2022 -2023

Programmation III : JAVA HelbAquarium

Implémentation d'un jeu de simulation d'aquarium

M.RIGGIO JONATHAN

KIZILIRMAK Enes

Tables des matières

Introduction

Dans le cadre du cours de Programmation Java III, nous devions mettre en place un jeu de simulation d'Aquarium en utilisant Swing qui est une bibliothèque graphique en Java. Le jeu est constitué de plusieurs types de poissons ayant leurs propres comportements. Mais aussi, des éléments fixes comme les insectes, les pastilles et les décors.

Tout d'abord, pour ce qui était de la réalisation du projet. J'ai utilisé une machine virtuelle Ubuntu disponible sur le réseau de l'école. Et pour pouvoir développer mon projet j'ai utilisé l'IDE d'Eclipse.

Deuxièmement, durant la réalisation du projet nous avions à notre disposition quelques outils très utiles. Par exemple, nous avions en notre possession trois vidéos sous forme de tutoriel posté par notre professeur M. Riggio. Ces vidéos nous ont permis de créer une ''base fonctionnelle'' pour pouvoir ensuite l'adapter et continuer pour le jeu d'Aquarium. Le jeu devait absolument respecter les principes de designs orientés objet comme vus au cours.

Pour finir, Durant ce rapport il y aura les fonctionnalités de base et les fonctionnalités supplémentaires du projet. Puis il y aura un schéma conceptuel qui permettra d'avoir une vue d'ensemble sur la structure du code. Après, on abordera les difficultés rencontrés et les limitations. Pour finir il y a la conclusion et la bibliographie.

Bonne lecture :)

Présentation des éléments graphique

Avant de passer aux fonctionnalités du jeu, voici une présentation des éléments du jeu.

Board :

Voici a quoi ressemble globalement l'interface graphique de mon jeu. Ce dernier est constitué de plusieurs éléments fixes et non fixes.

Eléments non-fixes :

Voici les quatre espèces de poissons. chacun de ces poissons a un comportement spécifique. Les poissons orange se déplacent aléatoirement.

 *FishOrange (hasardeux)*

 *FishPurple (prudent)*

 *FishRed (prédateurs)*

 *FishBlue (sociaux)*

Eléments fixes :

Spider (4 sec Bonus de Vitesse)

Butterfly (6 sec Bonus de Vitesse)

Ladybirds (10 sec Bonus de Vitesse)

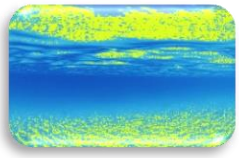
Pellet (Arrête tous les poissons sauf les poissons du même type)

Décoration (Un obstacle pour les poissons)

Température (Background image) :

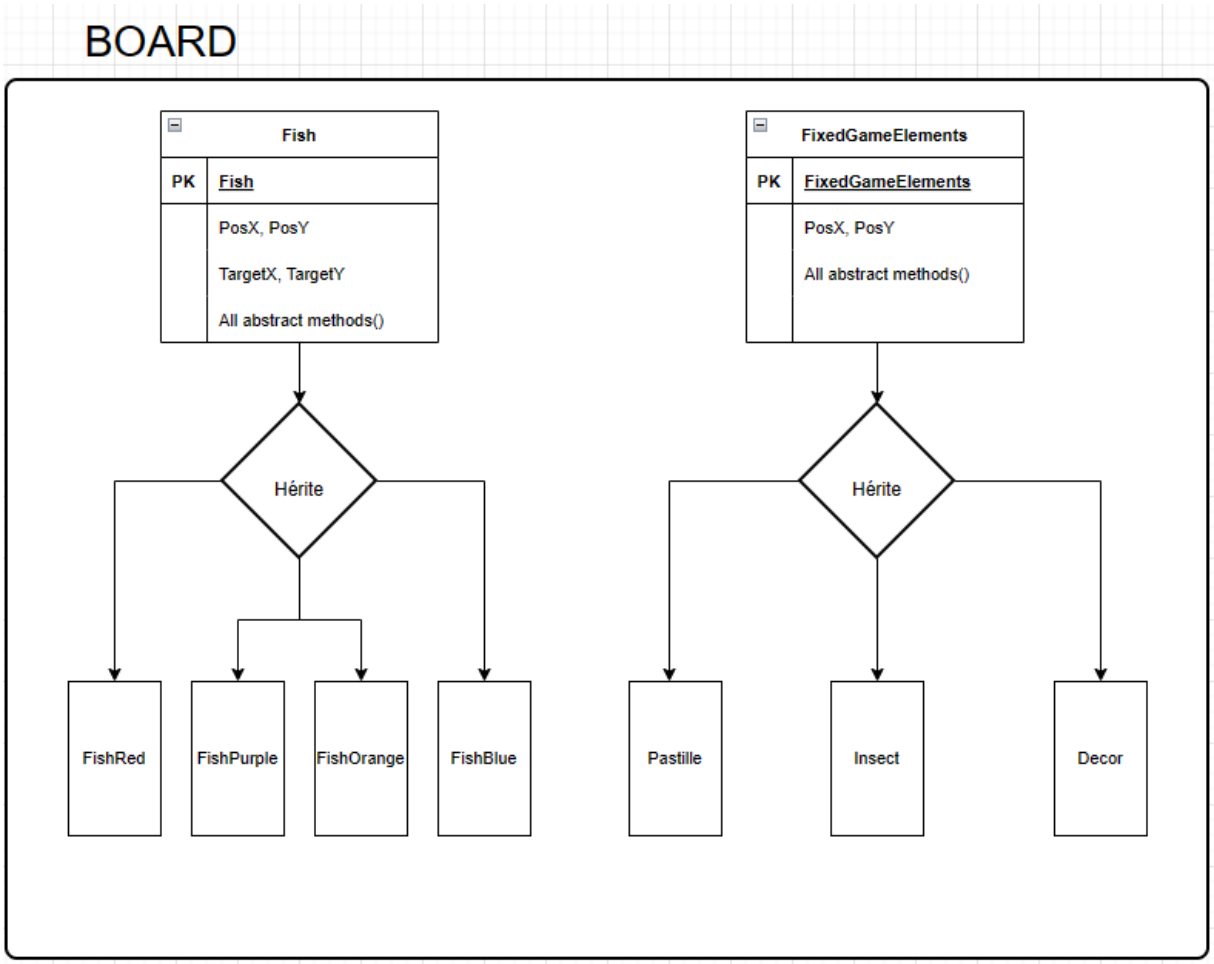
DefaultBackground

ColdBackground (Malus Vitesse FishRed)
PS : le blanc c'est pour représenter la glace



HotBackground (Bonus Vitesse FishRed)
PS : le jaune c'est pour représenter les rayons de soleil

Schéma



Les Fish héritent tous de la classe mère Fish. Pareil pour les éléments fixes, ils héritent tous de FixedGameElements.

Les deux classes mères sont des classes abstraites, donc toutes les méthodes hérités de la classe mère doivent être redéfinis. (@Override).

Les classes filles de Fish héritent des position X et Y, et des Target X et Y. Tandis que, les éléments fixes n'ont pas de Target ce qui est logique vu qu'ils ne se déplacent pas.

Fonctionnalités de base

Les espèces de poissons

Chaque poisson se déplace dans l'Aquarium verticalement, horizontalement et diagonalement. Parmi les poissons il y a des comportements qui divergent.

FishOrange :

Le FishOrange a un comportement de base, il se déplace aléatoirement dans l'Aquarium.

FishRed :

Le poisson rouge mange tous les autres poissons qui non-rouge :

```
private void checkIfFishRedEat() {          // RED FISH EAT OTHERS FISHES !

    myFishListCopy = new ArrayList<Fish>();     // A COPY OF MY LIST TO VERIFY CONDITIONS
    myFishListCopy.addAll(myFishList);

    for(Fish f1 : myFishListCopy) {
        for(Fish f2 : myFishListCopy) {

            if(f1.getType() == "fishRed" && f2.getType() != "fishRed") {
                if (f1.getPosX() == f2.getPosX() && f1.getPosY() == f2.getPosY()) {

                    f2.setPosX(voidPosition);    // OUT OF AQUARIUM
                    myFishList.remove(f2);

                    System.out.println(f1.getType() + " EATED "  + f2.getType());
                }
```

La vitesse du poisson rouge dépend du décor :

```
// >>>>>>>>>>>>>>>[FISH RED SPEED ]<<<<<<<<<<<<<<<

if(board.temperatureNumber == 1) // 1 == COLD TEMPERATURE
    setSpeed(defaultSpeed-malusSpeed);
if(board.temperatureNumber == 2) // 2 == DEFAULT TEMPERATURE
    setSpeed(defaultSpeed);
if(board.temperatureNumber == 3) // 3 == HOT TEMPERATURE
    setSpeed(defaultSpeed+BonusSpeed);
```

Le poisson rouge se déplace toujours vers le poisson non-rouge, je n'ai pas réussi à faire déplacer vers l'éléments le plus proche.

```
// >>>>>>>>>>>>>>>[FISH RED MOVE BEHAVIOR, (PREDATOR FISH) ]<<<<<<<<<<<<<<<

for(Fish f1: board.myFishList) {
    for(Fish f2: board.myFishList) {

        if(f1.getType() == "fishRed" && f2.getType() != "fishRed") {
            if (f2.getPosX() >= 0 && f2.getPosY() >= 0 && f2.getPosX() < board.B_WIDTH && f2.getPosY() < board.B_HEIGHT) {

                board.CheckIfPositionIsValid(f1.getTargetX(), f1.getTargetY());

                f1.setTargetX(f2.getPosX());
                f1.setTargetY(f2.getPosY());
```

FishPurple :

La vitesse du FishPurple augmente avec le nombre de décor dans le jeu

```
@Override
protected void move(Board board) {

    super.move(board);

    // >>>>>>>>>>>>>>>[FISH RED SPEED ]<<<<<<<<<<<<<<<

    int speedByDecorNumber = Decor.getDecorCounter();
    int fishPurpleSpeed = defaultSpeed + speedByDecorNumber;
    setSpeed(fishPurpleSpeed);

    setSpeed(defaultSpeed);
```

Je n'ai pas mis de comportement spécifique car, selon moi vu que le FishRed le suit constamment il se déplace toujours dans la direction opposé du FishRed (c'est mon interprétation personnelle).

FishBlue :

Le FishBlue a une vitesse de base supérieure aux autres poissons

```
// >>>>>>>>>>>>>>>[FISH BLUE SPEED  ]<<<<<<<<<<<<<<<

    final int fishBlueDefaultSpeedBonus=0;
    setSpeed(defaultSpeed + fishBlueDefaultSpeedBonus);
```

```
// >>>>>>>>>>>>>>>[FISH BLUE MOVE BEHAVIOR, (SOCIABLE FISH)  ]<<<<<<<<<<<<<<<
for(Fish f1: board.myFishList) {
    for(Fish f2: board.myFishList) {

        if(f1.getType() == "fishBlue" && f2.getType() == "fishBlue" || f1.getType() == "fishBlue" && f2.getType() == "fishPurple" ) {
            if (f2.getPosX() >= 0 && f2.getPosY() >= 0 && f2.getPosX() < board.B_WIDTH && f2.getPosY() < board.B_HEIGHT) {

                f1.setTargetX(f2.getTargetX());
                f1.setTargetY(f2.getTargetY());
```

Les insectes et pastilles

Dans mon jeu il y a actuellement 3 types d'**insectes** et des **pastilles** mais j'ai décidé d'enlever leur comportement qui permet d'avoir un boost de vitesse pour ce dépôt car, il y avait quelques problèmes.

En ce qui concerne le **mode Insectivore** et le **mode pastille**, c'est pareil ils fonctionnent mais je n'ai pas encore fait en sorte que les poissons se dirigent vers l'élément le plus proche. (Ils se dirigent vers le dernier élément de la liste.

Pas totalement fini ! (sera fait pour le deuxième dépôt)

Les décorations

L'aquarium contient un certain nombre aléatoire des décorations. Pour ce faire j'ai décidé de travailler avec deux variables **finalDecorWidth** et **finalDecorHeight.**

```
private final static int finalDecorWidth=60;        // THE WIDTH OF THE IMAGE IS 60 pixels
private final static int finalDecorHeight=120;     //  THE HEIGHT OF THE IMAGE IS 120 pixels
```

J'ai décidé de créer deux **Getter** pour pouvoir utiliser ces deux valeurs dans mon board. Evidemment, je ne crée pas de **Setter** car, la taille de l'image ne change pas. C'est pourquoi mes deux variables sont en **final**.

```
protected static int getfinalDecorWidth() {

    return finalDecorWidth;    // RETURN FINAL SIZE X OF THE DECOR
}

protected static int getfinalDecorHeight() {

    return finalDecorHeight;    // RETURN FINAL SIZE Y OF THE DECOR
}
```

Et j'ai décidé de les mettre en **static** pour pouvoir faire appel à ces méthodes directement sans créer d'instance.

J'ai créé une méthode **CheckIfPositionEqualToDecor**(**int**, **int**) dans mon board qui vérifie si la position des éléments du jeu est la même que la position du décor. Si oui elle retourne un Boolean true. Sinon il retourne false.

```
protected boolean CheckIfPositionEqualToDecor(int paramX, int paramY) {

    for (Decor d : myDecorList) {

        if ((paramX >= d.getPosX())
                && paramX <= d.getPosX() + Decor.getfinalDecorWidth()
                && paramY >= d.getPosY()
                && paramY <= d.getPosY() + Decor.getfinalDecorHeight()) {

            return true;
        }

    }
    return false;
}
```

Vitesse dans le jeu

Tous mes poissons ont vitesse de base appelé **defaultSpeed**. Sauf évidemment, le **FishBlue** qui est un peu plus rapide que le reste. Pour ce faire, dans ma classe mère Fish j'ai affecté la valeur de **defaultSpeed** dans ma variable **speed**. Pour que toutes mes classes filles héritent de la même vitesse.

```
public abstract class Fish{

    protected int defaultSpeed=3;         // DEFAULT SPEED
    protected int speed=defaultSpeed;    // FISHES SPEED IN GAME !

    protected final int MaxSpeed = defaultSpeed*2;      // MAXIMAL FISH SPEED
```

Dans la classe fille **FishBlue** j'ai **Override** la méthode **move()** pour mettre la **speed** par défaut du **FishBlue** plus élevé.

```
@Override
protected void move(Board board) {

    super.move(board);

    int fishBlueSpeed=defaultSpeed+2;
    setSpeed(fishBlueSpeed);     // FISH BLUE'S DEFAULT SPEED

}
```

Reproduction

Pour le checkFishReproduction(), je vérifie si c'est le même type de poisson, s'ils ont la même position mais que ce n'est pas **LE** même poisson et uniquement s'ils sont dans l'Aquarium.

```
private void checkFishReproduction() {

    myFishListCopy = new ArrayList<Fish>();     // A COPY OF MY LIST TO VERIFY CONDITIONS
    myFishListCopy.addAll(myFishList);

    for (Fish f1: myFishListCopy) {
        for (Fish f2: myFishListCopy) {
            if(f1 != f2) {
                if (f1.getPosX() == f2.getPosX() && f1.getPosY() == f2.getPosY()) {
                    if (f1.getPosX() >= 0 && f1.getPosY() >= 0 && f1.getPosX() < B_WIDTH && f1.getPosY() < B_HEIGHT) {
                        if (f2.getPosX() >= 0 && f2.getPosY() >= 0 && f2.getPosX() < B_WIDTH && f2.getPosY() < B_HEIGHT) {


                            if(AlgorithmForBornProbability() == true) {  // IF TRUE

                                // DO REPRODUCTION ACTION
                                f1.setPosX(voidPosition);
                                f2.setPosX(voidPosition);

                                if(f1.getPosX() == voidPosition && f2.getPosX() == voidPosition) {
                                    myFishList.remove(f1);      // REMOVE FROM LIST
                                    myFishList.remove(f2);      // REMOVE FROM LIST
                                }

                                addThreeFishInGame(f1, f2);
```

Si c'est condition sont remplies alors on appelle la méthode AlgorithmForBornProbability(), cette algorithme retourne un booléen. Plus il y a de **poisson** dans l'Aquarium plus la probabilité que la méthode renvoi **vrai** est basse.

```
private boolean AlgorithmForBornProbability() {

    boolean res = false;
    Random rand = new Random();

    int ListSize = myFishList.size()*10;    // If THERE IS SO MANY FISH THE PROBABILITY IS BIGGER !

    boolean val = rand.nextInt(ListSize)==0;   // MORE FISH (listSize bigger) PROBABILTY TON RETURN TRUE IS LOW !

    if (val == true )
        res = true; // REPRODUCTION IS OK ! RETURN TRUE;

    return res;     // FALSE IF VAL NOT FIND, TRUE IF FIND !
}
```

Si la méthode AlgorithmForBornProbability() retourne **vrai** alors la méthode addThreeFishInGame() est appelé cette méthode ajoute **3 Fish** du même **type** qui se sont reproduit.

```
private void addThreeFishInGame(Fish f1, Fish f2) {

    final int NumberOfFishToAdd = 3;

    if (f1.getType() == "fishOrange" && f2.getType() == "fishOrange") {     //IF IT'S A ORANGE FISH
        for (int i=0; i< NumberOfFishToAdd ; i++)
            myFishList.add(new FishOrange(this)); // ADD 3 FISHES ORANGE!
    }
    if (f1.getType() == "fishRed" && f2.getType() == "fishRed") {       //IF IT'S A ORANGE FISH
        for (int i=0; i< NumberOfFishToAdd ; i++)
            myFishList.add(new FishRed(this)); // ADD 3 FISHES RED !
    }
    if (f1.getType() == "fishPurple" && f2.getType() == "fishPurple") {     //IF IT'S A ORANGE FISH
        for (int i=0; i< NumberOfFishToAdd ; i++)
            myFishList.add(new FishPurple(this)); // ADD 3 FISHES PURPLE !
    }
    if (f1.getType() == "fishBlue" && f2.getType() == "fishBlue") {     //IF IT'S A ORANGE FISH
        for (int i=0; i< NumberOfFishToAdd ; i++)
            myFishList.add(new FishBlue(this)); // ADD 3 FISHES BLUE!
    }
}
```

Fonctionnalités supplémentaires

1) Performance (Condition de crash)

Lorsqu'il y a beaucoup d'éléments dans le jeu, mon jeu commençait à énormément lagé, j'ai donc décidé de comparer l'utilisation de mon processeur et de ma mémoire vive. Et effectivement il y avait une grande différence.

Avant (Peu d'élément dans le jeu) :

Après (Beaucoup d'éléments dans le jeu) :

J'ai donc trouvé une solution à cela. La solution c'est que **si** le nombre de tous les éléments est supérieur à 100. J'appelle ma fonction **getGameCrashed()**.

```
if( myFishList.size() + myDecorList.size() + myEatableElementList.size() > 100) {
    inGame=false;
}
```

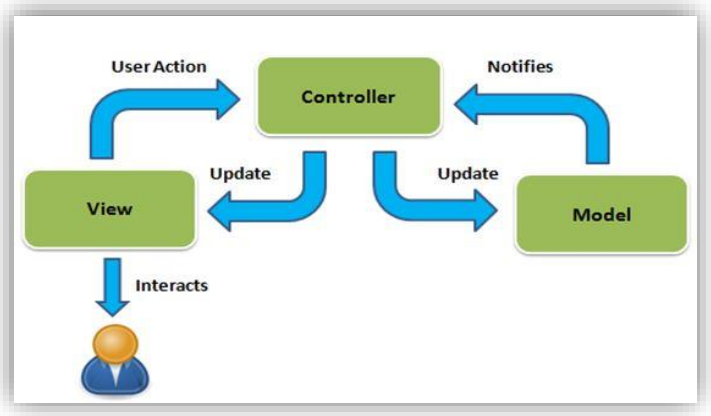
Si **inGame** = false, dans la méthode **doDrawing()** on appelle la fonction **getGameCrashed()** qui permet d'afficher le message.

```
private void getGameCrashed(Graphics g) {

    setBackground(Color.black);

    String msg = "THE GAME CRASHED BECAUSE YOU CREATED SO MANY ELEMENTS ! ";
    Font small = new Font("LucidaSans", Font.BOLD, 26);
    FontMetrics metr = getFontMetrics(small);

    g.setColor(Color.red);
    g.setFont(small);
    g.drawString(msg, (B_WIDTH - metr.stringWidth(msg)) / 2, B_HEIGHT / 2);

}
```

<u>Résultat message :</u>

2) TAdapter dans une autre classe que le Board.

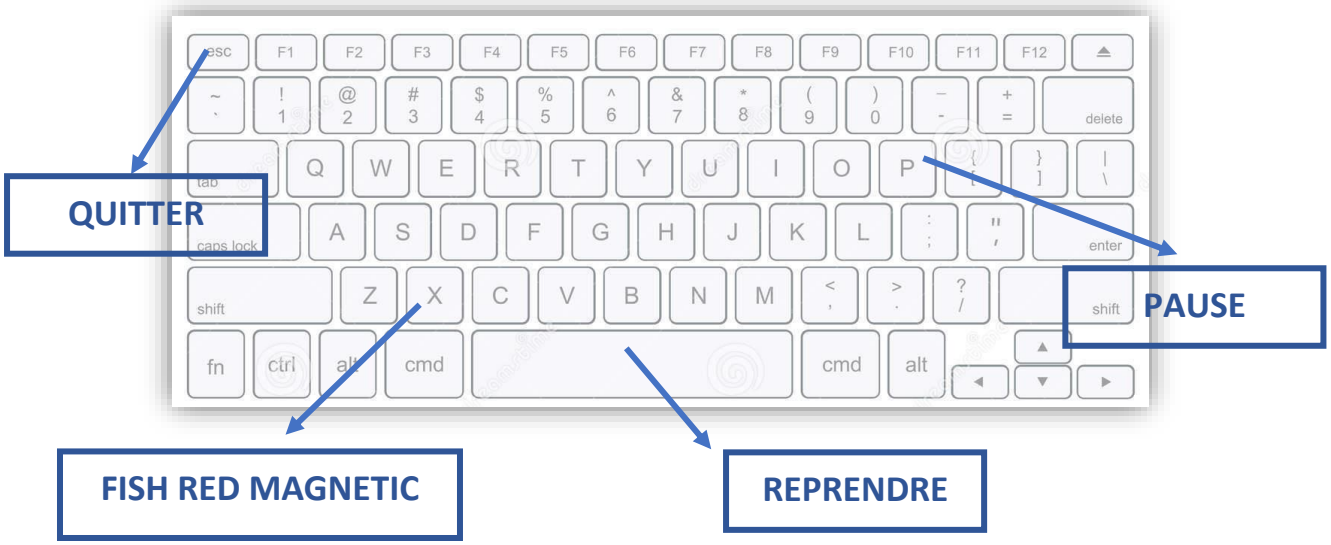
En ce qui concerne les **Key Event,** j'ai décidé de le mettre dans une classe à part pour ce faire j'ai lu la doc en ligne d'Oracle. La source se trouve dans la bibliographie. Le but c'était de libérer un peu de code de mon board. Et aussi de respecter d'avantages le modèle **MVC** (Modèle-vue-contrôleur).



L'utilisateur fait une action (**Key Event**) cette information est traitée et par rapport à la touche appuyé le **Board** qui est le **contrôleur** va envoyer des résultats. Mais mon cas à moi le modèle **MVC** n'est pas totalement respecté car, le **Board** joue le rôle à la fois du **contrôleur** et à la fois du **View**. Et le **model** on peut dire que ce sont les classes Fish, Insectes, Decor, etc.

Keyboard keys (BONUS KEYS) :

Voici la liste des touches supplémentaires :



3) Background music

J'ai également ajouté une musique en arrière-plan pour rendre le jeu plus vivant. J'ai donc effectué des recherches sur Google et je suis tombé sur ce un résultat intéressant sur Stack Overflow la source se trouve dans la bibliographie. J'ai dû télécharger une musique en MP3 puis la convertir en .wav Evidemment, la musique utilisée dans mon projet est aussi citée dans la bibliographie.

Pour ce faire j'ai d'abord importé ce qui était utile dans la classe HelbAquarium où se trouve le Main() du jeu.

HelbAquarium.java

```
5       // IMPORTED TO PLAY MUSIC IN BACKGROUND !
6       import java.io.File;
7       import javax.sound.sampled.*;
8       import java.io.IOException;
9
```

Et puis dans le Main(),

```
public static void main(String[] args) throws UnsupportedAudioFileException, IOException, LineUnavailableException {

    File file = new File("BackgroundMusic.wav");
    AudioInputStream audioStream = AudioSystem.getAudioInputStream(file);
    Clip clip = AudioSystem.getClip();

    clip.open(audioStream);
    clip.start();
```

On créer un objet de type file dans lequel je spécifie le nom du fichier.wav

4) Mode Magnetic for Fish Red

Comme fonctionnalité supplémentaire j'ai ajouté une un mode magnétique pour le FishRed. Lorsque le bouton 'X' est maintenu. Le poisson rouge attire vers lui tous les autres poissons qui ne sont pas rouge.

```
// >>>>>>>>>>>>>>|FISH RED MAGNETIC MODE !|<<<<<<<<<<<<<<<
if ((e.getKeyCode() == 88 )) {  // HOLD THE KEY --> (X), (IL FAUT MAINTENIR LA TOUCHE --> (X) ) !
    System.out.println(x: "----->>>>>>>>>>>>>>>|FISH RED MAGNETIC MODE IS ON !!!|<<<<<<<<<<----- ");

    if (f.getType() == "fishRed" && f2.getType() != "fishRed") {            // NOT RED FISH

        f2.setTargetX(f.getPosX());
        f2.setTargetY(f.getPosY());

    }
}
```

Difficultés rencontrées

1) La plus grosse difficultés que j'ai rencontrés a été la reproduction des poissons ce que j'essayait de faire était quasi pareil sauf que je bouclais avec ma propre Liste de Fish.

   Après des heures et des heures de réflexion j'ai compris qu'en fait si je boucle avec ma propre liste et que j'ajoute des nouveaux poissons. Ma liste n'arrête pas de grandir et donc == Boucle infinie.

```
private void checkFishReproduction() {

    myFishListCopy = new ArrayList<Fish>();     // A COPY OF MY LIST TO VERIFY CONDITIONS
    myFishListCopy.addAll(myFishList);

    for (Fish f1: myFishListCopy) {
        for (Fish f2: myFishListCopy) {
```

2) Lorsque mes poissons mangent un éléments fixes du jeu. Ils gagnent en vitesse mais ils s'arrêtent soudainement de bouger dans le board puis se redéplace. (J'ai donc décidé de commenter cette partie).

```
for (Fish f : board.myFishList) {
    for (FixedGameElement elem : board.myEatableElementList) {

        if ((f.getPosX() == elem.getPosX()) && (f.getPosY() == elem.getPosY())) {

            elem.setPosX(board.voidPosition);

            if (elem.getType() == "spider") {   // IF ELEM IS A SPIDER BONUS 4 SECONDS
                // f.setSpeed(defaultSpeed + bonusSpeedFinal );
                // Thread.sleep(4000);
                // f.setSpeed(defaultSpeed);

            }

            if (elem.getType() == "ladybirds") {    // IF ELEM IS A SPIDER BONUS 6 SECONDS
                // f.setSpeed(defaultSpeed + bonusSpeedFinal );
                // Thread.sleep(4000);
                // f.setSpeed(defaultSpeed);
            }

            if (elem.getType() == "butterfly") {// IF ELEM IS A BUTTERFLY BONUS 10 SECONDS
                // f.setSpeed(defaultSpeed + bonusSpeedFinal );
                // Thread.sleep(4000);
                // f.setSpeed(defaultSpeed);
            }

            if (elem.getType() == "pellet") {   // IF ELEM IS A PELLET STOP ALL OTHERS FISHES 10 SECONDS !
                f.fishCanMove = false;  // IF EAT PELLET HE STOPs !
            }
```

Limitations

1) Un défaut dans mon projet, c'est le fait d'avoir créé trois classes insectes Ladybirds, Spider et Butterfly. Je suis conscient du fait que ce n'est pas la manière la plus efficace de faire de l'orienter objet. Mais, je n'ai pas trouvé d'autre solution qui fonctionnait sans problèmes. Donc j'ai laissé ainsi.

2) Pour ce qui concerne le déplacement vers l'élément le plus proche par rapport un autre éléments du jeu. Je n'ai malheureusement pas trouvé une solution qui fonctionnait à 100%.

3) Le plus gros des limitations selon moi, c'est lorsque mes poissons mangent un éléments fixes du jeu. Ils gagnent en vitesse mais ils s'arrêtent soudainement de bouger dans le board. Donc j'ai décidé de commenter cette fonctionnalité dans mon code.

```
for (Fish f : board.myFishList) {
    for (FixedGameElement elem : board.myEatableElementList) {

        if ((f.getPosX() == elem.getPosX()) && (f.getPosY() == elem.getPosY())) {

            elem.setPosX(board.voidPosition);

            if (elem.getType() == "spider") {   // IF ELEM IS A SPIDER BONUS 4 SECONDS
                // f.setSpeed(defaultSpeed + bonusSpeedFinal );
                // Thread.sleep(4000);
                // f.setSpeed(defaultSpeed);

            }

            if (elem.getType() == "ladybirds") {    // IF ELEM IS A SPIDER BONUS 6 SECONDS
                // f.setSpeed(defaultSpeed + bonusSpeedFinal );
                // Thread.sleep(4000);
                // f.setSpeed(defaultSpeed);
            }

            if (elem.getType() == "butterfly") {// IF ELEM IS A BUTTERFLY BONUS 10 SECONDS
                // f.setSpeed(defaultSpeed + bonusSpeedFinal );
                // Thread.sleep(4000);
                // f.setSpeed(defaultSpeed);
            }

            if (elem.getType() == "pellet") {   // IF ELEM IS A PELLET STOP ALL OTHERS FISHES 10 SECONDS !
                f.fishCanMove = false;  // IF EAT PELLET HE STOPs !
            }
```

Conclusion

Pour conclure ce projet, je trouve que c'était une expérience très enrichissante. Le fait de devoir rendre un projet avant une deadline ça m'a permis de m'améliorer sur beaucoup de points.

Tout d'abord au niveau de mon organisation de travail. Je suis beaucoup plus autonome sur ma manière de travailler. Je fais de meilleures recherches sur Google (je veux dire par là que j'arrive à mieux cibler ce que je veux par rapport à mon code). Je m'intéresse de plus en plus sur le vidéos tutoriels sur l'orienté objets, etc.

Deuxièmement, ça ma permit de m'améliorer de façon général en programmation orienté objet. Maintenant je ne pense pas qu'au fonctionnalités du programme. Je pense aussi à comment je peux factoriser le code et le rendre le plus facile à maintenir en éliminant les constantes magiques et les répétitions.

Si j'avais eu plus de temps, je pense vraiment que j'aurais pu corriger le problème de speed et j'aurai pu avoir la distance le plus proche entre un élément du jeu par rapport à un autre.

J'ai trouvé ce projet très plaisant à réaliser malgré les moments plus difficile.

Bibliographie

1) IT-Point. (2020b, November 9). *Snake To Frogger - Part1 [Video]*. YouTube. https://www.youtube.com/watch?v=f-2_zZp4boo

2) IT-Point. (2020a, November 9). *Snake To Frogger - Part2 [Video]*. YouTube. https://www.youtube.com/watch?v=i0z2Og2W4hg

3) IT-Point. (2020c, November 22). *Snake To Frogger - Application de l'héritage et du polymorphisme* [Video]. YouTube. https://www.youtube.com/watch?v=oH_Wt8A7f0U

4) Oracle. (Java Platform SE 7). (2020, June 24). *Class KeyAdapter* docs.oracle.com.https://docs.oracle.com/javase/7/docs/api/java/awt/event/KeyAdapter.html

5) Les Insectes PNG - *31198 images de Les Insectes transparentes* | PNG gratuit. (freepng.fr). https://www.freepng.fr/telecharger/insect.html

6) Pngtree. *Fresh fish PNG Image and Clipart.* (Pngtree.com). https://pngtree.com/free-animals-png/fish

7) Couleur Jaune, Point PNG - *Jaune, Couleur, Point transparentes* | PNG gratuit. (freepng.fr). https://www.freepng.fr/png-lj7jm4/

8) How can I play sound in Java? (2008, August 25). *Stack Overflow*. https://stackoverflow.com/questions/26305/how-can-i-play-sound-in-java

9) stellarsheik. (2021, August 21). relaxing Pokémon water music + gentle waves [Video]. YouTube. https://www.youtube.com/watch?v=vMx4AS5Phbc