CR-C++ projet : Convertisseur de couleurs

CHEN Qingyue
ZHENG Wenhan

• Description de l’application développée

Il y a plusieurs fois dans la vie, nous avons besoin d'inspiration de correspondance des couleurs. Notre programme peut non seulement vous fournir une couleur aléatoire, mais également fournir différents schémas de couleurs de cette couleur: couleur de contraste, correspondance des couleurs du triangle sur le cercle de teinte, couleurs similaires, etc.Il y a plusieurs fois dans la vie, nous avons besoin d'inspiration de correspondance des couleurs. Notre programme peut non seulement vous fournir une couleur aléatoire, mais également fournir différents schémas de couleurs de cette couleur: couleur de contraste, correspondance des couleurs du triangle sur le cercle de teinte, couleurs similaires, etc.

C’est une application de convertisseur de couleurs avec une interface graphique.Après avoir exécuté le programme, vous obtiendrez l'interface suivante :

Sur la droite, il y a une couleur générée aléatoirement. Vous pouvez cliquer sur le bouton "afficher" pour obtenir une couleur complémentaire de cette couleur et 5 couleurs similaires (obtenues en ajustant le S et le V de la couleur générée

aléatoirement dans l'espace HSV) :

• Mettre en valeur l'utilisation des contraintes.

3 hiérarchies et 8 class :

-ColorSpace,

-RGB, HSV,

-Calculer, Complementaire, ColorBoard, RandomColor, TriColor

La première hiérarchie est ColorSpace. Afin de créer la structure de l'espace colorimétrique, il contient les trois coordonnées (x, y, z) de l'espace colorimétrique et quelques fonctions de base pour la construction de l'espace colorimétrique.

La deuxième hiérarchie comporte deux espaces colorimétriques différents, RGB et HSV. Ils héritent des coordonnées et des fonctions de ColorSpace. Différents espaces colorimétriques ont des représentations de coordonnées différentes. Chaque espace colorimétrique peut être converti l'un à l'autre et a une expression hexadécimale de coordonnées.

La troisième hiérarchie est les applications et les calculs des couleurs basés sur des deux espaces colorimétriques de la deuxième hiérarchie. Par exemple, rechercher des couleurs contrastées, des couleurs voisines, etc.

2 fonctions virtuelles différentes :

```
virtual string hexadecimal()const =0;  //Représentation hexadécimale de la couleur
virtual vector<size_t> toRGB()const=0;  //La valeur RGB de la couleur est stockée dans un conteneur
virtual vector<size_t> toHSV()const=0;  //La valeur HSV de la couleur est stockée dans un conteneur
```

Il existe trois fonctions virtuelles dans ColorSpace, qui sont requises dans différentes sous-classes, mais l'implémentation spécifique est différente, elles doivent donc être héritées et écrasées.

2 surcharges d'opérateurs :

```
  RGB operator*(const RGB&);
void operator=(const RGB&);
friend std::ostream& operator<<(std::ostream& , const RGB &);
};
```

Nous avons surchargé trois opérateurs:

Surcharge l'opérateur * afin de compléter la multiplication de deux couleurs dans l'espace RGB.

Surcharge l'opérateur = afin de lui attribuer une valeur avec un objet de même classe dans l'espace RGB et HSV.

Surcharge l'opérateur << afin de sortir de l'expression hexadécimale de tous les objets couleur dans toutes les classes.

2 conteneurs différents de la STL :

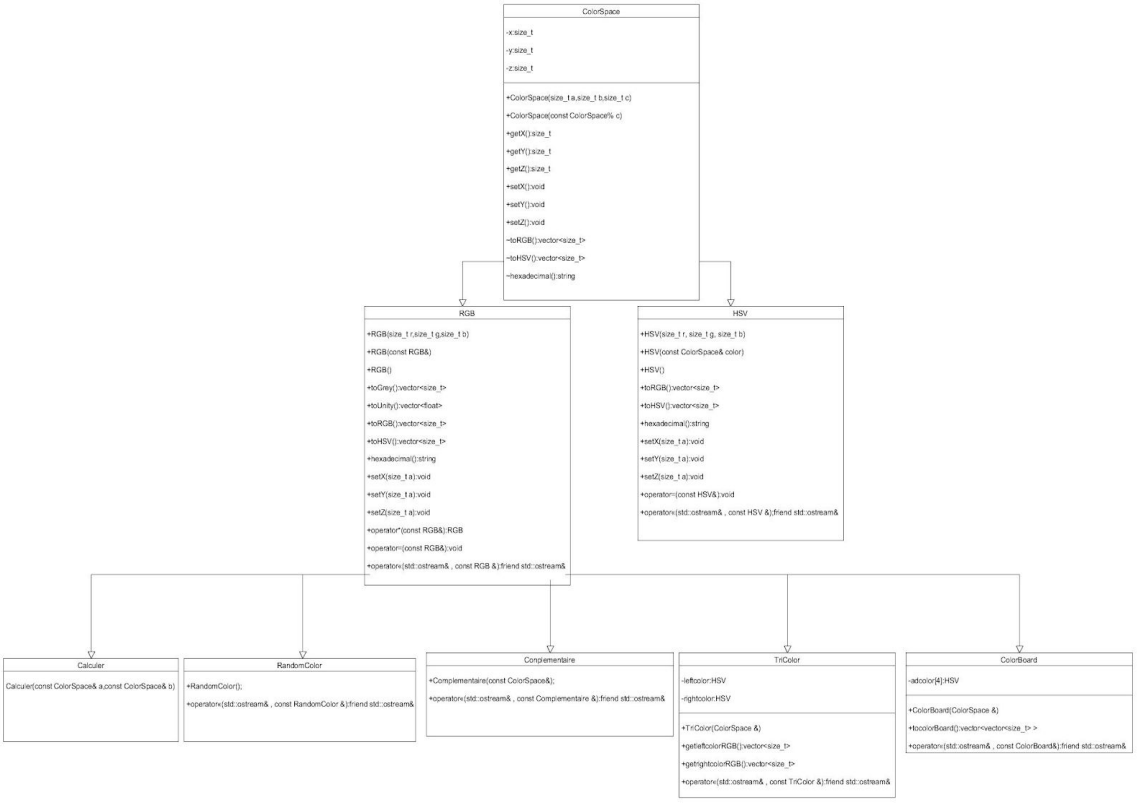
-vector<size_t>

-vector <vector<size_t>>

pas d'erreurs :

```
--------- Valeur hexadécimale ---------
#474766
------ Melange 2 RGB ------
#4b2ca5
------ Melange RGB et HSV ------
#3d3d83
------ Melange Cal et RGB -------
#5c34cf
------ Couleur complémentaire de RGB random ------
#de998b
------ Couleur complémentaire de HSV ------
#4a4b02
------ 3 Couleurs triangulaire  ---------
#664766
#666647
#476666
------ 5 Couleurs similaires  ---------
#d6d6a7
#afaf64
#648c32
#898931
#b7b75d
===============================================================================
All tests passed (8 assertions in 6 test cases)
```

• Diagramme UML de l'application.



• Procédure d'installation (bibliothèques...) et d'exécution du code.
Bibliothèque graphique : SDL2
Bibliothèque de texte : SDL_ttf

• Les parties de l'implémentation dont vous êtes les plus fières.

La partie dont nous sommes plus fiers est le suivant:

On a conçu une classe pour produire une série de couleurs similaires. On transfère la couleur spécifiée dans l'espace HSV, obtenons une série de couleurs en augmentant ou en diminuant progressivement la valeur de S et la valeur de V, et stocker les valeurs RGB de toutes les couleurs dans un conteneur et affichons ces couleurs dans l'interface visuelle.

```
ColorBoard::ColorBoard(ColorSpace &c):RGB(){
    vector<size_t> v1 = c.toRGB();
    x = v1[0];
    y = v1[1];
    z = v1[2];
    vector<size_t> v2 = c.toHSV();
    size_t s,v;

    for(int i=0;i<4;i++){
      adcolor[i].setX(v2[0]);                    //Augmenter ou diminuer la saturation et la value
        if(i<2){
            s=v2[1]-(2-i)*(v2[1]/3);
            v=v2[2]+(2-i)*((100-v2[2])/3);
        }
        else{
            s=v2[1]+(2-i)*((100-v2[2])/3);
            v=v2[2]-(2-i)*(v2[2]/3);
        }

        adcolor[i].setY(s);
        adcolor[i].setZ(v);
    }
```

Pour afficher les les cinq couleurs dans l'interface, on récupe les valeurs de RGB dans le fichier exécuter.cc .

```
RandomColor ran;

//initialiser les valeurs
char* getrgb[]={(char*)(ran.getX()),(char*)(ran.getX()),(char*)(ran.getX())};
const char* hex = (char*)(ran.hexadecimal()).c_str();
cout<<hex<<endl;
stringstream s;s<<hex;
Complementaire com(ran);
char* comrgb[] = {(char*)(com.getX()),(char*)(com.getX()),(char*)(com.getX())};
vector<size_t> vec = ran.toHSV();
char* gethsv[] = {(char*)vec[0],(char*)vec[1],(char*)vec[2]};
ColorBoard board(ran);
vector<vector<size_t>> vb = board.tocolorBoard();
```

On surveille le fonctionnement de la souris et lit la position de la souris lorsque l'utilisateur clique dessus.

```
while (!quit){

  /* Etape 1: RECUPERER LES EVENEMENTS */
  if (SDL_PollEvent(&event)){
    //printf("un event\n");
    switch (event.type){
    case SDL_QUIT:
      quit = 1;
      break;

    case  SDL_MOUSEBUTTONDOWN:
      SDL_GetMouseState( &mx, &my );
      printf("mx=%d my=%d\n",mx,my);

      break;
    }
  }
```

On juge la position de la souris, s'il clique sur le bouton "afficher", le jeu de couleurs sera affiché.

```
//METTRE A JOUR l'ETAT DE L'APPLICATION GRAPHIQUE
if ((mx>=600) && (mx<=700) && (my>=540) && (my<=590)){

  SDL_SetRenderDrawColor(renderer, com.getX(), com.getY(), com.getZ(), 230);
  rect = {150, 250, 200, 100};
  SDL_RenderFillRect(renderer, &rect);

  for(int i=0;i<5;i++){
    SDL_SetRenderDrawColor(renderer, vb[i][0], vb[i][1], vb[i][2], 230);
    rect = {150+i*100, 600, 100, 50};
    SDL_RenderFillRect(renderer, &rect);
  }
}
```