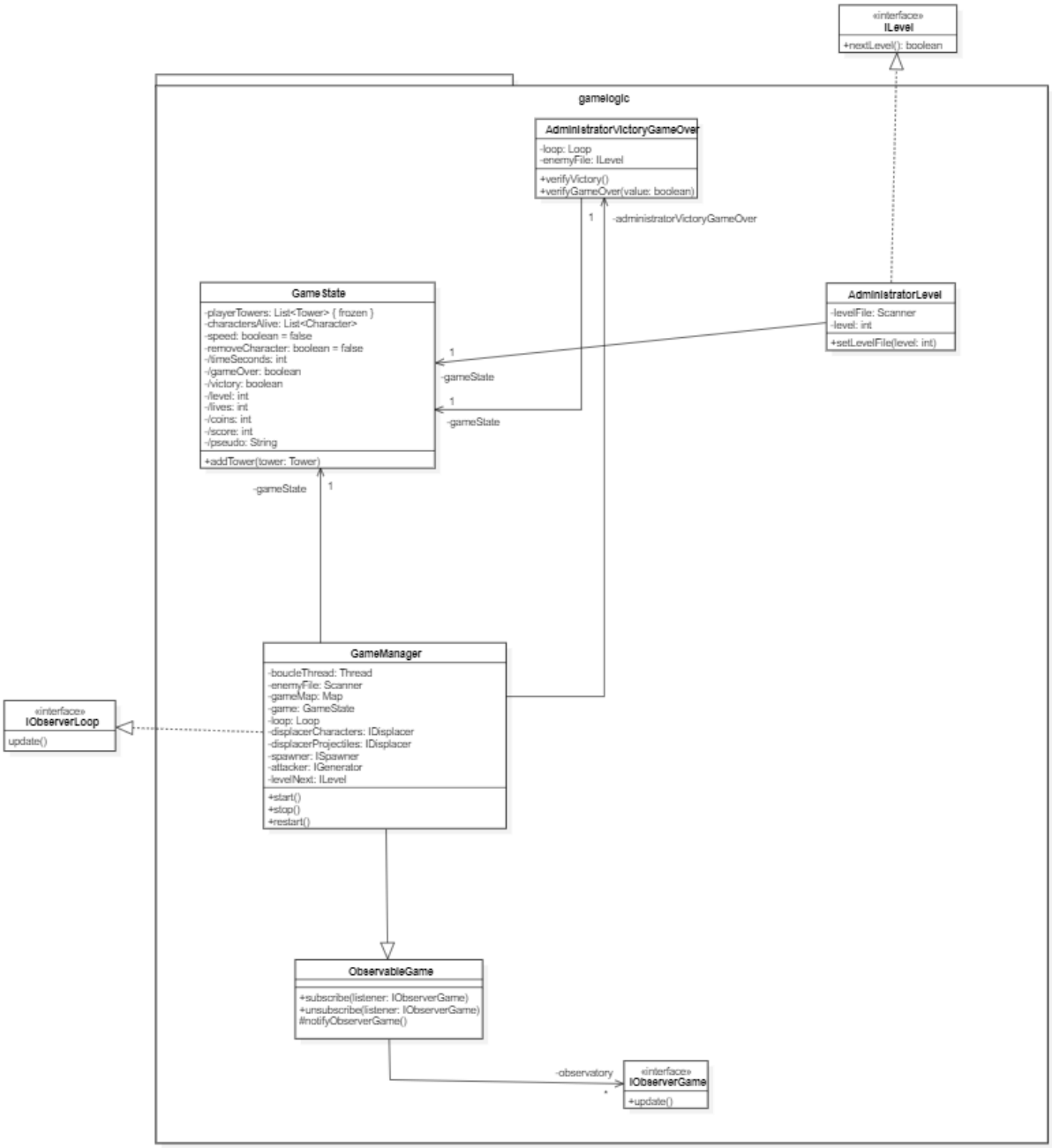
La classe Updater est une classe utilitaire permettant de mettre à jour les states de la partie lorsqu'un Character est tué ainsi que le temps de la partie

1.4 Package Map



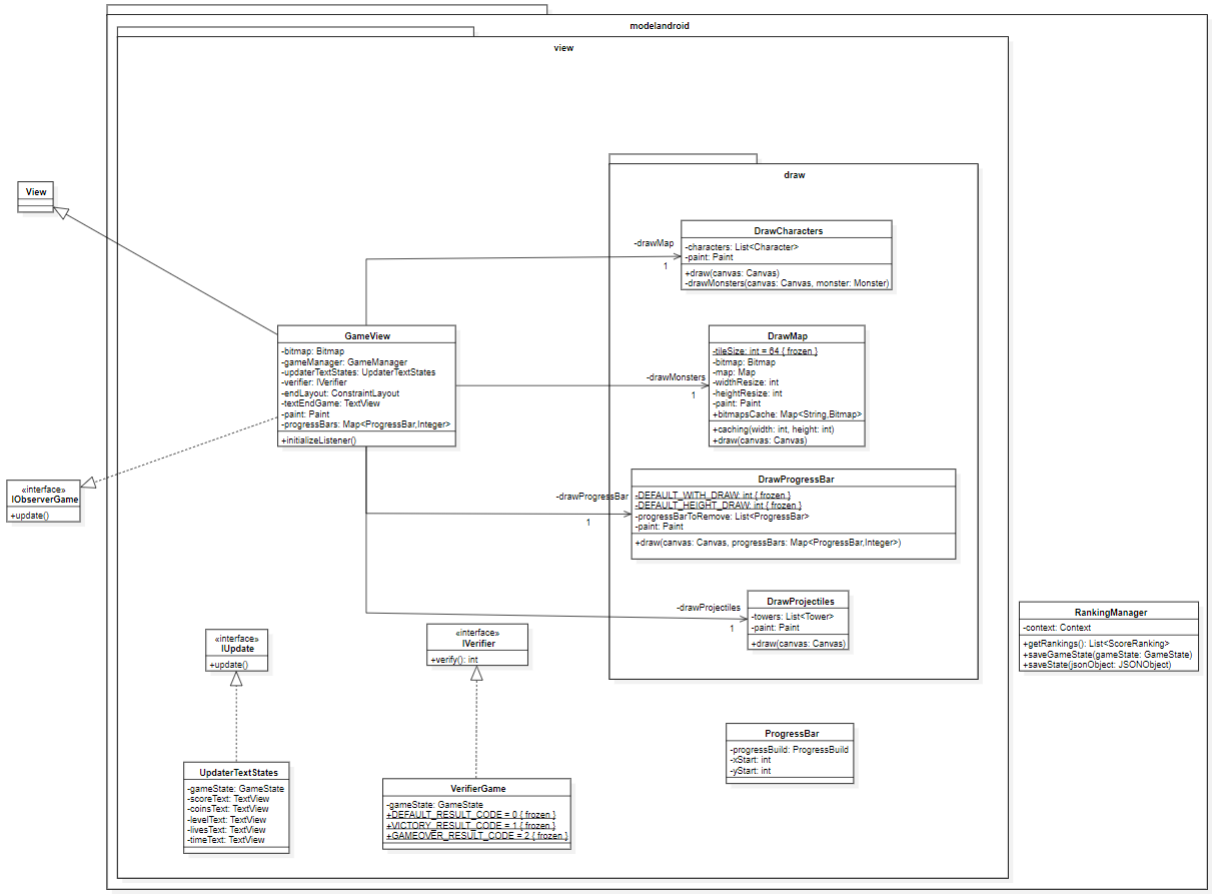
La classe Map est notre carte de jeu. Elle est formée de tuiles graphiques de 64 par 64 pixels. Cette classe va permettre de retourner un enchainement de positions qui sera le chemin prédéfini suivi par les ennemis durant les parties. Pour générer cette carte dans le modèle nous utilisons un tableau à 2 dimensions, ou chaque valeur, sera interprétée lors de la création de la carte visuelle, pour savoir quelle tuile dessiner.

1.5 Package gamelogic



Notre GameManager est là pour manager la partie en cours. Le GameManager est le seul Observateur de la boucle car nous avons besoin d'enchaîner des étapes séquentiellement, les unes après les autres dans un certain ordre. Notre classe GameState nous permet de contenir les states de la partie en cours. C'est l'AdministratorLevel qui gère les niveaux en lisant dans les fichiers des niveaux. L'AdministratorVictoryGameOver sert simplement à vérifier si la partie est gagnée ou perdu. L'observateur de la partie est la vue. La vue est notifiée à la fin des changements effectués.

2. Package modelandroid



La partie graphique de notre jeu est faite via la CustomView GameView. Quand la GameView est notifiée, la vue est invalidée pour que celle-ci soit redessinée. La partie de dessin est découpée en plusieurs classes pour dessiner la map, les Character, ProgressBar (pour les Tower) et les projectiles. DrawMap utilise un tileset pour dessiner les différentes tuiles. Petite particularité pour ne pas perdre en performance, les tuiles sont sauvegardées dans une HashMap via la méthode onSizeChanged de la GameView. VerifierGame sert à vérifier si la partie est perdue ou gagnée pour afficher le texte de fin de partie. UpdaterTextStates sert à modifié les différents textes relatifs aux states de la partie. ProgressBar est la barre de construction des tours, elle possède un x et un y pour avoir la position de dessin.

Le RankingManager à le role de faire la persistance profonde. Pour cela nous utilisons les préférences. Les données du tableau des scores sont enregistrées dans des JSONObject qui sont ensuite enregistrés dans un JSONArray