|  |
| --- |
| Bubble Shooter |
| Rapport final |
| Amélie Frappier et Marc-Antoine Larose, gr. 4119 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Jeu de tests - TP2** | | | |
| **Description** | **Test** | **Résultat attendu** | **Résultat** |
| **Exécution du jeu (première fois)** | Démarrage du jeu | Affichage de la fenêtre *Menu principal* | OK |
|  | Création du fichier d'enregistrement des données *scores.txt* | Si le fichier *scores.txt* n'existe pas, création du fichier *scores.txt* | Non implémenté |
|  | Le fichier est enregistré au bon endroit | Devrait être créé à la racine du dossier du projet | Non implémenté |
|  | Ouverture du fichier d'enregistrement des données | Si le fichier *scores.txt* existe, ouverture/lecture du fichier *scores.txt* | Non implémenté |
|  | Affichage "Meilleur pointage : XXXXX" dans Menu principal | Dans le haut, environ au 3/4 de la fenêtre, afficher "Meilleur pointage : XXXXX" selon les données contenu dans *scores.txt* | Non implémenté |
|  | Affichage du message pour commencer une nouvelle partie | Affichage du message : "Appuyez sur Espace pour commencer" | OK: S'affiche correctement |
|  | Affichage des messages d'options | Affiché au bon endroit :  “Appuyez sur “ESPACE” pour jouer” “Appuyez sur “Q” pour Quitter” “Appuyez sur “R” pour les règlements” | OK: S'affiche correctement, situé au bon endroit |
|  | Affichage de l'image d'arrière-plan | L'arrière-plan s'affiche correctement en occupant l'entièreté de la fenêtre *Menu principal* | OK: S'affiche correctement |
| **Exécution du jeu (*Menu principal*)** | Appui sur la touche "ESPACE" | Commencement d'une nouvelle partie | OK |
|  | Appui sur la touche "Q" | Affichage du message avec choix OUI/NON:  "Voulez-vous vraiment quitter le jeu?" | OK: Quitte le jeu, mais n'a pas de message de confirmation |
|  | Appui sur la touche "C" | Affichage de la fenêtre *Classement*, disparition du menu | Classement non implémenté |
|  | Appui sur la touche "R" | Affichage de la fenêtre *Comment jouer?*, disparition du menu | OK |
| **Exécution du jeu (*Classement)*** | Lecture du fichier *scores.txt*: ERREUR | Affichage du message d'erreur correspondant à la situation **(voir test : *Ouverture du fichier de lecture*)** | Non implémenté |
|  | Lecture du fichier *scores.txt*: OK | Affichage de la fenêtre de score avec les informations | Non implémenté |
|  | Ouverture de la fenêtre *Classement* | Affichage de la fenêtre *Classement*, disparition du menu | Non implémenté |
|  | Affichage des scores | Affichage de la position, des initiales et du score obtenu | Non implémenté |
|  | Ordre des scores précédents | Ordre décroissant | Non implémenté |
| **Exécution du jeu (*Comment jouer?)*** | Ouverture de la fenêtre *Comment jouer?* | Affichage de la fenêtre *Comment jouer?*, disparition du menu | OK |
|  | Affichage correcte des informations sur les règles du jeu | Les règles sont affichées à l'emplacement désiré, sans erreur | OK |
| **Exécution du jeu (*pendant une partie*) - Espace de jeu** | L'inclinaison maximale de la bouche du canon est de -70 et 70 degrés (gauche/droite) | Lorsque l'inclinaison maximale de -70 ou 70 degrés est atteinte, aucun dépassement supplémentaire | OK |
|  | Lorsque la touche "ESPACE" du clavier est pressée | Un projectile est lancé | OK |
|  | Si la touche "ESPACE" est maintenue | Un seul projectile est lancé pour chaque appui sur "ESPACE" | OK |
|  | Si la touche "ESPACE" est pressée à répétition, rapidement | Un seul projectile est lancé pour chaque appui sur "ESPACE", à intervalle de 3 secondes |  |
|  | Si touche "ESPACE" est pressée, génération aléatoire d'une couleur pour le nouveau projectile | La couleur et les propriétés du projectile doivent changer aléatoirement | OK |
|  | Le projectile suit une direction rectiligne à vitesse stable | Il ne dévie pas et son déplacement est régulier (même vitesse) | OK |
|  | Si le projectile touche un cadre (gauche/droite) | Il "rebondit" selon un angle de -90 ou 90 degrés | OK |
|  | Si le projectile en rencontre un autre | Il se fixe sur l'autre, au point de contact | Il passe par-dessus et se fixe vers le mur |
|  | Si le projectile touche le cadre du haut | Il se fixe au point de contact | Il reste effectivement en haut, mais ne reste pas dans la grille lorsque l'on appuie sur Espace à nouveau |
|  | Si 3 projectiles se touchent : vérification des bulles adjacentes | Disparition des 3 projectiles en question | Non implémenté |
|  | Disparition des projectiles adéquate : sans trace | Les points de contacts ou anciens projectiles ne sont plus visible et n'implique plus aucune collision résiduelle quelconque | Non implémenté |
|  | Si contact entre projectiles de différentes couleurs + 3 identiques | Disparition des projectiles de mêmes couleurs seulement | Non implémenté |
|  | Si tous les projectiles autour d'un autre disparaisse | Il conserve sa position, même s'il est dans le vide | Non implémenté |
|  | Si la touche "Q" est pressée dans l'espace de jeu | Mise en PAUSE du jeu Puis : Affichage du message de fermeture OUI/NON: "Êtes-vous bien certain de vouloir fermer le jeu maintenant, votre pointage ne sera pas enregistré." | Pause non implémentée en jeu |
|  | Si le choix au message de fermeture est OUI | Fermeture du jeu | Pause non implémentée en jeu |
|  | Si le choix au message de fermeture est NON | Reprise du jeu | Pause non implémentée en jeu |
|  | Si la touche "A" est pressée | Mise en PAUSE du jeu, affichage de la fenêtre *Comment jouer?*, sans effacement de l'espace de jeu | Règlements non implémentés en jeu |
|  | Appui sur la touche "A" dans la fenêtre *Comment jouer?* | Reprise du jeu | Règlements non implémentés en jeu |
| **Fin de la partie** | Si un projectile se fixe en dessous de la barrière de délimitation (ex.: 600px) | La partie est perdue | Non implémenté |
|  | Si partie perdue | Affichage du message avec choix OUI/NON: "Perdu! Voulez-vous enregistrer votre pointage?" | Non implémenté |
|  | Si le choix au message d'enregistrement est NON | Retour au menu principal | Non implémenté |
|  | Si le choix au message d'enregistrement est OUI | Affichage de la fenêtre de saisie des initiales du joueurs *Enregistrement du nouveau pointage!* | Non implémenté: le jeu n'a pas encore de fin |
| **Exécution du jeu (pendant une partie) - Affichage des informations de jeu** | Affichage de la section pointage du jeu | En haut, aligné sur la gauche de la fenêtre et au-dessus de l'espace de jeu, affichage de la section : "points:" | Non implémenté |
|  | Lorsque 3 projectiles se touchent | Accumulation des points dans la section "pointage" | Non implémenté |
|  | Lorsque 3 projectiles JAUNE se touchent | Ajout de 1000 points au pointage | Non implémenté |
|  | Lorsque 3 projectiles VERT se touchent | Ajout de 3000 points au pointage | Non implémenté |
|  | Lorsque 3 projectiles ROUGE se touchent | Ajout de 6000 points au pointage | Non implémenté |
|  | Ouverture/lecture des données de *scores.txt* | Affichage des données dans les sections : "Meilleur pointage:" | Non implémenté |
| **Fichier d'entrée (*scores.txt*)** | Appui sur la touche "C" dans le menu principal*,* **fichier trouvé, *données exploitables*** | Affichage de la fenêtre des scores précédents, avec les données du fichier | Non implémenté |
|  | Appui sur la touche "C" dans le menu principal, **fichier trouvé, *données non-lisible/inconnues*** | Affichage du message dans la fenêtre *Classement*:  "Il est actuellement impossible de lire les données de classement. Il semble que les scores sont illisibles ou corrompus." | Non implémenté |
|  | Appui sur la touche "C" dans le menu principal, **fichier introuvable** | Affichage du message dans la fenêtre *Classement*:  "Il est actuellement impossible de lire les données de classement. Il semble impossible de localiser les scores précédemment enregistrés." | Non implémenté |
|  | Appui sur la touche "C" dans le menu principal, **fichier trouvé, *vide*** | Affichage du message dans la fenêtre *Classement*:  "Aucun scores n'est encore enregistrer. Commencer une nouvelle partie afin de les afficher ultérieurement ici." | Non implémenté |
|  | Chargement de la page *Menu principal,* **fichier trouvé, *non-vide*** | Affichage dans le menu: "Meilleur pointage: " | Non implémenté |
|  | Chargement de la page *Menu principal,* **fichier *vide ou introuvable*** | Aucun affichage | Non implémenté |
| **Fichier de sortie (*scores.txt*)** | Lorsque la partie est terminée et que le choix est **OUI**, enregistrement du nouveau pointage dans *scores.txt* | Les données saisies et le pointage obtenu par le joueur sont enregistrés dans *scores.txt* | Non implémenté |
|  | Le nouvel enregistrement s'ajoute selon le tri décroissant | Chaque nouvel enregistrement s'ajoute à l'endroit approprié selon son rang décroissant *scores.txt* | Non implémenté |
|  | Le pointage enregistré est précédé par les initiales du joueur | L'enregistrement du nouveau pointage suit l'ordre suivant : "NOM" "POINTAGE" | Non implémenté |
|  | Si le fichier *scores.txt* est introuvable | Affichage du message : "Le fichier d'enregistrement est introuvable. Afin d'assurer l'enregistrement de votre dernière partie, il sera créé." | Non implémenté |
|  | Création du fichier d'enregistrement des données *scores.txt* | Si le fichier *scores.txt* n'existe pas, création du fichier *scores.txt* | Non implémenté |
|  | Le fichier *scores.txt* est enregistré au bon endroit | Devrait être créé à la racine du dossier du projet | Non implémenté |

**Code Source**

/\* En-tête du programme

=========================

Programme: Bubble Shooter

Fichier: bubbleShooterUtil

Auteur : Amélie Frappier et Marc-Antoine Larose

Date création : 23/03/2015

Date modification: 12/04/2015

Description : Fonctions utilitaires du jeu Bubble Shooter \*/

/\* Directives au pré-processeur

================================= \*/

#include <iostream>

#include <ctime>

#include "../SDL/SDL.h"

namespace BubbleShooterUtil

{

/\* Constantes du programme

=========================== \*/

extern const int SCREEN\_HEIGHT;

extern const int SCREEN\_WIDTH;

/\* Prototypes des fonctions

============================= \*/

void initVideo();

SDL\_Surface\* setupScreen(char\* title, char\* img);

SDL\_Surface\* loadBitmap(char\* imgName);

void setPosition(SDL\_Surface \*picture, SDL\_Rect &position, int posX, int posY);

void setTransparency(SDL\_Surface \*picture, int red, int green, int blue);

void updateScreen(SDL\_Surface \*screen, SDL\_Surface \*picture, SDL\_Rect &position);

void updateScreenWithSpriteSheet(SDL\_Surface \*screen, SDL\_Surface \*picture, SDL\_Rect &currentSprite, SDL\_Rect &position);

int getRandomValue(int maxValue);

double calculateDistance(int x1, int x2, int y1, int y2);

}

#include "bubbleShooterUtil.h"

using namespace std;

namespace BubbleShooterUtil

{

const int SCREEN\_HEIGHT = 600;

const int SCREEN\_WIDTH = 450;

/\* Initialise la SDL

====================== \*/

void initVideo()

{

SDL\_Init(SDL\_INIT\_EVERYTHING); //Démarrage de la SDL (charger le système vidéo)

if (SDL\_Init(SDL\_INIT\_EVERYTHING) == -1) //Si le démarrage de la librairie SDL échoue

{

//Gestion des erreurs de la librairie SDL

cout << "Erreur lors de l'initialisation de la SDL", SDL\_GetError();

exit(EXIT\_FAILURE);

}

}

/\* Initialise les paramètres de l'écran

========================================== \*/

SDL\_Surface\* setupScreen(char\* title, char\* img)

{

SDL\_Surface\* screen; //Écran à paramètrer

//Chargement de la vidéo

screen = SDL\_SetVideoMode(SCREEN\_WIDTH, SCREEN\_HEIGHT, 32, SDL\_HWSURFACE | SDL\_DOUBLEBUF);

if (screen == NULL)

{

cout << "Erreur lors de l'initialisation de la SDL" << SDL\_GetError();

exit(EXIT\_FAILURE);

}

//Modification des paramètres de la vidéo

SDL\_WM\_SetCaption(title, NULL);

SDL\_WM\_SetIcon(SDL\_LoadBMP(img), NULL);

return screen;

}

/\* Télécharge une image en format .bmp

========================================== \*/

SDL\_Surface\* loadBitmap(char\* imgName)

{

SDL\_Surface\* loadedImage = NULL; //stocke temporairement l'image à charger

SDL\_Surface\* optimizedImage = NULL; //l'image optimisée en format 32-bits

loadedImage = SDL\_LoadBMP(imgName); //Charge l'image

//si l'image est bien chargée en mémoire

if (loadedImage != nullptr)

{

//optimiser l'image en format 32-bits

optimizedImage = SDL\_DisplayFormat(loadedImage);

//libérer la surface de l'image temporaire de la mémoire

SDL\_FreeSurface(loadedImage);

}

return optimizedImage;

}

/\* Paramètre la position d'une image dans l'espace

==================================================== \*/

void setPosition(SDL\_Surface \*picture, SDL\_Rect &position, int posX, int posY)

{

position.x = posX;

position.y = posY;

}

/\* Met la couleur de fond d'une image en transparence

======================================================= \*/

void setTransparency(SDL\_Surface \*picture, int red, int green, int blue)

{

//Si l'image est bien chargée en mémoire

if (picture != NULL)

{

SDL\_SetColorKey(picture, SDL\_SRCCOLORKEY | SDL\_RLEACCEL, SDL\_MapRGB(picture->format, red, green, blue));

}

}

/\* Met à jour l'écran du jeu

Cette fonction ne prend pas en compte les charset

===================================================== \*/

void updateScreen(SDL\_Surface \*screen, SDL\_Surface \*picture, SDL\_Rect &position)

{

SDL\_BlitSurface(picture, NULL, screen, &position);

}

/\* Met à jour l'écran du jeu

Cette fonction prend pas en compte les charset

===================================================== \*/

void updateScreenWithSpriteSheet(SDL\_Surface \*screen, SDL\_Surface \*picture, SDL\_Rect &currentSprite, SDL\_Rect &position)

{

SDL\_BlitSurface(picture, &currentSprite, screen, &position);

}

/\* Assigne et retourne une valeur aléatoire

============================================ \*/

int getRandomValue(int maxValue)

{

srand(time(NULL)); //initialise le générateur de nombres aléatoires

int random;

random = (rand() % maxValue); //génère un nombre aléatoire

return random;

}

/\* Calcule la distance entre deux points (Théorème de Pythagore)

================================================================== \*/

double calculateDistance(int x1, int x2, int y1, int y2)

{

return sqrt((double)pow(x2 - x1, 2) + (double)pow(y2 - y1, 2));

}

}

/\* En-tête du programme

=========================

Programme: Bubble Shooter

Fichier: canon.h

Auteur : Amélie Frappier et Marc-Antoine Larose

Date création : 23/03/2015

Date modification: 12/04/2015

Description : Classe Canon du jeu Bubble Shooter \*/

/\* Directives au pré-processeur

================================= \*/

#include "bubbleShooterUtil.h"

#include "../SDL/SDL.h"

#include "../SDL/SDL\_rotozoom.h"

using namespace BubbleShooterUtil;

/\* Définition de la classe

================================= \*/

class Canon

{

private:

const int MAX\_CANON\_ANGLE = 60; //angle maximal où le canon peut aller

const int MIN\_CANON\_ANGLE = -60; //angle minimal où le canon peut aller

SDL\_Surface \*originalSprite; //image originale du canon

SDL\_Surface \*displayedSprite; //image du canon transformée par la fonction rotozoom

SDL\_Rect position; //position du canon dans l'espace

int angle; //angle du canon

public:

//orientation (droite ou gauche) du canon

enum Orientation

{

RIGHT,

LEFT

};

//Constructeur et destructeur

Canon();

~Canon();

//Fonctions utilitaires

void update(SDL\_Surface \*screen);

void rotate(Orientation dir);

int getWidth();

int getHeight();

//Getters

SDL\_Rect getPosition();

int getRotationAngle();

};

#include "canon.h"

/\* Constructeur du canon

========================= \*/

Canon::Canon()

{

originalSprite = loadBitmap("canon.bmp");

displayedSprite = originalSprite;

angle = 0;

}

/\* Destructeur du canon

========================= \*/

Canon::~Canon()

{

//Libérer l'image du canon de la mémoire

SDL\_FreeSurface(displayedSprite);

SDL\_FreeSurface(originalSprite);

}

/\* Affiche le canon à l'écran

============================================ \*/

void Canon::update(SDL\_Surface \*screen)

{

setPosition(displayedSprite, position, (SCREEN\_WIDTH - displayedSprite->w) / 2, (SCREEN\_HEIGHT - displayedSprite->h / 2));

setTransparency(displayedSprite, 0, 0, 0);

updateScreen(screen, displayedSprite, position);

}

/\* Retourne la largeur du sprite du canon

============================================ \*/

int Canon::getWidth()

{

return displayedSprite->w;

}

/\* Retourne la hauteur du sprite du canon

============================================ \*/

int Canon::getHeight()

{

return displayedSprite->h;

}

/\* Tourne l'image du canon selon la direction

=============================================== \*/

void Canon::rotate(Orientation dir)

{

switch (dir)

{

case Canon::RIGHT:

if (angle >= MIN\_CANON\_ANGLE) //si l'angle du canon est plus grand que l'angle minimal, tourner le canon à droite

{

angle -= 3;

displayedSprite = rotozoomSurface(originalSprite, angle, 1.0, 0);

}

break;

case Canon::LEFT: //si l'angle du canon est plus petit que l'angle maximal, tourner le canon à gauche

if (angle <= MAX\_CANON\_ANGLE)

{

angle += 3;

displayedSprite = rotozoomSurface(originalSprite, angle, 1.0, 0);

}

break;

default:

break;

}

}

/\* Retourne la position du canon

============================================ \*/

SDL\_Rect Canon::getPosition()

{

return position;

}

/\* Retourne l'angle de rotation du canon

============================================ \*/

int Canon::getRotationAngle()

{

return angle;

}

/\* En-tête du programme

=========================

Programme: Bubble Shooter

Fichier: bubble.h

Auteur : Amélie Frappier et Marc-Antoine Larose

Date création : 23/03/2015

Date modification: 12/04/2015

Description : Classe Bubble du jeu Bubble Shooter \*/

/\* Directives au pré-processeur

================================= \*/

#include "bubbleShooterUtil.h"

#include "../SDL/SDL.h"

using namespace BubbleShooterUtil;

#ifndef BUBBLE\_HEADER

#define BUBBLE\_HEADER

/\* Définition de la classe

=========================== \*/

class Bubble

{

private:

static const int BUBBLE\_RADIUS = 23; //rayon de la bulle en pixels

static const int SPEED\_BUBBLE = 6; //vitesse de l'hypothénuse de la balle

const double PI = 3.14159265358979323846; //valeur de pi

static const int NB\_COLORS = 3; //nombre de couleurs que peut avoir une bulle

SDL\_Surface \*spriteSheet; //charset des bulles

SDL\_Rect sprite[3]; //sprite individuel de la bulle

SDL\_Rect position; //position de la bulle dans l'espace

int color; //couleur de la bulle (0 - vert, 1 - rouge, 2 - jaune)

int velocityX; //velocité en X

int velocityY; //velocité en Y

bool inGameStatus; //identifie si la bulle est en-jeu (vrai) ou si elle me l'est pas (faux)

public:

/\* Structures du programme

=========================== \*/

struct Circle

{

int centerX; //position du centre du cercle en X

int centerY; //position du centre du cercle en Y

int radius; //rayon du cercle

};

Circle hitbox; //hitbox circulaire de la bulle

//Constructeur et destructeur

Bubble();

Bubble(int posX, int posY);

~Bubble();

//Setters

void setSprite(int colorValue);

void setHitbox(SDL\_Rect &position, int radius);

void setVelocity(int velX, int velY);

void setVelocity(int angle);

void setPosition(int x, int y);

void setPosition(SDL\_Rect otherPosition);

void setInGameStatus(bool status);

//Fonctions utilitaires

void move();

void initCharset();

bool checkCollisions(Bubble\* otherBubble);

void update(SDL\_Surface \*screen);

bool hasCollidedWithWall();

//Getters

SDL\_Surface\* getSpriteSheet();

SDL\_Rect getPosition();

int getWidth();

int getHeight();

int getColor();

int getVelocityX();

int getVelocityY();

bool getInGameStatus();

Circle getHitbox();

};

#endif

#include "bubble.h"

/\* Constructeur de la classe "Bubble"

====================================== \*/

Bubble::Bubble() //constructeur par défaut

{

inGameStatus = false;

velocityX = 0;

velocityY = 0;

position.w = 47;

position.h = 47;

}

Bubble::Bubble(int posX, int posY) //constructeur assignant une position donnée à la bulle

{

inGameStatus = false;

position.x = posX;

position.y = posY;

velocityX = 0;

velocityY = 0;

position.w = 47;

position.h = 47;

}

/\* Destructeur de la classe "Bubble"

====================================== \*/

Bubble::~Bubble()

{

//Libérer la surface de la bulle seulement si elle n'Eest pas déjà en jeu

if (inGameStatus)

{

SDL\_FreeSurface(spriteSheet);

}

}

/\*Retourne la position d'une bulle donnée

========================================== \*/

SDL\_Surface\* Bubble::getSpriteSheet()

{

return spriteSheet;

}

/\* Retourne la position d'une bulle donnée

============================================ \*/

SDL\_Rect Bubble::getPosition()

{

return position;

}

/\* Retourne la couleur d'une bulle donnée

========================================== \*/

int Bubble::getColor()

{

return color;

}

/\* Retourne la largeur d'une bulle donnée

========================================== \*/

int Bubble::getWidth()

{

return position.w;

}

/\* Retourne la hauteur d'une bulle donnée

========================================== \*/

int Bubble::getHeight()

{

return position.h;

}

/\* Retourne la veleur indiquant si une bulle donnée a explosé (et est donc inaccessible)

========================================================================================= \*/

bool Bubble::getInGameStatus()

{

return inGameStatus;

}

/\* Retourne la vitesse en X d'une bulle donnée

=============================================== \*/

int Bubble::getVelocityX()

{

return velocityX;

}

/\* Retourne la vitesse en Y d'une bulle donnée

=============================================== \*/

int Bubble::getVelocityY()

{

return velocityY;

}

/\* Retourne la hitbox d'une bulle donnée

=============================================== \*/

Bubble::Circle Bubble::getHitbox()

{

return hitbox;

}

/\* Assigne à la bulle un sprite donné et une couleur

===================================================== \*/

void Bubble::setSprite(int colorValue)

{

spriteSheet = loadBitmap("bubble.bmp");

color = colorValue;

initCharset();

inGameStatus = true;

}

/\* Calcule et assigne à la bulle une vitesse en X et en Y

============================================================= \*/

void Bubble::setVelocity(int velX, int velY)

{

velocityX = velX;

velocityY = velY;

}

/\* Calcule et assigne à la bulle une vitesse en X et en Y selon l'angle du canon

============================================================= \*/

void Bubble::setVelocity(int angle)

{

double angleRadian = (angle + 180) \* PI / 180; //l'angle en radian est donné par l'angle du canon

//auquel on applique une rotation de 180 degrés,

//considérant que rotozoom tourne dans le sens horaire.

velocityX = SPEED\_BUBBLE \* sin(angleRadian);

velocityY = SPEED\_BUBBLE \* cos(angleRadian);

}

/\* Met en place la hitbox de la bulle

================================================= \*/

void Bubble::setHitbox(SDL\_Rect &position, int radius)

{

hitbox.radius = BUBBLE\_RADIUS; //le rayon du cercle

//le centre du cercle en pixels

hitbox.centerX = position.x + BUBBLE\_RADIUS;

hitbox.centerY = position.y + BUBBLE\_RADIUS;

}

/\* Bouge une bulle dans l'espace et vérifie s'il y a des collisions

==================================================================== \*/

void Bubble::move()

{

//si la bulle n'a pas de vélocité, elle ne bouge pas.

if (velocityX == 0 && velocityY == 0)

{

return;

}

//si la bulle touche un mur, inverser la vélocité en X pour la faire aller de l'autre sens.

if (hasCollidedWithWall())

{

velocityX \*= -1;

}

position.x += velocityX;

position.y += velocityY;

}

/\* Affiche la bulle à une position donnée dans l'écran

======================================================== \*/

void Bubble::update(SDL\_Surface \*screen)

{

if (inGameStatus) //si les bulles sont bien en jeu

{

move();

setHitbox(position, BUBBLE\_RADIUS);

setTransparency(spriteSheet, 255, 255, 255);

updateScreenWithSpriteSheet(screen, spriteSheet, sprite[color], position);

}

}

/\* Vérifie la collision entre deux bulles

=========================================== \*/

bool Bubble::checkCollisions(Bubble\* otherBubble)

{

double distance; //la distance entre deux bulles

if (inGameStatus)

{

//Calcule la distance entre les centres du cercle

distance = calculateDistance(hitbox.centerX, otherBubble->hitbox.centerX, hitbox.centerY, otherBubble->hitbox.centerY);

//Il y a collision si la distance entre les centres du cercles est plus petite que la somme de leurs rayons

if (distance <= (hitbox.radius + otherBubble->hitbox.radius))

{

return true;

}

}

return false;

}

/\* Vérifie la collision entre une bulle et le mur

=================================================== \*/

bool Bubble::hasCollidedWithWall()

{

//Si elle touche le haut de l'écran, la vélocité en Y devient 0

if (position.y <= 0)

{

velocityY = 0;

velocityX = 0;

return true;

}

//Vérifie si la bulle touche les murs de côté et retourne si la condition est vraie ou fausse

return (position.x + position.w >= SCREEN\_WIDTH) || (position.x <= 0);

}

/\* Assigne une position à une bulle donnée selon des valeurs passées en paramètre

=================================================================================== \*/

void Bubble::setPosition(int x, int y)

{

position.x = x;

position.y = y;

}

/\* Assigne une position à une bulle donnée à partir de la position d'un autre objet

=================================================================================== \*/

void Bubble::setPosition(SDL\_Rect otherPosition)

{

position.x = otherPosition.x;

position.y = otherPosition.y;

}

/\* Assigne le statut en-jeu ou non à une bulle donnée

======================================================= \*/

void Bubble::setInGameStatus(bool status)

{

inGameStatus = status;

}

/\* Initialise le charset des bulles dans le tableau sprite[]

============================================================== \*/

void Bubble::initCharset()

{

for (int x = 0; x < NB\_COLORS; x++) //pour chaque ligne de l'image

{

sprite[x].x = x \* 47; //position du posCharset sur l'axe des X

sprite[x].y = 0; //position de posCharset sur l'axe des Y

sprite[x].w = 47; //largeur en pixels du rectangle

sprite[x].h = 47; //hauteur en pixels du rectangle

}

}

/\* En-tête du programme

=========================

Programme: Bubble Shooter

Fichier: gameGrid.h

Auteur : Amélie Frappier et Marc-Antoine Larose

Date création : 23/03/2015

Date modification: 12/04/2015

Description : Classe GameGrid du jeu Bubble Shooter \*/

/\* Directives au pré-processeur

================================= \*/

#include "bubbleShooterUtil.h"

#include "../SDL/SDL.h"

#include "bubble.h"

using namespace BubbleShooterUtil;

/\* Définition de la classe

================================= \*/

class GameGrid

{

private:

static const int GRID\_HEIGHT = 10; //hauteur de la grille de jeu

static const int GRID\_WIDTH = 8; //largueur de la grille de jeu

SDL\_Rect position; //coordonnées en pixels d'une case dans la grille

Bubble bubble[GRID\_HEIGHT][GRID\_WIDTH]; //bulle contenue dans une case donnée de la grille

int offsetX; //décalage en X de la grille par rapport à la position (0,0)

public:

//Constructeur et destructeur

GameGrid(int bubbleDiameter);

void freeSurfaces();

//Fonctions utilitaires

void initPosition(int bubbleDiameter, int line, int column);

void insertLine(SDL\_Surface \*screen);

void update(SDL\_Surface \*screen);

bool checkBubbleCollisions(Bubble\* activeBubble);

bool checkActiveBubbleCollision(Bubble\* activeBubble);

bool manageCollision(Bubble\* &activeBubble);

void stickBubbleInGrid(Bubble\* &activeBubble);

};

#include "gameGrid.h"

/\* Constructeur de la classe "GameGrid"

======================================== \*/

GameGrid::GameGrid(int bubbleDiameter)

{

offsetX = 35;

for (int i = 0; i < GRID\_HEIGHT; i++) //pour chaque ligne de la grille

{

for (int j = 0; j < GRID\_WIDTH; j++) //pour chaque colonne de la grille

{

//créé une nouvelle bulle hypothétique dans la grille

bubble[i][j].setPosition(bubbleDiameter \* j + offsetX, bubbleDiameter \* i);

}

}

}

/\* Insére une ligne de bulles dans la grille

============================================== \*/

void GameGrid::insertLine(SDL\_Surface \*screen)

{

int startingPattern[GRID\_WIDTH] = {0, 2, 1, 2, 0, 0, 1, 0};

for (int i = 0; i < GRID\_WIDTH; i++) //pour chaque case de la première ligne (index 0)

{

bubble[0][i].setSprite(startingPattern[i]);

bubble[0][i].update(screen);

}

}

/\* Met à jour l'affichage des bulles dans la grille de jeu

============================================================ \*/

void GameGrid::update(SDL\_Surface \*screen)

{

for (int i = 0; i < GRID\_HEIGHT; i++) //pour chaque ligne de la grille

{

for (int j = 0; j < GRID\_WIDTH; j++) //pour chaque colonne de la grille

{

bubble[i][j].update(screen);

}

}

}

/\* S'occupe du moteur de collisions des bulles

================================================ \*/

bool GameGrid::manageCollision(Bubble\* &activeBubble)

{

if (checkActiveBubbleCollision(activeBubble)) //s'il y a effectivement collision

{

//Stoppe le mouvement de la bulle

activeBubble->setVelocity(0, 0);

//insère la bulle au bon endroit dans la grille

/\*stickBubbleInGrid(activeBubble);\*/

return true;

}

return false;

}

/\* Vérifie si la bulle active entre en collision avec une autre bulle ou bien le mur du haut

============================================================================================= \*/

bool GameGrid::checkActiveBubbleCollision(Bubble\* activeBubble)

{

bool hasCollided; //flag indiquant la collision

hasCollided = false;

SDL\_Rect activeBubblePosition = activeBubble->getPosition();

//Si la bulle est en haut de l'écran, il y a collision

if (activeBubblePosition.y <= 0)

{

hasCollided = true;

}

//Vérifier si la bulle en touche une autre dans la grille de jeu

hasCollided = checkBubbleCollisions(activeBubble);

return hasCollided;

}

/\* Vérifie s'il y a collision pour chaque bulle de la grille de jeu

==================================================================== \*/

bool GameGrid::checkBubbleCollisions(Bubble\* activeBubble)

{

bool hasCollided; //flag déterminant s'il y a bien eu collision

//initialisation des variables

hasCollided = false;

for (int i = 0; i < GRID\_HEIGHT; i++) //pour chaque ligne de la grille

{

for (int j = 0; j < GRID\_WIDTH; j++) //pour chaque colonne de la grille

{

//vérifier seulement s'il n'y a pas eu collision encore

if (!hasCollided)

{

if (bubble[i][j].checkCollisions(activeBubble))

{

hasCollided = true;

break;

}

}

}

}

return hasCollided;

}

/\* Libère les surfaces contenues dans la grille de jeu

======================================================== \*/

void GameGrid::freeSurfaces()

{

for (int i = 0; i < GRID\_HEIGHT; i++) //pour chaque ligne de la grille

{

for (int j = 0; j < GRID\_WIDTH; j++) //pour chaque colonne de la grille

{

delete &bubble[i][j];

}

}

}

/\* Place la bulle dans la grille de jeu, selon la position dans laquelle elle se trouve

========================================================================================= \*/

void GameGrid::stickBubbleInGrid(Bubble\* &activeBubble)

{

double distance; //distance entre les cases inoccupées de la grille et la bulle active

double shortestDistance = DBL\_MAX; //stocke la valeur de la distance la plus courte

int shortestDistanceLine; //ligne où se trouve la distance la plus courte

int shortestDistanceColumn; //colonne où se trouve la distance la plus courte

int newColor; //nouvelle couleur de la bulle active

SDL\_Rect activeBubblePos; //position courante de la bulle active

SDL\_Rect otherBubblePos; //position courante de la bulle à comparer

activeBubblePos = activeBubble->getPosition();

for (int i = 0; i < GRID\_HEIGHT; i++) //pour chaque ligne de la grille

{

for (int j = 0; j < GRID\_WIDTH; j++) //pour chaque colonne de la grille

{

otherBubblePos = bubble[i][j].getPosition();

bool status = bubble[i][j].getInGameStatus();

//Si la case est libre, vérifier la distance entre la bulle active et la case vide

if (!status)

{

distance = calculateDistance(activeBubblePos.x, otherBubblePos.x, activeBubblePos.y, otherBubblePos.y);

//Si la distance entre les deuxx points est la plus courte, prendre l'index correspondant en note

if (distance < shortestDistance)

{

shortestDistanceLine = i;

shortestDistanceColumn = j;

}

}

}

}

//Met la bulle active dnas la grille au bon endroit

newColor = activeBubble->getColor();

bubble[shortestDistanceLine][shortestDistanceColumn].setSprite(newColor);

}

/\* En-tête du programme

=========================

Programme: Bubble Shooter

Fichier: main.cpp

Auteur : Amélie Frappier et Marc-Antoine Larose

Date création : 23/03/2015

Date modification: 12/04/2015

Description : Dans le jeu Bubble Shooter, il faut faire des combinaisons de 3 bulles de même couleur ou plus pour les faire disparaître de la grille de jeu.

En usant des flèches, le joueur fait tourner un canon dans la direction voulue, et en appuyant sur Espace, il tire une bulle dans l'aire de jeu. En touchant une bulle,

le jeu vérifie s'il y a 3 couleurs identiques ou plus autour de lui: si c'est le cas, les bulles disparaissent. Si les bulles descendent trop bas, le jeu se termine.\*/

/\* TODO: améliorations à apporter au programme

================================================= \*/

/\* ------ Amélie ---------- \*/

//Vérifier changement de la bulleActive: il ne se fait pas correctement s'il touche le mur du haut.

//Corriger gestion des collisions pour les bulles: la grille tend à se remplir à l'envers.

//Gestion de collisions entre bulles non fonctionnelle.

//Insérer algorithme de destruction et de check des bulles adjacentes.

/\* ------ Marc-Antoine ---------- \*/

//Intégrer module des scores

//Intégrer SDL\_ttf de façon portative

/\* Directives au pré-processeur

================================= \*/

#include <math.h>

#include "../SDL/SDL.h"

#include "../SDL/SDL\_rotozoom.h"

#include "bubbleShooterUtil.h"

#include "canon.h"

#include "bubble.h"

#include "gameGrid.h"

using namespace BubbleShooterUtil;

/\* Constantes du projet

======================== \*/

const int GRID\_HEIGHT = 10;

const int GRID\_WIDTH = 8;

const int REFRESH\_TIME = 600;

const int TIME\_IN\_MILLISECONDS\_BEFORE\_NEXT\_UPDATE = 16;

/\* Prototype des fonctions

=========================== \*/

void showRules(SDL\_Surface \*screen, SDL\_Surface \*menu, SDL\_Rect &menuPos, bool &screenIsActive);

void playGame(SDL\_Surface \*screen, SDL\_Surface \*bg, SDL\_Rect &bgPos, bool &screenIsActive);

void initGameGrid(GameGrid grid[GRID\_HEIGHT][GRID\_WIDTH]);

void insertLine(SDL\_Surface \*screen, GameGrid grid[GRID\_HEIGHT][GRID\_WIDTH]);

bool checkBubbleCollisions(Bubble \*thisBubble, GameGrid grid[GRID\_HEIGHT][GRID\_WIDTH]);

/\* Méthode principale

======================= \*/

int main(int argc, char \*argv[])

{

SDL\_Surface \*screen = NULL; //fenêtre du jeu

SDL\_Surface \*background = NULL; //image de fond du jeu

SDL\_Surface \*menu\_rect = NULL; //rectangle du menu

SDL\_Surface \*game\_logo = NULL; //logo (nom) du jeu)

SDL\_Event e; //événement capté par SDL\_PollEvent

SDL\_Rect bgPosition; //position du fond d'écran dans l'espace

SDL\_Rect menuPosition; //position du menu dans l'espace

SDL\_Rect logoPosition; //position du logo dans l'espace

bool isActive = true; //flag déterminant si la fenêtre est active

//Initialisation de la SDL

initVideo();

//Paramètres de l'écran

screen = setupScreen("Bubble Shooter", "greenBubble.bmp");

//Paramètres du fond d'écran

background = loadBitmap("menu\_bg.bmp");

setPosition(background, bgPosition, 0, 0);

//Paramètres du rectangle du menu

menu\_rect = loadBitmap("rectangle\_menu.bmp");

setPosition(menu\_rect, menuPosition, 39, 256);

//Paramètres du logo

game\_logo = loadBitmap("game\_logo2.bmp");

setPosition(game\_logo, logoPosition, 50, 56);

//Mettre à jour l'écran une première fois

updateScreen(screen, background, bgPosition);

updateScreen(screen, game\_logo, logoPosition);

updateScreen(screen, menu\_rect, menuPosition);

while (isActive) //tant que l'application est active

{

SDL\_Flip(screen);

SDL\_WaitEvent(&e);

switch (e.type)

{

case SDL\_QUIT: //si le joueur clique sur le X de la fenêtre, on ferme l'application

isActive = 0;

break;

case SDL\_KEYDOWN: //si l'on appuie sur une touche quelconque

switch (e.key.keysym.sym)

{

case SDLK\_q: //si on appuie sur Q, fermer le programme

isActive = 0;

break;

case SDLK\_r: //si on appuie sur R, montrer les règles du jeu

showRules(screen, menu\_rect, menuPosition, isActive);

break;

case SDLK\_SPACE: //si on appuie sur Espace, appeler la fonction playGame();

playGame(screen, background, bgPosition, isActive);

break;

}

}

}

//Libère les surfaces de la mémoire

SDL\_FreeSurface(menu\_rect);

SDL\_FreeSurface(game\_logo);

SDL\_FreeSurface(background);

SDL\_FreeSurface(screen);

//Arrêt de la SDL (vider la mémoire)

SDL\_Quit();

return 0;

}

/\* Montre les règles du jeu

============================= \*/

void showRules(SDL\_Surface \*screen, SDL\_Surface \*menu, SDL\_Rect &menuPos, bool &screenIsActive)

{

SDL\_Surface \*rules = NULL; //règlements du jeu

SDL\_Event e; //événement capté par SDL\_WaitEvent

SDL\_Rect rulesPos; //position des règles dans l'espace

bool rulesIsActive = true; //fenêtre des règles du jeu est active

//Paramètres du rectangle du menu

rules = loadBitmap("rectangle\_rules.bmp");

setPosition(rules, rulesPos, 39, 256);

//Mettre à jour l'écran une première fois

updateScreen(screen, rules, rulesPos);

while (rulesIsActive) //tant que les règlements sont actifs

{

SDL\_Flip(screen);

SDL\_WaitEvent(&e);

switch (e.type)

{

case SDL\_QUIT: //si le joueur clique sur le X de la fenêtre, on ferme l'application EN ENTIER

rulesIsActive = 0;

screenIsActive = 0;

break;

case SDL\_KEYDOWN: //si l'on appuie sur une touche quelconque

switch (e.key.keysym.sym)

{

case SDLK\_q: //si on appuie sur Q, fermer les règles et montrer le menu à nouveau

updateScreen(screen, menu, menuPos); //Redessiner le menu sur l'écran

rulesIsActive = 0;

break;

}

}

}

SDL\_FreeSurface(rules); //Libérer les règles de la mémoire

}

/\* Démarre le jeu en tant que tel

===================================== \*/

void playGame(SDL\_Surface \*screen, SDL\_Surface \*bg, SDL\_Rect &bgPos, bool &screenIsActive)

{

SDL\_Event moveE; //événement capté par SDL\_PollEvent

int randomColor; //couleur choisie au hasard

int newColor; //nouvelle couleur assignée à une bulle donnée

bool gameIsActive; //le jeu est actif

bool changeBubble; //détermine si l'on doit changer de bulle ou non

GameGrid\* bubbleGrid = new GameGrid(47); //grille de jeu contenant les bulles

Canon\* canon = new Canon(); //le canon tirant les bulles

Bubble\* activeBubble = new Bubble(SCREEN\_WIDTH / 2, SCREEN\_HEIGHT - 25); //la bulle active

Bubble\* nextBubble = new Bubble(SCREEN\_WIDTH / 2 + 47, SCREEN\_HEIGHT - 25); //la prochaine bulle en jeu

//Initialisation des variables

gameIsActive = true;

//Redessiner l'écran avec le background pour faire disparaître le menu

updateScreen(screen, bg, bgPos);

//Insérer une ligne de bulles dans la grille

bubbleGrid->insertLine(screen);

//Dessiner une première fois le canon, la bulle active et la prochaine bulle en jeu

canon->update(screen);

randomColor = getRandomValue(3);

activeBubble->setSprite(randomColor);

activeBubble->update(screen);

randomColor = getRandomValue(3);

nextBubble->setSprite(randomColor);

nextBubble->update(screen);

SDL\_EnableKeyRepeat(50, 50);

//prend le temps écoulé en millisecondes

long nextUpdateTimeInMilliseconds = clock();

while (gameIsActive) //tant que le jeu est actif

{

//S'assure que le jeu roule à 60 Frames par Secondes (FPS)

while (nextUpdateTimeInMilliseconds > clock());

nextUpdateTimeInMilliseconds += TIME\_IN\_MILLISECONDS\_BEFORE\_NEXT\_UPDATE;

//Charger l'écran avec toutes les images précédemment "blittées"

SDL\_Flip(screen);

SDL\_PollEvent(&moveE); //événements de type Poll

switch (moveE.type)

{

case SDL\_QUIT: //si le joueur clique sur le X de la fenêtre, on ferme l'application EN ENTIER

gameIsActive = 0;

screenIsActive = 0;

break;

case SDL\_KEYDOWN: //si l'on appuie sur une touche quelconque

switch (moveE.key.keysym.sym)

{

case SDLK\_LEFT: //tourne le canon à gauche avec la flèche de gauche

canon->rotate(Canon::LEFT);

break;

case SDLK\_RIGHT: //tourne le canon à droite avec la flèche de droite

canon->rotate(Canon::RIGHT);

break;

case SDLK\_SPACE: //lance la bulle lorsque l'on appuie sur Espace

activeBubble->setPosition(canon->getPosition());

activeBubble->setVelocity(canon->getRotationAngle());

//Vérifie si la bulle entre en collision

changeBubble = bubbleGrid->manageCollision(activeBubble);

//Si la bulle active doit être changée, la prochaine bulle devient la bulle active

//et une nouvelle prochaine bulle prend sa place.

if (changeBubble)

{

//insère la bulle au bon endroit dans la grille

/\*bubbleGrid->stickBubbleInGrid(activeBubble);\*/

//la bulle active devient la prochaine bulle

newColor = nextBubble->getColor();

activeBubble->setSprite(newColor);

//la prochaine bulle devient une nouvelle bulle

randomColor = getRandomValue(3);

nextBubble = new Bubble(SCREEN\_WIDTH / 2 + 47, SCREEN\_HEIGHT - 25);

nextBubble->setSprite(randomColor);

}

break;

}

break;

}

//Mettre à jour l'écran

updateScreen(screen, bg, bgPos);

//Mettre à jour la grille des bulles

bubbleGrid->update(screen);

//Mettre à jour le canon, la bulle active et la prochaine bulle

canon->update(screen);

activeBubble->update(screen);

nextBubble->update(screen);

}

//Libère les surfaces de la grille de jeu

bubbleGrid->freeSurfaces();

delete canon;

delete activeBubble;

delete nextBubble;

}

**Réflexion sur le projet**

**Amélie:**

Le jeu Bubble Shooter était définitivement très ambitieux et bien plus difficile que je ne l'aurais cru, même pour quelqu'un se considérant assez forte en programmation. Bien que j'ai beaucoup apprécié l'usage de la SDL, et aie appris beaucoup par rapport à l'implémentation correcte des classes ainsi que de l'usage des pointeurs, il fut difficile de le réaliser entièrement dans les temps donnés. En effet, on constate que le score n'a pas été implémenté dans le code final, que les conditions de défaite n'ont pas été implémentées, et que l'algorithme gérant les bulles adjacentes et la destruction de bulles n'est pas fonctionnel. Ainsi, le jeu est essentiellement incomplet et sans fin. Je suis toutefois fière de ce qui a été accompli jusqu'à présent, notamment au niveau de la gestion des collisions et du mouvement (bulles et rotation du canon), de la clarté de l'interface ainsi que de l'implémentation des classes, rendant entre autres le *main* clair et simple à lire.

J'aurais du prendre plus de temps à penser à la structure de mes classes ainsi que notre algorithme. J'ai du faire beaucoup de *refactoring*, car mon code devenait difficile à lire autrement, et j'ai du refaire ma structure et mes classes de nombreuses fois avant d'avoir un code lisible et aisément déboguable. Il aurait été plus judicieux de faire des schémas sur papier avant de me lancer les deux pieds joints dans le code. De plus, j'ai fini par faire la grande majorité du code seule malgré moi: en effet, Marc-Antoine ayant eu des difficultés au niveau de l'intégration du score ainsi que des imprévus majeurs au niveau personnel, j'ai pris sur mes épaules la grande majorité du code. Je m'attendais à ce qu'une telle situation puisse arriver: il n'en reste pas moins que le tout a été excessivement exigeant au niveau physique et mental, considérant les travaux des autres cours ainsi que les entrevues pour les stages ATÉ.

Si c'était à refaire, je prendrais un jeu moins compliqué à coder, non seulement afin de pouvoir mieux respecter les échéanciers et gérer les imprévus et autres bogues, et aussi moins me casser la tête au niveau de la structure des classes, mais aussi pour permettre à mon coéquipier de pouvoir en faire plus et de ne pas avoir une charge de travail débalancée. Nous aurions pu opter pour un jeu tel que Bonhomme Pendu ou Battleship, par exemple. Reste que j'ai tout de même apprécié de travailler sur un tel projet et avec la SDL, qui, avec de nouvelles librairies, possède une multitude d'options, j'ai appris énormément de choses, et je retravaillerais sur un jeu comme tel n'importe quand avec les connaissances nouvellement acquises.

**Marc-Antoine:**