

תצוגת TFT וארדואינו

תצוגות TFT נפוצות היום ביישומים רבים החל מתצוגות קטנות , עבור דרך תצוגה של טלפון נייד ועד למסכים גדולים . מחירם הזול יחסית והצגת הצבעים והטקסט שלהן גורמים לשימוש הרב. אנחנו נעסוק בפרק זה בתצוגות TFT שמתחברות אל כרטיסי ארדואינו והפונקציות הנפוצות להפעלתן. ישנם מספר אנשים או חברות שכתבו את הפונקציות שנביא בהמשך ויש לרשום בתחילת תכנית `#include` לקבצים אלו. בין הספריות נזכיר את :

1. האתר של `adafruit` שבו יש חומר רב על פרויקטים כולל מכירת ציוד.

<https://www.adafruit.com/>

הספרייה שלהם על תצוגות TFT שימושי מאד ואת הקבצים ניתן להוריד באתר :

<https://github.com/adafruit/Adafruit-GFX-Library>

הסבר באנגלית על הפונקציות ניתן למצוא בהסבריו של יוצר הספרייה `Phillip Burgess` שבהם נעזרתי בהסבריי.

ספרייה נוספת נכתבה על ידי `prentice david` עבור תצוגות של `mcufriend` ניתן להוריד את הקבצים באתר

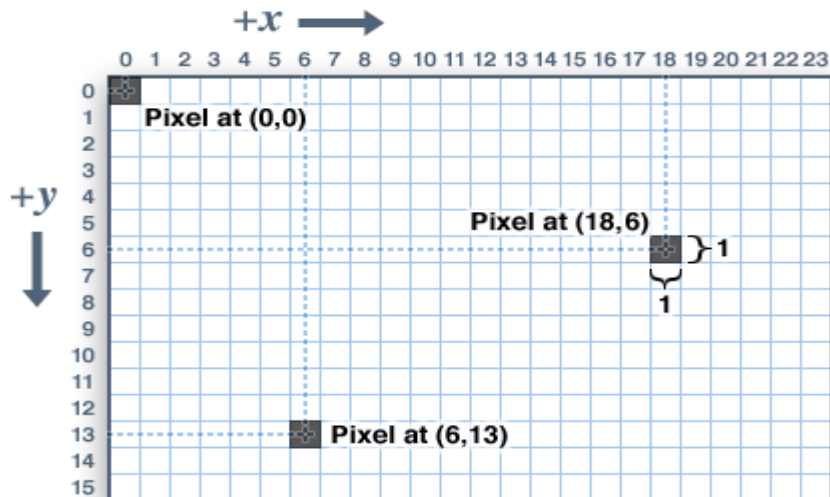
https://github.com/prenticedavid/MCUFRIEND_kbv

באתר של ארדואינו ניתן למצוא `TFT Library` שהיא הרחבה לאלו של `adafruit-GFX-Library` . באתר <https://www.arduino.cc/en/Reference/TFTLibrary> ניתן למצוא הרחבה בנושא.

1. מערכת הקואורדינטות והיחידות

התמונה בתצוגת ה `TFT` מורכבת מפיקסלים. תצוגת `TFT` של `320*480` מורכבת ממכפלה של 2 המספרים הנותנת 153600 פיקסלים. אנחנו יכולים לשלוט על הצבע של כל פיקסל ומכאן שניתן להציג בתצוגה מנקודה בודדת ועד תמונה מורכבת.

לכל פיקסל יש קואורדינטות של `X` ושל `Y` משלו. מערכת הקואורדינטות ממקמת את הראשית `(0,0)` בפינה השמאלית העליונה של התצוגה כאשר `X` גדל לכיוון ימין ו `Y` כלפי מטה כפי שנראה באיור הבא:



איור 1 : הפיקסלים בתצוגה

ברישום הקואורדינטות, קודם נרשמת הנקודה בציר ה X של הפיקסל ולאחריה בציר ה Y .

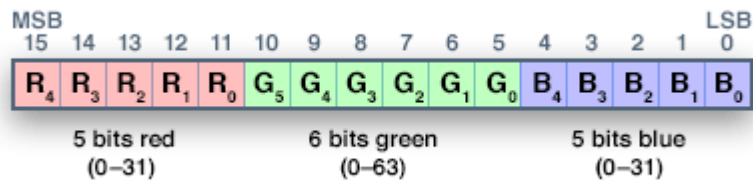
הקואורדינטות מבוטאות ביחידות של פיקסל ולא בגדלים של אורך (מ"מ או ס"מ אינצ'ים וכו). גודל מלבן שנצייר בהמשך יהיה במספר פיקסלים שלו בציר ה X ומספר הפיקסלים בציר ה Y ולא באורך של מ"מ או ס"מ. ניתן למדוד את אורך ורוחב המסך ואז לחלק בכמות הפיקסלים בכל ציר ולדעת את הגודל של כל פיקסל, אם כי הדבר פחות שימושי.

ניתן לשנות את האוריינטציה של התמונה כך שהאורך יהיה רוחב והרוחב יהיה אורך. כלומר ממצב של



התצוגה הימנית נקראת פורמט landscape (נוף) והימנית פורמט portrait (דיוקן). ראשית הצירים היא הנקודה השמאלית למעלה. ניתן לקבל 4 סוגי אוריינטציה, כלומר איזו אחת מ 4 הפינות של המלבן תהיה ראשית הצירים (0,0) .

בתצוגות התומכות בצבע, הצבע נרשם כערך בן 16 סיביות לא מסומן (unsigned – כלומר רק חיובי). בחלק מהתצוגות ניתן לקבל יותר צבעים ובחלק פחות – יותר או פחות סיביות בערך. רוב הספריות שעובדות עם תצוגות כאלו עובדות עם 16 סיביות. גם לארדואינו קל לעבוד עם צבעים בני 16 ביטים. שלושת צבעי היסוד - RGB – אדום Red ירוק Green כחול Blue - נכנסים בתוך ה 16 סיביות בצורה הבאה – איור 2 :



איור 2 : חלוקת הצבעים בתוך ערך של 16 סיביות

מהאיור רואים :

5 הביטים הגבוהים הם של הצבע האדום והם בני 5 ביטים (מספר בין 0 ל 31).

6 הביטים הבאים הם הירוק (מספר בין 0 ל 63) .

5 הביטים הנמוכים הם של הכחול (מספר בין 0 ל 31).

לצבע הירוק יש 6 ביטים כי העין של האדם רגישה יותר לצבע הירוק.

לדוגמא :

אם נרצה לצבוע פיקסל בצבע ירוק בלבד נשלח את הערך : **00000 11111 00000 = 07E0H**

הערך : **F800H=1111100000000000** יצבע את הפיקסל באדום והערך **FFFFH** יצבע את הפיקסל בלבן.

הצבעים הנפוצים הם : (שים לב שהתווים גדולים - CAPITAL LETTERS)

```

1. // Color definitions
2. #define BLACK    0x0000    // שחור
3. #define BLUE    0x001F    // כחול
4. #define RED     0xF800    // אדום
5. #define GREEN   0x07E0    // ירוק
6. #define CYAN   0x07FF    // כחול ירוק
7. #define MAGENTA 0xF81F    // גון אדום
8. #define YELLOW  0xFFE0    // צהוב
9. #define WHITE   0xFF      // לבן
    
```

בתצוגות ללא צבע נשים בפיקסל או 1 או 0 וכך הוא ייראה או לא ייראה.

פונקציות הגרפיקה

לכל תצוגת TFT יש בקר תצוגה פנימי המנהל את התצוגה עצמה ואת הקשר עם העולם שמחוץ לתצוגה שבו יש בדרך כלל מיקרו בקרים השולחים פקודות להפעלת התצוגה. לכל בקר יש פונקציית אתחול משלו. כדאי לראות בדפי הנתונים של התצוגה מי הבקר המפעיל אותה. במידה ולא יודעים אז בפונקציית

האתחול של התצוגה מבצעים קריאה של ה ID (זהות) של הבקר והפונקציה יודעת לאתחל את הבקר לפי ה ID שקראה ממנו.

הפונקציות שנרשום הן prototypes - אבי טיפוס. נצא מההנחה שבוצע אתחול נכון לתצוגה.

א. ציור פיקסל בצבע הרצוי

הפונקציה המציירת נקודת פיקסל בצבע הרצוי היא :

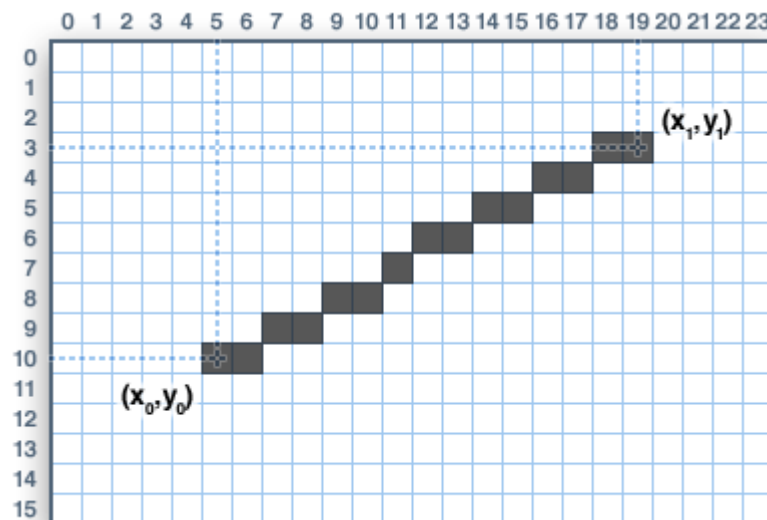
void drawPixel(uint16_t x, uint16_t y, uint16_t color);

הפונקציה מקבלת 3 ערכים מטיפוס uint16_t כלומר מספר שלם לא מסומן בן 16 ביט. הערך הראשון והשני הם קואורדינטות X ו Y של הנקודה והערך השלישי הוא הצבע. הפונקציה לא מחזירה ערך.

ב. ציור קו בצבע הרצוי

הפונקציה מציירת קו בין 2 נקודות. הקו יכול להיות אלכסוני. הפונקציה מקבלת 5 ערכים שלמים לא מסומנים המחולקים כך : 2 ערכים ראשונים הם קואורדינטות X ו Y של ראשית הקו והם נקראים x0,y0. 2 הערכים הבאים הם קואורדינטות X ו Y של סוף הקו והם נקראים x1,y1 והערך החמישי הוא הצבע הרצוי – color. הפונקציה לא מחזירה ערך.

void drawLine(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t color);



איור 3: ציור קו בין הנקודות x0,y0 ועד x1,y1 בצבע color הרצוי.

זימון הפונקציה במקרה של איור 3 יהיה :

```
drawLine(5,10,19,3,RED);
```

אם רוצים לצייר קו אופקי או אנכי (לא מלוכסן כמו באיור שלמעלה) נשתמש ב 2 פונקציות נוספות העובדות מהר יותר.

1. void drawFastVLine(uint16_t x0, uint16_t y0, uint16_t length, uint16_t color);

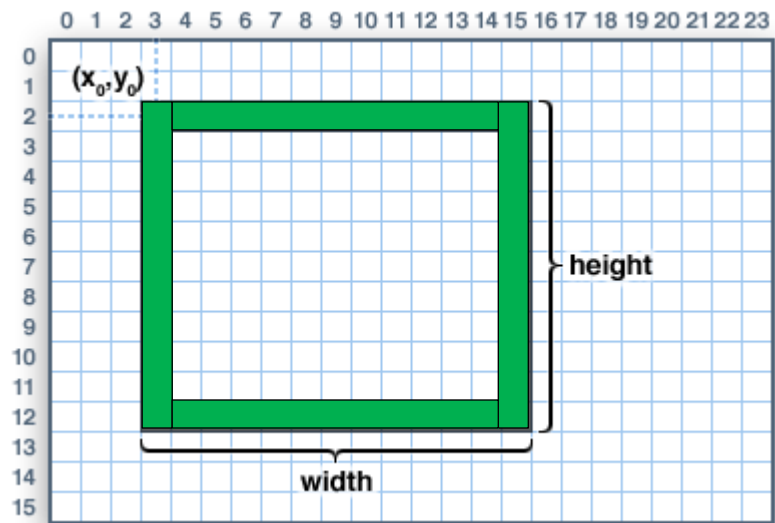
2. void drawFastHLine(uint8_t x0, uint8_t y0, uint8_t length, uint16_t color);

פונקציה 1 מציירת קו אנכי (V – Vertical) ופונקציה 2 קו אופקי (H-Horizontal) . הנתונים שהפונקציה מקבלת הם כמו בציור קו אלכסוני.

ג. ציור מלבן ומילוי מלבן בצבע

ניתן לצייר מלבן או ריבוע בצבע הרצוי. כמו כן ניתן למלא מלבן או ריבוע בצבע רצוי. יש לשלוח לפונקציה את קואורדינטות ראשית המלבן x_0, y_0 , את הרוחב שלו w - Width (מספר הפיקסלים בציר X) ואת הגובה שלו h - Height (מספר הפיקסלים בציר Y) ואת הצבע הרצוי $color$. הפונקציות לא מחזירות ערך.

void drawRect(uint16_t x0, uint16_t y0, uint16_t w, uint16_t h, uint16_t color);



איור 4 : ציור מלבן

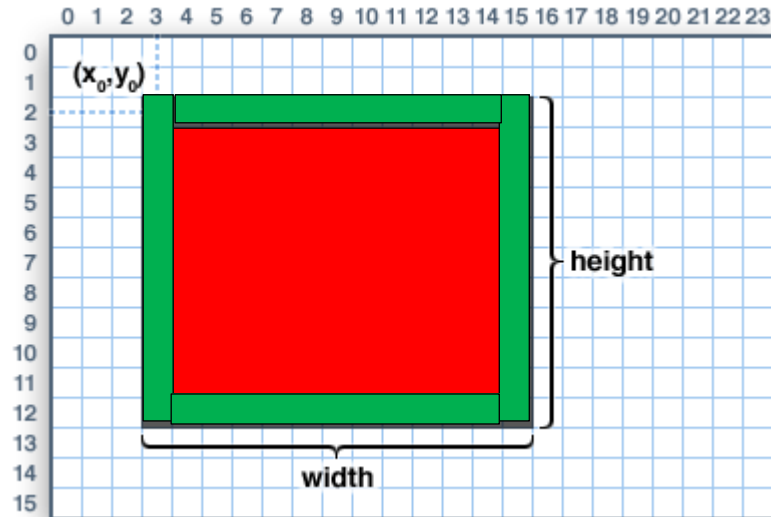
הפקודה שנשלח בדוגמה כאן היא :

`drawRect(3,2,13,11,GREEN);`

כדי למלא מלבן בצבע הרצוי נשתמש בפונקציה `fillRect` הנראית כך :

void fillRect(uint16_t x0, uint16_t y0, uint16_t w, uint16_t h, uint16_t color);

הפרמטרים שהפונקציה מקבלת הם כמו בפונקציה `drawRect`. הפונקציה איננה מחזירה ערך.



איור 5 : מילוי מלבן בצבע אדום

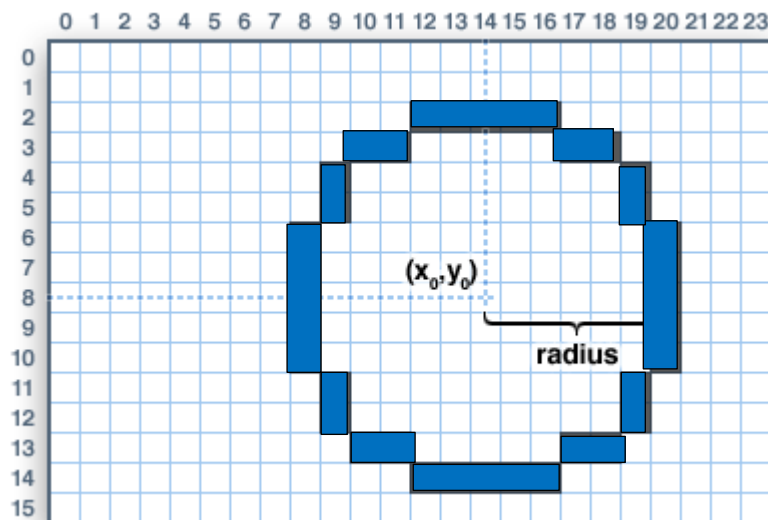
הפקודה שנשלח :

```
fillRect(4,3,12,10,RED);
```

ד. ציור מעגלים ומילוי מעגלים בצבע

גם כאן יש פונקציה לציור מעגל ויש פונקציה למילוי מעגל בצבע. כל אחת מהפונקציות מקבלת 4 ערכים מטיפוס שלם לא מסומן לפי הסדר הבא : 2 הערכים הראשונים הם קואורדינטות של x_0, y_0 המציינות את מרכז המעגל. הערך הבא הוא רדיוס המעגל r והערך הרביעי הוא הצבע $color$ הרצוי. הפונקציות לא מחזירות ערך.

```
void drawCircle(uint16_t x0, uint16_t y0, uint16_t r, uint16_t color);
```



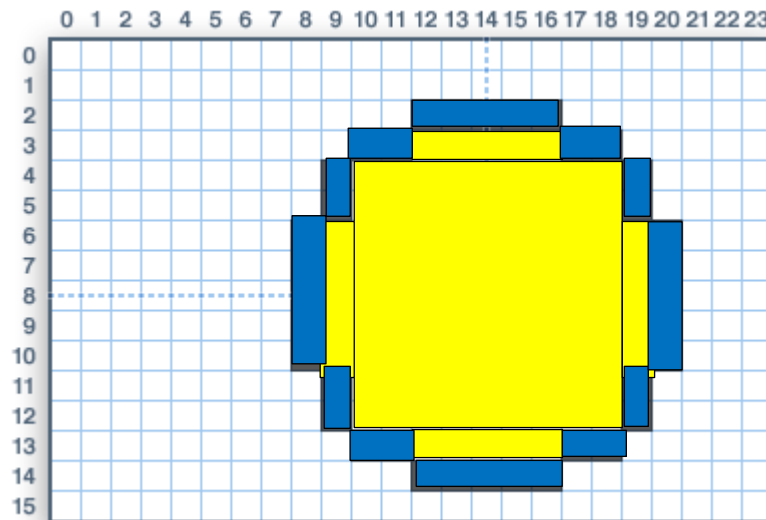
איור 6 : ציור מעגל בצבע כחול

הפקודה שניתן על פי איור 6 היא :

```
drawCircle ( 14,8, 7,BLUE);
```

כדי למלא מעגל נשתמש בפונקציה fillCircle(). היא ממלאת את המעגל בצבע רצוי.

```
void fillCircle(uint16_t x0, uint16_t y0, uint16_t r, uint16_t color);
```



איור 7 : מילוי מעגל בצבע צהוב

הפקודה שניתן היא :

```
fillRect(14,8,6,YELLOW);
```

ה. ציור מלבן עם פינות מעוגלות

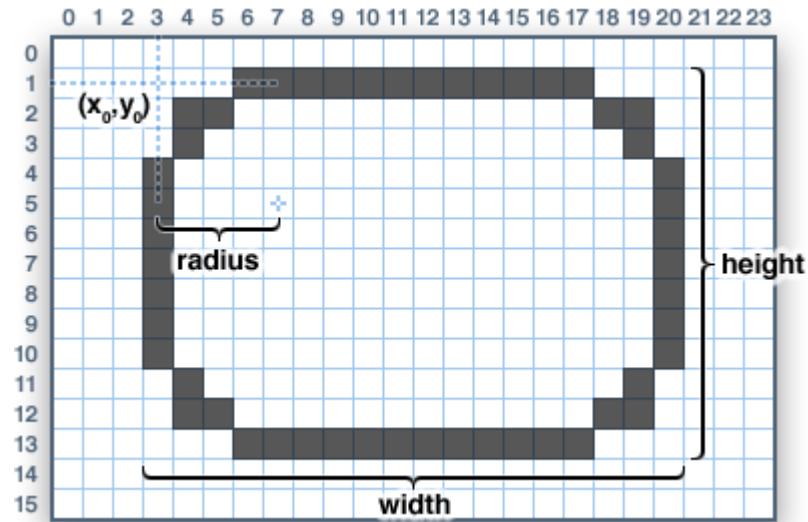
גם כאן יש 2 פונקציות. האחת של ציור של מלבן עם פינות מעוגלות (כמו באיור הבא) והשנייה היא מילוי המלבן בצבע. הפונקציות הן בהתאמה :

```
1. void drawRoundRect(uint16_t x0, uint16_t y0, uint16_t w, uint16_t h, uint16_t radius, uint16_t color);
```

```
2. void fillRoundRect(uint16_t x0, uint16_t y0, uint16_t w, uint16_t h, uint16_t radius, uint16_t color);
```

כל פונקציה מקבלת 6 ערכים ולא מחזירה ערך. הערכים המתקבלים הם : הנקודות x0,y0 של הפינה העליונה משמאל של המלבן (כאילו לא הייתה מעוגלת). הערך השלישי הוא הרוחב (כמות הפיקסלים

הערך הרביעי הוא הגובה (כמות הפיקסלים בציר Y) הערך החמישי הוא הרדיוס של הפינות המעוגלות והערך השישי הוא הצבע הרצוי.



איור 8 : ציור מלבן עם פינות מעוגלות

בדוגמה של האיור נשלח את הפקודה :

```
drawRoundRect(3,1,18,13,4,BLACK);
```

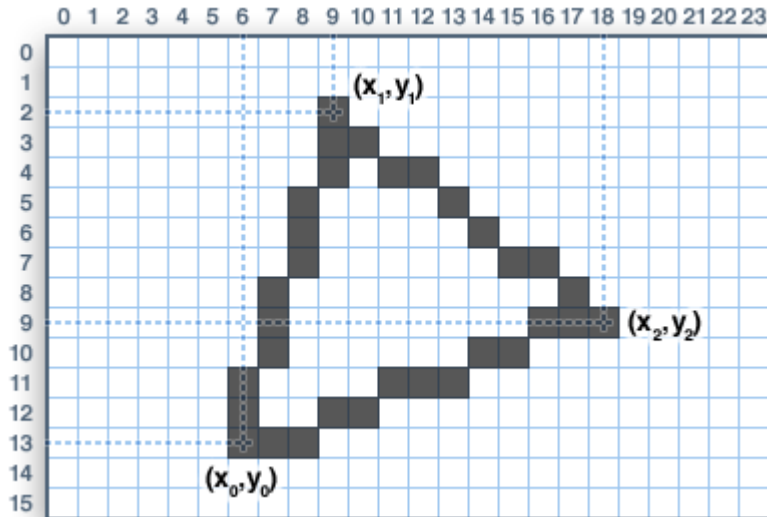
1. ציור משולש ומילוי משולש

גם בציור משולשים יש 2 פונקציות. האחת מאפשרת לצייר משולש בצבע רצוי והשנייה מאפשרת למלא משולש בצבע רצוי. הפונקציות הן :

1. `void drawTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t color);`
2. `void fillTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t color);`

כל אחת מהפונקציות מקבלת 7 ערכים ואיננה מחזירה ערך. הערכים שהפונקציה מקבלת הם 6 קואורדינטות של X ושל Y שהם קודקודי המשולש ועוד ערך שביעי שהוא הצבע. באיור שבהמשך רואים את המשולש המתקבל. בדוגמה הזו הפקודה שנשלח היא :

```
drawtriangle( 6,13,9,2,18,9,BLACK);
```

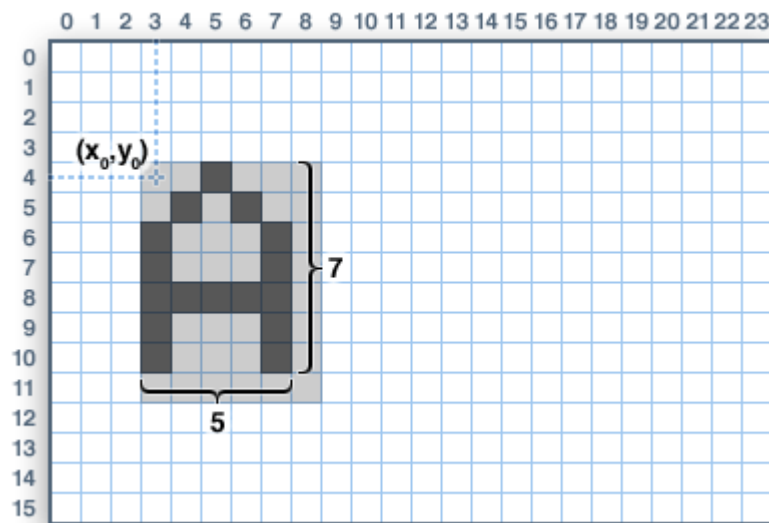


איור 9 : ציור משולש

ז. תווים וטקסט

ישנן 2 פונקציות כדי לצייר טקסט. האחת היא עבור תו בודד והשנייה עבור ציור טקסט (מספר תווים). ניתן למקם את התו בכל מקום שנרצה ובאיזה צבע רצוי. יש רק פונט אחד והוא צריך להיות של 5×8 פיקסלים. ניתן לשלוח מקדם או פקטור שיקבע את הגודל של הפונט במכפלות. למשל אם נשלח מקדם של 2 התו יהיה 16×10 פיקסלים. אם נשלח מקדם של 3 התו יהיה 15×24 פיקסלים וכך הלאה. הסיבה לשימוש בפונט אחד הוא הקטנת נפח התכנית. בגרסאות חדשות יותר ניתן להוסיף פונטים נוספים בעזרת המשפט Using font. הפונקציה המציירת תו נקראת drawChar והיא מקבלת 6 ערכים ולא מחזירה ערך. האיור הבא יעזור בהבנה של ציור תו.

```
void drawChar(uint16_t x, uint16_t y, char c, uint16_t color, uint16_t bg, uint8_t size);
```



איור 10 : ציור תו

הפונקציה מקבלת 6 ערכים לפי הסדר הבא: קואורדינטות תחילת כתיבת התו x,y (נקראים x0,y0 באיור). הערך השלישי הוא התו עצמו. הערך הרביעי הוא צבע התו color. הערך החמישי הוא רקע הפיקסלים bg – back ground, הערך השישי הוא המקדם שקובע את הגודל שלו size.

כתיבת טקסט עובדת מעט אחרת. אין פונקציה אחת שבה ניתן לקבוע גם את המיקום, גם את הצבע של הטקסט, גם את הרקע וגם את המקדם של הפונט. לפני ציור הטקסט, יש לזמן פונקציות הקובעות את מיקום תחילת הטקסט, את צבע הטקסט הרצוי, גודל מקדם הפונט ורק אז נזמן את הפונקציה print(). האפשרויות להדפסה דומות לאלו של Serial.print() של התקשורת הטורית. השלבים הם:

1. קביעת מיקום התחלת הצגת הטקסט (צד שמאל עליון של התו הראשון בטקסטטקסט – נקודות x0,y0).

1. void setCursor(uint16_t x0, uint16_t y0);

2. קביעת צבע הטקסט (ברירת המחדל – צבע לבן).

2. void setTextColor(uint16_t color);

3. אופציה נוספת היא לקבוע את צבע הטקסט ואת צבע הרקע שעליו הוא יוצג (במקום סעיף 2)

3. void setTextColor(uint16_t color, uint16_t backgroundcolor);

4. קביעת גודל מקדם הפונט.

4. void setTextSize(uint8_t size);

5. האם לבצע עטיפה (כיסוי) WRAP, אם שולחים לפונקציה w=true אז כאשר יש טקסט ארוך הוא חוזר לתחילת השורה ומכסה את התחלת הטקסט אם רשום שם טקסט אחר. ניתן להשתמש ב w=false ואז זה לא יקרה.

5. void setTextWrap(boolean w);

אחרי ביצוע השלבים הקודמים נשתמש בפונקציה print() או println(). לדוגמה print("Hello World"); תדפיס במסך את המחרוזת המבוקשת במיקום שרצינו ובצבע הטקסט הרצוי. ניתן להדפיס גם מספרים ומשתנים ואפילו בבסיס הרצוי כמו print(0xA1BF,HEX); תדפיס במסך A1BF. הפונקציה println() עובדת כמו הפונקציה print() רק שבסיום היא יורדת לשורה הבאה.

ה. מפת ביטים Biymaps

ניתן לצייר מפת ביטים מונוכרום (צבע אחד) . הדבר שימושי ביצירת אנימציות או icons (צלמיות). הפונקציה המבצעת זאת היא drawBitmap() .

1. void drawBitmap(int16_t x, int16_t y, uint8_t *bitmap, int16_t w, int16_t h, uint16_t color);

הפונקציה עוסקת בבלוק ביטים עוקבים להצגה , כאשר '1' שם בפיקסל את הצבע color ו'0' מדלג על הביט. x ו y הם נקודה שמאלית עליונה שבה מפת הביטים תצויר. *bitmap היא שם המערך של מפת הביטים. w - הוא הרוחב - כמות הפיקסלים בציר x ו h - כמות הפיקסלים בציר y . מפת הביטים חייבת להימצא בזיכרון התכנית ולכן בהגדרת מפת הביטים משתמשים בהנחיה PROGRAMMEM שאומרת לקומפיילר לשים את מפת הביטים בזיכרון התכנית. כדי ליצור מפת ביטים ניתן להשתמש במחולל הנמצא באתר :

[Here's a handy webtool for generating bitmap -> memorymaps](#)

ט. ניקוי או מילוי מסך

הפונקציה fillScreen() תעביר את כל התצוגה לצבע רצוי. על ידי מחיקה של כל התוכן שהיה במסך.

1. void fillScreen(uint16_t color);

הפונקציה מקבלת את הצבע שבו רוצים למלא את המסך והיא ולא מחזירה ערך.

י. סיבוב התצוגה

ניתן לסובב את הציור ב 0, 90, 180 או 270 מעלות . הפונקציה setRotation() מבצעת את הסיבוב.

void setRotation(uint8_t rotation);

הפרמטר rotation יכול להיות 0, 1, 2 או 3 . לתצוגות המותאמות " לשבת " על הדקי הארדואינו הפרמטר 0 נותן תמונת portrait . הפרמטר 1 נותן landscape . הפרמטר 2 נותן תמונת portrait הפוכה לזה של 0 והפרמטר 3 נותן תמונת landscape הפוכה לזה של 1 . כאשר מסובבים אז נקודת הראשית 0,0 משתנה.

אם רוצים לדעת את מידות המסך שמשתנה בין תמונת portrait ל landscape ניתן להשתמש בפונקציות **width()** המחזירה כמה פיקסלים יש בציר x והפונקציה **height()** המחזירה כמה פיקסלים יש בציר y . 2 הפונקציות לא מקבלות ערך ומחזירות ערך מטיפוס שלם חיובי.

יא. שימוש בפונטים (גופנים)

בגרסאות חדשות יותר של Adafruit GFX library יש אפשרות להשתמש בגופנים אחרים מהגופנים הסטנדרטיים עם הגודל הקבוע. יש גם אפשרות להוספת גופנים חדשים.

הגופנים הכלולים הם "Serif" המזכיר את Times New Roman , "sans" המזכיר את Helvetica או Arial ו "Mono" המזכיר את Courier . כל אחד יכול להיות במספר סגנונות (bold (מודגש) או italic (נטוי) וכו') ובמספר גדלים. הגופנים הם בפורמט של bitmap .

קבצי הכותר נמצאים בספרייה Fonts של Adafruit_GFX ויש לבצע הכללה #include לסגנון הרצוי . הקבצים בספרייה Fonts הם :

1. FreeMono12pt7b.h	FreeSansBoldOblique12pt7b.h
2. FreeMono18pt7b.h	FreeSansBoldOblique18pt7b.h
3. FreeMono24pt7b.h	FreeSansBoldOblique24pt7b.h
4. FreeMono9pt7b.h	FreeSansBoldOblique9pt7b.h
5. FreeMonoBold12pt7b.h	FreeSansOblique12pt7b.h
6. FreeMonoBold18pt7b.h	FreeSansOblique18pt7b.h
7. FreeMonoBold24pt7b.h	FreeSansOblique24pt7b.h
8. FreeMonoBold9pt7b.h	FreeSansOblique9pt7b.h
9. FreeMonoBoldOblique12pt7b.h	FreeSerif12pt7b.h
10. FreeMonoBoldOblique18pt7b.h	FreeSerif18pt7b.h
11. FreeMonoBoldOblique24pt7b.h	FreeSerif24pt7b.h
12. FreeMonoBoldOblique9pt7b.h	FreeSerif9pt7b.h
13. FreeMonoOblique12pt7b.h	FreeSerifBold12pt7b.h
14. FreeMonoOblique18pt7b.h	FreeSerifBold18pt7b.h
15. FreeMonoOblique24pt7b.h	FreeSerifBold24pt7b.h
16. FreeMonoOblique9pt7b.h	FreeSerifBold9pt7b.h
17. FreeSans12pt7b.h	FreeSerifBoldItalic12pt7b.h
18. FreeSans18pt7b.h	FreeSerifBoldItalic18pt7b.h
19. FreeSans24pt7b.h	FreeSerifBoldItalic24pt7b.h
20. FreeSans9pt7b.h	FreeSerifBoldItalic9pt7b.h
21. FreeSansBold12pt7b.h	FreeSerifItalic12pt7b.h
22. FreeSansBold18pt7b.h	FreeSerifItalic18pt7b.h
23. FreeSansBold24pt7b.h	FreeSerifItalic24pt7b.h
24. FreeSansBold9pt7b.h	FreeSerifItalic9pt7b.h

כל שם קובץ מתחיל עם השם Free Mono או FreeSerif וכו' ובהמשכו הסגנון (נטוי מודגש וכו') , גודל הגופן בנקודות והמספר 7b האומר שיש בו 7 ביטים של תו (לפי קוד אסקי מ התו רווח ועד התו ~). ייתכן שסמלים ושות נוספות יבואו בעתיד.

יב. שימוש בגופני GFX בתכניות של ארדואינו

אחרי הכללה #include של Adafruit_GFX יש לעשות הכללה לקבצי הגופנים שנרצה. לדוגמה:

1. #include <Adafruit_GFX.h> // ספריית הליבה
2. #include <Adafruit_TFTLCD.h> // ספריית החומרה הספציפית
3. #include <Fonts/FreeMonoBoldOblique12pt7b.h> // הגופנים הרצויים
4. #include <Fonts/FreeSerif9pt7b.h>

כל גופן לוקח נפח במרחב התוכנית. גופנים גדולים תופסים נפח גדול יותר. יש לבחור את הגופנים בתשומת לב כי במיקרו בקר ATmega328 שנמצא בחלק מרטיסי הארדואינו יש רק 32 קילו בייתים של זיכרון תכנית והזיכרון "ייגמר" כשנכתוב תכנית עם פונטים גדולים. הקומפיילר לא יתרגם את התכנית מעבר לנפח של 32 קילו או במקרים גרועים יותר לא יעלה את כל התכנית לזיכרון.

בתוך קבצי הכותר המסתיימים ב .h יש מספר מבני נתונים כולל מבנה גופן אחד ראשי שיש לו בדרך כלל את אותו השם כמו של הגופן חוץ מהסיומת .h . כדי להשתמש בגופן יש להשתמש בפונקציה setFont() ולהעביר לה את כתובת המבנה . לדוגמה :

```
tft.setFont(&FreeMonoBoldOblique12pt7b);
```