SIMULATION D'UN ECOSYSTEME

Groupe 2

Sommaire

Rappel du problème

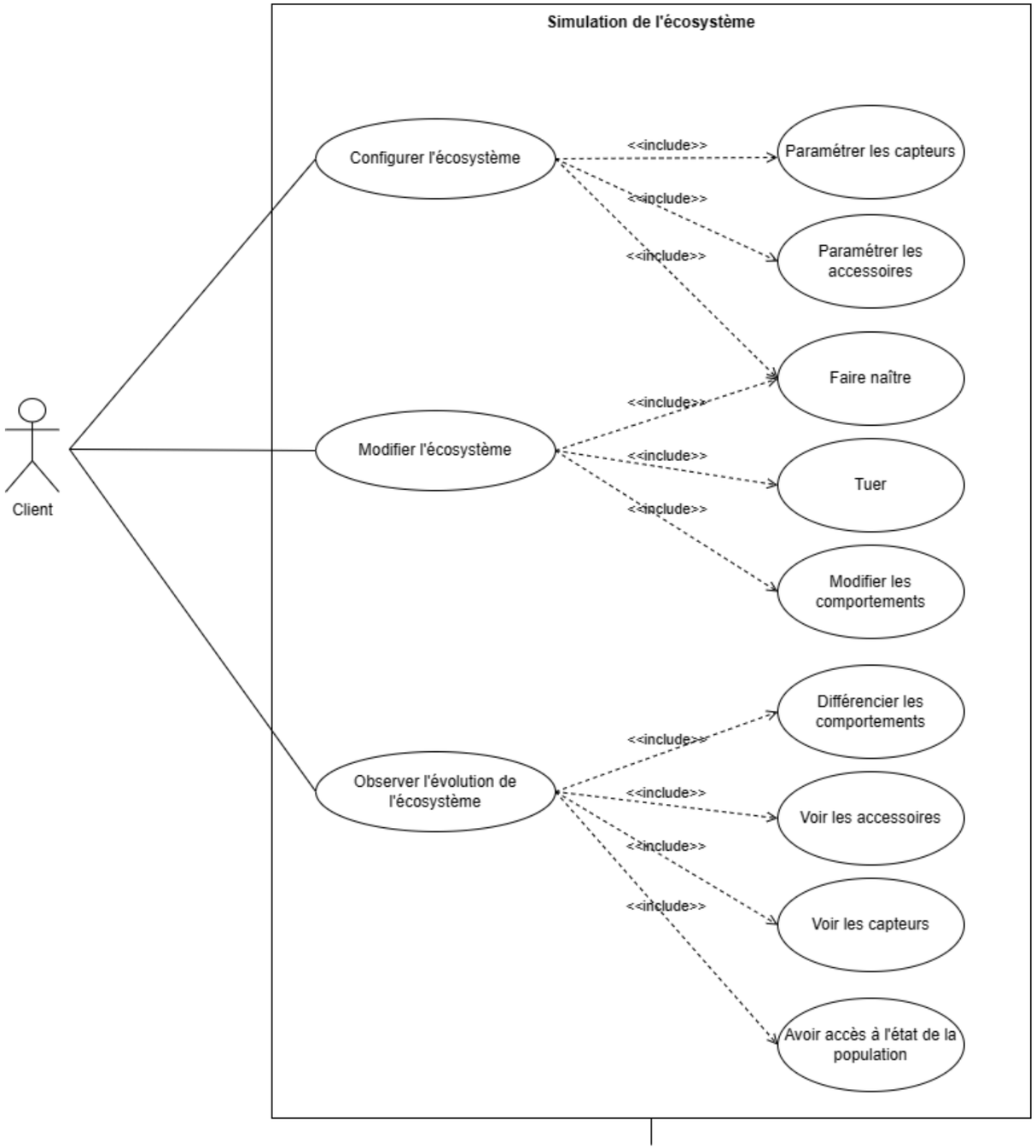
Simulation d'un écosystème d'aquarium avec des bestioles se déplaçant dans un milieu et interagissant entre elles.

Modifications à Apporter :

- Ajout de fonctionnalités telles que la naissance, la mort et le clonage des bestioles.
- Intégration d'équipements optionnels pour les bestioles comme des capteurs et des accessoires.
- Gestion des comportements spécifiques des bestioles en fonction de leur environnement et des événements extérieurs.

Phase de conception

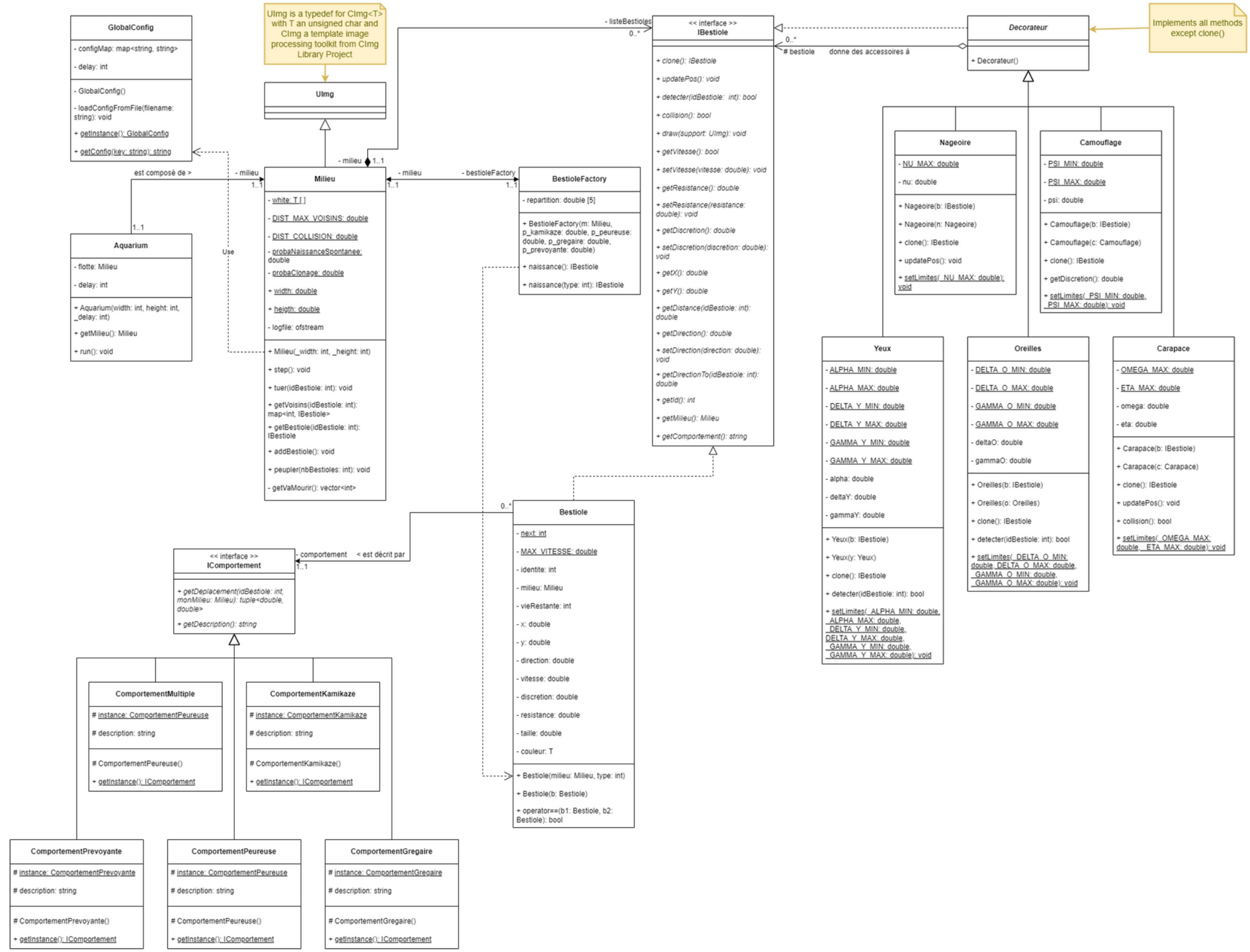
Diagramme de cas d'utilisation



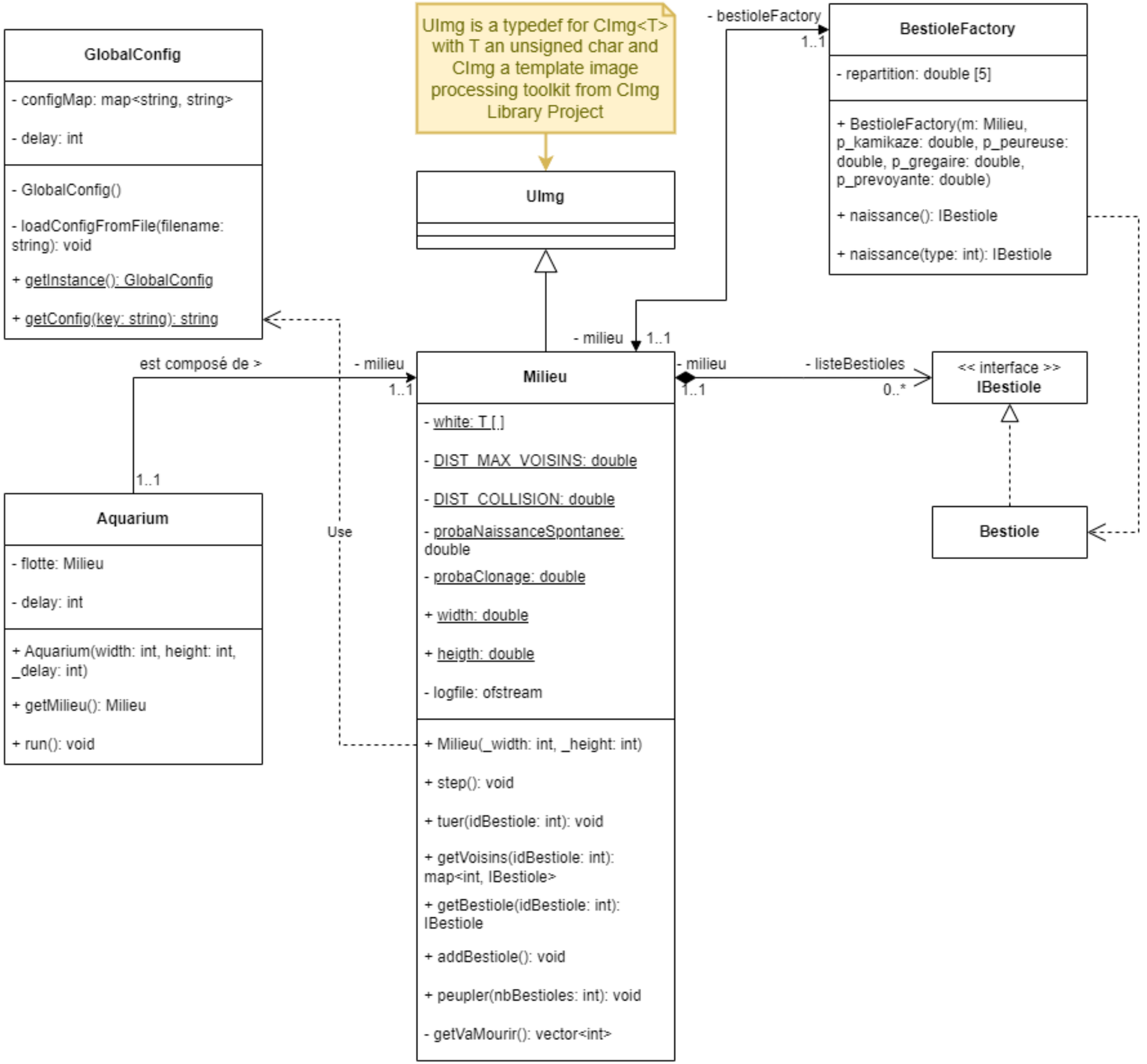
Diagrammes de classes

- Global
- Système
- Bestiole

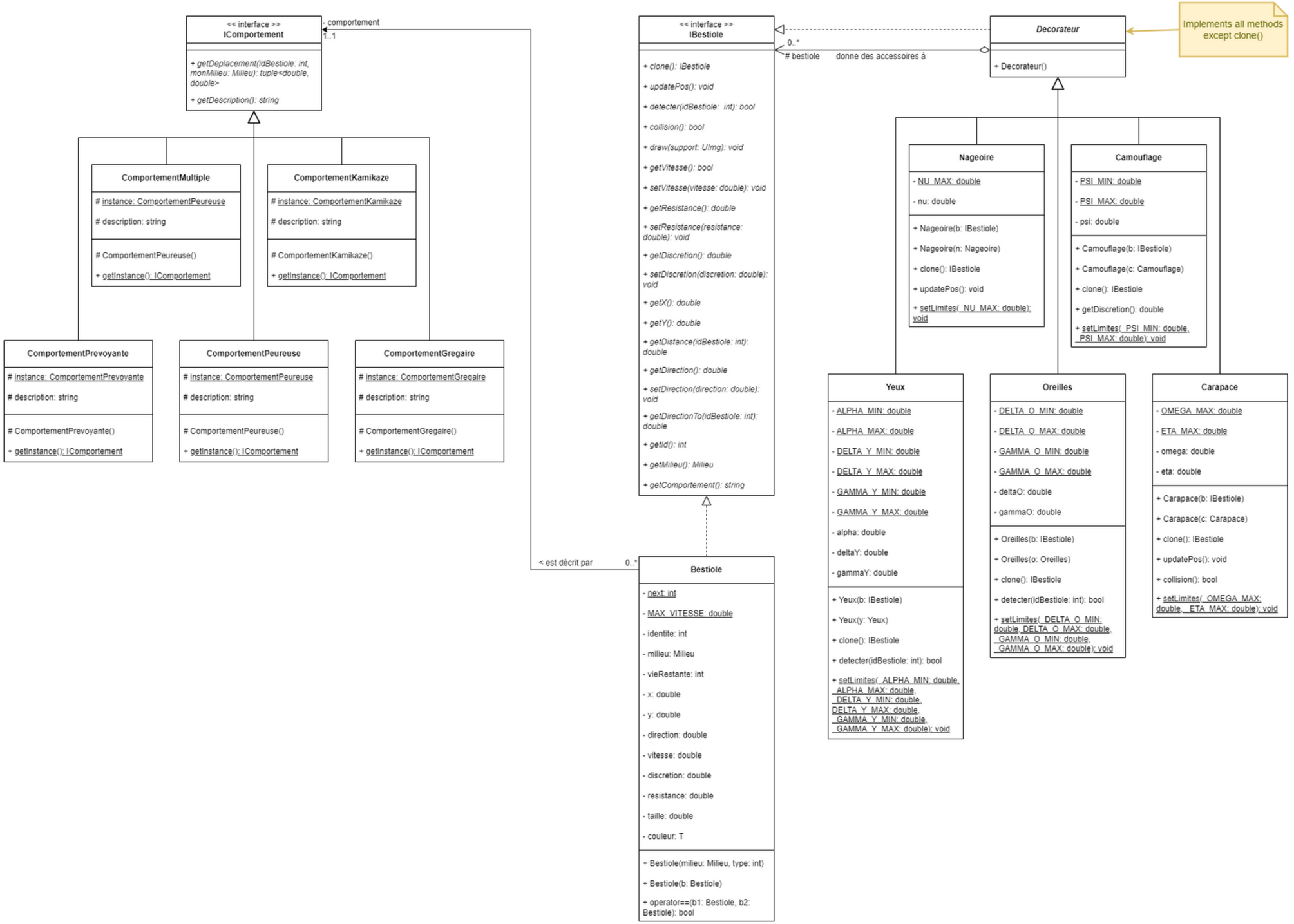
Global


GlobalConfig
- configMap: map<string, string>
- delay: int
- GlobalConfig()
- loadConfigFromFile(filename: string): void
+ getInstance(): GlobalConfig
+ getConfig(key: string): string
UImg is a typedef for CImg<T> with T an unsigned char and CImg a template image processing toolkit from CImg Library Project
UImg
- listeBestioles
0..*
<< interface >>
IBestiole
+ clone(): IBestiole
+ updatePos(): void
+ detecter(idBestiole: int): bool
+ collision(): bool
+ draw(support: UImg): void
+ getVitesse(): bool
+ setVitesse(vitesse: double): void
+ getResistance(): double
+ setResistance(resistance: double): void
+ getDiscretion(): double
+ setDiscretion(discretion: double): void
+ getX(): double
+ getY(): double
+ getDistance(idBestiole: int): double
+ getDirection(): double
+ setDirection(direction: double): void
+ getDirectionTo(idBestiole: int): double
+ getId(): int
+ getMilieu(): Milieu
+ getComportement(): string
Decorateur
+ Decorateur()
0..*
bestiole
donne des accessoires à
Implements all methods except clone()
- milieu
1..1
est composé de >
Milieu
- white_T [ ]
- DIST_MAX_VOISINS: double
- DIST_COLLISION: double
- probaNaissanceSpontanee: double
- probaClonage: double
+ width: double
+ heigth: double
- logfile: ofstream
+ Milieu(_width: int, _height: int)
+ step(): void
+ tuer(idBestiole: int): void
+ getVoisins(idBestiole: int): map<int, IBestiole>
+ getBestiole(idBestiole: int): IBestiole
+ addBestiole(): void
+ peupler(nbBestioles: int): void
- getVaMourir(): vector<int>
- bestioleFactory
BestioleFactory
- repartition: double [5]
+ BestioleFactory(m: Milieu, p_kamikaze: double, p_peureuse: double, p_gregaire: double, p_prevoyante: double)
+ naissance(): IBestiole
+ naissance(type: int): IBestiole
Aquarium
- flotte: Milieu
- delay: int
+ Aquarium(width: int, height: int, _delay: int)
+ getMilieu(): Milieu
+ run(): void
Use
Nageoire
- NU_MAX: double
- nu: double
+ Nageoire(b: IBestiole)
+ Nageoire(n: Nageoire)
+ clone(): IBestiole
+ updatePos(): void
+ setLimites(_NU_MAX: double): void
Camouflage
- PSI_MIN: double
- PSI_MAX: double
- psi: double
+ Camouflage(b: IBestiole)
+ Camouflage(c: Camouflage)
+ clone(): IBestiole
+ getDiscretion(): double
+ setLimites(_PSI_MIN: double, _PSI_MAX: double): void
Yeux
- ALPHA_MIN: double
- ALPHA_MAX: double
- DELTA_Y_MIN: double
- DELTA_Y_MAX: double
- GAMMA_Y_MIN: double
- GAMMA_Y_MAX: double
- alpha: double
- deltaY: double
- gammaY: double
+ Yeux(b: IBestiole)
+ Yeux(y: Yeux)
+ clone(): IBestiole
+ detecter(idBestiole: int): bool
+ setLimites(_ALPHA_MIN: double, _ALPHA_MAX: double, _DELTA_Y_MIN: double, DELTA_Y_MAX: double, _GAMMA_Y_MIN: double, _GAMMA_Y_MAX: double): void
Oreilles
- DELTA_O_MIN: double
- DELTA_O_MAX: double
- GAMMA_O_MIN: double
- GAMMA_O_MAX: double
- deltaO: double
- gammaO: double
+ Oreilles(b: IBestiole)
+ Oreilles(o: Oreilles)
+ clone(): IBestiole
+ detecter(idBestiole: int): bool
+ setLimites(_DELTA_O_MIN: double, DELTA_O_MAX: double, _GAMMA_O_MIN: double, _GAMMA_O_MAX: double): void
Carapace
- OMEGA_MAX: double
- ETA_MAX: double
- omega: double
- eta: double
+ Carapace(b: IBestiole)
+ Carapace(c: Carapace)
+ clone(): IBestiole
+ updatePos(): void
+ collision(): bool
+ setLimites(_OMEGA_MAX: double, _ETA_MAX: double): void
0..*
- comportement
< est décrit par
1..1
<< interface >>
IComportement
+ getDeplacement(idBestiole: int, monMilieu: Milieu): tuple<double, double>
+ getDescription(): string
Bestiole
- next: int
- MAX_VITESSE: double
- identite: int
- milieu: Milieu
- vieRestante: int
- x: double
- y: double
- direction: double
- vitesse: double
- discretion: double
- resistance: double
- taille: double
- couleur: T
+ Bestiole(milieu: Milieu, type: int)
+ Bestiole(b: Bestiole)
+ operator==(b1: Bestiole, b2: Bestiole): bool
ComportementMultiple
instance: ComportementPeureuse
description: string
ComportementPeureuse()
+ getInstance(): IComportement
ComportementKamikaze
instance: ComportementKamikaze
description: string
ComportementKamikaze()
+ getInstance(): IComportement
ComportementPrevoyante
instance: ComportementPrevoyante
description: string
ComportementPrevoyante()
+ getInstance(): IComportement
ComportementPeureuse
instance: ComportementPeureuse
description: string
ComportementPeureuse()
+ getInstance(): IComportement
ComportementGregaire
instance: ComportementGregaire
description: string
ComportementGregaire()
+ getInstance(): IComportement

Système


UImg is a typedef for CImg<T> with T an unsigned char and CImg a template image processing toolkit from CImg Library Project
GlobalConfig
- configMap: map<string, string>
- delay: int
- GlobalConfig()
- loadConfigFromFile(filename: string): void
+ getInstance(): GlobalConfig
+ getConfig(key: string): string
UImg
- bestioleFactory
1..1
BestioleFactory
- repartition: double [5]
+ BestioleFactory(m: Milieu, p_kamikaze: double, p_peureuse: double, p_gregaire: double, p_prevoyante: double)
+ naissance(): IBestiole
+ naissance(type: int): IBestiole
- milieu
1..1
est composé de >
- milieu
1..1
Milieu
- white: T [ ]
- DIST_MAX_VOISINS: double
- DIST_COLLISION: double
- probaNaissanceSpontanee: double
- probaClonage: double
+ width: double
+ heigth: double
- logfile: ofstream
+ Milieu(_width: int, _height: int)
+ step(): void
+ tuer(idBestiole: int): void
+ getVoisins(idBestiole: int): map<int, IBestiole>
+ getBestiole(idBestiole: int): IBestiole
+ addBestiole(): void
+ peupler(nbBestioles: int): void
- getVaMourir(): vector<int>
- milieu
1..1
- listeBestioles
0..*
<< interface >>
IBestiole
Bestiole
1..1
Aquarium
- flotte: Milieu
- delay: int
+ Aquarium(width: int, height: int, _delay: int)
+ getMilieu(): Milieu
+ run(): void
Use

Bestiole

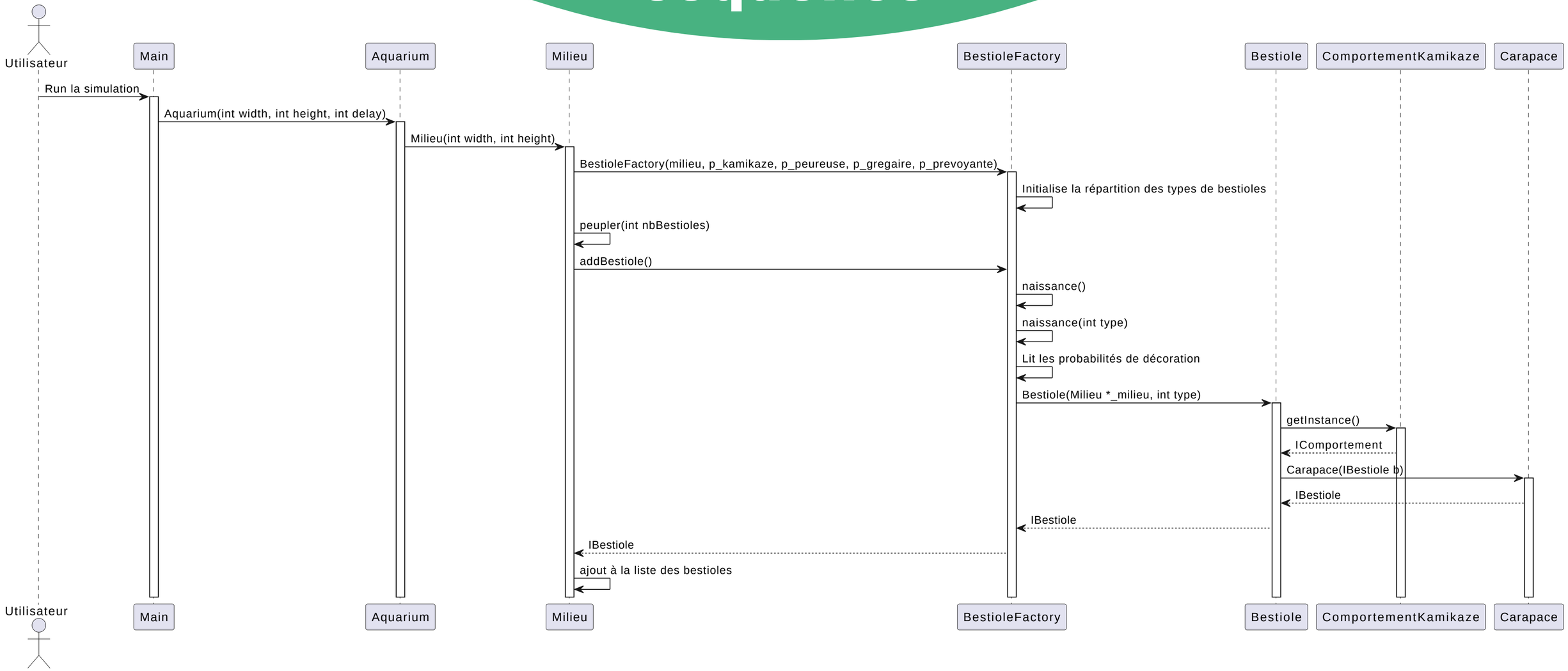

« interface »
IComportement
+ getDeplacement(idBestiole: int, monMilieu: Milieu): tuple<double, double>
+ getDescription(): string
- comportement
1..1
ComportementMultiple
instance: ComportementPeureuse
description: string
ComportementPeureuse()
+ getInstance(): IComportement
ComportementKamikaze
instance: ComportementKamikaze
description: string
ComportementKamikaze()
+ getInstance(): IComportement
ComportementPrevoyante
instance: ComportementPrevoyante
description: string
ComportementPrevoyante()
+ getInstance(): IComportement
ComportementPeureuse
instance: ComportementPeureuse
description: string
ComportementPeureuse()
+ getInstance(): IComportement
ComportementGregaire
instance: ComportementGregaire
description: string
ComportementGregaire()
+ getInstance(): IComportement
« interface »
IBestiole
+ clone(): IBestiole
+ updatePos(): void
+ detecter(idBestiole: int): bool
+ collision(): bool
+ draw(support: UImg): void
+ getVitesse(): bool
+ setVitesse(vitesse: double): void
+ getResistance(): double
+ setResistance(resistance: double): void
+ getDiscretion(): double
+ setDiscretion(discretion: double): void
+ getX(): double
+ getY(): double
+ getDistance(idBestiole: int): double
+ getDirection(): double
+ setDirection(direction: double): void
+ getDirectionTo(idBestiole: int): double
+ getId(): int
+ getMilieu(): Milieu
+ getComportement(): string
0..*
bestiole
donne des accessoires à
Decorateur
+ Decorateur()
Implements all methods except clone()
< est décrit par
0..*
Bestiole
- next: int
- MAX_VITESSE: double
- identite: int
- milieu: Milieu
- vieRestante: int
- x: double
- y: double
- direction: double
- vitesse: double
- discretion: double
- resistance: double
- taille: double
- couleur: T
+ Bestiole(milieu: Milieu, type: int)
+ Bestiole(b: Bestiole)
+ operator==(b1: Bestiole, b2: Bestiole): bool
Nageoire
- NU_MAX: double
- nu: double
+ Nageoire(b: IBestiole)
+ Nageoire(n: Nageoire)
+ clone(): IBestiole
+ updatePos(): void
+ setLimites(_NU_MAX: double): void
Camouflage
- PSI_MIN: double
- PSI_MAX: double
- psi: double
+ Camouflage(b: IBestiole)
+ Camouflage(c: Camouflage)
+ clone(): IBestiole
+ getDiscretion(): double
+ setLimites(_PSI_MIN: double, _PSI_MAX: double): void
Yeux
- ALPHA_MIN: double
- ALPHA_MAX: double
- DELTA_Y_MIN: double
- DELTA_Y_MAX: double
- GAMMA_Y_MIN: double
- GAMMA_Y_MAX: double
- alpha: double
- deltaY: double
- gammaY: double
+ Yeux(b: IBestiole)
+ Yeux(y: Yeux)
+ clone(): IBestiole
+ detecter(idBestiole: int): bool
+ setLimites(_ALPHA_MIN: double, _ALPHA_MAX: double, _DELTA_Y_MIN: double, DELTA_Y_MAX: double, _GAMMA_Y_MIN: double, _GAMMA_Y_MAX: double): void
Oreilles
- DELTA_O_MIN: double
- DELTA_O_MAX: double
- GAMMA_O_MIN: double
- GAMMA_O_MAX: double
- deltaO: double
- gammaO: double
+ Oreilles(b: IBestiole)
+ Oreilles(o: Oreilles)
+ clone(): IBestiole
+ detecter(idBestiole: int): bool
+ setLimites(_DELTA_O_MIN: double, DELTA_O_MAX: double, _GAMMA_O_MIN: double, _GAMMA_O_MAX: double): void
Carapace
- OMEGA_MAX: double
- ETA_MAX: double
- omega: double
- eta: double
+ Carapace(b: IBestiole)
+ Carapace(c: Carapace)
+ clone(): IBestiole
+ updatePos(): void
+ collision(): bool
+ setLimites(_OMEGA_MAX: double, _ETA_MAX: double): void

Diagrammes de séquences

- Peuplement
- Etape (step)

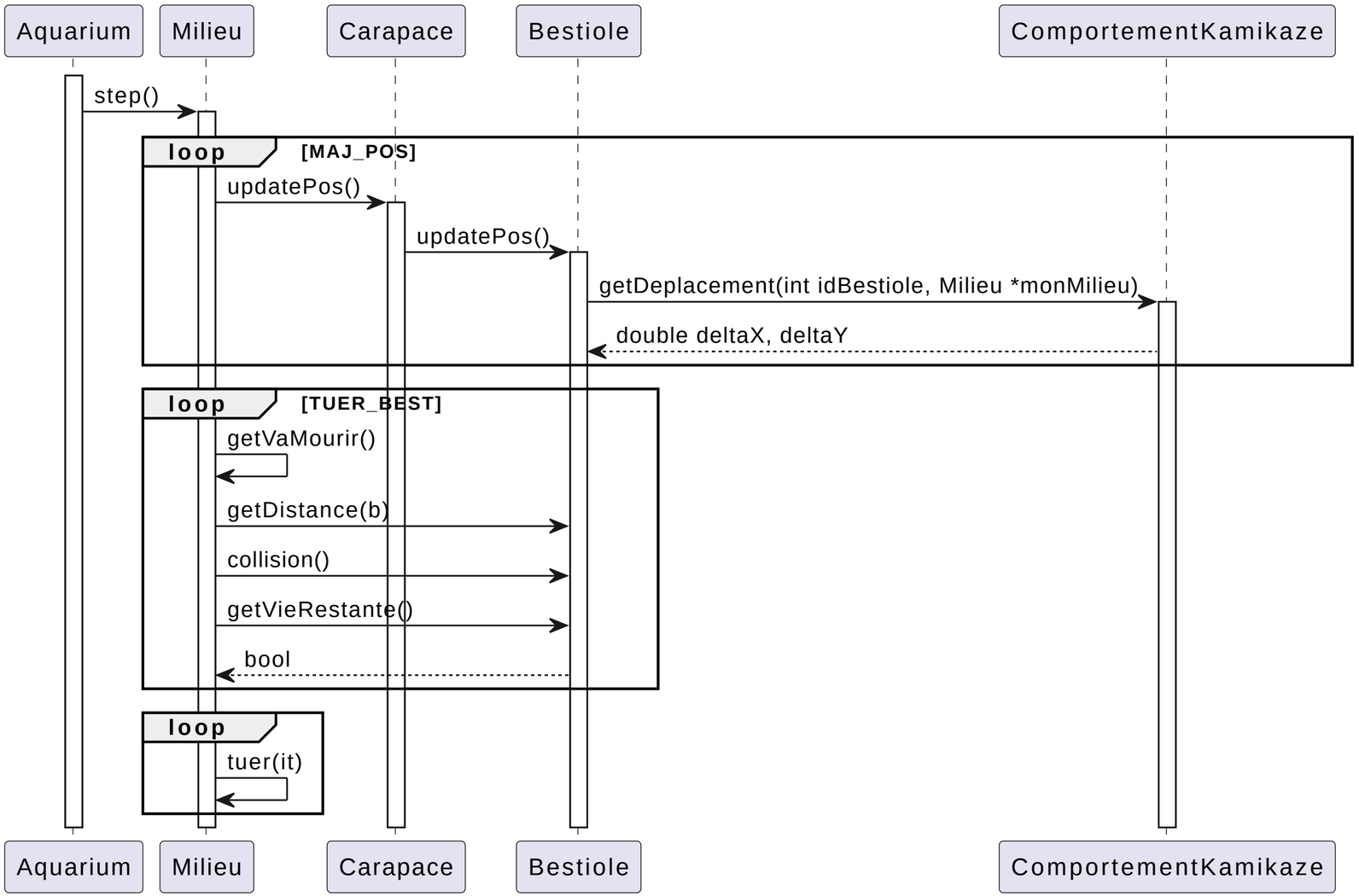
Peuplement

Diagramme de séquence



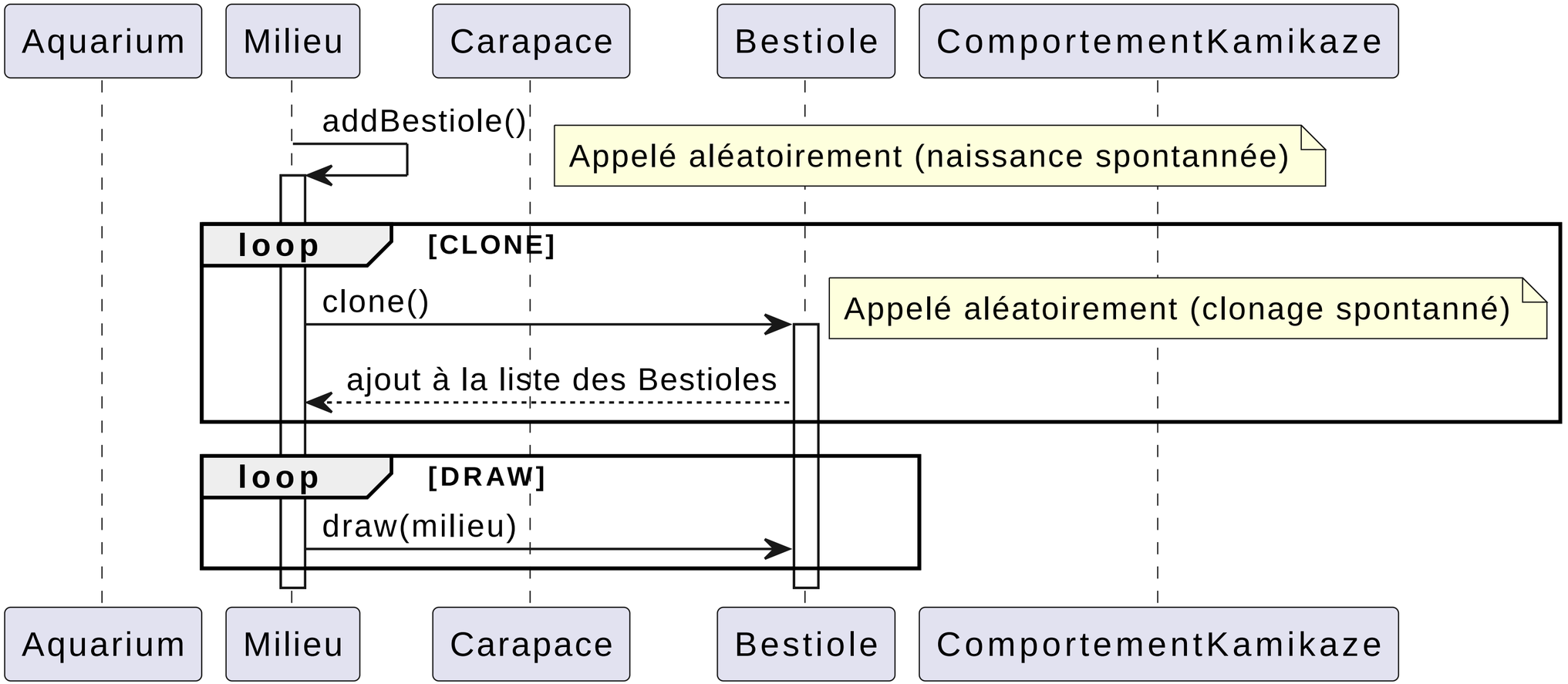
Step 1/2

Diagramme de séquence

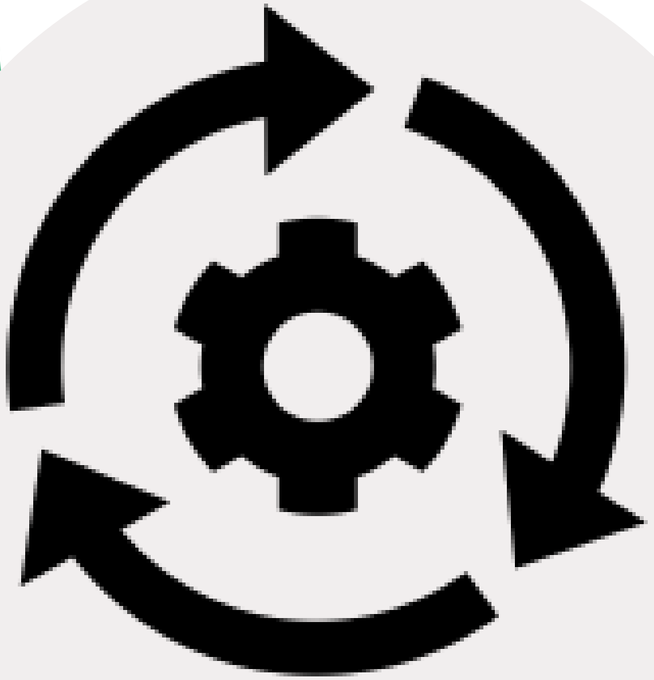


Step 2/2

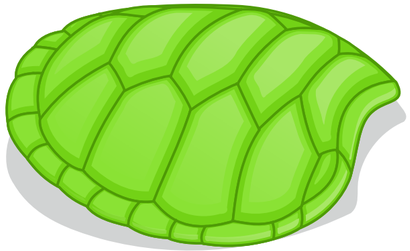
Diagramme de séquence



Phase d‘implémentation



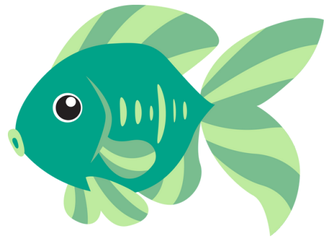
Accessoires



```
class Carapace : public Decorateur {
private:
    /**
     * Résistance.
     * 1 < omega < OMEGA_MAX
     */
    double omega;
    /**
     * Réduction de vitesse.
     * 1 < eta < ETA_MAX
    */
    double eta;
```

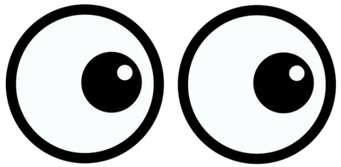
```
    void updatePos() override;
    bool collision() override;
```

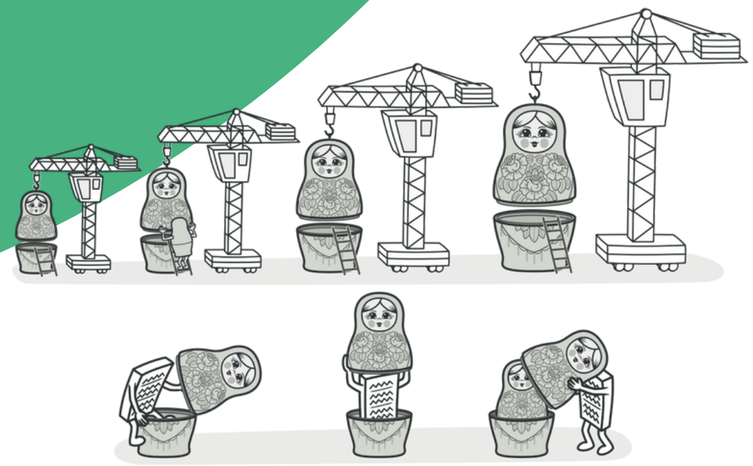
```
/**
 * Un accessoire des bestioles.
 * Augmente la probabilité que la bestiole soit indétectable.
 */
class Camouflage : public Decorateur {
private:
    /**
     * Probabilité de discrétion.
     * 0 < PSI_MIN < psi < PSI_MAX < 1
     */
    double psi;
    static double PSI_MIN;
    static double PSI_MAX;

public:
    Camouflage(std::shared_ptr<IBestiole> b);
    Camouflage(Camouflage &c);
    ~Camouflage() override;
    static void setLimites(double _PSI_MIN, double _PSI_MAX);
    std::shared_ptr<IBestiole> clone() override;
    double getDiscretion() const override;
};
```



```
class Nageoire : public Decorateur {
private:
    /**
     * Augmentation de vitesse.
     * 1 < nu < NU_MAX
     */
    double nu;
    static double NU_MAX;

public:
    Nageoire(std::shared_ptr<IBestiole> b);
    Nageoire(Nageoire &n);
    ~Nageoire() override;
    static void setLimites(double _NU_MAX);
    std::shared_ptr<IBestiole> clone() override;
    void updatePos() override;
```

Capteurs





```cpp
bool Yeux::detecter(int idBestiole) {
    bool detection;
    double distance = bestiole->getDistance(idBestiole);
    double directionTo = bestiole->getDirectionTo(idBestiole);
    double direction = bestiole->getDirection();

    if (direction < 0) {
        direction += 2 * M_PI;
    }
    if (directionTo < 0) {
        directionTo += 2 * M_PI;
    }
    direction = fmod(direction, 2 * M_PI);
    directionTo = fmod(directionTo, 2 * M_PI);
    double angle = fabs(direction - directionTo);

    if (angle > M_PI) {
        angle = 2 * M_PI - angle;
    }
    bool inField = (angle < alpha / 2);
    bool inDistance = (distance < deltaY);
    bool inVision = (inField && inDistance);
    // détection si dans le champ de vision et dans la distance
    auto b = bestiole->getMilieu()->getBestiole(idBestiole);
    detection = inVision && (gammaY > b->getDiscretion());
    return detection;
}
```

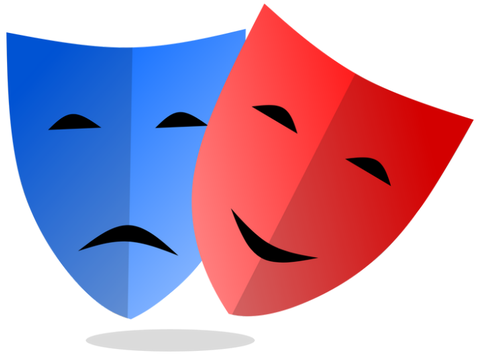
Yeux

Design Pattern :
Décorateur

```cpp
bool Oreilles::detecter(int idBestiole) {
    bool detection;
    bool inDistance = bestiole->getDistance(idBestiole) < deltaO;
    // detecte si la bestiole est dans la distance de detection et si elle est detectable
    auto b = bestiole->getMilieu()->getBestiole(idBestiole);
    detection = inDistance && (gammaO > b->getDiscretion());
    return detection;
}
```

Oreilles

Comportements (1/2)



```cpp
for (auto it = voisins->begin(); it != voisins->end(); ++it) {
    if (b->detecter(it->first)) {
        double dir = b->getDirectionTo(it->second->getId());
        if (dir < 0) {
            dir += 2 * M_PI;
        }
        direction += dir;
        count++;
    }
}
// Lorsqu'il y a trop de voisins, la bestiole fuit
if (count > MAX_COUNT) {
    direction /= count;
    // direction opposée
    direction += M_PI;
    b->setDirection(direction);
    // doublement de la vitesse
    vitesse = 2 * b->getVitesse();
    cout << "(" << idBestiole << "): je fuis" << endl;
} else {
    direction = b->getDirection();
    vitesse = b->getVitesse();
}

deltaX = cos(direction) * vitesse;
deltaY = -sin(direction) * vitesse;
```

Peureuse

```cpp
std::vector<std::shared_ptr<IComportement>> tous_comportements_ = {
    ComportementGregaire::getInstance(),
    ComportementKamikaze::getInstance(),
    ComportementPeureuse::getInstance(),
    ComportementPrevoyante::getInstance()
};
```

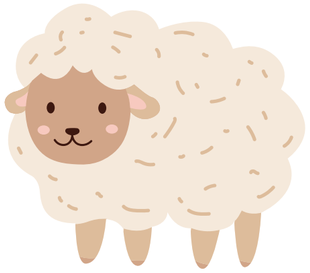
```cpp
// Choisir un comportement aléatoire (possible de logger le comportement choisi)
int comportement = rand() % tous_comportements.size();

return tous_comportements[comportement]->getDeplacement(idBestiole, monMilieu);
```

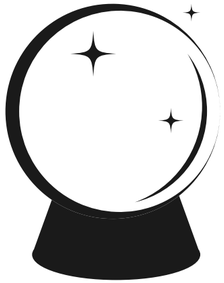
Multiple

```cpp
auto voisins = monMilieu->getVoisins(idBestiole);
for (auto it = voisins->begin(); it != voisins->end(); ++it) {
    if (b->detecter(it->first)) {
        distance = it->second->getDistance(idBestiole);
        if (distance < distanceMin) {
            distanceMin = distance;
            voisinDetecte = true;
            dir = b->getDirectionTo(it->second->getId());
        }
    }
}

double vitesse = b->getVitesse();
if (!voisinDetecte) {
    double direction = b->getDirection();
    deltaX = cos(direction) * vitesse;
    deltaY = -sin(direction) * vitesse;
}
else {
    deltaX = cos(dir) * vitesse;
    deltaY = -sin(dir) * vitesse;
    b->setDirection(dir);
}
```

Kamikaze

Comportements (2/2)





```
double direction = 0;
int count = 0;

for (auto it = voisins->begin(); it != voisins->end(); ++it) {
    if (b->detecter(it->first)) {
        voisinDetecte = true;
        double dir = it->second->getDirection();
        if (dir < 0) {
            dir += 2 * M_PI;
        }
        direction += dir;
        count++;
    }
}

if (voisinDetecte) {
    direction /= count;
    b->setDirection(direction);
} else {
    direction = b->getDirection();
}

double vitesse = b->getVitesse();
deltaX = cos(direction) * vitesse;
deltaY = -sin(direction) * vitesse;
```

Grégaire

```
for (auto it = voisins->begin(); it != voisins->end(); ++it) {
    int idVoisin = it->first;
    auto voisin = it->second;
    if (b->detecter(idVoisin)) {
        voisinDetecte = true;
        posX_voisin = voisin->getX();
        posY_voisin = voisin->getY();
        direction_voisin = voisin->getDirection();
        vitesse_voisin = voisin->getVitesse();
        posX_voisin += cos(direction_voisin) * vitesse_voisin;
        posY_voisin -= sin(direction_voisin) * vitesse_voisin;

        // Calcul de la direction du voisin (vers sa position estimée)

        double dir = atan2(posY_voisin - posY, posX_voisin - posX);
        if (dir < 0) {
            dir += 2 * M_PI;
        }
        directions.push_back(dir);

    }
}
```

```
if (voisinDetecte) {
    sort(directions.begin(), directions.end());

    // Générer les intervalles

    for (int i = 0; i < (int)directions.size() - 1; i++) {
        interval = directions[i + 1] - directions[i];
        if (interval < 0) {
            interval += 2 * M_PI;
        }
        intervalles.push_back(interval);
    }

    interval = directions[0] - directions[directions.size() - 1] + 2 * M_PI;
    intervalles.push_back(interval);

    // Aller dans la direction qui correspond au centre de l'intervalle le plus grand

    for (int i = 0; i < (int)intervalles.size(); i++) {
        if (intervalles[i] > maxInterval) {
            maxInterval = intervalles[i];
            indexMaxInterval = i;
        }
    }

    direction = directions[indexMaxInterval] + intervalles[indexMaxInterval] / 2;
}
```

Prévoyante

CONCLUSION