INFO-F-202
Langages de programmation 2
Université libre de Bruxelles
Laboratoire 4

Justin Dallant John Iacono
Yannick Molinghen Alexis Reynouard

28 Septembre 2021

1 Code fourni

Le code de démarrage qui vous est donné est en grande partie le même que la solution de la semaine précédente, avec des parties inutiles (telles que du texte) supprimées.

- La fonction draw de la classe Rectangle a été réécrite en utilisant différents appels de fonctions de fltk. Il fonctionne exactement de la même manière que l'ancien, mais quelques modifications ont été nécessaires pour que les transformations fonctionnent.
- Deux nouvelles classes Translation et Rotation sont fournies. Elles sont simples et ont juste un constructeur et un destructeur. Dans la classe Translation, vous fournissez un Point, et tant que l'instance de la classe Translation est vivante, tout ce qui est dessiné aura ses coordonnées décalées par celles fournies à la classe Translation. La classe de Rotation fonctionne de la même manière.
- Pour montrer comment fonctionne Translation, Canvas stocke un vecteur de Cells qui est initialisé avec 9 cellules :

```
Canvas::Canvas(){
  for (int x=50;x<500;x+=50)
  cells.push_back({{x,300},45,90});
}
```

  Au lieu d'appeler simplement draw sur chacune des cellules, nous créons un objet t de type Translation.

```
void Canvas::draw() {
  ++time;
  int i=0;
  for (auto &c: cells){
      ++i;
      Translation t{{0,static_cast<int>(100*sin(time/50.0+0.3*i))}};
      c.draw();
    }
}
```

  Cela illustre le fonctionnement de la Translation. Notez qu'une nouvelle instance de Translation est créée à chaque exécution de la boucle.
- Auparavant, la méthode draw de Cell appelait la méthode draw de son rectangle. Maintenant, La méthode draw de Cell appelle une nouvelle méthode `drawWithOutAnimate`, qui appelle draw sur le rectangle. La raison deviendra bientôt claire.

```
void Cell::drawWithoutAnimate(){
  r.draw();
}

```

```
5  void Cell::draw(){
6    drawWithoutAnimate();
7  }
```

2 Premier objectif : classe d'animation

Le premier objectif est de faire monter et descendre chaque rectangle UNE FOIS, puis de s'arrêter lorsque vous cliquez dessus.

Dans le code fourni, l'animation se fait dans la fonction draw de Canvas. Le but ici est d'abstraire l'animation dans une classe `Animation` que vous coderez. L'idée est que chaque Cell peut avoir une instance d'`Animation` ou non, lorsqu'une animation est active, elle a une instance, et sinon elle n'en a pas. Nous voulons qu'une instance d'animation puisse appliquer des transformations telles que translate ou rotate, et pour ce faire, elle doit créer ces transformations entre l'appel au draw de Cell et le `drawWithOutAnimate` de Cell. Ainsi :

- Si une cellule n'a pas d'animation, le draw d'une cellule doit appeler `drawWithOutAnimate` qui appelle draw sur son rectangle. C'est le comportement actuel.
- Si une cellule a une instance d'animation, elle doit appeler draw sur l'animation, ce qui créera une instance des transformations souhaitées puis appeler `drawWithOutAnimate` sur la cellule, qui appelle draw sur le rectangle.

La classe d'animation devrait ressembler à ceci :

```
1  class Animation{
2  ?????
3  public:
4    Animation(?????);
5    void draw();
6    bool isComplete();
7  };
```

Hints :

- La fonction draw de Canvas doit être restaurée en appelant simplement draw sur chaque cellule :

```
1  void Canvas::draw() {
2    for (auto &c: cells){
3        c.draw();
4     }
5  }
```

- Chaque instance de Cell peut avoir ou non une instance d'animation. Quel est le bon type de variable pour cela ? Un pointeur !
- Nous voulons que les animations démarrent lorsque vous cliquez sur une cellule. Utilisez new pour créer une animation lorsque vous cliquez sur une cellule.
- Lorsque l'animation est terminée, delete devrait être appelée (où ?). Une animation doit avoir une méthode isComplete pour déterminer quand une cellule doit la supprimer. **Si vous n'appelez jamais delete, vous avez une fuite de mémoire !**
- Les animations ont besoin d'accéder à la cellule qui les a créées pour pouvoir appeler `drawWithOutAnimate`. Comment est-ce que tu fais ça ?
- Utilisez `static_cast<int>(d)` pour convertir des doubles en entiers.
- Pendant l'exécution d'une animation, cliquer sur une cellule ne devrait avoir aucun effet.
- L'animation doit déplacer le rectangle vers le haut, puis vers le bas, puis s'arrêter. Choisissez une distance et une vitesse raisonnables pour le déplacer de haut en bas.

3 Tourner

Maintenant, vous voulez avoir trois types d'animations comme dans la démo : spin, bounce, spinAndBounce. Vous devez récrire le constructeur d'`Animation` pour prendre en paramètre le type d'animation. La meilleure façon de représenter un tel choix est d'utiliser une énumération, qui est un type que vous pouvez définir et qui prend l'une des valeurs que vous définissez :

```
1 class Animation{
2 public:
3   enum AnimationType {spin,bounce,spinAndBounce};
```

Ce faisant, AnimationType est maintenant un nouveau type qui prend l'une des trois valeurs données. (en dehors de la classe Animation, le type est Animation::AnimationType).

Vous pouvez forcer une conversion d'un int en un AnimationType en utilisant `static_cast`. Par exemple, `static_cast<AnimationType>(1)` est `bounce`.

Vous devez coder la classe d'animation pour faire les trois types d'animations, et lorsque vous cliquez sur une cellule, une animation aléatoire est choisie.