Dossier du Projet (19-20)

Groupe : Ilyès ANNOU, Mahoutondji ZINSOU, Byzid KHAN

Ce fichier reprend la description des documents demandés pour le projet, ainsi que quelques renseignements que vous devez fournir.
Vous devez le compléter et le mettre à disposition sur git.

Tous les diagrammes devront être fournis avec un texte de présentation et des explications.

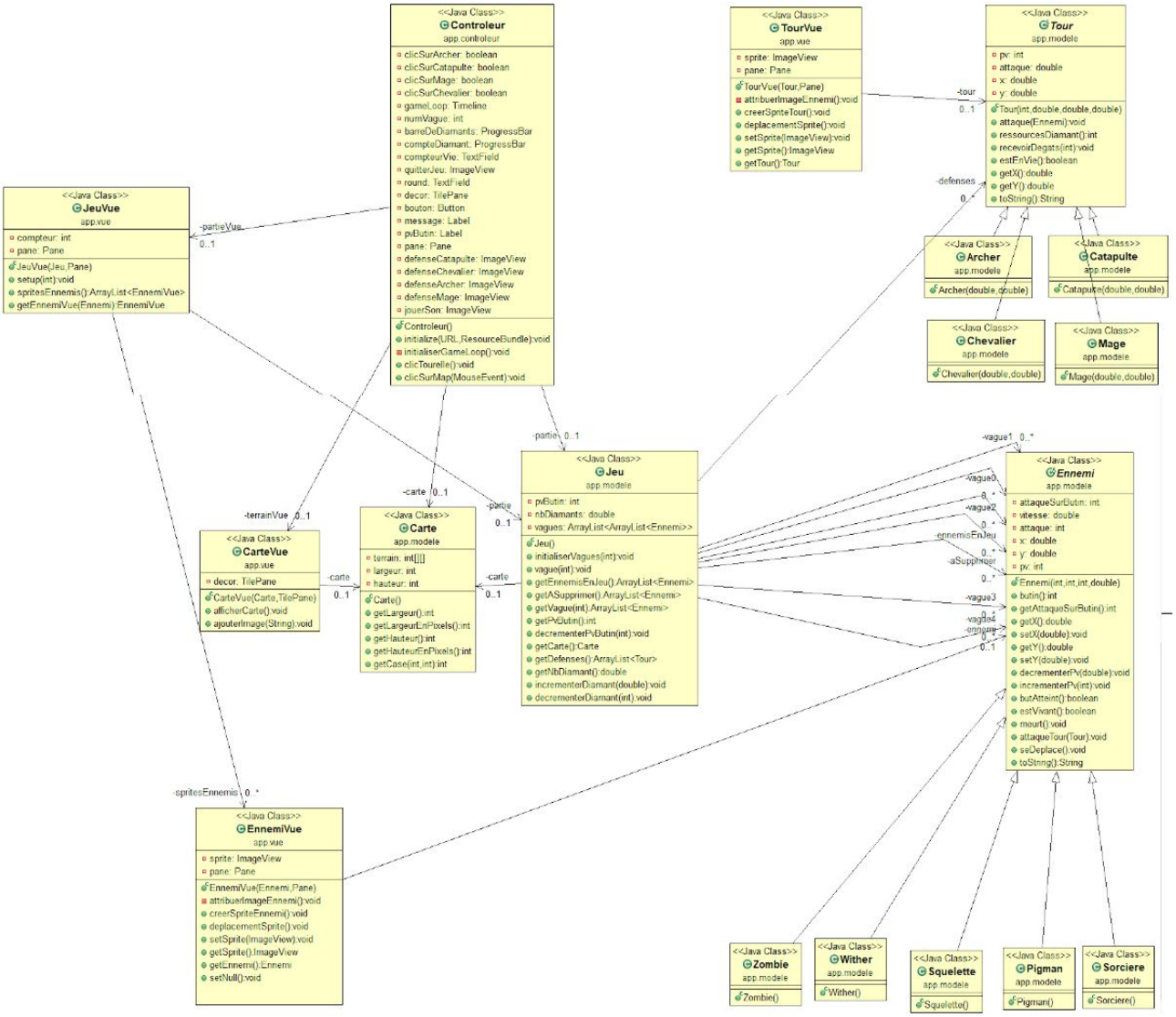
CPOO		IHM		Projet Tuteuré		Gestion de Projet	
diagrammes d'architecture	9	ergonomie et contenus	15	soutenance	10	document utilisateur	8
diagrammes de classes	9	programmation évènementielle et JavaFX	35	niveau et qualité du produit fini	15	Trello	6
diagrammes de séquence	9			notes de sprints	15	Git	6
POO	24						
Structures de données	6						
gestion des erreurs	5,5						
tests	5,5						
algos	17						
ampleur et qualités du code	25						
	110		50		40		20

Documents pour CPOO (sur 27)

Chaque diagramme utilisé à des fins de documentation doit être focalisé sur un objectif de communication. Il ne doit par exemple pas forcément montrer toutes les méthodes et dépendances, mais juste ce qui est nécessaire pour montrer ce que le diagramme veut montrer. Chaque diagramme doit être commenté.

1.1 Architecture (9 points)

Un texte présentant vos choix principaux d'architecture (en dehors des choix que nous vous imposons comme MVC par exemple), ainsi que des diagrammes de classe commentés permettant de la visualiser.

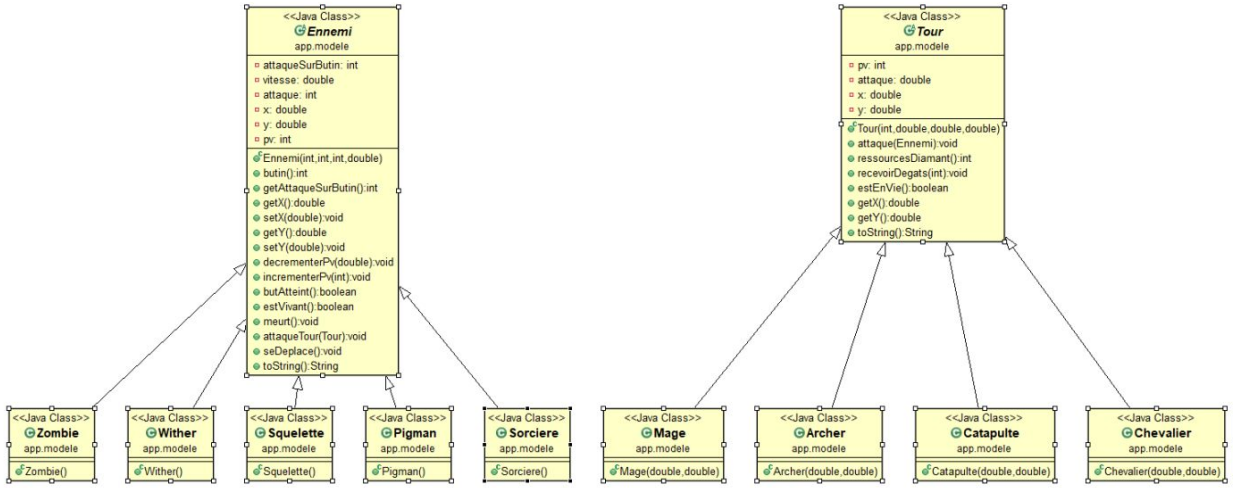


Nos choix principaux ont été les suivants :

Tout d'abord, nous avons utilisé le logiciel TileMap pour créer le design de la Map avec des tuiles formatés en 32 * 32 du célèbre jeu Minecraft. Ensuite, la TileMap a été converti en un tableau à deux dimensions grâce à Json où chaque case contient un Integer. Grâce à cette Integer, nous avons pu attribuer à chaque case la tuile correspondante. Les images correspondant aux tuiles ont été disposés sur un TilePane, la vue des Tours et Ennemi sur un Pane. La superposition du Pane sur le TilePane a été réalisé grace à un StackPane. Nous avons créé les classes Ennemi et Tour possédant des polymorphismes, et les classes Carte et Jeu, toutes ces classes possèdent une classe associée dans la vue.

1.2 Détails : diagrammes de classe (9 points)

Vous choisirez des parties du modèle que vous considérez particulièrement intéressantes du point de vue de la programmation à objets (héritage, composition, polymorphisme…). Vous expliquerez vos choix et les illustrerez par des diagrammes de classe.



La classe Ennemi et Tour sont particulièrement intéressante car il y a un polymorphisme. En effet, elles possèdent chacune des sous classes.

La classe Ennemi possède les sous classes suivantes : Zombie, Pigman, Wither et Squelette. Elle possède également les attributs suivants : pv, attaque, vitesse, attaqueButin et la position en x et y. Ainsi, les *super* des sous classes qui extends la classe Ennemi permettent d'insérer des attributs spécifiques et non modifiable (pv, attaque, vitesse, attaqueButin) depuis d'autres classes (ex : Jeu).

De plus, la classe Tour possède les sous classes suivantes : Chevalier, Mage, Catapulte et Archer. Elle possède également les attributs : pv, attaque et la position en x et y. Ainsi, les super des sous classes qui extends la classe Tour permettent d'insérer des attributs spécifiques et non modifiable (pv, attaque) depuis des autres classes (ex : Jeu).

1.3 Diagrammes de séquence (9 points)

Vous choisirez une ou deux méthodes intéressantes du point de vue de la répartition des responsabilités entre différentes entités du programme. Vous utiliserez des diagrammes de séquence pour expliquer l'exécution de ces méthodes.

```
private void initialiserGameLoop() {

    this.gameLoop = new Timeline();
    this.gameLoop.setCycleCount(Timeline.INDEFINITE);
    KeyFrame frame = new KeyFrame(Duration.seconds(0.016),
    (ev -> {
        if(partie.getPvButin() > 0) {
            compteDiamant.setProgress(partie.getNbDiamant());
            partie.vague(numVague);
            partieVue.setup(numVague);
            for(Ennemi e: partie.getEnnemisEnJeu()) {

                e.seDeplace();
                partieVue.getEnnemiVue(e).deplacementSprite();

                for(Tour t : partie.getDefenses()) {
                    t.attaque(e);
                }

                if(!e.estVivant()) {
                    partie.getASupprimer().add(e);
                }

                if(e.butAtteint()) {
                    e.meurt();
                    partie.decrementerPvButin(e.getAttaqueSurButin());
                }
            }

            for(Ennemi e : partie.getASupprimer()) {
                pane.getChildren().remove(partieVue.getEnnemiVue(e).getSprite());
                partieVue.spritesEnnemis().remove(partieVue.getEnnemiVue(e));
                partie.getEnnemisEnJeu().remove(e);
                partie.incrementerDiamant(e.butin());
            }

            partie.getASupprimer().clear();
            partie.incrementerDiamant(0.1);
            pvButin.setText("" +partie.getPvButin());
        }

        else {
            round.setText("Game Over");
        }
        barreDeDiamants.setProgress(partie.getNbDiamant() / 30);
    }));
    gameLoop.getKeyFrames().add(frame);
}
```

Tout d'abord, la méthode initialiserGameLoop() sert de game loop dont le cycle est effectué toutes les 0,016 secondes. On rentre dans une première conditions si la vie du butin est supérieur à 0. La barre de diamant représenté par une progressBar est initialisé. Ensuite, le modèle et la vue des ennemis sont initialisés avec les méthodes partie.vague(numVague); et partieVue.setup(numVague);. Puis chaque modèle suivi de la vue de chaque ennemi se déplace à l'aide d'une boucle for dans laquelle, une autre boucle fait attaquer chaque tour présent dans la liste getDefense(). Aussi, des conditions vérifie que lorsque l'ennemi est mort, il doit être ajouté à la liste getAsupprimer() qui le supprimera et que lorsque l'ennemi atteint le butin, il meurt et le butin doit être décrémenter en fonction du type de l'ennemi. A la sortie de la boucle, une nouvelle boucle parcourant les ennemis à supprimer les suppriment du pane, c'est à dire de la vue, puis du

modèle. La barre de diamant est incrémenté en fonction du type de l'ennemi car à chaque qu'un ennemi meurt, la barre de diamant reçoit des diamants.

De plus, à la sortie de cette nouvelle boucle, la liste getASupprimer est nettoyé et la barre de diamant est incrémenté de 0,1. L'entier représentant la vie du butin est mise à jour en fonction des dégâts reçu.

Enfin, on rentre dans une deuxième condition si la vie du butin est inférieur ou égal à 0. "Game over" est affiché dans le TextField round. La progressBar effectue les modifications sur l'interface.

Diagramme de séquence disponibles en pdf sur GitHub !

1.4 Structures de données :

indiquez ici quelles structures de données avancées vous utilisez et dans quelles classes.

Aucune structure de données avancées n'a été utilisé, la structure de donnée la plus avancée que nous avons utilisé est l'ArrayListe.

1.5 Exception :

indiquez ici quelle(s) classes(s) contiennent une gestion d'exception intéressante.

Aucune classe ne contient une gestion d'exception.

1.6 Utilisation maîtrisée d'algorithmes intéressants :

indiquez ici quels algorithmes sont à prendre en compte et le nom des classes qui les contiennent.

La méthode clicSurMap de la class Controleur.java contient un algorithme assez intéressent.

1.7 Junits :

indiquez ici quels classes sont couvertes par vos Junits.

La class EnnemiTest.java couvre une méthode de la class Ennemi.

2 Documents pour Gestion de projet

2.1 Document utilisateur (8 points) :

Description du jeu (son objectif, son univers…)

Le jeu est inspiré de l'univers de Minecraft, on peut y retrouver des monstres du jeu comme les squelettes, zombies ou pigmans. L'objectif principale du jeu est d'empêcher le vol de notre butin par les monstres en plaçant des tourelles pour le défendre. L'objectif du joueur est d'éliminer les ennemis qui convoitent nos diamants en formant des guerriers et défenses pour les défendre.
Les ennemis arrivent en nombre afin de voler le butin du joueur.
Le Joueur dispose d'une barre de diamant qui peut être rempli avec 30 diamants au maximum.
La barre de diamant se recharge à chaque tour de la GameLoop ainsi qu'à la mort d'un ennemi. Le nombre de diamant obtenu est différent selon le type d'ennemi abattu. Chaque tourelle a un coût différent.

Description des tours et des ennemis (force et faiblesse les une par rapport aux autres , tableau de relation).

Les tours sont réparties en quatre classes :

Archer : Cause peu de dommage et le coût de son placement est le moins cher.
Chevalier : Cause des dommages importants et a le coût en diamant est le plus cher.
Catapulte : Cause le plus de dommages et le coût en diamant est le troisième plus cher.
Mage : Cause le moins de dommages aux ennemis et était censé ralentir les ennemis.

Tour	Chevalier	Mage	Catapulte	Archer
Dégât	++++	+	+++++	++
Coût	++++	++	+++	+

Les ennemis sont réparti en cinq classes :

Wither : Fréquence d'apparition unique. L'ennemi final disposant de beaucoup de point de vie et de dégât d'attaque. Cependant, il est très lent.

Sorcière : Fréquence d'apparition rare. Elle inflige beaucoup de dégât, dispose d'une bonne vitesse et de beaucoup de point de vie.

Squelette : Fréquence d'apparition normal. Inflige peu de dégât et n'a pas beaucoup de point de vie mais il dispose d'une bonne vitesse

Pigman : .Fréquence d'apparition normal. Une version amélioré du Squelette.

Ennemi	Wither	Sorcière	Squelette	Pigman
Degat butin	+++++	+++	+	++
Diamant	Fin du jeu	+++	+	++

Scoring.

Cette fonctionnalité n'a pas pu être réalisé.

Toutes fonctionnalité définie et utilisable doit être documentée. A l'inverse il est hors de question (et donc pénalisant) de documenter une fonctionnalité non encore utilisable.

Voici une liste de toute les fonctionnalités définie et utilisable :

- Le joueur peut cliquer sur l'image view correspondant à une tour (Controleur.clicTourelle() : ligne 132) puis cliquer sur une partie du terrain (stone) pour la poser. (Controleur.clicSurMap() : ligne 157)

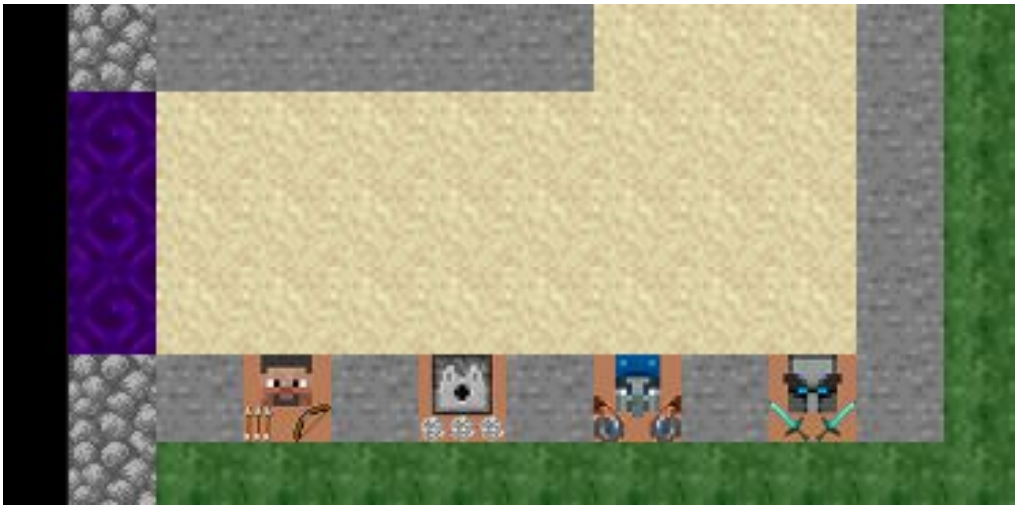
  clicTourelle

```
public void clicTourelle() {
    defenseArcher.setOnMouseClicked(e ->{clicSurMap(e);
        this.clicSurArcher = true;
    });

    defenseCatapulte.setOnMouseClicked(e ->{clicSurMap(e);
        this.clicSurCatapulte = true;
    });

    defenseMage.setOnMouseClicked(e ->{clicSurMap(e);
        this.clicSurMage = true;
    });

    defenseChevalier.setOnMouseClicked(e ->{clicSurMap(e);
        this.clicSurChevalier = true;
    });

    quitterJeu.setOnMouseClicked( e -> System.exit(0));

    jouerSon.setOnMouseClicked(e ->{
        AudioClip note = new AudioClip(this.getClass().getResourc
        note.play();
    });
}
```

clicSurMap

```
public void clicSurMap(MouseEvent ev) {
    for (int i = 0; i < decor.getChildren().size(); i++) {

        if(decor.getChildren().get(i).getId() == "stone"  && partie.getNbDiamant()>=5  ){
            decor.getChildren().get(i).setOnMouseClicked(e ->{
                Node tuile = (Node) e.getSource();
                if(clicSurArcher == true) {
                    Archer archer = new Archer(tuile.getLayoutX(),tuile.getLayoutY());
                    if( partie.getNbDiamant()>=archer.ressourcesDiamant()) {
                    partie.getDefenses().add(archer);
                    TourVue tourelle = new TourVue(archer, pane);
                    partie.decrementerDiamant(archer.ressourcesDiamant());
                    clicSurArcher = false;
                    }
                }

                else if(clicSurCatapulte == true) {
                    Catapulte catapulte = new Catapulte(tuile.getLayoutX(),tuile.getLayoutY());
                    if( partie.getNbDiamant()>=catapulte.ressourcesDiamant()) {
                    partie.getDefenses().add(catapulte);
                    TourVue tourelle = new TourVue(catapulte, pane);
                    partie.decrementerDiamant(catapulte.ressourcesDiamant());
                    clicSurCatapulte= false;
                    }
                }

                else if(clicSurMage == true) {
                    Mage mage = new Mage(tuile.getLayoutX(),tuile.getLayoutY());
                    if( partie.getNbDiamant()>=mage.ressourcesDiamant()) {
                    partie.getDefenses().add(mage);
                    TourVue tourelle=new TourVue(mage, pane);
                    partie.decrementerDiamant(mage.ressourcesDiamant());
                    clicSurMage= false;
                    }
                }

                else if(clicSurChevalier == true) {
                    Chevalier chevalier = new Chevalier(tuile.getLayoutX(),tuile.getLayoutY());
                    if( partie.getNbDiamant()>=chevalier.ressourcesDiamant()) {
                    partie.getDefenses().add(chevalier);
                    TourVue tourelle=new TourVue(chevalier, pane);
                    partie.decrementerDiamant(chevalier.ressourcesDiamant());
                    clicSurChevalier= false;
                    }
                }
            });
        }
    }
}
```

Tourelle posée



Interface :

- Chaque tour va attaquer les ennemis dans son rayon d'action (Tour.attaque(Ennemi e).

Rayon d'action de la tourelle :

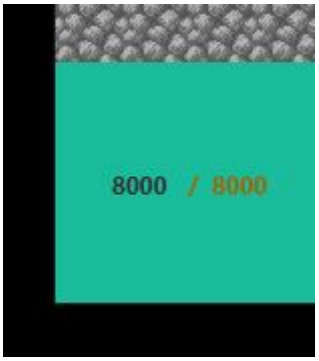
- Si les ennemis parviennent à voler tout le butin, c'est Game Over

- En cliquant sur le bouton accueil, on quitte le jeu. (nom methode, image accueil)

- Le joueur doit avoir le nombre de diamant nécessaire avant de pouvoir poser une tour. C'est à dire, la tour n'est pas posé au moment du clic. (Controleur.java)


```
127                 barreDeDiamants.setProgress(partie.getNbDiamant() / 30);
165                         if( partie.getNbDiamant()>=archer.ressourcesDiamant()) {
168                             partie.decrementerDiamant(archer.ressourcesDiamant());
```

- L'entier représentant le butin restant décroît à chaque dégats infligés sur l'interface.


8000 / 8000