

Module Programmation de spécialité *C++*

Travaux Pratiques 5

Programmation Objet - Mini Jeu ASCII Casse Brique

Objectif: L'objectif de ce TP est de mettre en pratique les notions de classe et de programmation C++ à travers la réalisation d'un mini jeu de casse brique en mode ASCII.

Partie I: Mise en place et affichage d'écran

Ce programme ne fonctionnera malheureusement pas correctement dans l'environnement de visual studio code mais cela sera l'occasion de pratiquer la compilation en ligne de commandes. En particulier, le TP pourra être réalisé simplement avec un éditeur de texte et un compilateur g++ disponible à travers l'utilisation de WSL.

1.1 Récupérez le squelette du programme disponible à l'adresse ci-dessous et vérifiez que votre programme compile bien en suivant les instructions de votre enseignant par rapport à l'utilisation de WSL.

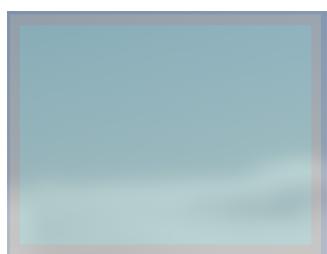
<https://moodle.univ-lyon2.fr/mod/resource/view.php?id=180865>

1.2 Compétez le constructeur de la classe **EcranASCII** de façon à ce que les attributs **monLargeur**, **monHauteur** et **monMatAffichage** soient bien initialisés. L'attribut **monMatAffichage** devra contenir **monLargeur * monHauteur** éléments de type **TypePixels::Fond**.

1.3 Dans le fichier contenant la fonction **main**, construisez un écran et affichez le en appelant la méthode **rafraichir()**. Vous devriez obtenir une image comparable à l'image (a) ci-dessous.

1.4 Complétez la méthode **reglePixel()** de façon à ce qu'elle puisse changer le pixel dont les coordonnées sont données en paramètres. La méthode devra vérifier si les coordonnées sont bien à l'intérieur de l'image et dans le cas contraire le méthode ne fera rien. Dans votre fichier principal, testez la méthode en affichant les pixels de coordonnées (1, 1), (3, 1), (3, 3), 1, 3 et (*largeur*/2, *hauteur* – 2).

1.5 Complétez votre méthode de façon à tester l'affichage avec des largeurs et hauteurs différentes. Par exemple, un écran défini avec une taille de (20, 30) donnera un affichage comme sur l'image (c) de la figure ci-dessous.



(a) base ecran 30x20



(b) test pixels (30x20)



(c) test ecran (20x30)

1.6 Déplacez le contenu de votre fonction `main` dans une nouvelle fonction `testEcran()` et appelez la fonction pour avoir le même comportement que précédemment.

1.7 Dans la classe `EcranASCII`, déclarez et implémentez une nouvelle méthode `efface()`, qui mettra tous les pixels de l'écran à la valeur *TypePixels : :Fond*.

1.8 En revenant dans votre fichier principal, dans la fonction `testEcran()`, rajoutez des instructions de façon à faire déplacer un pixel horizontalement au cours du temps. Le pixel devra effectuer une distance de 50 pixels en revenant au début de la ligne lorsque sa coordonnées dépasse les limites de l'écran. **Indication :** Il s'agira d'utiliser les méthodes `efface()`, `reglePixel()`, et `rafraichir()`. Pour temporiser l'affichage vous pourrez utiliser la fonction `usleep()`.

Partie II: Gestion de base déplacement du joueur

Dans cette partie, il s'agit de gérer les événements du clavier afin de pouvoir déplacer ensuite le joueur.

2.1 Testez d'appeler la fonction `jeuBouclePrincipale()` déjà écrite de votre programme principale et vérifiez que les touches a et z sont bien détectées et que les autres touches ne sont pas prises en compte. Améliorez cette fonction de façon à ce qu'elle prenne en compte un appuis sur la touche x permettant d'arrêter la boucle de la fonction.

2.2 Afin de préparer la gestion de l'affichage du joueur, dans la classe `EcranASCII`, rajoutez une variable représentant le symbole du joueur `monSymboleJoueur`. Vous pourrez choisir le code '44m' dans la chaîne de caractères associée à la couleur. Toujours dans cette classe, dans l'énumération TypePixels, rajoutez un nouvel élément nommé `Joueur` et enfin gér ce type de pixel dans la méthode `rafraichir()`.

2.3 Récupérez la base de la classe `CasseBrique` disponible sur *moodle* et complétez le constructeur de façon à initialiser la position du joueur au centre en bas de la fenêtre.

2.4 Dans la classe `CasseBrique`, complétez les fonctions `deplaceJoueurDroite()` et `deplaceJoueurGauche()`. Ces deux fonctions devront simplement calculer les nouvelles coordonnées du plateau et vérifier qu'un morceau du plateau du joueur ne dépasse pas des limites de l'écran du joueur.

2.5 Toujours dans la classe `CasseBrique`, complétez la méthode `miseAJour()` qui devra effacer l'écran puis afficher le plateau du joueur qui sera de taille 3. Testez cette fonction dans votre fonction main en créant un écran, un objet `CasseBrique` et en appelant plusieurs fois la méthode `deplaceJoueurGauche()` en faisant une pause pour voir le déplacement.

2.6 Dans la fonction `jeuBouclePrincipale()`, procéder comme pour la question précédente pour afficher le joueur qui puisse se déplacer à partir des touches a et z.

Partie III: Gestion de base de la balle

3.1 Création d'une structure de type `Balle` qui stockera la position en x et y (en coordonnées flottantes) et deux flottants représentant un vecteur de direction `vDirX` et `vDirY`. Dans la classe `CasseBrique` rajoutez une balle et l'initialiser en utilisant le centre de l'écran comme position et le vecteur de coordonnées (1, -1) (l'initialisation doit être faite dans la méthode `CasseBrique::initJoueur()`).

3.2 Dans la méthode `miseAJour()` de la classe `CasseBrique`, après l'affichage du joueur, mettez à jour les coordonnées de la balle en additionnant simplement les composantes du vecteur vitesse avec la position de la balle. Puis affichez la balle à partir de ses coordonnées.

3.3 Afin de gérer les rebonds de la balle, toujours dans la classe `CasseBrique`, rajoutez une méthode `miseAJourVectVitesse()`, qui prendra comme paramètre une référence d'un objet de type `Balle` et qui mettra à jour son vecteur vitesse en fonction de la localisation de la balle. Appelez cette fonction depuis la méthode `miseAJour()` et vérifiez que votre balle rebondit bien sur le cadre du jeu.

3.4 Complétez la méthode précédente (`miseAJourVectVitesse()`) de façon à faire aussi rebondir la balle sur le plateau du joueur. Testez le bon comportement de la balle par rapport au joueur.



Partie IV: Ajout et gestion des briques

4.1 Dans la classe `CasseBrique`, rajoutez une nouvelle structure `Brique3x1` qui pourra être définie dans le fichier .h comme pour la structure `Balle` et qui possédera deux coordonnées x et y ainsi qu'une méthode collision prenant en paramètre une balle et qui renverra vrai si la balle entre en collision avec la brique et renverra faux sinon.

4.2 Rajoutez un vecteur `monVectBriques` qui stockera l'ensemble des briques actives du jeu et initialiser le dans le constructeur de la classe `CasseBrique`. Le remplissage devra être calculé en fonction de la largeur de l'écran et occupera environ un sixième du haut de l'écran.

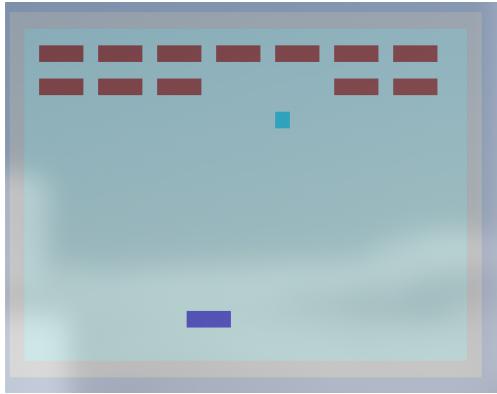
4.3 Comme cela a été fait dans la question 2.2 pour l'affichage du joueur, modifiez la classe `EcranASCII`, de façon à pouvoir gérer l'affichage des briques. Vous pourrez utiliser le code '41m' pour le choix de la couleur.

4.4 Dans la méthode `CasseBrique::miseAJour()`, complétez l'affichage de façon à afficher les briques.

4.5 Mettez à jour la méthode `CasseBrique::miseAJourVectVitesse()` pour prendre en compte la présence de brique et faire rebondir la balle en cas de présence d'une brique.

4.6 Pour gérer la disparition des briques en fonction d'une certaine résistance, rajoutez un attribut `resistance` de type entier qui sera initialisé par défaut à 1 puis mettez à jour cet attribut dans la méthode `CasseBrique::miseAJourVectVitesse()` en cas de collision avec une balle. Enfin pour finaliser l'affichage conditionnel des briques, mettez à jour la méthode `CasseBrique::miseAJour()` de façon à afficher une brique uniquement que si elle a encore de la résistance.

Si vous n'avez pas fait d'erreur, vous devriez avoir une base jouable comme sur l'illustration suivante.



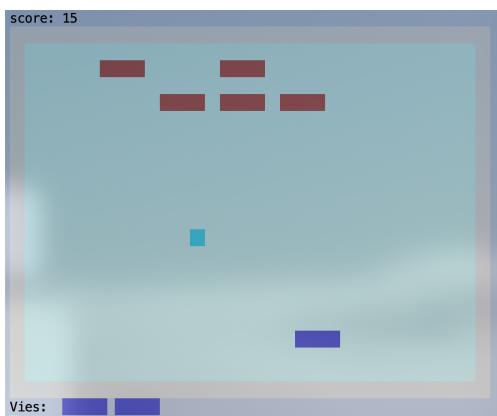
Partie V: Jeu de base avec gestion des vies

5.1 Pour détecter quand le joueur perd la balle, rajoutez dans la classe `CasseBrique` un attribut `monNbVie` initialisé à 1 avec une nouvelle méthode `gameOver()` qui renverra un booléen quand le joueur aura perdu. Dans votre programme principal, intégrez cette fonction pour arrêter le programme quand le joueur aura perdu. Pour l'instant la méthode pourra renvoyer toujours *false*.

5.2 Dans la fonction `CasseBrique::miseAJour()`, détectez quand la balle passe sous le joueur et décrémentez le nombre de vies et mettez à jour la fonction `CasseBrique::gameOver()` pour détecter la fin de jeu.

5.3 En exploitant les méthodes `EcranASCII::regleSousTitre()` et `EcranASCII::regleSurTitre()` affichez en temps réel le nombre de points du joueur tout comme son nombre de vies.

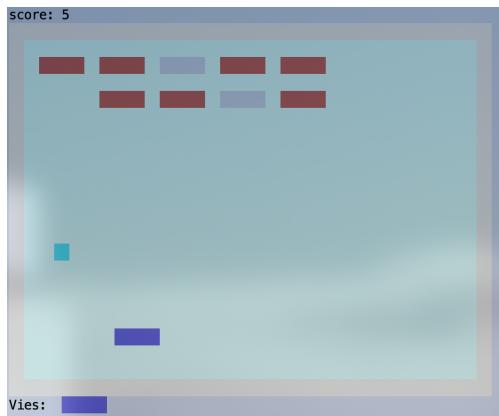
5.4 Améliorez l'affichage de façon à représenter graphiquement le nombre de vies restante comme sur l'affichage ci-dessous.



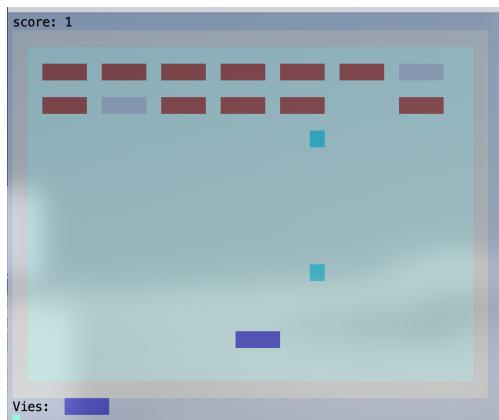
Partie VI: Jeu avancé

Dans cette partie, l'idée est d'obtenir une version un plus avancée avec l'intégration de brique bonus permettant d'avoir des balles multiples et de rajouter des niveaux.

6.1 Pour obtenir cette version améliorée, une première étape peut être de rajouter un booléen `estBonus` afin de différencier l'affichage des briques classiques et des briques bonus. Rajoutez cet attribut et modifiez l'affichage pour les briques bonus. Modifiez aussi le constructeur de façon à générer aléatoirement deux ou trois briques bonus. Vous pourrez obtenir une initialisation ressemblant à l'image ci-dessous.



- 6.2 Modifiez votre code de façon à considérer une double résistance des pièces bonus.
- 6.3 Pour préparer l'intégration de plusieurs balles dans le jeu, rajoutez un nouveau vecteur qui contiendra des balles, supprimez l'attribut `monBalle` et mettez à jour votre code de façon à prendre en compte la possibilité de gérer plusieurs balles.
- 6.4 Testez que la gestion multi-balles fonctionne bien en initialisant temporairement deux balles dans le jeu. Vous aurez alors normalement les deux balles gérées parallèlement.



- 6.5 Faire en sorte que la balle se dédouble uniquement en cas de contact avec une pièce bonus.

Partie VII: Pour ceux qui ont finis

Pour les personnes qui ont finis, de manière autonome, vous rajouterez les fonctionnalités suivantes :

- Ajouter un nouveau type de brique *malus* qui aura pour effet d'augmenter la vitesse de la balle par un coefficient 1.3.
- Ajouter une autre brique spéciale qui une fois cassée, tombera vers le bas et le joueur pourra la ramasser pour gagner une vie en plus.
- Ajouter un autre type de brique bonus qui fera augmenter la taille du joueur.
- Gestion de plusieurs niveaux : une fois le premier niveau terminé, le jeu pourra se poursuivre en intégrant différentes dispositions de briques. Il s'agira aussi d'intégrer les fonctionnalités précédentes de manières progressive dans les niveaux.