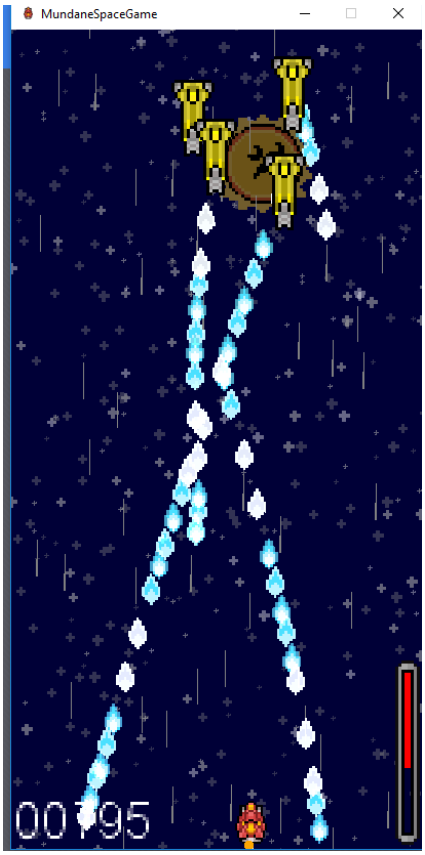
Le joueur doit survive à tout ce qui est sur son chemin. Il est équipée d'un blaster ayant la capacité d'évolué en ramassant les boites bleu. Il utilise les touches WASD et Espace pour se mouvoir et attaquer.



Il y a plusieurs type d'énemie et le challenge c'est d'évité toute les munitions pour ne pas perdre de la vie – qui est représenté par la bar rouge.

Si le joueur est à zero de vie, c'est game over et le jeu se ferme. Il doit alors recommencer.

Cependant, le score est sauvegardé et la position de la mort sera indiqué à l'écran quand le joueur passera.

C'est 5 par seconde de jeu jouer + l'hp des enemies tuer. Donc 50 pts pour un enemie avec 30 de vie.

Exigence – Prog:

1 enum avec 3 éléments : Atteint – Il y a un enum qui dirige quel type d'AI un acteur va utiliser- qui peut être changer dans le déroulement d'une partie. C'est plus souple à l'instar de créer des classes spécifiques pour chaque villains.

Itérations/Collections : Atteint - Tout les objects du jeu excepter quelque exception comme le joueur se retrouve dans une listes – De cet manière , il est possible d'ajouter des éléments en runtime simplement en itérant la liste d'objets et utiliser un Foreach/For pour appeler spiretedraw. Ajouter à cet liste et l'objet est visible en screen .

Tableau/Collisions /Discussion Algorithm: Atteint – Chaque texture est composer de pixel. De ce fait, si on les rentres dans un tableau multidimensionelle, il est possible de point un pixel spécifique. Avec ceci en tête, les collisions se font pixel par pixel : Car il suffit d'observer l'addition de deux alpha de certain pixel entre deux object qui on leur rectangle intercepter pour verifier si il y a une collision. Si l'alpha equivaut à zero – ce qui veut dire transparent, Il n'y a donc aucun pixel donc aucune colision. S'il y a un pixel, alors relativement au deux pixel, il y aura une force appliquées. S'il y a plusieurs pixel, il y aura donc une simulation de force puisque c'est additive, et donc , vers la bonne direction.

J'ai utiliser cet algorithme car je trouvrais que les rectangle n'était pas assez précis – ce qui est une éxigence minimal pour les Bullets hell. Auparavant, il était possible d'entrer en collision entre les vaisseaux pour changer leur trajectoire, mais avec une multitude de vaisseau et de balles sur l'écran, Il fallait que j'appelle à plusieurs reprise le GC due à une horrible performance. Donc, présentement, la collision est superbe, mais de il ne simule plus la physique – donnant à l'instar des dommages au contact.

Il n'y a pas de masse, donc les forces appliqués sont équivalentes entre elles.

Méthodes : Toute Atteint – Il y a plusieurs méthode dont qu'ils ont un paramètre optionnel qui sont nommées. Un exemple serait La Classe Projectile qui utilise 5 paramètre dont 3 optionnel qui sert de constructeur. Cela assure qu'il n'y a pas de situation aberrante est simple d'utilisation. C'est méthodes est d'autant plus Overloader, Car une autre permet de faire une copie d'une autre munitions en prenant comme paramètre Projectile – Facile d'utilisation et permets d'instancier de nouveau projectile vite . Dans un jeu de cet genre (Touhou/Bullet Hell), C'est important – surtout s'il y a plus de 200 projectile dans l'écran.

Données sauvegardée: %appdata%\Omicron\highscore.txt. Contient le score le plus récents et le temps records. (Le plus haut le mieux – c'est de la survie). Dans le jeu, cela change la position d'une acteur inerte nommées Lastscore – Elle sert d'indicatif visuelle pour informée le joueur qu'il a dépassé son record.

Héritage de classe : Atteint - La classe Image est père de toute les classes . Acteur est dérivées de Image – Ce qui à comme enfant Projectile, Player, Bonus . Acteur a plusieurs function qui est virtuel – Comme update – Car le joueur et un ennemie n'est pas la même chose. En polymorphisme, La function OnCollision() Recois comme paramètre Image – Car elle on la functionalité de collision. Cepedant, contrairement à Acteur, elle n'a pas de physique. Cela permet donc utiliser une seul fonction quelque soit la classe pour autant qu'elle derive initiallement de image. Et de la , la logique est simple – avec le as, certaine condition peut derouler de ceci.

Délégué : Atteint – La classe Acteur à pour déléguée OnTakeDommage et un OnTakeDamageHandler, comme évènement. À leur création, le programme principal leur assigne une fonction à OnTakeDamageHandler, ce qui permets – à leur mort – d'ajouter du pointage . Le déléguée se fait appeler dans OnTakeDamage, quand leur vie est <= 0.

Exigence – Artistique:
1 minutes de gameplay/Gametimes: Techniquement parlant, le jeu à une durée de vie infinie – Le Pattern des attaques n'est un algorithme créer pour soutenir la musique. Mais Officiellement, Si quelqu'un arrive à 203 seconde, le jeu s'arrête et le joueur gagne. Il y a plusieurs compteurs dans le jeu dont le rate à quel vitesse un acteur tirs ou à quel moment un enemy doit spawn – D'autant plus, le temps est interpolé pour être indépendant selon les plateformes.
Il y a aussi une caméra qui est interpolé par un gametime, mais puisque j'ai changer de direction artistique à une camera fixes, elle ne sert à rien.

3 Textures Différentes : 27 textures, dont 3 animations , 1 music et 1 font ;

Détection de Collision : @Tableau/Collsion

Difficulté Langage de programmation : Je n'avais jamais faire de constructeur de classe dérivée ayant un paramètre obligatoire - puisque pour la calculation de physique il faut un sprites. De ce fait, j'ai recherché en ligne et j'ai trouver que faire:base(T) ou T est une Texture2D dans le constructeur permets d'appeler le constructeur original .

Diagramme des Classes :