CAHIER DES CHARGES ET SPÉCIFICATIONS TECHNIQUES

LARVENGERS

BOYON Antonin
DEDRY Emeric
MERAND Yoann
TAOUD Noor

Encadrés par: **Nicolas SABOURET**

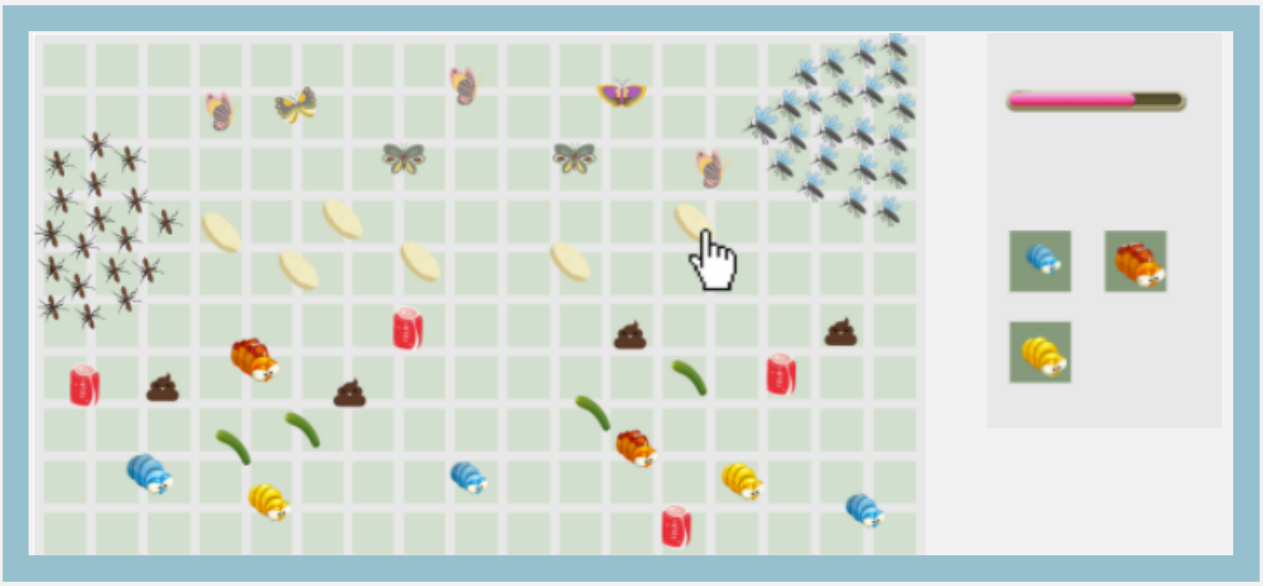
Introduction

L'objectif de ce projet consiste en la réalisation d'un jeu vidéo basé sur la stratégie en temps réel en essayant de reproduire le principe d'un jeu tel que Starcraft. Ce projet sert avant tout à se familiariser avec des bibliothèques JAVA utiles pour de la programmation concurrentielle et interfaces interactives. De plus, étant un projet de groupe cela nous permet aussi d'organiser et planifier notre projet par conséquent d'améliorer nos compétences en matière de gestion de projet.

Notre jeu "LARVENGERS" est donc un JCE (joueur contre environnement) où le joueur contrôle des larves. Il a pour objectif de les faire nourrir grâce à des ressources afin de les faire changer de forme pour atteindre la forme de papillon et remporter la partie. Cependant, il devra faire face à des ennemis qui chercheront à s'en prendre à ses cocon pour les empêcher d'évoluer.

Exemple d'interface du jeu

Résultat de l'interface attendu :



Interface du jeu finale

Analyse globale

Il y a 6 étapes majeurs dans la réalisation de ce jeu :

1) Architecture logicielle
2) Interface de jeu
3) Gestion des ressources
4) Déplacement des larves
5) Action des larves (manger, changer d'état, combat)
6) Ennemis

Nous les avons définis suivant un un ordre de priorité.
La fonctionnalité prioritaire est l'interface de jeu, qui permettra de tester les autres fonctionnalités. Elle n'est pas très difficile.

Ensuite nous avons la gestion des éléments (ressources, larves, ennemis). En effet les différents éléments possèderont beaucoup de caractéristiques communes. Cette fonctionnalité est vitale, les éléments étant le centre de notre jeu. Elle est assez difficile.

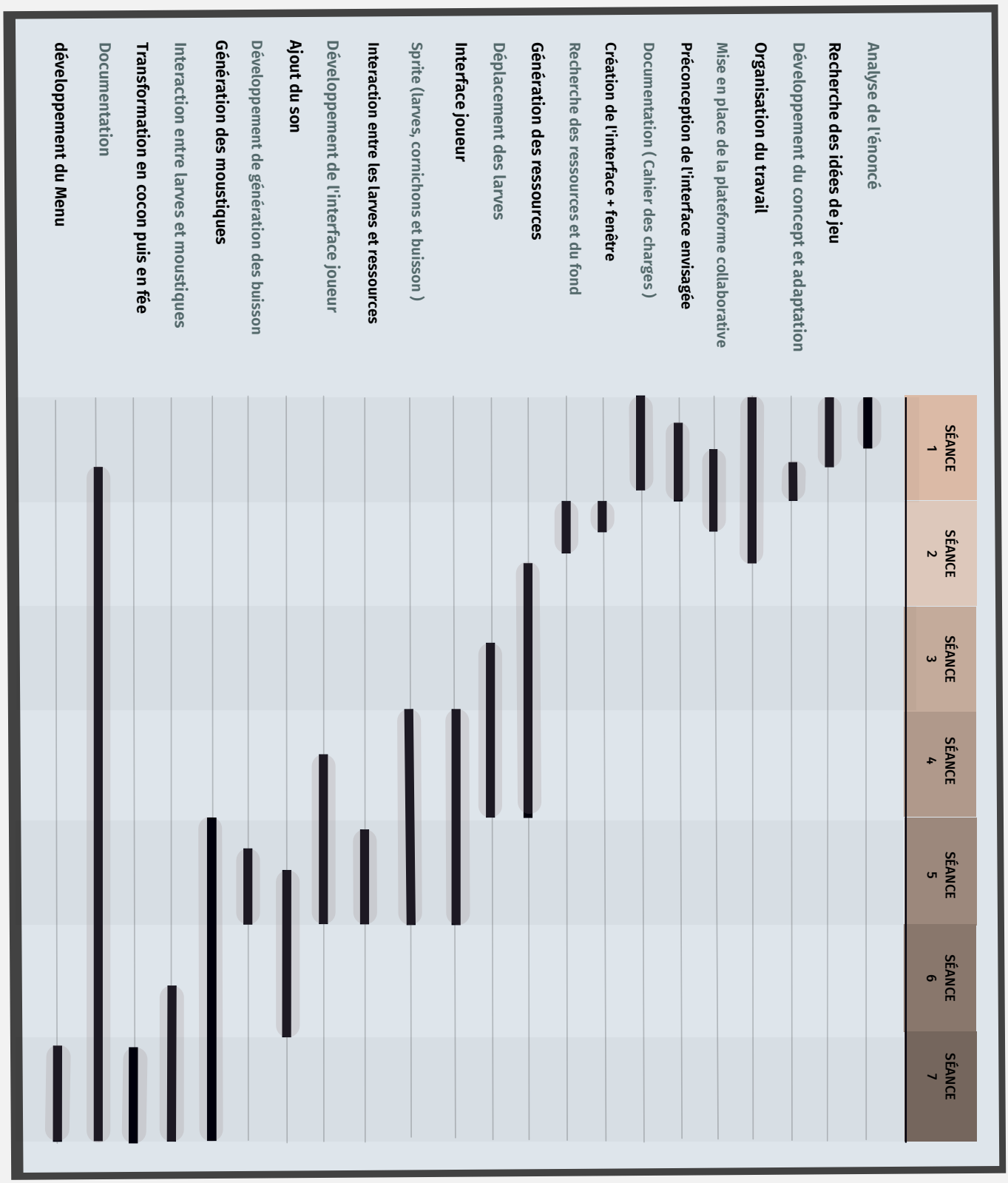
Puis nous pouvons trouver le déplacement des larves. De part sa dimension mathématique, cette tâche est très difficile. Elle est plus important que l'interaction avec les ressources ou les ennemis.

L'action des larves avec son environnement est également un point central de notre jeu. Cette étape est plutôt difficile.

Enfin, il y a les ennemis. C'est le dernier point de notre jeu. Le jeu étant jouable (mais moins intéressant) sans ses derniers, c'est l'étape la moins importante. Les ennemis partageront de nombreuses fonctionnalités avec les larves. Ce ne sera pas très difficile.

Plan de développement

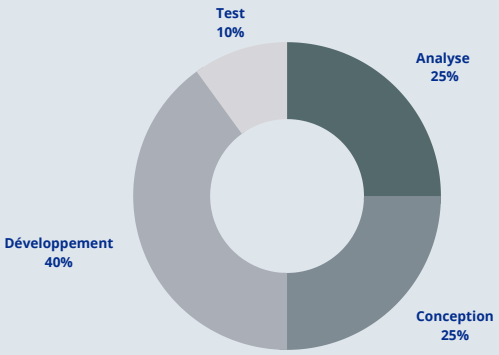
DIAGRAMME DE GANTT :



Plan de développement

DIAGRAMME DE TEMPS LIÉ À CHAQUE TÂCHE EFFECTUÉE

Création de l'interface

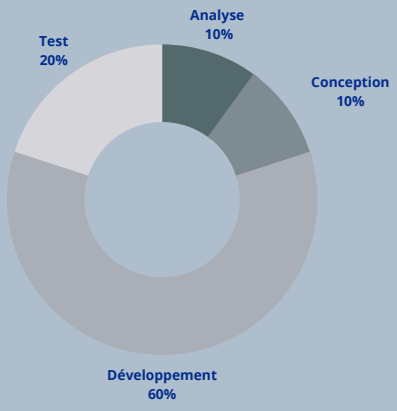


Temps consacré : 1h30

Recherche des ressources

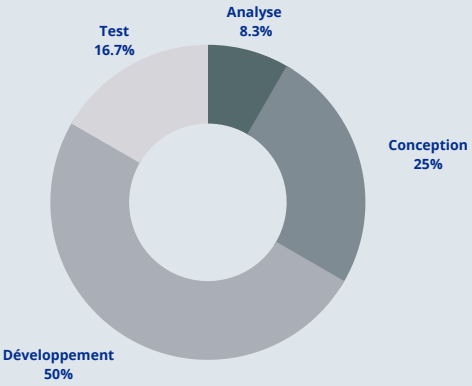
Temps consacré : 20min

Génération des ressources



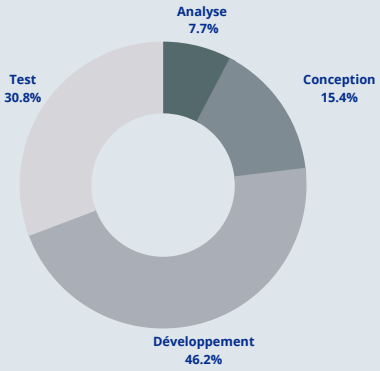
Temps consacré : 1h30

Déplacement des larves



Temps consacré : 2h00

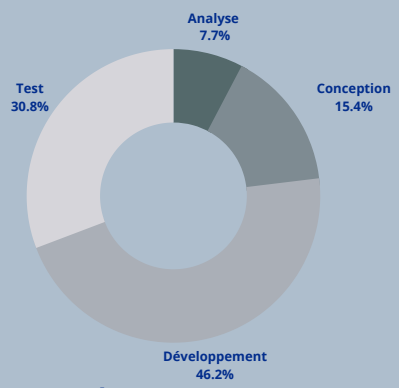
Interface joueur



Temps consacré : 1h30

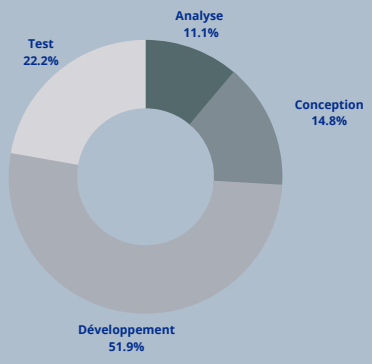
Intégration des sprites

(larves, cornichons et buisson)



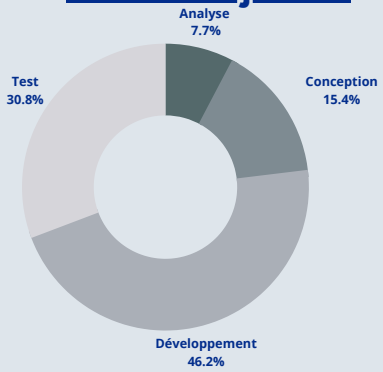
Temps consacré : 2h00

Interaction entre les larves et ressources



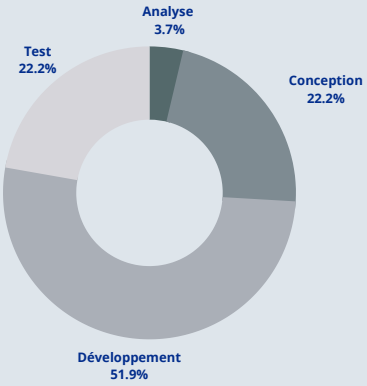
Temps consacré : 3h00

Développement de l'interface joueur



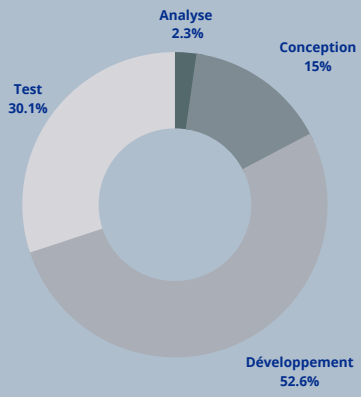
Temps consacré : 1h00

Ajout du son



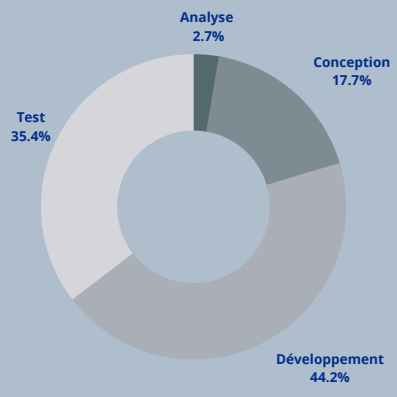
Temps consacré : 2h00

Génération des buissons



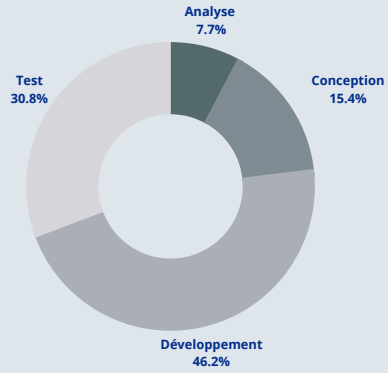
Temps consacré : 2h00

Génération des moustiques



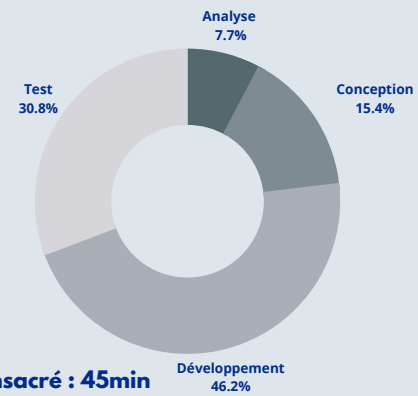
Temps consacré : 1h00

Interaction entre larves et moustiques



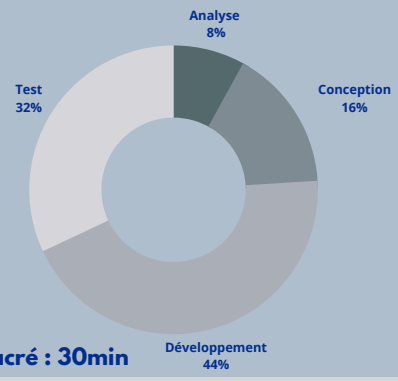
Temps consacré : 1h00

Transformation en cocon



Temps consacré : 45min

Transformation en fée

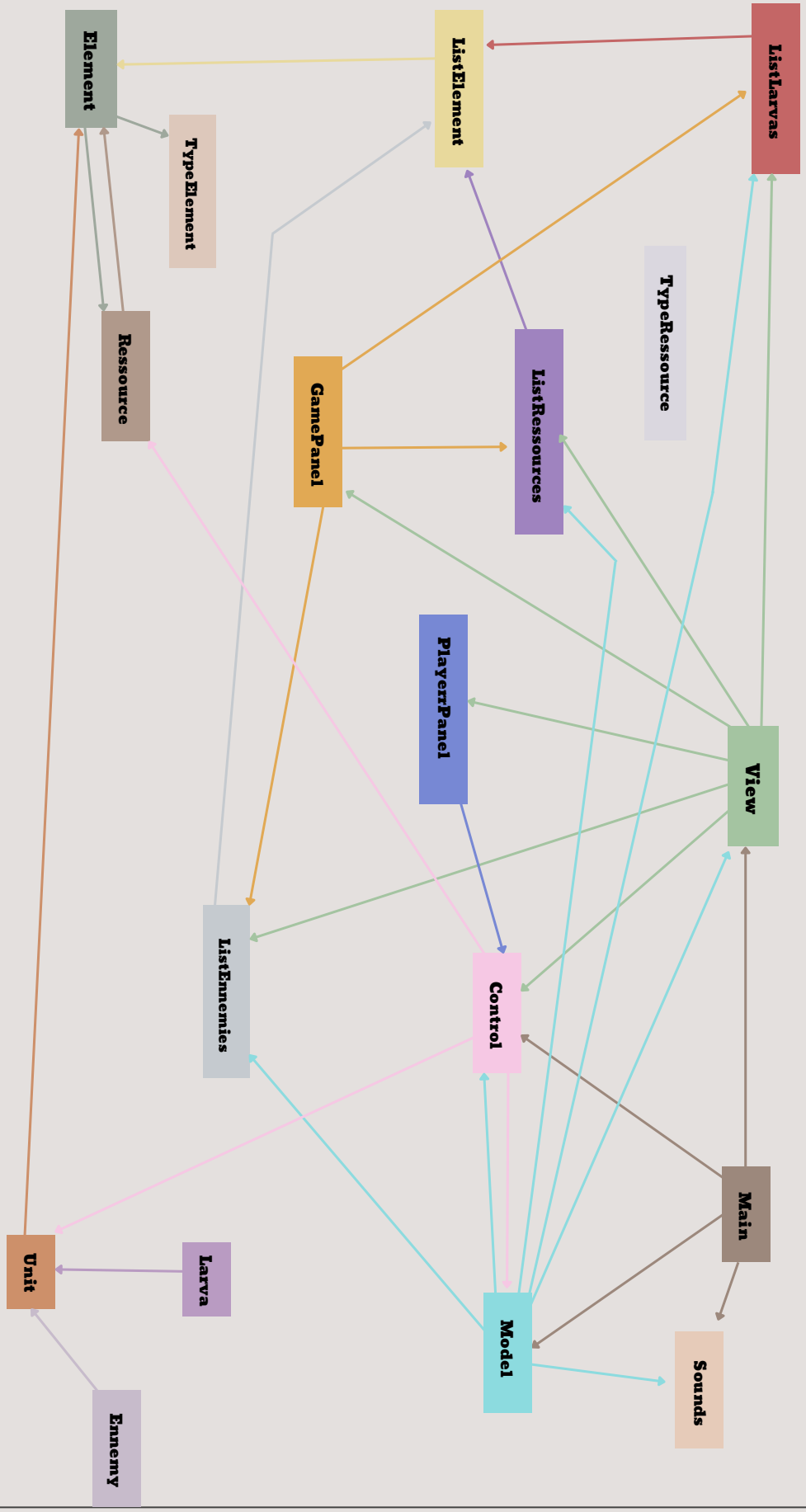


Temps consacré : 30min

Conception générale

Voici les principales fonctionnalités de notre application :

Architecture logicielle



Conception générale

Voici les principales fonctionnalités de notre application :

1 - Architecture logicielle

1. Element
2. MVC
3. Player
4. Ressource
5. Unit
6. Main

Tous les éléments sont stockés dans une liste. Cette liste possède un TypeElement définissant le type de la liste (Liste des larves, des ennenis ou de ressources).
La liste gère l'ajout de nouveau élément (différent en fonction du type de la liste) ou la position de ses éléments, afin de savoir si le joueur a cliqué sur l'un d'eux.

Le jeu possède une architecture MVC.

2 - Interface de jeu

1. Zone de jeu
2. Bouton d'actions cliquables
3. Panneau de contrôle (état et statistiques de la larve)

Tout d'abord, nous avons l'interface de jeu. Cette première étape est primordiale pour définir les dimensions d'affichage du jeu et organiser l'espace pour les différents éléments qui y seront apparents. Il y aura une première zone d'affichage pour le jeu et une deuxième correspondant à un panneau de contrôle pour le joueur où il pourra effectuer différentes actions. Cette première zone contient tous les éléments du jeu. La deuxième englobe des boutons d'actions et des informations concernant des éléments du jeu.

3 - Gestion des ressources

1. Génération de ressources aléatoires sur le plateau
2. 3 types de ressources

Deuxièmement, nous avons la gestion des ressources. Il y a trois types de ressources : Pickles, cocktails et poop. Les ressources apparaissent aléatoirement sur la carte (sauf dans la partie basse). l ne peut y avoir que 11 ressources sur le plateau. Elles sont apparentes sur la carte et pourront être consommées par les larves. Elles resteront présentes tant qu'elles n'auront pas été totalement consommées. Quand l'une l'est totalement, une autre apparait.

Conception générale

4 - Déplacement des larves

1. Déplacement des unités

Ensuite, il y a le déplacement des unités. Le joueur pourra cliquer sur la larve (unité) de son choix, et choisir où il souhaite la déplacer.

5 - Action des larves

1. Manger ressources et interaction avec
2. Se transformer en cocon

La quatrième étape est la configuration des actions des larves. Elles peuvent interagir avec les autres éléments sur la carte comme les ressources qu'elles peuvent manger. Pour évoluer en cocon, la larve devra consommer 10 cocktails et 10 pickles, ou un poop qui a 50% de chance de monter ses caractéristiques au maximum, ou sinon au minimum. Les larves peuvent se défendre de leur ennemis qui souhaite détruire les cocons en se placant entre le cocon et ces derniers.

6 - Ennemis

1. Génération des ennemis
2. IA
3. Interaction avec cocon et autres larves

Enfin, nous avons des ennemis. Ce sont des unités générées aléatoirement sur la carte qui vont cibler en priorité les larves qui seront en état de "cocon". Elles ont pour objectif de nous empêcher de gagner la partie. Quand un ennemie touche un cocon, il est détruit.

Toutes ces étapes représentent la manière dont nous évaluons leur importance.
Pour se faire, nous avons organisé notre code suivant le modèle MVC avec une première classe gérant l'affichage générale du projet, une autre correspondant à la gestion de l'état des éléments affichés et dernière s'occupant de toutes les interactions avec l'interface.

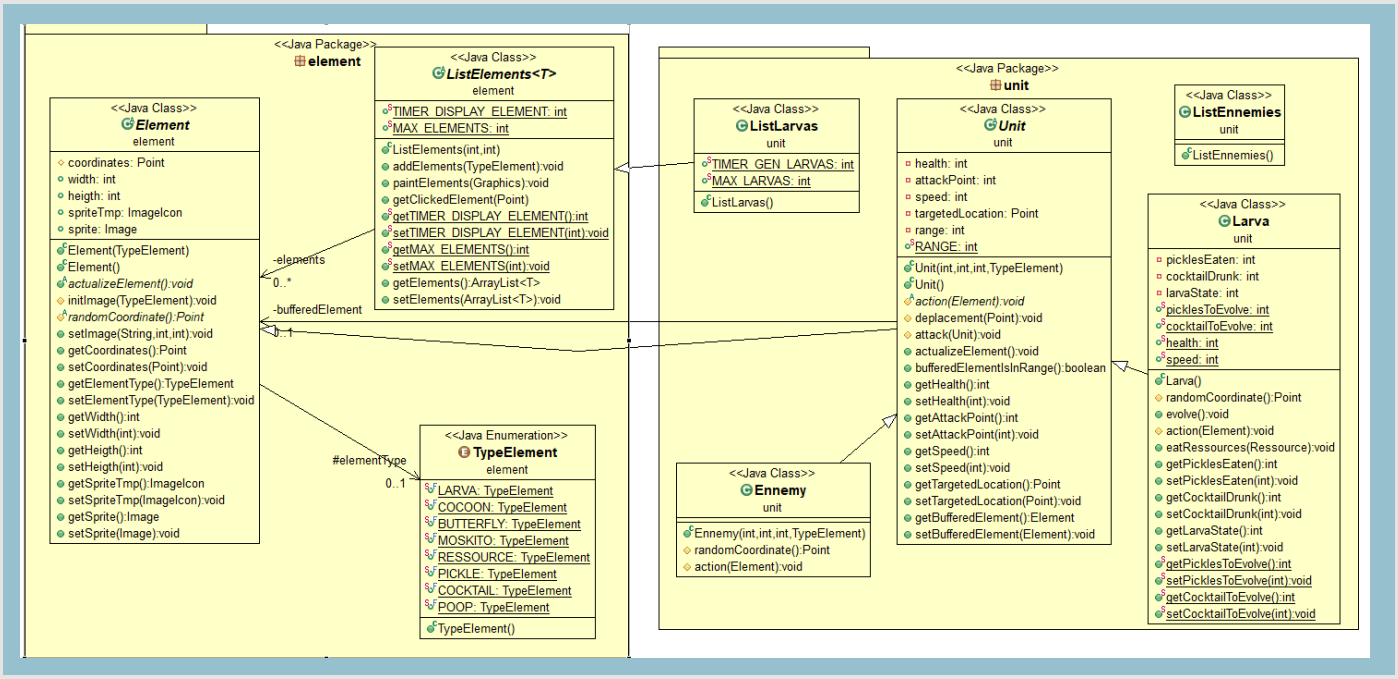
7 - Audio

1. Récupération des fichiers audios
2. Lecture des fichiers lors d'appels de méthodes

Certaines classes appellent les méthodes dédiés au son de façon statique afin de lire les fichiers audios.

Conception détaillée

1 - Architecture logicielle



ListElement est une liste de type abstrait "T" qui hérite d'Element. T est une sorte de simulation de Ressource ou d'Unit. ListElement implémente un thread pour générer des nouveaux Element continuelement.

Les éléments sont générés aléatoirement, les larves par exemple, sont générées dans le carré en bas à gauche.

Les éléments déplacables sont des Unit. Ils sont contenus dans une ListUnit. Element implémente un thread pour pouvoir effectuer des actions comme le déplacement.

Les unités partagent des attributs comme la vitesse (speed), la portée (range) mais également des fonctions : deplacement qui déplace l'élément à un Point donné en argument ou actualizeElement qui modifie l'élément en fonction des actions qu'il recoit.

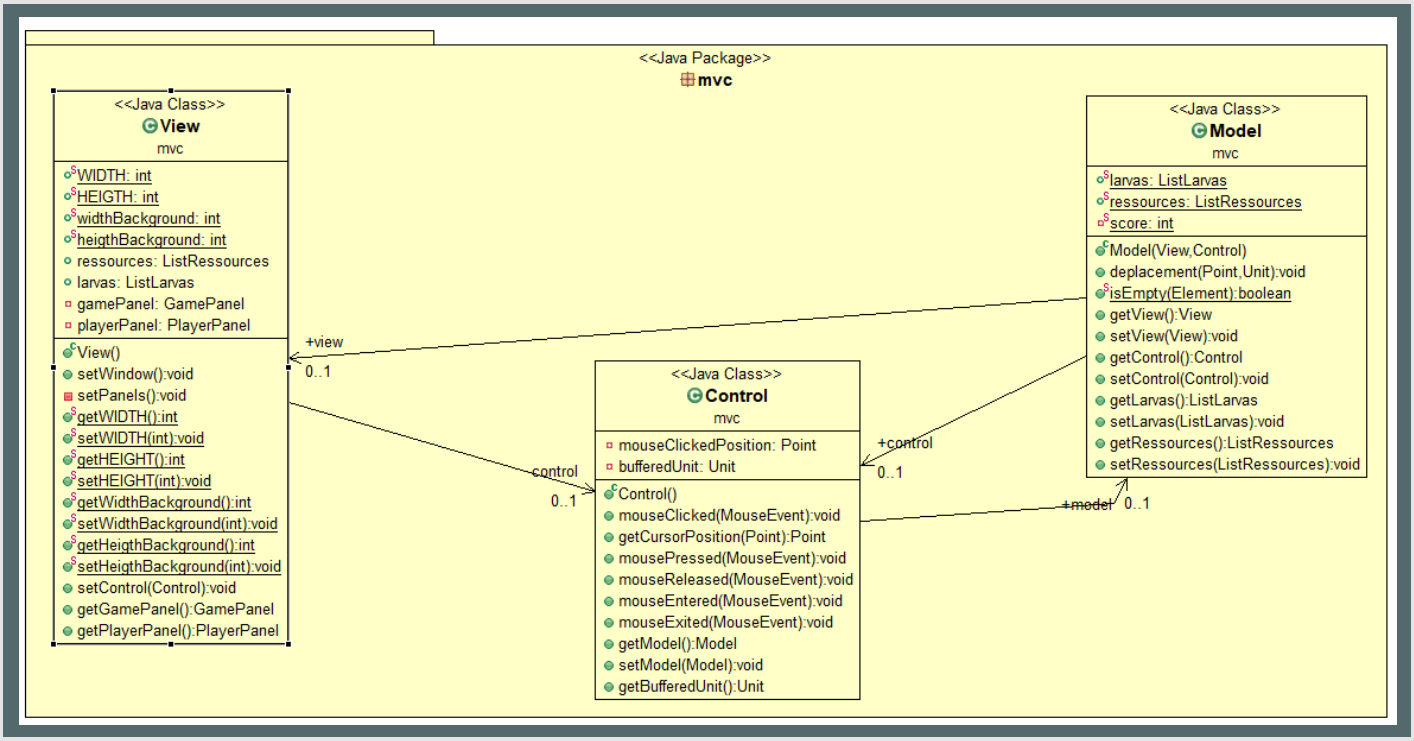
Enfin Larva et Ennemy, sont des Unit. Ils sont chacuns contenus dans une liste de leur type, ListLarva et ListEnnemies. Ces 2 listes héritent de ListUnit.

Les larves possèdent des actions spécifiques à elles. On peut retrouver eatRessources(Ressource) pour les larves.

Pour faciliter l'expérience utilisateur, tous les éléments sont encadrés par un carré noir représentant leur hitbox.

Conception détaillée

1.2 - Modèle MVC



Nous avons décidé de suivre le modèle MVC.

La classe "View" s'occupe principalement de récupérer les éléments à afficher dans la fenêtre. Les méthodes "setWidow" et "SetPanels" permettent de définir les dimensions ou autres caractéristiques importantes que doit prendre la fenêtre.

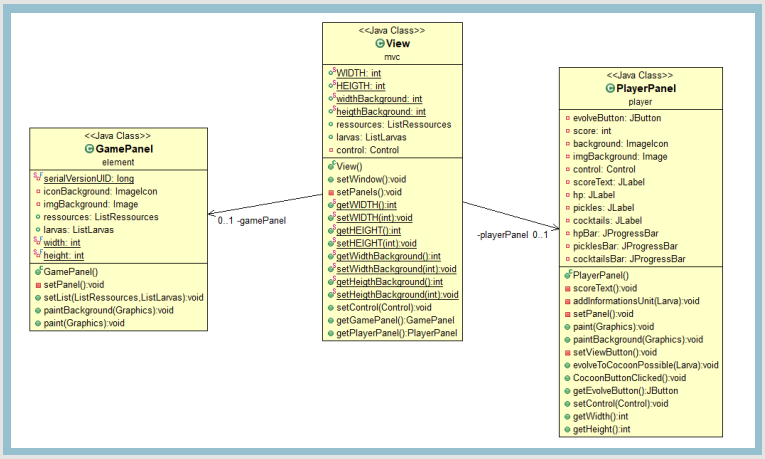
La classe "Model" vérifie l'état de nos éléments avec la méthode "isEmpty". Elle s'occupe de vérifier s'il existe un autre élément dans la zone de l'élément argument. Nous avons aussi la méthode "déplacement" qui déplace l'unité sélectionnée aux nouvelles coordonnées sélectionnées par le joueur.

La classe "Control" écoute les évènements. La méthode "mouseClicked" détecte un clic de souris et agit en conséquence suivant l'action que veut faire le joueur. Nous retrouverons la sélection d'une unité ou le déplacement de l'unité à de nouvelles coordonnées.

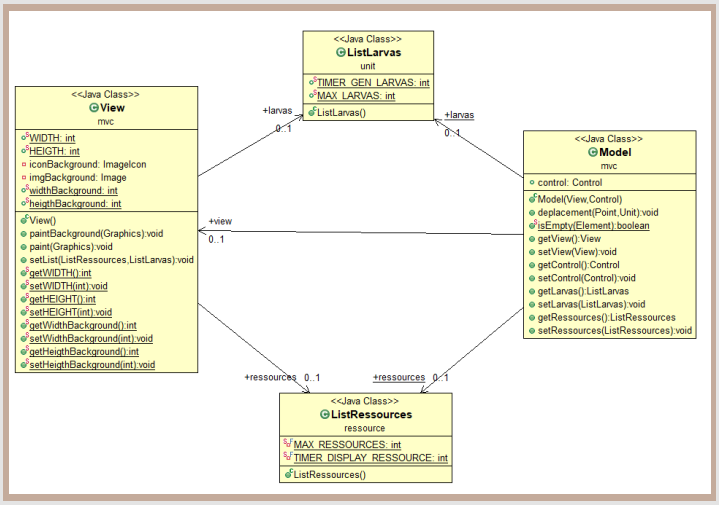
Conception détaillée

2 - Interface de jeu

Nous avons divisé notre fenêtre de jeu en deux : la partie de gauche contenant l'affichage des éléments du jeu et la partie de droite concernant le panneau de contrôle du joueur et l'affichage des statistiques de l'élément sélectionné.
L'ensemble de cette fenêtre est gérée par la classe View qui s'occupe d'initialiser les deux parties pour les afficher.

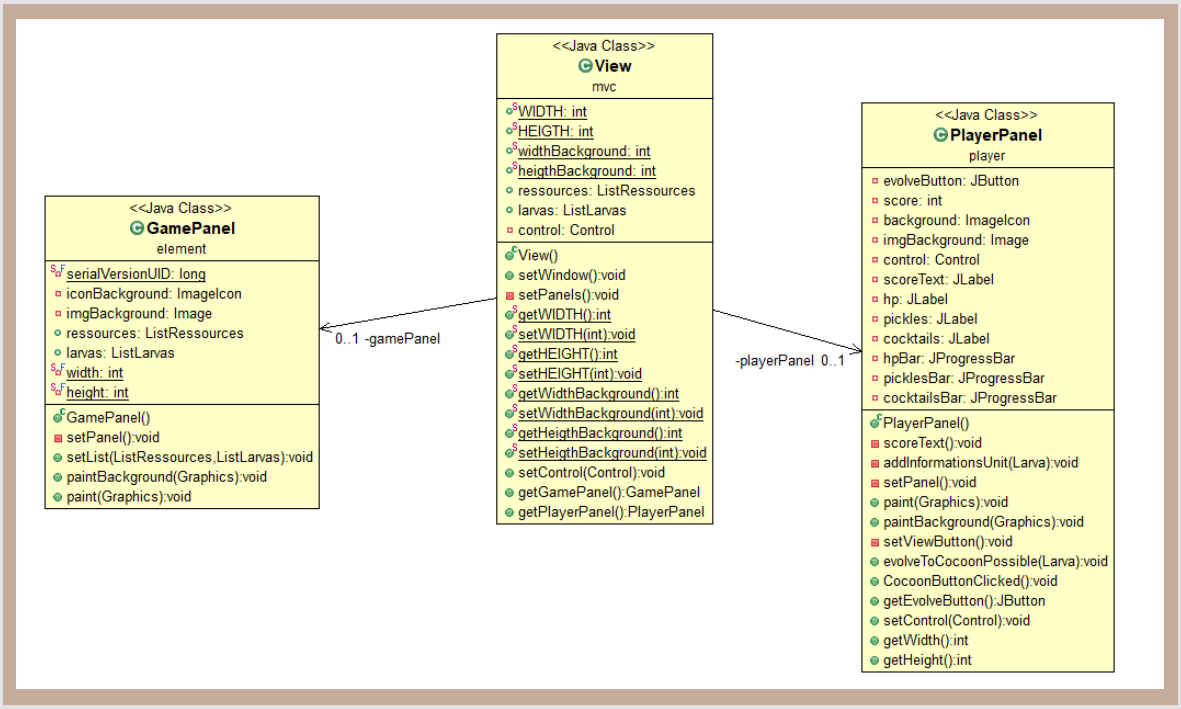


2.1 - Zone de jeu



Les classes "GamePanel" et "PlayerPanel" héritent de la classe JPanel. Chacune peut avoir son propre affichage.
La classe "GamePanel" possède deux attributs qui contiennent l'ensemble des éléments à afficher : "ressources" de type ListResources et "larvas" de type ListLarvas. Elles seront ensuite initialisées à l'aide de la méthode "setList". La méthode "paint", héritée de JPanel, permet d'afficher ses éléments en plus de la méthode "paintBackground" qui affiche l'arrière-plan du jeu.

2.2 - Bouton d'actions cliquables



Le panneau de contrôle du joueur possèdent plusieurs zones d'affichage.

Nous avons le bouton d'évolution en cocon "evolveButton" de type JButton.
La méthode "setViewButton" permet de définir l'emplacement de ce bouton sur le panneau.
Nous avons deux méthodes essentielles pour ce bouton : "evolveToCocoonPossible" et "CocoonButtonClicked".
La première méthode va nous permettre de vérifier si une larve sélectionnée a les prérequis pour pouvoir évoluer en cocon. Le prérequis rend le bouton cliquable.
En cas que clic, on va faire appel à la deuxième méthode. Cette dernière va faire évoluer la larve sélectionnée et changer son état pour qu'elle devienne un cocon.

2.3 - Panneau de contrôle

Tout d'abord, il y a le score du joueur. On utilise un compteur appelé "score" qui compte le nombre de larves qui ont pu évoluer au stade de papillons, grâce à la méthode "scoreText".

Enfin, nous avons l'affichage des statistiques de la larve sélectionnée. Nous avons utilisé des JLabel pour afficher les libellés des statistiques et des JProgressBar. Elles sont toutes initialisées avec une valeur minimale à 0 et une maximale à 100. <
"HpBar" , "PicklesBar" et "CocktailsBar" récupèrent respectivement la valeur de la santé, le nombre de cornichons mangés et le nombre de cocktails bus par la larve.Les barres de progression varient en récupérant ces valeurs.

Pour finir, la méthode "addInformationsUnit" va positionner les statistiques de la larve sur le panneau.

Conception détaillée

3.1 - Génération de ressources aléatoire sur le plateau

Les ressources sont générés aléatoirement sur le plateau.

Tout d'abord, leur position est générée de façon aléatoire sur presque tout le plateau. En effet, la partie basse est réservé à l'apparition de larves.
En fonction de la ressource, on va associer différentes images afin que les sprites de la ressources soient les bons.

Ensuite, leur type est également générée aléatoirement. Chaque ressource à autant de chance d'apparaitre sur le plateau.

3.2 - 3 types de ressources

Il y a trois types de ressources dans le jeu :
Les pickles : chaque buisson en contient entre 5 et 20. Cette ressource augmente le nombre de pickles mangés par la larve. Quand le nombre de pickle du buisson atteint 0, le buisson disparait.
Les cocktails : comme pour les pickles, chaque buisson en contient entre 5 et 20. La consommation de la ressource augmente le nombre de cocktails bu. Lorsque la quantité de cocktail du buisson atteint 0, il disparait.
Les poop : contrairement aux 2 autres ressources, sa quantité est toujours de 1. Manger un poop peut avoir 2 effets, chacun avec 50% de chance de se réaliser. Soit cela augmente les cocktails bu et la pickles mangés à 10 pour que la larves puissent évoluer en cocon, ou ça baisse ces deux ressources à 0.

Buisson de pickles

Buisson de poop

Buisson de cocktails

4 - Déplacement des larves

Les unités peuvent se déplacer grâce à la méthode deplacement(Point) de la classe Unit. L'unité va alors se déplacer jusqu'au Point passé en argument qui deviendra targetedLocation.
Ensuite la fonction actualizeElement va faire avancer l'unité, à chaque itération du thread, pour avoir un déplacement fluide, jusqu'au point targetedLocation.

Algorithme du déplacement fluide (en pseudo-code) :

Si vitesse>0 et pointArrivée non null
Alors distance = distance (coordoonéesDeLunite, pointArrive)

distanceAParcourir = distance/speed
Si distanceAParcourir < 1
Alors coordonnéesDeLunite = pointArrive
Sinon
coordoonéesDeLunit = coordoonéesDeLunit + distanceAParcourir

4.1 - Déplacement des unités

Pour l'ennemie, une IA définit la targetedLocation.
Pour la larve, le contrôle detecte si l'utilisateur a cliqué sur une larve à l'aide de la fonction getClickedElement(PointCliqueParUser) et va la sauvergarder dans bufferedUnit. Ensuite bufferedUnit va être déplacé au second clique de l'utilisateur.

Larve

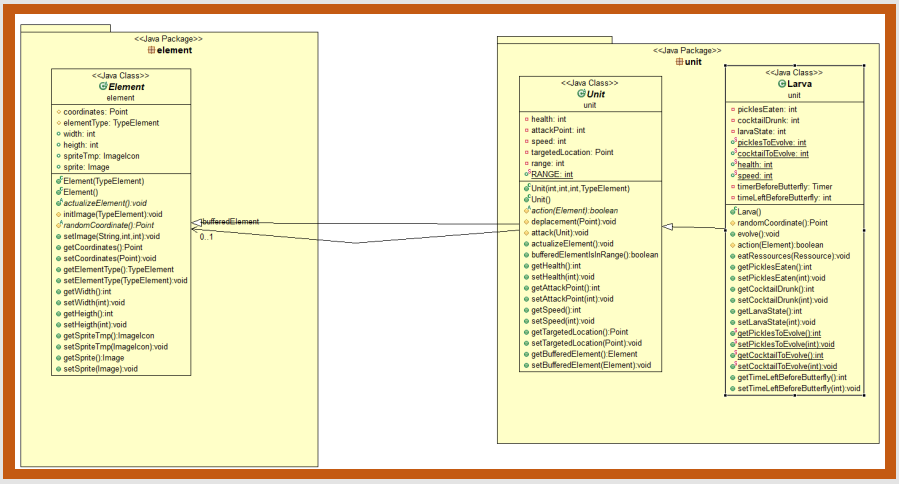
Cocon

Conception détaillée

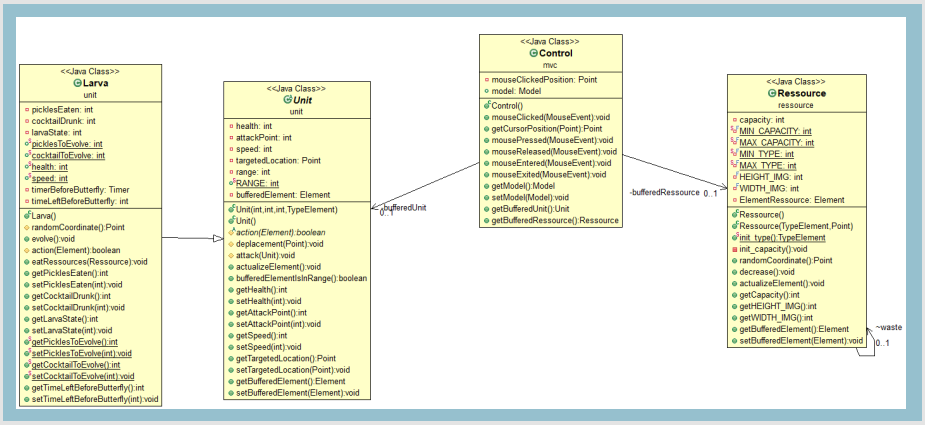
5 - Action des larves

5.1 - Manger des ressources et interaction avec

La classe "Element" comporte un thread qui permet aux éléments d'effectuer des actions. Ces actions sont définies dans les classes spécifiques des éléments avec la méthode "actualizeElement".
Pour les larves, la classe "Unit" comporte une fonction action qui effectue l'action spécifique de l'unité (manger pour la larve, viser un cocon pour les ennemies).



La fonction action vérifie si les conditions pour réussir l'action sont satisfaites. Pour la larve, elle vérifie que la ressources visée (stocké dans la variable bufferedElement lors du clic de déplacement) est à portée de la larve (variable portée, de base à 50 pixels).
Lorsque la larve a mangé une ressource, un temps d'attente de 1 seconde est lancé. Ainsi elle devra attendre que la seconde soit passée pour manger à nouveau une ressource.

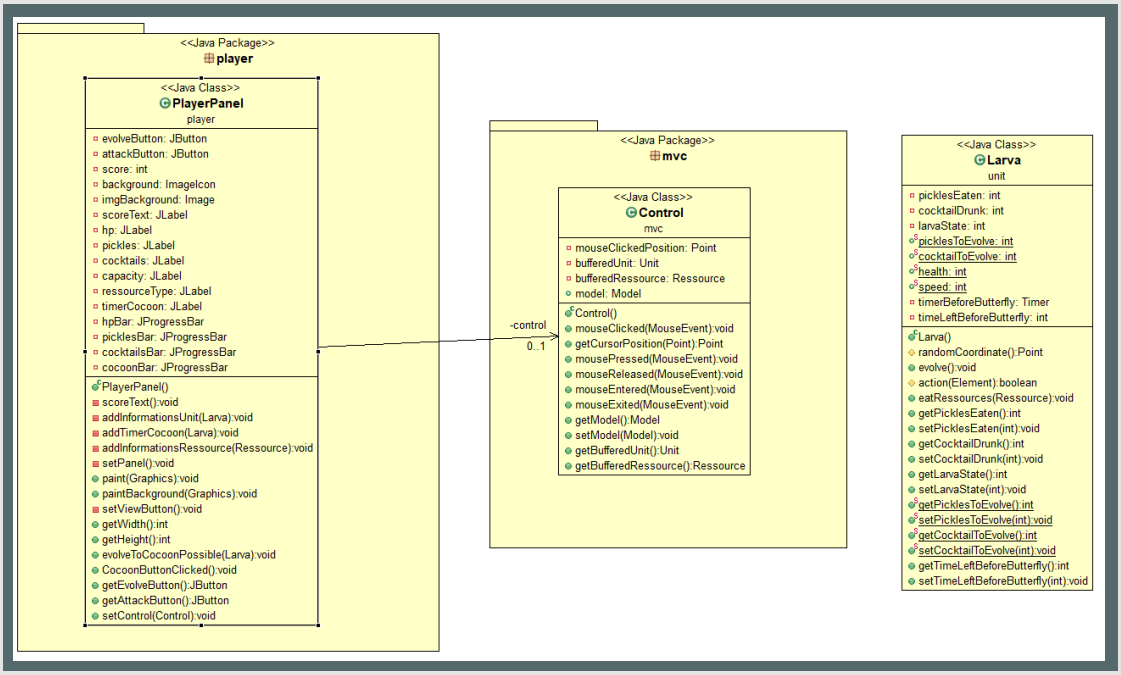


L'interaction des larves avec les ressources dépend avant tout des clics que fera le joueur. En effet, une fois qu'il a sélectionnée une larve il peut décider de la déplacer vers un point spécifique ou bien en direction d'une ressource. Maintenant qu'on a vu comment une larve mange, quand est-il vraiment de son interaction avec celle-ci ?
Une ressource possède avant tout une capacité et des coordonnées. Ses coordonnées sont définies à son initialisation de manière aléatoire dans les limites de la zone de jeu, puis sa capacité est elle aussi définie aléatoirement dans les bornes des constantes "MIN_CAPACITY" et "MAX_CAPACITY". Cette capacité va donc diminuer chaque fois qu'une larve va tenter de manger une ressource, elle va manger ainsi "prélever" une unité de capacité.

Conception détaillée

5 - Action des larves

5.2 - Se transformer en cocon



Concernant l'évolution d'un cocon, c'est au joueur de décider quand il souhaite faire évoluer sa larve vers son état suivant. Tout d'abord, dans la classe "PlayerPanel", nous avons un bouton nommé "evolveButton". Une fois cliqué, il lance le processus d'évolution si les conditions sont réunies. En effet, grâce à la méthode "CocoonButtonClicked", nous avons un évènement d'écoute sur le bouton qui vérifie que l'unité stockée dans le "Controller" est bien une larve et si cela est le cas, la méthode va le faire évoluer. Puis cette méthode est appelée dans l'autre méthode "evolvetoCocoonPossible" qui vérifie que les conditions sont réunies pour d'abord pour rendre cliquable le bouton et ensuite lance son évolution si le bouton est cliquée. De plus, on a une progressbar nommée "CocoonBar" qui montre l'avancée de l'évolution.

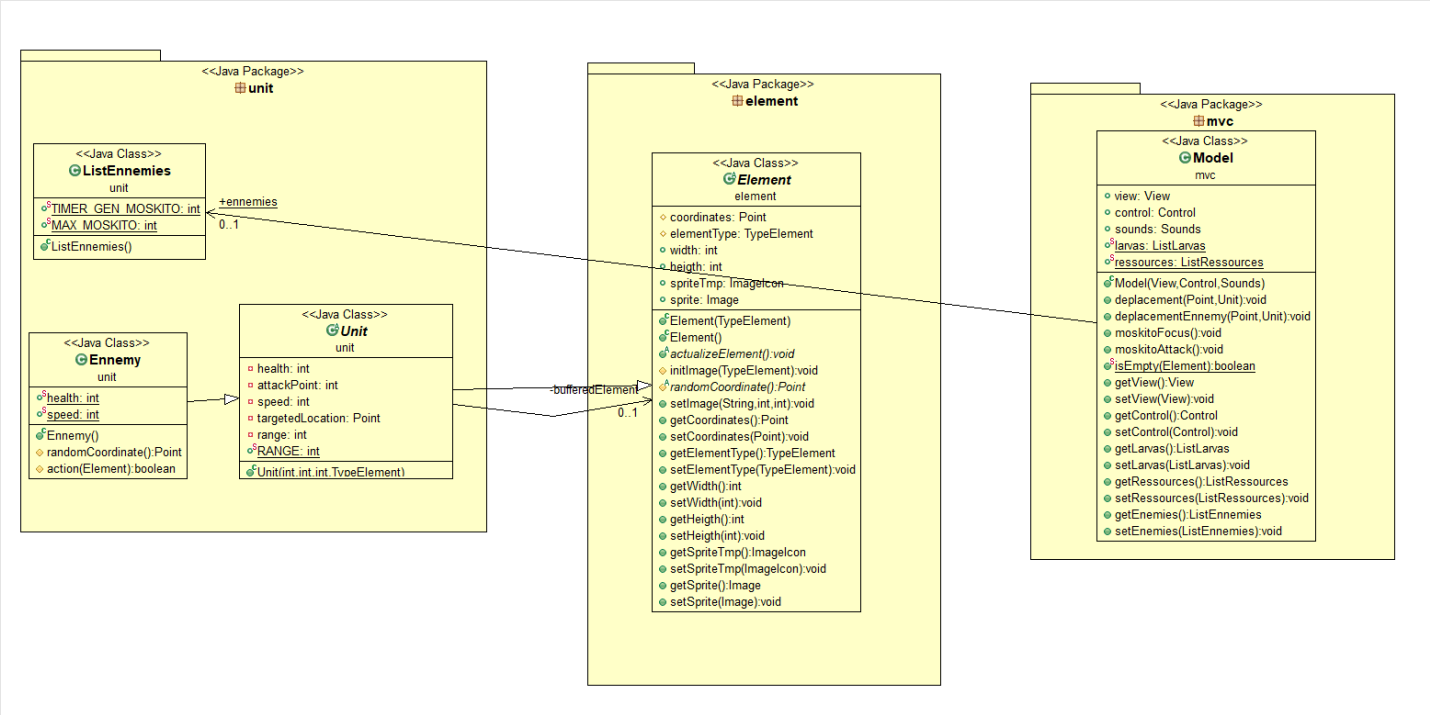
Au niveau de la classe "Larva", c'est la méthode "evolve" qui gère tous les changements c'est-à-dire de changer son état, son type, ses statistiques mais aussi de déclencher un "Timer" pour que l'utilisateur puisse avoir une progression de l'évolution. Nous avons deux attributs pour cela : "timerBeforeButterfly" de type Timer et "timerLeftBeforeButterfly" de type int. Ce dernier va varier de 0 à 59, ce qui représente le temps nécessaire pour que le cocon éclose en papillon, tandis que le premier lui va s'occuper de répéter l'incrémentation de ce int tant qu'il est inférieur à 60. Une fois le temps atteint, on arrête le timer puis on fait passer le cocon au stade de papillon et on change ses statistiques.

Conception détaillée

6 - Ennemis (moustiques)

6.1 - Génération des ennemis

ListEnemy est une liste de type abstrait qui hérite de ListElement.
L'affichage des moustiques se fait de la même manière identique à celles des larves.
De plus, lorsqu'un moustique apparaît il ne peut pas y avoir un autre moustique. Le moustique suivant est généré.



6.2 - IA et Combat

Model implémente un thread dans lequel est appelé une méthode dédié au déplacement des moustiques (moskitoFocus()) lorsqu'un cocon apparaît. Il récupère la position du dernier cocon généré et se déplace vers celui-ci.
Le moustique récupère donc la position des larves passe de LarvaState = 0 à LarvaState = 1,
Une fois, cela fait, il récupère leurs cordonnées et se déplacent vers eux.

Dans ce même thread, il y a une méthode dédié au combat (moskitoAttack()) qui vérifie pour chaque larves/cocons et chaque moustiques si ils rentrent en contact avec une méthode dédié (unitIsInRange(Unit u)). Si c'est le cas, le moustique ainsi que la larve/cocon meurent.

Moskito

Conception détaillée

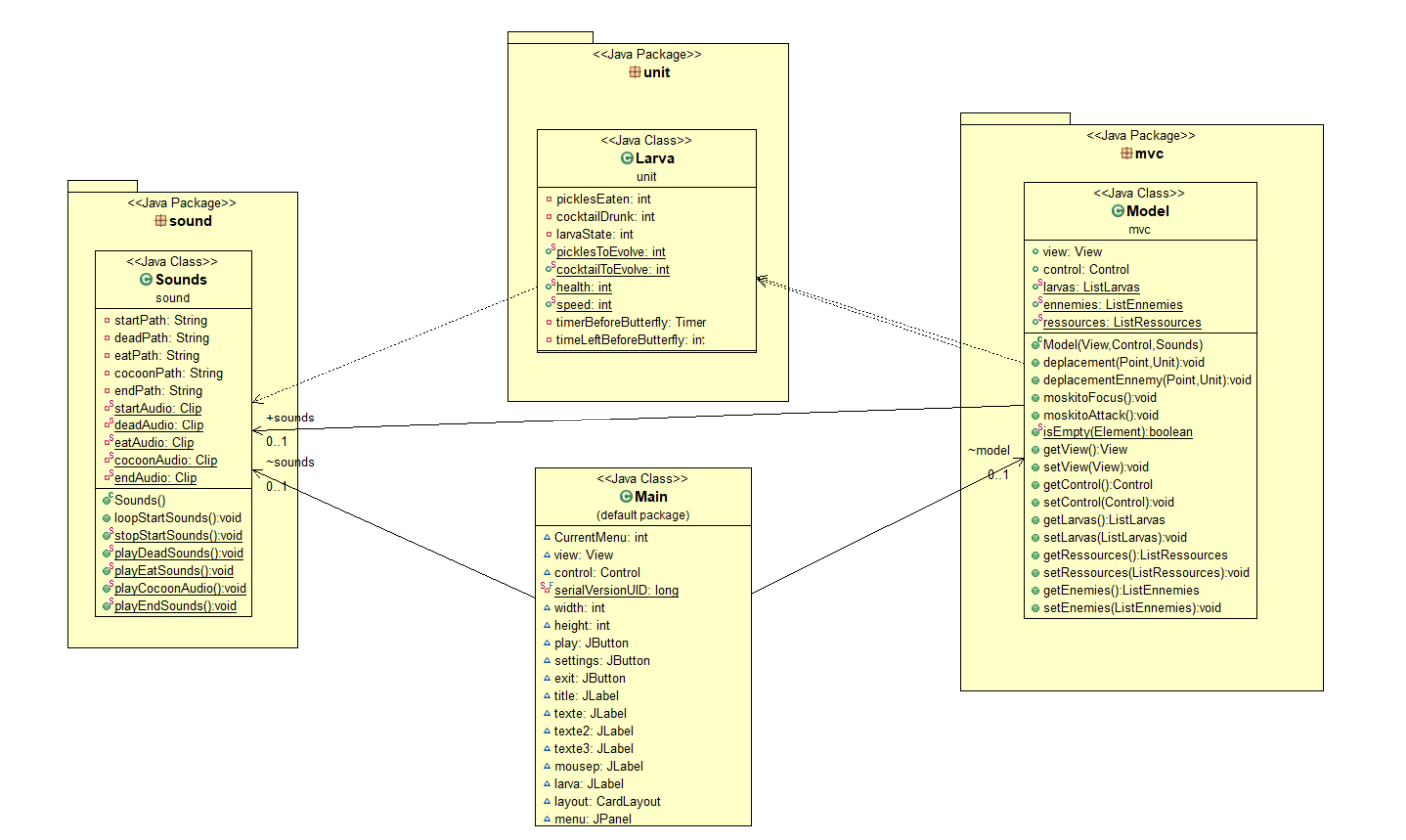
7 - Audio

Lecture des fichiers audio lors de la partie de jeu

La classe sounds permet de lire les fichiers audios lorsqu'une action est faite.

Nous avons ici plusieurs audios pour des actions différentes.
StartAudio : est le thème principal du jeux qui se lis en boucle et pendant toute la durée de la partie.
DeadAudio : l'audio qui se lance lorsqu'une larve meurt
EatAudio : se lance lorsqu'une larve mange une ressource
CocoonAudio : se lance lorsque la larve se transforme en cocon
EndAudio : se lance lorsque l'utilisateur a gagné le nombre de points nécessaire

Pour récupéré le chemin des fichiers audio, nous utilisons getAbsoluteFile.
Avec l'aide d'un try catch, on vérifie qu'il n'y a pas d'erreur dans la lecture des fichiers audio.



Conception détaillée

8 - Menu

Ce Menu permet à l'utilisateur lancer sa partie en cliquant sur le bouton Play, avec une brève explication affichée à l'écran .
Dans la classe Main qui hérite de JPanel, on a rajouté un JPanel contenant des JLabel et des boutons.
En cliquant sur "Play", on lance la partie grâce à:

```
sounds = new Sounds();
view = new View();
control = new Control();
model = new Model(view, control, sounds);
sounds.loopStartSounds();

view.setControl(control);
view.addMouseListener(control);
```

En détruisant l'ancienne JFrame du Menu.

JFrame du Menu:

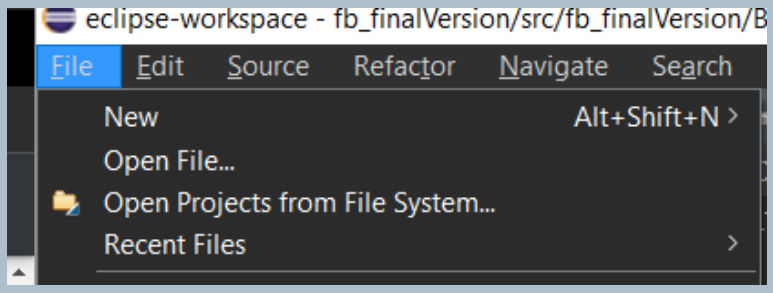
Résultat

Comme atteste cette capture d'écran, voici le résultat de notre jeu actuellement. Au niveau de la zone de jeu, vous observez des contours noirs autour de chaque élément. Cela correspond pour l'instant à sa "hitbox". De plus, tous les types de ressources mais aussi les états d'une larve sont affichées. Sur la partie du panneau de configuration, il y a beaucoup d'informations qui sont affichées notamment celle d'une ressource sélectionnée mais aussi celle d'une larve sélectionnée. Le score quant à lui correspond au nombre de larves qui ont éclos en papillon.

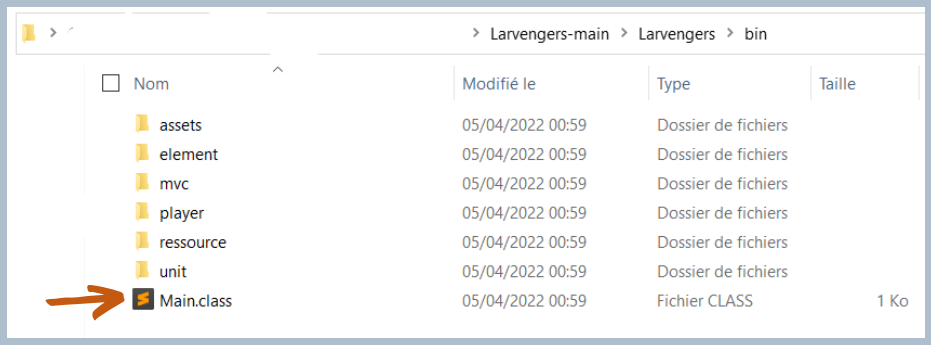
Documentation utilisateur

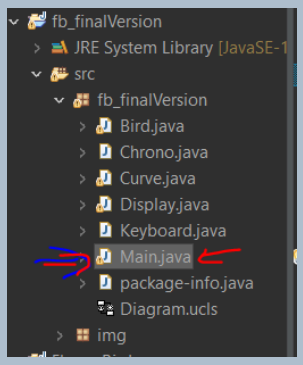
Si vous êtes un utilisateur de ce jeu,
vous devez avoir quelques prérequis tel que : Java avec IDE.
Voici un tutoriel afin de pouvoir exécuter le projet :

Ouvrez le dossier du projet:

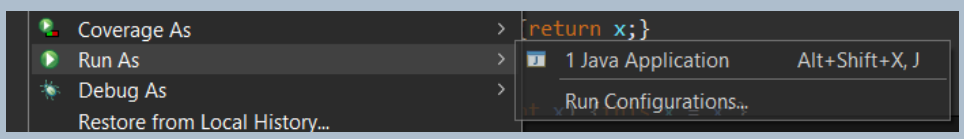


Séléctionnez la classe Main





Cliquez droit, ainsi cliquez sur Run as 1 Java Application



Vous pouvez également sélectionner le fichier.jar mis dans le dossier.
Double cliquez dessus et le jeu va apparaître .

Documentation utilisateur

Contrôle de base

Cliquer sur une larve et sélectionner l'endroit où vous souhaiter la déplacer.
Cliquer sur un buisson afin d'en manger une ressource. La larve va continuer à manger les ressources du buisson jusqu'à qu'il soit vide ou que vous lui donnez une autre action à réaliser.
Lorsque les barres cocktails et pickles de la larve sont pleines, vous pouvez la transformer en cocon avec le button 'cocoon mode'.

Documentation développeur

La méthode main se trouve dans le package par défault.

La classe "View" hérite de la classe JFrame provenant de la bibliothèque Java Swing. De ce fait, elle représente la classe "mère" de notre affichage. Elle possède deux attributs héritant de JPanel : "GamePanel" et "PlayerPanel". Elles représentent les parties scindées de notre fenêtre.

Les classes intéressantes et compliquées sont (dans l'ordre) Element, Unit, Larva, Ennemie, Ressources, ListElement, ListLarvas, ListEnnemies, ListRessources. En effet certaines classes héritent d'autres, il est donc intéressant de voir les classes 'mères' en premier. D'autres partagent beaucoup de fonctionnalités comme Larva et Ennemie.

Les constantes utiles à modifier sont la table des fenêtres :

Les constantes des classes liées à l'affichage qui peuvent être utiles à modifier sont les *width* et les *height* de chacune des classes vu qu'elles correspondent à leur dimensionnement.
Plus précisément pour la classe "PlayerPanel", il pourrait être intéressant de modifier les constantes liées au positionnement des éléments de ce panneau (les coordonnées en X et Y).

Et pour les élements : width et heigth pour la taille des sprites, speed pour le déplacements, range pour l'intéraction avec les autres éléments speed pour la vitesse de déplacement
Pour les larves : picklesToEvolve et cocktailToEvolve pour les conditions avant l'évolution en cocon et TimeBeforeButterFly pour le temps de mode cocon avant l'évolution finale.

Il peut être intéressant d'implémenter un système de collision pour le déplacement et la génération des ressources.
Amélioration des hitbox : il faut cliquer sur l'image (et pas les pixels transparants autour) pour sélectionner l'élement.

Conclusion

Nous avons réalisé les 6 étapes majeur :

- Architecture logicielle
- Interface de jeu
- Gestion des ressources
- Déplacement des larves
- Action des larves (manger, changer d'état, combat)
- Ennemis

Les principales difficultés étaient :

- Type abstrait de données

Nous avons utilisé de nombreux héritages afin de garder une certaine logique de réutilisation des fonctions génériques

- Déplacement

Implémentation de la notion de buffer pour avoir la larve 'active' et la ressource ou le point visé.
Calcul de la distance à parcourir avec la formule d= t/v

- Intéraction avec les ressources

Parcours de toutes les ressources et calcul de la distance à la larve

- Interface utilisateur

Emeric

- Génération aléatoire des éléments

Recherche de la présence, ou non, d'élément dans la futur zone de l'élément, mais cela n'est pas optimisé (recréation d'un objet pour retester jusqu'à trouver).

Nous avons appris à utiliser les threads, permettant d'avoir des actions visuels en temps réel.
Egalement les types génériques. Ils aident à factoriser le code et à créer des fonctions réutilisables.

Perspectives

Pour améliorer le jeu, il semble essentiel d'améliorer l'aspect graphique général du jeu.

Du côté technique, on peut imaginer un système de collision qui nous permettrait d'avoir un réel système de combat en les larves et les moustiques. Cela améliorerait aussi les interactions entre les larves et les ressources ainsi qu'entre les ennemis et la cocons.
Pour améliorer les collisions, celle des hitbox est nécessaire.
Le déplacement pourra utiliser ce système, et l'algorithme de Dijkstra par exemple, afin de faire un déplacement réaliste, évitant les obstacles.

Du côté du gameplay, un plus poussé améliorera nettement le jeu. Une utilité aux fées, une variété d'ennemis, des batiments pouvant améliorer des larves, d'autres ressources, des évenements aléatoires, ...

Conclusion personnelle

BOYON Antonin

Manager une équipe n'est pas simple, mais c'est très enrichissant. Organiser le travail en fonction des préférences de chacun est plus agréable pour tout le monde, et plus formateur.
Réaliser des algorithmes assez complexes comme le déplacement (et son système) est quelques choses d'assez intenses mais très satisfaisant. J'aime bien faire de l'IHM, mais je sais que je n'en ferais pas toute ma vie.

MERAND Yoann

Dans l'ensemble, ce projet a été particulièrement formateur pour moi, dans le sens où j'ai pu à nouveau expérimenter le travail de groupe. J'ai également pu découvrir de nouvelles librairies issues du langage JAVA telles que JAVA Sound ou JAVA Swing que je n'avais jamais utilisées avant ce cours, ce qui m'a permis de renforcer mes compétences et de gagner en capacité d'autoformation. Outre le côté développement de ce projet, nous avons également pu explorer certaines notions de l'ordre de la gestion de projets informatiques notamment côté conception, planification et répartition des tâches ce qui nous a permis d'être globalement efficaces. Enfin, je me suis senti très investi dans la réalisation des sprites du jeu, je suis assez satisfait du résultat sur ce point puisque finalement j'ai pu faire appel à ma créativité tout en créant quelque chose à l'image de notre groupe.

TAOUD Noor

Ce projet était très enrichissant pour nous tous, dans la mesure où ça représentait une approche concrète du métier de développeur.
Cela m'a permis d'apprendre davantage l'architecture logicielle des jeux. J'ai ainsi pu approfondir mes connaissances en informatique notamment en Java SWING, à la fois en effectuant mes tâches individuelles mais aussi grâce à l'échange dans le groupe.
Les contrainte les plus rencontrées étaient surtout des contraintes organisationnelles. Pour réussir à organiser ce projet de façon à ce qu'il soit modulaire tout en répartissant les tâches entre les différentes personnes a été certainement un vrai défi. Bien qu'on ait des idées qui divergent, on a réussi à collaborer ensemble de façon à nous entraider les uns les autres (en terme de backend, front end et même documentation).
Ce fut une très bonne expérience à programmer et à apprendre en groupe, ce projet est à mon gout une réussite, j'en suis fière. Nos principaux objectifs ont été atteints.

DEDRY Emeric

Cela a été un projet plutôt instructif, le fait de découvrir de nouveaux aspects de la programmation en langage JAVA m'a permis de consolider un peu plus mes notions dans ce langage. Travailler à 4 sur un projet a été une bonne expérience, j'avais déjà réalisé par le passé un projet avec une équipe beaucoup plus conséquente en effectif donc cette expérience ne m'a pas tellement plus déstabilisé que cela. Je suis plutôt satisfait du travail que j'ai pu fournir mais je reste quand même légèrement déçu envers moi-même de ne pas avoir pu développer des fonctionnalités auxquelles nous avions pensé.