

---

# SGG Brush Utility Functions

---

Επιμέλεια: Α.Δρακόπουλος

---

## Πίνακας περιεχομένων

Βιβλιοθήκη brush_utilities.h.....	3
Κώδικας brush_utilities.h.....	3
Πρόγραμμα δοκιμής brush_utils.h.....	13
Πρόγραμμα δοκιμής getBrushByName().....	15

# Βιβλιοθήκη brush\_utilities.h

## Κώδικας brush\_utilities.h

```
#pragma once
// brush_utils.h

#ifndef BRUSH_UTILS_H
#define BRUSH_UTILS_H

#include "sgg/graphics.h"
#include <iostream>
#include <map>
#include <string>
#include <tuple>

namespace brush_utils {

    /**
     * Ρυθμίζει το βασικό χρώμα γέμισης ενός Brush.
     * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
     * @param r Η κόκκινη συνιστώσα του χρώματος (0.0 - 1.0).
     * @param g Η πράσινη συνιστώσα του χρώματος (0.0 - 1.0).
     * @param b Η μπλε συνιστώσα του χρώματος (0.0 - 1.0).
     */
    void setFillColor(graphics::Brush& brush, float r, float g, float b) {
        brush.fill_color[0] = r;
        brush.fill_color[1] = g;
        brush.fill_color[2] = b;
    }

    /**
     * Ανακτά το κύριο χρώμα γεμίσματος του `Brush`.
     * @param brush Το `Brush` αντικείμενο.
     * @return Το χρώμα γεμίσματος ως `tuple` των (r, g, b).
     */
    std::tuple<float, float, float> getFillColor(const graphics::Brush& brush) {
        return { brush.fill_color[0], brush.fill_color[1], brush.fill_color[2] };
    }

    /**
     * Ρυθμίζει το δευτερεύον χρώμα γέμισης ενός Brush για gradient εφέ.
     * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
     * @param r Η κόκκινη συνιστώσα του δευτερεύοντος χρώματος (0.0 - 1.0).
     * @param g Η πράσινη συνιστώσα του δευτερεύοντος χρώματος (0.0 - 1.0).
     * @param b Η μπλε συνιστώσα του δευτερεύοντος χρώματος (0.0 - 1.0).
     */
    void setSecondaryFillColor(graphics::Brush& brush, float r, float g, float b) {
        brush.fill_secondary_color[0] = r;
    }
}
```

```
brush.fill_secondary_color[1] = g;
brush.fill_secondary_color[2] = b;
}
```

```
/**
```

```
* Επιστρέφει το δευτερεύον χρώμα γεμίσματος της βούρτσας `Brush`.
```

```
*
```

```
* @param brush Το αντικείμενο `graphics::Brush` από το οποίο θα ανακτηθεί το χρώμα.
```

```
* @return Μια `std::tuple` που περιέχει τις τιμές (r, g, b) του δευτερεύοντος χρώματος
```

```
γεμίσματος.
```

```
*
```

```
* Το δευτερεύον χρώμα γεμίσματος χρησιμοποιείται κυρίως σε καταστάσεις όπου έχει οριστεί  
gradient (διαβάθμιση).
```

```
*/
```

```
std::tuple<float, float, float> getFillSecondaryColor(const graphics::Brush& brush) {  
    return { brush.fill_secondary_color[0], brush.fill_secondary_color[1],  
brush.fill_secondary_color[2] };  
}
```

```
/**
```

```
* Ρυθμίζει την αδιαφάνεια γέμισης του Brush.
```

```
* @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
```

```
* @param opacity Η αδιαφάνεια γέμισης (0.0 για πλήρη διαφάνεια έως 1.0 για πλήρη  
αδιαφάνεια).
```

```
*/
```

```
void setFillOpacity(graphics::Brush& brush, float opacity) {  
    brush.fill_opacity = opacity;  
}
```

```
/**
```

```
* Επιστρέφει την αδιαφάνεια του κύριου χρώματος γεμίσματος της βούρτσας `Brush`.
```

```
*
```

```
* @param brush Το αντικείμενο `graphics::Brush` από το οποίο θα ανακτηθεί η αδιαφάνεια.
```

```
* @return Μια τιμή `float` που αντιπροσωπεύει την αδιαφάνεια του γεμίσματος (0.0 για πλήρως  
διαφανές έως 1.0 για πλήρως αδιαφανές).
```

```
*
```

```
* Η αδιαφάνεια της γέμισης καθορίζει το πόσο διαφανές ή αδιαφανές θα εμφανίζεται το χρώμα  
της βούρτσας.
```

```
*/
```

```
float getFillOpacity(const graphics::Brush& brush) {  
    return brush.fill_opacity;  
}
```

```
/**
```

```
* Ρυθμίζει την αδιαφάνεια για το δευτερεύον χρώμα γέμισης του Brush.
```

```
* @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
```

```
* @param opacity Η αδιαφάνεια δευτερεύοντος χρώματος (0.0 έως 1.0).
```

```
*/
```

```
void setSecondaryFillOpacity(graphics::Brush& brush, float opacity) {  
    brush.fill_secondary_opacity = opacity;  
}
```

```

/**
 * Ρυθμίζει το χρώμα του περιγράμματος του Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 * @param r Η κόκκινη συνιστώσα του χρώματος περιγράμματος (0.0 - 1.0).
 * @param g Η πράσινη συνιστώσα του χρώματος περιγράμματος (0.0 - 1.0).
 * @param b Η μπλε συνιστώσα του χρώματος περιγράμματος (0.0 - 1.0).
 */
void setOutlineColor(graphics::Brush& brush, float r, float g, float b) {
    brush.outline_color[0] = r;
    brush.outline_color[1] = g;
    brush.outline_color[2] = b;
}

/**
 * Επιστρέφει το χρώμα περιγράμματος της βούρτσας `Brush`.
 *
 * @param brush Το αντικείμενο `graphics::Brush` από το οποίο θα ανακτηθεί το χρώμα
 περιγράμματος.
 * @return Μια `std::tuple` που περιέχει τις τιμές (r, g, b) του χρώματος περιγράμματος.
 *
 * Το χρώμα περιγράμματος χρησιμοποιείται για να ορίσει το εξωτερικό χρώμα γύρω από την
 κύρια γεμισμένη επιφάνεια.
 */
std::tuple<float, float, float> getOutlineColor(const graphics::Brush& brush) {
    return { brush.outline_color[0], brush.outline_color[1], brush.outline_color[2] };
}

/**
 * Ρυθμίζει την αδιαφάνεια του περιγράμματος του Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 * @param opacity Η αδιαφάνεια περιγράμματος (0.0 έως 1.0).
 */
void setOutlineOpacity(graphics::Brush& brush, float opacity) {
    brush.outline_opacity = opacity;
}

/**
 * Επιστρέφει την αδιαφάνεια του περιγράμματος της βούρτσας `Brush`.
 *
 * @param brush Το αντικείμενο `graphics::Brush` από το οποίο θα ανακτηθεί η αδιαφάνεια του
 περιγράμματος.
 * @return Μια τιμή `float` που αντιπροσωπεύει την αδιαφάνεια του περιγράμματος (0.0 για
 πλήρως διαφανές έως 1.0 για πλήρως αδιαφανές).
 *
 * Η αδιαφάνεια του περιγράμματος επηρεάζει το πόσο έντονο ή διαφανές εμφανίζεται το
 περίγραμμα της βούρτσας.
 */
float getOutlineOpacity(const graphics::Brush& brush) {
    return brush.outline_opacity;
}

```

```

/**
 * Ρυθμίζει το πλάτος του περιγράμματος του Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 * @param width Το πλάτος του περιγράμματος.
 */
void setOutlineWidth(graphics::Brush& brush, float width) {
    brush.outline_width = width;
}

/**
 * Επιστρέφει το πλάτος του περιγράμματος της βούρτσας `Brush`.
 *
 * @param brush Το αντικείμενο `graphics::Brush` από το οποίο θα ανακτηθεί το πλάτος
 περιγράμματος.
 * @return Μια τιμή `float` που αντιπροσωπεύει το πλάτος του περιγράμματος.
 *
 * Το πλάτος του περιγράμματος καθορίζει το πόσο παχύ θα εμφανίζεται το περίγραμμα γύρω
 από το σχήμα που δημιουργείται με τη βούρτσα.
 */
float getOutlineWidth(const graphics::Brush& brush) {
    return brush.outline_width;
}

/**
 * Εφαρμόζει υφή στο Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 * @param texture Το όνομα του αρχείου υφής.
 */
void setTexture(graphics::Brush& brush, const std::string& texture) {
    brush.texture = texture;
}

/**
 * Ενεργοποιεί ή απενεργοποιεί το gradient για το Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 * @param gradient Η λογική τιμή για ενεργοποίηση (true) ή απενεργοποίηση (false) του
 gradient.
 */
void enableGradient(graphics::Brush& brush, bool gradient) {
    brush.gradient = gradient;
}

/**
 * Ρυθμίζει την κατεύθυνση του gradient για το Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 * @param dir_u Η κατεύθυνση x του gradient.
 * @param dir_v Η κατεύθυνση y του gradient.
 */
void setGradientDirection(graphics::Brush& brush, float dir_u, float dir_v) {
    brush.gradient_dir_u = dir_u;
    brush.gradient_dir_v = dir_v;
}

```

```

}

/**
 * Ελέγχει αν είναι ενεργοποιημένη η γραμμική διαβάθμιση (gradient) για ένα αντικείμενο
 * `Brush`.
 * @param brush Το αντικείμενο `Brush` που εξετάζουμε.
 * @return `true` αν η γραμμική διαβάθμιση είναι ενεργή, διαφορετικά `false`.
 */
bool isGradient(const graphics::Brush& brush) {
    return brush.gradient;
}

/**
 * Ελέγχει αν το αντικείμενο `Brush` είναι διαφανές, δηλαδή αν η αδιαφάνεια του γεμίσματος
 * και του περιγράμματος είναι μηδενική.
 * @param brush Το αντικείμενο `Brush` που εξετάζουμε.
 * @return `true` αν το αντικείμενο `Brush` είναι διαφανές, διαφορετικά `false`.
 */
bool isTransparent(const graphics::Brush& brush) {
    return brush.fill_opacity == 0.0f && brush.outline_opacity == 0.0f;
}

/**
 * Επαναφέρει το Brush στις προεπιλεγμένες ρυθμίσεις.
 * @param brush Το αντικείμενο Brush που θα επαναφερθεί.
 */
void resetBrush(graphics::Brush& brush) {
    setFillColor(brush, 1.0f, 1.0f, 1.0f);
    setSecondaryFillColor(brush, 1.0f, 1.0f, 1.0f);
    setFillOpacity(brush, 1.0f);
    setSecondaryFillOpacity(brush, 1.0f);
    setOutlineColor(brush, 1.0f, 1.0f, 1.0f);
    setOutlineOpacity(brush, 1.0f);
    setOutlineWidth(brush, 1.0f);
    brush.texture = "";
    brush.gradient = false;
    setGradientDirection(brush, 0.0f, 1.0f);
}

/**
 * Ενεργοποιεί απαλό gradient στο Brush με δύο χρώματα.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 * @param r1, g1, b1 Το αρχικό χρώμα για το gradient.
 * @param r2, g2, b2 Το τελικό χρώμα για το gradient.
 */
void setSoftGradient(graphics::Brush& brush, float r1, float g1, float b1, float r2, float g2, float
b2) {
    setFillColor(brush, r1, g1, b1);
    setSecondaryFillColor(brush, r2, g2, b2);
    enableGradient(brush, true);
}

```

```

/**
 * Θέτει το χρώμα γέμισης και την αδιαφάνεια για το Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 * @param r, g, b Το χρώμα γέμισης.
 * @param opacity Η αδιαφάνεια.
 */
void setTransparentFillColor(graphics::Brush& brush, float r, float g, float b, float opacity) {
    setFillColor(brush, r, g, b);
    setFillOpacity(brush, opacity);
}

/**
 * Θέτει το χρώμα και την αδιαφάνεια για το περίγραμμα του Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 * @param r, g, b Το χρώμα περιγράμματος.
 * @param opacity Η αδιαφάνεια περιγράμματος.
 */
void setTransparentOutlineColor(graphics::Brush& brush, float r, float g, float b, float opacity) {
    setOutlineColor(brush, r, g, b);
    setOutlineOpacity(brush, opacity);
}

/**
 * Εναλλάσσει τη διαφάνεια της γέμισης του Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 */
void toggleFillTransparency(graphics::Brush& brush) {
    brush.fill_opacity = brush.fill_opacity == 1.0f ? 0.5f : 1.0f;
    brush.fill_secondary_opacity = brush.fill_secondary_opacity == 1.0f ? 0.5f : 1.0f;
}

/**
 * Ενεργοποιεί το gradient σε διαφανές φόντο για το Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 */
void enableGradientOnTransparentBG(graphics::Brush& brush) {
    brush.gradient = true;
    setSecondaryFillOpacity(brush, 0.0f);
}

/**
 * Ρυθμίζει το χρώμα γέμισης και το περίγραμμα του Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 * @param r1, g1, b1 Το χρώμα γέμισης.
 * @param r2, g2, b2 Το χρώμα περιγράμματος.
 */
void setPatternColorScheme(graphics::Brush& brush, float r1, float g1, float b1, float r2, float
g2, float b2) {
    setFillColor(brush, r1, g1, b1);
    setOutlineColor(brush, r2, g2, b2);
    setOutlineOpacity(brush, 0.8f);
}

```



```

/**
 * Ρυθμίζει την ημιδιαφάνεια του Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 * @param opacity Η τιμή ημιδιαφάνειας για γέμιση και περίγραμμα.
 */
void setSemiTransparentBackground(graphics::Brush& brush, float opacity) {
    setFillOpacity(brush, opacity);
    setOutlineOpacity(brush, opacity / 2);
}

/**
 * Ενεργοποιεί διπλό gradient στο Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 * @param fillR, fillG, fillB Το βασικό χρώμα.
 * @param secR, secG, secB Το δευτερεύον χρώμα.
 */
void setDualGradient(graphics::Brush& brush, float fillR, float fillG, float fillB, float secR, float
secG, float secB) {
    setFillColor(brush, fillR, fillG, fillB);
    setSecondaryFillColor(brush, secR, secG, secB);
    enableGradient(brush, true);
}

/**
 * Αντιστρέφει την κατεύθυνση του gradient για το Brush.
 * @param brush Το αντικείμενο Brush που θα τροποποιηθεί.
 */
void reverseGradientDirection(graphics::Brush& brush) {
    float temp = brush.gradient_dir_u;
    brush.gradient_dir_u = brush.gradient_dir_v;
    brush.gradient_dir_v = temp;
}

/**
 * Συνάρτηση που δημιουργεί και επιστρέφει ένα αντικείμενο `Brush` με το συγκεκριμένο χρώμα
 * που ορίζεται από το όνομα του χρώματος. Η συνάρτηση παρέχει υποστήριξη για αρκετά
 * χρώματα,
 * αντιστοιχίζοντας τα ονόματα τους σε RGB τιμές. Εάν το όνομα του χρώματος δεν βρεθεί στον
 * χάρτη χρωμάτων, η συνάρτηση επιστρέφει ένα `Brush` με μαύρο χρώμα και εμφανίζει μήνυμα
 * σφάλματος στην κονσόλα.
 *
 * ### Παράδειγμα Χρήσης:
 * @code
 * graphics::Brush redBrush = getBrushByName("red"); // Δημιουργεί ένα Brush με κόκκινο
 * χρώμα
 * graphics::Brush unknownBrush = getBrushByName("unknown"); // Δημιουργεί μαύρο Brush,
 * καθώς το "unknown" δεν υπάρχει
 * @endcode
 *
 * ### Υλοποίηση:
 * 1. Χρησιμοποιεί έναν `std::map` για να αποθηκεύσει τα προκαθορισμένα ονόματα χρωμάτων
 * και τις αντίστοιχες RGB τιμές τους.

```

```

* 2. Κάνει αναζήτηση στο χάρτη `colors` για το όνομα του χρώματος που παρέχεται ως είσοδος.
* 3. Αν το χρώμα βρεθεί, επιστρέφει ένα `Brush` με τις συγκεκριμένες τιμές RGB.
* 4. Αν το χρώμα δεν βρεθεί, εμφανίζει μήνυμα σφάλματος στην κονσόλα και επιστρέφει μαύρο
`Brush`.
*
* @param colorName Το όνομα του χρώματος που θέλουμε να ανακτήσουμε (π.χ. "red", "blue").
* @return Ένα αντικείμενο `Brush` με το χρώμα που αντιστοιχεί στο `colorName`.
*     Αν το `colorName` δεν βρεθεί, επιστρέφεται μαύρο χρώμα (RGB: 0.0, 0.0, 0.0).
*/
graphics::Brush getBrushByName(const std::string& colorName) {
    // Χάρτης με ονόματα χρωμάτων και τις αντίστοιχες τιμές RGB
    std::map<std::string, std::tuple<float, float, float>> colors = {
        {"red", {1.0f, 0.0f, 0.0f}}, {"orange", {1.0f, 0.5f, 0.0f}}, {"yellow", {1.0f, 1.0f, 0.0f}},
        {"green", {0.0f, 1.0f, 0.0f}}, {"blue", {0.0f, 0.0f, 1.0f}}, {"purple", {0.5f, 0.0f, 0.5f}},
        {"pink", {1.0f, 0.75f, 0.8f}}, {"brown", {0.65f, 0.16f, 0.16f}}, {"black", {0.0f, 0.0f, 0.0f}},
        {"white", {1.0f, 1.0f, 1.0f}}, {"gray", {0.5f, 0.5f, 0.5f}}, {"cyan", {0.0f, 1.0f, 1.0f}},
        {"magenta", {1.0f, 0.0f, 1.0f}}, {"maroon", {0.5f, 0.0f, 0.0f}}, {"navy", {0.0f, 0.0f, 0.5f}},
        {"olive", {0.5f, 0.5f, 0.0f}}, {"teal", {0.0f, 0.5f, 0.5f}}, {"lime", {0.75f, 1.0f, 0.0f}},
        {"beige", {0.96f, 0.96f, 0.86f}}, {"salmon", {1.0f, 0.55f, 0.41f}}, {"coral", {1.0f, 0.5f,
0.31f}},
        {"indigo", {0.29f, 0.0f, 0.51f}}, {"violet", {0.93f, 0.51f, 0.93f}}, {"lavender", {0.9f, 0.9f,
0.98f}},
        {"mint", {0.6f, 1.0f, 0.6f}}, {"peach", {1.0f, 0.87f, 0.68f}}, {"gold", {1.0f, 0.84f, 0.0f}},
        {"silver", {0.75f, 0.75f, 0.75f}}, {"bronze", {0.8f, 0.5f, 0.2f}}, {"turquoise", {0.25f, 0.88f,
0.82f}},
        {"emerald", {0.31f, 0.78f, 0.47f}}, {"sapphire", {0.06f, 0.32f, 0.73f}}, {"amber", {1.0f,
0.75f, 0.0f}},
        {"ruby", {0.88f, 0.07f, 0.37f}}, {"chocolate", {0.82f, 0.41f, 0.12f}}, {"ivory", {1.0f, 1.0f,
0.94f}},
        {"rose", {1.0f, 0.0f, 0.5f}}, {"lilac", {0.78f, 0.64f, 0.78f}}, {"fuchsia", {1.0f, 0.0f, 1.0f}},
        {"periwinkle", {0.8f, 0.8f, 1.0f}}, {"plum", {0.56f, 0.27f, 0.52f}}, {"rust", {0.72f, 0.25f,
0.05f}},
        {"mustard", {1.0f, 0.86f, 0.35f}}, {"mahogany", {0.75f, 0.25f, 0.0f}}, {"khaki", {0.76f,
0.69f, 0.57f}},
        {"blush", {0.87f, 0.36f, 0.51f}}, {"denim", {0.08f, 0.38f, 0.74f}}, {"charcoal", {0.21f,
0.27f, 0.31f}},
        {"copper", {0.72f, 0.45f, 0.2f}}, {"sand", {0.76f, 0.7f, 0.5f}}, {"moss", {0.6f, 0.8f, 0.2f}},
        {"sienna", {0.53f, 0.18f, 0.09f}}, {"chestnut", {0.8f, 0.36f, 0.36f}}, {"azure", {0.94f, 1.0f,
1.0f}},
        {"slate", {0.44f, 0.5f, 0.56f}}, {"cream", {1.0f, 0.99f, 0.82f}}, {"tangerine", {1.0f, 0.58f,
0.2f}},
        {"fern", {0.31f, 0.47f, 0.26f}}, {"honey", {1.0f, 0.87f, 0.67f}}, {"lavender blush", {1.0f,
0.94f, 0.96f}},
        {"midnight blue", {0.1f, 0.1f, 0.44f}}, {"sky blue", {0.53f, 0.81f, 0.92f}}, {"electric blue",
{0.49f, 0.98f, 1.0f}},
        {"powder blue", {0.69f, 0.88f, 0.9f}}, {"royal blue", {0.25f, 0.41f, 0.88f}}, {"neon green",
{0.22f, 1.0f, 0.08f}},
        {"apple green", {0.55f, 0.71f, 0.0f}}, {"forest green", {0.13f, 0.55f, 0.13f}}, {"sea green",
{0.18f, 0.55f, 0.34f}},
        {"mint green", {0.6f, 1.0f, 0.6f}}, {"grass green", {0.24f, 0.8f, 0.2f}}, {"crimson", {0.86f,
0.08f, 0.24f}},

```

```

        {"scarlet", {1.0f, 0.14f, 0.0f}}, {"burgundy", {0.5f, 0.0f, 0.13f}}, {"wine", {0.45f, 0.18f,
0.22f}},
        {"eggshell", {0.94f, 0.92f, 0.84f}}, {"almond", {0.94f, 0.87f, 0.8f}}, {"sepia", {0.44f, 0.26f,
0.08f}},
        {"peach puff", {1.0f, 0.85f, 0.73f}}, {"coral pink", {1.0f, 0.5f, 0.5f}}, {"deep pink", {1.0f,
0.08f, 0.58f}},
        {"hot pink", {1.0f, 0.41f, 0.71f}}, {"baby pink", {1.0f, 0.8f, 0.87f}}, {"lemon", {1.0f, 0.97f,
0.53f}},
        {"canary", {1.0f, 1.0f, 0.6f}}, {"vanilla", {0.95f, 0.9f, 0.67f}}, {"biscuit", {0.87f, 0.72f,
0.53f}},
        {"mocha", {0.74f, 0.52f, 0.42f}}, {"cinnamon", {0.82f, 0.41f, 0.12f}}, {"taupe", {0.28f,
0.24f, 0.2f}},
        {"ecru", {0.76f, 0.7f, 0.5f}}, {"mushroom", {0.6f, 0.5f, 0.48f}}, {"steel blue", {0.27f, 0.51f,
0.71f}},
        {"light cyan", {0.88f, 1.0f, 1.0f}}, {"teal blue", {0.21f, 0.46f, 0.53f}}, {"cerulean", {0.0f,
0.48f, 0.65f}},
        {"ocean blue", {0.0f, 0.47f, 0.75f}}, {"cobalt", {0.0f, 0.28f, 0.67f}}, {"prussian blue", {0.0f,
0.19f, 0.33f}},
        {"lavender gray", {0.77f, 0.76f, 0.82f}}, {"flamingo", {0.99f, 0.56f, 0.67f}}, {"terracotta",
{0.89f, 0.45f, 0.36f}},
        {"chartreuse", {0.5f, 1.0f, 0.0f}}, {"pistachio", {0.58f, 0.77f, 0.45f}}, {"malachite", {0.0f,
0.6f, 0.47f}},
        {"ultramarine", {0.07f, 0.04f, 0.56f}}, {"honeydew", {0.94f, 1.0f, 0.94f}}, {"coral reef",
{0.99f, 0.5f, 0.31f}},
        {"persian blue", {0.11f, 0.22f, 0.73f}}, {"persian green", {0.0f, 0.65f, 0.58f}}, {"carnation",
{1.0f, 0.65f, 0.79f}},
        {"bittersweet", {1.0f, 0.44f, 0.37f}}, {"claret", {0.5f, 0.09f, 0.2f}}, {"wisteria", {0.79f,
0.63f, 0.86f}},
        {"brick red", {0.8f, 0.25f, 0.33f}}, {"powder pink", {1.0f, 0.87f, 0.91f}}, {"aqua green",
{0.0f, 0.85f, 0.75f}},
        {"dusty rose", {0.72f, 0.41f, 0.47f}}, {"slate gray", {0.44f, 0.5f, 0.56f}}
    };

```

```

graphics::Brush brush;
auto it = colors.find(colorName);
if (it != colors.end()) {
    brush.fill_color[0] = std::get<0>(it->second);
    brush.fill_color[1] = std::get<1>(it->second);
    brush.fill_color[2] = std::get<2>(it->second);
}
else {
    std::cout << "The color " << colorName << " does not exist, use black instead." <<
std::endl;
    brush.fill_color[0] = 0.0f;
    brush.fill_color[1] = 0.0f;
    brush.fill_color[2] = 0.0f;
}
return brush;
}

```

/\*\*

\* Συνάρτηση που δημιουργεί ένα Brush με ορισμένο χρώμα και texture.

```

* Αν δεν δοθούν παράμετροι, η συνάρτηση θα χρησιμοποιήσει το "red" για το χρώμα
* και κενό string για το texture.
* @param color Το όνομα του χρώματος (default: "red")
* @param texture Το όνομα του texture (default: "")
* @return Ένα Brush με το ορισμένο χρώμα και texture
*/
graphics::Brush createBrush(const std::string& color = "red", const std::string& texture = "") {
    graphics::Brush brush = getBrushByName(color);
    brush.texture = texture;
    return brush;
}

/** * Δημιουργεί ένα νέο `Brush` αντικείμενο με καθορισμένες ή προκαθορισμένες
παραμέτρους. * @param fill_r Κόκκινο χρώμα γεμίσματος (default: 1.0f). * @param fill_g
Πράσινο χρώμα γεμίσματος (default: 1.0f). * @param fill_b Μπλε χρώμα γεμίσματος (default:
1.0f). * @param fill_opacity Αδιαφάνεια γεμίσματος (default: 1.0f). * @param outline_r Κόκκινο
χρώμα περιγράμματος (default: 0.0f). * @param outline_g Πράσινο χρώμα περιγράμματος (default:
0.0f). * @param outline_b Μπλε χρώμα περιγράμματος (default: 0.0f). * @param outline_opacity
Αδιαφάνεια περιγράμματος (default: 1.0f). * @param outline_width Πλάτος περιγράμματος
(default: 1.0f). * @param texture Υφή (default: κενή). * @param gradient Αν θα ενεργοποιηθεί
γραμμική διαβάθμιση (default: false). * @return Ένα `Brush` αντικείμενο με καθορισμένες
παραμέτρους. */
graphics::Brush createNewBrush(float fill_r = 1.0f, float fill_g = 1.0f, float fill_b = 1.0f, float
fill_opacity = 1.0f, float outline_r = 0.0f, float outline_g = 0.0f, float outline_b = 0.0f, float
outline_opacity = 1.0f, float outline_width = 1.0f, const std::string& texture = "", bool gradient =
false)
{
    graphics::Brush brush;
    brush.fill_color[0] = fill_r;
    brush.fill_color[1] = fill_g;
    brush.fill_color[2] = fill_b;
    brush.fill_opacity = fill_opacity;
    brush.outline_color[0] = outline_r;
    brush.outline_color[1] = outline_g;
    brush.outline_color[2] = outline_b;
    brush.outline_opacity = outline_opacity;
    brush.outline_width = outline_width;
    brush.texture = texture;
    brush.gradient = gradient;

    return brush;
}

} // namespace brush_utils

#endif // BRUSH_UTILS_H

```

## Πρόγραμμα δοκιμής brush\_utils.h

```
#include "sgg/graphics.h"
#include "brush_utils.h"
#include <iostream>
#include <tuple>

void demonstrateBrushFunctions() {
    // 1. Δημιουργία βασικού Brush με προκαθορισμένα χρώματα και ιδιότητες
    graphics::Brush myBrush = brush_utils::createNewBrush();
    std::cout << "Brush created on default values." << std::endl;

    // 2. Εφαρμογή κύριου χρώματος γεμίσματος (κόκκινο)
    brush_utils::setFillColor(myBrush, 1.0f, 0.0f, 0.0f);
    std::cout << "Main Brush fill color defined on red." << std::endl;

    // 3. Ανάκτηση του κύριου χρώματος γεμίσματος
    //auto [fillR, fillG, fillB] = brush_utils::getFillColor(myBrush);
    std::tuple<float, float, float> fillColor = brush_utils::getFillColor(myBrush);
    float fillR = std::get<0>(fillColor);
    float fillG = std::get<1>(fillColor);
    float fillB = std::get<2>(fillColor);
    std::cout << "Main brush fill color using RGB(" << fillR << ", " << fillG << ", "
    << fillB << ")" << std::endl;

    // 4. Εφαρμογή δευτερεύοντος χρώματος γεμίσματος για gradient εφέ (μπλε)
    brush_utils::setSecondaryFillColor(myBrush, 0.0f, 0.0f, 1.0f);
    std::cout << "Secondry brush fill color defined on blue." << std::endl;

    // 5. Ανάκτηση του δευτερεύοντος χρώματος γεμίσματος
    //auto [secFillR, secFillG, secFillB] =
    brush_utils::getFillSecondaryColor(myBrush);
    std::tuple<float, float, float> secFillColor =
    brush_utils::getFillSecondaryColor(myBrush);
    float secFillR = std::get<0>(secFillColor);
    float secFillG = std::get<1>(secFillColor);
    float secFillB = std::get<2>(secFillColor);

    std::cout << "Secondary brush fill color using RGB(" << secFillR << ", " <<
    secFillG << ", " << secFillB << ")" << std::endl;

    // 6. Εφαρμογή αδιαφάνειας γεμίσματος
    brush_utils::setFillOpacity(myBrush, 0.8f);
    std::cout << "FillOpacity defined 0.8." << std::endl;

    // 7. Ανάκτηση αδιαφάνειας γεμίσματος
    float fillOpacity = brush_utils::getFillOpacity(myBrush);
    std::cout << "FillOpacity is " << fillOpacity << std::endl;

    // 8. Εφαρμογή χρώματος περιγράμματος (πράσινο)
    brush_utils::setOutlineColor(myBrush, 0.0f, 1.0f, 0.0f);
    std::cout << "Brush OutlineColor defined on green." << std::endl;

    // 9. Ανάκτηση του χρώματος περιγράμματος
    //auto [outlineR, outlineG, outlineB] = brush_utils::getOutlineColor(myBrush);
    std::tuple<float, float, float> outlineColor =
    brush_utils::getOutlineColor(myBrush);
    float outlineR = std::get<0>(outlineColor);
    float outlineG = std::get<1>(outlineColor);
    float outlineB = std::get<2>(outlineColor);

    std::cout << "OutlineColor using RGB(" << outlineR << ", " << outlineG << ", " <<
    outlineB << ")" << std::endl;
```

```

// 10. Εφαρμογή αδιαφάνειας περιγράμματος
brush_utils::setOutlineOpacity(myBrush, 0.9f);
std::cout << "OutlineOpacity defined on 0.9." << std::endl;

// 11. Ανάκτηση της αδιαφάνειας περιγράμματος
float outlineOpacity = brush_utils::getOutlineOpacity(myBrush);
std::cout << "Outline Opacity is " << outlineOpacity << std::endl;

// 12. Εφαρμογή πλάτους περιγράμματος
brush_utils::setOutlineWidth(myBrush, 2.0f);
std::cout << "OutlineWidth defined on 2.0." << std::endl;

// 13. Ανάκτηση πλάτους περιγράμματος
float outlineWidth = brush_utils::getOutlineWidth(myBrush);
std::cout << "OutlineWidth is " << outlineWidth << std::endl;

// 14. Εφαρμογή gradient στο Brush
brush_utils::enableGradient(myBrush, true);
std::cout << "Brush gradient is enabled ." << std::endl;

// 15. Έλεγχος αν το gradient είναι ενεργοποιημένο
bool gradientStatus = brush_utils::isGradient(myBrush);
std::cout << "Brush gradient is enabled : " << (gradientStatus ? "Ναι" : "Όχι") <<
std::endl;

// 16. Ρύθμιση κατεύθυνσης gradient
brush_utils::setGradientDirection(myBrush, 1.0f, 0.0f);
std::cout << "GradientDirection defined on (1.0, 0.0)." << std::endl;

// 17. Επαναφορά του Brush στις προκαθορισμένες ρυθμίσεις
brush_utils::resetBrush(myBrush);
std::cout << "Reset Brush on default." << std::endl;

// 18. Έλεγχος αν το Brush είναι διαφανές
bool transparencyStatus = brush_utils::isTransparent(myBrush);
std::cout << "Brush is transparent : " << (transparencyStatus ? "Yes" : "No") <<
std::endl;

}

int main() {
    graphics::createWindow(800, 600, "Brush Utilities Demo");

    demonstrateBrushFunctions(); // Κλήση της συνάρτησης επίδειξης για τις λειτουργίες
    του Brush

    graphics::destroyWindow();
    return 0;
}

```

## Πρόγραμμα δοκιμής getBrushByName()

```
#include "sgg/graphics.h"
#include "brush_utils.h" // περιέχει την getBrushByName
#include <iostream>

// Μέγεθος παραθύρου
const float WINDOW_WIDTH = 800;
const float WINDOW_HEIGHT = 600;

/**
 * Η συνάρτηση `drawExamples` χρησιμοποιεί τη `getBrushByName`
 * για να δημιουργήσει αντικείμενα `Brush` με διαφορετικά χρώματα
 * και να τα εφαρμόσει σε βασικά σχήματα για να δείξει τη λειτουργικότητα της
 * συνάρτησης.
 */
void drawExamples() {
    // Χρησιμοποιούμε διάφορα χρώματα μέσω της getBrushByName
    graphics::Brush redBrush = brush_utils::getBrushByName("red"); // Δημιουργία
    κόκκινου Brush
    graphics::Brush greenBrush = brush_utils::getBrushByName("green"); // Δημιουργία
    πράσινου Brush
    graphics::Brush blueBrush = brush_utils::getBrushByName("blue"); // Δημιουργία
    μπλε Brush
    graphics::Brush yellowBrush = brush_utils::getBrushByName("yellow"); // Δημιουργία
    κίτρινου Brush
    graphics::Brush unknownBrush = brush_utils::getBrushByName("unknown"); //
    Ανύπαρκτο χρώμα (επιστρέφει μαύρο)

    // Σχεδιάζουμε ένα κύκλο με κόκκινο χρώμα
    graphics::drawDisk(200, 150, 50, redBrush);

    // Σχεδιάζουμε ένα ορθογώνιο με πράσινο χρώμα
    graphics::drawRect(400, 150, 100, 50, greenBrush);

    // Σχεδιάζουμε ένα τετράγωνο με μπλε χρώμα
    graphics::drawRect(600, 150, 50, 50, blueBrush);

    // Σχεδιάζουμε έναν δίσκο με κίτρινο χρώμα
    graphics::drawDisk(300, 350, 40, yellowBrush);

    // Σχεδιάζουμε έναν κύκλο με μαύρο χρώμα (άγνωστο όνομα)
    graphics::drawDisk(500, 350, 40, unknownBrush);

    // Κείμενο που εξηγεί την χρήση του μαύρου Brush για άγνωστο χρώμα
    graphics::setFont("bin/assets/Roboto-Bold.ttf");
    graphics::drawText(400, 450, 20, "Unknown color defaults to black", unknownBrush);
}

int main() {
    // Δημιουργία παραθύρου
    graphics::createWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "getBrushByName() Example");

    // Ρύθμιση συνάρτησης σχεδίασης
    graphics::setDrawFunction(drawExamples);

    // Έναρξη κύκλου μηνυμάτων
    graphics::startMessageLoop();

    return 0;
}
```