

Le Verger

SAE R2.01/R2.02

LEGG Jimmy - PINHEIRO Nathan

02/06/2023

Sommaire

Introduction :	4
Présentation du client et de son besoin :	4
Présentation de la stack technique :.....	4
Version 1 : Des arbres fruitiers : Toujours gagnant !	5
Description :.....	5
Modifications :	5
Menu :	5
Lobby :	5
Jeu :	5
Jeux d'essais :.....	6
Version 2 : Des paniers pour la cueillette toujours gagnant !	8
Description :.....	8
Modifications :	8
Les paniers :.....	8
Jeux d'essais :	8
Version 3 : Apparition du corbeau : Gagnant ou Perdant	9
Description.....	9
Modifications :	9
Le puzzle :	9
Jeux d'essais :	9
Version 4 : Possibilité de choisir le fruit à récolter & bonus possible avec 2 fruits (début interactivité) :	10
Description :.....	10
Modifications :	10
La gestion de la collecte :	10
Le dé :	10
Jeux d'essais :	10
Version 5 : Choix du panier & pioche du puzzle : pour plus de créativité et de motricité !	13

Description :	13
Modifications :	13
Les paniers :	13
Intérieur des paniers :	13
La pioche de puzzle :	13
Le drag and drop :	14
Jeux d'essais :	14
Version 6 : Une équipe de joueurs : pour un jeu collaboratif plus proche de la réalité	16
Description :	16
Modifications :	16
Le nombre maximum de joueurs :	16
L'écran de fin :	16
Jeux d'essais :	16
Version 7 : créativité.....	19
Description :	19
Modifications :	19
La fenêtre règle :	19
Les Options :	19
Jeux d'essais :	20
A propos de la qualité	23
Choix des structures de données, notamment le plateau de jeu :	23
Qualité de code	23
Qualité du développement	24
Petit bilan :	27
Démarche réflexive sur le développement de l'application Le Verger	27
Problèmes rencontrés :	27

Introduction :

Présentation du client et de son besoin :

Lors de cette SAE nous avons dû créer une forme personnalisée du jeu du Verger qui est un jeu de coopération destiné aux enfants. Le but du jeu est de cueillir tous les fruits des arbres avant que le puzzle du corbeau ne soit terminé. (Voir annexe pour des règles précises)

Afin de simplifier le jeu,

L'entreprise client nous a demandé, pour pouvoir suivre le développement, de faire 7 versions distinctes, dans lesquels le jeu est codé petit à petit.

Présentation de la stack technique :

Pour cette SAE nous devions coder en Java, en utilisant un projet Maven, la librairie JUnit5 pour les tests automatisés et la librairie Java FX pour l'affichage graphique. De plus nous avons utilisé git unilim pour pouvoir travailler en équipe.

L'IDE que nous avons utilisée était Eclipse, mais nous avons aussi utilisé git Kraken pour visualiser les commit et StarUML pour réaliser les diagrammes UML.

Lien de notre projet : <https://git.unilim.fr/pinheiro7/orchard>

Nous avons réussi à coder les 7 versions.

Version 1 : Des arbres fruitiers : Toujours gagnant !

Description :

Dans cette étape, nous devions mettre en place les trois scènes (Menu, Lobby, Jeu), et dans la scène principale, celle du jeu, nous devions mettre en place le dé (avec les faces de dé associés), le plateau de jeu (les arbres, les fruits, une ébauche du puzzle afin de pouvoir l'afficher sans interactions) et un écran de fin de jeu simpliste.

Modifications :

Menu :

Le menu doit pouvoir permettre de se déplacer dans l'application. Pour l'instant, uniquement deux possibilités devront être proposées : quitter l'application et jouer. Lorsque l'on décide de jouer, la scène du lobby doit être affichée.

Lobby :

Le lobby doit pouvoir permettre d'ajouter des joueurs, tout en vérifiant la validité de son nom, en cas de non-validité, un message d'erreur est affiché. Quand tous les joueurs, sont créés, la partie pourra être lancée en appuyant sur un bouton.

Jeu :

La scène de jeu est la scène principale, dans laquelle tout le jeu se déroule. Dans cette scène de nombreux éléments étaient à développer :

- *Le dé :*

Le dé devait être capable de pouvoir changer de face aléatoirement, et d'être lancé (animation). De plus le dé doit pouvoir après être tombé sur une face,

faire une action en fonction de celle-ci. Parmi ses actions, pour l'instant uniquement celle de cueillir des fruits, avec la possibilité de choisir quel fruit.

- *Le plateau de jeu :*

Celui-ci sera constitué des éléments suivants :

- Les arbres :

Eux-mêmes constitués de fruits, les arbres sont la majeure partie du jeu. Ceux-ci comportent 10 fruits, et peuvent être remplis et cueillis (du moins ses fruits doivent l'être).

- Le puzzle :

Le puzzle est pour l'instant uniquement visuel, et ne contient qu'une image, à terme il pourra contenir des pièces de puzzle.

- *Une interface pour définir le comportement de collecte des fruits*

Le mode de collecte étant changeant (soit un fruit sera pris, soit aucun), nous avons décidé de mettre en place une interface pour pouvoir modifier ce comportement

- *Les parties visuelles :*

Tous les éléments cités précédemment devront être affichés et auront une classe afin de les afficher, ainsi il sera nécessaire de coder toute la partie visuelle de l'application. Nous avons décidé de faire en sorte de permettre de redimensionner notre fenêtre ainsi cette étape n'était pas des moindres.

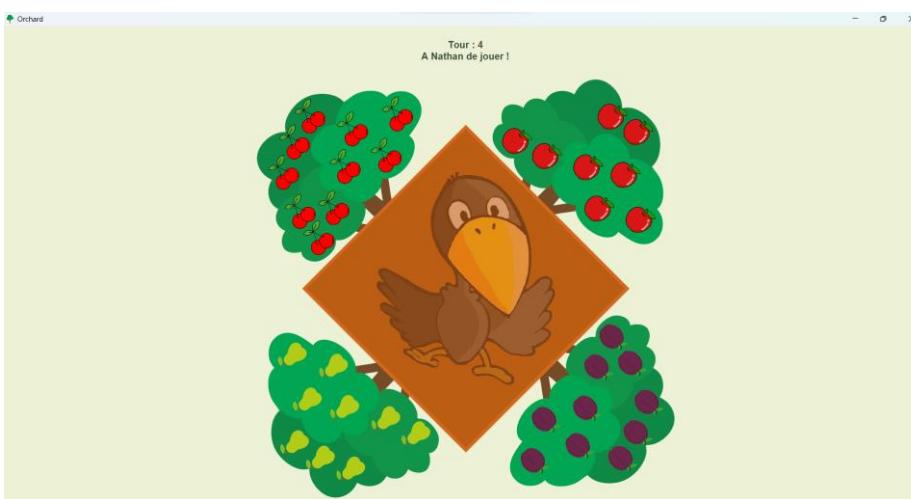
Jeux d'essais :



L'écran
de
menu.



L'écran de lobby.



L'écran de jeu, les fruits se récoltent automatiquement.



L'écran de fin de jeu.

Cette étape nous a pris environ 8 heures

Version 2 : Des paniers pour la cueillette toujours gagnant !

Description :

Dans cette étape, nous devions mettre en place des paniers pour la collecte des fruits. Ceux-ci devaient pouvoir stocker des fruits d'un unique type.

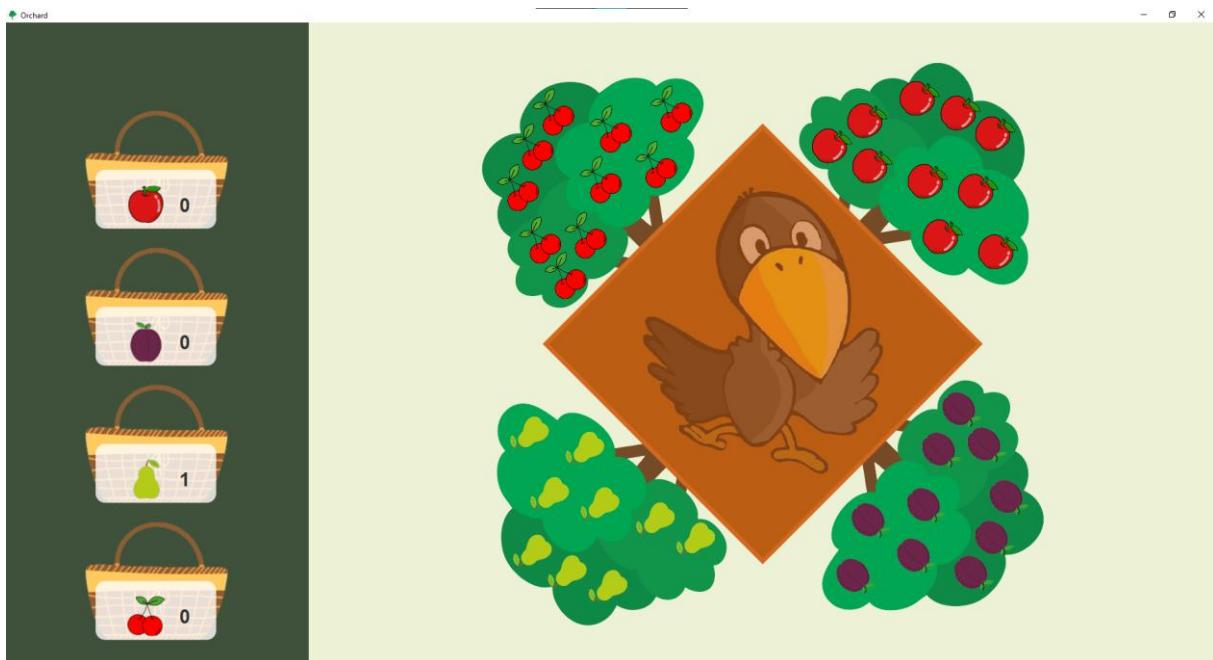
Modifications :

Les paniers :

Les paniers correspondent chacun à un fruit et ne peuvent contenir uniquement les fruits correspondants au type du panier.

Nous avons aussi dû développer la partie visuelle des paniers, qui devra comme tout élément de notre application, gérer la redimension de la fenêtre.

Jeux d'essais :



Cette étape nous a pris environ 4 heures

Version 3 : Apparition du corbeau : Gagnant ou Perdant

Description

Dans cette étape, nous devions ajouter la face corbeau sur le dé et mettre en place le puzzle de celui-ci il y a donc maintenant la possibilité de perdre en complétant le puzzle.

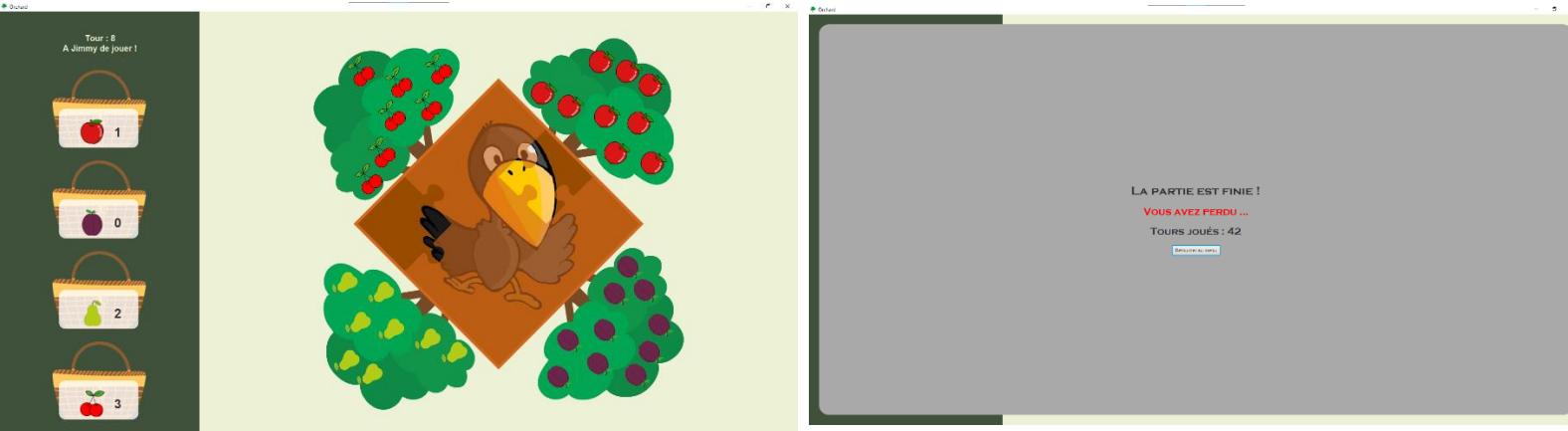
Modifications :

Le puzzle :

Il est composé de 9 pièces et chaque pièce est ajouté automatiquement lorsque le dé tombe sur la face corbeau

Nous avons dû développer le puzzle en l'implémenter à notre jeu. Nous avons aussi ajouté un écran de

Jeux d'essais :



On voit que le puzzle peut avoir de nouvelles pièces.

Cette étape nous a pris environ 3 heures.

Version 4 : Possibilité de choisir le fruit à récolter & bonus possible avec 2 fruits (début interactivité) :

Description :

Dans cette étape, les fruits devront désormais être récoltable via le clic du joueur. De plus, une nouvelle face apparaît, la face du panier, qui permet de collecter deux fruits de n'importe quel type.

Modifications :

La gestion de la collecte :

Le GameHandler, qui gérait jusqu'ici les tours, il va désormais gérer la collecte. Pour cela on supprime la collecte via l'interface Collectable mise en place dans les versions précédentes, et l'on ajoute deux variables dans le GameHandler : une liste de type, qui seront les types actuellement collectable, et un Integer qui définira le nombre de fruit récupérable. Les fruits pourront donc être cliqué uniquement si le type du fruit est compris dans la liste et que le nombre de fruits récoltable est supérieur à 1, et modifieront la valeur de ces variables après chaque collecte.

Le dé :

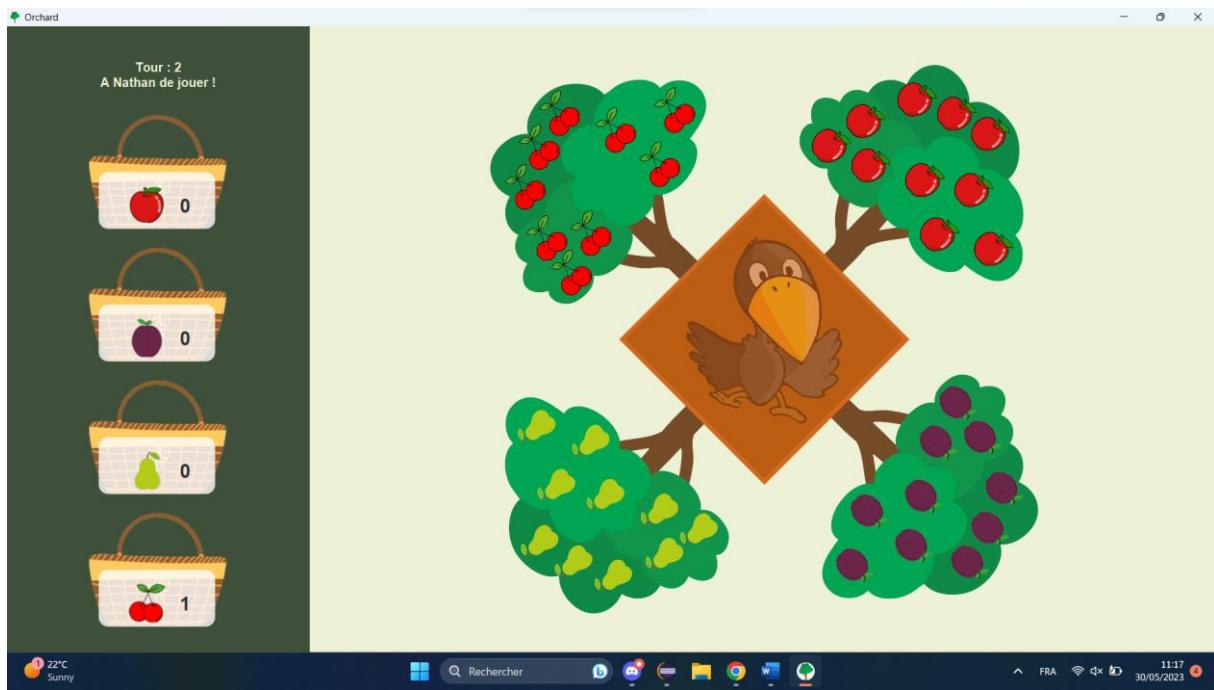
- *Une nouvelle face apparaît sur le dé : la face Panier*

Celle-ci va simplement modifier la valeur des variables en ajoutant à la liste tous les types disponibles et la valeur de 2 fruits récupérable.

- *La face de fruit :*

Ces faces ne collecteront plus automatiquement les fruits, elles modifieront juste la valeur des variables du GameHandler en mettant le nombre de fruits récupérable à 1 et le type correspondant au fruit de la face.

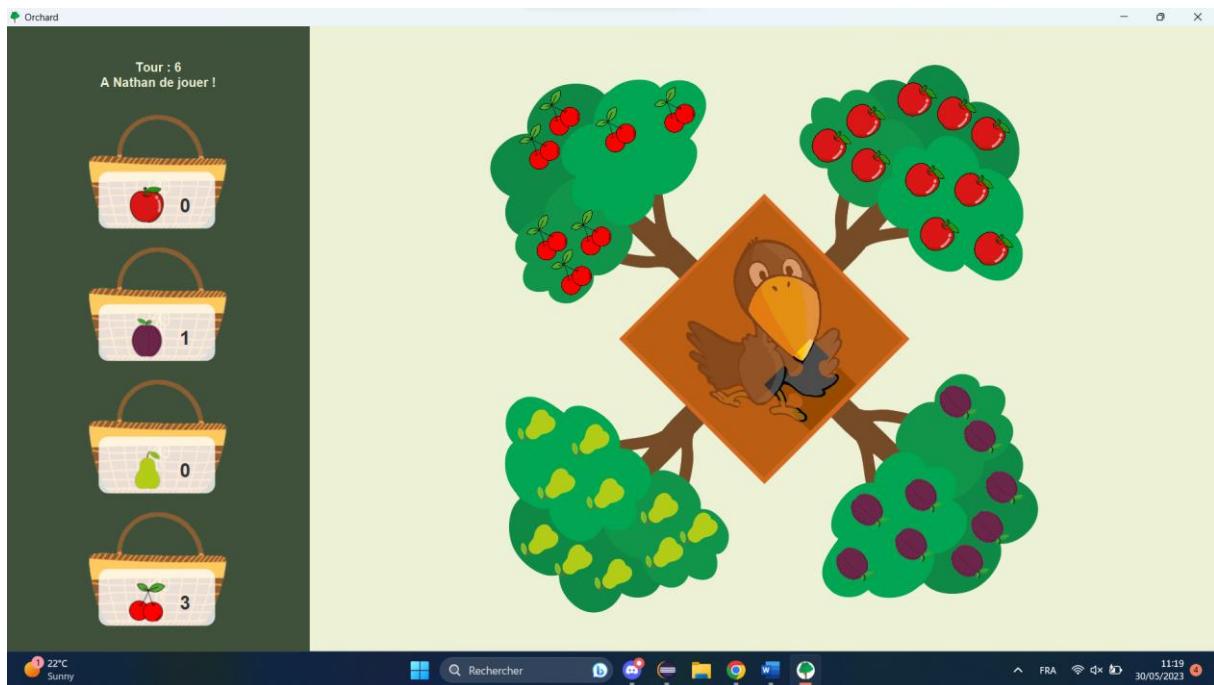
Jeux d'essais :



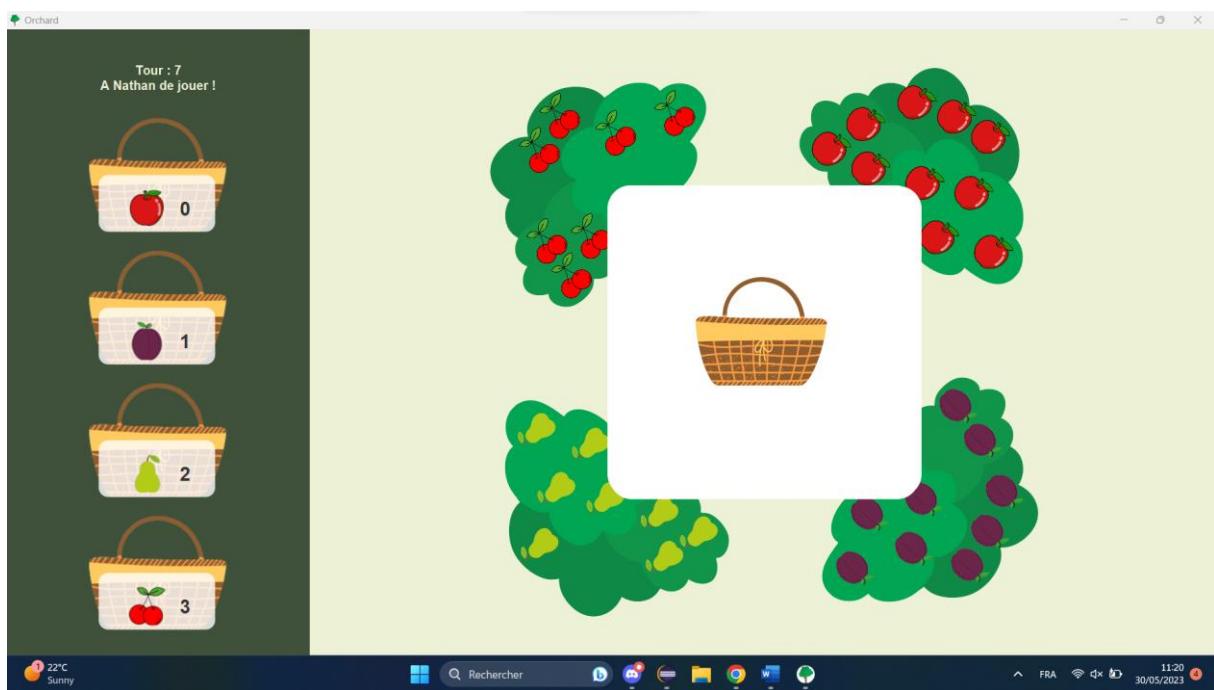
Ici les fruits le dé est tombé sur cerise : le jeu attend le clic sur le fruit



Après le clic, le fruit est récupéré, rangé dans les paniers et le dé réapparait



Ici les fruits le dé est tombé sur Panier : le jeu attend deux clics sur n'importe quel fruit.



Ici deux poires ont été cliqués et récoltés.

Cette étape nous a pris environ 4 heures

Version 5 : Choix du panier & pioche du puzzle : pour plus de créativité et de motricité !

Description :

Dans cette étape, nous devront permettre le drag and drop des fruits dans les paniers. De plus les paniers ne seront plus dédiés, ils pourront stocker n'importe quel fruit. Il sera possible de voir leur contenu.

La gestion du puzzle doit aussi se faire via le drag and drop, avec une pioche de pièce de puzzle.

Modifications :

Les paniers :

Les paniers pourront stocker n'importe quel fruit, ils auront alors 4 variables qui stockeront le nombre de fruit de chaque type. De plus ils auront une valeur maximale de fruits à pouvoir stocker, fixé à 10.

Intérieur des paniers :

L'intérieur du panier est maintenant visible puisque tous les fruits sont acceptés dans celui-ci pour cela nous avons rajouté un écouteur qui permet d'ouvrir le panier après avoir cliqué sur lui. Cela permet de voir chaque fruit avec son nombre associé.

La pioche de puzzle :

Une pioche a dû être développé, elle apparait à chaque fois que la face corbeau tombe. Cette pioche contient toutes les pièces, qui peuvent être drag and drop sur le puzzle pour les ajouter. Lorsque la pièce est drag par-dessus le puzzle, on voit une prévisualisation du puzzle avec la nouvelle pièce.

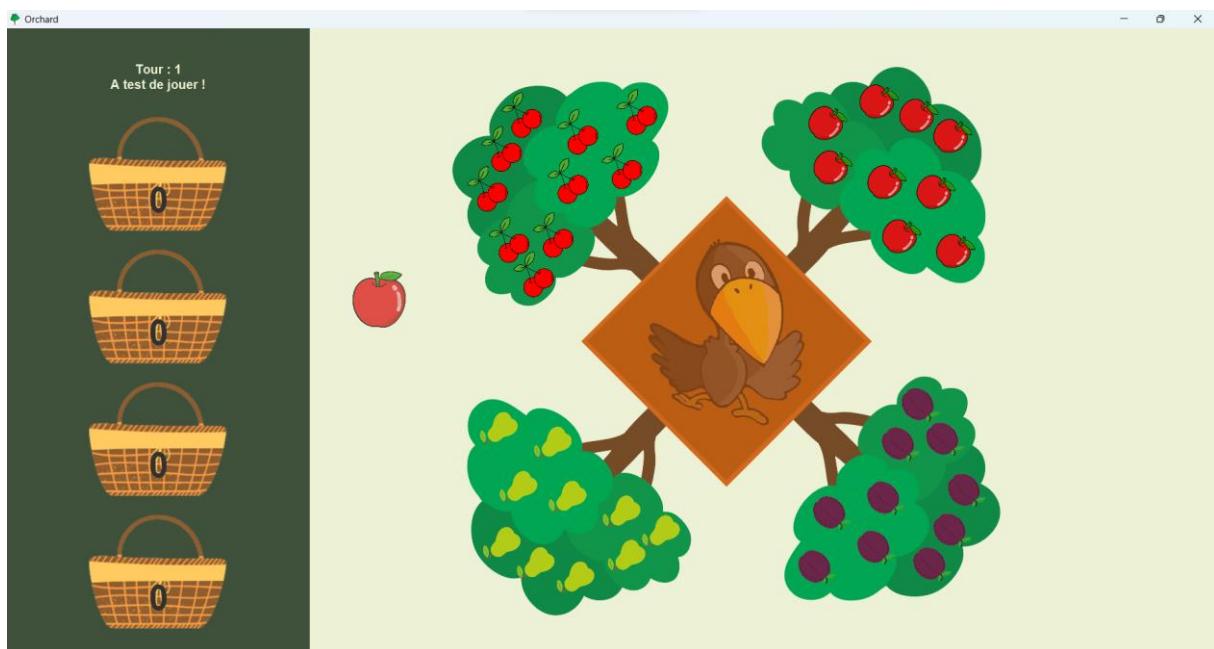
Le drag and drop :

Pour mettre en place le drag and drop, plusieurs objets ont été créés.

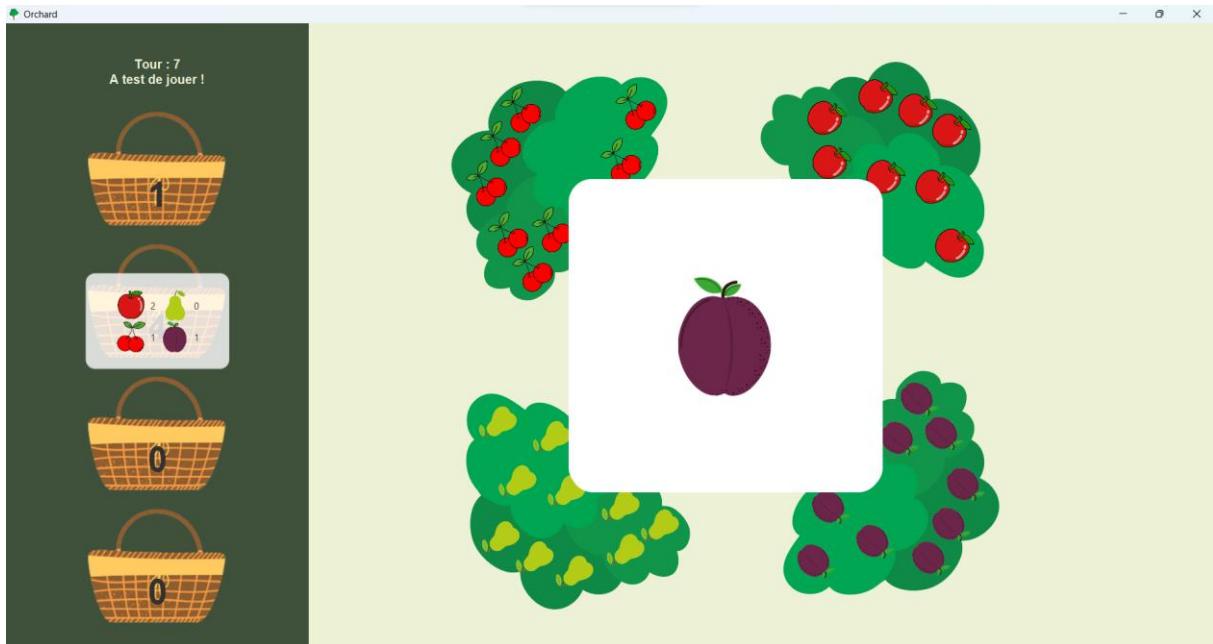
- *BasketDragAndDropController*
- *FruitDragAndDropController*
- *JigsawDragAndDropController*
- *JigsawPieceDragAndDropController*

Ces classes héritent de *EventHandler* et seront les comportements liés au drag and drop des objets auxquels ils seront associés,

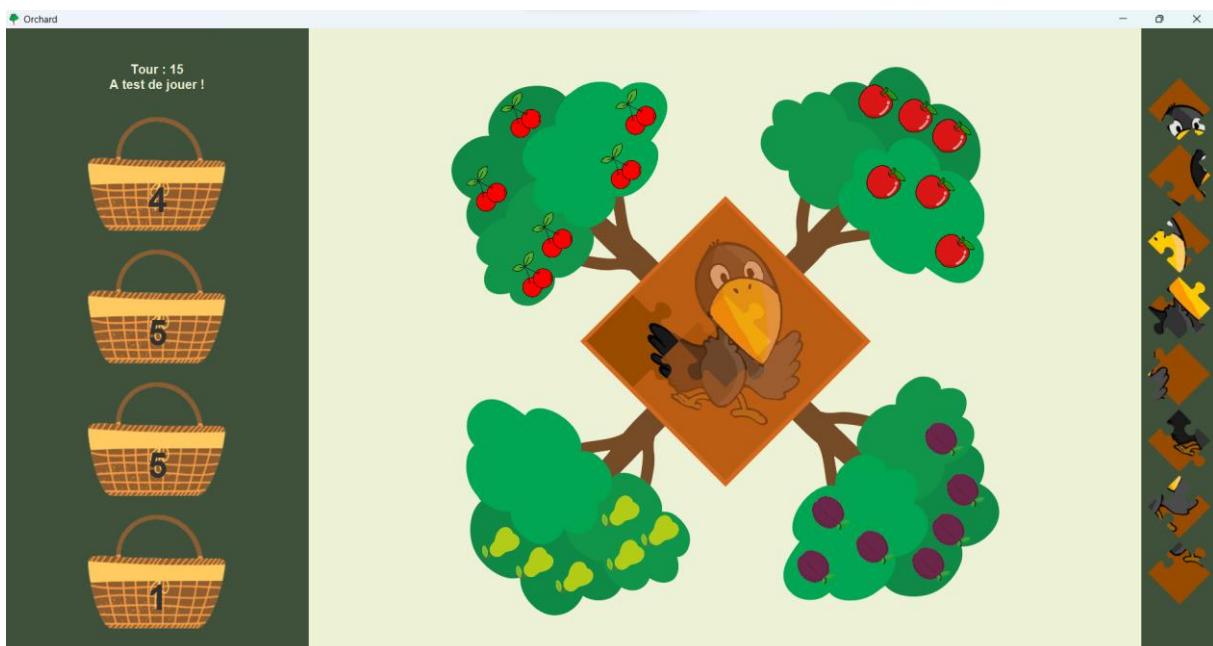
Jeux d'essais :



Ici le drag and drop est en cours, avec l'image de la pomme en déplacement.



Ici le panier est ouvert, et l'on voit son contenu.



Ici la face de corbeau est tombée, la pioche est ouverte et la pièce du milieu a été drag par-dessus le puzzle, on peut prévisualiser le puzzle avec la pièce.

Cette étape nous a pris environ 5 heures

Version 6 : Une équipe de joueurs : pour un jeu collaboratif plus proche de la réalité

Description :

Dans cette étape, le multijoueur devra être supporté. Les joueurs devront jouer chacun leur tour en local et lancer le dé.

Modifications :

Le nombre maximum de joueurs :

Notre application était déjà construite pour accueillir le multijoueur, une constante bloquait simplement l'ajout de joueurs supplémentaire. La seule modification pour permettre le multijoueur était de changer cette variable, la faisant passer de 1 à 3.

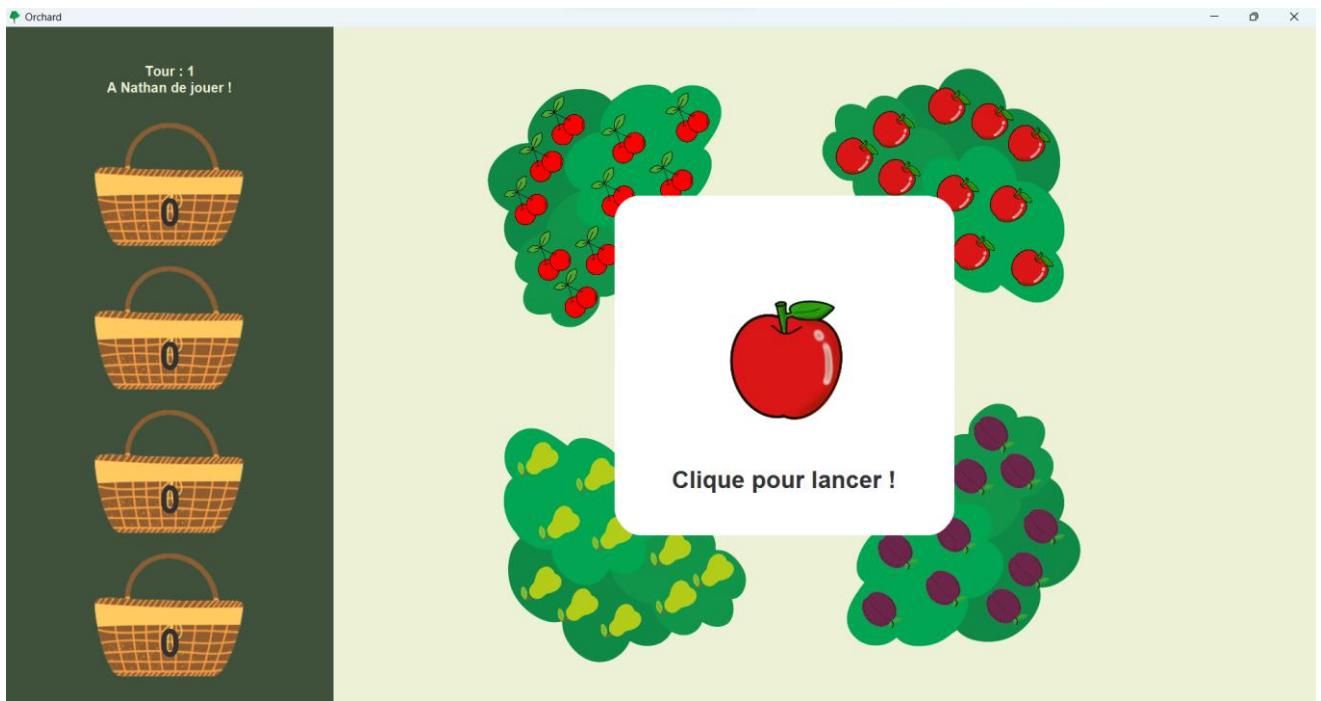
L'écran de fin :

L'écran de fin a donc été modifié afin d'afficher le classement des joueurs en fonction des fruits et des corbeaux collectés.

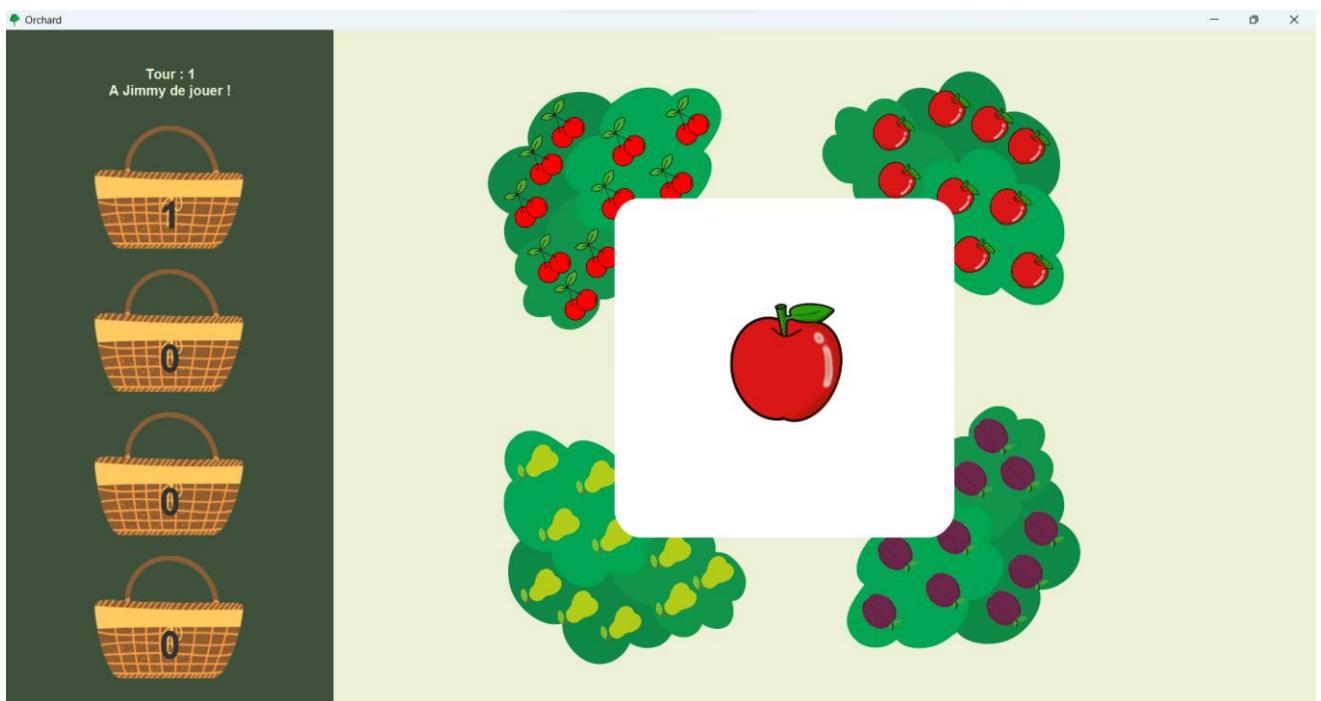
Jeux d'essais :



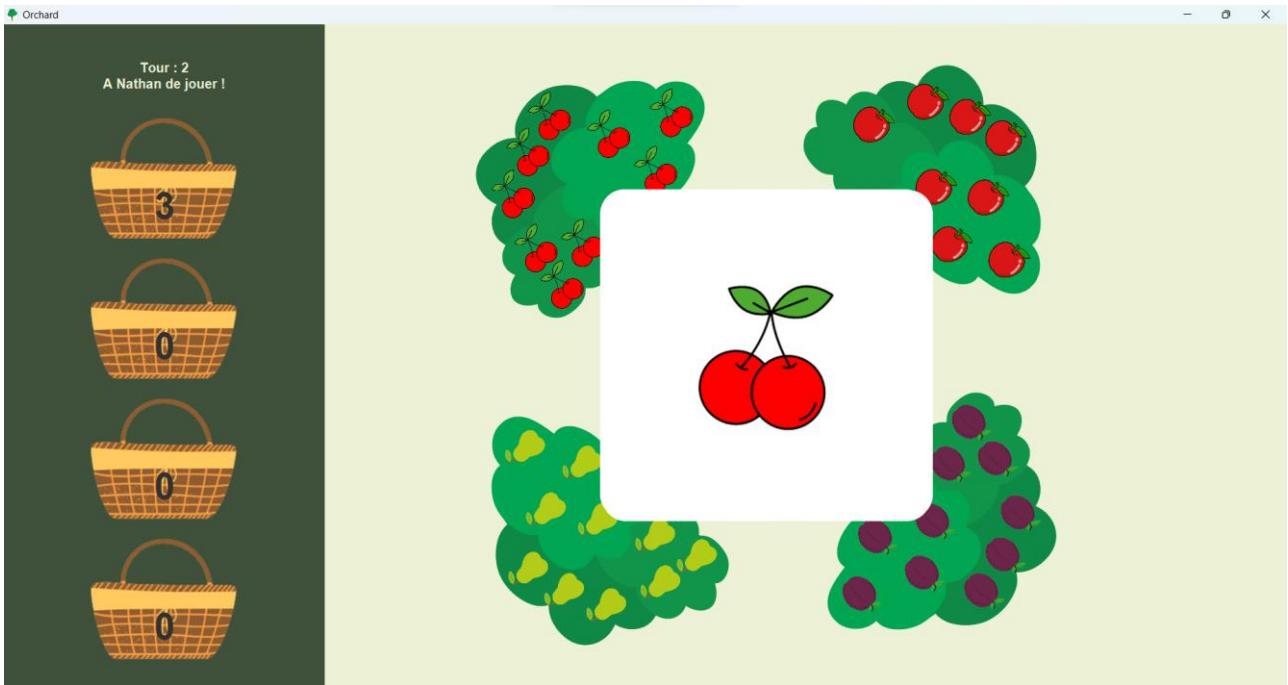
On voit que l'on peut ajouter jusqu'à trois joueurs.



Ici lorsque le premier joueur est Nathan



Après le tour de Nathan, le tour passe à Jimmy



Lorsque le troisième joueur à finit son tour, le nombre de tour augmente.



Voici l'écran de fin, les joueurs sont classés et l'on voit le nombre d'items récoltés.

Cette étape nous a pris environ 2 heures.

Version 7 : créativité

Description :

Dans cette étape, nous étions libres de choisir ce que l'on voulait ajouter. Nous avons décidé de rajouter une fenêtre de règles, une fenêtre option, la gestion des langues et des fonctionnalités supplémentaires diverses (musique de fond, bouton qui se modifient lorsque la souris passe par-dessus, bouton de retour sur la fenêtre de choix des joueurs, fruits en surbrillance lors de la collecte, écran de chargement). Sur la fenêtre d'ajout des joueurs nous avons ajouté le score passé des joueurs en enregistrant leur score à chaque fin de partie

Modifications :

La fenêtre règle :

La fenêtre règle comprends les règles du jeu mais aussi le fonctionnement de la fenêtre d'ajout des joueur et l'explication de l'écran de fin cette partie est uniquement visuel et permet l'explication du jeu dans son entièreté

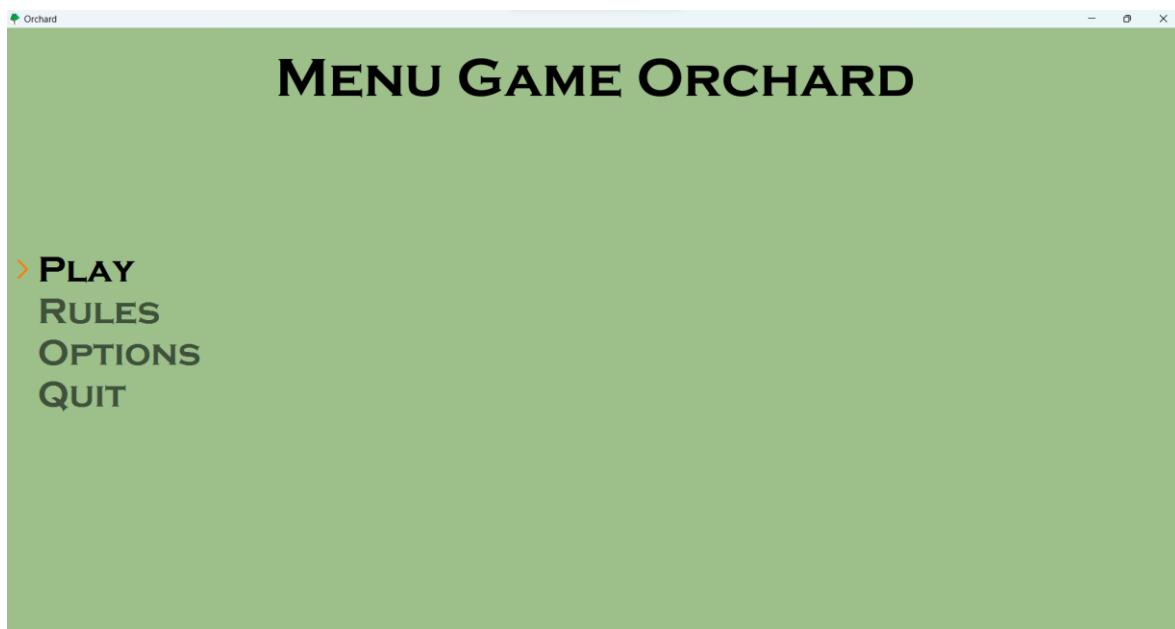
Les Options :

Pour les options nous avons ajouté :

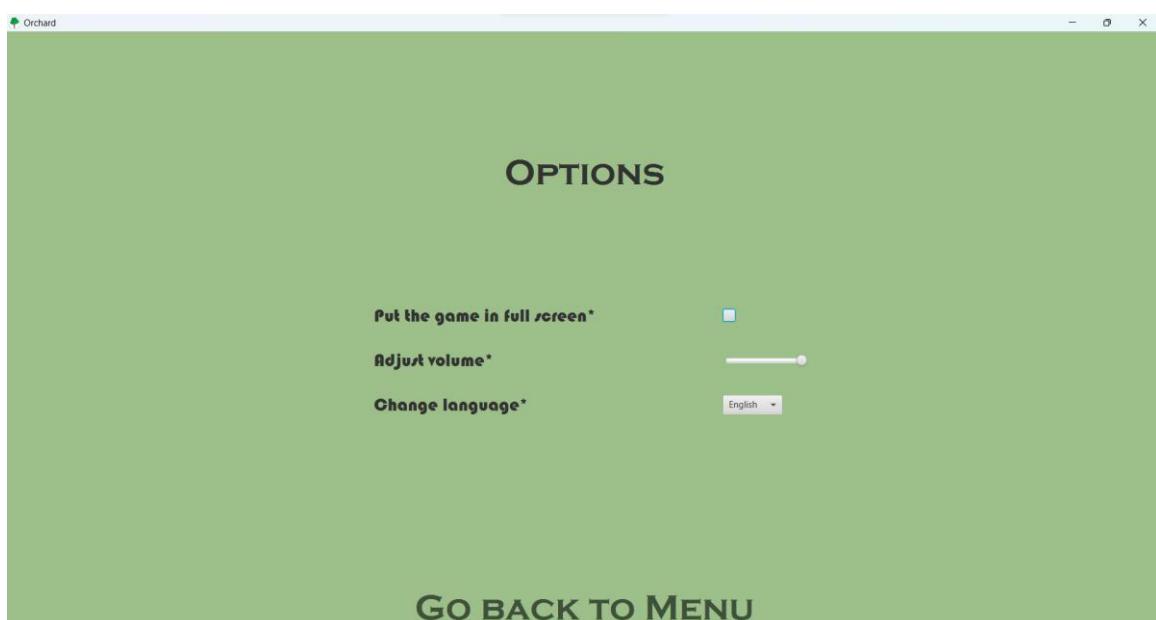
- Le plein écran qui est géré par le SceneController le plein écran est mis à l'appel de chaque scène et si jamais la touche échappe est appuyé le plein écran est désactivé.
- L'ajout de la musique est créé avec un media et un media Player qui est mis dans la scène Controller pour pouvoir lancer la musique au lancement de l'application et avec Controller on peut changer l'état de la musique depuis chaque scène.
- Pour les langues nous avons créé une interface qui permet de changer entre l'anglais le français et l'espagnol chaque langue comprends tous les textes du jeu et en fonction de la langue l'interface prendra dans la classe correspondant à la langue sélectionné le texte qui est appelé

- Nous avons changé le contrôleur pour hover les textes et l'avons mis sur chaque texte cliquable pour améliorer la qualité de notre application.
- Un écran de chargement a été ajouté, dans lequel une astuce est affiché, prise au hasard parmi une liste.
- Les fruits sont passé en surbrillance quand on peut les collecter

Jeux d'essais :



L'écran principal avec tous les nouveaux boutons.



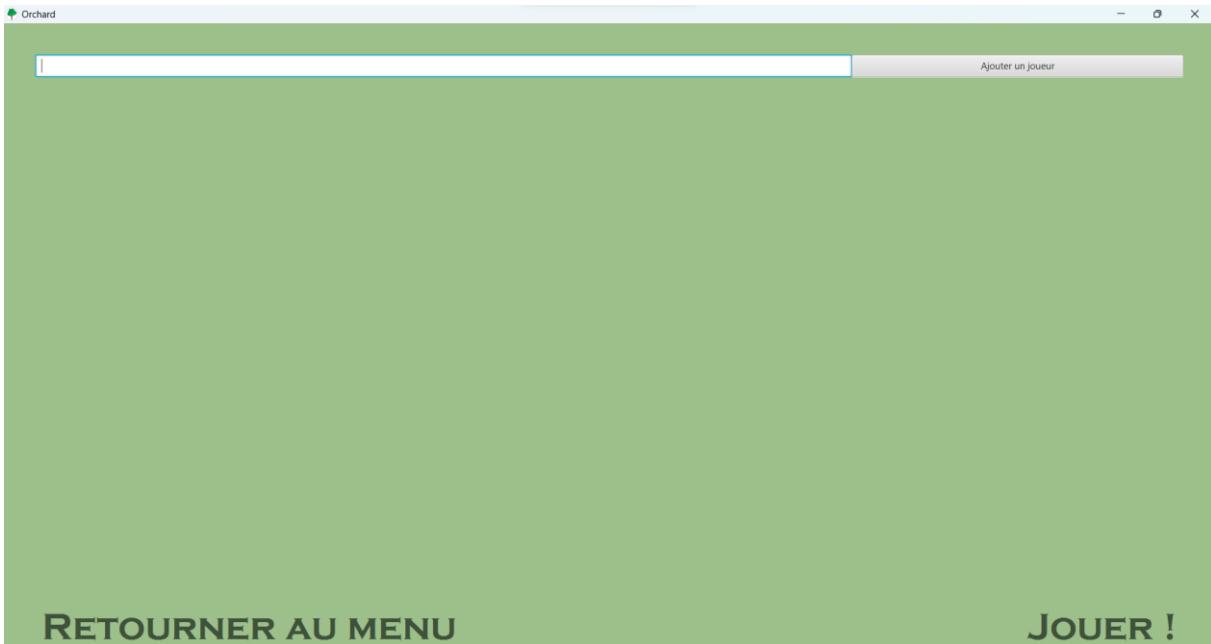
Ici on peut voir l'écran des options, il est possible de changer la langue, modifier le volume ou mettre en plein écran.



Le menu après avoir mis en français.



L'écran de règles, on peut descendre en bas de la page et retourner au menu.



Le lobby avec le bouton de retour au menu. A noter que les boutons changent de couleur lorsque la souris passe par-dessus



L'écran de chargement avec une astuce aléatoire parmi 3.

A noter que tous les textes s'adaptent à la langue choisie, avec l'anglais choisi par défaut.

A propos de la qualité ...

Choix des structures de données, notamment le plateau de jeu :

Pour notre plateau de jeux, nous avons décidé de lui attribuer 4 arbres et un puzzle. Chaque fruit particulier est un objet qui hérite de la classe fruit et chaque arbre particulier est un objet qui hérite de la classe Arbre.

Pour le dé nous avons décidé de lui affilier une liste de faces, qui sont en fait un objet qui stocke l'image de la face et son action, c'est à dire ce qu'il fera lorsque l'on tombe dessus.

Pour les joueurs, nous les avons séparés en deux classes distinctes, l'une qui permet de stocker les données qui seront sauvegardé dans un fichier, et une autre qui permet de stocker les informations relatives au jeu en cours, comme les fruits collectés ou le nombre de pièces de puzzle piochées.

Pour les paniers, rien de particulier, ils stockent simplement leur image et les fruits stockés.

Qualité de code

1. Terminologie métier :

Nous avons décidé de nommer nos variables et nos fonctions en anglais. Tout le code est écrit de cette manière, et de façon la plus compréhensible possible. De plus, nous avons expliqué toutes les fonctions.

2. Mauvaise odeur (code smell) :

Nous avons été attentif à ne pas répéter plusieurs fois les mêmes morceaux de code, à utiliser quelques interfaces et à séparer un maximum notre code en classe et en fonction pour que celui-ci soit le plus simple possible à comprendre, tout en lui permettant de rester évolutif et complet.

Nous avons utilisé Sonar Lint afin de vérifier la propreté de notre code.

3. Revue de code :

Nous avons très peu montré notre code à d'autres personnes, mis à part au professeur et à quelques groupes pour comparer nos solutions, néanmoins cela ne nous a pas beaucoup fait changer notre conception de l'application.

4. Détection et correction de bug :

Nous avons trouvé les bugs grâce à des tests fait en lançant l'application. Par rapport à notre plus beau bug, nous n'en avons pas réellement eu, les quelques bugs étaient mineurs et nous n'y passions pas beaucoup de temps. Si nous devions choisir nous dirions que le plus

gros bug était un bug de changement de scène, qui ne pouvaient pas s'afficher plus d'un fois, néanmoins nous avons rapidement résolu celui-ci.

Qualité du développement

Répartition des tâches au sein de votre équipe de développement :

A propos de la répartition, nous ne pouvons pas citer beaucoup de parties de code qui appartiennent à chacun car nous codions beaucoup à deux en communiquant beaucoup. Cela étant dit, certains ajouts ont été fait par l'un d'entre nous uniquement, tel que la gestion des langues par Nathan ou le menu des options et des règles par Jimmy.

Gestionnaire de version :

Pour la gestion de version nous avons eu du mal à prendre en main git unilim mais une fois que nous avons réussi nous avons fait des commit régulier. Notre seule erreur a été de ne pas merge les branches au fur et à mesure et nous avons donc du recréer chaque version. Pour les commits nous en avons fait très régulièrement à chaque modification pour que l'autre puisse modifier la dernière version et apporter ces modifications.

Architecture de votre application :

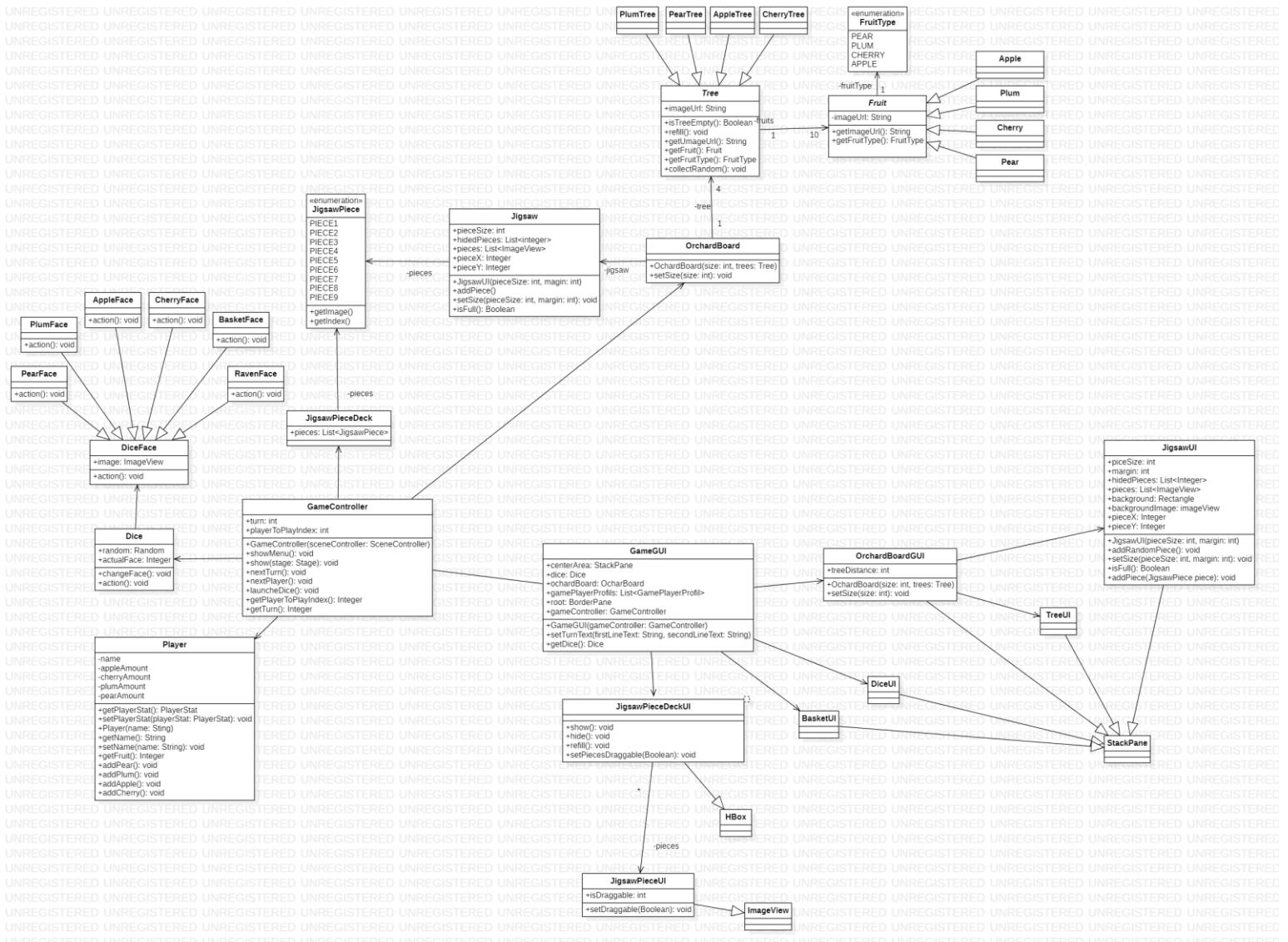
Nous avons séparé notre projet en 9 packages :

- application : comprends la classe main du projet qui lance l'application.
- gui : comprends tous les objets graphiques de chaque scène de notre projet
- scene : agis sur les scènes et gère toutes les interactions que le joueur fait avec l'interface
- ui : contient tous les éléments graphiques particulier, comme le dé, le lobby Player profil, les paniers de jeux, les fruits, les arbres, le puzzle et l'écran de chargement
- controller : Contient tous les contrôleurs.
- interface : comporte toutes les interfaces du jeu comme les langues, ou celles de drag and drop.
- service : Contient les services tels que le gestionnaire de police, de musique et de langue.
- model : comporte toutes les classes métier de notre projet.

Le fait de séparer le code de cette façon nous a permis de mieux tester notre code, et d'avoir de nombreuses classes dont les objectifs étaient assez clairs.

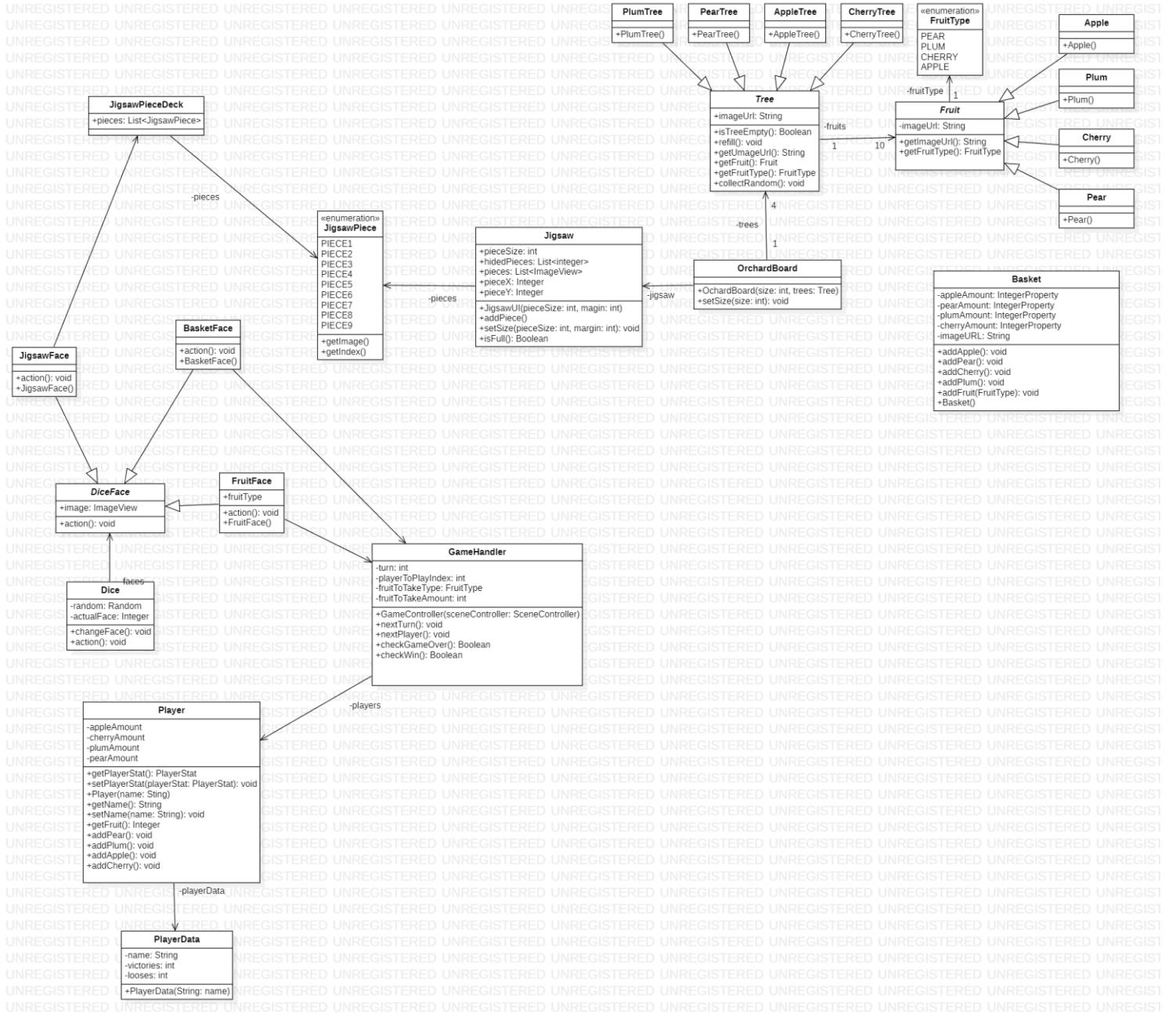
4. Evolution de la conception au cours du développement

✓ Diagramme de classes conçu initialement (prévisionnel)



Ce diagramme comportait les classes prévues pour permettre l'affichage. Si on supprime cette partie on obtient toute la partie haute/gauche de ce diagramme.

✓ Diagramme de classes réellement implémenté (réel)



✓ Evolution de la conception tout au long de l'implémentation

Dans notre conception, nous avons ajouté les paniers que nous avions oublié. Nous avons aussi modifié les faces de fruit en les regroupant dans une unique classe `FruitFace`. Nous avons aussi décidé de séparer en deux classes les parties du joueur que l'on sauvegardera et celle que l'on utilisera uniquement pour le jeu. On peut aussi noter quelques différences dans les relations qui ont été simplifiée. Dans ceci on n'affiche que les classes métiers, les classes permettant l'affichage ne sont pas mises dans ce second diagramme.

5. A propos des tests automatisés :

Pour les tests, nous les avons mis dès le début, pour ne pas à être surchargé de travail et avons utilisé la librairie Junit5. Nous avons décidé de couvrir uniquement le package model, car pratiquement tous les autres packages sont en lien avec l'interface graphique. Nous avons ainsi réussi à couvrir environ 40% de couverture pour chaque version, en couvrant pratiquement la totalité du package model, mis à part les rares parties graphiques.

Ces tests ne nous ont pas vraiment été utile pour le développement, car nous les avons écrit après avoir développé nos classes, en sachant que nos classes marchaient toutes. En toute honnêteté, les tests nous ont semblé une perte de temps, néanmoins nous avons eu le temps de les faire et d'avoir une bonne couverture de code malgré tout.

Petit bilan :

Pour conclure à propos de la qualité, seuls les tests nous ont posés problèmes, ceux-ci nous ont pris beaucoup de temps sans que nous n'y voyions l'utilité, néanmoins les autres facettes de la qualité de code permettent à notre code d'être bien plus explicite.

Démarche réflexive sur le développement de l'application Le Verger

Lors de cette SAE nous avons appris à améliorer notre qualité de code en séparant correctement nos classes de façon efficace et pertinente.

Apprentissages critiques :

AC 1 : Implémenter des conceptions simples

Nous avons réussi à implémenter nos classes métiers

AC 2 : Élaborer des conceptions simples

Nous avons réussi à conceptualiser d'une façon qui nous semble claire nos classes métier

AC 3 : Faire des essais et évaluer leurs résultats en regard des spécifications

Nous avons fait de nombreux essais pour vérifier et corriger la validité de notre code. De plus nous avons fait de nombreux tests automatisés avec Junit5 pour tester nos classes.

AC 4 : Développer des interfaces utilisateurs

Nous avons créé un IHM correcte, avec une séparation objet graphique et classe métier en permettant le changement de taille de notre fenêtre.

Problèmes rencontrés :

Nous avons eu beaucoup de mal avec le git, en effet nous n'étions pas habitués à utiliser git unilim et les push de celui-ci ne fonctionnaient pas correctement. De plus nous avons eu de nombreux problèmes à merge les branches à la branche principale, mais après de fastidieux essais nous y sommes parvenus.

Glossaire :

Jigsaw : (Puzzle) L'objet qui stocke les informations liées à un puzzle

JigsawUI : L'objet qui permet l'affichage d'un objet Jigsaw

Dice : (Dé) Stocke les informations liées à un dé

DiceUI : L'objet qui permet l'affichage d'un objet Dice

Fruit : (Fruit) Stocke les informations liées à un fruit

FruitUI : L'objet qui permet l'affichage d'un objet Fruit

Tree : (Arbre) Stocke les informations liées à un arbre

TreeUI : L'objet qui permet l'affichage d'un objet Tree

Basket : (Panier) Stocke les informations liées à un panier

BasketUI : L'objet qui permet l'affichage d'un objet Basket

GameHandler : Un objet qui permet de gérer les tours, et les types de fruits collectable et le nombre de fruit à prendre

Player : (Joueur) Stocke les informations liées au Joueur dans la partie actuelle

PlayerData : Stocke les informations liées au Joueur que nous devons sauvegarder

OrchardBoard : L'objet qui stocke 4 arbres et 1 puzzle.



ENGLISH

Annexe : Les Règles :

Orchard

Cooperative game for 1 to 8 children ages 3 to 6.

Author: Anneliese Farkaschovsky
Illustrations: Walter Matheis
Length of the game: 10 - 15 minutes approx.

The four fruit-trees are full of fruit. The apples, pears, cherries and plums are ripe and have to be picked quickly, because the crafty raven is eager to pinch some tidbits.

Contents

- 10 apples
- 10 pears
- 10 pairs of cherries
- 10 plums
- 4 baskets
- 1 raven jigsaw (9 pieces)
- 1 color die with symbols
- 1 game board



Aim of the game

*Short instructions:
harvest fruit*

The children try to pick the fruit from the trees before the raven can steal them.

*fruit on the trees,
take basket*

Preparation of the game

The fruit is distributed among the corresponding trees shown on the game board. Each player gets a basket. If there are more than 4 players, several players share one basket.

*pile up jigsaw pieces
get the die ready to roll*

The nine parts of the raven jigsaw are removed from the frame and piled up so that they are ready to use. Place the die next to the game board.



How to play

The youngest player starts and throws the die once.

throw die 1x

Which color or symbol appears on the die?

- **Red, yellow, green or blue:**

The player picks a fruit of the corresponding color and puts it into his/her basket. If there are no more fruits of that color on the tree nothing happens and the die is passed on immediately.



*red = cherry
yellow = pear
blue = plum
green = apple*

- **The basket:**

You can pick any two fruits and place them into your basket.



basket = 2 fruits

- **The raven:**

Place one piece of the jigsaw into the middle of the game board on the corresponding place.



raven = piece of jigsaw

End of the game

If the players succeed in gathering all the fruit before the raven jigsaw in the middle is finished, they all win the game – together they have been quicker than the voracious raven.

*all fruit picked:
all players win*

If the raven jigsaw is complete before all the fruit has been picked, the children lose together to the quick raven.

*raven jigsaw complete:
all players lose*

