MINI-PROJET D'INFORMATIQUE

Fait par Maël LECOEUCHE et Souley MOHAMAN BELLO

I. SPECIFICATIONS

1. CÔTÉ MENU

On proposera un menu à 4 entrées :

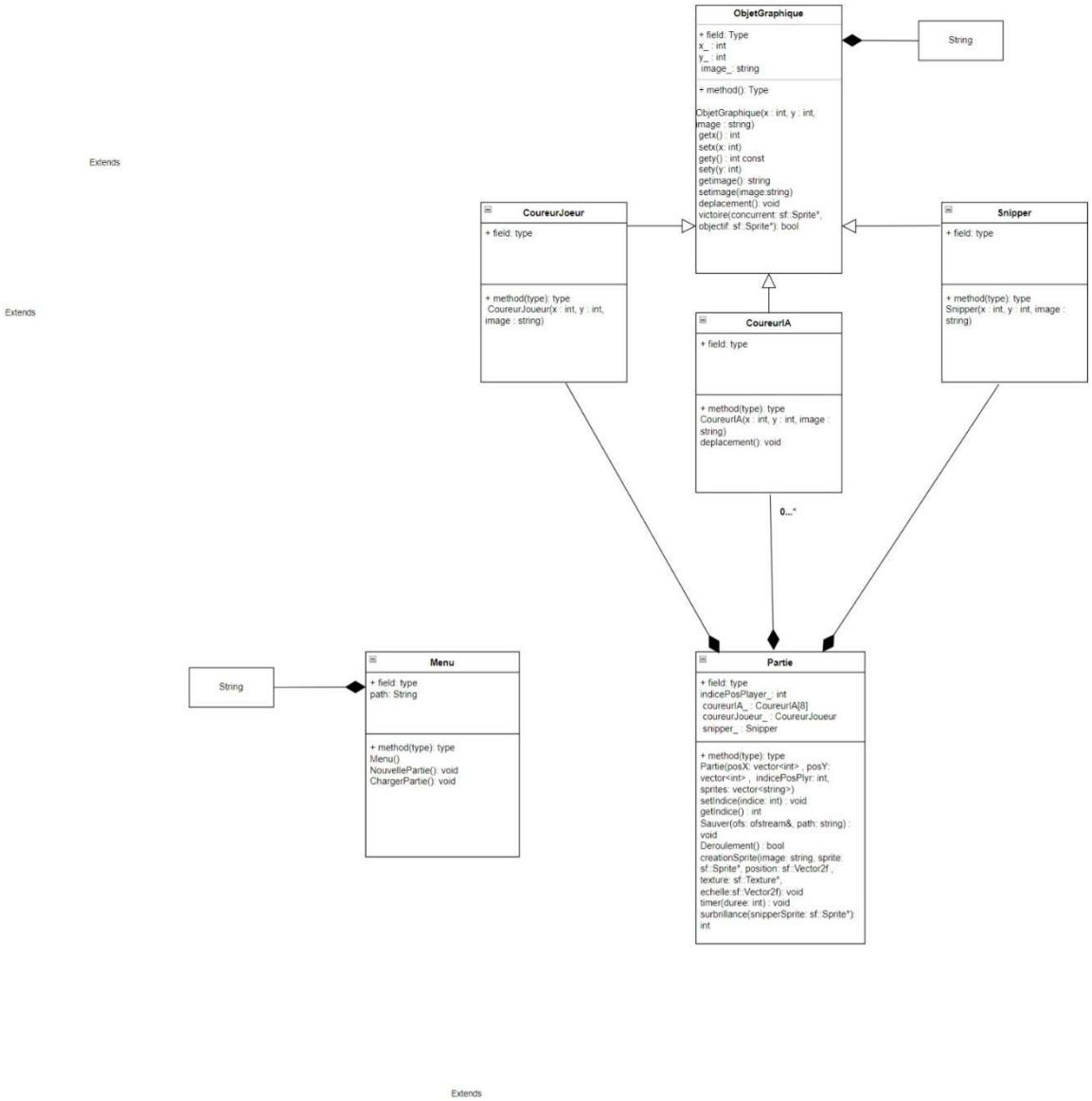
- Lancer partie : cette option permettra de lancer une nouvelle partie.
- Charger partie : parmi une liste de parties possibles, cette option permet de résumer une partie antérieure.
- Connaître les règles : affiche comment se joue le jeu.
- Quitter le jeu : cette option permet de fermer le programme

2. CÔTÉ FENÊTRE

La fenêtre de jeu devra assurer les fonction suivantes :

- Affichage de la map, des « coureurs » et du viseur
- Assurer le déplacement des « coureurs » et du viseur
- Avant le lancement de la partie, prévoir une phase d'attribution automatique du « coureur joueur »
- Assurer la capacité de tir du « snipper »
- Mise en surbrillance des coureurs que le snipper a en visée
- Être capable de déterminer la victoire de l'un ou l'autre des joueurs et afficher un évènement spécifique
- Mise en pause qui par une fermeture de la fenêtre avant la fin de la partie en appuyant sur la touche échappe du clavier et retour au menu console(le choix de sauvegarder ou pas la partie sera laissée aux utilisateurs)
- Jouer une musique

II. DIAGRAMME UML



Plus haut, nous avons le diagramme de classes de notre projet(une version jpeg sera fournie dans le dossier du projet sous le nom « UML.jpeg »). A noter que les classes héritières d'ObjetGraphique sont composées de String car elles héritent de ses champs, cependant pour pouvoir aérer le diagramme, nous avons préféré ne pas représenter ces relations de composition.

Nous avons choisi des faire de l'héritage afin de pouvoir faire des méthodes virtuelles en charge du déplacement du joueur(une pour joueur IA et une pour un coureur joueur). Nous avons décidé de mettre en composition Partie et CoureurJoueur, CoureurIA et Snipper afin qu'elle puisse les détruire lorsque la fenêtre est fermée et leur création lorsqu'une partie est lancée. La classe est ainsi censée tout contrôlée en ce qui concerne l'interface graphique, et dans une moindre mesure la console afin de sauvegarder ses données. Puisque l'on utilise que des tableaux statiques, il n'y a pas besoin de destructeur dans Partie. La classe Menu quand est charger de tout gérer au niveau de la console, et n'a donc pas besoin de connaître les éléments graphiques.

III. DÉFINITIONS DES CLASSES

N.B : Lorsque cela n'est pas rappelé, on précise que les paramètres des constructeurs sont les valeurs d'initialisations des champs qui portent le même nom(à un caractère « _ » près) et que ceux des méthodes « getChamp » sont les valeurs d'affectations encore une fois des champs du même nom.

1. MENU(ENTIÈREMENT RÉALISÉE PAR MAËL LECOEUCHE)

Cette classe gère le menu de base permettant de lancer une partie ou d'en charger une ancienne, d'afficher les règles et quitter l'application.

Menu est composée d'un champ constant path de type String

Menu est composée des méthodes :

- Menu() : qui est le constructeur de Menu
- void NouvellePartie() : qui permet de créer une nouvelle partie
- void ChargerPartie() : qui permet de charger une partie sauvegardée

2. PARTIE

Cette classe gère le déroulement d'une partie d'une partie et sa sauvegarde.

Partie est composée des champs suivants, réalisés par Maël LECOEUCHE :

- int indicePosPlayer_ : c'est l'indice du coureur joueur dans le tableau des coureurIA
- CoureurJoueur coureurJoueur_ : c'est le joueur jouant au clavier
- Snipper snipper_ : c'est le joueur jouant à la souris
- CoureurIA coureurIA_[8] : tableau contenant tous les coureurs IA. Une case correspondant au coureurjoueur sera laissée vide, elle sert juste à l'attribution aléatoire d'un personnage au coureur joueur

Partie est composée des méthodes :

- Partie(vector<int> posX, vector<int> posY, int indicePosPlyr, vector<string> sprites) : Constructeur de la partie prenant les positions de départ des joueurs en paramètre,la position du joueur dans le tableau et les sprites, par Maël LECOEUCHE
- void setIndice(int indice) : modifie le champ indicePosPlayer_ qui prend la valeur du paramètre indice, par Maël LECOEUCHE
- int getIndice() : retourne la valeur du champ indicePosPlayer_, par Maël LECOEUCHE
- void Sauver(ofstream &ofs, string path) const : Sauvegarde les données dans des fichiers txt. On prend en paramètre le flux dans lequel se fera la sauvegarde ainsi que le nom path du fichier de sauvegarde. Par Maël LECOEUCHE
- bool Deroulement() : Cette méthode gère le dessin de la fenêtre, la création et le dessin des sprites et la gestion des évènements clavier et souris(notamment tir et déplacement du snipper), par MOHAMAN BELLO Souley
- void creationSprite(string image, sf::Sprite* sprite, sf::Vector2f position, sf::Texture* texture, sf::Vector2f echelle=sf::Vector2f(0.125f,0.125f)) : cette méthode crée un sprite à partir du chemin d'accès vers l'image à utiliser pour la texture, un pointeur sur l'objet sprite concerné, un pointeur sur sa texture et elle initialise la position du sprite dans la fenêtre grâce au paramètre position et son échelle grâce au paramètre echelle, par MOHAMAN BELLO Souley
- void timer(int duree) : cette méthode réalise une boucle permettant de faire passer <<valeur de duree>> millisecondes, par MOHAMAN BELLO Souley
- int surbrillance(sf::Sprite* snipperSprite) : cette méthode permet d'indiquer à travers l'indice entier retourné à la fonction déroulement si le snipper à en visée un coureur, afin qu'il soit affiché en rouge plus tard dans deroulement. Elle prend en paramètre un pointeur sur le sprite du snipper. Par MOHAMAN BELLO Souley.

3. ObjetGrapghique(entièrement réalisée par MOHAMAN BELLO Souley)

C'est la classe générique qui définit ce qu'est un objet(coureurs, snippers) à l'écran, sans gérer leur affichage(dessin de sprites).

Elle comporte les champs :

- int x_: abscisse de l'objet
- int y_: ordonnée de l'objet
- string image_ : nom du fichier contenant l'image de l'objet

Elle comporte également les méthodes :

- ObjetGraphique(int x=0, int y=0, string image=" ") : Constructeur d'ObjetGraphique qui initialise tous les champs
- int getx() const : retourne la valeur du champ x_
- int gety() const : retourne la valeur du champ y_
- string getimage() const : retourne la valeur du champ image
- void setx(int x) : affecte au champ x_ la valeur de x

- void sety(int y) : affecte au champ y_ la valeur de y
- void setimage(string image) : affecte au champ image_ la valeur de image
- virtual void deplacement() : gère le déplacement du coureurJoueur
- bool victoire(sf::Sprite* concurrent, sf::Sprite* objectif) const : Cette méthode virtuelle renvoie un booléen qui indique si un coureur a franchi la ligne d'arrivée ou pas. Elle prend en paramètre un pointeur sur le sprite du joueur considéré et un pointeur sur le sprite de l'objectif, ici la ligne d'arrivée

4. SNIPPER(ENTIÈREMENT RÉALISÉE PAR MOHAMAN BELLO SOULEY)

C’est une classe héritière d'ObjetGraphique qui gère le joueur à la souris. Elle comporte en plus un constructeur Snipper(int x = 0, int y = 0, string image = " ").

5. COUREURJOUEUR(ENTIÈREMENT RÉALISÉE PAR MOHAMAN BELLO SOULEY)

C’est une classe héritière d'ObjetGraphique qui gère le joueur au clavier. Elle comporte en plus un constructeur CoureurJoueur(int x = 0, int y = 0, string image = " ").

6. COUREURIA(ENTIÈREMENT RÉALISÉE PAR MOHAMAN BELLO SOULEY)

C’est une classe héritière d'ObjetGraphique qui gère un coureur contrôlé par l'ordinateur. Elle comporte en plus un constructeur CoureurIA(int x = 0, int y = 0, string image = " ") et une méthode virtuelle deplacement(), de type void, qui gère le déplacement aléatoire du coureur.