Notre GameManager est là pour manager la partie en cours. Il implémente l'interface IObserver. Notre classe Loop implémente l'interface IObservable, ce système nous permet de ne pas faire de l'attente active (patron de conception Observateur). 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.

Nous avons délégué la responsabilité de vérification de victoire ou de jeu terminé à la classe AdministratorVictoryGameOver

Notre classe GameState nous permet de contenir les states de la partie en cours

Nos classes Monster et Tower nous permettent de créer un point d'extensibilité et de pouvoir rajouter facilement à l'avenir des nouveaux monstres et des nouvelles tours. Il en est de même pour la classe Character, si un jour nous voulons rajouter d'autres types d'ennemis.

Notre classe ScoreRanking est notre « tableau des scores », nous permettant de stocker les différents states des parties. Si le nombre maximum de scores dans le tableau est atteint, la méthode updateRanking va vérifier, à l'ajout d'un autre GameState, si le GameState est supérieur à au moins un autre GameState. Si c'est le cas, le plus petit GameState du tableau des scores sera remplacé par celui-ci en fonction de notre comparaison (compareTo) des GameState.

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.

Dans le package action, il y a toutes les classes, interfaces relatives aux actions durant la partie. Nous avons globalement créé des points d'extensibilité pour chaque action, permettant de faire des rajouts pour chaque action. Cependant nous n'avons pas trouvé pertinent de créer des points d'extensibilité pour l'action Updater ainsi que pour WaitingBuild. La classe WaitingBuild est un peu particulière car elle implémente l'interface Runnable. Elle va nous permettre, pour une tour, de désactiver son attaque, d'attendre 2 secondes puis d'activer son attaque. Les 2 secondes correspondent au temps de construction d'une tour. Pour faire cette attente nous faisons cet enchainement d'action dans un autre Thread. Cela nous permet de ne pas endormir le Thread principale.

Dans le package view, notre classe DrawMap hérite d'ImageView et nous permet de dessiner notre carte visuelle en interprétant les valeurs de notre tableau à 2 dimensions. DrawMap est ensuite interprétée comme une ImageView par la vue.

Dans notre jeu, notre ScoreRanking ainsi que notre GameState ne sont pas sérialisables car ils possèdent des propriétés. Pour remédier à ce problème nous avons créé les classes ScoreRankingSerializable et StateSerializable qui se font passer pour ScoreRanking et GameState mais sans avoir de propriété. Ce procédé fait penser au patron de conception Procuration, cependant nous nous sommes juste inspirés de l'idée. Nous avons implémenté cette idée à notre façon. De ce fait, lors de la sauvegarde il est persisté un ScoreRankingSerializable possédant des StateSerializable. Lors du chargement il est récupéré les données sérialisées qui sont transformées en ScoreRanking possédant des GameState. La responsabilité de persistance des données est déléguée à la classe AdministratorPersistence, qui est un point d'extensibilité nous permettant de rajouter, si on le souhaite, des façons de persister les données. Nous avons utilisé le patron de conception Stratégie,

nous permettant de changer l’algorithme de chargement et de sauvegarde des données en fonction de la classe AdministratorPersistence instanciée.