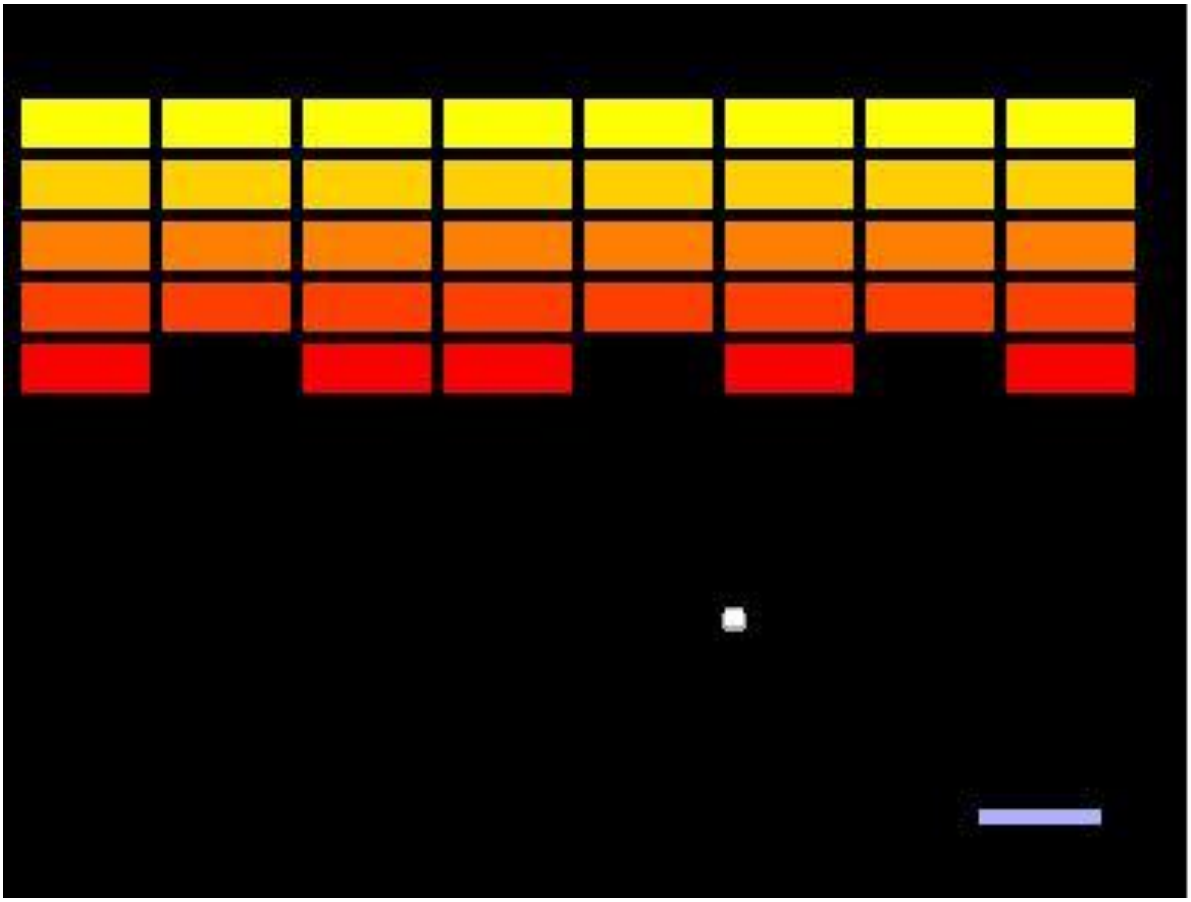
CASSE-BRIQUES



RAPPORT TECHNIQUE

DOCUMENTATION INTÉGRÉE AU CODE SOURCE

BROQUET RAYAN

EMT-DIVTEC

Table des matières

Description du projet

Mon projet est un simple casse-briques fait en 2D qui gère les collisions, les vies du joueur et la balle qui s'y déplace pour détruire les blocs. Un menu pour commencer, redémarrer ou quitter le jeu est présent dans le projet.

Convention de nommage

Les variables

Les variables ont 2 types de nommages, sois ***m_maVariable*** qui représente une variable membre au sein du programme, sois le deuxième type de variable qui sont les pointers, exemple : ***m_pMaVariable*** qui représente un pointer comme un Sprite ou un GameScene.
Après les différents préfixes, commencent par une majuscule et sont camélisées

Ordre des préfixes :

'***m_***', et finalement '***p***'

Les booléens

Les booléens sont nommés comme les variables membres et peuvent avoir le mot ***is*** devant le nom si ça représente un état du jeu, exemple : bool m_isDead = false;

Les Constantes

Les constantes sont définies en haut de page, par exemple dans le fichier GameCore.cpp et sont écrites en majuscule suivies d'un « _ » pour séparer les mots, exemple : const int BRICK_SIZE = 15;

Les fonctions

Les fonctions sont définies comme une variable lambda, elle a son nom séparé par une majuscule entre chaque mot, exemple : void createSceneWin();

Eléments techniques

Les signals

J'utilise des signals durant le déroulement de mon jeu.
Ce signal me permet de vérifier que le pad ne sort pas de l'écran lors que le joueur joue avec la souris et en le replacement à la bordure si il essaie d'en sortir.

```
//! Traite le déplacement de la souris.
//! \brief GameCore::mouseMoved
//! \param newMousePosition Nouvelle position de la souris.
void GameCore::mouseMoved(QPointF newMousePosition) {
    emit notifyMouseMoved(newMousePosition);

    // Fais en sorte que le pad ne puisse pas sortir des zones de collisions
    if (newMousePosition.x() >= MIN_VALUE_WALL && newMousePosition.x() <= MAX_VALUE_WALL) {
        m_pPlayer->setX(newMousePosition.x());
        m_pPlayer->setOffset(-m_pPlayer->boundingRect().width()/2, -m_pPlayer->boundingRect().height()/2);
    }
}
```

Le code ci-dessous me permet de gérer les clics de l'utilisateur sur mes sprites boutons sur l'écran du menu. Tout le reste du code est expliqué via les commentaires.

```
//! Traite l'appui sur un bouton de la souris.
//! \brief GameCore::mouseButtonPressed
//! \param mousePosition Position de la souris.
//! \param buttons Bouton de la souris.
void GameCore::mouseButtonPressed(QPointF mousePosition, Qt::MouseButtons buttons) {
    emit notifyMouseButtonPressed(mousePosition, buttons);
    m_onClick = true;

    // Test si l'utilisateur fait un clique gauche et qu'il est sur le menu.
    if (buttons.testFlag(Qt::LeftButton) && m_pGameCanvas->currentScene() == m_pSceneMenu) {

        // Si oui, vérifie si il clique sur le bouton reprendre le jeu .
        if (m_pButtonResume == m_pSceneMenu->spriteAt(mousePosition)) {
            m_pGameCanvas->setCurrentScene(m_pSceneGame);

            // Sinon vérifie si il clique sur recommencer une partie
        } else if (m_pButtonRestart == m_pSceneMenu->spriteAt(mousePosition)) {
            // Recréer la scène de jeu et réinitialise les valeurs.
            m_pButtonRestart->setOpacity(0.7);
            m_isRestart = true;
            m_pSceneGame = nullptr;
            m_pSceneGame = m_pGameCanvas->createScene(0, 0, SCENE_WIDTH, SCENE_WIDTH / GameFramework::screenRatio());
            m_pSceneGame->setBackgroundColor(m_colorBackGround);

            // Réinitialise les valeurs pour que les blocs ne soient pas décalés et repositionne la boule sur le pad
            m_spaceColumns = 0;
            m_counterBlock = 54;
            m_isWaiting = true;

            createBlock();
            createPlayer();
            createLifePlayer();
            setupBouncingArea();
            m_pGameCanvas->setCurrentScene(m_pSceneGame);
            // Sinon vérifie si il clique sur le bouton quitter.
        } else if (m_pButtonLeave == m_pSceneMenu->spriteAt(mousePosition)) {
            QCoreApplication::quit();
        }
    }
    m_pButtonResume->setOpacity(1);
    m_pButtonRestart->setOpacity(1);


}
```

Le code suivant gère si le joueur appuie sur une touche, exemple si l'utilisateur appuie sur la touche Echap, il est emmené au menu du jeu.

```
//! Traite la pression d'une touche.
//! \brief GameCore::keyPressed
//! \param key Numéro de la touche (voir les constantes Qt).
void GameCore::keyPressed(int key) {

    emit notifyKeyPressed(key);

    switch(key) {
    // Si le joueur presse la flèche de gauche
    case Qt::Key_Left:
        if(m_pPlayer->left()){
            m_pPlayer->setX(m_pPlayer->x() - 40);
        }
        break;

    // Si le joueur press la fléche de droite
    case Qt::Key_Right:
        if(m_pPlayer->right() < m_pSceneGame->width() - 10) {
            m_pPlayer->setX(m_pPlayer->x() + 40);
        }
        break;

    // Si le joueur presse Espace
    case Qt::Key_Space:
        m_keySpacePressed = true;
        break;
    // Si le joueur presse Echap
    case Qt::Key_Escape:
        // Retour au menu si l'utilisateur presse la touche Esc.
        m_pGameCanvas->setCurrentScene(m_pSceneMenu);
        m_textMenuResume->setText("Reprendre");
    }
}
```

Les slots

J'utilise un slot pour faire la connexion entre les blocs détruits et ma variable qui compte le nombre de blocs. Pour se faire, j'utilise la fonction onSpriteDestroyed() définit dans GameCore.h

Initialisation du jeu

```
//! Initialise le contrôleur de jeu.
//! \brief GameCore::GameCore
//! \param pGameCanvas  GameCanvas pour lequel cet objet travaille.
//! \param pParent      Pointeur sur le parent (afin d'obtenir une destruction automatique de cet objet).
GameCore::GameCore(GameCanvas* pGameCanvas, QObject* pParent) : QObject(pParent) {

    // Mémorise l'accès au canvas (qui gère le tick et l'affichage d'une scène)
    m_pGameCanvas = pGameCanvas;

    // Création des différentes scènes du jeu.
    createSceneGame();
    createSceneMenu();
    createSceneWin();
    createSceneLoss();

    // Création de la zone de rebond.
    setupBouncingArea();

    // Création des blocs à détruire.
    createBlock();

    // Création du joueur (le rectangle).
    createPlayer();

    // Création des sprites coeurs vie du joueur.
    createLifePlayer();

    // Création du background pour le menu du jeu.
    setupBackGroundMenu();

    // Création des boutons pour le menu.
    createButton();

    // Définis la scène actuelle du jeu (le menu)
    m_pGameCanvas->setCurrentScene(m_pSceneMenu);

    // Démarre le tick pour que les animations qui en dépendent fonctionnent correctement.
    m_pGameCanvas->startTick();

}
```

Voici comment je créer mes différentes scènes et autres méthodes utile au fonctionnement de mon jeu, ce qu'il est important à retenir est le ***m_pGameCanvas->startTick();*** qui va démarrer le tick (la cadence) du jeu pour permettre à la boule et aux différents éléments de fonctionner et de se déplacer, sans cela, le jeu resterait immobile et rien ne se passerait.