SAE 1.01-1.02

Jeu vidéo WPF

FRAPPIN Mathis, VILLARET Enzo
INFO 1 TD1 TP12

Table des matières

1. Présentation

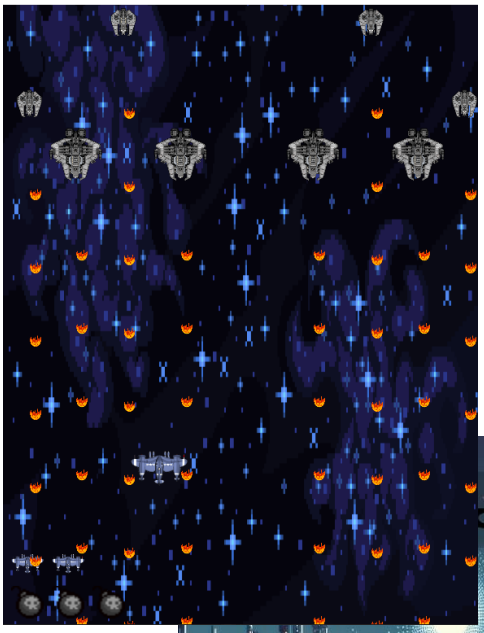
1.1. Description générale

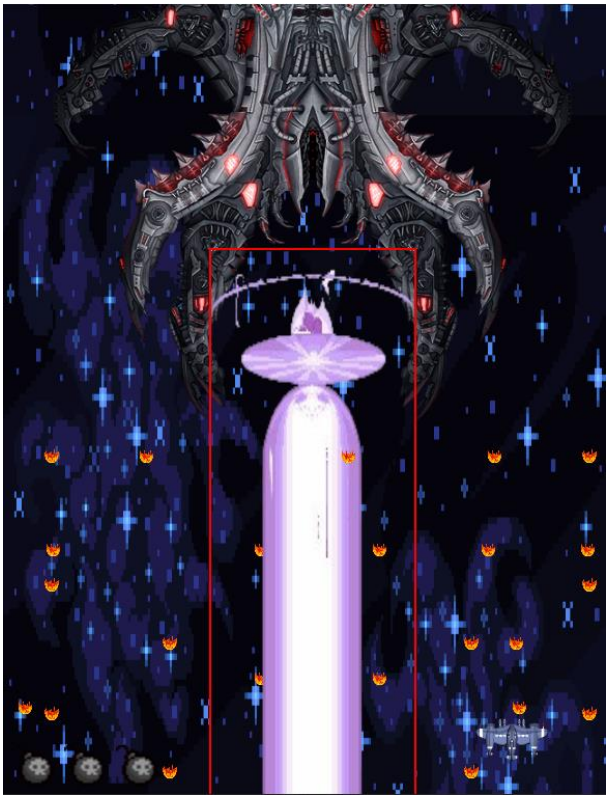
"Galactic Shooter vous plonge dans un univers futuriste captivant, où vous, en tant que pilote intrépide, êtes confronté à des vagues incessantes d'envahisseurs de l'espace. Le défi ultime réside dans votre capacité à esquiver habilement les innombrables tirs ennemis qui convergent vers vous.

Dans ce jeu de type "shoot them up" et "bullet hell" (Danmaku), la rapidité de vos réflexes et la précision de vos mouvements sont cruciales. Les ennemis, variés et redoutables, viennent de tous les horizons cosmiques, chacun apportant son propre style de menace. Certains adversaires déchaînent des rafales de projectiles rapides, créant un écran de balles défiant toute logique, tandis que d'autres préfèrent des tirs plus lents mais dévastateurs, exigeant une anticipation minutieuse.

Comme on peut le voir sûr les images ci-dessous, il sera très facile pour le joueur de se retrouver submerger par les attaques de l'ennemi

La conception visuelle époustouflante de Galactic Shooter accompagnera l'action frénétique. Que ce soit des nébuleuses chatoyantes aux vaisseaux ennemis détaillés, chaque élément graphique contribuera à immerger les joueurs dans ce monde futuriste et hostile.





Lors de sa partie, le joueur sera amené à devoir survivre et à éliminer les ennemis qu'il rencontrera pendant 12 vagues consécutive avec un boss toutes les 4 vagues (chaque boss étant à chaque fois plus difficile que le précédent)
Avant de commencer son périple, le joueur pourra choisir entre quatre difficultés nommées « Facile », « Normal », « Difficile » et « Lunatic » dans l'ordre de difficulté croissante. Ainsi, le joueur pourra soit trouver une difficulté qui lui correspond ou alors se mettre au défi et repousser ses limites.

Originellement, le joueur peut effectuer plusieurs actions durant sa partie :

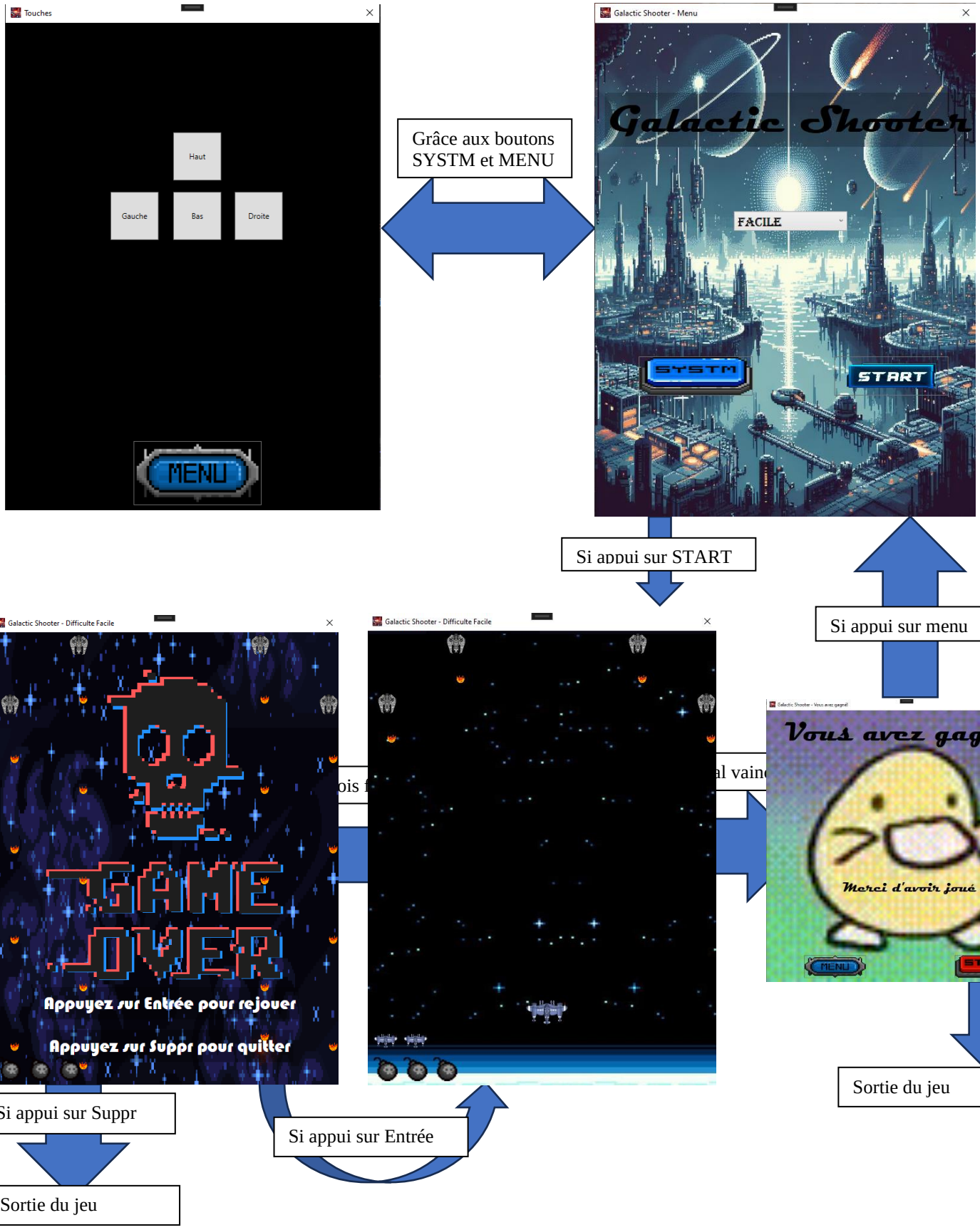
- Se déplacer (avec les **touches directionnelles** à moins qu'il souhaite les changer)
- Lancer des bombes (avec la touche **B**) qui nettoierons l'écran à moins que le joueur soit face à un boss. Dans ce cas, la bombe lui enlèvera une partie de sa vie.
- Utiliser des codes de triches pour se téléporter jusqu'au différents boss (à l'aide des touches **F1**, **F2** et **F3**)
- Utiliser un code de triche pour obtenir un tir surpuissant qui détruit presque tous les ennemis en un seul coup (à l'aide de la touche **F4**)

De plus, lorsque le joueur se fera toucher par un ennemi, celui-ci bénéficiera de frames d'invincibilité afin de pouvoir se déplacer et se mettre en sûreté.

De plus, le joueur pourra mettre le jeu en pause (en appuyant sur la touche **P**). Il pourra ensuite enlever cet état de pause (en appuyant sur **Echap**).
Mais aussi, à la manière des vieilles consoles de jeu, il pourra réinitialiser sa partie si celui-ci était amener à se retrouver en mauvaise posture (en appuyant sur **Entrée**).
Il pourra aussi bien évidemment quitter le jeu à tous moments (en appuyant sur **Suppr**).

Si le joueur parvient à vaincre les 12 vagues, celui-ci sera récompensé sur l'écran de fin de partie.

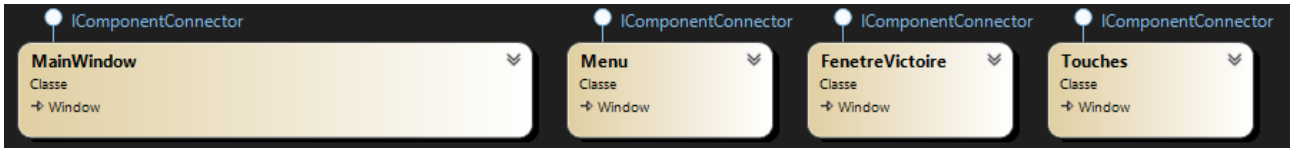
1.3. Cinématique des écrans



2. Conception – Diagramme de classe

2.1. Présentation générale

Pour façonner ce jeu, nous avons créés les quatre classes ci-dessous.



- La classe MainWindow contient l'essentiel du code. Elle contient tout ce qui touche à la création et à la gestion des ennemis ainsi que du joueur. Cette classe permet aussi la génération des sons dans le jeu ainsi que de la création de l'environnement visuel du jeu
- La classe Menu permet de faire apparaitre la fenêtre de Menu dans lequel le joueur pourra sélectionner la difficulté dans laquelle il souhaitera jouer.
- La classe FenetreVictoire permet de faire apparaitre la fenêtre de fin de partie lorsque le joueur bat le boss final du jeu. Cette fenêtre lui permettra soit de quitter le jeu soit de retourner au menu pour rejouer en changeant ou non de difficulté.
- La classe Touches permet au joueur depuis la fenêtre de menu, d'accéder à un sous menu lui permettant de réallouer ses touches pour jouer.

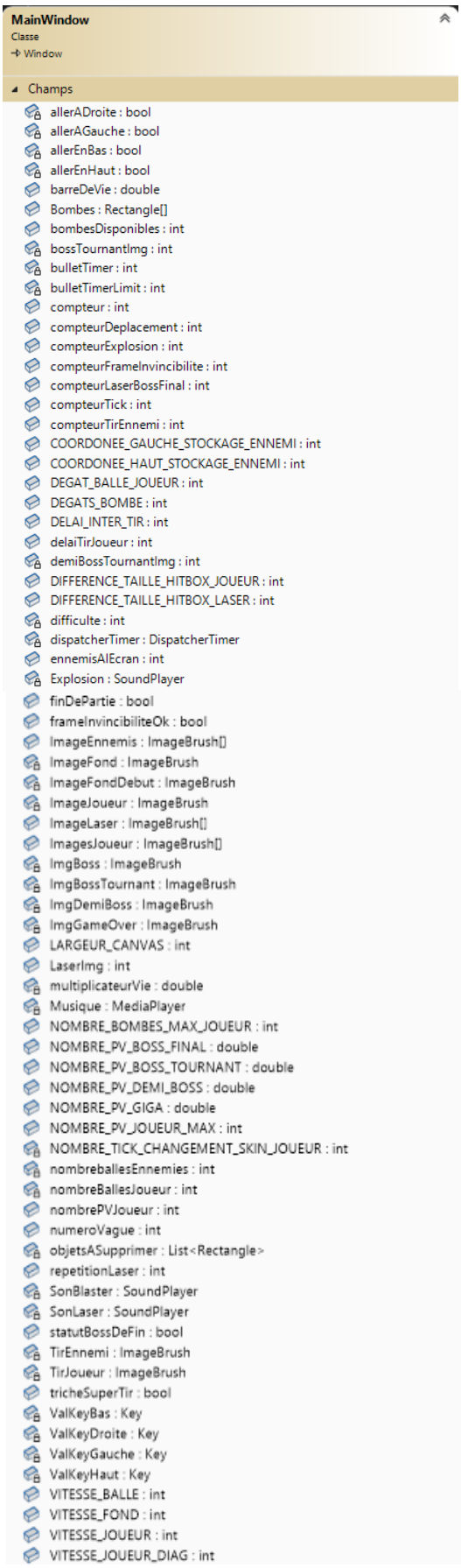
Dans le cas de la factorisation de la mainWindow, l'ajout de classes secondaires est compliqué étant donné que la plupart des actions faites dans cette classe sont destinées à interagir de manière directe ou indirecte avec des rectangle de la fenêtre de jeu (fenêtre ne semblant pas être accessible à partir d'une classe secondaire). C'est pourquoi nous n'avons donc pas pu créer d'autres classes que celles présentes à l'origine lors de la création de nouvelles fenêtres. Cependant, malgré ce problème, ces classes communiquent entre elles notamment avec des échanges Menu/MainWindow et Touches/MainWindow lors de la saisie de la difficulté et lorsque les touches sont réallouées.

2.2. Présentation détaillée MainWindow

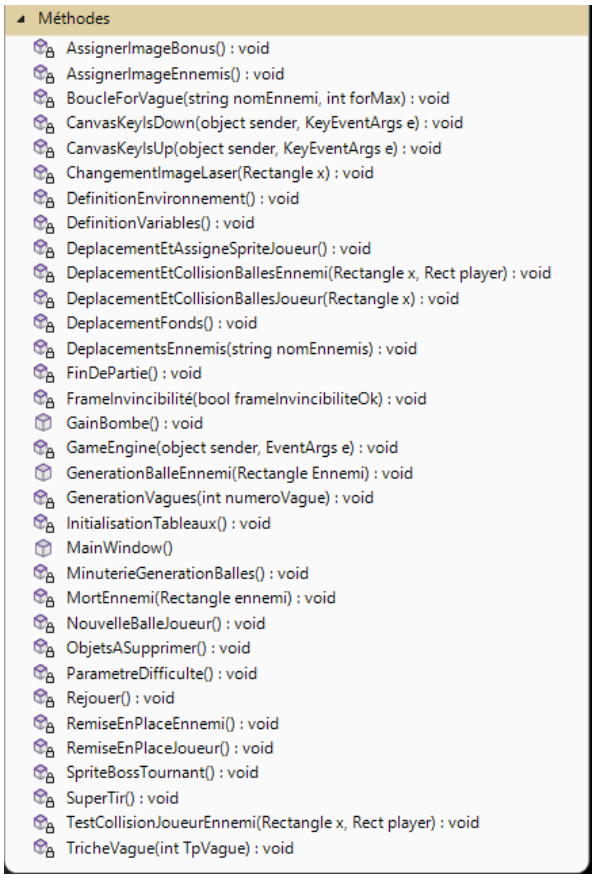
Comme on peut l'observer sur la capture d'écran à droite, la classe MainWindow est une classe massive par sa taille étant donnée que celle-ci contient l'ensemlbe des constantes et des valeurs utilisées par l'ensemble du code. Qu'il s'agisse de compteur ou bien même de champs qui servent à décompter les points de vie du joueur ou des ennemis. Comme par exemple barreDeVie de type int, nombrePVJoueur de type int mais aussi les constantes comme NOMBRE_PV_BOSS_FINAL de type int une fois de plus, qui permet de changer la barre de vie pour qu'elle corresponde à celle du boss de fin.

Parmis les conteurs, on peut relever compteurDeplacement qui est le compteur qui permet la transition dynamique entre les différents sprite du joueur et compteurLaserBossFinal qui permet de prévenir le joueur avant que le laser du boss final soit généré.

De plus, la MainWindow contient des booléen nécessaires au bon fonctionnement du jeu comme les quatre booléen de déplacement AllerAGauche/droite/Haut/Bas, qui permettent une bonne gestion du déplacement du joueur qui soit réactif à l'appui sur les touches. On peut aussi noter le booléen frameInvincibilitéOk qui permet la gestion du temps de répit que possède le joueur entre chaque tir qu'il recevra.



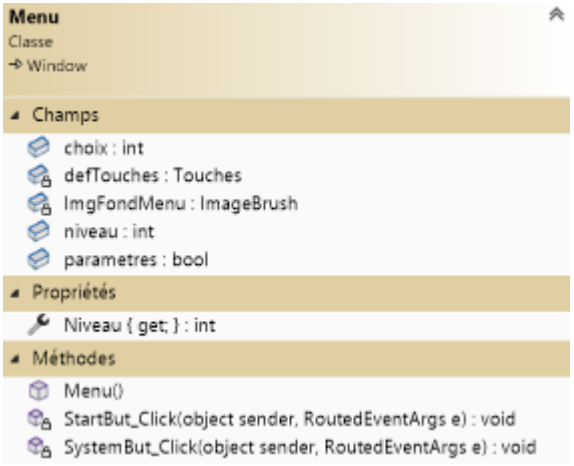
De plus, la classe MainWindow contient toutes les méthodes nécessaires au fonctionnement du jeu donc en soit à la gestion des rectangles du joueur et des ennemis. On retrouve par exemple les méthodes DeplacementEnnemis ou DeplacementEtAssigneSpriteJoueur.



De plus ces méthodes peuvent gérer différents évènements qui se dérouleront au cours du jeu comme le gain de bombes par le joueur à chaque fois qu'un boss est vaincu où bien simplement la gestion de balles à têtes chercheuses qui seront là exprès pour rendre votre expérience de jeu plus stressante.

Enfin, la classe MainWindow permet également de s'occuper d'une bonne partie des événements qui se passent en arrière-plan. Elle gère donc la remise en place des ennemis en dehors des espaces de jeu, la possibilité de SpeedRun le jeu à l'aide de certaines fonctions cachées, etc…
Et pour finir elle permet de gérer l'apparition de la fenêtre de fin de jeu en cas de victoire ou simplement l'affichage de votre mort.

2.3. Présentation détaillée de la classe Menu



Comme on peut également le voir, à l'aide de la figure ici présente, la classe Menu est une classe bien moins imposante que la MainWindow mais elle n'est pas inutile pour autant.

Elle permet, entre autre, d'accéder à la fenêtre « System », grace aux bouton system et au champ defTouches qui n'est autre que l'affichage des touches assignées aux différentes commandes du jeu, mais aussi de choisir le niveau de difficulté qui vous convient.

Les champs comme niveau et parametres vous laissent l'opportunité de pouvoir choisir un niveau de difficulté adapté à vos compétences en matières de, réactivité, d'anticipation mais également votre aptitude à ne pas paniquer, permet d'améliorer considérablement l'expérience de chacun. La fenetre menu vous permet donc de personnaliser votre expérience de jeu en fonction de votre envie, si vous souhaitez vivre une petite promenade de santé comme surmonter un véritable défi.

3. Partie Algorithmie

3.1. Déplacement du joueur

Nous avons décidé de dynamiser le mouvement du joueur en le mettant sur deux « trames ». En effet, en jeu, lorsqu'on se déplace dans la même direction, le Sprite du joueur est mené à changer. Cet effet est le résultat de l'algorithme suivant :
Pour une direction donnée.

```
Deplacement à gauche = False
Deplacement à droite = False
Deplacement en haut = True
Deplacement en bas = False
valeur compteur tick pour image joueur = 1
```

Figure 1: exemple déplacement de rang 1

Compteur initialisé à 0

Si touche appuyée
    Déplacement du joueur dans une direction autorisée
    Compteur incrémenté de 0.2 seconde
Fin Si

Si touche relâchée
    Déplacement du joueur dans une direction non autorisé
    Compteur réinitialisé à 0
Fin Si

```
Deplacement à gauche = False
Deplacement à droite = False
Deplacement en haut = True
Deplacement en bas = False
valeur compteur tick pour image joueur = 14
```

Figure 2: Déplacement de rang 2

Si déplacement autorisé dans aucune direction
    Assigne au joueur une image immobile
Fin Si
Si compteur est plus petit qu'une seconde
    Si déplacement autorisé et que le joueur est dans la fenêtre
        Déplacement du joueur dans la direction donnée
        Assigne au joueur l'image correspondant à son déplacement (de rang 1)
    Fin Si
Sinon
    Déplacement du joueur dans la direction donnée
    Assigne au joueur l'image correspondant à son déplacement (de rang 2)
Fin Si

Le but de cet algorithme qui est composé de trois méthodes différentes est de changer le Sprite du joueur en lui attribuant une inclinaison plus ou moins élevé (rang 1 = peu incliné, rang 2 = très incliné) Ainsi, si le joueur garde la même trajectoire pendant plus d'une seconde, son image change de manière « dynamique ».

Les trois méthodes utilisées dans cet algorithme sont disponibles dans la classe MainWindow :

- CanvasKeyIsDown (lignes 299 à 378)
- CanvasKeyIsUp (lignes 379 à 402)
- DeplacementEtAssignerSpriteJoueur (lignes 445 à 536)

A la vue de cet algorithme, on peut observer qu'il génère 5 tests à chaque tick du GameEngine. Ainsi, étant donné qu'un tick dure 20ms, pour une direction donnée, le programme effectue 250 tests par seconde. Donc 1 000 tests par seconde pour gérer les 4 directions.

3.2. Création du laser du Boss final

Pour le dernier boss du jeu, nous avons décidés de lui créer une attaque spéciale unique, un laser capable de tuer en un coup le joueur. Cependant, cette attaque peut être inévitable pour le joueur. C'est pour cela que nous avons décidé de lui créer un compte à rebours.

Compteur général initialisé à 0 et qui augment à chaque appel du GameEngine.
Compteur création laser = 7

Méthode créer balles ennemies toutes les x secondes
    Appelle création balles boss final

Si boss final à l'écran
    Créer balles boss final
    Compteur création laser décrémenté de 1
    Si compteur création laser = 0
        Créée un rectangle de tag « pré laser » et qui est un rectangle vide aux bords rouge
        Remet le compteur général à 0

Pour chaque rectangle Y dans le Canvas
    Méthode déplacement et collision des balles ennemis
        Si compteur général est plus grand ou égal à deux secondes et tag de Y = « prélaser »
            Tag « prélaser » devient « laser »
        Si Y est une balle « normale » du ou de tag « laser »
            Donne une collision a Y
            Déplace les balles « normale »
            Si le tag de Y est « laser »
                Appel métohde animation du laser
            Fin Si
            Si Y rentre en collision avec le joueur
                Si le tag de Y n'est pas « laser »
                    Supprime Y et enlève une vie au joueur
                Sinon
                    Enlève tous ses points de vie au joueur et affiche le Game Over
                Fin Si
            Fin Si
        Fin Si
Fin pour chaque

Une fois que l'animation du laser est finie le compteur création laser revient à 7

Dans cet algorithme, l'utilisation de compteurs et du changement de tag nous assure que le laser ne soit pas tiré à chaque image et que le joueur soit informé de son arrivée. Ainsi, le laser est une attaque qui apparait toutes les 8 attaques normale et qui au lieu de se déplacer, fait appel à une méthode de génération d'animation. Lest méthodes utilisées dans l'algorithme sont :

- MinuterieGenerationBalle (lignes 746 à 775)
- GénérationBalleEnnemi (lignes 777 à 883)
- DeplacementEtCollisionBallesEnnemi (lignes 673 à 744)

3.3. Mouvement des tirs ennemis et détection des collisions avec le joueur

Attardons-nous un peu plus sur la méthode de déplacement et de collision entre les tirs ennemis et le joueur car, l'entièreté du principe du jeu se base sur cette méthode. Ainsi, nous avons fait une méthode nous permettant de donner des déplacements variés aux tirs ennemis, tout en nous assurant de leur bonne collision. Ainsi, pour la variété des tirs ennemis, nous avons créés 7 tirs différent

Chaque 20 millisecondes : création de la collision du joueur (plus petite que le rectangle joueur pour donner de la liberté de déplacement)

Pour chaque rectangle Y dans le canvas
- Si compteur général est plus grand ou égal à deux secondes et tag de Y = 7
  - Tag 7 devient 7bis
- Fin Si
- Si le tag de Y est 1 ou 2 ou 3 ou 4 ou 5 ou 6 ou 7bis (7bis représente le laser)
  - Descend Y a une vitesse donnée
  - Si tag Y est le tag 1
    - Y se déplace horizontalement selon le comportement des balles de tag 1
  - Sinon si tag Y est le tag 2
    - Y se déplace horizontalement selon le comportement des balles de tag 2
  - …
  - Sinon (lorsque tag Y est le tag 7bis)
    - Appel Méthode changement image laser
- Fin Si
- Si Y est hors du canvas (il y a deux tests pour tester l'axe x et l'axe y)
  - Supprime Y
- Fin Si

- Donne une collision a Y
- Si le tag de y n'es pas le laser
  - La hitbox du tir est égale à celle du rectangle représentant le tir
- Sinon
  - La hitbox est plus petite pour convenir à l'image du laser

- Si Y rentre en collision avec le joueur
  - Si le tag de Y n'est pas « laser » ou que le joueur n'a pas 0 PV
    - Supprime Y et enlève une vie au joueur
    - Cache l'image correspondant à la vie perdue du joueur
    - Donne des frames d'invincibilité au joueur
  - Sinon
    - Enlève tous ses points de vie au joueur et affiche le game Over
  - Fin Si
- Fin Si
- Fin Si

Fin pour chaque

Cette méthode est disponible dans MainWindow.xaml.cs : DeplacementEtCollisionBallesEnnemi (lignes 673 à 744)

Durant les phases de test, nous avons pu compter un maximum de 34 balles générées au maximum à l'écran lors de la vague 9 (ci-dessous). De plus, cette méthode nécessite au moins 14 tests pour chaque tir, ainsi à raison de 34 tirs max à gérer en simultané, on peut estimer à 476 tests toutes les 20 ms, soit un total de 23 800 tests par seconde. Bien que ce nombre de test soit important, on peut tout de même le considérer comm un « sacrifice nécessaire » afin de pouvoir donner un jeu varié au joueur.


```
Nombre de balles ennemies à l'écran = 34
```

Bien qu'il n'y ait pas exactement 34 balles à l'écran sur la capture d'écran à gauche, ces balles « non visibles » doivent être comptabilisées car elles sont à la limite du canvas donc bien que leur sprite ne soit pas visible, elles sont tout de même présentes.

4. Conception graphique

Les images et sons que nous avons utilisés provenaient de sites proposant des png et des sons libres de droit comme PngEgg ou DeviantArt. Pour les sons, nous avons utilisés un site de SoundBoard SoundBoardGuy et pour la musique de fond, celle-ci provient aussi d'un site proposant des .mp3 libres de droits

La quasi-totalité des image trouvées provenaient d'ensemble d'images (exemple à droite).

Ainsi, nous avons dû découper à la main chaque image que nous souhaitions utiliser.

De plus pour les animations du laser et de l'image de victoire. Nous avons dû utiliser des logiciels pour transformer les gifs en image puis plus spécifiquement encore pour le laser, le détourer pour enlever le fond qui était sur l'image originale.

Pour la musique, quant-à-elle, nous avons dû par moment la retoucher grâce à des logiciels comme Audacity pour enlever des parties parasites. De plus, nous avons essayé de dupliquer la musique de fond pour la faire boucler étant donné que la classe MediaPlayer ne dispose pas de système de Loop intégré.

Mais encore, pour la musique, nous avons dû faire attention au niveau de l'algorithme à ce que les sons ne se jouent qu'une fois uniquement (afin d'éviter que les sons chargent mal ou qu'ils ne commencent chaque fois que le GameEngine ne fasse un tour).

5. Cahier de recettes

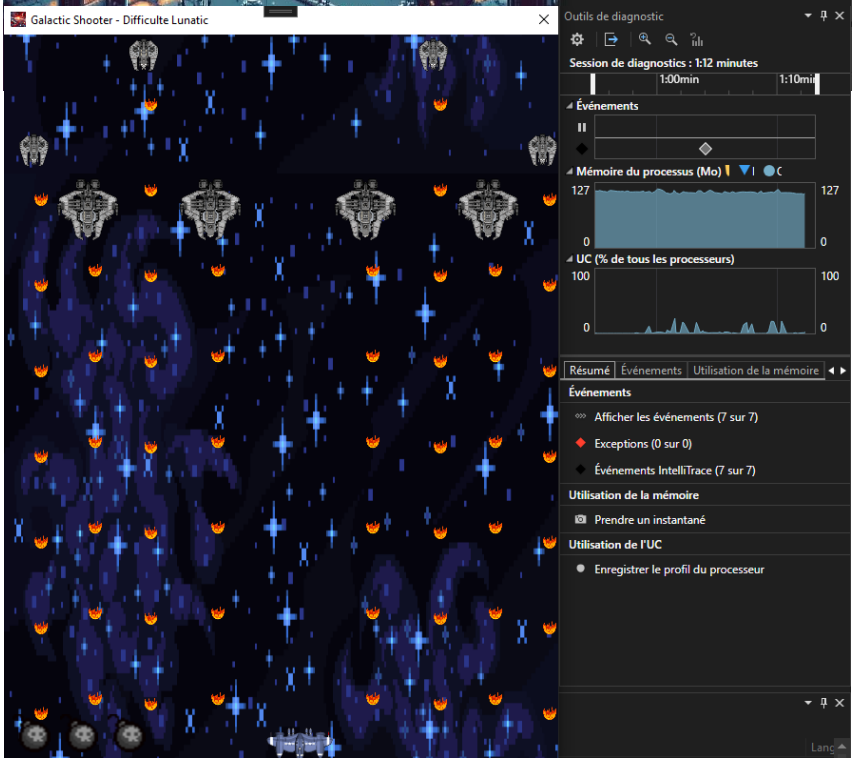
5.1 Tests de validation

Nom	Fonctionnalités/ Dialogues/Classes	Etat
Mathis	Fenêtre sélection difficulté	OK
Mathis	Déplacement des fonds	OK
Enzo	Déplacements et tirs ennemis	OK
Enzo	Frame d'invincibilité	OK
Enzo	Système de bombes	OK
Mathis	Code de triche	OK
Enzo	Apparition vagues	OK
E & M	Fenêtre Victoire	OK
Enzo	Déplacement joueur	OK
Enzo	Changement sprite dynamique joueur	OK
Mathis	Sprite tournant ennemis	OK
Enzo	Interface graphique joueur	OK
Mathis	Système de réallocation des touches	Non achevé
Mathis	Dialogue entre les fenêtres	OK
Mathis	Gestion tir joueur	OK
Enzo	Tests collision	OK
Mathis	Système de Pause et Rejouer + animation GameOver	OK
Enzo	Mode Debug	OK
E&M	Ajouts de sons et musiques de fond	OK

5.2 Tests de performance

Pour effectuer les tests de performance, nous avons décidés de prendre comme valeur étalon, l'écran de début de jeu (le menu) étant donné que c'est l'écran qui a été le moins gourmand en ressources durant le déroulement du jeu. Ainsi, en temps normal, on peut considérer que le processus requiert 87 Mo de mémoire et qu'il utilise moins de 1% du processeur.

Sur la capture d'écran à droite, on peut observer le moment où il y a le plus de balles générées à l'écran. On peut donc supposer qu'étant donné qu'il y a le plus de rectangle à l'écran, donc le plus de tests de collisions effectués, que c'est à ce moment précis que la mémoire consommée est la plus élevée. Ainsi, on obtient à ce moment précis, une consommation de 127 Mo de mémoire vive. De plus on observe des pics de consommation du processeur qui peuvent atteindre dans les environs de 15%.



Finalement, on peut observer qu'une fois l'écran de fin chargé, il y a un pic de consommation à 124 Mo puis que la consommation descend et se stabilise à 97 Mo, cela s'explique par le fait que bien que tous les rectangles aient été supprimés, la consommation ne revient pas à 87 Mo car il faut charger l'animation de fin de jeu. La consommation du processeur, quant à elle, revient dans les environs de 1%