SIMCITREE

Simulateur de foret

PROJET S3, DUT Informatique

Florian COLLE
Matthieu GIACCAGLIA
Mateusz BIREMBAUT
Walter DENEUVILLE

Tuteur: Marc JOANNIDES

15/01

SIMCITREE

SOMMAIRE

01 CONTEXTE

02 DÉROULEMENT ET ORGANISATION

03 ANALYSE

SOMMAIRE

01

CONTEXTE

CADRE DE RÉALISATION

ACTUEL

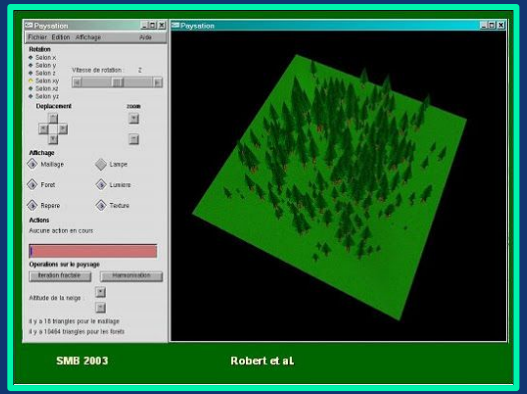
L'écologie est un enjeu très important

SIMULATION

Nous souhaitions créer un simulateur

BIOINFORMATIQUE

Utilisation de l'informatique pour étudier le vivant



OUTIL SCIENTIFIQUE

Modèle utilisable pour des recherches

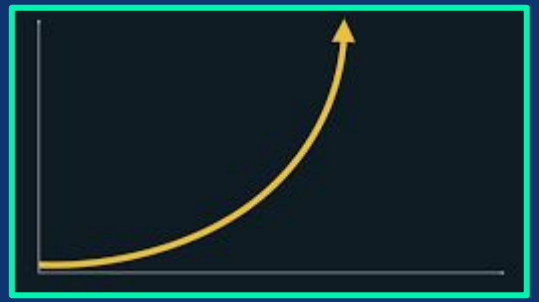
OBJECTIFS

SIMULER

Simuler l'évolution d'une forêt avec des paramètres limités

CRÉER

Un terrain de 1x1 en tore



EVOLUTION

Utilisation de formule de mathématiques

OPTIMISER

Optimiser le code

02

DÉROULEMENT ET ORGANISATION

LANCEMENT

Affectation sujet : 01/10/2020

Première Réunion : 15/10/2020

Première Présentation : 21/10/2020

Vacances : 24/10/2020 - 1/11/2020

Tâches Effectuées :

- Prise en main du sujet
- Réalisation Premiers Diagrammes
- Analyse Sujet 3-4 Jours

Vacances :

- Premiere Version
- Premières Questions

ORGANISATION

RÉUNION RÉGULIÈRES

Equipe

RÉUNION HEBDOMADAIRES

Tuteur

DIVISION DES TACHES

Création de sous tâches

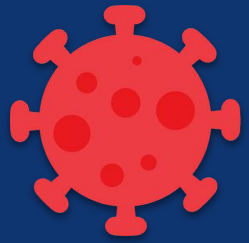
RÉPARTITION DES TÂCHES

Répartition des sous tâches

PLANIFICATION

Création Diagramme GANT

RÉORGANISATION



CONFINEMENT

Bouleversement des plans d'organisation

SIMPLIFICATION

La Covid-19 a simplifié l'organisation de réunions

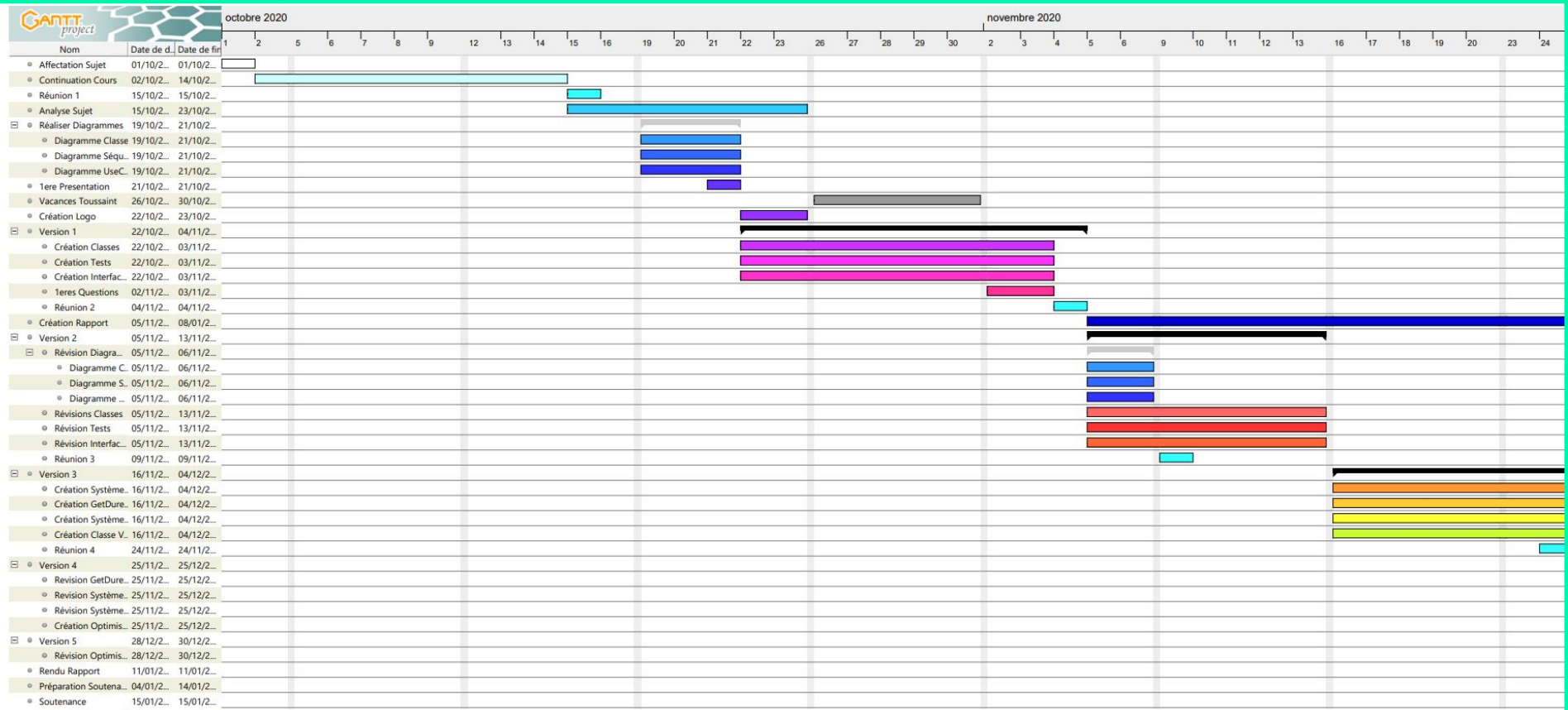
REPLANNIFICATION

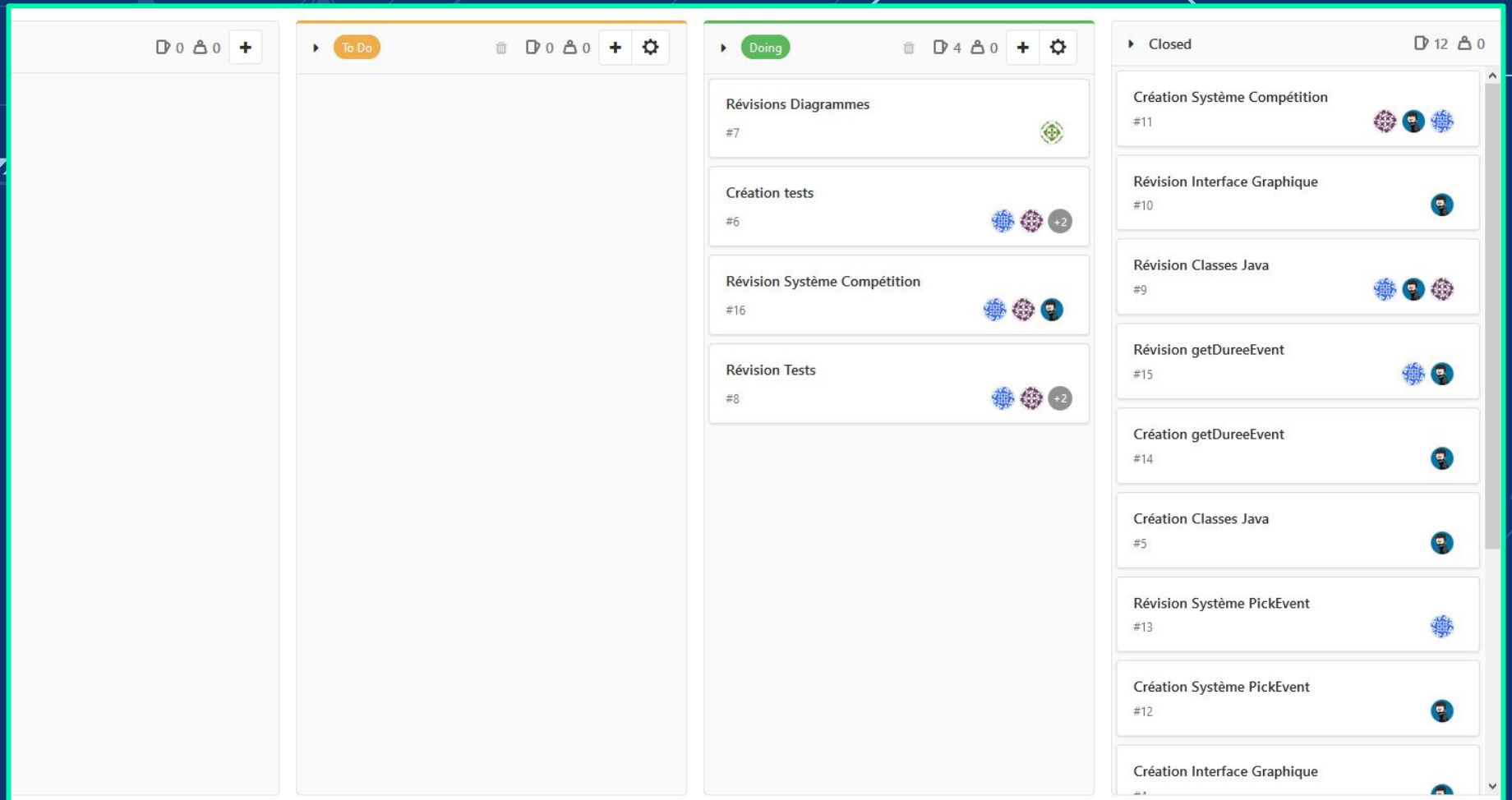
Modification GANT

TECHNOLOGIES ET MÉTHODES UTILISÉES

Logo **GitLab**

- GitLab
- Méthode agile SCRUM
- Diagramme GANT
- Utilisation Sprints, Backlog, feedback


GANTT project
octobre 2020
novembre 2020
1 2 5 6 7 8 9 12 13 14 15 16 19 20 21 22 23 26 27 28 29 30 2 3 4 5 6 9 10 11 12 13 16 17 18 19 20 23 24
Nom
Date de d...
Date de fin
Affectation Sujet 01/10/2... 01/10/2...
Continuation Cours 02/10/2... 14/10/2...
Réunion 1 15/10/2... 15/10/2...
Analyse Sujet 15/10/2... 23/10/2...
Réaliser Diagrammes 19/10/2... 21/10/2...
Diagramme Classe 19/10/2... 21/10/2...
Diagramme Séqu... 19/10/2... 21/10/2...
Diagramme UseC... 19/10/2... 21/10/2...
1ere Presentation 21/10/2... 21/10/2...
Vacances Toussaint 26/10/2... 30/10/2...
Création Logo 22/10/2... 23/10/2...
Version 1 22/10/2... 04/11/2...
Création Classes 22/10/2... 03/11/2...
Création Tests 22/10/2... 03/11/2...
Création Interfac... 22/10/2... 03/11/2...
1eres Questions 02/11/2... 03/11/2...
Réunion 2 04/11/2... 04/11/2...
Création Rapport 05/11/2... 08/01/2...
Version 2 05/11/2... 13/11/2...
Révision Diagra... 05/11/2... 06/11/2...
Diagramme C... 05/11/2... 06/11/2...
Diagramme S... 05/11/2... 06/11/2...
Diagramme ... 05/11/2... 06/11/2...
Révisions Classes 05/11/2... 13/11/2...
Révision Tests 05/11/2... 13/11/2...
Révision Interfac... 05/11/2... 13/11/2...
Réunion 3 09/11/2... 09/11/2...
Version 3 16/11/2... 04/12/2...
Création Système... 16/11/2... 04/12/2...
Création GetDure... 16/11/2... 04/12/2...
Création Système... 16/11/2... 04/12/2...
Création Classe V... 16/11/2... 04/12/2...
Réunion 4 24/11/2... 24/11/2...
Version 4 25/11/2... 25/12/2...
Revision GetDure... 25/11/2... 25/12/2...
Revision Système... 25/11/2... 25/12/2...
Révision Système... 25/11/2... 25/12/2...
Création Optimis... 25/11/2... 25/12/2...
Version 5 28/12/2... 30/12/2...
Révision Optimis... 28/12/2... 30/12/2...
Rendu Rapport 11/01/2... 11/01/2...
Préparation Soutena... 04/01/2... 14/01/2...
Soutenance 15/01/2... 15/01/2...


0 0
To Do
0 0
Doing
4 0
Closed
12 0
Révisions Diagrammes
#7
Création tests
#6
+2
Révision Système Compétition
#16
Révision Tests
#8
+2
Création Système Compétition
#11
Révision Interface Graphique
#10
Révision Classes Java
#9
Révision getDureeEvent
#15
Création getDureeEvent
#14
Création Classes Java
#5
Révision Système PickEvent
#13
Création Système PickEvent
#12
Création Interface Graphique

03

ANALYSE

LES PARAMÈTRES DE LA FORÊT

- Rayon de dispersion
- Rayon de compétition
- Taux de mortalité
- Taux de naissance
- Position (x,y)
- Intensité de compétition

DIAGRAMME DE CLASSE

Premier diagramme réalisé après la première réunion :

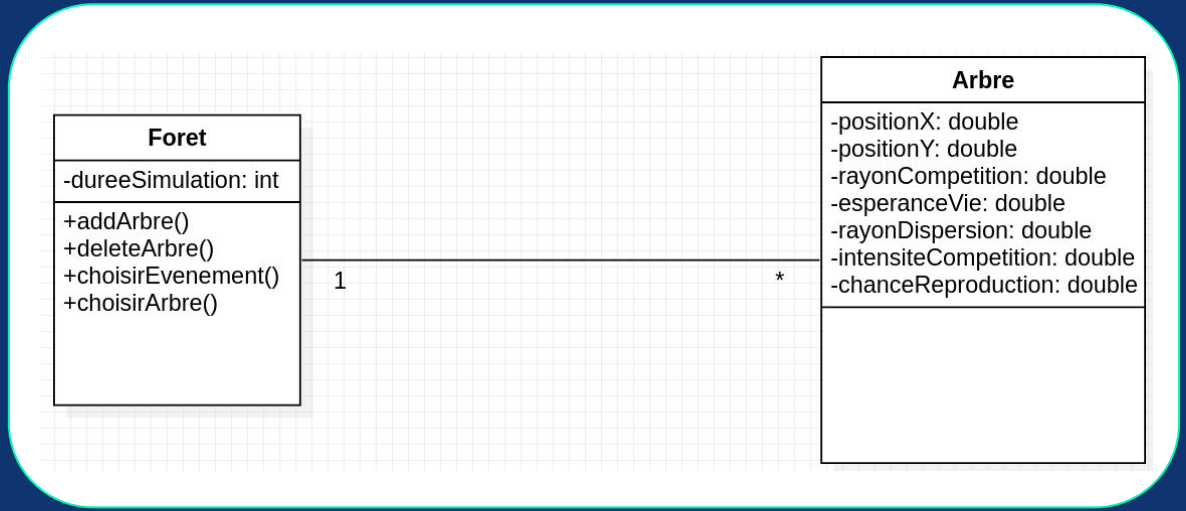


DIAGRAMME DE CLASSE

Deuxième diagramme réalisé :

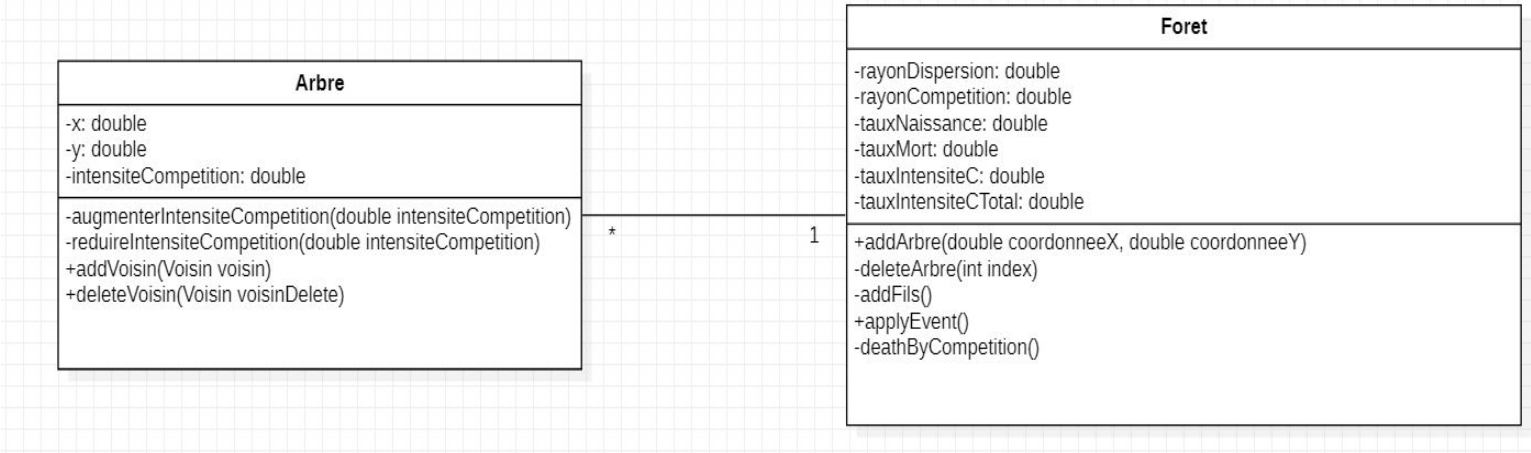


DIAGRAMME DE CLASSE

Dernier diagramme réalisé :

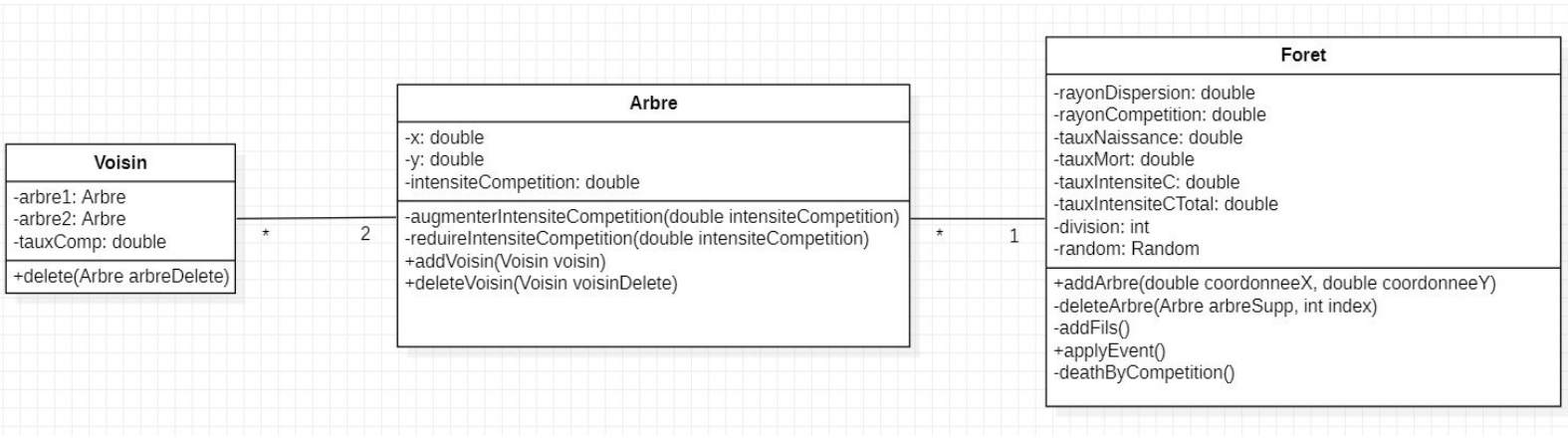
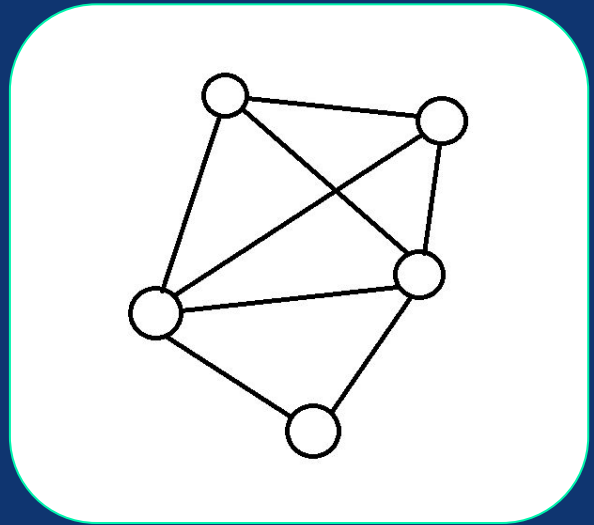


DIAGRAMME DE CLASSE

Avec l’ajout de la classe voisin nous avons obtenu une structure en graphe qui nous permet de réduire le coût en mémoire de notre simulateur.



04

RÉALISATION

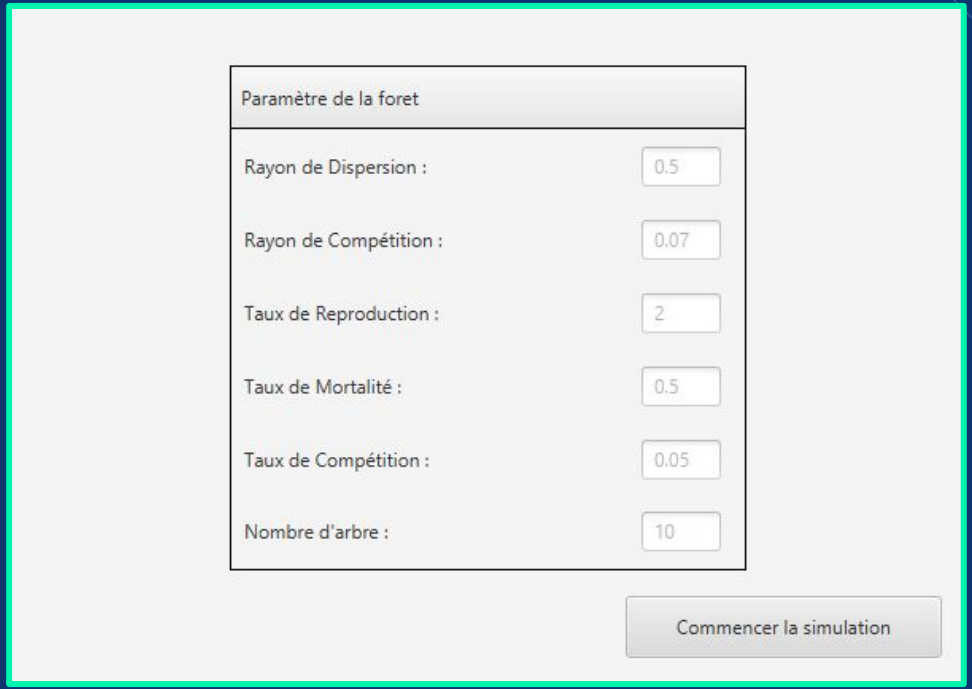
TECHNOLOGIE UTILISÉE

ALGORITHME DE BASE

```
F ←initForet()
initHorloge()
T ← 0
TANT QUE  T < Tmax do
        λglobal ← tauxGlobal(horloge)
        T ← T+ tireExp(λglobal)
        e ← tireEvent(horloge)
        appliqueEvent(e)
        miseAJour(horloge)
END WHILE
```

ALGORITHME DE BASE

F ←initForet()



ALGORITHME DE BASE

F ←initForet()

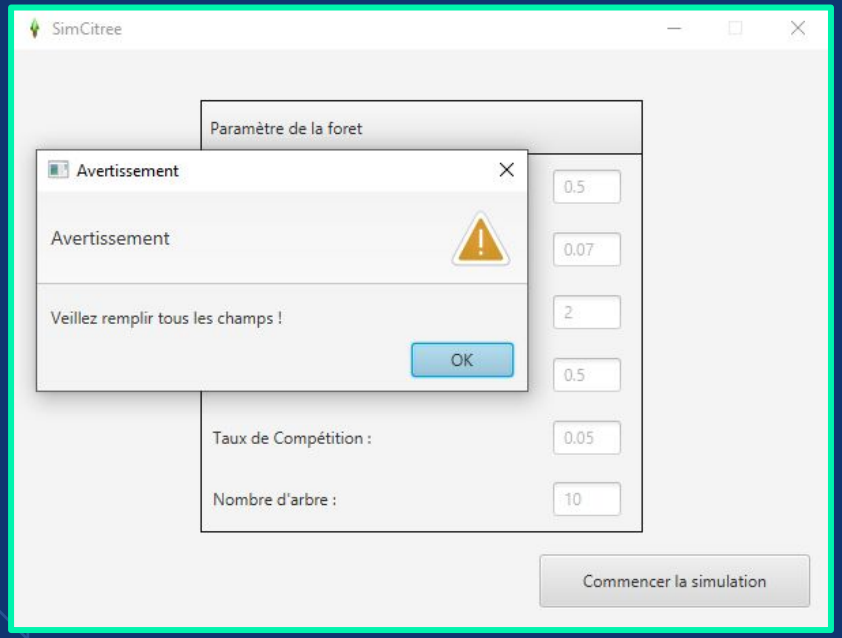
```
public class ControllerSetupForest {

	public TextField textRayonDisp;

	public void setupForest() throws IOException {

		String rayonDisp = textRayonDisp.getText();
		if (rayonDisp.equals("")){
			Alert alert = new Alert(Alert.AlertType.WARNING, "Erreur", ButtonType.OK);
			alert.showAndWait();
		}
		else if ( !rayonDisp.matches("[0-9]*[.][0-9]*") && !rayonDisp.matches("[0-9]*"){
			Alert alert = new Alert(Alert.AlertType.WARNING, "Erreur", ButtonType.OK);
			alert.showAndWait();
		}
		else {
			Main.foret = new Foret(Double.parseDouble(rayonDisp));
			Main.stage.close();
			Main.changeScene("/layout/forest.fxml");
		}
}
```

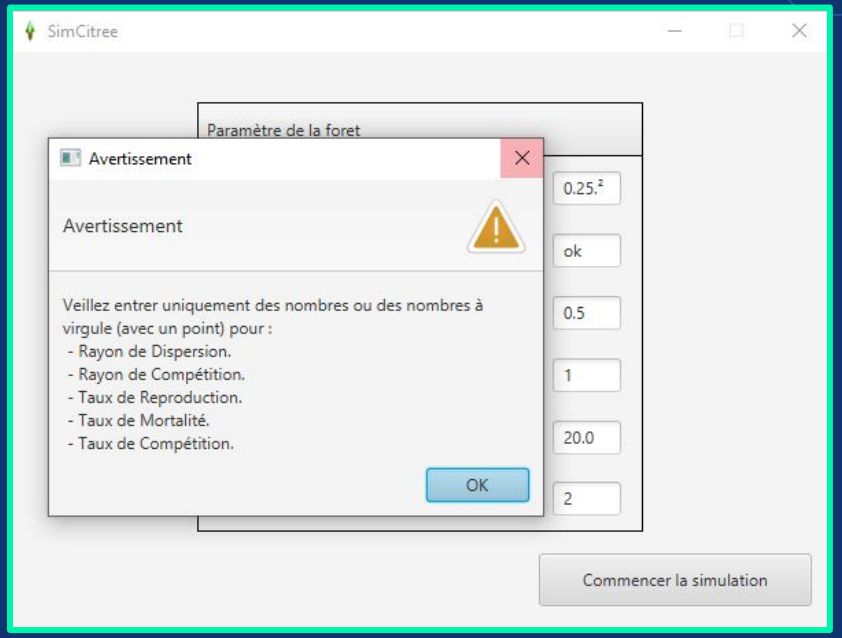
Si un champ de texte est vide

Si un champ de texte n'est pas un chiffre ou chiffre à virgule

ALGORITHME DE BASE

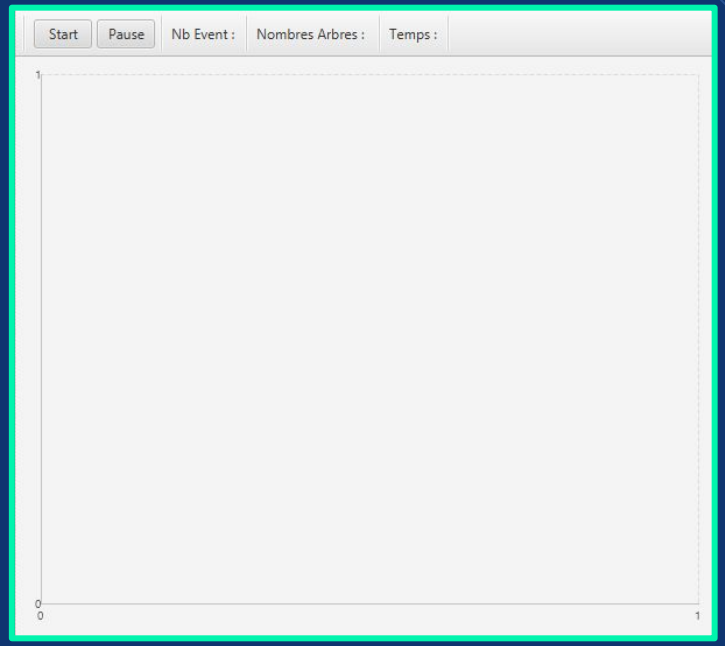
F ←initForet()





ALGORITHME DE BASE

```
initHorloge()
T ← 0
TANT QUE  T < Tmax do
        λglobal ← tauxGlobal(horloge)
        T ← T+ tireExp(λglobal)
        e ← tireEvent(horloge)
        appliqueEvent(e)
        miseAJour(horloge)
END WHILE
```



ALGORITHME DE BASE

```
public class ControllerForest implements Initializable {

	private AnimationTimer animationTimer;
	private Chrono chrono;

	@Override
	public void initialize(URL url, ResourceBundle resourceBundle) {
		chrono = new Chrono();
		animationTimer = new AnimationTimer() {
			private long lastTimeEvenement = 0;
			private long lastSecond = 0;
			private double nextTimeEvent = Main.foret.getDureeNextEvent();

			@Override
			public void handle(long now) {
				//PROCHAINE DIAPO
			}

	};
}
```

Initialisation des variables

Initialisation des variables pour l'animationTimer

Réécriture de la fonction handle() de l'animationTimer

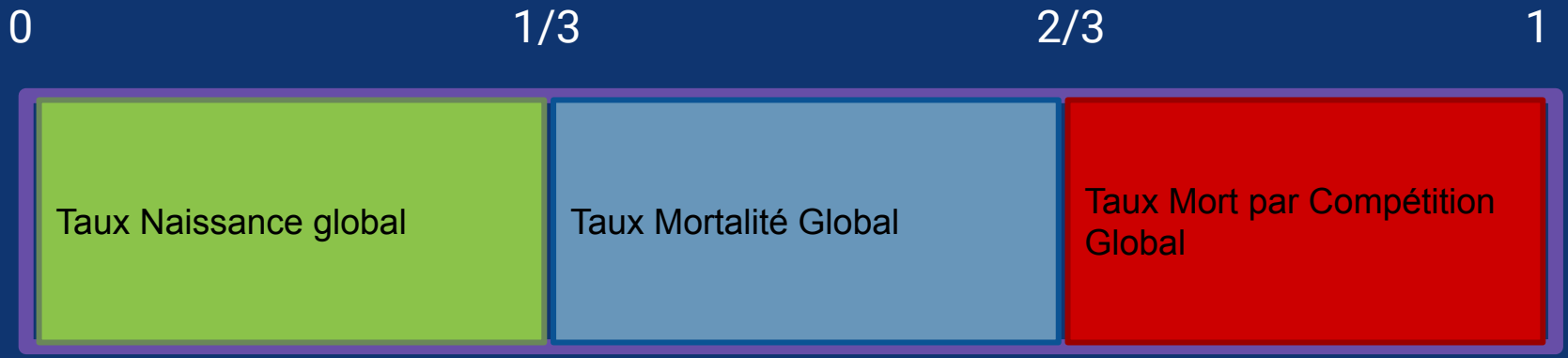
Classe Chrono créée par JeffProd

ALGORITHME DE BASE

```
if ((now - lastTimeEvenement)/ 1_000_000_000.0 >= nextTimeEvent && Main.foret.getList().size() > 0){
    nbEvent++;
    Main.foret.applyEvent();
    labelNbEvent.setText(String.valueOf(nbEvent));
    labelNbArbres.setText(String.valueOf(Main.foret.getList().size()));
    ecrireFichier();
    lastTimeEvenement = now;
    nextTimeEvent = Main.foret.getDureeNextEvent();
}
if (Main.foret.getList().size() == 0) {
    stop();
    chrono.stop();
}
if ((now - lastSecond)/1 000 000 000.0 >= 1) {
    labelTime.setText(chrono.getActuelDureeTxt());
    lastSecond = now;
}
```

Algorithme de “base”

TIRAGE ÉVÈNEMENT ALÉATOIRE



Tirage aléatoire entre 0 et 1

loi discrète

TIRAGE D'UN ARBRE ALÉATOIRE

SI (Événement Tirée = Naissance ou Mort Naturelle)

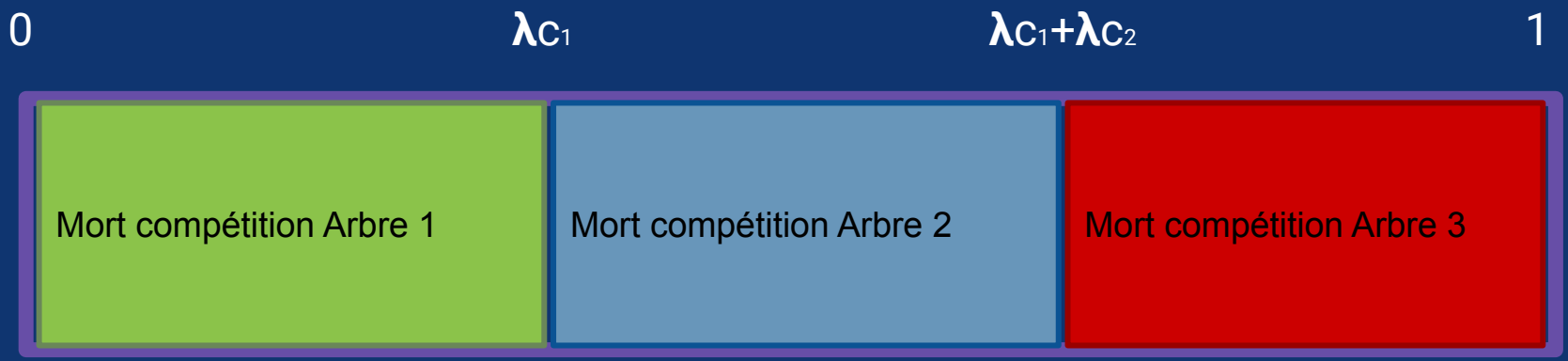
ALORS Tirage Arbre selon une loi uniforme

SINON SI (Événement Tirée = Mort par compétition)

ALORS Tirage Arbre selon une loi discrète

Appliquer Événement Tirée sur Arbre Tiré

TIRAGE MORT PAR COMPÉTITION



Tirage Arbre grâce à une loi discrète

PROCHAIN ÉVÈNEMENT

Taux Naissance

Taux Mort Naturelle

Taux Mort par compétition

TAUX GLOBAL = (Taux Naissance + Taux Mort Naturelle) * n + Taux Intensité Compétition Total

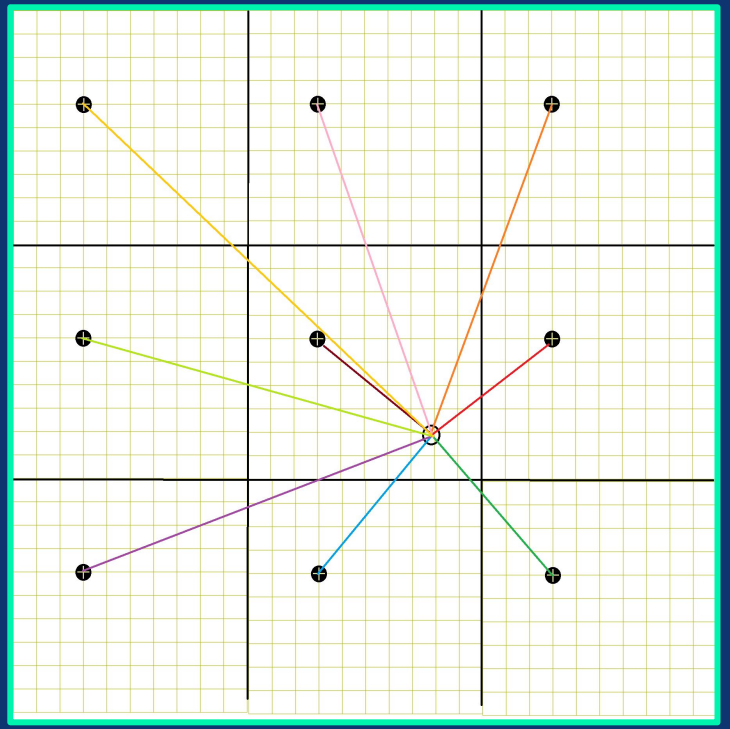
n - Nombre Arbres

PROCHAIN ÉVÈNEMENT

```
private double getTauxGlobal() {
    return ( (this.tauxNaissance+this.tauxMort) * listArbre.size()) +
tauxIntensiteCTotal;
}

public double getDureeNextEvent(){
    double event = -Math.log(random.nextFloat())
            / getTauxGlobal() ;
    return event;
}
```

OPTIMISATION - CHERCHER LES VOISINS

9 distances possibles

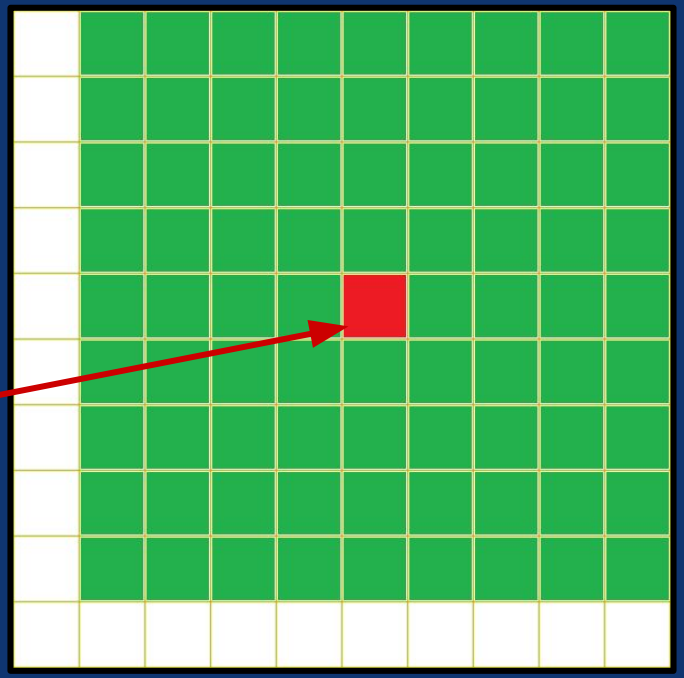


OPTIMISATION - CHERCHER LES VOISINS
TERRAIN DÉCOUPÉ

Prérequis :
Rayon de Compétition > 0 et <= 0.4

Les arbres se trouvant en :
- X >= 0.5 et X < 0.6
- Y >= 0.5 et Y < 0.6

Dans la “case” (5 ; 5)



OPTIMISATION - CHERCHER LES VOISINS
TERRAIN DÉCOUPÉ

Plus le rayon de compétition est petit, plus le terrain sera découpé :

SI (RC >= 0.1 && RC <= 0.4)
    ALORS Découpage en 10*10

SINON SI (RC >= 0.01 && < 0.1)
    ALORS Découpage en 100*100

ETC...

OPTIMISATION - CHERCHER LES VOISINS
TERRAIN DÉCOUPÉ

```
private final ArrayList<ArrayList<ArrayList<Arbre>>> tableauDivision = new ArrayList<>();
private int division = 1;
```

OPTIMISATION - CHERCHER LES VOISINS
TERRAIN DÉCOUPÉ

```
if (rayonCompetition > 0 && rayonCompetition <= 0.4) {
    double rayonCompetitionTemp = this.rayonCompetition;
    while (rayonCompetitionTemp < 1) {
        this.division *= 10;
        rayonCompetitionTemp *= 10;
    }

    for (int i = 0; i<division;i++) {
        tableauDivision.add(new ArrayList<>());
        for (int j = 0; j < division; j++)
            tableauDivision.get(i).add(new ArrayList<>());
    }
}
```

Comment découper le terrain

Initialiser le terrain

OPTIMISATION - CHERCHER LES VOISINS
TERRAIN DÉCOUPÉ

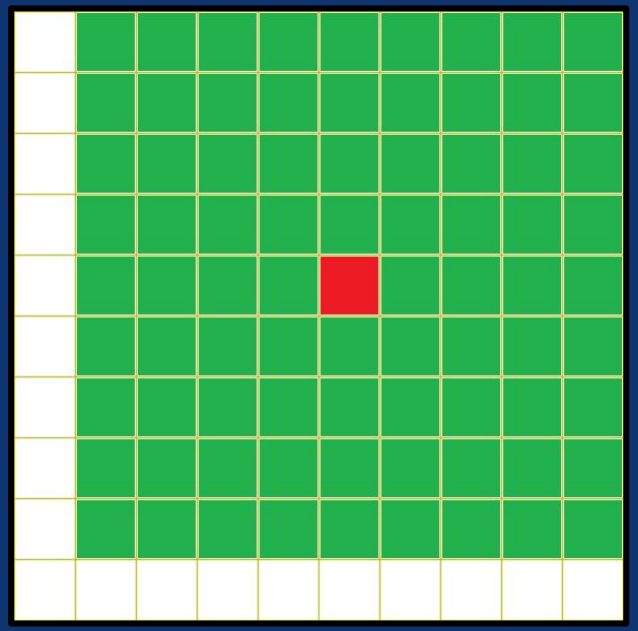
```
tableauDivision.get((int) (arbre.getX() * division)).get((int) (arbre.getY() * division))
```

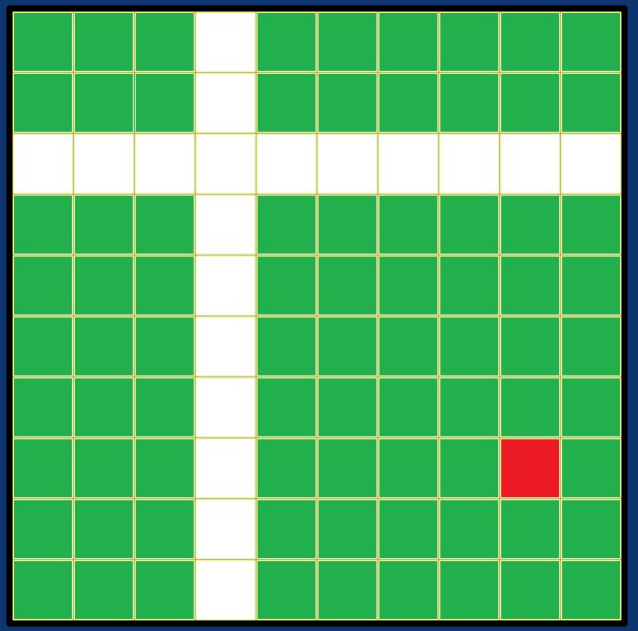
SI Division = 10 && (arbreX = 0.5 && arbreY = 0.68)
ALORS tableau.get(5).get(6)

SI Division = 100 && (arbreX = 0.5 && arbreY = 0.68)
ALORS tableau.get(50).get(60)

OPTIMISATION - CHERCHER LES VOISINS

TERRAIN DÉCOUPÉ





OPTIMISATION - CHERCHER LES VOISINS
TERRAIN DÉCOUPÉ

```
private void checkVoisinFast(Arbre arbreNouveau){
    double X = arbreNouveau.getX();
    int xmin = (int) ((X - rayonCompetition) * division);
    int xmax = (int) ((X + rayonCompetition) * division);
    int debordX;

    for (int i = xmin; i <= xmax; i ++) {
        debordX = 0;
        int indexX = i;
        if (indexX < 0) {
            indexX += division;
            debordX = 1;
        } else if (indexX >= division) {
            indexX -= division;
            debordX = -1;
        }
        ....
        for (Arbre arbreCourant : tableauDivision.get(indexX).get(indexY))
            checkInsideRayon(arbreNouveau, arbreCourant, debordX, debordY);
        }
    }
}
```

Initialisation variable

Vérification de “débordement”

Vérification si les arbres de la case (X, Y) sont biens des voisins

OPTIMISATION - CHERCHER LES VOISINS SANS DÉCOUPAGE

Notre division se fait en fonction du rayon de compétition, mais si ce rayon est > 0.4, nous testons tous les arbres du terrain.

Nous avons donc créer une autre fonction dans le cas ou nous n'avons pas besoin de découper le terrain.

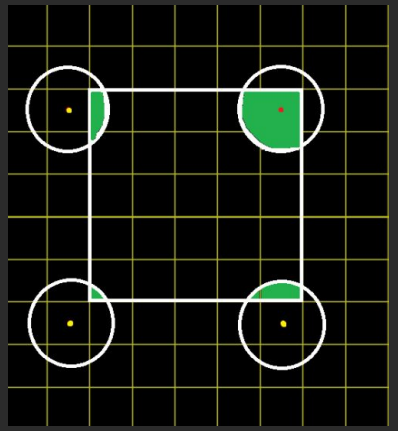
OPTIMISATION - CHERCHER LES VOISINS SANS DÉCOUPAGE

```
private void checkVoisinsSlow(Arbre arbre) {
   double rayon = rayonCompetition;

   for ( Arbre arbreCourant : list) {
       if (arbreCourant != arbre) {

           if (arbre.getX() + rayonCompetition > 1
                       && arbre.getY() + rayonCompetition > 1)
               checkInsideRayonQuatre(arbre, arbreCourant, -1, -1);

           else if (arbre.getX() + rayonCompetition > 1
                       && arbre.getY() - rayonCompetition < 0)
               checkInsideRayonQuatre(arbre, arbreCourant, -1, 1);

           else if ()....
```

Prenons le cas où le rayon de compétition dépasse du repère en x et y.

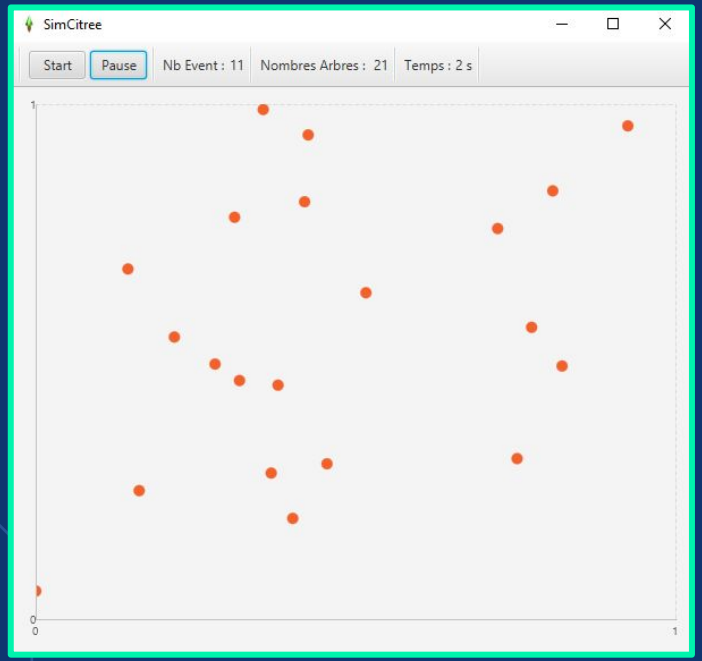
OPTIMISATION - CHERCHER LES VOISINS SANS DÉCOUPAGE

```
private void checkInsideRayonQuatre(Arbre arbre , Arbre arbreCourant , int debordementX, int debordementY) {

   double distance0 = Math.hypot(( arbre.getX()- arbreCourant.getX()), ((arbre.getY() - arbreCourant.getY() ));

   double distance1 = Math.hypot(( (arbre.getX() + debordementX) - arbreCourant.getX()), ((arbre.getY() + debordementY) -
arbreCourant.getY() ));

   double distance2 = Math.hypot(( (arbre.getX() + debordementX) - arbreCourant.getX()), (arbre.getY() - arbreCourant.getY() ));

   double distance3 = Math.hypot(( arbre.getX() - arbreCourant.getX()), ((arbre.getY() + debordementY) - arbreCourant.getY() ));


   double distancePlusPetite = distance0;

   if (distance1 <= distancePlusPetite)
       distancePlusPetite = distance1;

   if (distance2 <= distancePlusPetite)
       distancePlusPetite = distance2;

   if (distance3 <= distancePlusPetite)
       distancePlusPetite = distance3;

   if(distancePlusPetite < rayonCompetition)
       addEachOther(arbre,arbreCourant,distancePlusPetite);
}
```

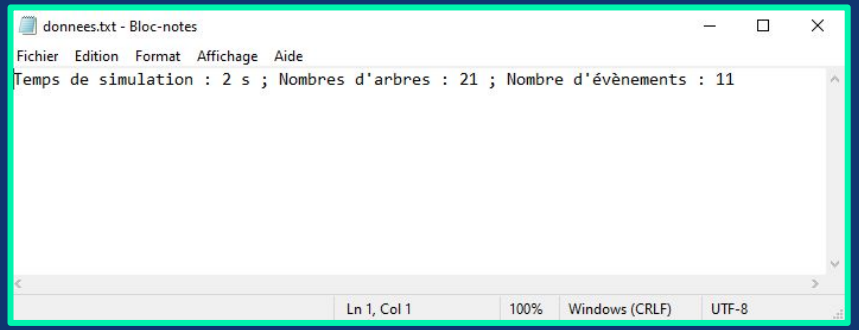


TYPE DE SORTIE DES DONNÉES

VISUELLE



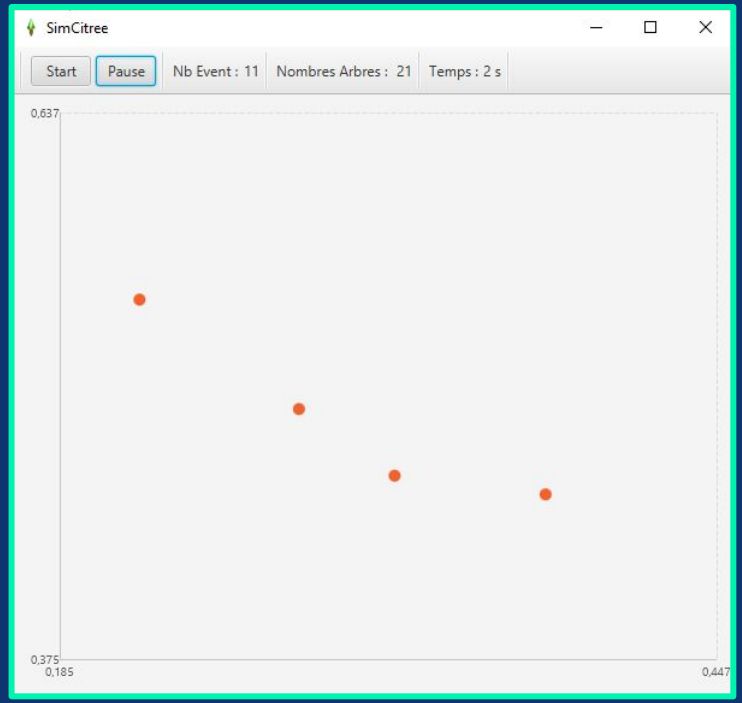
TEXTUELLE



TYPE DE SORTIE DES DONNÉES - VISUELLE

- Zoomer*
- Déplacer avec la souris*
- Réinitialiser le zoom en double cliquant*
- Mettre pause

*bibliographie jfxutils de Gillius



05

RÉSULTATS

INSTALLATION ET VALIDATION

INSTALLATION

- Lancer depuis Intellij
- Possibilité de créer un .jar du projet
- Dépend uniquement de Java

VALIDATION

- Nombres limités de test dûs à la nature du projet
- Nous avons tout de même vérifié certaines fonctionnalités

EVOLUTIONS POSSIBLES

LANGUAGE

- Migrer sur python
- Langage adapté au calcul scientifique

BIÔME

- Paramètre
- Influe sur la compétition
- 1 biome global ou plusieurs petits biomes

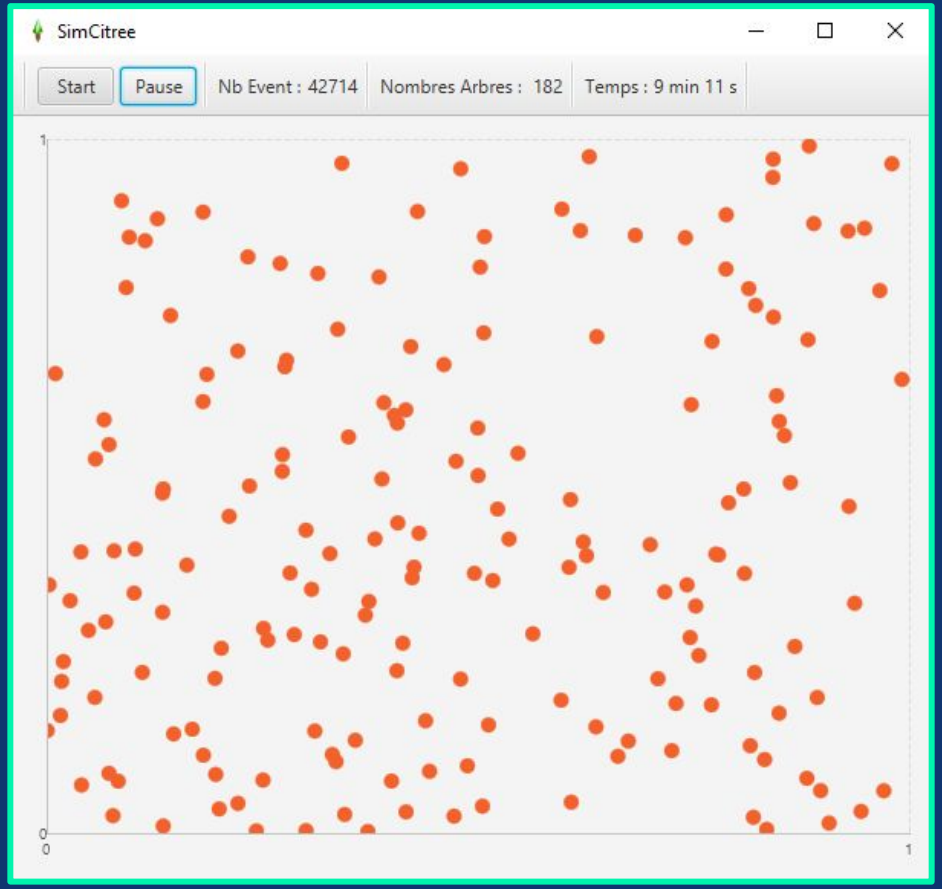
TYPE D'ARBRE

- Paramètre
- Influe sur le taux naissance/vie
- Plusieurs type d'arbre mélangés

06

CONCLUSION

CONCLUSION


SimCitree
Start
Pause
Nb Event : 42714
Nombres Arbres : 182
Temps : 9 min 11 s
1
0
0
1

DÉMONSTRATION

Rayon Dispersion	0.05	0.5	0.2
Rayon Compétition	0.05	0.3	0.1
λ birth	2.0	2.5	3.0
λ mort	0.5	1.0	0.01
λ compet	0.01	0.9	0.01
Nb Arbres	20	20	20

MERCI POUR VOTRE ATTENTION

Si vous avez des questions,
n’hésitez pas.