Table des matières

[1 Présentation 3](#_Toc124344774)

[1.1 Description générale 3](#_Toc124344775)

[1.2 Règles du jeu 3](#_Toc124344776)

[1.3 Cinématique des écrans 4](#_Toc124344777)

[2 Conception – Diagramme de classe 5](#_Toc124344778)

[2.1 Présentation générale 5](#_Toc124344779)

[2.2 Présentation détaillée Game1 7](#_Toc124344780)

[2.3 Présentation détaillé Menu 9](#_Toc124344781)

[2.4 Présentation détaillé ChoixNiveau 10](#_Toc124344782)

[2.5 Présentation détaillé Regle 10](#_Toc124344783)

[2.6 Présentation détaillé Desert 11](#_Toc124344784)

[2.7 Présentation détaillé Snow 11](#_Toc124344785)

[2.8 Présentation détaillé GameOver 11](#_Toc124344786)

[2.9 Présentation détaillé Win 11](#_Toc124344787)

[2.10 Présentation détaillé Pingouin 11](#_Toc124344788)

[2.11 Présentation détaillé Snowball 11](#_Toc124344789)

[2.12 Présentation détaillé MonstreVolant 11](#_Toc124344790)

[2.13 Présentation détaillé MonstreRampant 11](#_Toc124344791)

[2.14 Présentation détaillé Trap 11](#_Toc124344792)

[2.15 Présentation détaillé Recompenses 11](#_Toc124344793)

[2.16 Présentation détaillé Collision 11](#_Toc124344794)

[2.17 Présentation détaillé Camera 11](#_Toc124344795)

[2.18 Présentation détaillé Chrono 11](#_Toc124344796)

[2.19 Présentation détaillé Life 11](#_Toc124344797)

[3 Conception graphique 11](#_Toc124344798)

[4 Partie Algorithmie – Intelligence artificielle 11](#_Toc124344799)

[4.1 Explications 12](#_Toc124344800)

[4.2 Extrait de code 12](#_Toc124344801)

[5 Cahier de recettes 13](#_Toc124344802)

[5.1 Tests de validation 13](#_Toc124344803)

[5.2 Tests de performance 13](#_Toc124344804)

Attention : toute vos impressions écrans doivent être lisibles !

# Présentation

## Description générale

## 

Man-chaud est un jeu de plateforme dans lequel l’utilisateur incarne un pingouin. L’objectif est de récupérer les différents morceaux de portail disséminer dans la map afin qu’il puisse revenir à sa banquise. Chaque niveau du jeu se trouve sur une map différente.

*Genre et principe du jeu, but du joueur.*

*Impressions écran, avec des phrases introductives et explicatives.*

## Règles du jeu

Durant la partie, le joueur est susceptible de rencontrer différent prédateur. Le pingouin pourra se défendre mais s’il se fait toucher il perdra une de ses vies. Lorsque le pingouin perd ses trois vies ou tombe dans le vide, il meurt et la partie se termine.



Barre de vie

Contrôle :

* Afficher le menu : Tab

Les déplacements :

Le pingouin peut marcher, a droite ou a gauche avec les flèches respective.

Pour sauter, touche espace. Attention, le pingouin ne peut sauter que s’il a un contact avec le sol.

Pour glisser, la flèche du bas. Ainsi, le pingouin se déplacera plus vite.

Le pingouin peut attaquer ses ennemis en leur sautant dessus ou avec la touche « entrer », lui faisant lancer une boule de neige devant lui.

Les éléments récoltables :

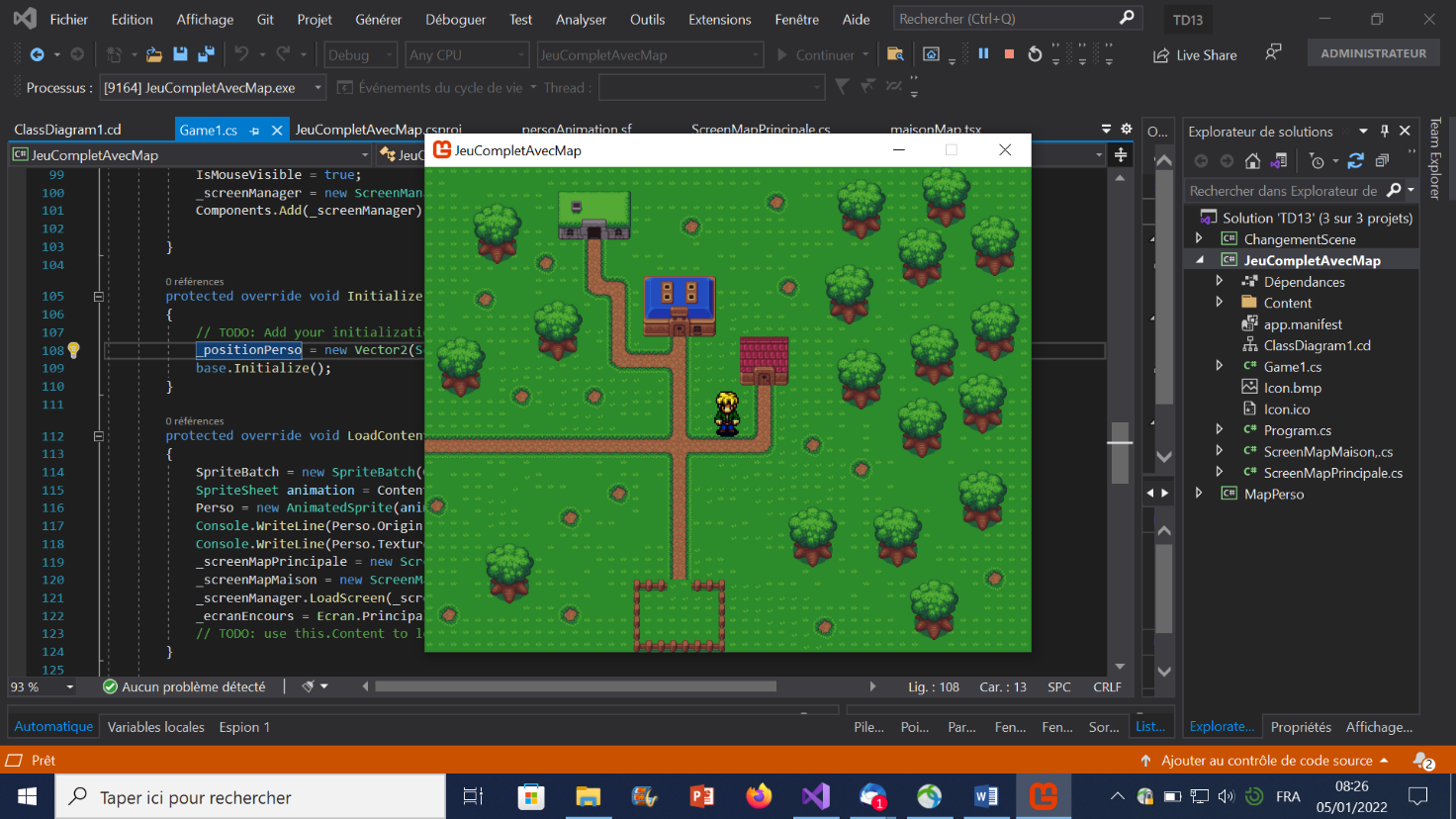
* Les pièces : Attention, si les pièces peuvent redonner de la vie lorsque le pingouin en a perdu, celles-ci peuvent également le faire ralentir par leur poids !
* Les morceaux de portail : Afin de remporter le niveau, il est nécessaire de tous les récolter. Leur nombre dépend du niveau, il est affiché sous la barre de vie, en haut à gauche de l’écran.

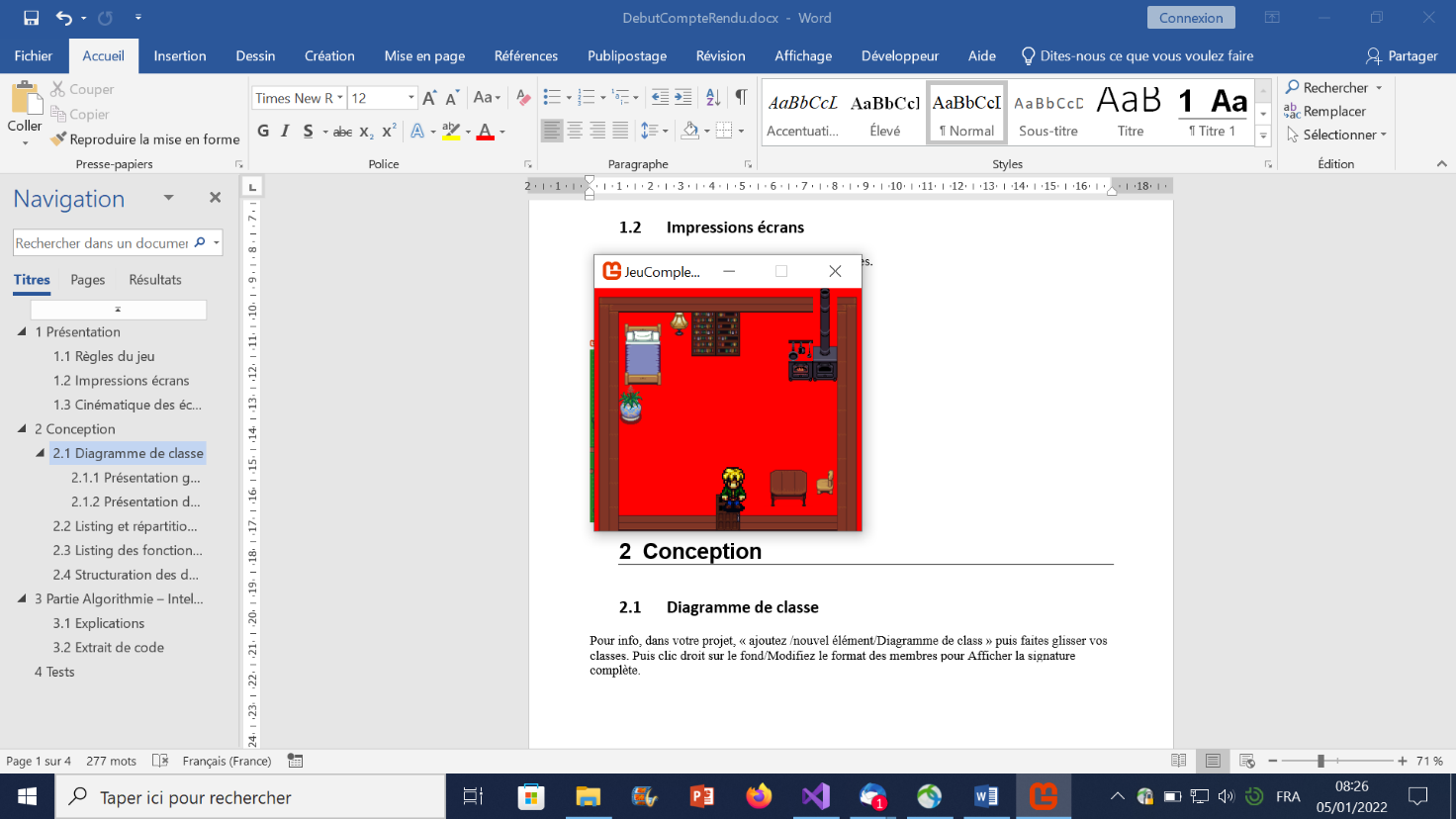
Codes triche :

* Pour les activer : F2
* Pour rendre toute sa vie au pingouin : V
* Pour voler et traverser les murs au-dessus du pingouin : F
* Pour avoir tous les morceau de portail : C
* Pour se téléporter au point de départ : Inser
* Pour se téléporter a la fin de la map ou le portail de fin s’ouvre : Fin
* Pour remettre le pingouin a sa vitesse initiale : P

Règle du jeu détaillée ainsi que la description des touches ou autre nécessaire pour jouer (code triche par exemple …)

## Cinématique des écrans





Arrivé à la porte bleue



Clic sur Notes de pingouin



Clic sur charger un niveau

# Conception – Diagramme de classe

## Présentation générale

*Faites une impression écran générale (sans le détail des classes ) accompagnée d’explications synthétiques pour chaque classe. N’héitez pas à regouper et découper si schéma trop grand.*

*Aide : sur votre projet, « Ajoutez un nouvel élément / Diagramme de classes » puis faites glisser vos classes depuis l’explorateur de solution dans la fenêtre*

*Justifiez vos choix de conception : expliquez et mettez en évidence si vous avez fait des classes pour améliorer, factoriser votre code .(Ex : classe Sprite ou Personnage, ….)*

*Ex :*

*Game1 est le point d’entrée : elle contrôle tous les enchainements des différents écrans ainsi que les pauses ou sortie du jeu.*

****

****

## Présentation détaillée Game1



Game1 : instancie les 7 GameScreen : ChoixNiveau, Desert, GameOver, Menu, Regle, Snow et Win.

Game1 charge en tout 1er Menu, puis selon le clic l’utilisateur les booléen changent permettant à Game1 de charger Regle, ChoixNiveau, ou Desert. Si Game1 charge ChoixNiveau, il peut alors de nouveau charger Menu, ou charger Desert ou Snow.

Elle contient 28 champs :

* \_choixNiveau : c’est un objet de classe ChoixNiveau qui permet à Game1 de charger le GameScreen ChoixNiveau.
* \_desert : c’est un objet de classe Desert qui permet à Game1 de charger le GameScreen Desert.
* \_gameOver : c’est un objet de classe GameOver qui permet à Game1 de charger le GameScreen GameOver.
* \_graphics : c’est un objet de classe GraphicsDeviceManager, il permet de stocker et de modifier les informations de la fenêtre de jeu.
* \_menu : c’est un objet de classe Menu qui permet a Game1 de charger Menu.
* \_regle : c’est un objet de classe Regle qui permet à Game1 de charger le GameScreen Regle.
* \_screenManager : c’est un objet de classe ScreenManager qui sert à gérer le chargement de scène/GameScreen
* \_snow : c’est un objet de classe Snow qui permet à Game1 de charger le GameScreen Regle.
* \_win : c’est un objet de classe Win qui permet à Game1 de charger le GameScreen Regle.
* Attaquer : c’est un objet de classe Keys, cette variable permet de stocker la touche d’attaque
* clicMenu : c’est un objet de classe bool
* clicWin : c’est un objet de classe bool
* dernierePositionPingouin : c’est un objet de classe Vector2 qui sert a garder en mémoire dans game1 la position du pingouin durant la parti
* droite : c’est un objet de classe Keys, cette variable permet de stocker la touche de déplacement vers la droite
* gauche : c’est un objet de classe Keys, cette variable permet de stocker la touche de déplacement vers la gauche
* glisser : c’est un objet de classe Keys, cette variable permet de stocker la touche de glissade
* sauter : c’est un objet de classe Keys, permet de stocker la touche de saut
* goDead : c’est un objet de classe bool qui permet de gérer si Game1 doit charger le gameScreen GameOver.
* goDesert : c’est un objet de classe bool qui permet de gérer si Game1 doit charger les GameScreen Desert.
* goRules : c’est un objet de classe bool qui permet de gérer si Game1 doit charger les GameScreen Rules.
* goSnow : c’est un objet de classe bool qui permet de gérer si Game1 doit charger les GameScreen Snow.
* goStop : c’est un objet de classe bool qui permet de gérer si Game1 doit fermer la fenêtre.
* HAUTEUR\_FENETRE : c’est un objet de classe int qui permet de définir la hauteur de la fenêtre.
* LARGEUR\_FENETRE : c’est un objet de classe int qui permet de définir la largeur de la fenêtre.
* nivActu : c’est un objet de classe int qui permet de savoir la parti ou en est l’utilisateur.
* pause : c’est un objet de classe bool qui permet de gérer si la partie en cour est en pause.
* police : c’est un objet de classe SpriteFont qui permet de globaliser une police pour toute les classe.
* reprendre : c’est un objet de classe bool qui permet de gérer si une partie est a reprendre.

## Présentation détaillé Menu



Menu : GameScreen faisant le lien visuelle entre Game1 et les classe GameScreen :ChoixNiveau, Regle, et Desert ou Snow selon la partie actuelle lancer.

Elle contient 15 champs :

* \_mouseState : c’est un objet de classe MouseState qui permet de savoir l’état de la souris.
* \_myGame : c’est un objet de classe Game1 qui permet d’hériter des fonctions de Game1.
* \_pingouin : c’est un objet de classe Pingouin qui permet d’utiliser la classe Pingouin.
* \_pingouinG : c’est un objet de classe Pingouin qui permet d’utiliser la classe Pingouin.
* \_positionFond : c’est un objet de classe Vector2 qui permet de donner une position au fond.
* \_textureFond : c’est un objet de classe Texture2D qui permet de charger une texture.
* clicChoixNiv : c’est un objet de classe bool qui permet de renvoyer a Game1 si l’utilisateur clic sur Charger un niveau.
* jouer : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* niv : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* positionJouer : c’est un objet de classe Vector2 qui permet de donner une position au string jouer.
* positionNiv : c’est un objet de classe Vector2 qui permet de donner une position au string niv.
* positionQuitter : c’est un objet de classe Vector2 qui permet de donner une position au string quitter.
* positionRegle : c’est un objet de classe Vector2 qui permet de donner une position au string regle.
* quitter : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* regle : c’est un objet de classe string qui permet d’initialiser le texte a afficher.

## Présentation détaillé ChoixNiveau

ChoixNiveau : : GameScreen faisant le lien visuelle entre Game1 et les classe GameScreen : Menu, Desert et Snow.

Elle contient 9 champs :

* \_messMenu : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_mousState : c’est un objet de classe MouseState qui permet de prendre l’action de la souris sur la fenetre de jeu.
* \_\_myGame : c’est un objet de classe Game1 qui permet d’heriter des fontion de Game1.
* \_niv1 : c’est un objet de classe String qui permet d’initialiser le texte à afficher.
* \_niv2 : c’est un objet de classe String qui permet d’initialiser le texte à afficher.
* \_positionMenu : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_messMenu.
* \_positionNiv1 : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_niv1.
* \_positionNiv2 : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_niv2.
* \_textureBackground : c’est un objet de classe Texture2D qui permet de mettre un fond lors de l’affichage.

## Présentation détaillé Regle

Une image contenant texte

Description générée automatiquement

Regle : GameScreen faisant le lien visuel entre Game1 et la classe GameScreen Menu, sert a présenter les règle du jeu.

Elle contient champs :

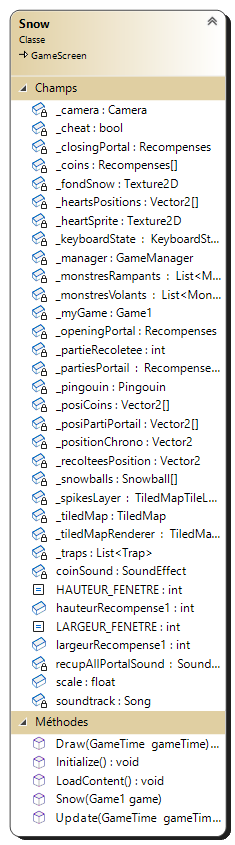
* \_attaquer : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_avancer : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_glisser : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_menuTXT : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_mouseState : c’est un objet de classe MouseState qui permet de savoir l’etat de la souris.
* \_myGame : c’est un objet de classe Game1 qui permet d’heriter des fontion de Game1.
* \_piece : c’est un objet de classe
* \_pingouinAvancer : c’est un objet de classe Pingouin qui permet d’afficher un pingouin.
* \_pingouinGlisser : c’est un objet de classe Pingouin qui permet d’afficher un pingouin.
* \_pingouinSauter : c’est un objet de classe Pingouin qui permet d’afficher un pingouin.
* \_portail : c’est un objet de classe Recompences qui permet d’afficher un portail.
* \_positionAttaquer : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_attaquer.
* \_positionAvancer : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_avancer.
* \_positionGlisser : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_glisser.
* \_positionMenu : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_menuTXT.
* \_positionMonaieCas1 : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_texteMonaieCas1.
* positionMonaieCas2 : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_texteMonaieCas2.
* \_positionRelever : c’est un objet de classe Vector2 qui permet d’initialiser la position.
* \_positionSauter : c’est un objet de classe Vector2 qui permet d’initialiser la position.
* \_positiontxtControle : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_texteControle.
* \_positiontxtEnnemi : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_texteEnnemi.
* \_positiontxtIntroEnnemi : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_texteIntroEnnemi.
* \_positiontxtMonaie : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_texteMonaie.
* \_positiontxtTeleport : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_texteTeleporteur.
* \_positiontxtTeleportDesc : c’est un objet de classe Vector2 qui permet d’initialiser la position de \_texteTeleporteurDesc.
* \_relever : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_sauter : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_texteControle : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_texteEnnemi : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_texteIntroEnnemi : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_texteMonaie : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_texteMonaieCas1 : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_texteMonaisCas2 : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_texteTeleporteur : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* \_texteTeleporteurDesc : c’est un objet de classe string qui permet d’initialiser le texte a afficher.
* policePetite : c’est un objet de classe SpriteFont qui permet d’utiliser une autre police que celle initialiser dans game1.

## Présentation détaillé Desert

Desert : C’est dans ce GameScreen que se déroule le niveau 1.

* \_camera : c’est un objet de classe Camera, il stocke les informations de la caméra.
* \_cheat : c’est un objet de classe bool qui sert a savoir si les cheat sont ou non activés.
* \_closingPortal : c’est un objet de classe Recompenses qui permet de faire apparaitre un portail.
* \_coins : c’est un tableau de classe Recompenses qui sert à afficher les différentes pieces dans la map.
* \_fondDesert : c’est un objet de classe Texture2D qui permet de charger le fond.
* \_hauteurPingouin : c’est un objet de classe int qui permet de connaitre la hauteur du pingouin.
* \_heartsPositions : c’est un tableau de classe Vector2 qui permet d’initialiser la position des cœurs en même temps que la caméra.
* \_keyboardState : c’est un objet de classe KeyboardState qui permet de prendre en compte le clavier.
* \_largeurPingouin : c’est un objet de classe int qui permet de connaitre la largeur du pingouin.
* \_manager : c’est un objet de classe GameManager, qui permet de piloter les différentes étapes lors du niveau.
* \_monstreRampants : c’est une liste de classe MonstreRampant qui permet de gérer tous les monstres de classe MonstreRampant de la map.
* \_monstresVolant : c’est une liste de classe MonstreVolant qui permet de gérer tous les monstres de classe MonstreVolant de la map.
* \_myGame : c’est un objet de classe Game1 qui permet d’hériter des fonctions dans Game1.
* \_openingPortal : c’est un objet de classe Recompenses qui permet de faire apparaitre un portail.
* \_partiesPortail : c’est un tableau de classe Recompenses qui permet de gérer tous les morceaux de portail de la map.
* \_partiesRecoltees : c’est un objet de classe int qui permet de connaitre le nombre de morceau de portail récolté.
* \_pingouin : c’est un objet de classe Pingouin qui permet d’utiliser un pingouin.
* \_posiCoins : c’est un tableau de classe Vector2 qui permet d’initialiser la position des pièces sur la map.
* \_posiPartiPortail : c’est un tableau de classe Vector2 qui permet d’initialiser la position des diffèrent morceau de portail sur la map
* \_positionChrono : c’est un objet de classe Vector2 qui permet d’initialiser la position du chrono et la mettre à jour en même temps que la caméra.
* \_recolteesPosition : c’est un objet de classe Vector2 qui permet d’initialiser la position des morceau récoltées et de la mettre à jour avec la caméra.
* \_rFox : c’est un objet de classe Rectangle, il s’agit de la hitbox du renard.
* \_rKillingFow : c’est un objet de classe Rectangle, il s’agit du rectangle grâce auquel le renard peut mourir.
* \_rRecompense : c’est un objet de classe Rectangle, il s’agit de la hitbox des pièces et des portails.
* \_rTrap : c’est un objet de classe Rectangle, il s’agit de la hitbox du piège.
* \_snowballs : c’est un tableau de classe Snowball qui permet de gérer les boules de neige.
* \_tiledMap : c’est un objet de classe TiledMap, il s’agit de la map.
* \_tiledMapRenderer : c’est un objet de classe TiledMapRenderer, il s’agit du rendu visuel de la map.
* \_traps : c’est une liste de classe Trap qui permet de gérer les pièges sur la map.
* \_coinSound : c’est un objet de classe SoundEffect qui permet de charger un son et de le jouer lorsque l’utilisateur récupère une pièce.
* HAUTEUR\_FENETRE : c’est un objet de classe int qui permet de connaitre la hauteur de la fenêtre.
* hauteurRecompense1 : c’est un objet de classe int qui permet de connaitre la hauteur des pièces.
* LARGEUR\_FENETRE : c’est un objet de classe int qui permet de connaitre la largeur de la fenêtre.
* largeurRecompense1 : c’est un objet de classe int qui permet de connaitre la largeur des pièces.
* recupAllPortalSound : c’est un objet de classe SoundEffect qui permet de charger un son et de le jouer lorsque l’utilisateur récupère tous les morceau de portail.
* scale : c’est une variable de type float, cette variable stocke le zoom de la caméra.
* soundtrack : c’est un objet de classe Song qui permet de diffuser une musique en arrière-plan durant la partie.

## Présentation détaillé Snow



La classe Snow gère le deuxième niveau du jeu. Elle est composée de 33 champs.

* \_camera : c’est un objet de classe Camera, il stocke les informations de la caméra.
* \_cheat : c’est un objet de classe bool qui sert a savoir si les cheat sont ou non activés.
* \_closingPortal : c’est un objet de classe Recompenses qui permet de faire apparaitre un portail.
* \_coins : c’est un tableau de classe Recompenses qui sert à afficher les différentes pieces dans la map.
* \_fondSnow : c’est un objet de classe Texture2D qui permet de charger le fond.
* \_heartsPositions : c’est un tableau de classe Vector2 qui permet d’initialiser la position des cœurs en même temps que la caméra.
* \_keyboardState : c’est un objet de classe KeyboardState qui permet de détecter les entrées liées au clavier.
* \_manager : c’est un objet de classe GameManager, qui permet de piloter les différentes étapes lors du niveau.
* \_monstreRampants : c’est une liste de classe MonstreRampant qui permet de gérer tous les monstres de classe MonstreRampant de la map.
* \_monstresVolants : c’est une liste de classe MonstreVolant qui permet de gérer tous les monstres de classe MonstreVolant de la map.
* \_myGame : c’est un objet de classe Game1 qui permet d’hériter des fonctions dans Game1.
* \_openingPortal : c’est un objet de classe Recompenses qui permet de faire apparaitre un portail.
* \_partiesPortail : c’est un tableau de classe Recompenses qui permet de gérer tous les morceaux de portail de la map.
* \_partiesRecoltees : c’est un objet de classe int qui permet de connaitre le nombre de morceau de portail récolté.
* \_pingouin : c’est un objet de classe Pingouin qui permet d’utiliser un pingouin.
* \_posiCoins : c’est un tableau de classe Vector2 qui permet d’initialiser la position des pièces sur la map.
* \_posiPartiPortail : c’est un tableau de classe Vector2 qui permet d’initialiser la position des diffèrent morceau de portail sur la map
* \_positionChrono : c’est un objet de classe Vector2 qui permet d’initialiser la position du chrono et la mettre à jour en même temps que la caméra.
* \_recolteesPosition : c’est un objet de classe Vector2 qui permet d’initialiser la position des morceau récoltées et de la mettre à jour avec la caméra.
* \_snowballs : c’est un tableau d’objet de classe Snowball, qui permet de gérer les boules de neige.
* \_spikesLayer : est un objet de classe TiledMapTileLayer, il stocke toutes les tiles représentant des piques
* \_tiledMap : c’est un objet de classe TiledMap, il s’agit de la map.
* \_tiledMapRenderer : c’est un objet de classe TiledMapRenderer, il s’agit du rendu visuel de la map.
* \_traps : c’est une liste de classe Trap qui permet de gérer les pièges sur la map.
* \_coinSound : c’est un objet de classe SoundEffect qui permet de charger un son et de le jouer lorsque l’utilisateur récupère une pièce.
* HAUTEUR\_FENETRE : c’est une constante de type int qui permet de connaitre la hauteur de la fenêtre.
* hauteurRecompense1 : c’est une variable de type int qui permet de connaitre la hauteur des pièces.
* LARGEUR\_FENETRE : c’est une constante de type int qui permet de connaitre la largeur de la fenêtre.
* largeurRecompense1 : c’est une variable de type int qui permet de connaitre la largeur des pièces.
* recupAllPortalSound : c’est un objet de classe SoundEffect qui permet de charger un son et de le jouer lorsque l’utilisateur récupère tous les morceau de portail.
* scale : c’est une variable de type float, cette variable stocke le zoom de la caméra.
* soundtrack : c’est un objet de classe Song qui permet de diffuser une musique en arrière-plan durant la partie.

## Présentation détaillé GameOver

Une image contenant table

Description générée automatiquement

GameOver : GameScreen faisant le lien visuel entre Game1 et les classes GameScreen  Menu et Desert ou Snow selon la parti lancer. Apparait lorsque le pingouin a perdu toute sa vie.

Elle contient 11 champs :

* \_mouseState : c’est un objet de classe MouseState qui permet de savoir l’état de la souris.
* \_myGame : c’est un objet de classe Game1 qui permet d’hériter des fonctions de Game1.
* \_pingouin : c’est un objet de classe Pingouin qui permet d’afficher un pingouin.
* \_policeGO : c’est un objet de classe SpriteFont qui permet d’appliquer une police sur les textes à l’affichage.
* messageMenu : c’est un objet de classe string qui permet d’initialiser le texte à afficher.
* messagePerdu : c’est un objet de classe string qui permet d’initialiser le texte à afficher.
* messageRejouer : c’est un objet de classe string qui permet d’initialiser le texte à afficher.
* positionMesageMenu : c’est un objet de classe Vector2 qui permet d’initialiser la position de messageMenu.
* positionMessagePerdu : c’est un objet de classe Vector2 qui permet d’initialiser la position de messagePerdu.
* positionMessageRejouer : c’est un objet de classe Vector2 qui permet d’initialiser la position de messageRejouer.

## Présentation détaillé Win

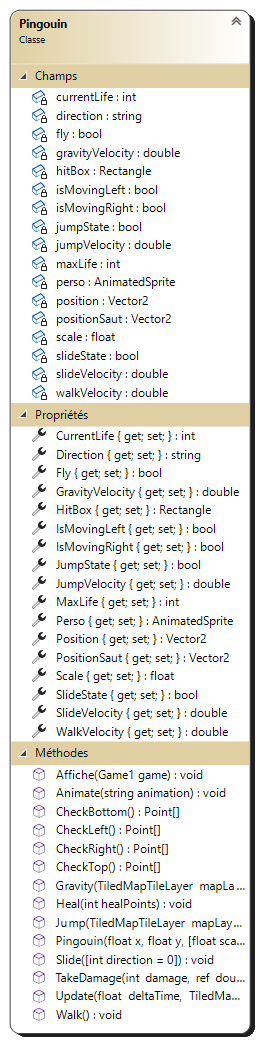


Win : GameScreen faisant le lien visuel entre Game1 et les classes GameScreen Menu et Snow si la partie lancer est Desert. Apparait lorsque la partie a été remporter.

Elle contient 9 champs :

* \_mouseState : c’est un objet de classe MouseState qui permet de savoir l’état de la souris.
* \_myGame : c’est un objet de classe Game1 qui permet d’hériter des fonctions de Game1.
* \_pingouin : c’est un objet de classe Pingouin qui permet d’afficher un pingouin.
* messageGagner : c’est un objet de classe string qui permet d’initialiser le texte à afficher.
* messageMenu : c’est un objet de classe string qui permet d’initialiser le texte à afficher.
* messageNivSuiv : c’est un objet de classe string qui permet d’initialiser le texte à afficher.
* positionMessageGagner : c’est un objet de classe Vector2 qui permet d’initialiser la position de messageGagner.
* positionMessageMenu : c’est un objet de classe Vector2 qui permet d’initialiser la position de messageMenu.
* positionMessageNivSuiv : c’est un objet de classe Vector2 qui permet d’initialiser la position de message .

## Présentation détaillé Pingouin



La classe Pingouin stocke toutes les informations du pingouin. Elle permet de le faire bouger, de l’animer et de le faire prendre des dégâts. Elle est composée de 17 champs.

* currentLife : c’est un objet de classe int qui permet de savoir combien de cœur le pingouin a.
* direction : c’est un objet de classe string, cette variable stocke une chaîne de caractère ("Right" ou "Left"), cette variable par la méthode Animate afin de sélectionner les animations en fonction de la direction.
* fly : c’est un objet de classe bool qui permet de savoir si le pingouin touche ou non le sol.
* gravityVelocity : c’est une variable de type Double, qui permet de régler la force de gravité appliquée au pingouin
* hitBox : c’est un objet de classe Rectangle, qui sert à vérifier les collisions avec les autres Sprite.
* isMovingLeft : c’est un objet de classe bool qui permet de savoir si le pingouin se déplace vers la gauche.
* isMovingRight : c’est un objet de classe bool qui permet de savoir si le pingouin se déplace vers la droite.
* jumpVelocity : c’est une variable de type Double, elle permet de régler la vitesse de saut du pingouin.
* maxLife : c’est un objet de classe int qui permet d’initialiser la vie maximal du pingouin.
* perso : c’est un objet de classe AnimatedSprite, cela permet au personnage d’être animé.
* position : c’est un objet de classe Vector2, cette variable permet de stocker la position du pingouin.
* positionSaut : c’est un objet de classe Vector2, cette variable permet de stocker la position du pingouin au déclenchement du saut.
* scale : c’est une variable de type float, cette variable stocke le zoom de la caméra, cela permet de redimensionner le pingouin.
* slideState : c’est un booléen, qui permet de savoir si le pingouin est en train de glisser ou non.
* slideVelocity : c’est une variable de type double, qui permet de régler la vitesse du pingouin lorsqu’il glisse.
* walkVelocity : c’est une variable de type double, qui permet de régler la vitesse de marche du pingouin.

## Présentation détaillé Snowball

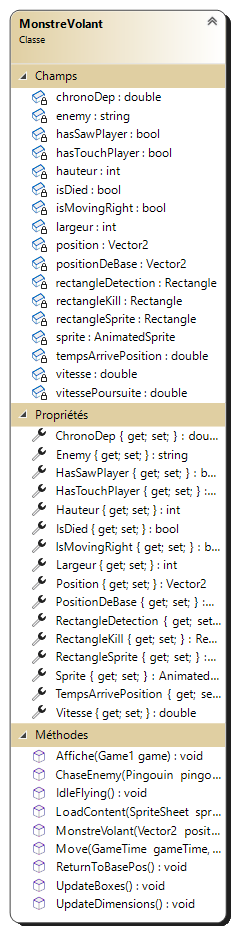
Une image contenant table

Description générée automatiquement

La classe Snowball stocke toutes les informations liées à une boule de neige. Cette classe permet également d’appliquer un mouvement à une boule de neige. Elle est composée de 8 champs.

* distance : est une variable de type float, elle stocke la distance parcourue par la boule de neige.
* Height : est une variable de type int, elle stocke la hauteur de la texture de la boule de neige.
* hitBox : c’est un objet de classe RectnangleF, il s’agit du rectangle pouvant rentrer en collision avec les autres Sprite.
* middle : c’est un objet de classe Vector2, cette variable stocke les coordonnées du point d’origine de la boule de neige.
* position : c’est un objet de classe Vector2, cette variable stocke la position de la texture de la boule de neige.
* texture : c’est un objet de classe Texture2D, cette variable stocke l’image de la boule de neige.
* velocity : c’est un objet de classe Vector2, elle stocke le mouvement de la boule de neige, sa vitesse et le sens.
* width : est une variable de type int, elle stocke la largeur de la texture de la boule de neige.

## Présentation détaillé MonstreVolant



La classe MonstreVolant stocke toutes les informations liées à un aigle. Elle permet également de le faire bouger, de l’animer. Cette classe est composée de 17 champs.

* chronoDep : est une variable de type double, cette variable stocke le temps écoulé depuis le changement de direction.
* enemy : est une variable de type string, qui permet de changer la taille de la box de collision du monstre selon le type d'ennemi, les largeurs et hauteurs sont alors renseignées en conséquence selon la taille du sprite du monstre.
* hasSawPlayer : est un booléen permettant de savoir si le pingouin est dans le champ de vision de l’aigle.
* hasTouchPlayer : est un booléen permettant de savoir si l’aigle a touché le pingouin.
* hauteur : est une variable de type int, cette variable stocke la hauteur de la hitbox de l’aigle.
* isDied : est un booléen permettant de savoir si l’aigle est mort.
* isMovingRight : est un booléen permettant de savoir si l’aigle se déplace vers la droite.
* largeur : est une variable de type int, elle stocke la largeur de la hitbox de l’aigle.
* \_position : c’est un objet de classe Vector2 qui permet d’initialiser la position.
* positionDeBase : c’est un objet de classe Vector2 qui permet d’initialiser la position.
* rectangleDetection : c’est un objet de classe Rectangle, qui définit le rectangle dans lequel l’aigle détecte le pingouin.
* rectangleKill : c’est un objet de classe Rectangle, qui définit le rectangle permettant au pingouin de tuer l’aigle.
* rectangleSprite : c’est un objet de classe Rectangle, qui définit la hitbox de l’aigle.
* sprite : est un objet de classe AnimatedSprite, il permet à l’aigle d’être animer.
* tempsArrivePosition : est une variable de type double, cette variable stocke le temps que met l’aigle pour se rendre à sa prochaine position.
* vitesse : est une variable de type double, cette variable stocke la vitesse de l’aigle.
* vitessePoursuite : est une variable de type double, cette variable stocke la vitesse de l’aigle lorsqu’il attaque le pingouin.

## Présentation détaillé MonstreRampant

La classe MonstreRampant stocke toutes les informations liées à un renard. Elle permet également de le faire bouger, de l’animer. Cette classe est composée de 13 champs.

* chronoDep : est une variable de type double, cette variable stocke le temps écoulé depuis le changement de direction.
* enemy : est une variable de type string, qui permet de changer la taille de la box de collision du monstre selon le type d'ennemi, les largeurs et hauteurs sont alors renseignées en conséquence selon la taille du sprite du monstre.
* deathSong : est un objet de classe SoundEffect, il stocke le son jouer lors de la mort du renard.
* hauteur : est une variable de type int, elle stocke la hauteur de la hitbox de l’aigle.
* isDied : est un booléen permettant de savoir si le renard est mort.
* isMovingRight : est un booléen permettant de savoir si le renard se déplace vers la droite.
* largeur : est une variable de type int, elle stocke la largeur de la hitbox de l’aigle.
* position : c’est un objet de classe Vector2 qui permet d’initialiser la position
* rectangleKill : c’est un objet de classe Rectangle, qui définit le rectangle permettant au pingouin de tuer le renard.
* rectangleSprite : c’est un objet de classe Rectangle, qui définit la hitbox du renard.
* sprite : est un objet de classe AnimatedSprite, il permet au renard d’être animer.
* tempsArrivePosition : est une variable de type double, cette variable stocke le temps que met le renard pour se rendre à sa prochaine position.
* vitesse : est une variable de type double, cette variable stocke la vitesse du renard.

## Présentation détaillé Trap

Une image contenant texte

Description générée automatiquement

Trap :

Elle contient 8 champs :

* \_position :
* \_sprite :
* canCollindingTrap :
* chronoActivation :
* hauteur :
* largeur :
* rectangleSprite :
* trapType :

## Présentation détaillé Recompenses

Une image contenant texte

Description générée automatiquement

Recompenses :

Elle contient 8 champs :

* effetPiece : c’est un objet de classe int qui permet de connaitre l’effet d’une récompense de type piece.
* etat : c’est un objet de classe int qui permet de vérifier si la récompense a été prise ou non.
* hauteur : c’est un objet de classe int qui permet de connaitre la hauteur de la récompense.
* largeur : c’est un objet de classe int qui permet de connaitre la largeur de la récompense.
* position : c’est un objet de classe Vector2 qui permet de connaitre la position de la récompense.
* rectangleSprite : c’est un objet de classe rectangle qui permet de créé un rectangle autour de la récompense.
* sprite : c’est un objet de classe AnimatedSprite qui permet de mettre un AnimatedSprite sur la récompense.
* typeRecompense : c’est un objet de classe string qui permet de savoir quel type de récompense est entré.

## Présentation détaillé Collision



La classe Collision permet de vérifier les collisions entre les différents objets et avec la map. Cette classe est composée de 5 méthodes.

* IsCollidingRecompense : est une méthode retournant un booléen, elle permet de détecter si le pingouin entre en collision avec une pièce ou un fragment de portail.
* MapCollision : est une méthode retournant un booléen, elle permet de détecter si un objet de type Point se trouve dans une tile du layer de la map, renseigné en paramètre. Cette méthode est surchargée afin de prendre en compte les tableaux de Point.
* SpriteCollision : est une méthode retournant un booléen, elle permet de détecter si deux objets rentrent en collision, grâce à leur hitbox. Cette méthode est surchargée afin de prendre en compte un tableau d’objet de type RectangleF.

## Présentation détaillé Camera



Camera :

Elle contient 3 champs :

* \_cameraPosition : c’est un objet de classe Vector2 qui permet d’initialiser la position de
* \_hauteurFen : c’est un objet de classe int
* \_orthographicCamera : c’est un objet de classe OrthographicCamera qui permet de…

## Présentation détaillé Chrono



Chrono :

Elle contient 2 champs :

* \_chrono : c’est un objet de classe double
* \_chronoInvincibility : c’est un objet de classe double

## Présentation détaillée GameManager

Une image contenant texte

Description générée automatiquement

La classe GameManager permet de centraliser et de piloter les différentes étapes durant un niveau (collisions, entrées …). Elle est composée de 14 champs.

* \_attaquer : c’est un objet de classe Keys, cette variable permet de stocker la touche d’attaque.
* coinSong : c’est un objet de classe SoundEffect, il stocke le son jouer lorsque le pingouin récolte une pièce.
* \_droite : c’est un objet de classe Keys, cette variable permet de stocker la touche de déplacement vers la droite.
* \_gauche : c’est un objet de classe Keys, cette variable permet de stocker la touche de déplacement vers la gauche.
* \_glisser : c’est un objet de classe Keys, cette variable permet de stocker la touche de glisse.
* \_hitSnowball : c’est un objet de classe SoundEffect, il stocke le son jouer lorsqu’une boule de neige touche un monstre.
* \_monstreSong : c’est un objet de classe SoundEffect, il stocke le son jouer lorsque le pingouin entre en collision avec un monstre.
* \_portalSong : c’est un objet de classe SoundEffect, il stocke le son jouer lorsque le pingouin récupère un fragment de portail.
* \_sauter : c’est un objet de classe Keys, cette variable permet de stocker la touche de saut.
* \_snowballTexture : c’est un objet de classe Texture2D, qui stocke la texture d’une boule de neige.
* \_throwSnowball : c’est un objet de classe SoundEffect, qui stocke le son jouer lorsqu’une boule de neige est lancée.
* \_timer : c’est une variable de type float, qui stocke le temps écoulé depuis le dernier lancé de boule de neige.
* \_timerSpike : c’est une variable de type float, qui stocke le temps passé depuis que le a été touché par un pique.
* \_trapSong : c’est un objet de classe SoundEffect, il stocke le son jouer lorsque le pingouin est touché par un piège.

# Conception graphique

Nous avons réalisé nos deux maps nous même, en utilisant des tiles Sheets trouver sur itch.io. Nous avons réalisé nos maps a l’aide de Tiled et avons effectuer plusieurs séries de teste afin de vérifié que le parcours était possible avec le pingouin.

Nos décors ont été trouver eux aussi sur internet, tout comme nos sprites que nous avons par la suites animé sans modifié les originaux trouvé sur itch.io.

Indiquez si vos maps, décors,sons sont des sources existantes, dans ce cas donnez leur provenance Expliquez les retouches ou création que vous avez peut être réalisés.

# Partie Algorithmie – Intelligence artificielle

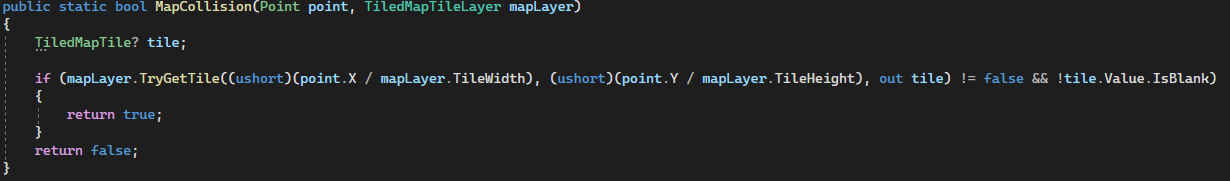
## Explications

Afin de détecter les collisions entre les différents objets, nous avons utilisé des hitbox propre à chaucn. Il s’agit d’un rectangle (invisible) délimitant la partie de l’objet pris en compte pour les collisions. Pour savoir si deux objets entre en collision il nous suffit donc de vérifier si les deux rectangle ne se coupent pas, grâce à la méthode Intersects.

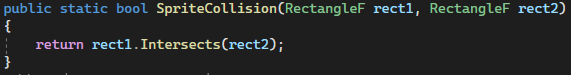
Afin de détecter les collisions entre un objet et la map, nous avons utilisé un ou plusieurs point(s), dont nous comparons les coordonnées avec celles des tuiles d’un calque de la map. Si le point/l’un de ces points ont les mêmes coordonnées qu’une tuile, l’objet et la map se touchent.

L’aigle est le seul monstre à se déplacer en fonction du pingouin. En effet, lorsque le pingouin est à une certaine distance de l’aigle, ce dernier se dirige vers lui. L’aigle détecte le pingouin grâce à une grande hitbox. Ainsi lorsque cette hitbox entre en collision avec celle du pingouin, l’aigle se dirige vers le joueur.

## Extrait de code



Fonction permettant de détecter les collisions entre un objet la map.

Fonction permettant de détecter les collisions entre deux objets.

# Cahier de recettes

## 5.1 Tests de validation

|  |  |  |
| --- | --- | --- |
| **Nom** | **Fonctionnalité** | **Etat** |
| Labauve | Menu principal | OK |
| Labauve | Sprite Coin | OK |
| Labauve | Map niveau 1 (map desert) | OK |
| Labauve | Animations portail | OK |
| Labauve | Classe Recompenses | OK |
| Labauve | Scène GameOver | OK |
| Labauve | Scène Regle | OK |
| Labauve | Scène Win | Non achevé |
|  |  |  |
|  |  |  |
|  |  |  |
| Clerc-Renaud | Map niveau 2 (map neige) | OK |
| Clerc-Renaud | Classe Trap | OK |
| Clerc-Renaud | Animations trap | OK |
| Clerc-Renaud | Animations renard | OK |
| Clerc-Renaud | Animations eagle | OK |
| Clerc-Renaud | Classe MonstreRampant | OK |
| Clerc-Renaud | Classe MonstreVolant | OK |
|  |  |  |
|  |  |  |
| Sauthier | Animations pingouin | Ok |
| Sauthier | Classe Pingouin | OK |
| Sauthier | Classe Snowball | OK |
| Sauthier | Détection et gestion des collisions | OK |
| Sauthier | Centralisation dans GameManager | Non achevé |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## 5.2 Tests de performance

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

