RISE OF SERPENTARD



Par Daoudi Lylia p2105694 Ramdane Amaouz p2002623 ET Cyrine Boudjelel p2103034

-- Appuie sur la touche Enter --

Introduction :

Plongez dans une aventure magique où le mystère rencontre le défi. Dans ce jeu captivant, nous vous invitons à explorer un donjon mystique, où un serpent légendaire, le Basilic, se lance dans une quête pourchassant des sorciers à travers des couloirs sinueux.

Inspiré par l'univers ensorcelant d'Harry Potter, RiseOfSerpentard offre une expérience de jeu unique, combinant l'excitation classique du jeu "Snake" avec les éléments enchanteurs de la saga littéraire bien-aimée.

Préparez-vous à plonger dans un monde où la magie et l'aventure vous attendent à chaque tournant.

Bienvenue dans RiseOfSerpentard !

1- Classe jeu :

```
Jeu::Jeu(): bas(), gal() {
        vitesse = .8;
        temps = 6.0e7;
        score = 0;
        niveau = 1;
        mur = false;
        csc = false;
}

unsigned int directionnormal(unsigned int y)
{

        if (dir==1 && y ==4)
        {
                dir=1;
        }
        else if (dir==2 && y==3)
        {
                dir=2;
        }
        else if (dir==3 && y==2)
        {
                dir=3;
        }

        else if (dir==4 && y==1)
        {
                dir=4;
        }
        else {dir=y;}
        return dir;
}



void Jeu::controleJeu(unsigned int a)
{
        a = directionnormal(a);


        if(bas.corp.getvaleur(1).x == sorc1.getPosX() && bas.corp.getvaleur(1).y == sorc1.getPosY()){
                bas.mouvement(a ,gal);
                bas.corp.setIemeElement(bas.corp.getvaleur(0),bas.corp.getT()+1);
                score++;    //pouffe-souffle
                sorc1.reinitialisePos(gal);
        }else if(bas.corp.getvaleur(1).x == sorc2.getPosX() && bas.corp.getvaleur(1).y == sorc2.getPosY()){
                bas.mouvement(a, gal);
                bas.corp.setIemeElement(bas.corp.getvaleur(0),bas.corp.getT()+1);
                score+=3;   //serre-daigle
                sorc2.reinitialisePos(gal);
        }else if(bas.corp.getvaleur(1).x == sorc3.getPosX() && bas.corp.getvaleur(1).y == sorc3.getPosY()){
                bas.mouvement(a, gal);
                bas.corp.setIemeElement(bas.corp.getvaleur(0),bas.corp.getT()+1);
                score+=5;   //griffon d'or
```

1- ***Constructeur (Jeu::Jeu()):*** initialise les membres de la classe, tels que la vitesse du jeu, le temps, le score, le niveau et quelques indicateurs booléens

.

2- La méthode directionnormal(): normalise une direction donnée « y » en fonction de la direction de actuelle « dir ». Elle est utilisée pour limiter les mouvements du joueur dans certaines situations spécifiques. // 3: gauche 2:droite 4: haut 1: bas.

3- ***Méthode « Jeu::controleJeu() » :*** Gère le contrôle du jeu en fonction des actions du joueur. Elle prend en paramètre une direction a et effectue différentes actions den fonction de la position du joueur et des autres éléments du jeu. Les actions incluent le mouvement du joueur, la mise à jour du score, la modification de la vitesse du jeu, etc. Cette méthode gère également l'évolution du jeu e fonction du score, en augmentant le niveau et en introduisant de nouveaux éléments du jeu au fur et à mesure que le score augmente.

```
        csc = true;   //deuxieme serpentard

}else if(bas.corp.getvaleur(1).x == sorc6.getPosX() && bas.corp.getvaleur(1).y == sorc6.getPosY()){
        csc = true;   //troisieme serpentard

}else if(bas.corp.getvaleur(1).x == sorc7.getPosX() && bas.corp.getvaleur(1).y == sorc7.getPosY()){
        csc = true;   //quatrieme serpentard

}else if(bas.corp.getvaleur(1).x == sorc8.getPosX() && bas.corp.getvaleur(1).y == sorc8.getPosY()){
        csc = true;   //cinquieme serpentard

}else if (gal.dansLeMur(bas.corp.getvaleur(bas.corp.getT()-1)) ){
        mur = true;

}else if(bas.corp.getvaleur(1).x == pilule1.getX() && bas.corp.getvaleur(1).y == pilule1.getY()){
        bas.mouvement(a, gal);
        vitesse = 2.;   //accelere x1

        pilule1.reinitialisePos(gal);

}else if (bas.corp.getvaleur(1).x == pilule2.getX() && bas.corp.getvaleur(1).y == pilule2.getY()){
        bas.mouvement(a, gal);
        vitesse = 0.5;
        temps = 10.0e7;     //ralentisseur x1
        pilule2.reinitialisePos(gal);

}else if (bas.corp.getvaleur(1).x == pilule3.getX() && bas.corp.getvaleur(1).y == pilule3.getY()){
        bas.mouvement(a, gal);
        vitesse = 0.5;
        temps = 2.0e7;     //accelere x2
        pilule3.reinitialisePos(gal);

}else if (bas.corp.getvaleur(1).x == pilule4.getX() && bas.corp.getvaleur(1).y == pilule4.getY()){
        bas.mouvement(a, gal);
        vitesse = 0.5;
        temps = 15.0e7;     //ralentisseur x2
        pilule4.reinitialisePos(gal);

}else{
        bas.mouvement(a, gal);
        sorc1.bougeAuto(gal);
        sorc2.bougeAuto(gal);
        sorc3.bougeAuto(gal);
        sorc4.bougeAuto(gal);
}

if (score >= 10 && score <30) {
        niveau = 2;
        sorc5.bougeAuto(gal);
} else if (score >= 30 && score <50) {
        niveau = 3;
        sorc6.bougeAuto(gal);
} else if (score >= 50 && score <100){
        niveau = 4;
        sorc7.bougeAuto(gal);
}else if (score >= 100){
        niveau = 5;
        sorc8.bougeAuto(gal);
}
```

Explications un peu plus approfondies

Cette méthode ***controleJeu*** gère le déroulement du jeu en fonction des actions du joueur (a) :

Normalisation de la direction : La direction donnée est normalisée en appelant la fonction directionnormal, qui semble ajuster la direction en fonction de certaines conditions spécifiques.

Gestion des collisions avec les sorciers (sorc1 à sorc8) : Si le joueur entre en collision avec un sorcier, différentes actions sont effectuées en fonction du sorcier rencontré : La méthode mouvement de l'objet bas (peut-être le serpent contrôlé par le joueur) est appelée pour déplacer le joueur.

- Le score est mis à jour en fonction du sorcier rencontré.
- La position du sorcier est réinitialisée.

Gestion des collisions avec les pilules (pilule1 à pilule4) : Si le joueur entre en collision avec une pilule, différentes actions sont effectuées en fonction de la pilule rencontrée :

- La méthode mouvement de l'objet bas est appelée pour déplacer le joueur.
- La vitesse du jeu et le temps restant sont ajustés en fonction de la pilule rencontrée.
- La position de la pilule est réinitialisée.

Gestion des collisions avec le mur : Si le joueur entre en collision avec le mur, un indicateur (mur) est défini sur vrai.

Déplacement automatique des sorciers : Si aucune collision n'est détectée, les sorciers restants (s'ils sont dans le jeu) se déplacent automatiquement.

Évolution du jeu en fonction du score : En fonction du score atteint, le niveau du jeu est ajusté et de nouveaux sorciers peuvent être introduits.

.

```cpp
void Jeu::reinitialise(){
        vitesse = 0.8;
        temps = 6.0e7;
        score = 0;
        niveau = 1;
        bas.reinitialise();
}



double Jeu::gettemps() const {
        if (vitesse == 2) return temps/2;

        else return temps - (score * 4.0e5);
}


void Jeu::testRegression() {
    Jeu jeu_test;

    // Test d'un mouvement normal
    jeu_test.controleJeu(1); //manque les assert

    // Affichage des résultats des tests
    std::cout << "Les tests de la classe Jeu sont OK !!" << std::endl;
}
```



- ***jeu::reinitialise ():*** permet de réinitialiser les paramètres du jeu pour recommencer une partie une partie depuis le début.

- ***Jeu::gettemps() const() :*** Calcule et retourne le temps restant en fonction de la vitesse du jeu et du score

- ***jeu::testRegression() :*** Un ensemble de tests pour vérifier le bon fonctionnement des méthodes de la classe. Cependant, cette méthode ne contient pas de vérifications (assertions) pour confirmer le bon fonctionnement des méthodes testées.


Rise of Serpentard
Score :
5
Niveau :
1
Perdu!
tu t'es pris un mur!
R : rejouer
ESC : quitter

Rise of Serpentard

Le mal est libere! La maison Serpentard est sur le point de renaitre!
Le basilic est resuisite et a soif de sang! Oh, mais que vois-je, les autres maisons qui se mobilisent pour tenter de le stopper. Hahaha! Ils vont servir de casse-croute!
Dans ce jeu, tu incarne le basilic, et ton but est d'accumuler des points en mangeant le plus de sorciers possible.
Mais tous les sorciers n'ont pas la meme valeur nutritive :
- un sorcier de la maison Griffon-d'or te rapporte 5 points
- un sorcier de la maison Sert-d'aigle te rapporte 3 points
- un sorcier de la maison Pouffe-souffle te rapporte 1 point
Mais un sorcier de la maison Serpentard n'est pas commestible ne te rappote rien. Ca ne se fait pas de manger ses allies!


pouffe-souffle


serre-d'aigle


griffon-d'or


serpentard

appuie sur ESPACE pour jouer

2- Classe Sorcier :

```
class Sorcier {

private:
    Vec2 position; ///< La position du sorcier dans un espace 2D
    unsigned int dir;///< la direction du sorcier

public:

    /**
     * @brief Constructeur par défaut de la classe Sorcier qui initialise le sorcier à la position (0,0), la direction (1,0) et l'identifiant
     */
    Sorcier();

    /**
     * @brief constructeur par copie
    */

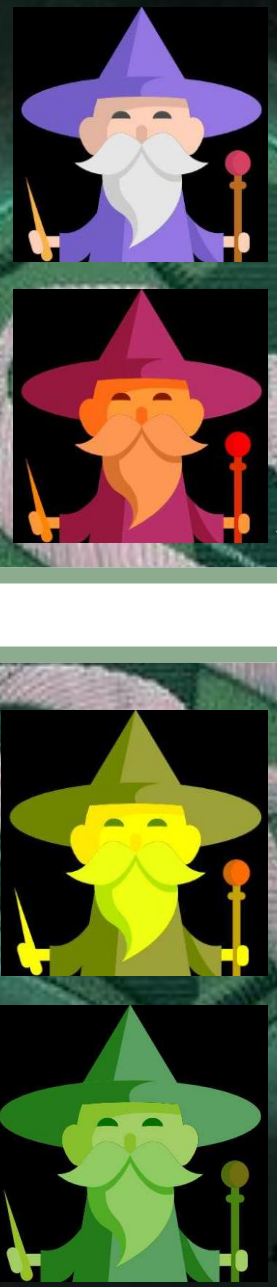
    Sorcier(Vec2 pos, int d);

    /**
     * @brief Récupère la composante x de la position du sorcier
     * @return la composante x du sorcier
     */
    unsigned int getPosX() const;

    /**
     * @brief Récupère la composante y de la position du sorcier
     * @return la composante y du sorcier
     */
    unsigned int getPosY() const;

    /**
    * @brief Déplace automatiquement le sorcier dans la galerie
    * @param g La galerie dans laquelle le sorcier se déplace
    */
    void bougeAuto(const Galerie &g);

    /**
    * @brief Rénitialise la position aux coordonées (1,5)
    *
    */
    void reinitialisePos(const Galerie &g);
```



la classe Sorcier représente des entités mobiles dans un jeu, capables de se déplacer automatiquement et de réinitialiser leur position de manière aléatoire.

Les méthodes getPosX() et getPosY() recuperent la position du sorcier

Méthode reinitialisePos :

Cette méthode réinitialise la position du sorcier de manière aléatoire dans la galerie, en veillant à ce que la nouvelle position ne se trouve pas dans un mur.

Méthode testRegression :

Cette méthode effectue des tests sur les fonctionnalités de la classe Sorcier.
Elle crée deux instances de Sorcier, teste les méthodes getPosX, getPosY, reinitialisePos et bougeAuto, puis affiche un message indiquant que les tests sont réussis.

3- Classe Basilic :

```
#include "TableauDynamique.h"
#include "Galerie.h"

/**
 * @class Sorcier
 * @brief La classe Basilic représente un serpent, personnage principale du jeu
 */
class Basilic{
    private:


    public:

        TableauDynamique corp; ///< Corps du basilic
        Vec2 direction;  ///< Direction du basilic, décrit en coordonées

        /**
         * @brief Constructeur par défaut de la classe Basilic qui initialise mord, galerie, et direction
         */
        Basilic();

        /**
        * @brief Destructeur de la classe Basilic
        */
        ~Basilic();

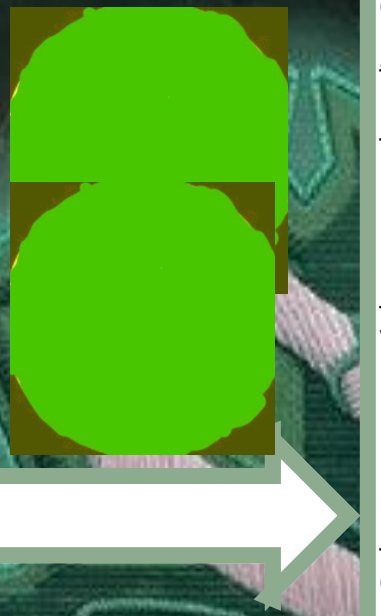
        /**
        * @brief Effectue le mouvement du Basilic dans une direction donnée
        * @param y La direction du mouvement (1: vers le haut, 2: vers la droite, 3: vers la gauche, 4: vers le bas)
        */
        void mouvementdir(unsigned int y);

        /**
        * @brief Effectue le mouvement du Basilic en fonction d'une direction et de la galerie
        * @param y La direction du mouvement (1: vers le haut, 2: vers la droite, 3: vers la gauche, 4: vers le bas)
        * @param d La galerie dans laquelle progresse le Basilic
        */
        void mouvement(unsigned int y , const Galerie & d);

        /**
        * @brief Effectue des test sur la classe Basilic pour s'assurer que toute les fonctions fonctionnes correctement
        *
        */
        static void testRegression();

        /**
         * @brief reinitialise la position
        */
        void reinitialise();

};
```



Dans l'ensemble, la classe Basilic semble représenter un objet mobile dans un contexte de jeu, capable de se déplacer dans différentes directions.

La méthode testRegression est censée être utilisée pour tester le bon fonctionnement de la classe, mais elle est actuellement vide de tests.

Basilic::Basilic() : Ceci est le constructeur par défaut de la classe Basilic. Il initialise l'objet Basilic avec une direction spécifique.

Basilic::~Basilic() : Ceci est le destructeur de la classe Basilic. Il est vide dans cet exemple car il n'y a pas de ressources dynamiques à libérer explicitement.

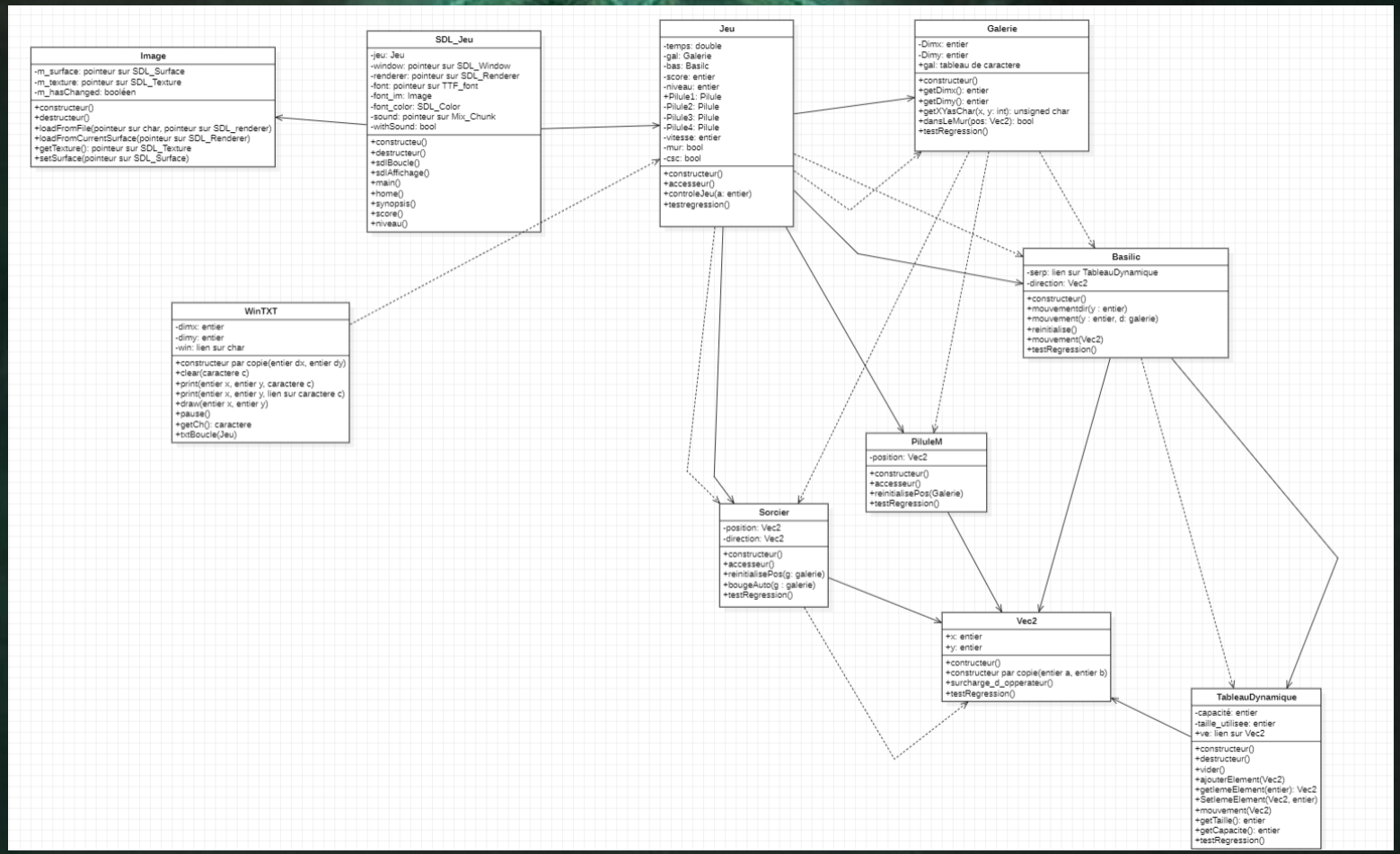
void Basilic::reinitialise() : Cette méthode réinitialise l'état de l'objet Basilic, en réinitialisant la position et la direction.

void Basilic::mouvementdir(unsigned int y) : Cette méthode déplace le Basilic dans une direction spécifique. La direction est déterminée par la valeur y. Cette méthode utilise des opérations arithmétiques pour mettre à jour les coordonnées du Basilic en fonction de la direction.

void Basilic::mouvement(unsigned int y, const Galerie& d) :
Cette méthode déplace le Basilic en fonction de la direction spécifiée par y. Elle utilise la méthode mouvementdir pour effectuer le déplacement et utilise également une référence constante à un objet Galerie, d, bien qu'elle ne l'utilise pas dans cet exemple.

void Basilic::testRegression() : Cette méthode est destinée à tester la classe Basilic. Cependant, dans cet exemple, elle ne contient aucun test actif. Elle crée simplement un objet Basilic et un objet Galerie, puis affiche un message indiquant que les tests sont OK.

Diagramme des classes mis à jour:



-- Appuie sur la touche Entrer

Diagramme de gantt mis à jour:

Tâches \ Semaines	1	2	3	4	5	6	7	8
Mise en place d'un espace de jeu (galerie)								
Mise en place d'un basilic								
Mise en place d'un sorcier simple								
Gestion des mouvements du basilic								
Mise en place d'une IA qui gère les mouvements du sorcier								
Afichage texte								
Mise en place des types de sorciers								
Gestion des conditions de jeu								
Mise en place d'un score								
Mise en place d'une mini-partie								
Mise en place des pilules magiques								
Mise en place d'un systeme de niveaux								
Mise en place de l'affichage graphique								
Tests de régréssions								
Soutenance projet								

-- Appuie sur la touche Enter --

En conclusion :

Après la finalisation du projet, nous pouvons dire que les objectifs de notre cahier des charges ont bel et bien été atteints.

Les délais sont dans les temps et chacun d'entre nous a respecté sa charge de travail et les taches qui lui ont été confiées.

Comme dans tout groupe de travail nous avons eu de légers désaccords sur certains points, mais nous avons toujours réussi à trouver des solutions et à nous mettre d'accord.

MERCI