***Une image contenant capture d’écran, dessin humoristique, hibou, oiseau

Description générée automatiquement***

Table des matières

[1. Présentation 3](#_Toc156742893)

[1.1. Description générale 3](#_Toc156742894)

[1.2. Règles du jeu 3](#_Toc156742895)

[1.3. Cinématique des écrans 4](#_Toc156742896)

[2. Conception – Diagramme de classe 4](#_Toc156742897)

[2.1. Présentation générale 4](#_Toc156742898)

[2.2. Présentation détaillée de la classe MainWindow 5](#_Toc156742899)

[2.3. Présentation détaillée de la classe MenuAccueil 7](#_Toc156742900)

[3. Partie Algorithmie 8](#_Toc156742901)

[3.1. Détection des collisions des obstacles avec le joueur 8](#_Toc156742902)

[3.2. Création des obstacles aléatoires 8](#_Toc156742903)

[3.3. Suppression des obstacles 9](#_Toc156742904)

[3.4. Le système de score 9](#_Toc156742905)

[4. Conception graphique 10](#_Toc156742906)

[5. Cahier de recettes 11](#_Toc156742907)

[5.1 Tests de validation 11](#_Toc156742908)

[5.2 Tests de performance 11](#_Toc156742909)

# Présentation

## Description générale

## 

Sous les conseils avisés de nos mentors, nous avons décidé de nous réunir autour d’une table afin de lister toutes nos idées. Nous savions d’ores et déjà que le projet tournerait autour d’un jeu grâce aux consignes de la SAE. Nous avons établi la difficulté ainsi que le style pour chacune des idées, pour enfin choisir ce qui correspondait le mieux à nos critères.

Après moultes réflexions, nous avons finalement choisi un jeu qui nous représente : Acharnement, Challenge et Talent, nommé FlappySimulator®. C’est un jeu infini qui s’inspire du légendaire jeu FlappyBird, et qui consiste à faire passer un oiseau entre différents types de structures. Le joueur aura le choix entre plusieurs cartes en début de partie ou encore choisir sa difficulté.

## Règles du jeu

Le jeu est très simple : après avoir choisi la carte et la difficulté avec lesquelles vous voulez jouer, vous apparaissez sur l’écran, représenté par un oiseau, il suffira alors d’appuyer sur la touche Espace pour pouvoir sauter (à la base, la touche est bien la touche Espace, mais il est possible de changer cette touche dans les paramètres du menu). Le but sera alors de passer votre joueur entre les obstacles en faisant attention à ne pas les toucher, de même que le sol et le haut de votre écran.

Néanmoins, nous n’avons pas mis de code triche car nous en n’avions pas la nécessité.

## Cinématique des écrans

Une image contenant texte, capture d’écran, Logiciel multimédia, logiciel

Description générée automatiquementUne image contenant texte, capture d’écran, Logiciel multimédia, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, Logiciel multimédia, Système d’exploitation

Description générée automatiquementUne image contenant texte, capture d’écran, Logiciel multimédia, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquementUne image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Une image contenant texte, capture d’écran, Logiciel multimédia, logiciel

Description générée automatiquement

Une image contenant texte, capture d’écran, oiseau

Description générée automatiquement

# Conception – Diagramme de classe

## Présentation générale

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Nous avons utilisé 2 classes : la classe MainWindow et la classe MenuAccueil.

La classe MenuAccueil nous a servit à réaliser le menu dans lequel l’utilisateur va se retrouver en début du jeu. Dans cette classe, nous avons implémenter des fonctionnalités nous permettant de modifier quelques éléments que nous allons utiliser plus tard dans MainWindow : le son, les touches, la difficulté…

La classe MainWindow nous a servit à réaliser le jeu en lui-même, se trouvant après le menu. Dans cette classe, nous avons implémenter le code de la boucle du jeu et son gameplay.

Pour finir, la classe App nous sert uniquement à choisir la fenêtre que l’on souhaitait lancer en premier. Cette classe nous a permis de travailler sur deux fenêtres en simultanées.

Pour chaque classe, nous avons choisi de faire un nommage universel pour chaque variables, méthodes ou bien encore chaque constante. En effet, nous avons mis une majuscule à chaque nom de méthodes, une minuscule à chaque variable et toutes les constantes en majuscules. Ce type de nommage nous a permis d’être plus organisé et de ne pas nous mélanger.

De plus, nous avons fait attention d’utilisé des constantes pour des variables qui sont fixes. Nous avons essayé de faire le moins de tests possibles ainsi que de créer le moins de variables possibles dans les méthodes.

## Présentation détaillée de la classe MainWindow



La fenêtre MainWindow est appelée après la fenêtre MenuAccueil. Elle possède un moteur de jeu de 20 ms nous permettant d’appeler plusieurs méthodes.

Nous réutilisons également quelques champs et méthodes venant de la fenêtre MenuAccueil, tels que touche, numCarte ou MouvementFond.

Dans le moteur du jeu, nous appelons de nombreuses méthodes :

* La méthode CreationObstacle : elle crée un obstacle du haut et un obstacle du bas, toute 0.5 seconde. L’obstacle du haut et celui du bas sont séparés plus ou moins grandement en fonction de la difficulté choisie par l’utilisateur. Leur position dans l’axe des ordonnées est aléatoire. A chaque création d’obstacle, l’objet est ajouté dans une liste qui nous permet ainsi de mieux les gérer.
* La méthode MouvementObstacle : elle bouge chaque obstacle présent à l’écran en fonction de la vitesse du sol.
* La méthode MouvementJoueur : elle bouge le joueur à l’écran en fonction de sa vitesse, calculée par CalculeVitesseJoueur utilisant une formule mathématique pour mettre à jour la vitesse du joueur en fonction du temps depuis le dernier saut (tempsDepuisSaut). Ce temps va s’incrémenter à chaque boucle, et se réinitialiser au moment d’un saut.
* La méthode TesteCollisionJoueur : cette méthode crée une hitbox pour chaque obstacle et grâce à la hitbox créée pour le joueur précédemment, il est possible de détecter une collision entre celui-ci et un obstacle.
* La méthode EnleverObstacle : cette méthode va parcourir la liste de rectangle obsctaclesAEnlever et retirer tous ses éléments du Canvas et de la liste d’obstacle créée précédemment (listeObstacles). De ce fait, le programme ne garde en mémoire que les obstacles visibles à l’écran.

## Présentation détaillée de la classe MenuAccueil



MenuAccueil est la fenêtre que l’on appelle au début du jeu, avant MainWindow. Elle possède également un moteur de jeu d’intervalle 20ms nous permettant d’avoir un bon affichage et appelant plusieurs méthodes.

Dans cette fenêtre, nous avons implémenté de nombreuses méthodes et champs, dont quelques d’entre eux qui vont nous servir plus tard dans MainWindow en fonction des choix de l’utilisateur :

* La méthode public MouvementFond : elle prend en paramètres 3 fonds qui vont bouger infiniment en fonction de la vitesse du fond (vitesseFond). Dans MainWindow, nous réutilisons la méthode de la même façon, en revanche, la vitesse du fond va changer, ou non, en fonction de la difficulté choisie.
* La méthode public MouvementSol : il s’agit de la même méthode MouvementFond, mais c’est le sol qui est bougé infiniment en fonction de la vitesse du sol (vitesseSol) et sera de la même façon, utilisée dans MainWindow en fonction de la difficulté.

Les méthodes pour changer de fond, de son et de touche (respectivement ChangerCarte, ChangerSon et ChangerTouche) : ces méthodes sont appelées dans le moteur de jeu quand l’utilisateur décide de changer de carte (ChangerCarte et ChangerSon) et de touches (ChangerTouche).

# Partie Algorithmie

Pour tous les algorithmes, nous avons essayé de faire le moins de tests possible avec les différentes boucles utilisées. Nous avons utilisé principalement des boucles « for » car c’était le type le plus adaptés pour nous (pour le parcours des listes et pour le parcours des enfants du canva).

De plus, afin de ne pas avoir de la redondance dans le code, nous avons créé des méthodes (les méthodes pour la gestion de l’encadrement des boutons, pour la gestion des fonds et des sols).

Pour finir, nous avons essayé d’adapter la meilleure structure conditionnelle pour les différentes situations. Par exemple, pour la gestion des effets de vol du joueur, nous avons utilisé un « switch ».

## Détection des collisions des obstacles avec le joueur

Méthode MoteurdeJeu()  dans MainWindow.xaml.cs ligne 159:

*on crée d'un carré pour la hitbox du joueur*

*si le joueur sort de la taille de l'écran*

*on affiche l'affichage quand il perd*

*on joue le son du Game Over*

*on arrête la minuterie*

*on prévoit de supprimer les obstacles*

*finsi*

Méthode TesteCollisonJoueur(Rectangle) dans MainWindow.xaml.cs ligne 230:

*pour tous les rectangles dans le canva*

*si le joueur intersecte un obstacle*

*on affiche l'affichage quand il perd*

*on joue le son du Game Over*

*on arrête la minuterie*

*on prévoit de supprimer les obstacles*

*finsi*

*sinon*

*on mets à jour le score*

*on affiche le score*

*finsinon*

*finpour*

Pour cette méthode on peut compter 24 tests toutes les 20ms (au maximum 1 par obstacles).

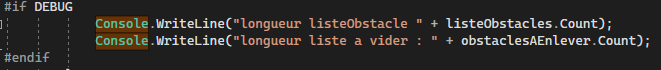


Figure : DEBUG pour voir la suppression des listes

## Création des obstacles aléatoires

Méthode CréationObstacle() dans MainWindow.xaml.cs ligne 266:

*on génére un nombre aléatoire donnant la position haut de l'obstacle*

*on crée un rectangle pour l'obstacle en bas*

*on positionne le rectangle sur le canva*

*on crée un nombre aléatoire pour la distance entre les 2 obstacles*

*on crée l'obstacle du haut*

*on positionne l'obstacle du haut sur le canva*

*on pivote cet obstacle de 180 degré*

*on ajoute les obstacles en tant que enfant du canva*

*on ajoute les obstacles à la liste des obstacles*

Pour cette méthode on génère 2 obstacles toutes les 0.5 secondes ce qui fait au maximum 12 obstacles dans la liste

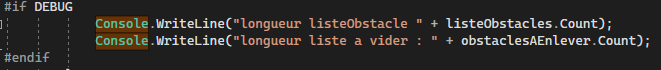


Figure : DEBUG afin de voir la taille des listes

## Suppression des obstacles

Méthode EnleverObstacle() dans MainWindow.xaml.cs ligne 343 :

*Pour tous les rectangles qui sont sorties de l'écran*

*On enlève les rectangles de la liste des autres rectangles présent sur le canva*

*On enlève les rectangles enfants du canva*

*finpour*

*On réinitialise la liste des obstacles sorties de l'écran*

Pour cette méthode, à chaque fois qu'un rectangle est sorti, on l'ajoute à une liste, puis on l’efface. Finalement, cette liste reçoit 2 rectangles au maximum

## Le système de score

Méthode MiseaJourScore(Rectangle) dans MainWindow.xaml.cs ligne 355:

*pour tous les rectangles du canva*

*si le rectangle à un tag différent de "Passé"*

*si la position du rectangle est inférieur ou égale à la position du joueur*

*on augmente le score de 0.5*

*on mets le tag du rectangle à "Passé"*

*finsi*

*finsi*

*finpour*

Cette méthode permet de compter le score du joueur. Nous avons mis un système de Tag car nous voulions compter uniquement le rectangle une seule fois quand il a passé la position du joueur. Ainsi, il y a uniquement que 4 rectangles qui portent le tag « Passé ».



Figure : DEBUG pour voir les tags des obstacles

# Conception graphique

La plupart des éléments affichés (fond, boutons…) à l’écran sont des images prises sur internet avec lesquelles on a rempli les Rectangle :

**FONDS**

Pour chacun des fonds, nous avons découpé la partie du haut (le fond) avec celle du bas (le sol) afin de les manipuler individuellement. Ils apparaissent tous en fond du menu et du jeu quand ils ont été sélectionnés dans le menu par le bouton ayant le même fond.

Premier fond :

Une image contenant ciel, plein air

Description générée automatiquement

<https://www.shutterstock.com/fr/image-vector/seamless-layered-parallax-ready-runner-shooter-457397332>

Deuxième fond :

Une image contenant Magenta, art

Description générée automatiquement

<https://fr.vecteezy.com/art-vectoriel/524760-fairy-sci-fi-alien-landscape-pour-ui-game>

Troisième fond :

Une image contenant ciel, vert, capture d’écran, dessin humoristique

Description générée automatiquement

<https://wallpapersafari.com/w/3DTXAj>

**PERSONNAGE**

Le personnage est un oiseau dont on change l’image à chaque saut. Il s’agit de 2 images extraites d’un gif :

Une image contenant pixel, capture d’écran, Modélisation 3D

Description générée automatiquement

Une image contenant pixel, capture d’écran, Modélisation 3D, Logiciel de jeu vidéo

Description générée automatiquement

<https://www.behance.net/gallery/111165503/Pixel-art-animations?locale=fr_FR>

**OBSTACLES**

Pour tous les obstacles du jeu, nous utilisons une seule et même image :



<https://www.deviantart.com/xpoltergeistcatx/art/Pixel-Doll-Display-Shelf-base-533446270>

# Cahier de recettes

## 5.1 Tests de validation

Durant ce projet, nous avons décidé de découper le travail à faire en deux afin d’essayer d’avoir la même charge de travail :

|  |  |  |
| --- | --- | --- |
| Nom | Fonctionnalités/ Dialogues/Classes | Etat |
| GERIFAUD | MainWindow + MenuAccueil | OK |
| MAULINE | MenuAccueil + MainWindow | OK |
| GERIFAUD | Système de score | OK |
| MAULINE | Génération des obstacles | OK |
| GERIFAUD | Collision obstacles/joueur | OK |
| MAULINE | Choix des difficultés + choix des cartes | OK |
| GERIFAUD | Musique | OK |
| MAULINE | Déplacement du joueur | OK |
| GERIFAUD | Mouvement des obstacles | OK |
| MAULINE | Suppression des obstacles | OK |
| GERIFAUD | Effet de mouvement du joueur | OK |
| MAULINE | Système du changement de touches | Bug pour « ESPACE » |
| MAULINE | Graphisme des cartes | OK |

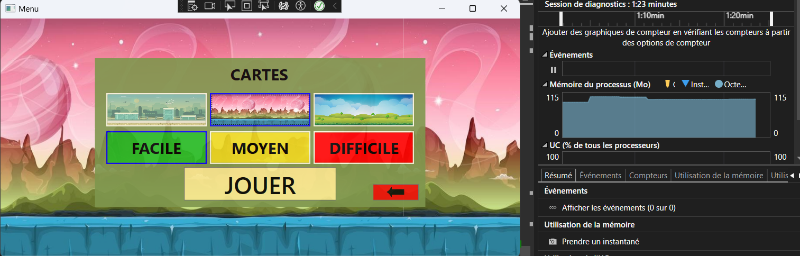
## 5.2 Tests de performance

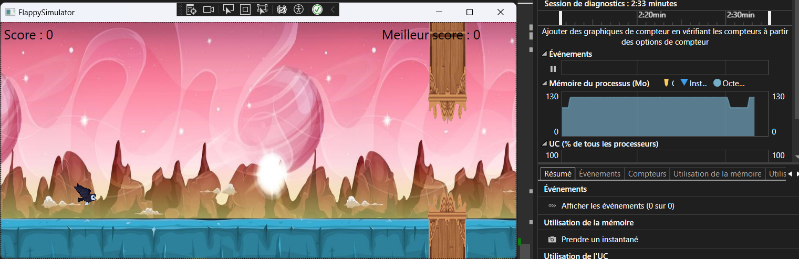
A l’aide des outils de diagnostiques : vous prendrez quelques mesures (captures écrans) à des moments clefs de votre jeu de l’utilisation de la mémoire et du processeur. Vous commenterez bien évidemment les moments choisis et les mesures.

Une image contenant texte, Logiciel multimédia, logiciel, Logiciel de graphisme

Description générée automatiquement

Malheureusement, le fait d’avoir ajouter un moteur de jeu au menu, pour avoir un fond défilant, fait que celui-ci est très énergivore en mémoire de processus.



De plus, l’utilisation de mémoire augmente au moment de changer de fond…

L’utilisation de la mémoire pendant le jeu est relativement haute mais pas si différente qu’au menu.