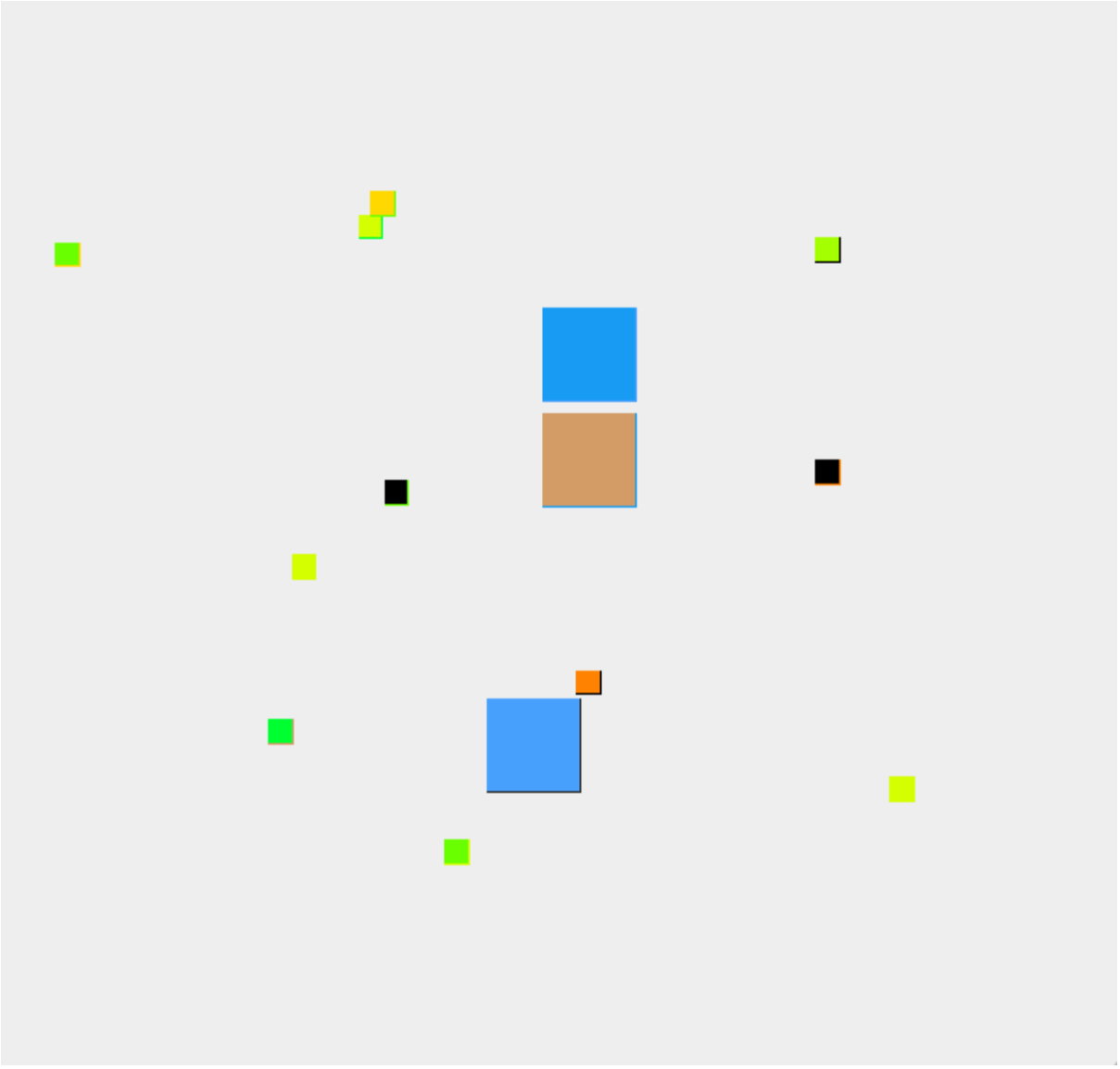
Alix Dufour et Théo Bouguet

Session Automne 2022

TP2
Programmation orientée objet avancée

Question 1

Le sujet consiste à implémenter une simulation d'alimentation de pigeon dans un espace public. Ci-dessous un visuel de notre simulation :



Dans notre simulation, les pigeons sont représentés par les carrés de plus grande taille, dans des couleurs générées aléatoirement et dans une position de départ aléatoire. La nourriture est représentée par des plus petits carrés, dans des couleurs qui vont de rouge à vert qui représentent leur score de fraîcheur respectivement de 10 à 1. Quand la nourriture atteint 0 de fraîcheur elle devient noire et ne sera plus mangée par les pigeons.

Nous verrons ensuite comment est implémenté chacun des points du sujet :

- **Chaque pigeon est contrôlé par un thread**

```
animation=new Thread(this,"Pigeon");
animation.start();
```

En effet, comme vu dans le constructeur de la classe pigeon, un thread est lancé pour déterminer ses déplacements.

- **Si rien ne se passe, les pigeons s'endorment et ne bougent pas**

En effet, la boucle while() qui correspond au thread de déplacements n'est composée que d'instructions if/else : si rien ne se passe, ils ne feront rien.

- **En apercevant de la nourriture, un pigeon se déplace vers la nourriture la plus fraîche**

La fonction checkNourriture() permet en effet de mettre la nourriture la + fraiche de la liste dans l'attribut de destination du pigeon.

```
public boolean checkNourriture() {
    if(p.listNourritures.size() == 0) {
        destN = null;
        return false;
    } else if (destN == null) {
        try {
            destN = p.listNourritures.get(0);
        } catch (IndexOutOfBoundsException e) {
            System.err.println("Caught IndexOutOfBoundsException: " + e.getMessage());
        }
        return true;
    }
    else {
        synchronized (p.listNourritures) {
            for(Nourriture n : p.listNourritures) {
                if (n.fraicheur > destN.fraicheur || !p.listNourritures.contains(destN))
                    destN = n;
            }
        }

        return true;
    }
}
```

A noter que nous avons défini la fraîcheur de la nourriture comme une variable comprise entre 0 et 10, avec 10 la + fraîche. Lorsque la nourriture est créée, elle a une valeur de fraîcheur aléatoire comprise entre 1 et 10, celle-ci baisse au cours du temps et peut ainsi atteindre 0, valeur à laquelle les pigeons refusent de manger la nourriture.

- **Une nourriture fraîche touchée est changée, donc elle doit disparaître immédiatement de la scène**

```java
public synchronized void checkCollision() {

        if(destN != null) {

            double dist = (double) Math.sqrt(Math.pow(destN.getX() - x, 2) + Math.pow(destN.getY() - y, 2));
            //System.out.println(dist);

                // On ne retire la nourriture que si la nourriture est fraiche et que le verrou n'est pas activé
                if (dist < r + destN.getSize() && destN.getFraicheur() > 0) {
                    try {
                        p.listNourritures.remove(destN);
                    } catch (NullPointerException e) {
                        System.err.println("Caught NullPointerException: " + e.getMessage());
                    }catch (ConcurrentModificationException e) {
                        System.err.println("Caught ConcurrentModificationException: " + e.getMessage());
                    }
                }

        }
}
```

En effet la fonction checkCollision() de la classe pigeon permet de vérifier si un pigeon touche une nourriture et si oui, de la supprimer.

- **Si plus qu'un pigeon touche la nourriture simultanément, uniquement un seul pourra la supprimer**

On s'assure qu'un seul pigeon puisse supprimer la nourriture en indiquant que la méthode checkCollision est synchronisée, cela empêche la modification de la liste de nourriture par deux threads pigeons en même temps.

- **Nos pigeons sont gâtés ; un pigeon qui touche une nourriture pas fraîche, il l'ignore**

En effet, si la fraîcheur de la nourriture est à 0 alors il sera ignoré par les pigeons.

- **Même en l'absence de la nourriture, des fois les pigeons se font effrayer et ils se dispersent à des positions aléatoires. Intégrer ce mécanisme dont la probabilité d'occurrence change d'un tour à l'autre.**

A chaque tour, une probabilité de frayeur est calculée et est valable pour tous les pigeons : (Main.java)

```java
// Calcul de la probabilité des pigeons de se faire effrayer pour ce tour-ci
Random r = new Random();
probaFrayeur = r.nextFloat(0.003f);
```

(Main.java)

A chaque boucle du thread d'animation du pigeon, il va utiliser cette valeur pour déterminer si se fera effrayé ce tour-ci. Quand les pigeons sont effrayés ils deviennent rouge (afin de mieux le voir lors des test), prennent une destination aléatoire et leur vitesse est doublée. Une fois leur destination de frayeur atteinte, ils reviennent à leur état normal.

```java
float f = rand.nextFloat(1);

if (f < p.probaFrayeur) {
    frayeur = true;
    destN = null;
    slp = 5;
    fx = rand.nextInt(500);
    fy = rand.nextInt(500);
}
```

Pigeon.java

- **Gérer les exceptions**

Des try-catch ont été ajoutées aux endroits jugés “à risque”, notamment lorsque le programme essaye d’accéder ou de modifier des listes. Par exemple :

```java
try {
    p.listNourritures.remove(destN);
} catch (NullPointerException e) {
    System.err.println("Caught NullPointerException: " + e.getMessage());
}catch (ConcurrentModificationException e) {
    System.err.println("Caught ConcurrentModificationException: " + e.getMessage());
}
```

- **Gérer les modification concurrentes**

Une erreur possible dans ce programme est la modification de la liste de nourritures par plusieurs thread de pigeons en même temps ou lors d’une itération sur la liste, par exemple dans la méthode paint() du panel permettant de mettre à jour l’affichage.

Pour éviter cela, nous utilisons un ArrayList synchronisé, qui nous permet de verrouiller l’accès à la liste lorsque l’on itère dessus. Il faut ajouter un bloc synchronized sur la liste.

Nous avons par exemple dans la méthode paint() :

```java
synchronized (this.listNourritures) {
    for(Nourriture n : listNourritures) {
        n.decFraicheur();
        time.start();
    }
}
```