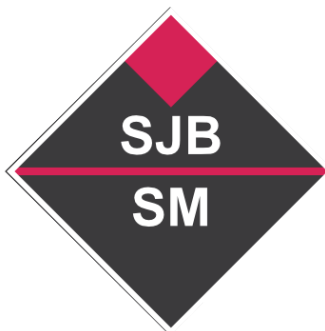


INSTITUT SAINT JEAN BERCHMANS – SAINTE MARIE

Section Transition Informatique



PROGRAMMATION PYVENTURE

Travail de fin d'études
réalisé par
Vanherf Julien

Année académique 2020 - 2021

Remerciements

Je tiens à remercier...

M. Carrera pour les quelques années passées dans sa classe, avec sa bonne humeur et sa manière de travailler.

M. Rodrigues pour l'apprentissage de HTML.

M. Tanier pour m'avoir appris à développer en #C pour faire de petits montages et les petites blagues de temps à autre.

M. Klich pour l'apprentissage des bases de données qui vont m'être utiles.

Et enfin tous les autres profs des cours généraux qui m'ont accompagné pendant mes cinq ans à l'école.

Introduction	11
Présentation	11
Élément	12
Interface utilisateur	16
Analyse	26
Arborescence	26
Fichier de configuration	30
config.ini	30
SOUND	30
Attributs	30
CHARACTERS	30
Attributs	30
GAME	30
Attributs	30
Classes	31
player.py	31
Player	31
Attributs	31
Méthodes	32
wizard.py	34
Wizard	34
Attributs surchargés	34
fairy.py	34
Fairy	34
Attributs surchargés	34
mob.py	35
Mob	35
Attributs	35

Méthodes	36
orc_brawler.py	37
Orc_Brawler	37
Attributs surchargés	37
orc_chaman.py	37
Orc_Chaman	37
Attributs	37
Attributs surchargés	37
orc_fighter.py	37
Orc_Figther	37
Attributs surchargés	37
orc_nain.py	38
Orc_Nain	38
Attributs surchargés	38
Blue_Zombie	38
Attributs	38
Attributs surchargés	38
zombie.py	39
Zombie	39
Attributs surchargés	39
skelet.py	39
skelet	39
Attributs	39
Attributs surchargés	39
tiny_zombie.py	40
Tiny_Zombie	40
Attributs surchargés	40
knight.py	40
Knight	40

Attributs surchargés	40
knight_brute.py	40
Knight_Brute	40
Attributs surchargés	40
bow_man.py	41
Bow_Man	41
Attributs	41
Attributs surchargés	41
bad_chest.py	41
Bad_Chest	41
Attributs surchargés	41
boss.py	42
Boss	42
Attributs	42
Méthodes	43
orc_boss.py	45
Orc_Boss	45
Attributs surchargés	45
undead_boss.py	46
Undead_Boss	46
Attributs	46
Attributs surchargés	46
knight_boss.py	47
Knight_Boss	47
Attributs	47
Attributs surchargés	47
chest.py	48
Chest	48
Attributs	48

Méthodes	48
coin.py	49
Coin	49
Attributs	49
Méthodes	49
heart_container.py	50
Heart_container	50
Attributs	50
Méthodes	50
heart.py	50
Heart	50
Attributs	50
Méthodes	51
spell_book.py	52
Spell_Book	52
Attributs	52
Méthodes	52
attack_book.py	53
Attack_Book	53
Attributs surchargés	53
Méthode surchargé	53
cooldown_book.py	53
Attack_Book	53
Attributs surchargés	53
Méthode surchargé	53
projectile_speed_book.py	53
Projectile_Speed_Book	53
Attributs surchargés	53
Méthode surchargé	53

speed_book.py	54
Speed_Book	54
Attributs surchargés	54
Méthode surchargé	54
effect.py	54
Death_Effect	54
Attributs	54
Méthodes	54
fire_ball.py	55
Fire_Ball	55
Attributs	55
Méthodes	55
ball.py	56
Ball	56
Attributs	56
Méthodes	56
brain.py	57
Brain	57
Attributs	57
Méthodes	57
arrow.py	58
Arrow	58
Attributs	58
Méthodes	58
game.py	59
Game	59
Attributs	59
Méthodes	60
tilemap.py	62

Map	62
Attributs	62
Camera	62
Attributs	62
Méthodes	62
environnement.py	63
Wall	63
Attributs	63
Floor	63
Attributs	63
Door	64
Attributs	64
Méthodes	64
Ladder	65
Attributs	65
Méthodes	65
Rock	66
Attributs	66
Spike	66
Attributs	66
Méthodes	66
Wall_Hole	67
Attributs	67
Méthodes	67
Win_Ladder	68
Attributs	68
Méthodes	68
menu.py	69
Menu	69

Attributs	69
Méthodes	69
MainMenu	70
Méthodes	70
OptionsMenu	71
Méthodes	71
CreditsMenu	71
Méthodes	71
ControlMenu	72
Méthodes	72
Quit	72
Méthodes	72
Death_Menu	72
Méthodes	72
Pause_Menu	73
Méthodes	73
Confirm_Quit	73
Méthodes	73
Sound_Menu	74
Méthodes	74
Win_Menu	74
Méthodes	74
Select_Player_Menu	75
Méthodes	75
Problèmes rencontrés	76
Conclusion	77
Bibliographie	78
Ressources Web	78
Code	78

Style du jeu	79
Trello	79

Introduction

Mon application sera un jeu du style rogue-like. Le principe est de parcourir les salles dans un donjon. Ce dernier sera composé de trois différents étages générés aléatoirement. À chaque fin d'étage vous rencontrerez un boss. Le but du jeu est de battre les 3 boss dans ce donjon ce qui vous permettra de débloquent un nouveau personnage jouable !

Présentation

Au début du jeu, vous incarnez un Mage mais, vous pourrez débloquent la Fée après plusieurs parties.

Le donjon sera composé de 3 étages générés aléatoirement. Le premier comporte 10 salles, le deuxième, lui, en aura entre 10 et 15 et le 3e, quant à lui, en aura entre 15 et 20. Dans chacun de ces étages il y aura une salle de départ, une salle de boss, une salle comprenant un magasin et les salles remplies d'ennemis à combattre.

Les salles seront aussi remplies de différents obstacles comme des pierres, des piques, mais aussi des trous dans les murs tirant des flèches.

De temps en temps, il sera possible aussi d'y trouver des coffres contenant un peu d'or.

Plus on descend dans le donjon, plus les adversaires sont forts, ce qui fait qu'à chaque étage, on pourra trouver des arcanes divers et variés pour permettre de finir ce donjon.

Quand vous aurez battus 5 fois les 3 boss, vous débloquentez la Fée qui est un personnage plus rapide et qui vole.

Le jeu possède un menu debug, celui-ci permet de mettre les pièces du joueur à 99 et d'abattre tous les monstres.

Élément

- Personnages

Les statistiques des personnages restent fixes. Le seul moyen d'améliorer les statistiques est de récupérer des objets durant la partie ou d'activer les différents artefacts.

Les **pv (Points de vie)** lorsqu'ils atteignent 0, le personnage est déclaré mort.

Les **dégâts de base** sont les dégâts infligés par le personnage lorsqu'il n'utilise aucune arme.

La **vitesse d'attaque** est le temps d'attente entre deux attaques.

La **vitesse de déplacement** est la vitesse à laquelle le personnage se déplace.

- Personnages joueur

- Le **mage** attaque à distance. Il utilise un sceptre qui tire des boules de feu.
- La **fée** attaque aussi à distance. Plus rapide que le mage, elle vole, mais fait moins de dégâts et commence avec moins de points de vie.

- Personnages Monstre

Les statistiques des monstres ne varient que pour la vitesse de déplacement. Tous les monstres retirent un **pv (point de vie)**.

Les **pv** des monstres différents selon leurs classes

- Orc
 - Le **Combattant** attaque au corps à corps. Il est plus puissant qu'un être humain. Sa vitesse de déplacement est lente.
 - L'**orc de base** attaque aussi au corps à corps. Vitesse de déplacement supérieure au **Combattant**.
 - Le **Chaman** attaque à distance avec des boules d'énergie. Sa vitesse d'attaque est aussi lente que celle du combattant.
 - Le **Nain** attaque au corps à corps. Il est beaucoup plus rapide que l'orc de base, mais a très peu de points de vie.

- Mort vivant
 - Le **Zombie** attaque au corps à corps. Sa vitesse de déplacement est supérieure à celle du zombie de glace.
 - Le **Bébé zombie** attaque au corps à corps. Il est aussi rapide que l'Orc Nain mais à moins de pv.
 - Le **Zombie de glace** attaque à distance avec des morceaux de cerveau. Sa vitesse de déplacement est inférieure à celle du Zombie.
 - Le **Squelette** attaque à distance avec des morceaux de cerveau. Il a une vitesse de déplacement égale au Mage
- Chevalier Corrompu
 - La **Brute** a le double de PV de base du Mage et à une vitesse de déplacement lente. Inflige-lui deux points de dégâts.
 - **L'Archer** à le même nombre de pv que le joueur et à une vitesse de déplacement égale. Il attaque à distance.
 - Le **Chevalier** à plus de pv que le Mage et à une vitesse de déplacement plus rapide que le mage. Il attaque au corps à corps.
- Boss

Les boss ont plus de PV, infligent plus de dégâts et ont plus de vitesse que les monstres simples.







 - **L'Orc** fonce sur le personnage joueur et lui inflige des dégâts lorsqu'il le touche.
 - Le **Mort Vivant** lancera des morceaux de cerveau autour de lui.
 - Le **Chevalier** fonce sur le personnage joueur et pendant qu'il fonce, il tire autour de lui des flèches, lui aussi casse les rochers et lorsqu'il arrive à la moitié de sa vie, il augmente sa vitesse.

- Équipement

- Power-up

Équipement utilisé uniquement durant la partie actuelle, sera supprimé à la fin de celle-ci. Voici un tableau reprenant les objets.

Les livres peuvent uniquement être obtenus dans les magasins.

Objet	Image	Effet	Prix
Arcane de dégâts		Augmente les dégâts du joueur.	20
Arcane de vitesse d'attaque		Baisse le temps entre deux attaques.	10
Arcane de vitesse de déplacement		Augmente la vitesse de déplacement du joueur.	10
Arcane de vitesse des projectiles		Augmente la vitesse des projectiles du joueur.	15
Cœurs vide		Augmente le nombre maximum de cœurs du joueur.	/
Cœurs plein		Rend un cœur au joueur.	/

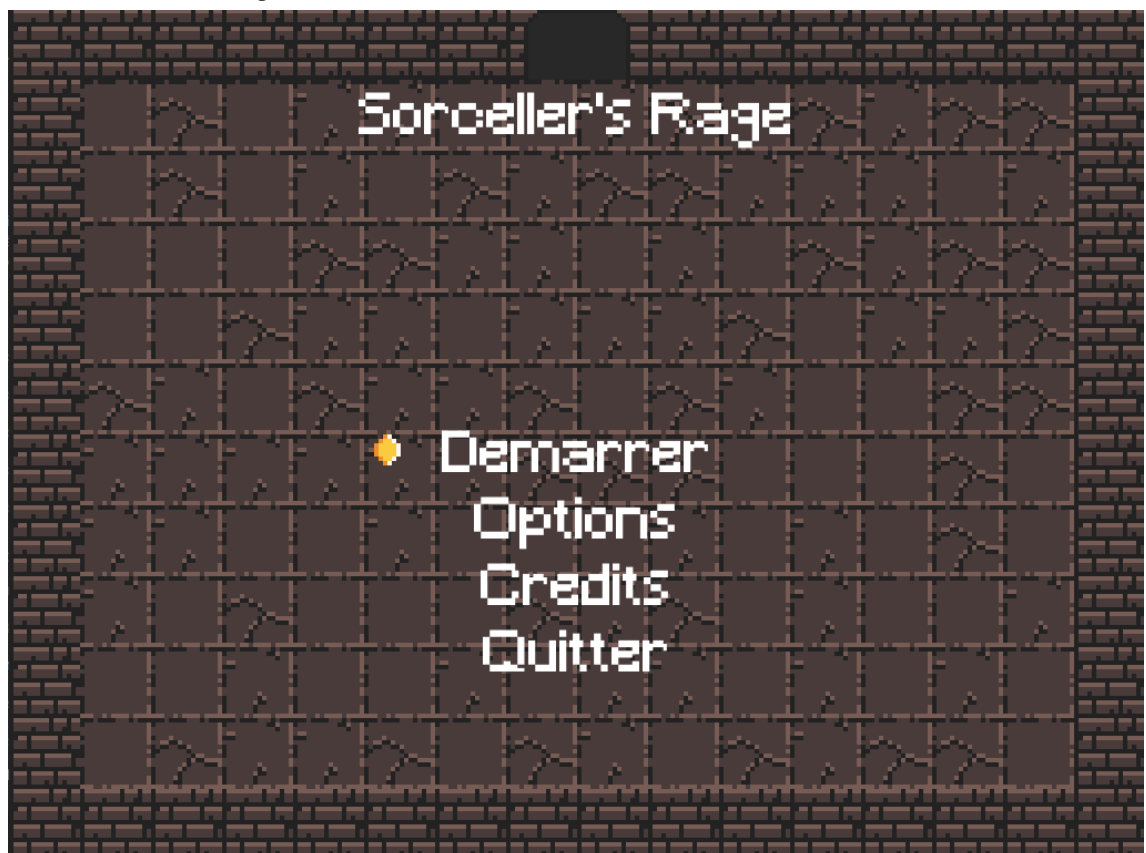
- Donjon
 - Type de salle
 - Le **magasin** permet d'acheter des arcanes ou des équipements avec l'or gagné dans le donjon.
 - La **zone de boss** est la zone de combat contre le boss de l'étage, il pourrait y avoir quelques monstres avec lui.
 - Les **zones de combat** sont des salles reliant la salle de départ avec les autres. Elles contiennent des ennemis, des coffres, des pièges.
 - La **salle de départ** est la zone d'où part le personnage pour chacun des étages.
 - Salle
 - Les **murs** sont autour de la salle, bloquant l'accès aux autres salles.
 - Les **portes** permettent l'accès aux différentes salles du donjon. Elles sont fermées et ne s'ouvrent que lorsque tous les monstres sont vaincus.
 - Les **échelles** permettent, après avoir vaincu le boss, d'accéder à l'étage suivant du donjon.
 - Objet
 - Les **coffres** contiennent un peu d'or. Présents dans les zones de combat, ces derniers ont une probabilité de ne rien vous donner, et de se transformer en un horrible coffre mangeur de voyageur.
 - Les **pics** sont des cases qui infligent des dégâts.
 - Les **pièces** sont la monnaie du jeu.

Interface utilisateur

Lorsque l'on lance le jeu, un menu s'affiche, celui-ci nous permet de choisir parmi différentes options : le personnage, accéder aux options, aux crédits ou encore quitter le jeu.

Dès que l'on a choisi notre personnage, le jeu se déroule, on a accès à des salles comme le magasin ou encore les salles de boss.

Menu de démarrage



- L'option "Démarrer" permet d'accéder au menu de choix de personnage
- L'option "Options" permet d'accéder au menu de sélections des options
- L'option "Crédits" permet d'afficher les crédits
- L'option "Quitter" permet de quitter le jeu

Menu de choix des personnages



- Choix entre deux personnages
- Affichage si le deuxième personnage est débloqué :

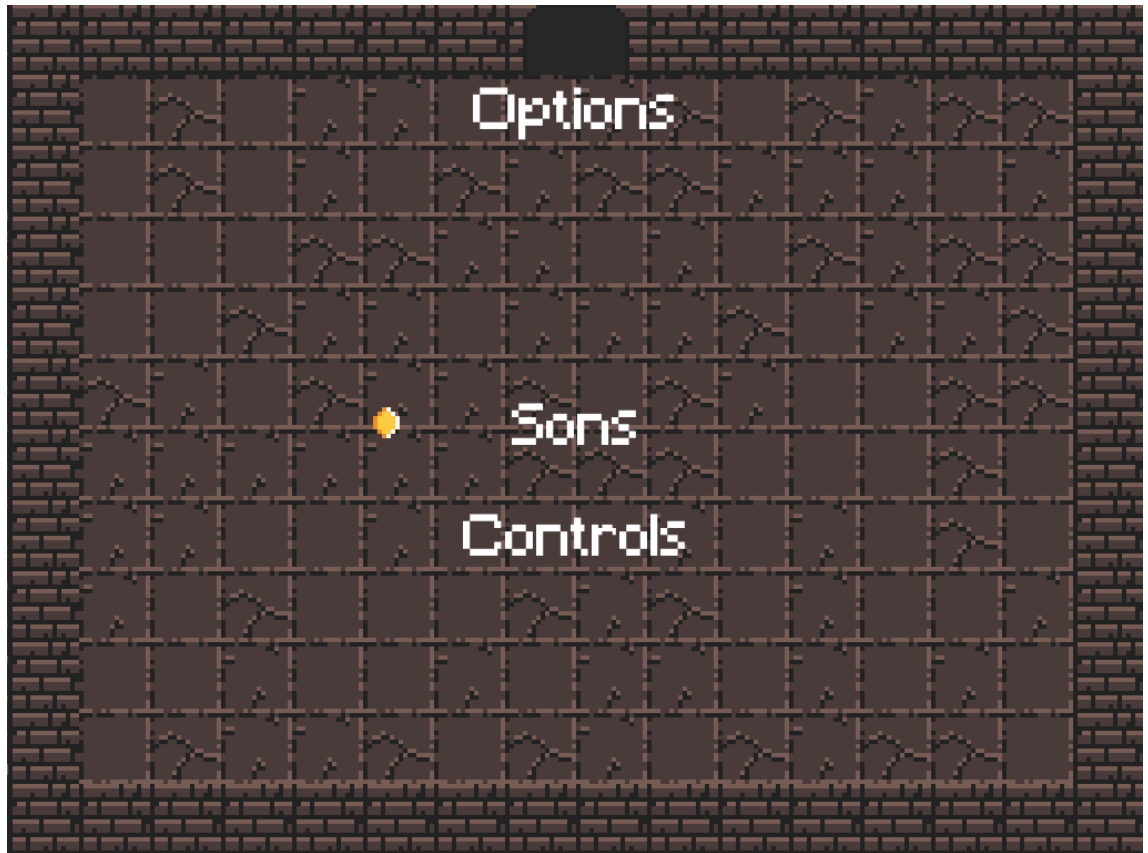


Menu pause



- Bouton d'accès aux options
- Bouton pour quitter le donjon

Menu options



- Bouton pour accéder aux paramètres de sons
- Bouton pour accéder aux contrôles du joueur

Menu Sons



- Permet de gérer le volume des sons.
La flèche de gauche baisse le son.
La flèche de droite augmente le son.

Menu Controls



- Affichage des contrôles du joueur

Menu Crédit



- Affichage des crédits du jeu

HUD du Jeu

Head Up Display (Affichage tête haute, en français) sont les informations que l'on retrouve en périphérie de l'écran permettant de voir les informations de notre personnage.



- Les cœurs qui représentent la vie du personnage sont affichés en haut à gauche.
- À droite, affichage de l'or gagné.
- En bas à droite, affichage de l'étage où le personnage se trouve.

Salle de boss



- La vie du boss est affichée en haut de la fenêtre.

Salle de shop



- Les livres sont au nombre de 4, leurs prix sont affichés juste en dessous de ces derniers.
Leurs effets ont été listé plutôt

Analyse

Arborescence

- `font/` regroupe toutes les polices de caractères utilisées dans le jeu.
- `map/` regroupe toutes les “cartes” du jeu
- `objects/` dossier comprenant tous les objets du jeu
 - `chest.py` Coffre
 - `class Chest`
 - `coin.py` Pièce
 - `class Coin`
 - `heart.py` Coeur remplis
 - `class Heart`
 - `heart_container.py` Réceptacle de Coeur
 - `class Heart_container`
 - `spell_book/` dossier comprenant tous les livres
 - `spell_book.py` Fichier général des livres.
 - `class Spell_Book` Tous les livres qui suivent héritent de cette classe.
 - `attack_book.py` Livre augmentant l'attaque
 - `class Attack_Book`
 - `cooldown_book.py` Livre diminuant le temps entre deux attaques
 - `class Cooldown_Book`
 - `projectile_speed_book.py` Livre augmentant la vitesse des projectiles
 - `class Projectile_Speed_Book`
 - `speed_book.py` Livre augmentant la vitesse du personnage joueur
 - `class Speed_Book`

- `perso/` contient toutes les classes associées aux personnages du jeu.
 - `player/` Contient tous les personnages joueurs
 - `player.py` Fichier général des personnages joueurs.
 - `class Player` Tous les personnages qui suivent héritent de cette classe.
 - `wizard.py` Mage
 - `class Wizard`
 - `fairy.py` Fée
 - `class Fairy`
 - `mob/` Contient tous les personnages monstres
 - `mob.py` Fichier général des personnages monstres.
 - `class Mob` Tous les personnages qui suivent héritent de cette classe.
 - `bad_chest.py` Coffre maudit
 - `class Bad_Chest`
 - `orc/` Contient tous les monstres orcs
 - `orc_brawler.py` Combattant orc
 - `class Orc_Brawler`
 - `orc_chaman.py` Chaman orc
 - `class Orc_Chaman`
 - `orc_fighter.py` Orc de base
 - `class Orc_Fighter`
 - `orc_nain.py` Orc nain
 - `class Orc_nain`
 - `undead/` Contient tous les monstres “zombie”
 - `zombie.py` Zombie
 - `class Zombie`
 - `blue_zombie.py` Zombie de glace
 - `class Blue_Zombie`
 - `skelet.py` Squelette
 - `class Skelet`
 - `tiny_zombie.py` Bébé zombie
 - `class Tiny_Zombie`

- `knight/` Contient tous les monstres chevaliers
 - `knight_brute.py` Chevalier corrompu brute
 - `class Knight_Brute`
 - `knight.py` Chevalier corrompu
 - `class Knight`
 - `bow_man.py` Archer
 - `class Bow_man`
- `boss/` Contient tous les monstres chevaliers
 - `boss.py` Fichier général des personnages boss.
 - `class Boss` Tous les personnages qui suivent héritent de cette classe.
 - `knight_boss.py` Boss Chevalier
 - `class Knight_Boss`
 - `orc_boss.py` Boss Orc
 - `class Orc_Boss`
 - `undead_boss.py` Boss Mort Vivant
 - `class Undead_Boss`
- `projectile/` contient toutes les classes associées aux projectiles du jeu
 - `arrow.py` Flèche
 - `class Arrow`
 - `ball.py` Boule
 - `class Ball`
 - `brain.py` Projectile de morceau de cerveau
 - `class Brain`
 - `fire_ball.py` Boule de feu
 - `class Fire_ball`
- `sprites/` regroupe toutes les images du jeu. Les images sont organisées par dossiers.
- `environnement.py` gestion des scripts de l'environnement
 - `class Wall` Objet Mur
 - `class Floor` Objet Sol
 - `class Door` Objet Porte
 - `class Ladder` Objet Echelle
 - `class Rock` Objet Pierre
 - `class Spike` Objet Piques
 - `class Wall_Hole` Objet Mur qui tire des flèches

- `class Win_Ladder` Objet Échelle de victoire
- `game.py` Contient la logique du jeu et la boucle principale
 - `class Game`
- `tfe.py` Lance le jeu.
- `menu.py` Contient les menus
 - `class Menu`
 - `class MainMenu`
 - `class OptionsMenu`
 - `class CreditsMenu`
 - `class ControlMenu`
 - `class Quit`
 - `class Death_Menu`
 - `class Pause_Menu`
 - `class Confirm_Quit`
 - `class Sound_Menu`
 - `class Win_Menu`
 - `class Select_Player_Menu`
- `tilemap.py` gestion de la carte et de la camera
 - `class Camera`
 - `class Map`
- `config.ini` Fichier de configuration du jeu

Fichier de configuration

config.ini

Le fichier de configuration sera utilisé pour gérer les sons, savoir si le deuxième personnage est débloqué et enfin le nombre de parties jouée/perdue/gagnée. Si le fichier est malheureusement supprimé, il sera automatiquement recréé en remettant tous les attributs à leurs valeurs par défaut.

SOUND

La division `SOUND` permettra de sauvegarder les paramètres de sons du jeu.

Attributs

- `bg: int` → Valeur du paramètre du fond sonore.
- `shoot: int` → Valeur du paramètre du son du tir.
- `coin: int` → Valeur du paramètre du son du ramassage de pièces.
- `menu: int` → Valeur des paramètres des sons du menu.

CHARACTERS

La division `CHARACTERS` permettra de sauvegarder si le deuxième personnage est débloqué ou non.

Attributs

- `fairy_unlocked: Bool` → Booléen pour savoir si la fée est débloquée.

GAME

La division `GAME` permettra de sauvegarder le nombre de victoires, défaites et le nombre de parties jouées.

Attributs

- `game_winned: int` → Nombre de parties gagnées.
- `game_loosed: int` → Nombre de parties perdues.
- `game_played: int` → Nombre de parties jouées.

Classes

player.py

Player

La classe `Player` hérite de la classe `Sprite`. Cela permettra de gérer tout ce qui utilisera les collisions du personnage.

La classe `Player` permettra de créer un personnage joueur.

Ces attributs et méthodes permettent de définir tout type de personnage. Certaines méthodes seront surchargées dans les différentes classes, selon le besoin du personnage.

Attributs

- `groups : sprite.Group` → Récupère le `all_sprite` de game.
- `projectile: sprite.Group` → Groupe contenant tous les projectiles.
- `time: int` → Utilisé pour trouver un temps d'attente.
- `cooldown: int` → Utilisé pour le délai entre deux attaques.
- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `max_health: int` → point de vie maximum du personnage.
- `health: int` → point de vie du personnage.
- `coin: int` → Nombre de pièces récupérées.
- `attack_point: int` → Points d'attaque du personnage.
- `player_speed: int` → Vitesse de déplacement du personnage..
- `game: object` → Permet d'accéder à tout ce que le jeu contient.
- `sprites_idle_l, sprites_idle_r, sprite_move_l, sprite_move_r: list` → Liste des images du personnage.
- `hearts: list` → Listes des images des coeurs.
- `current_sprite: int` → Nombre indiquant quelle image représente le personnage.
- `image: int` → Image actuelle du personnage.
- `move: bool` → Permet de savoir si le personnage se déplace.
- `rect: rect` → Rectangle de l'image du personnage.
- `vel: vec` → Vecteur de déplacement du personnage.
- `pos: vec` → Positions du personnage sur la grille.
- `direc: str` → Permet de connaître la direction du personnage pour

l'animation.

- `direc_idle: str` → Permet d'avoir la direction dans laquelle le personnage est immobile pour son animation.
- `debug: bool` → Savoir si le menu debug est activé.
- `projectile_speed: int` → Vitesse du projectile tiré par le personnage.
- `fly: bool` → indique si le personnage vole ou non.

Méthodes

- `def set_position(x, y)` → La fonction permet de placer le personnage sur la grille de jeu.
- `def set_money(coin_value)` → La fonction permet de mettre à jour le nombre de pièces dont le personnage dispose.
En prenant compte que le joueur ne peut pas dépasser les 99 pièces et aller en dessous de 0.
- `def set_health(value)` → La fonction permet de mettre à jour le nombre de points de vie dont le personnage dispose.
En prenant compte que le personnage ne peut pas dépasser le nombre de points de vie maximum.
- `def set_max_health(value)` → La fonction permet de mettre à jour le nombre maximum de points de vie que le personnage a.
En prenant compte que le personnage ne peut pas dépasser les 10 cœurs maximum donc 20 pv.
- `def action()` → La fonction permet le déplacement du personnage en détectant si une touche est enfoncée.
Lorsqu'une touche est enfoncée, la vélocité change selon le sens.
La direction de déplacement changera et si le déplacement se fait soit à gauche soit à droite, alors la direction lorsque le personnage sera immobile changera en conséquence.
Et enfin la variable move changera pour True ce qui veut dire que le personnage se déplace et permet à la fonction qui gère l'animation de passé d'immobile à mobile.
- `def debug_menu()` → La fonction permet d'activer les touches de debug lorsqu'une touche est enfoncée.
Dans ses "options" de debug on retrouve le moyen de tuer tous les monstres d'une salle, de mettre les pièces du personnage à 99, de "suicider" le personnage et d'activer l'invincibilité

- `def shooting()` → La fonction permet de tirer un projectile en détectant si une touche est enfoncée.
Lorsqu'une touche est enfoncée, on appelle la fonction de tir dans le sens ou la touche a été enfoncée.
La direction de déplacement changera, et s'il le fait, la direction immobile changera en conséquence.
- `def shoot(direc, attack_point)` → La fonction permet de créer un projectile en utilisant les points d'attaque du personnage mais aussi en récupérant la direction dans laquelle tirer.
Pour que ce dernier soit créé, il faut que le temps entre deux attaques soit atteint.
- `def collide_with_walls(dir)` → La fonction permet de savoir si le personnage rentre en contact avec un obstacle.
Si le personnage rentre en contact avec ce dernier, alors il sera immobilisé pour éviter qu'il ne rentre dans l'obstacle, sauf s'il peut voler, dans ce cas, il évitera ce dernier.
- `def animator(player_speed)` → La fonction permet de gérer l'animation du personnage Elle permet d'afficher l'image adéquate en fonction de la position du personnage.
- `def draw_health()` → La fonction permet de dessiner les cœurs du personnage en prenant compte qu'un cœur vaut 2 points de vie.
Lorsque les nombres de points de vie sont impairs, cela permet d'afficher un demi cœur à la place d'un complet.
Et pour afficher des cœurs vides, elle regarde combien de points de vie max le personnage possède, s'il est supérieur à 2 par rapport aux points de vie actuels, alors il affiche un cœur vide.
Tant qu'il y a moins de 6 coeurs sur l'écran, ils sont sur une ligne, mais si jamais on atteint 6 coeurs, alors ils commenceront à être dessiner en dessous de la première ligne
- `def is_alive()` → La fonction permet de savoir si le personnage est toujours en vie et s'il ne l'est plus, il va afficher le menu de mort et supprimer le personnage de la scène.
- `def update()` → La fonction appellent toutes les autres fonctions définies ci-dessus, ce qui permet qu'à chaque frame, le jeu utilise toutes ses fonctions pour que le personnage fonctionne

wizard.py

Wizard

La classe `Wizard` héritera de la classe `Player`.

Attributs surchargés

- `sprites_idle_l, sprites_idle_r, sprite_move_l, sprite_move_r: list` → Liste des images du Mage.
- `player_speed: int` → vitesse de déplacement du Mage.
- `attack_point: int` → Points d'attaque du Mage.
- `max_health: int` → Points de vie.
- `health: int` → Points de vie.
- `projectile_speed: int` → Vitesse des projectiles.
- `cooldown: int` → Vitesse d'attaque.

fairy.py

Fairy

La classe `Fairy` héritera de la classe `Player`.

Attributs surchargés

- `sprites_idle_l, sprites_idle_r, sprite_move_l, sprite_move_r: list` → Liste des images de la fée
- `player_speed: int` → vitesse de déplacement de la fée..
- `attack_point: int` → Points d'attaque.
- `max_health: int` → Points de vie.
- `health: int` → Points de vie.
- `projectile_speed: int` → Vitesse des projectiles.
- `cooldown: int` → Vitesse d'attaque.
- `fly: bool` → La fée vole.

mob.py

Mob

La classe `Mob` permettra de créer les monstres.

La classe `Mob` hérite de la classe `Sprite`. Cela permettra de gérer tout ce qui utilisera les collisions du personnage.

Ces attributs et méthodes permettent de définir tout type de personnage. Certaines méthodes seront surchargées dans les différentes classes selon le besoin du personnage.

Attributs

- `can_shoot : bool` → Permet de savoir si le monstre peut tirer un projectile.
- `player: obj` → Permet d'avoir accès à l'objet personnage.
- `coin: sprite.Group` → Groupe contenant les pièces.
- `health: int` → Point de vie du monstre.
- `game: int` → Permet de récupérer tout ce que comporte le fichier game afin de les utiliser.
- `state: str` → État dans lequel se trouve le monstre, il peut être soit vivant soit mort.
- `_layer: int` → Couche sur laquelle sera dessiné mis l'objet.
- `mob_speed: int` → Vitesse de déplacement du mob.
- `sprite: list` → Liste des images du monstre.
- `current_sprite: int` → Nombre qui dit sur quelle image on est dans la liste d'images du monstre.
- `image: int` → Image actuelle du monstre.
- `rect: rect` → Rectangle de l'image du monstre.
- `dirvect: vec` → Vecteur de direction du monstre pour le déplacement.
- `pos: vec` → Position du personnage sur la grille.
- `cooldown: int` → Est le délai entre deux attaques.
- `forced : bool` → Permet de savoir si le monstre est forcé de lâcher des pièces lorsqu'il meurt.
- `vect: vec` → Vecteur de direction pour le tir du monstre.

Méthodes

- `def move_toward_player(player)` → La fonction permet de calculer un vecteur en direction du joueur.
Ce vecteur permet au monstre de se déplacer vers le joueur.
- `def avoid_other()` → La fonction fait en sorte d'éviter que les monstres ne se chevauchent lors de leur déplacement. Et ne chevauchent pas le personnage du joueur.
- `def collide_with_walls(dir)` → La fonction permet de savoir si le MONSTRE rentre en contact avec un obstacle.
Si le personnage rentre en contact avec ce dernier, alors il sera immobilisé pour éviter qu'il ne rentre dans l'obstacle.
- `def animator()` → La fonction permet de gérer l'animation du monstre en utilisant un chiffre qui donnera quelle image doit être affichée dans la liste d'animation adéquate afin d'afficher l'image du monstre.
- `def shoot()` → La fonction permet de créer un projectile en direction du joueur grâce à un vecteur.
Pour que ce dernier soit créé, il faut que le temps entre deux attaques soit atteint.
- `def death()` → La fonction permet de faire apparaître une pièce sur le terrain si le nombre aléatoire vaut 1 ou alors si il est forcé à la faire apparaître.
Une fois la pièce apparue, le monstre sera supprimé du terrain.
- `def update()` → La fonction appellent toutes les autres fonctions définies ci-dessus, ce qui permet qu'à chaque frame, le jeu utilise toutes ses fonctions pour que le personnage fonctionne.
De plus, elle vérifiera si le monstre à 0 pv, si oui, elle activera un effet de mort et la fonction de mort.
Par contre s'il n'est pas mort, il jouera son animation, ensuite se déplacera vers le joueur, s'il peut tirer alors il tirera, il évitera les autres monstres, mais aussi le joueur, ensuite sa position sera changée et on vérifiera si le monstre touche un mur ou non

orc_brawler.py

Orc_Brawler

La classe `Orc_Brawler` héritera de la classe `Mob`.

Attributs surchargés

- `sprite: list` → Liste des images du combattant
- `mob_speed: int` → vitesse de déplacement.
- `health: int` → Points de vie.

orc_chaman.py

Orc_Chaman

La classe `Orc_Chaman` héritera de la classe `Mob`.

Attributs

- `projectile: list` → Liste des projectiles dans le jeu.

Attributs surchargés

- `sprite: list` → Liste des images du chaman.
- `mob_speed: int` → vitesse de déplacement.
- `health: int` → Moins de points de vie que le combattant mais plus que le nain et l'orc normal.
- `can_shoot: bool` → Permet au monstre d'utiliser le tir.

orc_fighter.py

Orc_Fighter

La classe `Orc_Fighter` héritera de la classe `Mob`.

Attributs surchargés

- `sprite: list` → Liste des images de l'orc.
- `mob_speed: int` → Vitesse de déplacement.
- `health: int` → Moins de points de vie que le combattant mais plus que le nain et l'orc normal.

orc_nain.py

Orc_Nain

La classe `Orc_Nain` héritera de la classe `Mob`.

Attributs surchargés

- `sprite: list` → Liste des images du nain.
- `mob_speed: int` → vitesse de déplacement la plus haute de tous les orcs.
- `health: int` → Orc avec le moins de points de vie.

blue_zombie.py

Blue_Zombie

La classe `Blue_Zombie` héritera de la classe `Mob`.

Attributs

- `projectile: list` → Liste des projectiles dans le jeu.

Attributs surchargés

- `sprite: list` → Liste des images du zombie de glace.
- `mob_speed: int` → vitesse de déplacement égale à l'orc chaman.
- `health: int` → Le zombie avec le plus de points de vie.
- `can_shoot: bool` → Permet au monstre d'utiliser le tir.

zombie.py

Zombie

La classe `Zombie` héritera de la classe `Mob`.

Attributs surchargés

- `sprite: list` → Liste des images du zombie.
- `mob_speed: int` → vitesse de déplacement au-dessus de celle du zombie de glace mais plus lent que le bébé zombie et le squelette.
- `health: int` → Moins de points de vies que le zombie de glace, mais plus que le squelette et le bébé zombie.

skelet.py

skelet

La classe `skelet` héritera de la classe `Mob`.

Attributs

- `projectile: list` → Liste des projectiles dans le jeu.

Attributs surchargés

- `sprite: list` → Liste des images du squelette.
- `mob_speed: int` → vitesse de déplacement au-dessus de celle du zombie de glace et du zombie mais plus lent que le bébé zombie.
- `health: int` → Le squelette à moins de vie que le zombie et le zombie de glace mais plus que le bébé zombie.
- `can_shoot: bool` → Permet au monstre d'utiliser le tir.

tiny_zombie.py

Tiny_Zombie

La classe `Tiny_Zombie` héritera de la classe `Mob`.

Attributs surchargés

- `sprite: list` → Liste des images du bébé zombie.
- `mob_speed: int` → Le zombie avec la plus haute vitesse.
- `health: int` → Le zombie avec le moins de points de vie.

knight.py

Knight

La classe `Knight` héritera de la classe `Mob`.

Attributs surchargés

- `sprite: list` → Liste des images du chevalier.
- `mob_speed: int` → Le plus rapide des trois chevaliers.
- `health: int` → Le chevalier a plus de points de vie que l'archer, mais moins que la brute.

knight_brute.py

Knight_Brute

La classe `Knight_Brute` héritera de la classe `Mob`.

Attributs surchargés

- `sprite: list` → Liste des images de la brute.
- `mob_speed: int` → Le chevalier avec le moins de vitesse.
- `health: int` → Le chevalier avec le plus de points de vie.

bow_man.py

Bow_Man

La classe `Bow_Man` héritera de la classe `Mob`.

Attributs

- `projectile: list` → Liste des projectiles dans le jeu.

Attributs surchargés

- `sprite: list` → Liste des images du mage.
- `mob_speed: int` → Moins de vitesse que le chevalier, mais plus que la brute.
- `health: int` → Le monstre de cette race avec le moins de points de vie.
- `can_shoot: bool` → Permet au monstre d'utiliser le tir.

bad_chest.py

Bad_Chest

La classe `Bad_Chest` héritera de la classe `Mob` et créera un coffre mangeur d'homme.

Attributs surchargés

- `sprite: list` → Liste des images du coffre.
- `mob_speed: int` → Vitesse du coffre.
- `forced: bool` → Force l'apparition de pièce lors de la mort du coffre.

boss.py

Boss

La classe `Boss` permettra de créer les boss.

La classe `Boss` hérite de la classe `Sprite`. Cela permettra de gérer tout ce qui utilisera les collisions du personnage.

Ces attributs et méthodes permettent de définir tout type de personnage. Certaines méthodes seront surchargées dans les différentes classes selon le besoin du personnage.

Attributs

- `window : win` → Permet de récupérer la fenêtre de jeu.
- `player: obj` → Permet d'avoir accès à l'objet joueur.
- `coin: sprite.Group` → Groupe contenant les pièces.
- `health: int` → Point de vie du monstre.
- `max_health: int` → Point de vie maximum du monstre.
- `health_bar_lenght: int` → Taille de la représentation de la barre de vie.
- `health_ratio: int` → Transforme les points de vie du boss en distance pour la barre de vie.
- `game: int` → Permet de récupérer tout ce que comporte le fichier game afin de les utiliser.
- `state: str` → État dans lequel se trouve le Boss, il peut être soit vivant soit mort.
- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `mob_speed: int` → Vitesse de déplacement du boss.
- `sprite: list` → Liste des images du monstre.
- `current_sprite: int` → Nombre qui dit sur quelle image on est dans la liste d'images du monstre.
- `image: int` → Image actuelle du boss.
- `rect: rect` → Rectangle de l'image du boss.
- `dirvect: vec` → Vecteur de direction du boss pour le déplacement.
- `pos: vec` → Position du boss sur la grille.
- `cooldown: int` → Est le délai entre deux attaques.
- `is_moving : bool` → Permet de savoir si le boss est en train de se

déplacer.

Méthodes

- `def avoid_other()` → La fonction fait en sorte d'éviter que les monstres ne se chevauchent lors de leur déplacement et ne chevauchent également le personnage joueur.
- `def health_bar()` → La fonction permet d'afficher la barre de vie du Boss sur la fenêtre de jeu,
- `def collide_with_walls(dir)` → La fonction permet de savoir si le boss rentre en contact avec un obstacle.
Si le personnage rentre en contact avec ce dernier, alors il sera immobilisé pour éviter qu'il ne rentre dans l'obstacle.
Aussi, la fonction retournera s'il y a eu contact avec un mur.
- `def destroy_rock()` → La fonction permet de vérifier s'il y a contact avec un rocher, si oui, on regarde dans quel étage on se trouve, si on est dans le premier, on supprime le rocher.
Par contre si on est à l'étage trois, alors on tire une valeur entre 0 et 1.
Si c'est 0 on appelle la fonction `shooting()` en envoyant en paramètre 'cross' pour faire un tir en croix autour du boss.
Si c'est 1 on appelle la fonction `shooting()` en envoyant en paramètre 'plus' pour faire un tir en 'plus' autour du boss.
Enfin, on supprime le rocher du terrain.
- `def shooting(choose_type)` → La fonction permet de faire apparaître des projectiles dans la partie autour du boss en comparant le paramètre 'choose_type' avec un dictionnaire, cela donnera des valeurs de directions qui seront envoyées dans la création d'un projectile.
Ce dernier est ajouté à tous les autres projectiles.
- `def animator()` → La fonction permet de gérer l'animation du monstre en utilisant un chiffre qui donnera quelle image doit être affichée dans la liste d'animation adéquate afin d'afficher l'image du monstre.
- `def death()` → La fonction permet de vérifier si le boss est en vie, si oui, alors il active l'effet de mort du fichier `effects.py`. Une fois fait, le boss sera supprimé du terrain et fera apparaître des pièces.
- `def attack()` → La fonction permet de détecter quand le boss rentre en collision avec le joueur, si c'est le cas, alors il inflige 2 points de dégât
- `def dash()` → La fonction permet de créer un vecteur vers le joueur qui

sera sauvegardé pour son déplacement futur

- `def move()` → La fonction permet de changer la position du boss en utilisant le vecteur créé plus tôt.
- `def update()` → La fonction ne fait rien. Elle est destinée à être surchargée dans les classes enfants.

orc_boss.py

Orc_Boss

La classe `Orc_Boss` héritera de la classe `Boss`.

Attributs surchargés

- `sprite: list` → Liste des images du boss.
- `mob_speed: int` → Vitesse du boss orc.
- `max_health: int` → Ce boss aura le moins de PV des 3.
- `health: int` → Ce boss commencera avec le moins de PV des 3.
- `health_ratio: int` → Permet de remettre la barre de vie avec les bonnes proportions.

Méthodes surchargés

- `def update()` → La fonction commencera par vérifier si le boss est toujours en vie, si oui alors elle commencera par afficher la barre de vie et jouer l'animation.
Ensuite, elle vérifiera si le boss bouge, si il ne bouge pas elle appellera `dash()` qui créera le vecteur de déplacement une fois fait, il appliquera ce vecteur sur la position du boss afin de le faire se déplacer ensuite de vérifier si le boss rencontre le joueur afin de lui infliger des dégâts.
Pour finir, on vérifie si il y a collision avec les murs, si oui, alors le boss s'arrêtera.
Enfin, on vérifie si le boss touche un rocher, si oui, il casse le rocher et s'arrête.

undead_boss.py

Undead_Boss

La classe `Undead_Boss` héritera de la classe `Boss`.

Attributs

- `projectile: list` → Liste des projectiles du jeu.
- `val: int` → Valeur qui permet de savoir quel type de tir sera effectué par le boss.
- `move_cooldown: int` → Temp d'attente entre deux déplacements.

Attributs surchargés

- `sprite: list` → Liste des images du boss
- `mob_speed: int` → Vitesse du boss mort-vivant
- `max_health: int` → Ce boss a plus de PV que le boss orc mais moins que le chevalier
- `health: int` → Ce boss commencera avec plus de PV que le boss orc mais moins que le chevalier
- `health_ratio: int` → Permet de remettre la barre de vie avec les bonnes proportions

Méthodes

- `def move_aleat()` → La fonction vérifie si le temps d'attente entre deux déplacements est atteint, si oui alors il crée un vecteur avec des valeurs au hasard qui sera utilisé pour le déplacement du boss

Méthodes surchargés

- `def collide_with_wall()` → La fonction vérifie si une collision intervient entre le monstre et un élément du décor (mur et rocher) et si il y a contact, alors on arrête le déplacement du boss.
- `def update()` → La fonction commencera par vérifier si le boss est toujours en vie, si il l'est alors il commencera par afficher la barre de vie et jouer l'animation.
Ensuite, il vérifiera si le temps passé entre deux tirs est atteint, si oui, alors le boss tirera en croix, en rond ou en cercle.
Ensuite, on appelle la fonction de mouvement aléatoire pour faire

déplacer le boss aléatoirement sur le plateau.

Pour finir, on vérifiera s'il y a collision avec les murs ou les rochers, si oui, alors le boss s'arrête de bouger et si le boss touche un rocher, il casse le rocher et s'arrête.

knight_boss.py

Knight_Boss

La classe `Knight_Boss` héritera de la classe `Boss`.

Attributs

- `projectile: list` → Liste des projectiles du jeu.

Attributs surchargés

- `sprite: list` → Liste des images du boss.
- `mob_speed: int` → Vitesse du boss orc.
- `max_health: int` → Ce boss aura le plus de PV de tous les boss.
- `health: int` → Ce boss commencera avec le plus de PV de tous les boss.
- `health_ratio: int` → Permet de remettre la barre de vie avec les bonnes proportions.

Méthodes surchargés

- `def update()` → La fonction est un mixe des deux précédentes elle commencera par vérifier si le boss est toujours en vie, si il l'est alors il commencera afficher la barre de vie et jouera l'animation.
Ensuite il vérifiera si le boss bouge, s'il ne bouge pas la fonction `dash()` sera appelée et permet au boss de foncé sur le joueur; pendant qu'il le fait, on vérifie si le temps entre deux tir sont atteint, si oui on créer des projectiles en cercles autour du boss, ensuite elle vérifie si le boss touche le joueur afin de lui infliger des dégâts.
Pour vérifier s'il y a collision avec les murs, si oui, alors le boss s'arrêtera, et créera des projectiles autour de lui. Pour finir, on vérifie si le boss touche un rocher, si oui, il casse le rocher et s'arrête.

chest.py

Chest

La classe `Chest` Permettra de créer des coffre sur le plateau de jeu

Attributs

- `pg.sprite.Sprite.__init__ : init` → Initie le module Sprite.
- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `game: img` → Permet de récupérer tout ce que comporte le fichier game afin de les utiliser.
- `coin: sprite.Group` → Groupe contenant les pièces.
- `sprite_c: list` → Liste des images du coffre.
- `current_sprite: int` → Nombre qui dit sur quelle image on est dans la liste d'images du coffre.
- `image: int` → Image actuelle du coffre.
- `rect: rect` → Rectangle de l'image du coffre.
- `x: int` → Coordonnée x de placement de l'objet sur la grille.
- `y: int` → Coordonnée y de placement de l'objet sur la grille.
- `rect.x: int` → Permet de centrer la pièce en x sur une case.
- `rect.y: int` → Permet de centrer la pièce en y sur une case.
- `coin: sprite.Group` → Groupe contenant toutes les pièces du jeu.
- `c_open: bool` → Permet de savoir si le coffre est ouvert.
- `pos: vec` → Positions du coffre sur la grille.

Méthodes

- `def chest_col()` → Si le joueur rentre en collision avec ce dernier, alors le coffre jouera son animation d'ouverture
- `def animator()` → Gère les animations, lorsque l'animation d'ouverture est finie, il va appeler la fonction `state()`
- `def state()` → La fonction permettra de soit faire apparaître un monstre soit un certain nombre de pièces.
- `def update()` → appel de la fonction `chest_col()`

coin.py

Coin

La classe `Coin` permettra de créer des pièces sur le plateau de jeu.

Attributs

- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `game: obj` → Permet de récupérer tout ce que comporte le fichier game afin de les utiliser.
- `sprite_y, sprite_r, sprite_g: list` → Liste des images des pièces.
- `current_sprite: int` → Nombre qui dit sur quelle image on est dans la liste d'images des pièces.
- `image: img` → Image actuelle de la pièce.
- `rect: rect` → Rectangle de l'image de la pièce.
- `c_value: int` → Valeur de la pièce.
- `x: int` → Coordonnée x de placement de l'objet sur la grille.
- `y: int` → Coordonnée y de placement de l'objet sur la grille.
- `rect.x: int` → Permet de centrer la pièce en x sur une case.
- `rect.y: int` → Permet de centrer la pièce en y sur une case.

Méthodes

- `def coin_collect()` → permet de savoir si le joueur rentre en collision, et s'il le fait, alors la fonction `player.set_money()` est appelée. Elle permettra d'augmenter l'argent du joueur selon la valeur qui lui est donnée.
Ensuite, elle jouera le son de récupération et la pièce sera supprimée du plateau de jeu.
- `def animator()` → La fonction permet de gérer l'animation de la pièce en fonction de sa valeur et en utilisant un chiffre qui donnera quelle image doit être affiché dans la liste d'animation adéquate afin d'afficher l'image de la pièce.
- `def update()` → appel de la fonction `animator()`, ensuite `coin_collect()`.

heart_container.py

Heart_container

La classe `Heart_container` permettra de créer des réceptacles de cœurs sur le plateau de jeu.

Attributs

- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `game: obj` → Permet de récupérer tout ce que comporte le fichier game afin de les utiliser.
- `image: img` → Image du réceptacle de cœur .
- `rect: rect` → Rectangle de l'image.
- `x: int` → Coordonnées x de placement de l'objet sur la grille.
- `y: int` → Coordonnées y de placement de l'objet sur la grille.
- `rect.x: int` → Permet de centrer le réceptacle en x sur une case.
- `rect.y: int` → Permet de centrer la réceptacle en y sur une case.

Méthodes

- `def heart_collect()` → permet de savoir si le joueur rentre en collision, et s'il le fait, alors on vérifie qu'il a moins de vingt points de vie maximum et on lui en ajoute deux, une fois fait, on supprime le cœur.
- `def update()` → appel la fonction `heart_collect()`

heart.py

Heart

La classe `Heart` permettra de créer des réceptacles de cœurs sur le plateau de jeu.

Attributs

- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `game: obj` → Permet de récupérer tout ce que comporte le fichier game afin de les utiliser.
- `image: img` → Image du cœur.
- `rect: rect` → Rectangle de l'image.

- `x: int` → Coordonnée x de placement de l'objet sur la grille.
- `y: int` → Coordonnée y de placement de l'objet sur la grille.
- `rect.x: int` → Permet de centrer le cœur en x sur une case.
- `rect.y: int` → Permet de centrer le cœur en y sur une case.

Méthodes

- `def heart_collect()` → permet de savoir si le joueur rentre en collision, et s'il le fait, alors on vérifie qu'il a moins de points de vie plus deux que de vie maximum.
S'il en a plus, alors on ajoute 1 point de vie au joueur (demi-cœur)
Sinon on lui ajoute 2 points de vies.
- `def update()` → appel de la fonction `heart_collect()`

spell_book.py

Spell_Book

La classe `Spell_Book` permettra de créer des livres avec des effets sur le plateau de jeu.

Attributs

- `pg.sprite.Sprite.__init__ : init` → Initie le module Sprite.
- `_layer: int` → Couche sur laquelle sera mis l'objet .
- `game: obj` → Permet de récupérer tout ce que comporte le fichier game afin de les utiliser.
- `image: img` → Image du livre.
- `rect: rect` → Rectangle de l'image.
- `x: int` → Coordonnée x de placement de l'objet sur la grille.
- `y: int` → Coordonnée y de placement de l'objet sur la grille.
- `rect.x: int` → Permet de centrer la pièce en x sur une case.
- `rect.y: int` → Permet de centrer la pièce en y sur une case.
- `font_name: str` → Nom de la police utilisée pour afficher le prix du livre.
- `price: int` → Permet d'avoir le prix du livre.

Méthodes

- `def book_collect()` → Permet de savoir si le joueur rentre en collision, et s'il le fait, alors on vérifie qu'il a l'argent nécessaire pour acheter le livre.
S'il a assez, alors on retire l'équivalent du prix du livre en pièces d'or de la bourse du personnage, on applique son effet et on le supprime du jeu .
- `def draw_price()` → Permet d'afficher le prix d'un livre juste en dessous de ce dernier,
- `def effect()` → Effets engendré par le ramassage d'un livre
- `def update()` → Appel la fonction `draw_price()` et `book_collect()`

attack_book.py

Attack_Book

La classe `Attack_Book` héritera de la classe `Spell_Book`.

Attributs surchargés

- `image: img` → Image du livre.
- `price: int` → Prix du livre.

Méthode surchargé

- `def effect()` → Augmente les dégâts du joueur

cooldown_book.py

Attack_Book

La classe `Cooldown_Book` héritera de la classe `Spell_Book`.

Attributs surchargés

- `image: img` → Image du livre.
- `price: int` → Prix du livre.

Méthode surchargé

- `def effect()` → Baissera le délai entre deux attaques du joueur

projectile_speed_book.py

Projectile_Speed_Book

La classe `Projectile_Speed_Book` héritera de la classe `Spell_Book`.

Attributs surchargés

- `image: img` → Image du livre.
- `price: int` → Prix du livre.

Méthode surchargé

- `def effect()` → Augmente la vitesse du projectile que le joueur tire

speed_book.py

Speed_Book

La classe `Speed_Book` héritera de la classe `Spell_Book`.

Attributs surchargés

- `image: img` → Image du livre.
- `price: int` → Prix du livre.

Méthode surchargé

- `def effect()` → Augmente la vitesse du joueur

effect.py

Death_Effect

La classe `Death_Effect` permettra de créer des effets visuels lors de la mort des monstres.

Attributs

- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `sprite: list` → Liste des images de l'effet.
- `current_sprite: int` → Nombre qui dit sur quelle image on est dans la liste d'images de l'effet.
- `image: int` → Image actuelle de l'effet.
- `rect: rect` → Rectangle de l'image du monstre.
- `x: int` → Coordonnée x de placement de l'objet sur la grille.
- `y: int` → Coordonnée y de placement de l'objet sur la grille.
- `rect.x: int` → Permet de centrer la pièce en x sur une case.
- `rect.y: int` → Permet de centrer la pièce en y sur une case.

Méthodes

- `def death_anim()` → La fonction permet de gérer l'animation de l'effet en utilisant un chiffre qui donnera quelle image doit être affiché dans le liste d'animation adéquate afin d'afficher l'image de l'effet.
Lorsque l'effet est fini, on le supprime de la partie.

- `def update()` → Appel la fonction `death_anim()`

fire_ball.py

Fire_Ball

La classe `Fire_Ball` permettra de créer des projectiles qui sont des boules de feu.

Attributs

- `_layer: int` → Couche sur laquelle sera mis le projectile.
- `game: obj` → Permet de récupérer tout ce que comporte le fichier game afin de les utiliser.
- `image: img` → Image de la boule de feu.
- `rect: rect` → Rectangle de l'image centré avec deux coordonnées.
- `direct: str` → Direction dans laquelle ira le projectile.
- `damage: int` → Nombre de dégâts que le projectile fera.
- `v_fb: int` → Vitesse de déplacement de la boule de feu sur la grille.

Méthodes

- `def shoot_dir()` → La fonction commence par vérifier dans quelle direction il doit tirer la boule de feu, une fois vérifié, il change la position de la boule de feu
- `def collide_with_wall()` → La fonction vérifie si le projectile rentre en contact avec un mur ou un rocher, si il le fait, alors il est supprimé de la scène.
- `def attack()` → La fonction vérifie si le projectile touche un monstre, si oui, il joue le son de contact, ensuite il soustrait au monstre les dégâts de l'attaque de ses pv et supprime la boule de feu de la grille.
- `def update()` → Appel la fonction `shoot_dir()` ensuite `collide_with_wall()` enfin elle appelle `attack()`

ball.py

Ball

La classe `Ball` permettra de créer des projectiles.

Attributs

- `_layer: int` → Couche sur laquelle sera mis le projectile.
- `game: obj` → Permet de récupérer tout ce que comporte le fichier game afin de les utiliser.
- `image: img` → Image du livre.
- `rect: rect` → Rectangle de l'image centré avec deux coordonnées.
- `pos: str` → Position du projectile.
- `vect: str` → Vecteur donné à la classe par les monstres qui peuvent tirer.
- `damage: int` → Nombre de dégâts que le projectile fera.
- `v_b: int` → Vitesse de déplacement de la boule de feu sur la grille.

Méthodes

- `def collide_with_wall()` → La fonction vérifie si le projectile rentre en contact avec un mur ou un rocher, si il le fait, alors il est supprimé de la grille.
- `def attack()` → La fonction vérifie si le projectile touche le personnage joueur, si oui, il joue le son de contact, ensuite il soustrait au joueur les dégâts de l'attaque de ses pv et supprime la boule de feu de la grille
- `def update()` → La fonction commence par modifier la position du projectile en fonction du vecteur récupéré, ensuite elle modifie le rectangle en x et y à partir de la position.
Pour finir, elle vérifie s'il y a contact avec le joueur ou un mur.

brain.py

Brain

La classe `Brain` permettra de créer des projectiles qui sont des morceaux de cerveaux.

Attributs

- `_layer: int` → Couche sur laquelle sera mis le projectile .
- `game: obj` → Permet de récupérer tout ce que comporte le fichier game afin de les utiliser.
- `image: img` → Image du cerveau choisis au hasard dans la liste.
- `rect: rect` → Rectangle de l'image centré avec deux coordonnées.
- `direct: str` → Direction dans laquelle ira le projectile.
- `damage: int` → Nombre de dégâts que le projectile fera.
- `v_b: int` → Vitesse de déplacement du cerveau sur la grille.

Méthodes

- `def shoot_dir()` → La fonction commence par vérifier dans quelle direction le cerveau doit avancer, une fois vérifié, il change la position du rectangle soit en x soit en y ou même en x et y en fonction de la direction
- `def collide_with_wall()` → La fonction vérifie si le projectile rentre en contact avec un mûrs ou un rocher, si il le fait, alors il est supprimé de la scène.
- `def attack()` → La fonction vérifie si le projectile touche un joueur, si oui, il joue le son de contact, ensuite il soustrait au joueur les dégâts de l'attaque de ses pv et supprime le morceau de cerveau de la grille
- `def update()` → Appel la fonction `shoot_dir()` ensuite `collide_with_wall()` enfin elle appelle `attack()`

arrow.py

Arrow

La classe `Arrow` permettra de créer des projectiles qui sont des flèches.

Attributs

- `_layer: int` → Couche sur laquelle sera mis le projectile .
- `game: obj` → Permet de récupérer tout ce que comporte le fichier game afin de les utiliser.
- `image: img` → Image de la flèche.
- `rect: rect` → Rectangle de l'image centré avec deux coordonnées.
- `direct: str` → Direction dans laquelle ira le projectile.
- `damage: int` → Nombre de dégâts que le projectile fera.
- `v_f: int` → Vitesse de déplacement du projectile sur la grille.

Méthodes

- `def shoot_dir()` → La fonction commence par vérifier dans quelle direction le cerveau doit avancer, une fois vérifié, il change la position du rectangle soit en x soit en y en fonction de la direction
- `def collide_with_wall()` → La fonction vérifie si le projectile rentre en contact avec un murs ou un rocher, si il le fait, alors il est supprimé de la scène.
- `def attack()` → La fonction vérifie si le projectile touche un joueur, si oui, il joue le son de contact, ensuite il soustrait au joueur les dégâts de l'attaque de ses pv et supprime le morceau de cerveau de la grille
- `def update()` → Appel la fonction `shoot_dir()` ensuite `collide_with_wall()` enfin elle appelle `attack()`

game.py

Game

La classe `Game` permet de gérer toute la logique du jeu.

Ces attributs et méthodes permettent de gérer tout le jeu.

Le fichier gère l'affichage, les modifications du jeu, la mise à jour toutes les millisecondes.

Attributs

- `pg.init : init` → Initie le module pygame.
- `icon: img` → Image de l'icône du jeu.
- `running, playing: bool` → Booléen qui permettent de savoir si l'application tourne et si on est dans une partie.
- `self.UP_KEY, self.DOWN_KEY, self.START_KEY, self.BACK_KEY, self.PAUSE_KEY, self.LEFT_KEY, self.RIGHT_KEY: bool` → Booléen qui permettent de savoir si une touche est enfoncée pour gérer les menus.
- `DISPLAY_W, DISPLAY_H: int` → Nombres qui définissent la largeur et la hauteur d'une fenêtre.
- `display: surface` → Créer une surface de jeu de la largeur et la hauteur donnée précédemment.
- `window: surface` → Créer la fenêtre de la largeur et la hauteur donnée précédemment.
- `BLACK, WHITE: color` → Couleur Noir et Blanche.
- `main_menu: obj` → Menu principal.
- `options: obj` → Menu option.
- `credits: obj` → Menu crédit.
- `quit: obj` → Menu quitter.
- `death: obj` → Menu de mort du personnage.
- `pause: obj` → Menu pause.
- `confirm_quit: obj` → Menu pour confirmer si on quitte.
- `sound: obj` → Menu de gestion des sons.
- `control: obj` → Menu des contrôles.
- `win: obj` → Menu de victoire.

- `select_player: obj` → Menu du choix de personnage.
- `clock: obj` → Horloge de jeu.
- `room_maps: list` → Liste des pièces utiles pour créer les étages du donjon.
- `forced_maps: list` → Liste des pièces forcées d'être dans le donjon, il y en aura obligatoirement chaque pièces de cette liste dans un étage.
- `boss_maps: list` → listes des cartes de boss du jeu, à chaque étage seulement une de ces cartes sera dans l'étage.
- `stage_num: int` → Numéro de l'étage où l'on se trouve.
- `stage: int` → Numéro utilisé dans la création des étages.
- `font_name: str` → Nom de la police d'écriture.
- `m_change: bool` → Booléen qui permet de savoir si on change de carte.
- `boss_created: bool` → Booléen qui permet de savoir si un boss a été créé.
- `book_created: bool` → Booléen qui permet de savoir si un livre a été créé.
- `book_collected: bool` → Booléen qui permet de savoir si un livre a été créé.
- `fb_sound: Sound` → Initialise le son du tir de la boule de feu.
- `hit_sound: Sound` → Initialise le son de l'impact de la boule de feu.
- `c_sound: Sound` → Initialise le son du ramassage de la pièce.

Méthodes

- `def game_loop()` → La fonction s'occupe de gérer la boucle principale du jeu, lorsqu'on lance le jeu, on va faire un petit fondu et lancer la fonction `new()` qui va s'occuper de créer la grille sur laquelle les images et objets vont apparaître.
Tant que le jeu est lancé, une boucle va enchaîner des actions.
Durant cette boucle, on va commencer par vérifier si on appuie sur une touche, ensuite on va afficher les images sur la surface de jeu.
Pendant le jeu, si on appuie sur la touche escape, on fera apparaître le menu pause.
Pour finir, on vas mettre à jour tous les élément du jeu et annulé l'appui d'une touche si il y a eu
- `def quit_game()` → La fonction permet de quitter le jeu et l'application
- `def check_event()` → La fonction permet de vérifier si une touche a été

enfoncée.

Ensuite, si on arrête l'application en plein jeu, alors on ajoute une partie perdue à notre fichier `config.ini`

- `def reset_keys()` → La fonction permet de remettre toutes les touches à False.
- `def draw_text(text, size, x, y)` → La fonction permet de créer des textes à afficher sur les écrans de menu
- `def random_maps(length)` → La fonction permet de créer une liste contenant des cartes aléatoirement tiré d'une liste contenant toutes les cartes possible du jeu
- `def load_data()` → La fonction permet d'appeler `random_maps()` qui nous retournera les listes de cartes pour les étages.

Ensuite, on charge la liste de cartes de l'étage numéro 1 et la première carte de cette dernière.

- `def new(m_change, player)` → La fonction permet pour commencer d'initialiser les différents groupes de sprites, en prenant compte que `all_sprite` aura des couches, il fera la mise à jour des objets en commençant par le numéro 1 permettant au objet du niveau 10 d'apparaître en dernier et donc être au premier plan.

Ensuite, on vérifie la carte caractère par caractère, qui permettra d'associer des lettres, des chiffres et des symboles à des objets créés comme les personnages, murs ou même les sols.

- `def update()` → La fonction permet de remettre tous les objets dans la même liste, ensuite il appliquera la fonction de mise à jour de chacun de ces objets ajoutés.

Pour finir on mettra la position de la caméra à jour.

- `def draw()` → La fonction permet d'afficher toutes les images de tous les objets, mais aussi afficher le HUD du joueur et des boss, mais aussi les livres.

tilemap.py

Map

La classe `Map` permet de gérer les cartes, principalement décoder les fichiers texte.

Attributs

- `data: list` → L'attribut principal de la classe `Map` est une liste de caractère qui sera utilisé pour faire apparaître les bons objets au bon endroit sur la grille.
- `width: int` → Largeur de la carte.
- `height: int` → Hauteur de la carte.

Camera

La classe `Camera` permettra de gérer la caméra qui sera centrée sur le joueur.

Attributs

- `camera: Rect` → Rectangle de la caméra qui utilise `width` et `height` de la classe `Map`.
- `width: int` → Largeur de la carte.
- `height: int` → Hauteur de la carte.

Méthodes

- `def apply()` → Dessine l'image de l'objet sur la surface de jeu en fonction de ses coordonnées respectives
- `def update()` → La fonction permet de définir les limites que la caméra ne dépassera pas, pour la bloquée sur les murs extérieurs des cartes.

environnement.py

Le fichier `environnement.py` permettra de gérer tous les objets qui ont un rapport avec le décor du terrain de jeu .

Ces attributs et méthodes permettent de gérer les objets.

Wall

La classe `Wall` permettra de gérer tous les murs sur la carte. La classe hérite de la classe `Sprite`.

Attributs

- `pg.sprite.Sprite.__init__ : init` → Initie le module `Sprite`.
- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `image: img` → Image du mur.
- `rect: rect` → Rectangle de l'image.
- `x: int` → Coordonnée x de placement de l'objet sur la grille.
- `y: int` → Coordonnée y de placement de l'objet sur la grille.
- `rect.x: int` → Coordonnée x du rectangle de la case.
- `rect.y: int` → Coordonnée y du rectangle de la case.

Floor

La classe `Floor` permettra de gérer tous les sols sur la carte. La classe hérite de la classe `Sprite`.

Attributs

- `pg.sprite.Sprite.__init__ : init` → Initie le module `Sprite`.
- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `image: img` → Image du sol tirée au hasard parmi 3 images.
- `rect: rect` → Rectangle de l'image.
- `x: int` → Coordonnée x de placement de l'objet sur la grille.
- `y: int` → Coordonnée y de placement de l'objet sur la grille.
- `rect.x: int` → Coordonnée x du rectangle de la case.
- `rect.y: int` → Coordonnée y du rectangle de la case.

Door

La classe `Door` permet de gérer toutes les portes sur la carte. La classe hérite de la classe `Sprite`.

Attributs

- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `maps: list` → Liste des cartes de l'étage.
- `image: img` → Image de la porte de gauche ou de droite.
- `image_o: img` → Image de la porte de gauche ou de droite ouverte.
- `d: int` → Nombre pour déterminer si la porte est celle de gauche ou de droite.
- `game: obj` → Permet de récupérer l'objet `game` et accéder au attribut qu'il comporte.
- `rect: rect` → Rectangle de l'image.
- `x: int` → Coordonnée x de placement de l'objet sur la grille.
- `y: int` → Coordonnée y de placement de l'objet sur la grille.
- `rect.x: int` → Coordonnée x du rectangle de la case.
- `rect.y: int` → Coordonnée y du rectangle de la case.
- `lvl: int` → Numéro de la salle dans l'étage actuel.
- `door_open: bool` → Booléen qui permet de savoir si la porte est ouverte ou non.

Méthodes

- `def change_level()` → La fonction vérifie si il y a un contact avec le personnage joueur, si oui, on augmente le numéro de la salle de un, ensuite on change la carte, on sauvegarde le joueur, on passe le changement de carte à `True`.
La carte est changée on recharge tout à l'aide de la fonction `new()` du fichier `Game`, pour finir on referme les portes.
- `def update()` → La fonction vérifie si il reste des monstres dans la salle actuelle, si il n'en reste aucun, les portes s'ouvrent et change d'image et la fonction `change_level()` sera appelée

Ladder

La classe `Ladder` permettra de gérer les échelles de fin d'étage. La classe hérite de la classe `Sprite`.

Attributs

- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `image: img` → Image de l'échelle pas encore accessible.
- `image_o: img` → Image de l'échelle accessible.
- `game: obj` → Permet de récupérer l'objet game et accéder au attribut qu'il comporte.
- `rect: rect` → Rectangle de l'image.
- `x: int` → Coordonnée x de placement de l'objet sur la grille.
- `y: int` → Coordonnée y de placement de l'objet sur la grille.
- `rect.x: int` → Coordonnée x du rectangle de la case.
- `rect.y: int` → Coordonnée y du rectangle de la case.

Méthodes

- `def change_stage()` → La fonction vérifie si il y a un contact avec le personnage joueur, si oui, on augmente le numéro de l'étage de un, ensuite on change la liste de cartes correspondant à l'étage, on sauvegarde le joueur, et on change la carte
On remet le numéro de salle à 0.
Une fois la liste de cartes changée, on recharge tout à l'aide de la fonction `new()` du fichier Game.
- `def update()` → La fonction vérifie si il reste des monstres dans la salle actuelle, si il n'en reste aucun, l'échelle apparaît et change d'image enfin on appelle la fonction `change_level()`

Rock

La classe `Rock` permettra de gérer toutes les pierres sur la carte. La classe hérite de la classe `Sprite`.

Attributs

- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `image: img` → Image de la pierre.
- `rect: rect` → Rectangle de l'image.
- `x: int` → Coordonnée x de placement de l'objet sur la grille.
- `y: int` → Coordonnée y de placement de l'objet sur la grille.
- `rect.x: int` → Coordonnée x du rectangle de la case.
- `rect.y: int` → Coordonnée y du rectangle de la case.

Spike

La classe `Spike` permettra de gérer les piques dans les salles. La classe hérite de la classe `Sprite`.

Attributs

- `pg.sprite.Sprite.__init__ : init` → Initie le module `Sprite`.
- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `sprite: list` → Liste des images des piques.
- `current_sprite: int` → Nombre indiquant quelle image représente les piques.
- `image: int` → Image actuelle des piques.
- `rect: rect` → Rectangle de l'image.
- `x: int` → Coordonnée x de placement de l'objet sur la grille.
- `y: int` → Coordonnée y de placement de l'objet sur la grille.
- `rect.x: int` → Coordonnée x du rectangle de la case.
- `rect.y: int` → Coordonnée y du rectangle de la case.
- `cooldown: int` → Temps d'attente entre deux dégâts.
- `state: int` → Etat dans lequel se trouve les piques actuellement.

Méthodes

- `def attack()` → La fonction vérifie si il y a un contact avec le personnage joueur, si oui, on vérifie si le temps entre deux attaques est

atteint, si oui, on inflige les dégâts au personnage joueur.

- `def check_state()` → La fonction vérifie si l'état du pique est de 1, si oui il appelle la fonction `attack()`
- `def animator()` → La fonction gère l'animation des piques, si ils sont sortis à fond, alors l'état atteindra 1 ce qui permet aux piques d'infliger des dégâts avec la fonction `attack()`
- `def update()` → Appel de la fonction `check_state()` ensuite de la fonction `check_state()`.

Wall_Hole

La classe `Wall_Hole` permet de gérer tous les murs avec des trous qui tirent des projectiles. La classe hérite de la classe `Sprite`.

Attributs

- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `image: img` → Image du mur.
- `rect: rect` → Rectangle de l'image.
- `game: obj` → Permet de récupérer l'objet game et accéder au attribut qu'il comporte
- `x: int` → Coordonnée x de placement de l'objet sur la grille.
- `y: int` → Coordonnée y de placement de l'objet sur la grille.
- `rect.x: int` → Coordonnée x du rectangle de la case.
- `rect.y: int` → Coordonnée y du rectangle de la case.
- `cooldown: int` → Temps d'attente entre deux attaques.
- `projectile: list` → Liste des projectiles du jeu.

Méthodes

- `def shoot()` → La fonction va commencer par générer un nombre de millisecondes, celle-ci, permettent de savoir le temps d'attente entre deux attaques, ensuite il génère un projectile.
- `def update()` → La fonction appelle `shoot()`

Win_Ladder

La classe `Win_Ladder` permettra de gérer les échelles de fin d'étage.

Attributs

- `_layer: int` → Couche sur laquelle sera mis l'objet.
- `image: img` → Image de l'échelle pas encore accessible.
- `image_o: img` → Image de l'échelle accessible.
- `game: obj` → Permet de récupérer l'objet game et accéder au attribut qu'il comporte
- `rect: rect` → Rectangle de l'image.
- `x: int` → Coordonnée x de placement de l'objet sur la grille.
- `y: int` → Coordonnée y de placement de l'objet sur la grille.
- `rect.x: int` → Coordonnée x du rectangle de la case.
- `rect.y: int` → Coordonnée y du rectangle de la case.

Méthodes

- `def check_win()` → La fonction vérifie si il y a un contact avec le personnage joueur, si oui, on affiche le menu de victoire.
- `def update()` → La fonction vérifie si il reste des monstres dans la salle actuelle, si il n'en reste aucun, l'échelle apparaît et change d'image enfin la fonction `change_level()` sera appelée

menu.py

Le fichier `menu.py` permettra de gérer tous les menus du jeu

Ces attributs et méthodes permettent de gérer les différentes interactions et affichages des menu.

Menu

La classe `Menu` sera héritée dans tous les menus qui suivent .

Attributs

- `move_sound: sound` → Sons de déplacement dans les menus.
- `select_sound: sound` → Sons de sélections du menu.
- `move_sound: sound` → Met à jour le volume du son de déplacement dans les menu.
- `select_sound.set_volume: sound` → Met à jour le volume du son de sélections du menu.
- `fairy_image_unlocked: img` → Image de la fée débloquée.
- `fairy_image: img` → Image de la fée non débloquée.
- `wizard_image: img` → Image du mage.
- `coin_image: img` → Image de la pièce comme curseur.
- `game: Game` → Permet de récupérer les attributs de game.
- `mid_w, mid_h: int` → Permet de savoir le milieu de la hauteur et de la largeur de la fenêtre de l'application.
- `run_display: bool` → Booléen qui permet de savoir si le menu est affiché.
- `cursor_rect: Rect` → Permet d'avoir le rectangle de l'image de curseur.
- `offset: int` → Nombre qui définit le décalage d'un élément.

Les attributs des menus enfants vont être des positions pour l'affichage des noms des autres menus, un état pour savoir sur quel menu on va se trouver lorsque l'on appuie pour aller dans ce dernier et enfin avoir le milieu du rectangle du curseur.

Méthodes

- `def draw_cursor()` → La fonction permet d'afficher le curseur sur l'écran.

- `def blit_scren()` → La fonction permet de dessiner le menu sur la surface de la fenêtre, elle vérifie aussi si on est dans le menu de sélection et changera l'image de la fée si elle est débloquée.
- `def blit_background()` → La fonction permet de dessiner le fond du menu sur la surface de la fenêtre.
- `def fade(width, height, time)` → La fonction permet de créer un effet de fondu, celui-ci sera créé lorsque l'on change de menu, ou encore lorsque l'on lance le jeu.

Le principe est qu'on va remplir la surface avec une couleur noire qui de base sera invisible et on va augmenter sa transparence ce qui rendra le noir opaque.

MainMenu

La classe `MainMenu` est le menu principal.

Méthodes

- `def display_menu()` → La fonction jouera l'effet de fondu, ensuite affichera les noms des différents sous-menus.
La fonction, ensuite, vérifiera les entrées du joueur et affichera le curseur sur la surface.
Pour finir elle mettra à jour l'écran.
- `def move_cursor()` → La fonction permet de déplacer le curseur à côté du nom du sous-menu ou on veut se rendre.
- `def check_input()` → La fonction permet de vérifier sur quel menu on se trouve et lorsque l'on appuie sur la touche pour changer de menu, effectué le changement de menu.

OptionsMenu

La classe `OptionsMenu` est le menu d'options.

Méthodes

- `def display_menu()` → La fonction jouera l'effet de fondu, ensuite affichera les noms des différents sous-menus.
La fonction, ensuite vérifiera les entrées du joueur et affichera le curseur sur la surface.
Pour finir elle mettra à jour l'écran.
- `def check_input()` → La fonction permet de vérifier les entrées du joueur et d'effectuer une action selon ces dernières, elle permet aussi de dire sur quel menu on se trouve et lorsque l'on appuie sur la touche pour changer de menu, effectuée le changement de menu.
La fonction permet aussi de déplacer le curseur à côté du nom du sous-menu ou on veut se rendre.
Si on quitte le menu, la fonction vérifie si on est en jeu ou pas, si on y est, on revient dans le menu pause sinon on revient au menu principale

CreditsMenu

La classe `CreditsMenu` est le menu d'affichage des crédits.

Méthodes

- `def display_menu()` → La fonction jouera l'effet de fondu, ensuite affichera le texte de crédit.
La fonction, ensuite, vérifiera les entrées du joueur, s'il appuie sur la touche pour quitter, il reviendra un menu en arrière.
Pour finir elle mettra à jour l'écran.

ControlMenu

La classe `ControlMenu` est le menu d'affichage des touches.

Méthodes

- `def display_menu()` → La fonction jouera l'effet de fondu, ensuite affichera l'image avec les crédits.
La fonction ensuite vérifiera les entrées du joueur, s'il appuie sur la touche pour quitter, il reviendra un menu en arrière.
Pour finir, elle mettra à jour l'écran.

Quit

La classe `Quit` permet de fermer l'application.

Méthodes

- `def quit()` → La fonction fermera l'application

Death_Menu

La classe `Death_Menu` est le menu d'affichage lorsque le joueur meurt.

Méthodes

- `def display_menu()` → La fonction jouera l'effet de fondu, ensuite affichera le texte pour dire que le personnage est mort.
La fonction, ensuite, vérifiera les entrées du joueur, s'il appuie sur la touche pour confirmer, alors il retournera au menu et on remettra toutes les variables pour que le joueur puisse relancer une nouvelle partie, aussi lorsqu'il fait ça dans le fichier `Config.ini`, on ajoute une partie perdue.
Pour finir, elle mettra à jour l'écran.

Pause_Menu

La classe `Pause_Menu` est le menu de pause du jeu.

Méthodes

- `def display_menu()` → La fonction jouera l'effet de fondu, ensuite affichera les noms des différents sous-menus.
La fonction, ensuite, vérifiera les entrées du joueur et affichera le curseur sur la surface.
Pour finir elle mettra à jour l'écran.
- `def check_input()` → La fonction permet de vérifier les entrées du joueur et d'effectuer une action selon ces dernières, elle permet aussi de dire sur quel menu on se trouve et lorsque l'on appuie sur la touche pour changer de menu, effectué le changement de menu.
La fonction permet aussi de déplacer le curseur à côté du nom du sous-menu ou on veut se rendre, si on est sur le sous-menu "Menu principale" alors on sera envoyé vers le menu pour confirmer que l'on quitte.
Si on quitte le menu de pause, la fonction nous remettra directement dans le jeu.

Confirm_Quit

La classe `Confirm_Quit` est le menu pour confirmer que l'on quitte la partie.

Méthodes

- `def display_menu()` → La fonction jouera l'effet de fondu, ensuite affichera les noms des différents choix.
La fonction, ensuite, vérifiera les entrées du joueur et affichera le curseur sur la surface.
Pour finir, elle mettra à jour l'écran.
- `def check_input()` → La fonction permet de vérifier les entrées du joueur et vérifié si on quitte la partie ou non
La fonction permet aussi de déplacer le curseur à côté du nom du choix.
Si on choisi de ne pas quitter la partie, on est redirigé sur le menu pause.
Par contre si on quitte alors on écrit dans le fichier de configuration une partie perdue de plus ensuite, il retournera au menu et on remettra toutes

les variables pour que le joueur puisse relancer une nouvelle partie.

Sound_Menu

La classe `Sound_Menu` est le menu de paramètre qui gère les sons du jeu.

Méthodes

- `def sound_modifiser()` → La fonction permet, en récupérant le son en paramètre d'effectuer l'opération elle aussi envoyée en paramètre, si on ajoute, alors on augmente la valeur du son de un, par contre si on soustrait, on baisse la valeur du son de un.
- `def display_menu()` → La fonction jouera l'effet de fondu, ensuite affichera les noms des différents sons modifiables.
La fonction, ensuite, vérifiera les entrées du joueur et affichera le curseur sur la surface.
Pour finir, elle mettra à jour l'écran.
- `def move_cursor()` → La fonction permet de déplacer le curseur à côté du son que l'on veut modifier.
- `def check_input()` → La fonction permet de vérifier si une action est effectuée par le joueur et selon sur quel son on se trouve, si on appuie sur la flèche de gauche, on baisse le son de un et si on clique sur la flèche de droite on l'augmente.
Enfin, pour finir, on met à jour le volume de tous les sons et on sauvegarde les différentes valeurs dans le fichier de configuration.

Win_Menu

La classe `Win_Menu` est le menu victoire du jeu.

Méthodes

- `def display_menu()` → La fonction jouera l'effet de fondu, ensuite affichera le texte de victoire de la partie, et si le joueur a gagné 5 fois la partie avec le mage et que la fée n'est pas débloquée, alors il débloquent cette dernière.
La fonction, ensuite, vérifiera les entrées du joueur et si le joueur quitte ou appuie sur le bouton de sélection, alors le menu disparaîtra et il retournera au menu.
Elle remettra toutes les variables pour que le joueur puisse relancer une

nouvelle partie.

Pour finir, elle mettra à jour l'écran.

Select_Player_Menu

La classe `Select_Player_Menu` est le menu de choix des personnages.

Méthodes

- `def display_menu()` → La fonction jouera l'effet de fondu, ensuite affichera les noms des différents personnages et leurs images, aussi elle vérifiera si la fée est débloquée ou non
La fonction, ensuite, vérifiera les entrées du joueur et affichera le curseur sur la surface.
- `def play_game()` → La fonction permet de lancer une partie avec le personnage choisis, ce dernier variera selon le choix du joueur, de plus la fonction ajoutera une partie jouée dans le fichier de configuration du jeu.
- `def check_input()` → La fonction permet de vérifier si une action est effectuée par le joueur et selon sur quel personnage on se trouve, si on appuie sur la flèche de gauche ou de droite on choisit notre personnage, mais seulement si la fée est débloquée .
Enfin pour finir lorsque le personnage est choisi et que l'on appuie sur la touche d'action, alors on démarre la partie à l'aide de `play_game()`.

Problèmes rencontrés

Le premier gros souci a été la planification, mais pour finir, je me suis servi de Trello (<https://trello.com>) vraiment pratique pour organiser mon travail à l'aide d'un tableau qui permet de définir les objectifs à atteindre semaine en semaines.

Un autre souci a été de me brider, car j'ai voulu faire beaucoup trop de choses pour le temps et surtout pour ce que j'étais capable à ce moment-là.

Maintenant, les soucis de développement commencent, j'ai dû trouver une sorte de chemin à suivre pour commencer, comprendre comment faire déplacer un joueur avec des vecteurs, grâce à un tutoriel, j'ai très vite compris.

Ensuite la gestion d'une carte, comment faire pour créer une carte qui me permet d'afficher des murs, sol, monstres, portes, etc.

Dans cette même playlist, la personne apprend à gérer des fichiers séquentiels pour créer des cartes, cette gestion a pu me permettre de facilement faire des cartes.

Tout ça réglé, je me suis dit pourquoi ne pas créer un menu, le souci c'est que je savais comment afficher dans la fenêtre, mais comment faire pour pouvoir avoir une "interaction" avec ce dernier, donc j'ai fait mes recherches et trouvé des personnes qui expliquaient que faire un menu est quelque chose de simple, il suffit de jouer avec une "state machine" une machine à état en français.

En pressant certaines touches, on pouvait changer l'état du menu et donc changer le menu ou on va se rendre, une fois la logique, il suffisait d'afficher le texte correspondant et le souci était réglé.

Comme je voulais pousser le jeu encore plus loin, je me suis dit pourquoi ne pas ajouter un système de gestion des volumes de sons, un petit ajout, mais un ajout que j'ai trouvé assez compliqué.

Pour sauvegarder les données modifiées, j'ai dû apprendre à utiliser les fichiers avec extension ".ini" mais surtout apprendre à utiliser le module "configparser" de python.

Enfin, un des problèmes a été ce rapport, car j'ai beaucoup de mal à expliquer en français ce que je mets en code, je sais l'expliquer, mais trouver les bons mots à placer, c'est quelque chose d'assez compliqué pour moi, mais je suis assez fier de moi d'avoir fait le rapport complet de l'application en essayant d'être le plus

précis et lisible.

Conclusion

Je suis très fier de mon travail, je me suis fixé la barre assez haut dès le début pour me surpasser, apprendre beaucoup de choses pour le futur, ce qui est quand même important.

Ensuite comme j'ai revus mes objectifs un peu plus bas, ils sont atteints, effectivement je n'ai pas 3 personnages joueurs, mais 2 ce qui me convient déjà beaucoup, le soucis à surtout été de trouvé un but pour ce troisième personnage, car j'avais déjà un personnage à distance qui était correcte, mais je me suis dit pourquoi ne pas en créer un qui vole, mais je n'ai pas trouvé d'idée pour le suivant, si ce n'est un qui invoquait des monstres pour se battre à sa place.

Je n'ai qu'un seul regret qui est que je n'aie pas réussi à trouver un moyen simple de suivre la génération de donjon prévue, comme cette dernière était fortement compliquer.

Enfin comme j'ai dit plus haut, le but de me fixer la barre assez haut pour moi-même était important, les challenges, c'est quand même plus fun.

En conclusion, je dirais que le développement de mon jeu m'a appris beaucoup de choses : python et pygame, mais aussi sur des modules comme configparser et enfin l'utilisation de Trello pour planifier un projet avant de le commencer me semble important à mentionner, il m'a permis d'avoir une gestion correcte d'un projet et ne pas commencer à me disperser à gauche ou à droite et terminer ce sur quoi je travail avant de passer à autre chose et donc me perdre.

Bibliographie

Ressources Web

Code

Christian Duenas. (2020, 23 juillet). **Menu System Tutorial**. Consulté le 21 mars 2020, à l'adresse

<https://www.youtube.com/playlist?list=PLVFWKkB2K-TmYDRlFFm-RkhZNNj0waXw5>

Code, C. (2020, mai 22). **Python / Pygame Tutorial : Creating a basic bullet shooting mechanic**. Consulté le 23 janvier 2021, à l'adresse

https://www.youtube.com/watch?v=JmpA7TU_0Ms&feature=youtu.be

Code, C. (2020, août 11). **Pygame/Python tutorial : Creating Zelda-style hearts**. Consulté le 3 avril 2021, à l'adresse

<https://www.youtube.com/watch?v=WLYEsgYkEvY&feature=youtu.be>

Code, C. (2020, octobre 2). **Creating a health bar in pygame [Dark Souls style]**. Consulté le 6 mars 2021, à l'adresse

<https://www.youtube.com/watch?v=pUEZbUAMZYA&feature=youtu.be>

configparser — **Configuration file parser** — Python 3.9.5 documentation. (2020). Consulté le 19 mai 2021, à l'adresse

<https://docs.python.org/3/library/configparser.html>

freeCodeCamp.org. (2019, 22 octobre). **Pygame Tutorial for Beginners - Python Game Development Course**. Consulté le 21 octobre 2020, à l'adresse

<https://www.youtube.com/watch?v=FfWpgLFMI7w&feature=youtu.be>

KidsCanCode. (2017, 17 janvier). **Pygame Tutorial # 3 : Tile-based Game**. Consulté le 6 janvier 2021, à l'adresse

<https://www.youtube.com/playlist?list=PLsk-HSGFjnaGOq7ybM8Lgkh5EMxUWPm2i>

pygame.org. (s. d.). **Pygame documentation**. Consulté le 27 Novembre 2020, à l'adresse

<https://www.pygame.org/docs/>

Style du jeu

0x72. (2018). **16x16 DungeonTileset II by 0x72**. Consulté le 1 novembre 2020, à l'adresse <https://0x72.itch.io/dungeontileset-ii>

Pixilart - Share & Create Art Online. (2013). Consulté le 14 mars 2021, à l'adresse <https://www.pixilart.com>

Superdark. (2019). **16x16 Fantasy RPG Characters by Superdark**. Consulté le 2 juin 2021, à l'adresse <https://superdark.itch.io/16x16-free-npc-pack>

Superdark. (2019). **16x16 Enchanted Forest Characters by Superdark**. Consulté le 2 juin 2021, à l'adresse <https://superdark.itch.io/enchanted-forest-characters>

jsfxr. (2011). Consulté le 12 avril 2021, à l'adresse <https://sfxr.me>

Trello

Julien Vanherf. (2021, 1 janvier). **Trello**. Consulté à l'adresse <https://trello.com/invite/accept-board>