Projet : Super Simon

Martin Niombela

Table des matières

But du projet **2**

Descriptif du projet **2**

Conception du programme **2**
- Gestion des actions du joueur . . . 2
- Affichage des différents éléments du jeu . . . 2
- Séquence de couleur aléatoire . . . 3

Développement du programme **3**
- Préambule . . . 3
- Définition des variables importantes . . . 3
  - Pour la mise en place de l'interface . . . 3
  - Pour le fonctionnement du jeu . . . 3
  - Autres variables . . . 4
- Boucle de jeu . . . 4
  - Démonstration du Super Simon . . . 4
  - Reproduction du schéma par le joueur . . . 4
- Fin de partie . . . 4

Conclusion du projet **4**

But du projet

Ce projet a été réalisé en langage C pendant mon temps libre et dans le but de vérifier mes acquis dans ce langage de programmation. Ce projet est au départ un projet de deux semaines de programmation pour les 1er années de DUT Informatique. Ce projet avait aussi pour but de me permettre de pratiquer du C, ce qui n'avait pas été le cas depuis un certains temps.

Descriptif du projet

Ce projet, "Super Simon", concerne la programmation du jeu bien connu en une interface graphique. Il doit faire apparaitre une fenêtre ainsi que des couleurs, et laisser le joueur retrouver la séquence de couleur donné par le programme. Le joueur doit pouvoir voir le nombre de manche qu'il a passé, et l'affichage de son score. Il est aussi possible de pousser la difficulté du projet plus loin, en proposant plusieurs niveaux de difficulté.

Ce projet se base sur une bibliothèque graphique : *libgraphique*. Cette bibliothèque est une sur-couche de la bibliothèque graphique SDL, et permet d'utiliser des fonctions simples pour gérer une interface graphique.

Conception du programme

Ce projet nécessite plusieurs éléments, qui seront des fonctions à implémenter dans le programme :

- La gestion des actions du joueur
- L'affichage d'un plateau de jeu, du nombre de manche, du score
- La génération d'une séquence de couleur aléatoire pour le jeu
- Un affichage spécial pour une couleur faisant partie de la séquence du jeu

Pour savoir comment implémenter ces différentes fonctionnalités dans le programme, il faut prendre en compte les fonctions que la bibliothèque "libgraphique" possède.

Gestion des actions du joueur

Dans un jeu, il est possible d'interagir soit avec les touches du clavier, soit avec la souris. La bibliothèque "libgraphique" possède des fonctions permettant de détecter les clics de l'utilisateur et leur emplacement, donc le joueur devra interagir avec le jeu en utilisant sa souris.

De plus, il faut déterminer si les actions du jeu doivent être gérer en temps réel ou non. Dans le cadre du Super Simon, le jeu se fait tour par tour : Super Simon montre la séquence à reproduire, puis le joueur la reproduit. On peut donc gérer les évènements de clic hors du temps réel, avec des commandes bloquantes.

Affichage des différents éléments du jeu

Le Super Simon nécessite d'afficher un plateau de jeu, mais aussi le nombre de manche et le score. La bibliothèque "libgraphique" possède des fonctions pour l'affichage du texte et pour l'affichage de formes. Le plateau de jeu est constitué de formes. Le nombre de manche et l'affichage du score sont du texte.

L'interface créé par "libgraphique" nécessite d'être mise à jour lorsqu'on la modifie. Comme le plateau de jeu, le nombre de manche et le score ne seront pas mis à jour au même moment dans le programme, l'affichage de ces trois éléments doit se faire dans des fonctions séparées.

Séquence de couleur aléatoire

Pour le jeu du Super Simon, il est nécessaire de générer une séquence aléatoire de couleurs que le joueur doit reproduire. Comme il faut gérer les clics de l'utilisateur, et que les fonctions de la bibliothèque "libgraphique"

repose sur l'emplacement de "Point", il faut faire une séquence de **point** aléatoire, où chaque point représente la forme associée à une couleur du Super Simon.

> *Les fonctions utilisées dans le programme seront détaillées dans la documentation, à la fin de ce document.*

Développement du programme

Cette section sert à expliquer le fonctionnement du programme ainsi que les solutions qui ont été implémentées pour ajouter les fonctionnalités demandées dans le programme.

Préambule

L'utilisation de la bibliothèque est pratique, car elle permet de faire de l'interfaçage graphique via des fonctions de haut niveau par rapport à la bibliothèque SDL. Cependant, elle demande à être gérée **pixel par pixel**. Si une possible amélioration de ce projet porterait sur la gestion des dimensions dans le programme, à cette heure, les coordonnées utilisées dans le programme sont notées en dur.

Pour l'interface, il sera affiché :

— Au milieu de l'interface, le plateau de jeu, constitué de quatre carrés de couleurs différentes
— Au dessus du plateau de jeu : Le numéro de la manche et le nom du jeu
— A la fin du jeu : un écran de fin, pour la victoire ou la défaite du joueur

Définition des variables importantes

Pour la mise en place de l'interface

Le plateau de jeu est composé de quatre carrés. Chaque carré a une position précise, désigné par son point supérieur gauche (voir le fonctionnement de la fonction `dessiner_rectangle()`). Il faut donc commencer par déclarer quatre variables `Point`, un pour chaque carré du plateau.

L'interface affiche aussi le nom du jeu, "Super Simon", au dessus du plateau. Pour cela, il faut déclarer des variables `Point` pour l'emplacement du texte et du cadre du texte.

Pour le fonctionnement du jeu

Comme mentionné dans la section Séquence de couleur aléatoire, il faut choisir des couleurs de manière aléatoire pour le joueur. Dans le programme, cela se caractérise par la création de deux tableaux de `Point` :

— Un tableau contenant plusieurs fois les quatre `Point` correspondant au plateau
— Un tableau de `Point`, de taille 10, pour accueillir la séquence aléatoire

Il faut aussi créer des variables pour gérer les autres aspects du jeu :

— Le numéro de la manche en cours (nombre entier)
— Le nombre de tour de boucle à faire dans le jeu (nombre entier)
— La condition de victoire ou défaite (booléen)
— Le score du joueur

Autres variables

En langage C, il existe plusieurs normes définissant le langage. Dans certaines de ces normes, la notion de booléen n'est pas considéré comme un type de variable à part entière. J'ai donc choisi de définir moi-même des variables qui feront office de booléen, via un `#define`.

Boucle de jeu

Le jeu est censé tourner tant que le joueur n'a pas atteint la condition de victoire ou de défaite. Pour le jeu du Super Simon, la condition de victoire est de tenir jusqu'à la dixième manche sans se tromper sur la séquence aléatoire des couleurs.

La condition ressemble donc à cela :

```
while(manche <= 10 && echec == false) {}
```

Dans le Super Simon, chaque tour se déroule en deux étapes :

1. Phase de démonstration du Super Simon
2. Tentative de reproduction par le joueur

La boucle de jeu dans le programme suit aussi ce schéma. Une fois ce schéma passé, si le joueur n'est pas en situation de défaite, il peut passer à la manche suivante. Sinon, c'est la fin de la boucle, et la fin de la partie.

Démonstration du Super Simon

La démonstration du Super Simon se fait en quelques étapes :

1. Désignation de la couleur choisie par le curseur
2. Temps d'attente
3. Disparition du curseur

Ce schéma se répète autant de fois que le nombre de manche passé dans le jeu.

Reproduction du schéma par le joueur

Les étapes du tour du joueur diffèrent un peu de la démonstration du Super Simon :

1. Attente de la sélection d'une couleur via un clic dans le plateau
2. Vérification du choix de la couleur

Fin de partie

En cas de victoire comme en cas de défaite, le programme affichera l'écran de fin, le score et fermera la fenêtre du programme. Le contenu de l'écran de fin dépend de la situation du joueur à la fin de la partie : victoire ou défaite. L'affichage du score se fait dans la console.

Conclusion du projet

Ce projet de réalisation d'un "mini-jeu" en langage C, est assez intéressant car il permet de voir les différents aspects les plus importants de la programmation, comme la conception d'un programme ou encore les spécificités du langage utilisé pour le développement du projet.

Ce projet est intéressant car il permet aussi d'avoir un aperçu basique de l'utilisation de la bibliothèque graphique SDL en langage C.

Bien sûr, ce projet étant assez basique, il possède une grande marge d'amélioration possible, comme le stockage des scores et des joueurs qui les ont faits, un meilleur écran de fin ou encore des niveaux de difficulté variés.

Documentation : Fonction du projet "Super Simon"

Martin Niombela

Ce document va détailler les différentes fonctions écrites et utilisées dans le cadre du projet Super Simon.

> *Certaines fonctions utilisées dans le projet sont issus de la bibliothèque "libgraphique". Ces fonctions seront détaillées dans la documentation de cette bibliothèque.*
>
> *Les fonctions courtes seront écrites entièrement dans la documentation. Les fonctions plus longues feront l'objet d'une syntaxe en pseudo-code.*

Initialisation du plateau de jeu

```
void init_tab(Point, Point, Point, Point)
```

Cette fonction va servir à afficher le plateau du Super Simon dans la fenêtre du jeu. Le plateau est composé de quatre couleurs : bleu, rouge, jaune et vert. Ces quatre couleurs prennent la forme d'un carré, divisé en quatre carrés plus petits.

Paramètre(s)

- `Point p1` : Correspond au coin supérieur gauche du premier carré du plateau.
- `Point p2` : Correspond au coin supérieur gauche du deuxième carré du plateau.
- `Point p3` : Correspond au coin supérieur gauche du troisième carré du plateau.
- `Point centre` : Correspond au coin supérieur gauche du quatrième carré du plateau. Ce point est aussi le point situé au centre du plateau.

Retour

Cette fonction ne renvoie rien.

Corps de la fonction

```
void init_tab (Point p1, Point p2, Point p3, Point centre) {
        dessiner_rectangle(p1, 125, 125, bleu);
        dessiner_rectangle(p2, 125, 125, rouge);
        dessiner_rectangle(p3, 125, 125, vert);
        dessiner_rectangle(centre, 125, 125, jaune);
        actualiser();
}
```

Affichage d'un curseur

Curseur du Super Simon : `void curseur(Point)`
Curseur du joueur : `void j_curseur(Point)`

Dans le jeu Super Simon, pour afficher l'ordre des couleurs que le joueur devra imiter, certaines sont mises en relief pendant un court laps de temps. Pour ce programme, l'affichage d'un curseur est nécessaire afin de montrer l'ordre des couleurs à imiter.

Ce programme affiche aussi un curseur pour le joueur, à titre indicatif, lorsqu'il sélectionne une couleur du plateau.

Curseur du Super Simon

Dans cette fonction, le but est d'afficher un "curseur" dans l'un des carrés du plateau. Au vu des limitations induites par la bibliothèque "libgraphique", il est nécessaire d'utiliser une des fonctions de dessin de formes pour faire un curseur. Le curseur est un carré blanc, dessiné par dessus l'ancien carré du plateau de jeu. Par dessus ce carré blanc, on redessine un carré de la bonne couleur, mais en plus petit., ce qui simule une apparence de curseur.

Paramètre(s)

`Point p` : Point correspondant au coin supérieur gauche du carré sélectionné.

Retour

Cette fonction ne renvoie rien.

Corps de la fonction

```
void curseur (Point p) {
        Couleur col = couleur_point(p);
        dessiner_rectangle(p, 125, 125, blanc);
        Point p_fonc = {p.x + 13, p.y + 13};
        dessiner_rectangle(p_fonc, 100, 100, col);
        actualiser();
}
```

Curseur du joueur

Dans cette fonction, le but est d'afficher un curseur sur la couleur sélectionné par le joueur et de récupérer la couleur choisie par le joueur. Si le joueur ne clique pas dans une des couleurs du plateau, un point vide (`x` et `y` égales à 0) est retourné.

Paramètre(s)

`Point c` : Point correspondant à l'emplacement du clic du joueur.

Retour

Cette fonction utilise la fonction `curseur()`.

Corps de la fonction

```
Point j_curseur(Point c) {
    Point ref = {0, 0};

    if (/* Clic dans carré bleu*/) { curseur(carre_bleu); ref = carre_bleu; }
    else if (/* Clic dans carré rouge*/) { curseur(carre_rouge); ref = carre_rouge; }
    else if (/* Clic dans carré vert*/) { curseur(carre_vert); ref = carre_vert; }
    else if (/* Clic dans carré jaune*/) { curseur(carre_jaune); ref = carre_jaune; }

    return ref;
}
```

Affichage du nombre de manche

Dans cette fonction, le but est d'afficher dans la fenêtre du jeu le numéro de la manche en cours. Chaque manche correspond au nombre de bonne réponse du joueur. La première manche commence à 1.

Pour la concaténation du texte avec le numéro de la manche, on utilise la fonction `strcat()` de `<string.h>`.

Paramètre(s)

`int manche` : Entier correspondant au numéro de la manche en cours.

Retour

Cette fonction ne renvoie rien.

Corps de la fonction

```
void affiche_manche (int manche) {
        Point txt = {200,170}, ligneD = {200,190}, ligneF = {288,190};
        char affiche[20] = "Manche ";
        dessiner_rectangle(txt,120,30,noir);


        char num_manche[3]; sprintf(num_manche, "%d", manche);
        strcat(affiche, num_manche);


        afficher_texte(affiche,17,txt,blanc);
        dessiner_ligne(ligneD,ligneF,blanc);
        actualiser();
}
```

Génération de nombre aléatoire

Dans cette fonction, le but est de récupérer un nombre aléatoire compris entre deux bornes.

Paramètre(s)

- `int a` : La première borne, un entier

- `int b` La deuxième borne, un entier

Retour

Renvoie un nombre aléatoirement, compris entre `a` et `b`.

Corps de la fonction

```
int aleatoire (int a, int b) {
        if (a < b) return rand()%(b-a) + a;
        else return rand()%(a-b) + b;
}
```

Ecran de fin

Dans cette fonction, le but est d'afficher à la fin de la partie si le joueur a remporté la partie ou non. Si le joueur a gagné, la phrase "Vous avez gagné !" est affiché. Sinon, c'est "Vous avez perdu..." qui s'affiche.

Paramètre(s)

`int défaite` : Booléen indiquant si le joueur a gagné ou perdu.

Retour

Cette fonction ne renvoie rien.

Corps de la fonction

```
void ecran_fin (int defaite) {
        Point txt = {240,325}, ligne_debut = {240,350}, ligne_fin = {465, 350};
        Point blk_screen = {0,0};
        dessiner_rectangle(blk_screen, 700, 700, noir);


        if (defaite) afficher_texte("Vous avez perdu...",20,txt,blanc);
        else afficher_texte("Vous avez gagne !", 20, txt, blanc);


        dessiner_ligne(ligne_debut, ligne_fin, blanc);
        actualiser();
}
```

(version 18/10/2017)

Notice d'utilisation de la bibliothèque libgraphique

Types

La `libgraphique` définit et utilise deux nouveaux types : `Point` et `Couleur`

Le type `Point`

C'est un type structuré : il est composé de deux **attributs** (ou **champs)** nommés `x` et `y` qui correspondent à des coordonnées (abscisse et ordonnée).
Exemple de déclaration et d'utilisation d'un point.

```
Point p1 ; //declaration d'un Point nomme p1

p1.x = 22 ; //on initialise le champ x de p1 a 22

p1.y =100 ; //on initialise le champ y de p1 a 100

if (p1.x <0) //on peut manipuler les champs comme n'importe quelle variable
   ...
```

Initialisation d'un `Point` (et plus généralement d'une structure)

On peut déclarer et initialiser IMMEDIATEMENT tous les champs d'une structure :

```
Point p1 = {34 , 45} ;
```

...mais **ATTENTION** on ne peut pas tout initialiser d'un coup APRES la déclaration :

```
Point p1 ;
p1 = {34 , 45} ; //CA NE COMPILERA PAS!
```

Vous êtes alors obligé.e d'intialiser chacun des champs un par un :

```
p1.x = 34 ;
p1.y = 45 ;
```

Le type `Couleur`

C'est un type entier qui décrit une couleur en 32 bits. On trouvera dans `libgraphique.h` (et à la fin de ce document) une longue liste de couleurs que l'on peut utiliser directement, dont 16 couleurs standard en français : `bleu, noir, blanc, ....`
On peut fabriquer facilement une nouvelle couleur avec la fonction `fabrique_couleur` (voir ci-dessous).

Fonctions

Gestion de la fenêtre

```
void ouvrir_fenetre(int largeur, int hauteur)
```

Ouvre une fenêtre graphique d'après les dimensions données, en pixels. Au départ tous les pixels sont noirs.

```
void fermer_fenetre()
```

Termine la session graphique (pas nécessairement le programme principal, attention).

```
void actualiser()
```

Toute modification du type "dessiner" sera invisible tant que la fonction `actualiser` n'a pas été appelée. Ceci permet de faire plusieurs modifications avant de changer l'affichage.

Dessin

```
void changer_pixel(Point pix, Couleur couleur)
```

Change la couleur du pixel `pix`, donné par un `Point`, de la couleur `Couleur`.
Exemple :

```
Point p = {50 , 100};
changer_pixel(p, bleu);
```

```
void dessiner_rectangle(Point coin, int largeur, int hauteur, Couleur couleur)
```

Dessine un rectangle de la couleur donnée ; le point `coin` est le coin en haut à gauche du rectangle, et la largeur et la hauteur correspondent respectivement aux dimensions horizontales et verticales, en pixels.

```
void dessiner_ligne(Point p1, Point p2, Couleur couleur)
```

Dessine une ligne de la couleur donnée entre les deux points. Utilise un algorithme simple sans *anti-aliasing*.

```
void dessiner_disque(Point p, int rayon, Couleur couleur)
```

Dessine un disque de centre `p`, de rayon `rayon`, et de la couleur donnée. Utilise un algorithme simple sans *anti-aliasing*.

```
void afficher_image(char *nom, Point coin);
```

La chaîne de caractères `nom` doit être le nom d'un fichier bitmap `.bmp` situé dans le même répertoire que le programme. L'image sera affichée selon ses dimensions, le coin supérieur gauche étant donné par le Point `coin`.

Interaction avec l'utilisateur

```
int attendre_touche()
```

Cette instruction bloque l'exécution du programme jusqu'à ce que l'utilisateur appuie sur une touche. Le code SDL de la touche est alors renvoyé. Les codes SDL des touches figurent sur

http ://www.libsdl.org/release/SDL-1.2.15/docs/html/sdlkey.html

mais la plupart ont des noms simples commençant par `SDLK_`, comme

`SDLK_A, SDLK_B, SDLK_UP (`*flèche haut*`), SDLK_LEFT, SDLK_SPACE, SDLK_ESCAPE ...`

Exemple :

```
int touche ;
touche = attendre_touche() ;
if (touche == SDLK_DOWN) ...
```

`int attendre_touche_duree(int ms)`

Même effet que la fonction précédente, mais la fonction est bloquante pendant `ms` millisecondes, après quoi elle renvoie 0 si aucune touche n'a été pressée

`Point attendre_clic() ;`

Fonctionne comme `attendre_touche` mais attend un clic de l'utilisateur et renvoie un Point correspondant au pixel cliqué.

Exemple :

```
Point p ;
p = attendre_clic() ;
```

`Point attendre_clic_gauche_droite() ;`

Comme la fonction précédente, mais ajoute un signe négatif devant les coordonnées du point si le clic qui a eu lieu est un clic droit.

`void attente(int ms) ;`

Met le programme en attente durant `ms` millisecondes.

Gestion des couleurs

`Couleur fabrique_couleur(int r, int g, int b);`

Renvoie une Couleur RGB fabriquée d'après les quantités de rouge `r`, vert `g`, et bleu `b` données, qui sont des `int` entre 0 et 255. Par exemple le violet est obtenu en appelant `fabrique_couleur(128,00,128)` ;

`Couleur couleur_point(Point p) ;`

Renvoie la couleur du point `p`. Renvoie `noir` si le point est hors de l'écran.

Tirage aléatoire

`int entier_aleatoire(int n) ;`

Renvoie un entier aléatoire compris entre 0 et n-1 inclus.

Gestion des événements en temps "réel" (optionnel)

Partie optionnelle à ne lire que lorsque vous aurez tout terminé, si vous en avez besoin pour votre projet. Les fonctions comme `attendre_clic` ou `attendre_touche` sont bloquantes : elles bloquent l'exécution du programme en attente d'un clic ou d'une touche. Si l'on désire que le programme continue à s'exécuter tout en surveillant quand même si l'utilisateur clique ou appuie sur des touches, on pourra utiliser les fonctions suivantes. Il s'agit d'une solution sommaire et on peut faire bien plus efficace. L'exemple 5 de la partie 4 utilise ces fonctions sur divers exemples.

```
void reinitialiser_evenements() ;
```

Cette fonction remet à zéro les événements enregistrés.

```
void traiter_evenements() ;
```

Le programme qui gère les graphiques enregistre au fur et à mesure toutes les actions de l'utilisateur (touches et mouvement de souris). Un appel à cette fonction permet de traiter tous les événements n'ayant pas été encore traités. On interrogera ce mécanisme à l'aide des fonctions `touche_a_ete_pressee` et `clic_a_eu_lieu`.

```
int touche_a_ete_pressee(int code) ;
```

Renvoie un booléen indiquant si la touche de code en question (voir `attendre_touche` ci-dessus) a été pressée entre les derniers appels à `reinitialiser_evenements` et `traiter_evenements`.

```
Point clic_a_eu_lieu() ;
```

Renvoie un `Point` dont les coordonnées sont celles du dernier clic entre les derniers appels à `reinitialiser_eveneme` et `traiter_evenements`. Par convention s'il n'y a eu aucun clic, les coordonnées du `Point` renvoyé sont $(-1, -1)$.

```
Point deplacement_souris_a_eu_lieu() ;
```

Renvoie un point de coordonnées relatives souris mesuré entre la dernière réinitialisation et le dernier traitement. Au démarrage, renvoie $(0, 0)$. Si on sort de la fenêtre, renvoie la dernière position reçue.

Affichage de texte

```
void afficher_texte(char *texte, int taille, point coin, Couleur couleur);
```

Affiche le texte `texte` de taille `taille` à la position `coin` (supérieur gauche de la zone de texte) et de couleur `couleur`.

```
Point taille_texte(char *texte, int taille);
```

Renvoie un point qui contient la taille en pixels $(largeur, hauteur)$ de ce texte si on l'affichait. Cette fonction permet de créer un rectangle adapté à la taille d'un texte à afficher.

Comment installer la libgraphique

Si vous souhaitez utiliser la libgraphique sur votre système, vous devez installer les paquets suivants : `libsdl1.2debian, libsdl1.2-dev, libsdl-ttf2.0-0, libsdl-ttf2.0-dev`.

Comment compiler

Si on doit taper la commande en entier, il faut se placer dans le répertoire contenant l'exercice, qui doit être au même niveau que le répertoire `lib` contenant `libgraphique.c`, et taper

```
gcc ../lib/libgraphique.c nom_du_code_source.c -o nom_du_resultat `sdl-config
                    --libs --cflags` -lm -lSDL -lSDL_ttf
```

Attention les ` ne sont pas des apostrophes mais des apostrophes inversées (ALTGR + 7 sur la plupart des claviers). Dans les répertoires préparés pour vous il suffit de taper "make".

Liste des couleurs

En français : argent, blanc, bleu, bleumarine, citronvert, cyan, magenta, gris, jaune, marron, noir, rouge, sarcelle, vert, vertclair, vertolive, violet

En anglais : aliceblue, antiquewhite, aqua, aquamarine, azure, beige, bisque, black, blanchedalmond, blue, blueviolet, brown, burlywood, cadetblue, chartreuse, chocolate, coral, cornflowerblue, cornsilk, crimson, cyan, darkblue, darkcyan, darkgoldenrod, darkgray, darkgreen, darkkhaki, darkmagenta, darkolivegreen, darkorange, darkorchid, darkred, darksalmon, darkseagreen, darkslateblue, darkslategray, darkturquoise, darkviolet, deeppink, deepskyblue, dimgray, dodgerblue, firebrick, floralwhite, forestgreen, fuchsia, gainsboro, ghostwhite, gold, goldenrod, gray, green, greenyellow, honeydew, hotpink, indianred, indigo, ivory, khaki, lavender, lavenderblush, lawngreen, lemonchiffon, lightblue, lightcoral, lightcyan, lightgoldenrodyellow, lightgreen, lightgrey, lightpink, lightsalmon, lightseagreen, lightskyblue, lightslategray, lightsteelblue, lightyellow, lime, limegreen, linen, magenta, maroon, mediumaquamarine, mediumblue, mediumorchid, mediumpurple, mediumseagreen, mediumslateblue, mediumspringgreen, mediumturquoise, mediumvioletred, midnightblue, mintcream, mistyrose, moccasin, navajowhite, navy, oldlace, olive, olivedrab, orange, orangered, orchid, palegoldenrod, palegreen, paleturquoise, palevioletred, papayawhip, peachpuff, peru, pink, plum, powderblue, purple, red, rosybrown, royalblue, saddlebrown, salmon, sandybrown, seagreen, seashell, sienna, silver, skyblue, slateblue, slategray, snow, springgreen, steelblue, tann, teal, thistle, tomato, turquoise, violetlight, wheat, white, whitesmoke, yellow, yellowgreen