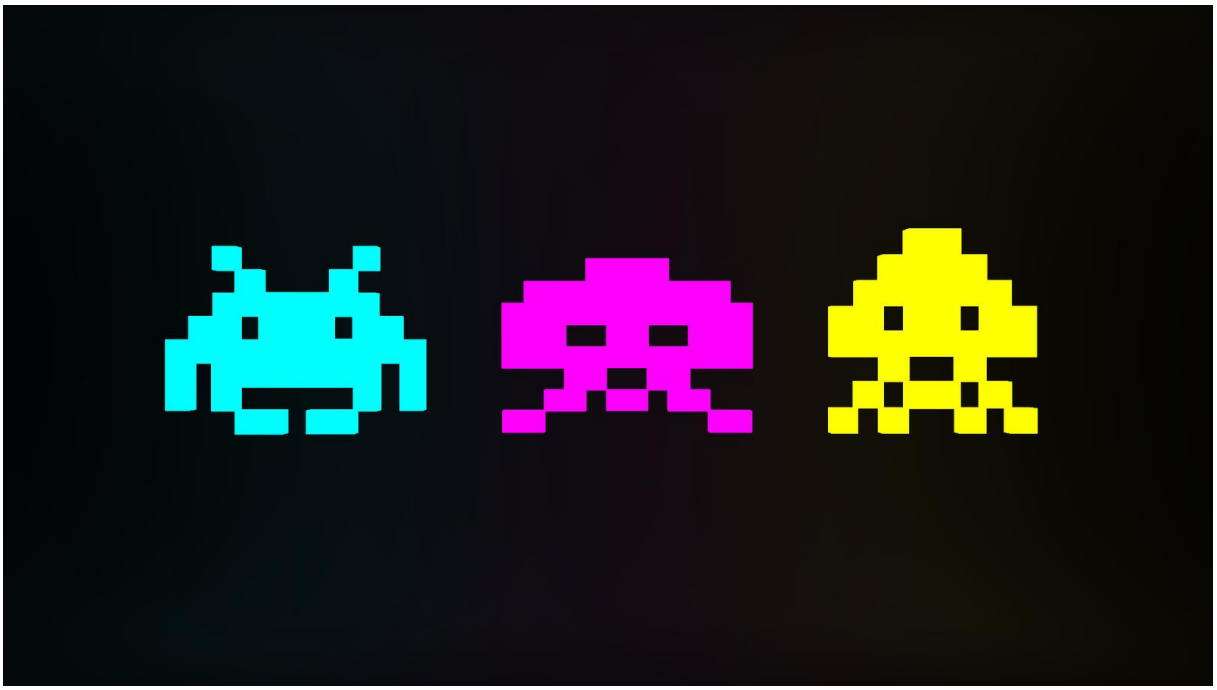
P_Space_Invaders



Lucas Lordon MID2/GRP2D

Vennes – ETML

28.08.23 – 03.11.23

Aurélie Curchod, Mathieu Meylan, Xavier Carrel

Table des matières

Interface Utilisateur

Introduction

Le projet « *SpicyInvaders* » nécessite la création d'un prototype cliquable réaliser grâce au logiciel « *Figma* » pour la partie du projet en lien avec le module I322 (UX).

La maquette doit comporter les éléments suivants :

- le choix de lancer une partie en multijoueur ou en solo
- le choix de changer de palette graphique
- le choix d'avoir un design différent pour les ennemis
- une page contenant les meilleurs scores
- le choix de Gameplay (jouabilité, accessibilité, éco-conception)

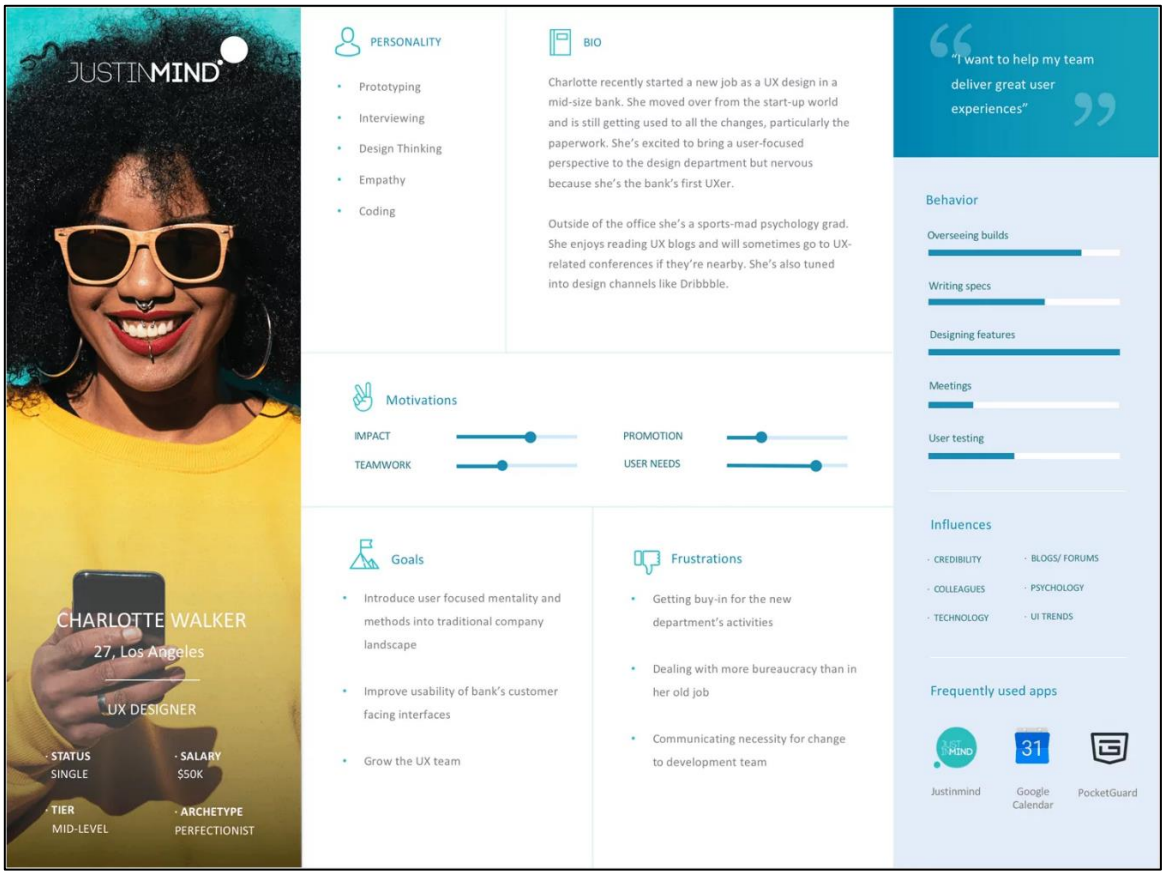
D'autres élément tels que 2 Persona nous son demander.

Analyse

Persona

Dans le cadre de ce projet il est nécessaire de crée 2 Persona. En UX une pratique importante et de commencer par crée des Persona afin de se rendre compte du public que notre design cible. Ces Persona contienne des informations pouvant s'avérer utiles lors de la création de nos maquettes et lorsqu'il faut prendre des choix artistiques, de plus il arrive que les Persona soulève des problèmes d'accessibilité au quel il est important de s'adapter. Par exemple un restaurant asiatique devra comporter une option pour afficher son site dans la langue du pays d'où sa cuisine provient car une partie de c'est client/utilisateur ne parlerons que cette langue.

Pour la création de mes Persona j'ai recréé deux Template afin de m'entrainer à l'utilisation de Figma, ces Template de persona son inspirer ce Template :



Inspiration n°1

Lien de l'inspiration n°1 : https://userguiding.com/blog/user-persona-examples/

Malheureusement je ne retrouve pas le deuxième Template dont je me suis inspiré.

Persona N°1

Thomas Martin est un ingénieur en informatique basé à Paris. Depuis son enfance, il est passionné par les jeux vidéo rétro, tels que Space Invader. Il cherche à équilibrer sa carrière professionnelle réussie avec sa passion pour les jeux vidéo, et il est toujours en quête de moments de plaisir en jouant à ses jeux préférés, que ce soit sur sa vieille console ou sur son ordinateur de jeu.

Thomas Martin

AGE	METIER	LIEU DE RESIDENCE
32	Ingénieur informaticien	Paris

"La vie est un jeu, et je veux en profiter autant que possible."

BUTS

Trouver un équilibre entre sa carrière professionnelle et sa vie personnelle. Il souhaite également rester connecté avec sa passion pour les jeux vidéo, qu'il chérit depuis son enfance.

Applications préférées

Motivations

- Expérience immersive
- Défis stimulants
- Communauté de joueurs
- Personnalisation et progression

Frustrations

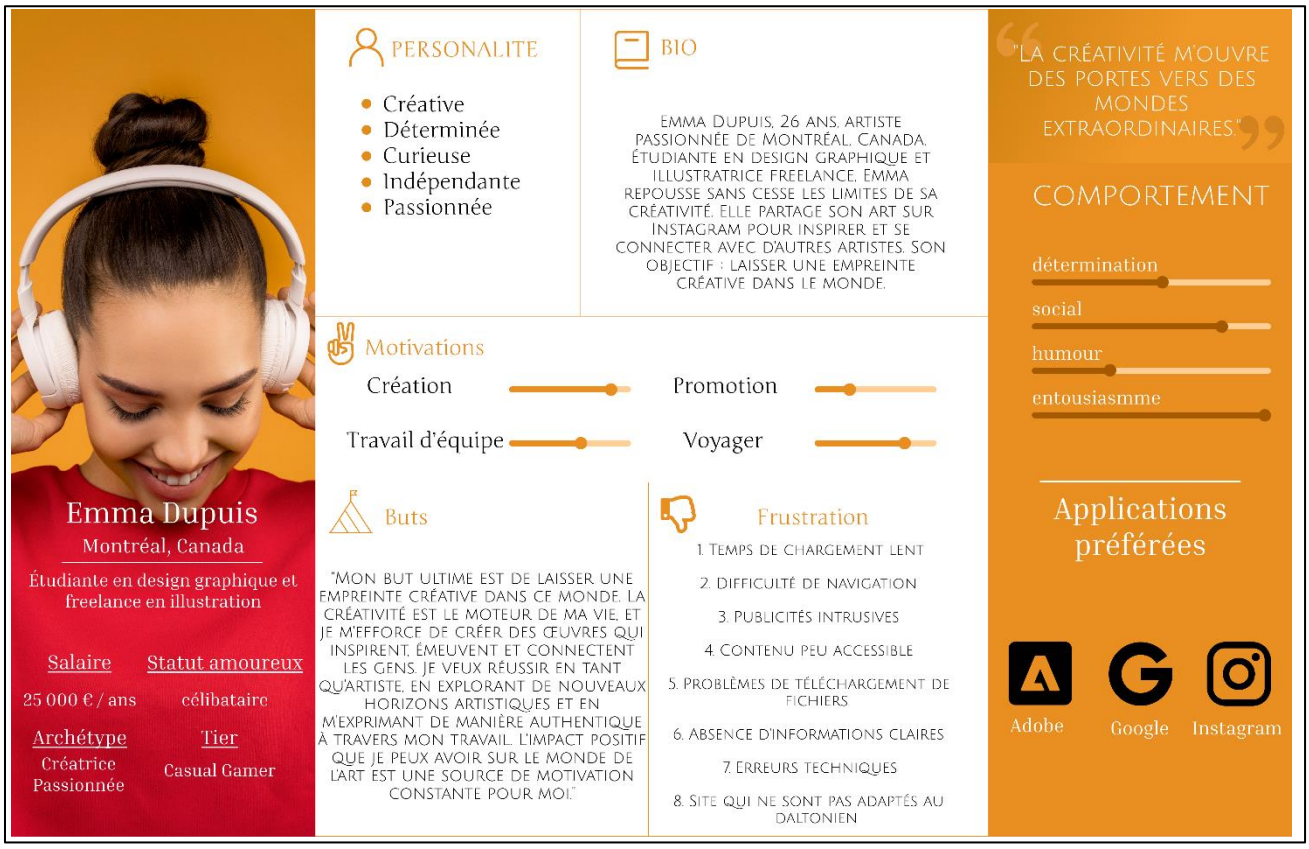
- Complexité excessive
- Problèmes techniques
- Manque de variété
- Absence de récompenses significatives

Persona N°1

Ce premier Persona a pour objectif de cibler le type de gameplay attendu par les utilisateurs cela va être surtout utile dans la partie programmation du jeu mais certain élément peut être intéressant pour le design/les maquettes du jeu Space Invader. Les 2 fonctionnalité essentielle qui ressorte de ce Persona et qui sont implémentable dans maquette sont le manque de variété et l'absence de récompenses significative.

J'ai réussi à implémenter ces deux « fonctionnalité » dans ma maquette en essayant de crée un Menu « High Score » et un « Shop » qui sorte du style général de l'application afin de provoquer une envie chez l'utilisateur. Je détaillerais plus ces éléments lors de la description de ces deux maquettes et dans le chapitre sur les palettes de couleurs.

Persona N°2



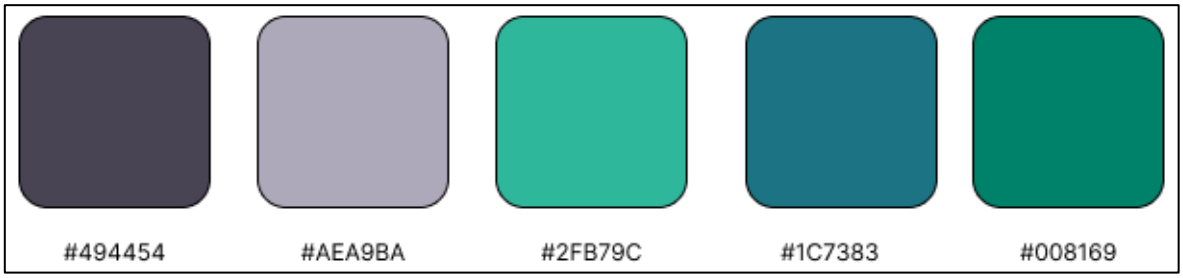
Persona N°2

Ce deuxième Persona a pour objectif de cibler l'accessibilité attendu par les utilisateurs. Parmi les éléments que ce Persona j'en ai retenu deux qui peuvent être intéressant pour le design/les maquettes du jeu Space Invader. Les 2 « fonctionnalité » essentielle qui ressorte de ce Persona et qui sont implémentable dans maquette sont l'accessibilité pour les daltoniens et les difficultés de navigation.

J'ai réussi à implémenter ces deux « fonctionnalité » dans ma maquette grâce à un réglage permettant de passer l'application en noir et blanc et à une navigation dans le menu du jeu très instinctive n'utilisant pas de bar de recherche où d'autres option rendant la navigation obsolète.

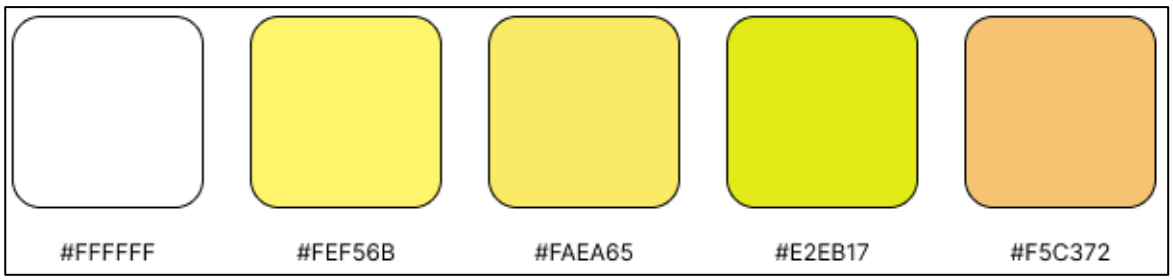
Palette Graphique

Durant ce projet j'ai opté pour deux palettes graphiques principales :



Palette Graphique N°1

Cette première palette graphique est une palette que j'ai généré grâce au site mycolor.space/ . J'ai choisi d'opter pour des couleurs plutôt reposantes, mais en gardant la couleur verte afin de rappeler les aliens du jeu Space Invader.



Palette Graphique N°2

Pour cette deuxième palette de couleur qui elle aussi a été générer grâce au site : mycolor.space/ ,j'ai opté pour la couleur jaune car on l'associe souvent à la vitalité, à l'or, au soleil et à la jovialité. De plus la couleur jaune procure des sentiments tels que la vitalité, l'énergie, la jovialité et l'enthousiasme. Le tout afin de procurer un sentiment de désir chez l'utilisateur. Ce sentiment de désir/cette palette de couleur je l'ai principalement utilisé dans la page du Shop et celle du High Score afin de garder l'utilisateur le plus longtemps dans mon jeu. Car l'utilisateur vas vouloir avoir les meilleurs vaisseaux ce qui va lui demander beaucoup de parties afin de collecter des crédits pour la boutique. De plus si la concurrence est dure et qu'il aime le jeu il va vouloir se hisser dans le top 10 de la page High Score, cette envie sera accentuée par la palette de couleur jaune.

Eco-conception

Voici les éléments d'écoconception pour lesquels j'ai fait attention durant la création de ma maquette :

- J'ai créé des titres de bouton/ une navigation pertinente pour que l'utilisateur n'ait pas à charger un plus grand nombre de page pour trouver ce qu'il veut.
- J'ai utilisé une polices web plutôt que d'inclure des polices personnalisées lourdes, ce qui réduit la charge des serveurs.
- J'ai conçu des maquettes modulaires où des éléments peuvent être réutilisés plutôt que de créer de multiples instances d'éléments similaires.

Afin de savoir quel point concerne l'éco-conception lors de la création de maquette j'ai utiliser le site web des 115 bonnes pratiques d'Eco-conception.

Lien du site : Collectif Conception responsable de service numérique (greenit.fr)

Accessibilité

Voici les différentes fonctionnalités d'accessibilités :

Le jeu est disponible en français et en anglais, le but étant de toucher le plus grand public.

Il y a un thème en couleur et un autre en noir et blanc afin de pouvoir aider les personne ayant des problème de vue tels que le daltonisme.

La navigation dans mon jeu est intuitive.

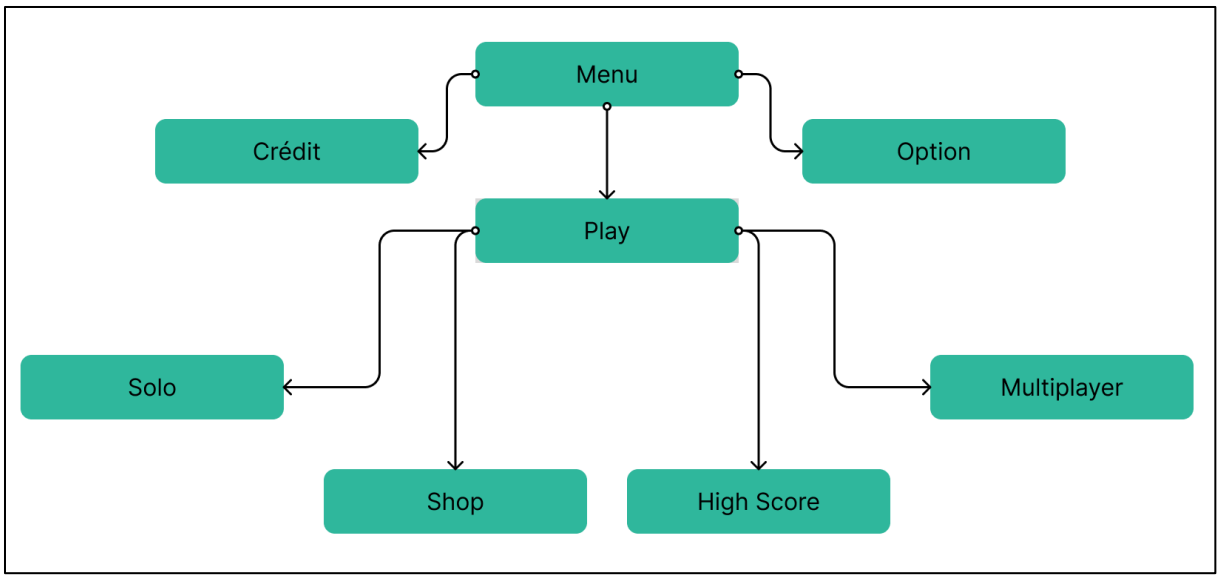
Conception

Dans la partie ci-dessous je vais décrire les différents menus que j'ai réalisé dans ma maquette haute-fidélité sur Figma. Chaque menu possède 4 style différent :

- Couleur - Français
- Couleur - Anglais
- Noir et blanc - Français
- Noir et blanc – Anglais

Selon moi le menu qui sera le plus utiliser sera Celui en Couleur –avec la langue Anglais, c'est pour cela que je le prendrais comme référence.

Ci-dessou voici l'arborecence de mes maquettes-hautes fidélitée.



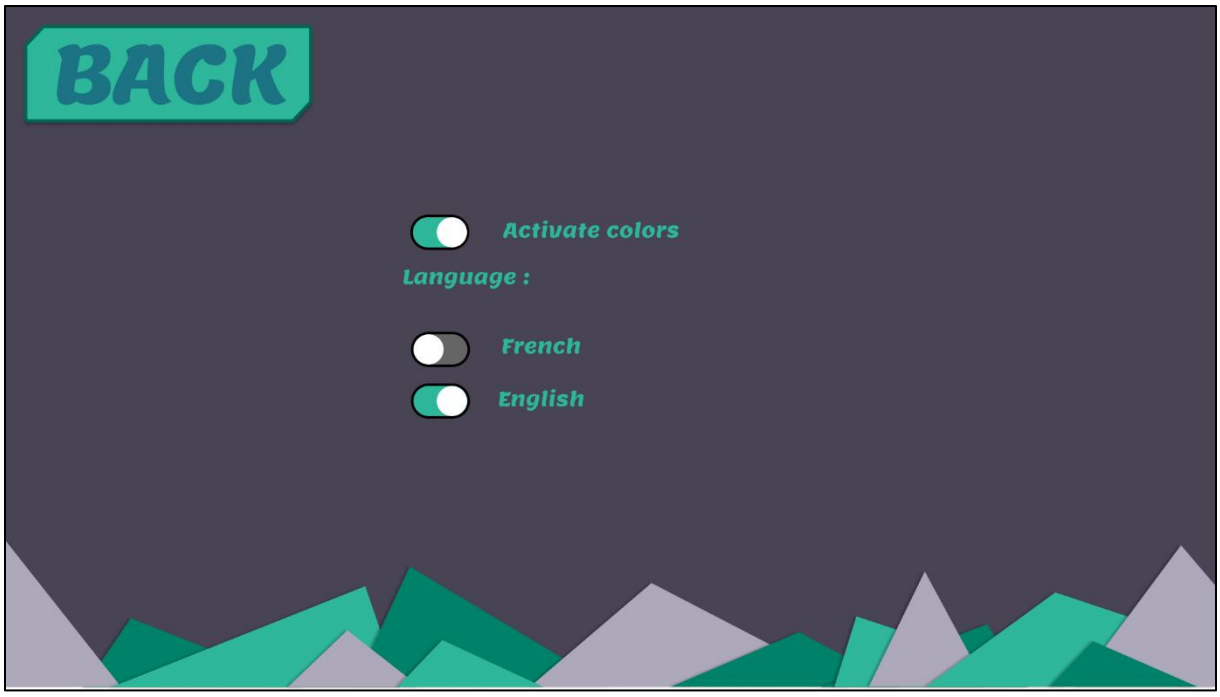
Définition des menus :

Dans cette partie je vais détailler chaque menu

Maquette « Menu »

La maquette « menu » est la page principale de mon jeu c'est pour cela quel comporte le logo « SpicyInvader » sur cette page il y a trois boutons.

Pour la page des crédits je n'ai pas d'image mais une vidéo que l'on peut retrouver sous : « SpaceInvader\Doc\Maquette_UX\Maquette_Crédis »
Cette page des crédits est inspirer des crédits de fin de film. Il y a un bouton permettant de retourner au « Menu »



Maquette « Option »

On peut voir sur ce menu « Option » les deux options essentielles au niveau de l'accessibilité (Noir et Blanc/Couleur et Français/Anglais) il y a aussi un bouton pour retourner au « menu »

Maquette « Play »

Sur cette page on retrouve les 2 bouton de lancement de parti (Solo et Multil). Deux autres boutons pour afficher le shop et le highscore sont présent. Le denier bouton (« Menu ») permet de retourner au menu.

Maquette « Shop »

Cette maquette représente le shop avec des prix, différentes armes et notre solde actuel. J'ai utilisé des armes car je ne savais pas si le client mettrait des vaisseaux ou des canons dans le shop. Il y a aussi un bouton permettant de retourner au menu « Play ».

Maquette « HighScore »

Dans cette dernière maquette on peut y voir le top 10 des meilleurs joueurs du mois et de tout les temps classer par score du meilleur au moins bon. Il y a aussi un bouton permettant de retourner au menu « Play ».

Tests

Voici mes tests :

Menu	Description	Pre-Condition	Test Steps	Test data	Expected Output	Actual Output
Main	Pouvoir aller à la page des credis et la page "play" en cliquant sur les boutons	L'application doit être installer	1. Cliquer sur play 2. Retourner au menu via le bouton menu 3. aller au menu credis 4. Retourner au menu via le bouton menu	Aucun	Sa fonctionne	.....
Shop	Dans le 2 "shop" il est possible d'acheter des armes	Avoir suffisamment d'argent dans le jeu	1. Cliquer sur le bouton acheter	Aucun	L'arme est achetée	.....
Reglage	Changer la laugue	Savoir parler les deux langue	1. Changer la langue 2. Report toutes les mauvaise traduction	Aucun	Un document Word indiquant chaque problème de traduction	.....

Conclusion

Au cours de ce projet, j'ai tiré plusieurs enseignements. J'ai pu apprendre à maîtriser Figma sans rencontrer de difficultés techniques majeures. Toutefois, en raison de mon manque d'expérience au début du projet, j'ai négligé la mise à jour d'IceScrum, ce qui a affecté la clarté de mon travail dans le domaine de l'UX. De plus, je n'ai pas suivi précisément le temps passé sur certaines tâches.

Personnellement j'ai eu beaucoup de plaisir à utiliser figma un peu moins pour Ice Scrum.

Base de Données

Importer la base de données

Pour importer la base de donner dans docker j'ai utilisé cette commande :

```
docker exec -i db mysql -uroot -proot < db_space_invaders.sql
```

Explication du fonctionnement de cette commande :

1. docker exec :
C'est la commande principale pour exécuter une commande à l'intérieur d'un conteneur Docker.

2. -i :

Cela signifie que l'on souhaite fournir une entrée interactive pour la commande.

3. db :

C'est le nom du conteneur Docker dans lequel l'on souhaite exécuter la commande. Le conteneur doit déjà être en cours d'exécution.

4. mysql :
C'est le début de la commande que l'on souhaite exécuter à l'intérieur du conteneur Docker.

5.-uroot :

Cela spécifie que l'on souhaite se connecter à la base de données MySQL en tant qu'utilisateur "root".

6.-proot :
Cela spécifie le mot de passe pour l'utilisateur "root".

7. < db_space_invaders.sql:

Cela signifie que le contenu du fichier SQL est utilisé comme entrée pour la commande MySQL.

8.Résumer :

En résumé, cette commande exécute un script SQL stocké dans le fichier db_space_invaders.sql à l'intérieur du conteneur Docker en utilisant l'utilisateur "root" avec le mot de passe "root" pour se connecter à la base de données MySQL. Attention à bien s'assurer que le conteneur Docker est en cours d'exécution et que le fichier SQL est accessible depuis l'emplacement où l'on exécute cette commande.

Gestions des utilisateurs

Introduction

Cette partie a pour but principale de décrire les commandes nécessaires à la création des rôles « Administrateur du jeu », « Joueur » et « Gestionnaire de la boutique ». Afin que les différents utilisateurs qui vont devoir interagir avec la base de données ait accès uniquement aux commandes nécessaires. Le tout dans le but de maintenir la sécurité et le bon fonctionnement de la base de données.

1. Administrateur du jeu

Enoncé :

Le rôle d'administrateur du jeu peut créer, lire, mettre à jour et supprimer (CRUD) n'importe quelle table. Il peut aussi gérer les utilisateurs et leurs privilèges.

Création du rôle :

```
CREATE ROLE 'Admin-game';
GRANT CREATE, SELECT, UPDATE, DELETE ON *.* TO 'Admin-game';
```

Création d'un utilisateur ayant le rôle « Admin- game » :

```
CREATE USER 'admin-Lucas'@'localhost' IDENTIFIED BY 'Password';
GRANT 'Admin-game' TO 'admin-Lucas'@'localhost';
```

Explication des requêtes :

1. `CREATE ROLE 'Admin-game';`

- Cette commande crée le rôle 'Admin-game'.

2. `GRANT CREATE, SELECT, UPDATE, DELETE ON *.* TO 'Admin-game';`

- Cette commande accorde les autorisations *CRUD* sur toutes les tables au rôle 'Admin-game'.

3. `CREATE USER 'admin-Lucas'@'localhost' IDENTIFIED BY 'Password';`

- Cette commande crée l'utilisateur 'admin -Lucas' qui peut se connecter depuis l'hôte 'localhost' en utilisant le mot de passe 'Password'.

4. `GRANT 'Admin-game' TO 'admin-Lucas'@'localhost';`

- Cette commande accorde le rôle 'Admin-game' à l'utilisateur 'admin-Lucas' Cela signifie que l'utilisateur 'admin-Lucas' se voit attribuer les autorisations définies dans le rôle 'Admin-game' lorsqu'il se connecte depuis 'localhost'.

5. Résumer

En résumé, ces requêtes SQL créent le rôle 'Admin-game', et créent l'utilisateur « admin-Lucas », puis attribuent le rôle 'Admin-game' à l'utilisateur 'admin-Lucas' Cela permet à 'admin-Lucas' d'exécuter les actions CRUD sur toutes les tables la base de données.

2. Joueur

Enoncé :

Le rôle de joueur peut lire les informations des armes, créer une commande et lire toutes les commandes.

Création du rôle :

```
CREATE ROLE `Player`;
GRANT SELECT ON db_space_invaders.t_arme TO `Player`;
GRANT CREATE ON db_space_invaders.t_commande TO `Player`;
GRANT SELECT ON db_space_invaders.t_commande TO `Player`;
```

Création d'un utilisateur ayant le rôle « Player » :

```
CREATE USER 'player-Lucas'@'localhost' IDENTIFIED BY 'Password';
GRANT 'Player' TO 'player-Lucas' @'localhost';
```

Explication des requêtes :

1. `CREATE ROLE 'Player';`

- Cette commande crée le rôle 'Player'.

2. `GRANT SELECT ON db_space_invaders.t_arme TO 'Player';`

- Cette commande accorde l'autorisation SELECT sur la table 't_arme' dans la base de données 'db_space_invaders' au rôle 'Player'.

3. `GRANT CREATE ON db_space_invaders.t_commande TO 'Player' ; `

- Cette commande accorde l'autorisation CREATE sur la table 't_commande' dans la base de données 'db_space_invaders' au rôle 'Player'.

4. `GRANT SELECT ON db_space_invaders.t_commande TO 'Player' ; `

- Cette commande accorde l'autorisation SELECT sur la table 't_commande' dans la base de données 'db_space_invaders' au rôle 'Player'.

5. `CREATE USER 'player-Lucas'@'localhost' IDENTIFIED BY 'Password' ; `

- Cette commande crée l'utilisateur 'player -Lucas' qui peut se connecter depuis l'hôte 'localhost' en utilisant le mot de passe 'Password'.

6. `GRANT 'Player' TO 'player-Lucas'@'localhost';`

- Cette commande attribue le rôle 'Player' à l'utilisateur 'player-Lucas'. Cela signifie que l'utilisateur se voit attribuer les autorisations du rôle 'Player' lorsqu'il se connecte depuis 'localhost'.

7. Résumer

En résumé, ces commandes SQL créent le rôle 'Player', lui accordent diverses autorisations SELECT et CREATE sur différentes tables de la base de données 'db_space_invaders', créent l'utilisateur 'player-Lucas', puis attribuent le rôle 'Player' à l'utilisateur 'player-Lucas'.

3. Gestionnaire de la boutique

Enoncé :

Le rôle de gestionnaire de la boutique peut lire les informations sur tous les joueurs, mettre à jour, lire et supprimer des armes et lire toutes les commandes.

Création du rôle :

```
CREATE ROLE `ShopKeeper`;
GRANT SELECT ON db_space_invaders.t_joueur TO `ShopKeeper`;
GRANT UPDATE, SELECT, INSERT, DELETE ON db_space_invaders.t_arme TO
`ShopKeeper`;
GRANT SELECT ON db_space_invaders.t_commande TO `ShopKeeper`;
```

Création de d'un utilisateur ayant le rôle 'Player' :

```
CREATE USER 'shopkeeper-Lucas'@'localhost' IDENTIFIED BY
'Password';
GRANT 'ShopKeeper' TO 'shopkeeper-Lucas'@'localhost';
```

Explication des requêtes :

Bien sûr, je vais expliquer ces nouvelles commandes SQL une par une :

1. `CREATE ROLE 'ShopKeeper';`

- Cette commande crée le rôle 'ShopKeeper'.

2. `GRANT SELECT ON db_space_invaders.t_joueur TO 'ShopKeeper' ; `

- Cette commande accorde l'autorisation SELECT sur la table 't_joueur' dans la base de données 'db_space_invaders' au rôle 'ShopKeeper'

3. `GRANT UPDATE, SELECT, INSERT, DELETE ON db_space_invaders.t_arme TO 'ShopKeeper' ; `

- Cette commande accorde les autorisations UPDATE, SELECT, INSERT et DELETE sur la table 't_arme' dans la base de données 'db_space_invaders' au rôle 'ShopKeeper'.

4. `GRANT SELECT ON db_space_invaders.t_commande TO 'ShopKeeper' ; `

- Cette commande accorde l'autorisation SELECT sur la table 't_commande' dans la base de données 'db_space_invaders' au rôle 'ShopKeeper'.

5. `CREATE USER 'shopkeeper-Lucas'@'localhost' IDENTIFIED BY 'Password' ; `

- Cette commande crée un utilisateur nommé 'shopkeeper-Lucas' qui peut se connecter depuis l'hôte 'localhost' en utilisant le mot de passe 'Password'.

6. `GRANT 'ShopKeeper' TO 'shopkeeper-Lucas'@'localhost' ; `

- Cette commande attribue le rôle 'ShopKeeper' à l'utilisateur 'shopkeeper -Lucas'. Cela signifie que l'utilisateur se voit attribuer les autorisations du rôle 'ShopKeeper' lorsqu'il se connecte depuis 'localhost'.

7. Résumer

- En résumé, ces commandes SQL créent le rôle appelé 'ShopKeeper', lui accordent diverses autorisations SELECT, UPDATE, INSERT et DELETE sur différentes tables de la base de données 'db_space_invaders', créent l'utilisateur 'shopkeeper-Lucas' puis attribuent le rôle 'ShopKeeper' à l'utilisateur 'shopkeeper-Lucas'.

Requêtes de sélection

Introduction

Cette partie a pour but principale de décrire les 10 requêtes SQL que l'on doit réaliser dans le cadre du projet Space Invader (DB)

Requête n°1

Enoncé :

Sélectionner les 5 joueurs qui ont le meilleur score et les classés dans l'ordre décroissant.

```sql
SELECT * FROM `t_joueur` ORDER BY jouNombrePoints DESC LIMIT 5;
```

Explication :

Cette requête SQL effectue un SELECT de toute les données (*) de la table t_joueur.

Ensuite ces données sont triées (ORDER BY) des plus grands scores au plus petits (DESC)

Enfin, on limite le nombre de résulta au 5 premier via l'emploi de la clause (LIMIT 5).

Requête n°2

Enoncé :

Trouver le prix maximum, minimum et moyen des armes. Les colonnes doivent avoir pour nom « Prix Maximum », « PrixMinimum » et « PrixMoyen »

```sql
SELECT MAX(armPrix) as PrixMaximum,
MIN(armPrix) as PrixMinimum,
AVG(armPrix) as PrixMoyen
FROM `t_arme`;
```

Explication :

Cette requête SQL sélectionne trois valeurs calculées à partir de la colonne armPrix de la table t_arme.

La première valeur est le prix maximum (MAX) des armes et est renommée en tant que "PrixMaximum" à l'aide de l'alias (AS).

La deuxième valeur est le prix minimum (MIN) des armes et est renommée en tant que "PrixMinimum".

La troisième valeur est le prix moyen (AVG) des armes et est renommée en tant que "PrixMoyen".

La source de données est la table t_arme, ce qui signifie que ces valeurs seront calculées à partir des données de cette table.

Requête n°3

Enoncé :

Trouver le nombre total de commandes par joueur et trier du plus grand nombre au plus petit. La 1ère colonne aura pour nom "IdJoueur", la 2ème colonne aura pour nom "NombreCommandes"

```sql
SELECT fkJoueur as idJoueur,
COUNT(idCommande) as NombreCommandes
FROM `t_commande`
GROUP BY fkJoueur;
```

Explication :

Cette requête SQL sélectionne deux valeurs calculées à partir de la table t_commande.

La première valeur est renommée en tant que "idJoueur" à l'aide de l'alias (AS), et elle correspond à la colonne fkJoueur. Cela signifie qu'elle renverra les identifiants de joueurs associés aux commandes et pas leur pseudo.

La deuxième valeur est renommée en tant que "NombreCommandes" à l'aide de l'alias (AS). Elle est calculée en utilisant la fonction (COUNT) et correspond à la quantité de commandes (idCommande) effectuées par chaque joueur.

La clause GROUP BY fkJoueur regroupe les résultats en fonction de la colonne fkJoueur. Cela signifie que les commandes sont regroupées par joueur, de sorte que l'on obtient le nombre de commandes pour chaque joueur distinct.

Requête n°4

Enoncé :

Trouver les joueurs qui ont passé plus de 2 commandes. La 1ère colonne aura pour nom "IdJoueur", la 2ème colonne aura pour nom "NombreCommandes "

```sql
SELECT fkJoueur as IdJoueur,
COUNT(idCommande) as NombreCommandes
FROM t_commande
GROUP BY fkJoueur
HAVING COUNT(idCommande) > 2;
```

Explication :

Cette requête est similaire à la requête précédente mais dans ce cas-là on utilise la clause (HAVING) afin de filtrer le groupe fkJoueur

Requête n°5

Enoncé :

Trouver le pseudo du joueur et le nom de l'arme pour chaque commande.

```sql
SELECT t_joueur.jouPseudo, t_arme.armNom
FROM t_joueur
INNER JOIN t_commande ON t_joueur.idJoueur = t_commande.fkJoueur
INNER JOIN t_detail_commande ON t_commande.idCommande = t_detail_commande.fkCommande
INNER JOIN t_arme ON t_detail_commande.fkArme = t_arme.idArme;
```

Explication :

Cette requête sélectionne les colonnes `jouPseudo` de la table `t_joueur` et `armNom` de la table `t_arme`. Afin d'accéder à `armNom` la requête utilise 3 jointures qui passe par les tables t_commande, t_detail_commande et t_arme

En résumé, cette requête utilise des informations des tables `t_joueur`, `t_commande`, `t_detail_commande`, et `t_arme` en utilisant des jointures pour obtenir le nom du joueur (`jouPseudo`) et le nom de l'arme (`armNom`) associés à chaque commande dans la base de données.

Requête n°6

Enoncé :

Trouver le total dépensé par chaque joueur en ordonnant par le montant le plus élevé en premier, et limiter aux 10 premiers joueurs. La 1ère colonne doit avoir pour nom "IdJoueur" et la 2ème colonne "TotalDepense".

```sql
SELECT t_joueur.idJoueur as IdJoueur,
SUM(t_detail_commande.detQuantiteCommande * t_arme.armPrix) as TotalDepense
FROM t_joueur
INNER JOIN t_commande ON t_joueur.idJoueur = t_commande.fkJoueur
INNER JOIN t_detail_commande ON t_commande.idCommande = t_detail_commande.fkCommande
INNER JOIN t_arme ON t_detail_commande.fkArme = t_arme.idArme
GROUP BY t_joueur.idJoueur
ORDER BY TotalDepense DESC
LIMIT 10;
```

Explication :

Cette requête SQL permet de fournir une liste des 10 joueurs qui ont dépensé le plus d'argent dans des achats d'armes.

La requête commence par sélectionner deux colonnes : `IdJoueur` et `TotalDepense`.

`IdJoueur` provient de la table `t_joueur`, et `TotalDepense` est une valeur calculée résultant de la somme des dépenses de chaque joueur. On doit utiliser une multiplication car plusieurs armes peuvent êtres acheter en une seule commande.

La première jointure relie les joueurs aux commandes qu'ils ont passées, la deuxième jointure relie les commandes aux détails de ces commandes, et la troisième jointure relie les détails des commandes aux armes qui ont été achetées.

La clause `GROUP BY` est utilisée pour regrouper les résultats par joueur, en utilisant la colonne `IdJoueur`. Cela signifie que les dépenses de chaque joueur sont sommées individuellement.

Les résultats sont triés en fonction de la colonne calculée `TotalDepense` dans un ordre décroissant, ce qui signifie que les joueurs qui ont dépensé le plus apparaissent en premier.

Enfin, la requête limite les résultats aux 10 premiers joueurs en utilisant la clause `LIMIT 10`.

Requête n°7

Enoncé :

Récupérez tous les joueurs et leurs commandes, même s'ils n'ont pas passé de commande. Dans cet exemple, même si un joueur n'a jamais passé de commande, il sera quand même listé, avec des valeurs 'NULL' pour les champs de la table 't_commande',

```sql
SELECT t_joueur.idJoueur,
t_joueur.jouPseudo,
t_joueur.jouNombrePoints,
t_commande.comNumeroCommande
FROM t_joueur
LEFT JOIN t_commande ON t_joueur.idJoueur = t_commande.fkJoueur;
```

Explication :

Cette requête SQL permet de récupérer une liste de joueurs et de leurs commandes, même s'ils n'ont pas passé de commande. Elle utilise une jointure de type LEFT JOIN pour cela.

La requête commence par sélectionner plusieurs colonnes dans les tables `t_joueur` et `t_commande`. Elle récupère les colonnes `idJoueur`, `jouPseudo`, `jouNombrePoints` de la table `t_joueur`, ainsi que la colonne `comNumeroCommande` de la table `t_commande`.

La requête utilise ensuite une jointure de type LEFT JOIN entre les tables `t_joueur` et `t_commande`. Cela signifie que tous les enregistrements de la table `t_joueur` seront inclus dans le résultat, même s'ils n'ont pas de correspondance dans la table `t_commande`.

En résumé, cette requête donne une liste de tous les joueurs, y compris ceux qui n'ont pas passé de commande, en utilisant une jointure de type LEFT JOIN entre les tables `t_joueur` et `t_commande`. Cela permet d'obtenir un résultat qui inclut les joueurs sans commande, avec des valeurs NULL pour les champs de la table `t_commande` correspondants.

Requête n°8

Enoncé :

Récupérer toutes les commandes et afficher le pseudo du joueur si elle existe, sinon montrer 'NULL' pour le pseudo.

```sql
SELECT t_joueur.jouPseudo,
t_commande.comNumeroCommande
FROM t_commande
RIGHT JOIN t_joueur ON t_commande.fkJoueur = t_joueur.idJoueur;
```

Explication :

Cette requête SQL permet de récupérer toutes les commandes et d'afficher le pseudo du joueur s'il existe. Si le joueur n'est pas associé à la commande, le résultat affichera 'NULL' pour le pseudo.

La requête commence par sélectionner deux colonnes : `jouPseudo` de la table `t_joueur` et `comNumeroCommande` de la table `t_commande`.

La requête utilise ensuite une jointure de type RIGHT JOIN entre les tables `t_commande` et `t_joueur`. Cela signifie que tous les enregistrements de la table `t_commande` seront inclus dans le résultat, même s'ils n'ont pas de correspondance dans la table `t_joueur`.

En résumé, cette requête permet de récupérer toutes les commandes et d'afficher le pseudo du joueur s'il est associé à la commande. Si un joueur n'est pas associé à une commande donnée, le résultat affichera 'NULL' pour le pseudo, en utilisant une jointure de type RIGHT JOIN entre les tables `t_joueur` et `t_commande`.

Requête n°9

Enoncé :

Trouver le nombre total d'armes achetées par chaque joueur (même si ce joueur n'a acheté aucune Arme).

```sql
SELECT t_joueur.jouPseudo, SUM(t_detail_commande.detQuantiteCommande)
FROM t_joueur
LEFT JOIN t_commande ON t_joueur.idJoueur = t_commande.fkJoueur
LEFT JOIN t_detail_commande ON t_commande.idCommande = t_detail_commande.fkCommande
LEFT JOIN t_arme ON t_detail_commande.fkArme = t_arme.idArme
GROUP BY t_joueur.idJoueur;
```

Explication :

Cette requête SQL permet de trouver le nombre total d'armes achetées par chaque joueur, même si un joueur n'a acheté aucune arme.

La requête commence par sélectionner deux colonnes : `jouPseudo` de la table `t_joueur` et `SUM(t_detail_commande.detQuantiteCommande)` pour obtenir le nombre total d'armes achetées par chaque joueur. Il est important de noter que dans cette commande l'utilisation du (SUM) est dû aux faites que detQuantiteCommande peut contenir plusieurs armes.

La requête utilise ensuite trois jointures de type LEFT JOIN pour connecter les tables. Cela signifie que tous les joueurs seront inclus dans le résultat, même s'ils n'ont pas passé de commande ni acheté d'armes.

- La première jointure relie les joueurs aux commandes qu'ils ont passées.
- La deuxième jointure relie les commandes aux détails de ces commandes.
- La troisième jointure relie les détails des commandes aux armes achetées.

La clause `GROUP BY` est utilisée pour regrouper les résultats par joueur, en utilisant la colonne `idJoueur` de la table `t_joueur`. Cela signifie que le nombre total d'armes achetées est calculé pour chaque joueur distinct.

En résumé, cette requête permet de trouver le nombre total d'armes achetées par chaque joueur, même si un joueur n'a acheté aucune arme. Elle utilise des jointures de type LEFT JOIN pour inclure tous les joueurs dans le résultat, et le résultat affiche 'NULL' si un joueur n'a pas passé de commande ni acheté d'armes.

Requête n°10

Enoncé :

Trouver les joueurs qui ont achetés plus de 3 types d'armes différentes.

```sql
SELECT t_joueur.jouPseudo, COUNT(DISTINCT t_arme.idArme)
FROM t_joueur
LEFT JOIN t_commande ON t_joueur.idJoueur = t_commande.fkJoueur
LEFT JOIN t_detail_commande ON t_commande.idCommande = t_detail_commande.fkCommande
LEFT JOIN t_arme ON t_detail_commande.fkArme = t_arme.idArme
GROUP BY t_joueur.idJoueur
HAVING COUNT(DISTINCT t_arme.idArme) > 3;
```

Explication :

Cette requête SQL permet de trouver les joueurs qui ont acheté plus de 3 types d'armes différentes.

La requête commence par sélectionner deux colonnes : `jouPseudo` de la table `t_joueur` et `COUNT (DISTINCT t_arme.idArme)` pour compter le nombre d'armes différentes achetées par chaque joueur.

La requête utilise trois jointures de type LEFT JOIN pour connecter les tables. Cela signifie que tous les joueurs seront inclus dans le résultat, même s'ils n'ont pas passé de commande ni acheté d'armes.

- La première jointure relie les joueurs aux commandes qu'ils ont passées.
- La deuxième jointure relie les commandes aux détails de ces commandes.
- La troisième jointure relie les détails des commandes aux armes achetées.

La clause `GROUP BY` est utilisée pour regrouper les résultats par joueur, en utilisant la colonne `idJoueur` de la table `t_joueur`. Cela signifie que le nombre d'armes différentes achetées est calculé pour chaque joueur distinct.

La clause `HAVING` est utilisée pour filtrer les résultats en ne montrant que les joueurs pour lesquels le nombre d'armes différentes achetées est supérieur à 3.

En résumé, cette requête permet de trouver les joueurs qui ont acheté plus de 3 types d'armes différentes en utilisant des jointures de type LEFT JOIN et en appliquant un filtre avec la clause HAVING. Le résultat affiche le pseudo du joueur et le nombre d'armes différentes achetées pour chaque joueur qui satisfait la condition.

Création des index

Pourquoi certains index sont déjà présents dans la base de données ?

Certains index existent déjà car MySQL génère automatiquement des index sur les clés primaires et les clés étrangères. Ces index sont créés de manière automatique en raison de la fréquence d'utilisation des clés primaires et des clés étrangères dans les requêtes, particulièrement lorsqu'il s'agit de réaliser des jointures entre les tables.

Quels sont les avantages et les inconvénients des index ?

Avantages :

Les index offrent une structuration des données, améliorant considérablement les performances des opérations de recherche dans les données.

Inconvénients :

Les index consomment de l'espace mémoire.

Ils peuvent entraîner des ralentissements dans les requêtes, car l'index doit être mis à jour à chaque modification des données, que ce soit une insertion, une modification ou une suppression.

De plus, les index peuvent ralentir les opérations de mise à jour (UPDATE) des données, car chaque mise à jour doit également mettre à jour l'index correspondant.

Sur quel champ cela pourrait être pertinent d'ajouter un index ?

Sur les clefs primaires et les clefs étrangères (dans le cas où elle n'ont pas été automatiquement créé par le SGBDR utilisé). Cela permettrait d'effectuer des requêtes plus rapidement lorsque l'on souhaite effectuer des jointures. De plus des indexes peuvent être rajoutés

Sauvegardes et Restaurations

Mysqldump -u votre_nom_utilisateur -proot db_space_invades > backup_db_space_invaders.sql

Mysql -u votre_nom_utilisateur -p db_space_invaders < backup_db_space_invaders.sql

Dans le cadre de la gestion de bases de données, l'une des tâches cruciales est de garantir la sécurité des données. Cela implique de mettre en place des mécanismes robustes pour sauvegarder régulièrement les informations stockées dans une base de données et de disposer de la possibilité de restaurer ces données en cas de besoin. Dans cette section, je vais aborder les processus de sauvegarde et de restauration d'une base de données, en mettant l'accent sur la base de données `db_space_invaders`.

Sauvegarde de la Base de Données `db_space_invaders`

La sauvegarde d'une base de données MySQL peut être effectuée à l'aide de la commande `mysqldump` suivante :

```
mysqldump -u votre_nom_utilisateur -proot db_space_invaders
> backup_db_space_invaders.sql
```

Explication :

1. `mysqldump` est l'utilitaire MySQL pour les sauvegardes.
2. `-u` spécifie le nom de l'utilisateur MySQL utilisé pour se connecter.
3. `-p` demande le mot de passe de l'utilisateur.
4. `db_space_invaders` est le nom de la base de données à sauvegarder.
5. `> backup_db_space_invaders.sql` redirige la sortie de la commande vers le fichier `backup_db_space_invaders.sql`, qui contiendra la sauvegarde de la base de données.

Restauration de la Base de Données `db_space_invaders`

La restauration de la base de données à partir d'une sauvegarde s'effectue à l'aide de la commande `mysql` suivante :

```
mysql -u votre_nom_utilisateur -p db_space_invaders < backup_db_space_invaders.sql
```

Explication :

1. `mysql` est l'utilitaire MySQL pour restaurer une base de données.
2. `-u` spécifie le nom de l'utilisateur MySQL utilisé pour se connecter.
3. `-p` demande le mot de passe de l'utilisateur.
4. `db_space_invaders` est le nom de la base de données dans laquelle la sauvegarde sera restaurée.
5. `< backup_db_space_invaders.sql` indique à MySQL de lire le fichier `backup_db_space_invaders.sql` et de restaurer la base de données à partir de ce fichier.

Utilisation de la DB dans le code

Introduction

Cette partie a pour but principale de décrire le code nécessaire à la connexion de la base de données. Afin de pouvoir consulter et enregistrer le score des joueurs. Le tout dans le but de rendre le jeu Space Invader plus compétitif.

Afin d'effectuer la connexion j'ai dû installer le package NuGet Mysql.Data dans mon programme.

Sauvegarde du score

Afin de sauvegarder le score dans la base de données je dois d'abord avoir accès au score que l'utilisateur à effectuer. Pour ce faire dans la page MainWindows.xaml.cs (page ou le jeu se déroule) je sauvegarde le score effectuer dans la page GameOver.xaml.cs (page qui s'ouvre automatiquement lorsque l'on perd la partie) via cette commande : « *gameOverWindow.score = score ;* » Dans le cas où le joueur veut sauvegarder son score (en appuient sur le bouton « Save Score » qui redirige vers la page SaveScore.xaml), il faut faire passer la valeur du score effectuer via cette commande : « *saveScoreWindow.score = score ;* ».

Ensuite il faut avoir accès au pseudo du joueur, pour ce faire dans le code SaveScore.xaml la TextBox « *TextBoxPseudo* » à été crée ce qui permet à l'utilisateur de rentrer son pseudo. La valeur de « *TextBoxPseudo »* et récupérer grâce à cette commande : « *string player = TextBoxPseudo.Text ;* ».

Maintenant que l'on dispose des valeurs importantes il faut que l'on se connecte à la base de données. Il est facile de se connecter à la base de données via le package NuGet Mysql.Data, car il ne faut que deux commandes :

```
string connectionString = "server=localhost; uid=root;
pwd=root; database=db_space_invaders; port=6033;";
```

Commande n°1

Explication :

Cette commande est un élément clé pour établir une connexion à une base de données MySQL, car c'est le string « connectionString » qui contient toutes les informations nécessaires à la connexion.

1. `server=localhost` : indique l'emplacement du serveur MySQL auquel l'on souhaite se connecter. Dans mon cas, le serveur est sur le même PC que l'application, d'où l'utilisation de "localhost". Si le

serveur MySQL est sur une autre machine, il est important de spécifierez son adresse IP ou son nom de domaine à la place.

2. `uid=root` : C'est le nom d'utilisateur utilisé pour se connecter à la base de données. Dans mon cas, l'utilisateur est "root".

3. `pwd=root` : C'est le mot de passe de l'utilisateur. Dans mon cas, le mot de passe est également "root".

4. `database=db_space_invaders` : Il s'agit du nom de la base de données à laquelle l'on souhaite se connecter. Dans mon cas, la base de données est "db_space_invaders".

5. `port=6033` : C'est le numéro de port sur lequel le serveur MySQL écoute les connexions entrantes. Par défaut, le port MySQL est 3306, mais dans mon cas, je le spécifie comme 6033 car il m'est arrivé d'utiliser un autre port pour docker.

6. En résumé, cette chaîne de connexion contient toutes les informations nécessaires pour se connecter à la base de données MySQL.

```
MySqlConnection connection = new MySqlConnection(connectionString);
```

Commande n°2

Explication :

Cette commande établie une connexion à la base de données MySQL. La variable `connection` devient l'objet de connexion que l'on utilise pour interagir avec la base de données, exécuter des requêtes SQL et effectuer d'autres opérations de base de données.

1. `MySqlConnection` : C'est le type de l'objet que l'on est en train de créer. Dans ce cas, il s'agit d'une instance de la classe `MySqlConnection`. Cette classe est utilisée pour représenter une connexion à une base de données MySQL.

2. `connection` : C'est le nom de la variable que l'on a créez pour stocker l'instance de la connexion.

3. `= new MySqlConnection(connectionString); ` : Cela crée une nouvelle instance de la classe `MySqlConnection` en utilisant la chaîne de connexion `connectionString` que l'on a déclaré plus tôt dans le code.

Une fois la connexion établis il suffit d'effectuer la requête suivante afin d'enregistrer le score et le player :

```
INSERT INTO db_space_invaders.t_joueur (jouPseudo, jouNombrePoints) VALUES
(@jouPseudo, @jouNombrePoints)
```

Affichage du HighScore

Pour pouvoir afficher le HighScore (le pseudo, le score et le rang) des 10 meilleur joueur dans la page highScore.xaml. Il faut tout d'abord établir la connexion à la base de données (cette partie est similaire à celle de la section « *Sauvegarde du score* » du rapport). Une fois la connexion établie il ne reste plus qu'à exécuter cette requête :

```
SELECT jouPseudo, jouNombrePoints FROM db_space_invaders.t_joueur ORDER BY
jouNombrePoints DESC LIMIT 10;
```

Conclusion

Dans cette partie « *Utilisation de la DB dans le code* » j'ai principalement détailler les commande et requête essentiel à la connexion de la DB. Dans le cas où vous voulez en savoir plus sur la logique du code il faut aller regarder ces 4 fichier : GameOver.xaml, GameOver.xaml.cs , SaveScore.xaml et SaveScore.xaml.cs .Attention dans le cas d'une erreur désinstallée le package NuGet "MySql.Data" et réinstaller cela devrait réglé l'erreur.

Conclusion de la partie Base de Donnée

Cette partie du projet m'a permis de renforcer mes capaciter MySQL et m'a fait découvrir comment intégrer la base de données dans un code c#.

Toutes les demandes du cahier des charges ont été respecté.

Dans le cas où le projet devrait être réaliser à nouveau je passerais moins de temp à crée les requêtes et à les expliquer. A la place je créerais une planification plus détaillée.

Globalement je suis plutôt content de mon travail sur cette partie du projet. J'aurais aimé pouvoir crée un code contenant plus de requête, mais étant donné que je me suis lancer le défi de coder un jeu en WPF je n'ai pas eu le temps de me pencher sur l'ajout de fonctionnalités supplémentaires.

Programmation Orientée objets

Introduction

Dans le cadre du projet de programmation orientée objet à l'ETML, l'objectif est de créer une réplique du célèbre jeu "Space Invaders".

Le projet peut être mis en œuvre soit en version console, soit en utilisant WPF. Pour ma part, j'ai décidé de le développer en utilisant WPF, car je considère que c'est une opportunité exceptionnelle d'approfondir mes connaissances en programmation WPF. Avant ce projet, je n'avais jamais eu l'occasion de m'y essayer, mais j'ai toujours ressenti l'envie de le faire.

Pour la gestion du projet, IceScrum est utilisé, et toutes les Stories doivent être validées par le chef de projet avant de commencer l'implémentation. Chaque Story comprend des tests d'acceptance et éventuellement des maquettes. Le projet se déroule en un seul sprint et doit inclure des tests unitaires pour évaluer la fonctionnalité de chaque fonction.

Analyse fonctionnelle

Mon analyse fonctionnelle a été généré automatiquement par Ice Scrub et se situe dans « doc/Analyse_fonctionnel(JDT) »

Analyse Technique

Diagramme de classe

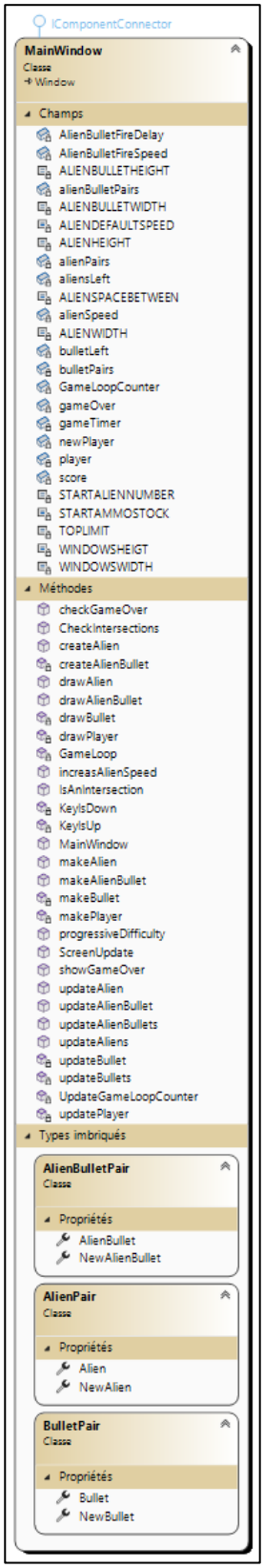


Diagramme de la page Main Windows

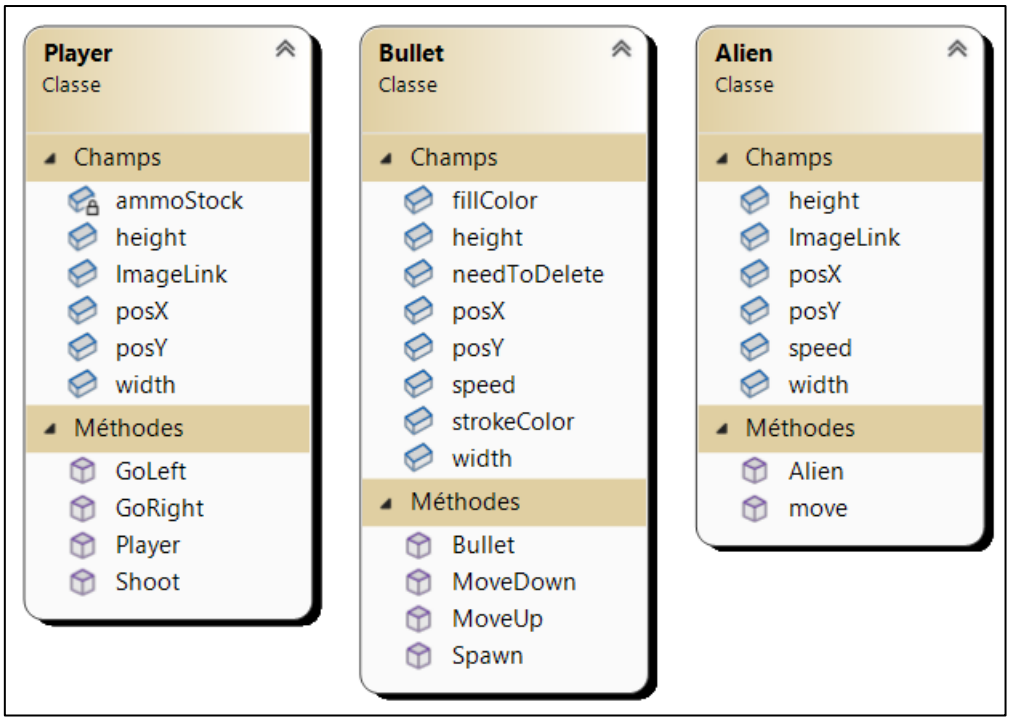


Diagramme du projet Model.csproj

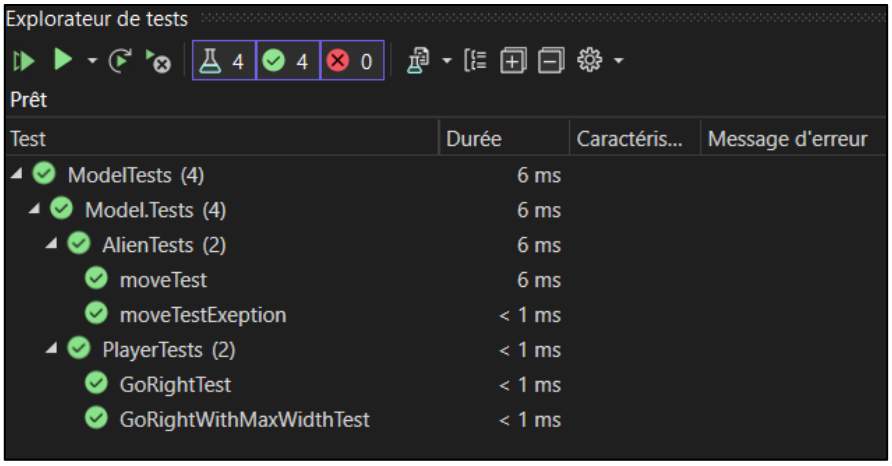
Doc Fx

J'ai généré l'explication de mon code à l'aide de Docfx, et les fichiers correspondants se trouvent dans le répertoire "doc/Analyse_technique_(docfx)". Ils sont ensuite organisés en dossiers en fonction du projet de classe auquel ils appartiennent. Si vous voulez voir plus de détail il suffit d'aller voir le code qui est facilement compréhensible grâce au bon nommage des variables et des méthodes et à l'utilisation de la POO.

Tests Unitaires

Dans le cadre de ce projet j'ai réalisé quatre Tests unitaires c#. Deux pour la méthode « move » de la classe alien et deux autres pour la méthode « GoRight » de la classe Player. Ces test vérifie si le déplacement fonctionne correctement. Les voici :

Nom test	Classe	Méthode testée	Description	Condition réussite
moveTest	Alien	Move	Déplace l'Alien	L'Alien se déplace à droite
moveTestExeption	Alien	Move	Déplace l'Alien en dehors de l'espace de jeu	L'Alien se déplace en bas à gauche) une colonne en dessous
GoRightTest	Player	GoRight	Déplace le Player à droite	Le Player ce déplace à droite
GoRightWith MaxWidthTest	Player	GoRight	Déplace le Player à droite lorsqu'il Est coller à la bordure	Le Player ne bouge pas



Preuve que les tests réussissent (Visual Studio)

```
C:\Users\lucas\OneDrive\Documents\GitHub\SpaceInvader\ModelTests>dotnet test ModelTests.csproj
  Identification des projets à restaurer...
  Aucune action. Aucun des projets spécifiés ne contient des packages à restaurer.
```

Message d'erreur lors de l'exécution des tests dans la console

Ci-dessous on peut voir que les tests fonctionnent dans Visual Studio mais pas dans la console (il ne se lance pas). N'ayant pas mis en place les GitHub action je ne peut pas fournir d'autres preuve de la réussite de mes tests.

Conclusion

Ce projet m'a offert une opportunité précieuse d'approfondir mes compétences en programmation orientée objet et notamment en WPF.

J'estime avoir satisfait toutes les exigences du cahier des charges, mais je reconnais qu'il subsiste des possibilités d'amélioration, notamment en ce qui concerne la séparation MVC. Par exemple les trois liste « Pair » qui contienne un modèle logique et un modèle graphique, je devrais aussi séparer les méthodes « draw » de mes updates et les implémenter dans la méthode UpdateScreen.

De plus, il reste à implémenter des fonctionnalités telles que le menu des options, le menu des règles, le shop et le jeu multijoueur bien qu'il ne soit pas obligatoire.

Le recours à IceScrum a été complexe au début, principalement parce que nous n'avons pas immédiatement eu l'information que nous l'utiliserons. La gestion par un unique Scrum Master pour l'ensemble des élèves a parfois posé des problèmes car on devait attendre la validation des story, car nous ne pouvions techniquement pas commencer tant que l'User Story n'était pas validée. De plus durant l'ensemble du projet j'ai repoussé mon utilisation d'iceSrum car je le voyais surtout coder et apprendre de nouvelle chose.

A la fin du projet après une discussion avec le chef de projet j'ai compris que l'utilisation d'iceScrum dans ce projet servait à nous familiariser avec cette méthode de gestion de projet. Et que pour notre future vie professionnelle il sera essentiel de savoir travailler avec ce genre de logiciel/gestion de projet.

Dans le cas ou je devrais recommencer ce projet je ne changerais pas grand-chose de ma méthodologie de travaille. La chose la plus importante que je dois changer et mon utilisation de iceScrum qui fait tache dans mon projet. J'installerais aussi le fichier gitIgnore des le début du projet car ayant des images dans le bin le gitIgnore ne fonctionnait pas.

Utilisation de l'IA

Interface utilisateur

Pour la partie UX j'ai utilisé l'IA chatGPT durant la création des Persona. J'ai aussi utilisé le site SpaceColor (générateur de palette de couleur) qui utilise peut-être une IA.

Base de données

Pour la partie base de données je n'ai pas utiliser d'IA.

POO

Dans la partie POO j'ai principalement utiliser l'IA ChatGPT afin de commenter mon code.

Conclusion Final

Ce projet était probablement mon préférer depuis le début de notre formation car l'on a eu une liberté plus grande que d'habitude (choix WPF ou console, totalement libre dans les choix graphique figma). La faite d'appliquer les requêtes dans notre code permet de rendre notre travail plus concret d'acquérir plein de nouvelle connaissance. Si je devais retenir un élément négatif c'est clairement Ice Scrum qui à ralentit totalement l'avancée du projet.