Voici les exigences du code pour ce travail :

1 enum avec au moins 3 éléments

Les objets présents dans le jeu sont des enums. Il y en a 5 en tout : le croissant, le fromage, le lait, la soupe et les graines. Le joueur peut ramasser ces objets et les mettre dans son inventaire.

1 instruction d'itération (for, foreach, while)

Il y en a beaucoup dans le code, mais par exemple j'ai utilisé un for pour dessiner la carte, ligne 143, j'ai aussi utilisé un switch pour déterminer la texture de l'objet que le joueur ramasse ligne 466, et un foreach pour les collisions et update des objets à la ligne 371.

1 tableau

Pour créer la carte, j'ai utilisé un tableau qui lit la carte dans un document texte pour ensuite le retranscrire dans le jeu, ligne 142.

1 appel de méthode avec 2 arguments nommés

Pour faire les collisions d'objets avec le joueur, il fallait que j'utilise à la ligne 373 la méthode update de la classe Objet, utilisant les arguments gameTime et playerRectangle, pour qu'elle soit appelée en fonction du temps de jeu et de gérer la collision entre le rectangle de l'objet et le rectangle du joueur.

1 définition de méthode avec 1 argument optionnel

Malheureusement je n'ai pas pensé à implémenter de méthode avec argument optionnel. Je n'arrive pas à trouver dans quel contexte dans mon code j'en aurai besoin.

1 définition de méthode surchargé (overloaded)

Les méthodes Update d'objet sont surchargées, l'une pour les collisions, l'autre pour les conditions de victoire (ligne 493).

Héritage de classes :

1 classe abstraite de base

La classe TuileBase est une classe abstraite.

3 classes dérivées

Les classes grass_tile, water_tile, rock_tile, forest_tile et sand_tile sont toutes dérivées de TuileBase.

1 méthode à substituer (virtual, override)

Les classes dérivées ci-dessus comportent toutes un override sur le virtual Draw de TuileBase.

1 utilisation de polymorphisme

Le polymorphisme est utilisé lors de la création de la carte, ligne 143, car la liste de la carte est TuileBase, mais toutes les tuiles elles-mêmes sont des Water_tile ou des rock_tile, etc…

1 délégué et 1 événement

Dans la classe objet, j'ai utilisé un délégué et un événement pour PickUp des objets. Ainsi, dans mon code, lorsque le rectangle du joueur entre en collision avec le rectangle de l'objet, il va ramasser (événement) l'objet et l'ajouter dans son inventaire (délégué) ligne 374.

1 collection utilisée (List, HashSet, Dictionary)

Pour créer l'inventaire, j'ai pu créer mon propre dictionnaire en utilisant des génériques. Ainsi, l'inventaire possède deux génériques, l'un pour le type d'objet, l'autre pour le nombre d'objets ramassés.

1 donnée sauvegardée

Pour lire la carte, j'utilise le streamline pour lire des données déjà écrites contenues dans un document texte (ligne 142).

Voici les exigences au niveau du jeu pour ce travail :

Doit utiliser au moins le clavier ou la souris pour les contrôles

Le clavier est utilisé pour contrôler le personnage autour de la carte.

1 minute de jeu (gameplay)

Cela prend environ 1 minute pour comprendre et trouver tous les objets. Puisque la génération d'objets est aléatoire, il se peut que cela prenne moins de temps, mais à chaque lancement du jeu, le joueur devra trouver les mêmes objets à différents endroits.

3 textures différentes

En utilisant le site Piskel, j'ai créé moi-même les différentes textures du jeu. Il y en a à peu près une dizaine.

1 détection de collision

Il y en a plusieurs, notamment les ennemis – personnage, personnage – objets, et personnage – éléments du décor (roches et eau impossible à passer) ligne 410.

Utilisation de la classe GameTime pour le temps

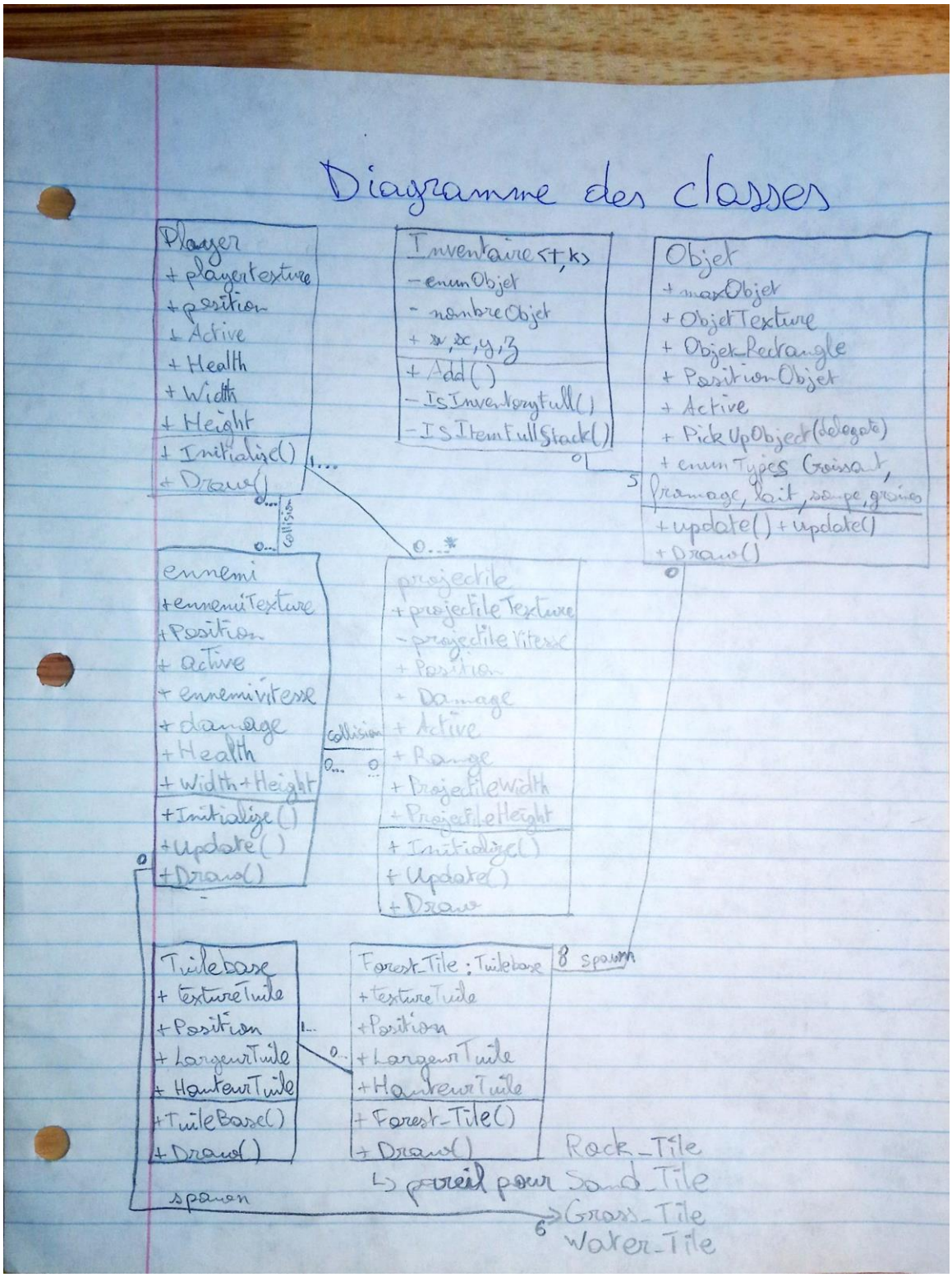
Utilisée plusieurs fois, la classe gameTime permet dans mon jeu de vérifier en fonction du taux de trame (frame rate) les collisions.

Décrire le design du jeu et utilisez des captures d'écran

Alphonse est un jeu vidéo dans lequel le joueur incarne un petit rat montréalais – Alphonse – qui doit essayer de survivre et trouver de la bonne nourriture pour se repaître.

Comme on peut le voir sur l'image ci-dessus, le personnage, Alphonse est à peu près au milieu de l'écran (rat). Tout en bas de l'écran il y a l'inventaire, avec les différents objets déjà ramassés (ici, croissant et soupe). Tout en bas de l'écran de jeu, au milieu, on peut voir une icône blanche, c'est le lait, un autre objet pouvant être ramassé par le joueur. Au milieu gauche, on peut voir deux silhouettes bleues, ce sont les ennemis, des oiseaux, voulant manger notre protagoniste. Au niveau de la carte, le joueur peut aller sur l'herbe, le sable et la forêt, mais ne peut pas nager, ni escalader les rochers, donc il ne peut pas passer à travers ces tuiles-là.

Expliquez la hiérarchie de classe avec un diagramme



Alors pour expliquer ce diagramme de classes :

Player est une classe qui va s'initialiser une fois au début de la partie. C'est le joueur. Elle possède un rectangle (initialisé dans Game1) qui va déterminer si le joueur entre en collision avec l'ennemi ou le départ du projectile. Ensuite, l'ennemi est quasiment la même classe que le joueur mais possède sa propre Update() qui va déterminer « l'IA » basique de l'ennemi : dès que le joueur entre dans une certaine zone d'aggro de l'ennemi, l'ennemi va se diriger vers le joueur.

Concernant la classe TuileBase : elle est abstraite et hérite les classes suivantes : water_tile, forest_tile, grass_tile, sand_tile et rock_tile. La grass_tile permet aussi le spawn aléatoire des ennemi, tandis que la forest_tile permet le drop aléatoire des objets.

Pour les classes Objet et Inventaire<T, K>, Objet va déterminer les caractéristiques de l'objet (c'est un enum qui indique s'il s'agit d'un croissant, d'un fromage, d'une soupe, du lait ou des graines), puis le dessiner à la bonne position. La collection Inventaire va faire en sorte de créer une liste des objets que possède le personnage, puis la méthode Update() surchargée d'Objet va déterminer la condition de victoire, qui est atteinte une fois 20 objets ramassés.

Discutez des difficultés rencontrées lors du projet et vos solutions

Au moins une discussion sur le développement d'un algorithme pour résoudre un problème en particulier

Premièrement, les tuiles de la carte. J'ai eu beaucoup de difficulté à trouver un algorithme pour créer la carte, j'avais essayé plusieurs solutions auparavant mais comme c'était la première fois que j'en codais une et tout ce que je savais faire, c'était afficher des rectangles à l'écran… Ainsi, après plusieurs essais sur plusieurs semaines, je suis passé de « hardcoder » la carte à utiliser le streamline pour plus de simplicité. En essayant d'implémenter chaque tuile de 32 pixels par 32 pixels, j'y passais beaucoup trop de temps, et cela prenait une trop grande place dans le code. Après le cours sur les librairies et les doc textes, et après plusieurs discussions avec mes camarades, j'ai commencé à découvrir le streamline. Au début, j'avais encore du mal à comprendre comment cela fonctionnait, mais au fil du temps j'ai pu déterminer l'algorithme exact pour lire les différentes lettres du texte et déterminer quelle tuile correspond à quelle lettre. Une fois cela fait, l'instruction « for » de mon algorithme permet de placer la tuile au bon endroit, en multipliant les x et les y par le nombre de pixels (32), pour éviter la superposition d'images. Une fois tout ce code écrit, j'ai été très satisfait du résultat et j'ai presque bondi de joie dans le Starbucks dans lequel je travaillais cet après-midi-là!

Au moins une discussion sur une difficulté liée à la langue de programmation

Ma deuxième principale difficulté était les collisions. Ici, j'ai eu beaucoup de mal à les faire fonctionner à cause d'un simple petit détail du langage… Le problème avec cela, c'est que c'est

assez difficile à trouver, ou déterminer la source du problème, et j'ai passé à peu près une semaine à essayer de trouver le bogue. Toute ma logique dans mon code était bonne, j'avais vraiment vérifié les lignes de code, trouvé plusieurs façons différentes de tester, mais à chaque fois, il n'y avait qu'un seul rectangle avec lequel mon personnage pouvait interagir, et c'était la dernière tuile d'eau. Après maints essais, je me suis dis que s'il n'y avait qu'un seul rectangle, c'est qu'il y avait un problème de *range* : en étudiant cela d'un peu plus près, je me suis rendu compte que les rectangles se réinitialisaient à chaque fois que j'en créais un. Pour régler tous mes problèmes, il a simplement fallu que j'initialise ma variable quelques lignes plus haut, et tout fonctionnait. Parfois, de simples erreurs d'encapsulation vont tout faire boguer sans que l'on sache réellement ce qui se passe, et c'est les bogues les plus durs à trouver je pense…

Citez les sources utilisées pour le projet

https://www.piskelapp.com/

http://www.monogame.net/documentation/?page=main