IUT Institut Universitaire Technologie LENS

Rapport de projet du second semestre

2020/2021

Louis Iwinski / Lucas Goulois / Cameron Delsol / Alexis Grislin

Objectif :

Réalisation du jeu de société Labyrinthe en jeu vidéo avec l'aide de langages informatiques (Java…)

Professeur tuteur : Monsieur Blomme

Rapport de projet du second semestre

Tables des matières

Notre rapport de projet se divise en six parties :

1) Introduction : Présentation

Voici une petite introduction qui présente notre vision du projet.

Lors de ce second semestre en DUT Informatique, on nous a proposé de se mettre en groupe afin de réaliser le jeu de société Labyrinthe en jeu vidéo à l'aide de divers outils comme le langage de programmation Java ou encore plantUML pour le dessin de diagrammes de classes. En groupe de quatre, nous devions réaliser ce jeu vidéo en grande partie à partir d'un énoncé. Ce projet était séparé en six étapes.

Ce projet était réparti en six étapes. La première étape consistait à nous familiariser avec la librairie graphique proposée, d'utiliser l'environnement Git ainsi que de créer le projet Java dans un environnement de programmation. La deuxième permettait d'implémenter les classes représentant les pièces du jeu. L'objectif de la troisième était d'implémenter les classes représentant les objets et le plateau du jeu. La quatrième étape permettait de créer les classes représentant les joueurs. L'avant-dernière étape consistait à l'implémentation de classes représentant l'ensemble des éléments constituant une partie et la partie. La dernière avait pour objectif principal l'implémentation de joueurs de type *ordinateur*. Nous devions rendre une étape chaque semaine afin que le professeur tuteur puisse voir notre avancée dans le projet.

De plus, on avait pour objectif de s'améliorer dans les différents langages, mais plus précisément dans le langage Java pour la conception du jeu vidéo. Certains membres se sont servis de UML pour réaliser les diagrammes de classes. L'utilisation d'un Environnement de Développement Intégré (IDE) était primordiale pour la réalisation de ce projet.

Le jeu qu'on devait programmer, durant ce projet, était donc *Labyrinthe*. Depuis sa création en 1986, Labyrinthe est un jeu de société qui consiste en une chasse aux trésors dans un labyrinthe en mouvement. Chaque joueur déplace les murs du labyrinthe pour se frayer un chemin qui mène aux trésors. On croit pouvoir atteindre un trésor quand soudain les murs coulissent… Notez qu'un joueur est toujours obligé de modifier le labyrinthe avant de pouvoir déplacer son pion et ce même si son «tréson» est accessible directement.

Un jeu de réflexion et de suspense pour toute la famille, de 2 à 4 joueurs à partir de 7 ans. Le contenu est le suivant : 49 pièces (trois modèles différents), 18 objets et 3 joueurs. Nous avons aussi choisi de réaliser un thème Pop Culture pour que cela corresponde à n'importe quel joueur.

Photo du jeu de société Labyrinthe

Pour finir, nous avons choisi de partager tous nos codes sur GitHub car cela permet de regrouper les codes des différents membres sans avoir de soucis.

2) Explications et détails de certains algorithmes

a) Tirage aléatoire de valeurs entières (voir annexe 2)

Cet algorithme se trouve dans la classe Utils. Pour cet algorithme, on a utilisé une constante génératrice qui est un objet de type Random. Dans la méthode, on a généré un nombre aléatoire entre 0 et un entier passé en paramètre.

Ensuite, on a initialisé un tableau de longueur égale au paramètre mis (ici, 49). Par la suite, on a créé un tableau de valeurs exclues. On a donc fait une boucle qui parcourt tous les éléments du dernier tableau. On récupère les numéros tirés et on les met dans une liste « exclude », on enlève les doublons.

b) Recherche de chemins (voir annexe 3)

Cet algorithme se trouve dans la classe Plateau. Tout d'abord, on vérifie si la case de départ et la case d'arrivée ne sont pas identiques. Puis, on définit un booléen à « True », on réalise une boucle while. Tant que le booléen est égal à « True », on fait un système de parcours en profondeur (pile). En enlevant les cases qui sont déjà visitées pour ne pas les considérer à nouveau.

On fait, par la suite, une sous-méthode qui se nomme possibiliteCasesAdjacentes. On réalise une boucle de 0 à 3 et on utilise un « switch » pour voir si on peut se déplacer sur les cases adjacentes. On parcourt la liste avec une boucle et on réalise un « exclude.stream.noOneMatch » qui permet d'éviter d'aller à une case déjà rencontrée.

On a maintenant toutes les possibilités, on peut enfin réaliser le parcours en profondeur. On récupère le chemin avant la place actuelle, on crée une variable

du chemin parcouru et on ajoute le chemin en cours. Puis, nous testons si nous avons atteint la case d'arrivée (variable chemin final), on parcourt la « map » du chemin. Donc on retourne le contenu de la variable cheminFinal. S'il n'y pas de chemin, on retourne la valeur null.

c) Boucle principale (voir annexe 4)

Cet algorithme se situe dans la classe Partie. Nous initialisons deux messages, un message pour le choix de l'orientation puis un message du choix du chemin. On crée une variable qui va contenir l'indice du joueur dont c'est le tour.

On fait une boucle « while ». On récupère le joueur et on change l'objet qu'il va récupérer. On change le message énoncé juste avant par le nom du joueur. On lui demande de choisir là où il va se déplacer. Tant que le joueur choisit une case valable, tout d'abord, nous regardons si le joueur choisit une case qui retourne null alors la case d'arrivée devient la case de départ. On a donc la case temporaire, on fait appel à la méthode calculeChemin (expliqué au 2. B). Si un chemin existe, on le parcourt et on déplace le joueur.

Toujours dans cette boucle, on va voir si à la case d'arrivée il y a un objet à récupérer, si oui le joueur le prend. Puis, on vérifie si le joueur a récupéré tous les objets qu'il devait prendre. Si c'est le cas, nous le désignons comme gagnant et nous l'affichons. Sinon, on passe au prochain joueur.

3) Comment les modules de POO et COO ont été utilisés

a) POO : Programmation orientée objet

La Programmation orientée objet (POO) est un module que l'on a rencontré durant ce second semestre du DUT Informatique. La Programmation orientée objet est un paradigme (c'est-à-dire une façon d'approcher la programmation et de traiter les solutions aux problèmes) de la programmation informatique, elle consiste en la définition ainsi qu'en l'interaction de briques logicielles appelées, entre autres, « objets ».

Dans la réalisation de ce projet, la POO était un module important car nous avons utilisé le langage orienté objet qui se nomme **Java**. Dans la totalité du projet, ce langage était exploité.

```java
public Objet[] getObjetsJoueurGeneral(Objet objets[]) {
    Objet resultat[] = new Objet[objetsJoueur.length];
    for (int i = 0; i < objetsJoueur.length; i++)
        for (int j = 0; j < objets.length; j++)
            if (objets[j].getNumeroObjet() == objetsJoueur[i].getNumeroObjet())
                resultat[i] = objets[j];
    return resultat;
}
```

Exemple de code Java

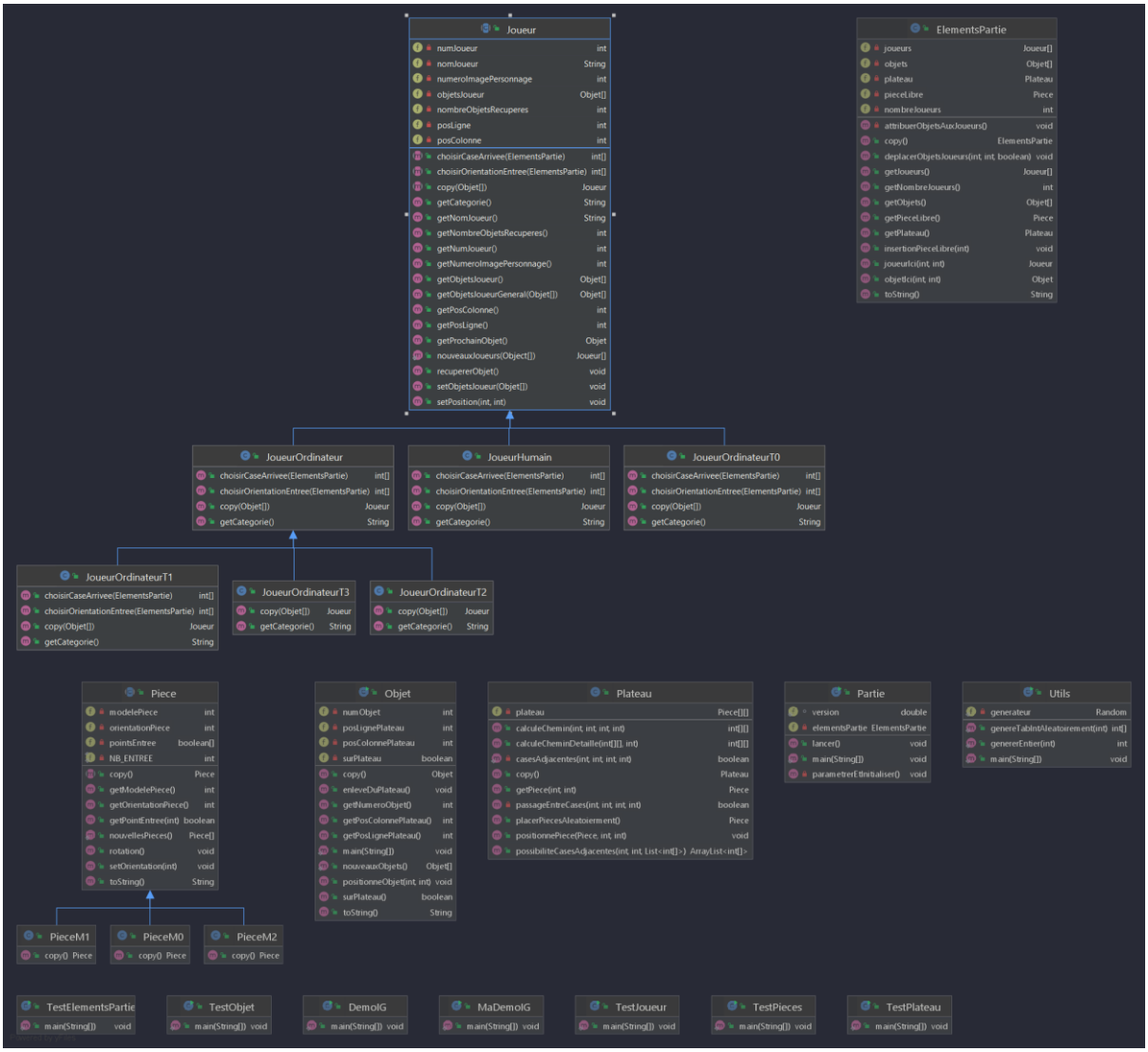
On a vu, durant ce module POO, des méthodes que nous avons réutilisées dans ce projet du second semestre. Comme par exemple, les méthodes *«equals » et « toString ».*

b) COO : Conception orientée objet

La Conception orientée objet (COO) est un module qu'on a rencontré durant ce second semestre. Néanmoins, nous l'avons commencé au début de la deuxième partie de ce semestre. La Conception orientée objet consiste en la modélisation graphique d'un logiciel pour mieux le comprendre ou encore pour mieux le

construire. C'est comme une simplification de la réalité, une vue subjective. Le langage le plus utilisé en Conception orientée objet est l'UML.

Pour ce projet, nous avons utilisé une **génération automatique de diagrammes de classes Java** proposée par les différents IDE (IntelliJ ou encore Eclipse). Un diagramme de classes est une représentation de la structure interne du logiciel et qui représente aussi l'état du logiciel (c'est-à-dire les objets et leurs relations).



Exemple de diagramme généré au cours du projet

On a vu aussi durant ce module l'utilisation de GitHub, qui nous a beaucoup aidé pendant le projet pour mettre en commun nos différents codes.

4) Répartition du travail entre les membres du groupe

On a procédé à la répartition du travail suivante :

- Louis (chef de projet) :
  - S'occuper du rendu des étapes
  - Création des diagrammes de classes
  - Génération de la JavaDoc
  - Récupération des codes sur GitHub
  - Création des sprites (voir annexe 1)
- Lucas :
  - Programmation
  - Gérer le GitHub
  - Tester les différents codes
  - Génération de la JavaDoc
- Cameron :
  - Programmation
  - Création de sprites
  - GitHub
- Alexis :
  - Programmation
  - GitHub

5) Retour d'expérience

Au niveau global, ce projet avait un sujet plus intéressant avec un jeu qui, à la base, n'est pas évident. Ce qui le rendait compliqué à coder à première vue. Nous avons donc utilisé le langage de programmation Java dans la totalité de ce projet, ce qui est bien car c'est un langage apprécié par la plupart des membres du groupe. Les IDE ainsi que Paint.net sont des outils que nous avons beaucoup utilisés durant ce projet.

Néanmoins, certaines méthodes (comme calculeChemin par exemple) étaient complexes à résoudre. Mais nous avons, tout de même, réussi à s'en sortir.

Les différents retours d'expérience :

- Lucas : « Le second projet de cette année m'a particulièrement intéressé étant donné le langage imposé. J'ai pu découvrir de nouvelles façons d'implémenter certaines fonctions, mais aussi de pouvoir découvrir de nouvelles fonctionnalités sur Java. J'ai pu, à l'aide de recherches, terminer les différentes fonctions que l'on nous a imposées. »
- Cameron : « Ce projet m'a permis d'améliorer mon travail d'équipe. En effet dans une équipe de 4 il faut savoir se répartir les tâches. Je me suis amélioré sur Git, et j'ai appris à résoudre les problèmes en découpant élément par élément comme pour la méthode calculeChemi. J'ai apprécié travailler avec mes camarades sur ce projet. »
- Alexis : « Le projet m'a permis de m'améliorer en Java notamment en me confrontant à des méthodes complexes qui m'ont fait réfléchir, comme celles de la phase 5 (insertionPieceLibre). Il m'a aussi permis de mieux comprendre et de mieux me servir de Git, ainsi que de renforcer mon travail d'équipe et ma coordination. »

- Louis : « Le projet du second semestre fut, pour moi, une expérience de plus. En tant que chef de projet, j'ai pu donner de nombreuses directives pour rendre un travail plus que correct. Je me suis perfectionné dans l'utilisation d'un IDE et de GitHub. Cela m'a permis aussi d'améliorer mon travail d'équipe avec les différents membres de ce projet. »

6) Conclusion

Pour conclure, ce projet fut une bonne expérience pour la totalité du groupe. Cela nous a beaucoup aidé à nous améliorer par exemple : une meilleure compréhension du langage Java ou encore l'utilisation de GitHub. Certaines méthodes étaient complexes mais nous avons réussi à les résoudre. Ce projet nous a aussi permis d'utiliser les modules COO et POO. Pour finir, nous avons réussi à bien répartir le travail par rapport aux différentes qualités de chaque membre.

Annexe 1 : les sprites

Annexe 2 : Algorithme de tirage aléatoire de valeurs entières

```java
public static int[] genereTabIntAleatoirement(int longTab) {
    int tab[] = new int[longTab];

    ArrayList<Integer> exclude = new ArrayList<>();
    for (int i = 0; i < longTab; i++) {
        int x = genererEntier(longTab);
        while (exclude.contains(x)) {
            x = genererEntier(longTab);
        }
        exclude.add(x);
        tab[i] = x;
    }

    return tab;
}
```

Annexe 3 : Algorithme de recherche de chemins

```
public int[][] calculeChemin(int posLigCaseDep, int posColCaseDep, int posLigCaseArr, int posColCaseArr) {
    if (posLigCaseDep == posLigCaseArr && posColCaseDep == posColCaseArr) {
        return new int[][]{new int[]{posLigCaseArr, posColCaseArr}};
    }
    List<int[]> listExecution = new ArrayList<>();
    List<int[]> exclusion = new ArrayList<>();
    HashMap<List<Integer>, int[][]> cheminsMap = new HashMap<>();

    listExecution.add(new int[]{posLigCaseDep, posColCaseDep});
    boolean cheminPossible = true;

    while (cheminPossible) {
        for (int[] ints : new ArrayList<>(listExecution)) {
            exclusion.add(new int[]{ints[0], ints[1]});
            ArrayList<int[]> possibilites = possibiliteCasesAdjacentes(ints[0], ints[1], exclusion);
            possibilites.forEach(possible -> {
                if (exclusion.stream().noneMatch(x -> x[0] == possible[0] && x[1] == possible[1])) {
                    exclusion.add(possible);
                }
            });
            listExecution.addAll(possibilites);
            listExecution.removeIf(remove -> remove[0] == ints[0] && remove[1] == ints[1]);

            if (cheminsMap.size() == 0) {
                cheminsMap.put(Arrays.asList(ints[0], ints[1]), new int[][]{new int[]{ints[0], ints[1]}});
            } else {
                ArrayList<int[]> cheminsAvant = possibiliteCasesAdjacentes(ints[0], ints[1], exclude: null);
                ArrayList<int[][]> cheminsTemp = new ArrayList<>();
                cheminsAvant.forEach(chemin -> {
                    if (cheminsMap.containsKey(Arrays.asList(chemin[0], chemin[1]))) {
```

```
                        cheminsTemp.add(new int[][]{chemin});
                        int[][] test = cheminsMap.get(Arrays.asList(cheminsTemp.get(0)[0][0], cheminsTemp.get(0)[0][1]));
                        int[][] test2 = new int[test.length + 1][2];
                        for (int i = 0; i < test2.length; i++) {
                            if (i >= test.length) {
                                test2[i][0] = ints[0];
                                test2[i][1] = ints[1];
                            } else {
                                test2[i][0] = test[i][0];
                                test2[i][1] = test[i][1];
                            }
                        }
                        cheminsMap.put(Arrays.asList(ints[0], ints[1]), test2);
                    }
                });
                if (ints[0] == posLigCaseArr && ints[1] == posColCaseArr) {
                    int[][] cheminFinal = new int[0][];
                    for (Map.Entry<List<Integer>, int[][]> entry : cheminsMap.entrySet()) {
                        if (entry.getValue()[entry.getValue().length - 1][0] == posLigCaseArr && entry.getValue()[entry.getValue().length - 1]
                            if (cheminFinal.length <= entry.getValue().length) {
                                cheminFinal = entry.getValue();
                            }
                        }
                    }
                    return cheminFinal;
                }
            }
        }
```

```
        if (listExecution.size() == 0) {
            cheminPossible = false;
        }
    }
    return null;
}
```

Annexe 4 : Algorithme de la boucle principale

```java
public void lancer() {
    String[] messageOrientation = {
            "",
            "%joueur% : Vous pouvez orienter",
            "la pièce hors du plateau",
            "Ensuite selectionnez une flèche"
    };

    String[] messageChoixChemin = {
            "",
            "%joueur% : Choisissez un",
            "chemin"
    };
    int intJoueurs = 0;

    while (true) {
        //On récupère le joueur
        Joueur joueur = this.elementsPartie.getJoueurs()[intJoueurs];
        IG.changerObjetSelectionne(joueur.getProchainObjet().getNumeroObjet());

        //On modifie %joueur% par le nom du joueur actuel
        String[] modifMessage = messageOrientation.clone();
        modifMessage[1] = modifMessage[1].replace( target: "%joueur%", joueur.getNomJoueur());

        String[] modifMessage2 = messageChoixChemin.clone();
        modifMessage2[1] = modifMessage2[1].replace( target: "%joueur%", joueur.getNomJoueur());

        IG.afficherMessage(modifMessage);
        IG.changerJoueurSelectionne(intJoueurs);
        IG.miseAJourAffichage();
```

```java
        int[] choixOrientation = joueur.choisirOrientationEntree(elementsPartie);
        elementsPartie.insertionPieceLibre(choixOrientation[1]);
        IG.afficherMessage(modifMessage2);
        IG.miseAJourAffichage();

        //On oblige le joueur a choisir un chemin possible
        int[] caseTmp;
        while (true) {
            caseTmp = joueur.choisirCaseArrivee(elementsPartie);
            if (caseTmp == null) {
                caseTmp = new int[]{joueur.getPosLigne(), joueur.getPosColonne()};
            }
            int[][] chemin = elementsPartie.getPlateau().calculeChemin(joueur.getPosLigne(), joueur.getPosColonne(), caseTmp[0], caseTmp[1]);
            if (chemin != null) {
                for (int[] ints : chemin) {
                    IG.miseAJourAffichage();
                    IG.placerJoueurSurPlateau(joueur.getNumJoueur(), ints[0], ints[1]);
                    joueur.setPosition(ints[0], ints[1]);
                    IG.pause( duree: 150);
                }
                break;
            }
        }

        Objet objet = elementsPartie.objetIci(caseTmp[0], caseTmp[1]);
        if (objet != null && joueur.getProchainObjet() != null) {
            if (joueur.getProchainObjet().equals(objet)) {
                if (objet.surPlateau()) {
                    IG.changerObjetJoueurAvecTransparence(joueur.getNumJoueur(), joueur.getProchainObjet().getNumeroObjet(), joueur.getNombreO
                    objet.enleveDuPlateau();
                    joueur.recupererObjet();
                }
```

```
        IG.miseAJourAffichage();

        if (joueur.getNombreObjetsRecuperes() == (18 / elementsPartie.getNombreJoueurs())) {
            IG.afficherGagnant(joueur.getNumJoueur());
            String[] messageGagnant = {
                    "",
                    "%joueur% : Félicitation !",
                    "Vous avez gagné ! :o"
            };
            modifMessage2 = messageGagnant.clone();
            modifMessage2[1] = modifMessage2[1].replace( target: "%joueur%", joueur.getNomJoueur());
            IG.afficherMessage(modifMessage2);
            IG.miseAJourAffichage();
            break;
        }
        intJoueurs = (intJoueurs + 1) % elementsPartie.getNombreJoueurs();
    }
}
```