

## מדריך PYGAME



בחלק זה נלמד לכתוב משחקים באמצעות מודול PYGAME של פייתון.

שלבי הלימוד:

- שימוש בפונקציות בסיסיות של PYGAME - ניצור מסך עם גרפיקה נעה, אשר מגיבה למקלדת ועכבר ומשמיעה צלילים
- שילוב OOP (תכנות מונחה עצמים), שיאפשר לנו ליצור משחקים מבוססים על שכפול של עצמים ובדיקה אם שני עצמים נוגעים זה בזה (לדוגמה משחקי יריות, משחקי כדור ומשחקי מבוכים)

תוכן עניינים:

- כתיבת שלד של PYGAME
- הוספת תמונת רקע
- הוספת צורות גאומטריות
- הזזת צורות על המסך
- הוספת דמויות Sprites
- קבלת קלט מהעכבר
- קבלת קלט מהמקלדת
- השמעת צלילים
- שילוב OOP
- בדיקת התנגשויות בין עצמים

## PYGAME שלד של

```
1 import pygame
2
3 # Constants
4 WINDOW_WIDTH = 700
5 WINDOW_HEIGHT = 500
6
7 # Init screen
8 pygame.init()
9 size = (WINDOW_WIDTH, WINDOW_HEIGHT)
10 screen = pygame.display.set_mode(size)
11 pygame.display.set_caption("Game")
12
13 pygame.quit()
```

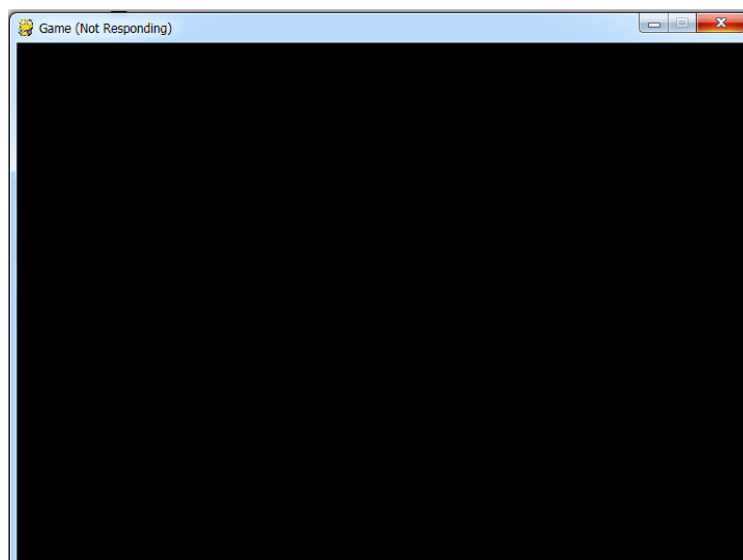
ראשית עלינו לבצע import למודול pygame.

נגדיר בתור קבועים את גודל החלון- כמות הפיקסלים לרוחב ולגובה החלון. לאחר מכן נאתחל את pygame, כך שנוכל להתחיל להשתמש בפונקציות שונות שלו.

ניצור מסך בגודל שקבענו ונקבע לו את השם "Game".

מיד לאחר מכן תתבצע הפקודה pygame.quit().

כאשר נריץ את התוכנית, יופיע לרגע קצר מסך של PYGAME ואז המסך ייסגר.



אם נרצה שהמסך יישאר, נכניס לפני שורה 13 לולאה אינסופית, אך שימו לב שכרגע – באופן זמני –הדרך היחידה לסגור את התוכנית היא באמצעות לחצן העצור של Pycharm או באמצעות מנהל המשימות. לחצן סגירת החלון שבקצה העליון של המסך עדיין אינו עובד. מיד נסדר את זה 😊

אנחנו רוצים לגרום למצב בו לחיצה על כפתור הסגירה של מסך המשחק שלנו מבצעת מה שהיא אמורה לעשות וסוגרת את המסך. לכן עלינו לתת לתוכנית שלנו הוראה מה לעשות במקרה שיש לחיצה על הכפתור:

```
13 finish = False
14 while not finish:
15     for event in pygame.event.get():
16         if event.type == pygame.QUIT:
17             finish = True
```

כל פעולה שאנחנו מבצעים מגיעה אל רשימה, ולכל פעולה כזו מוצמד שם. לדוגמה לחיצה על כפתור סגירת המסך נקראת `pygame.QUIT`. המתודה `pygame.event.get()` מספקת לנו את הרשימה של כל הפעולות שהמשתמש ביצע. נעבור על כל הפעולות שהמשתמש ביצע ונבדוק אם אחת מהן היא `pygame.QUIT`. אם כן- נצא מהלולאה האינסופית שלנו.

## שינוי רקע

הרקע השחור נחמד אבל מה אם נרצה לצבוע את המסך שלנו בצבע, לדוגמה צבע לבן.

נצטרך להגדיר את הצבע הלבן בתור tuple של 3 מספרים-

WHITE = (255, 255, 255)

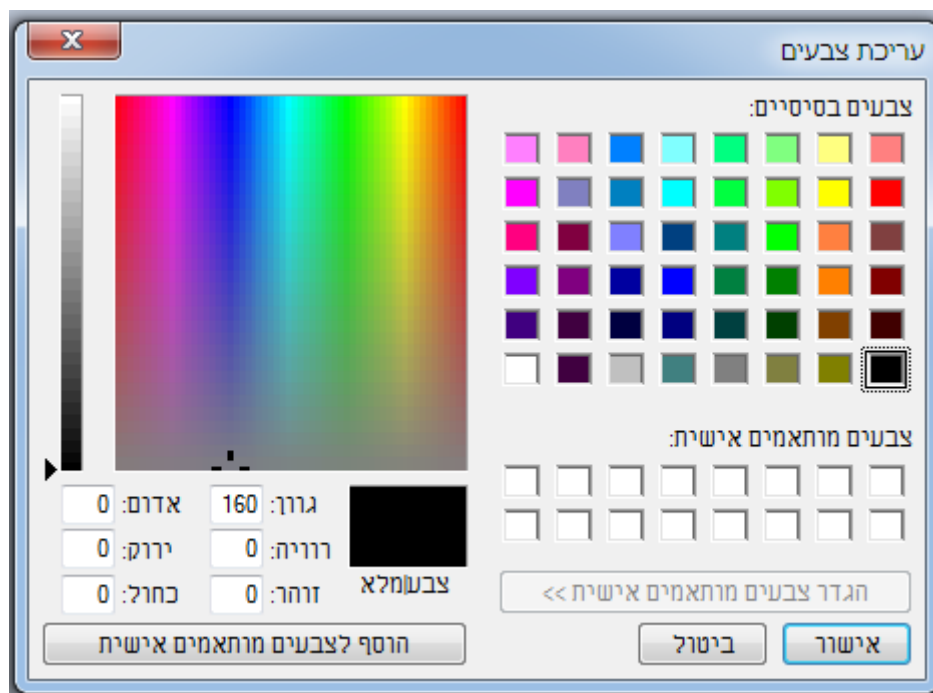
אלו ערכי RGB (Red, Green, Blue) של הצבע הלבן. לכל צבע יש שלשת מספרים שונה. לדוגמה:

RED = (255, 0, 0)

וכמובן:

BLACK = (0, 0, 0)

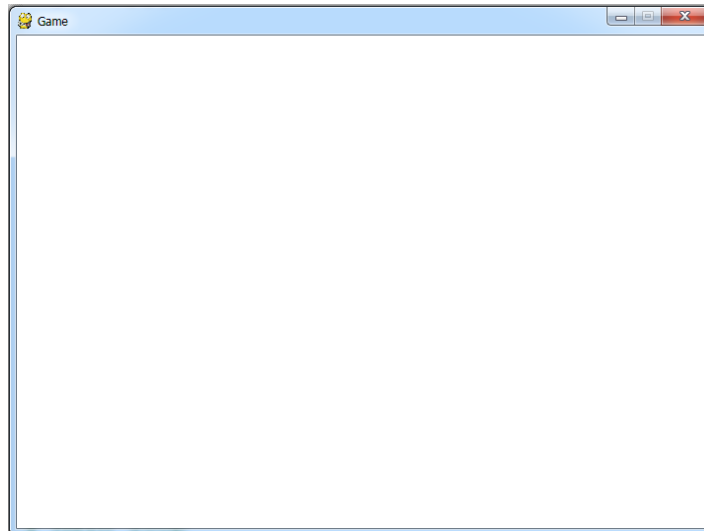
אפשר למצוא את שלשת ה-RGB של כל צבע שתרצו באמצעות תוכנת paint (צייר) הבסיסית שמגיעה עם windows, תחת תפריט "עריכת צבעים"



כעת נגדיר בתוכנית שהצבע של המסך שלנו הוא לבן. לפני הכניסה ללולאה, נוסיף את השורות 15 ו-16:

```
14 # Fill screen and show
15 screen.fill(WHITE)
16 pygame.display.flip()
17
18 finish = False
```

והתוצאה:

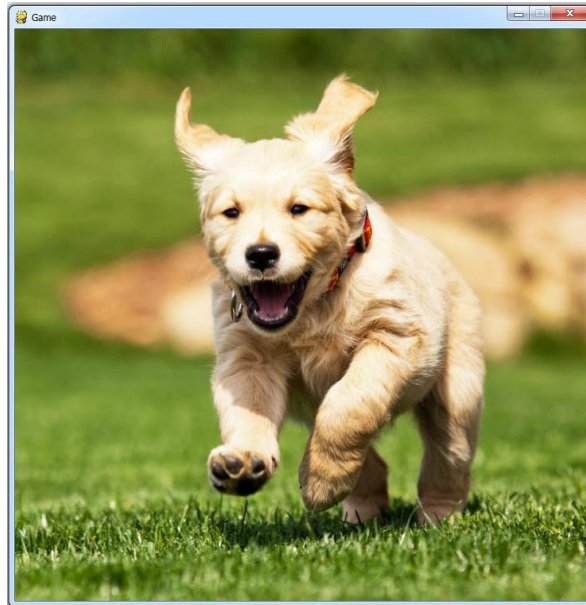


מה מבצעת שורה 16?

מסך המחשב קורא את המצב של כל פיקסל ממקום מוגדר בזיכרון המחשב (video memory). בשורה 15 אנחנו עדיין לא משנים את המצב של הזיכרון ממנו המסך מציג, אלא רק את האובייקט screen. בשורה 16 אנחנו מורים לתוכנית שלנו לבצע "flip", להחליף את המידע ב-video memory במידע ששמרנו בתוך האובייקט screen. רק ביצוע ה-flip הוא שמשנה בפועל את מצב המסך שלנו.

במילים אחרות, אנחנו יכולים לערוך את האובייקט screen כפי רצוננו, אך ללא flip כל השינויים שיבצעו יהיו נסתרים ולא יוצגו על המסך.

רקע לבן הוא משעמם קצת, בואו נעלה תמונת רקע!



ראשית נבחר תמונה כרצוננו. שימו לב לגודל התמונה בפיקסלים. אפשר למצוא את הגודל באמצעות ה-properties של הקובץ. במקרה זה הגודל הוא 720 על 720 פיקסלים.



לאחר ששינינו את גודל המסך כדי שיתאים לתמונה, נטען אותה כתמונת רקע:  
ראשית נגדיר קבוע בשם IMAGE שיכיל את שם הקובץ שלנו, לדוגמה:

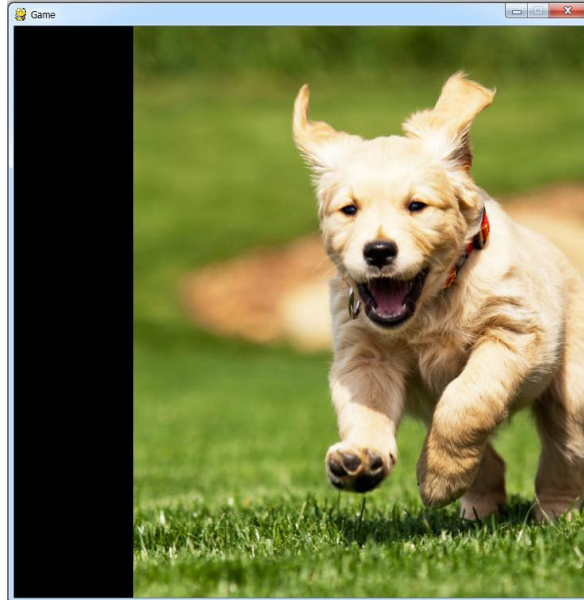
```
IMAGE = 'example.jpg'
```

וכעת נטען את התמונה כתמונת רקע:

```
15 # Fill screen and show
16 img = pygame.image.load(IMAGE)
17 screen.blit(img, (0, 0))
18 pygame.display.flip()
```

שורה 17 טוענת את התמונה לתוך האובייקט screen, החל ממיקום (0, 0). זוהי הפינה השמאלית העליונה של המסך. כיוון שדאגנו מראש שגודל המסך יהיה שווה לגודל התמונה, תמונת הרקע תופסת כעת את כל גודל המסך.

שימו לב מה היה קורה אילו היינו מנסים לטעון את התמונה ממקום התחלתי (150, 0):



כלומר ככל שאנחנו עולים בפרמטר הראשון אנחנו זזים ימינה על המסך. ככל שאנחנו עולים בפרמטר השני אנחנו זזים למטה. אם הגדרנו מסך בגודל 720 על 720, הפיקסל הימני התחתון הוא במספר (719, 719).

## הוספת צורות

בואו נוסיף כמה צורות על הרקע שלנו. ספריית PYGAME מאפשרת לנו לצייר קווים, עיגולים, מלבנים, אליפסות וקשתות.

כדי לצייר קו נשתמש במתודה `pygame.draw.line`. מתודה זו מקבלת בתור פרמטרים:

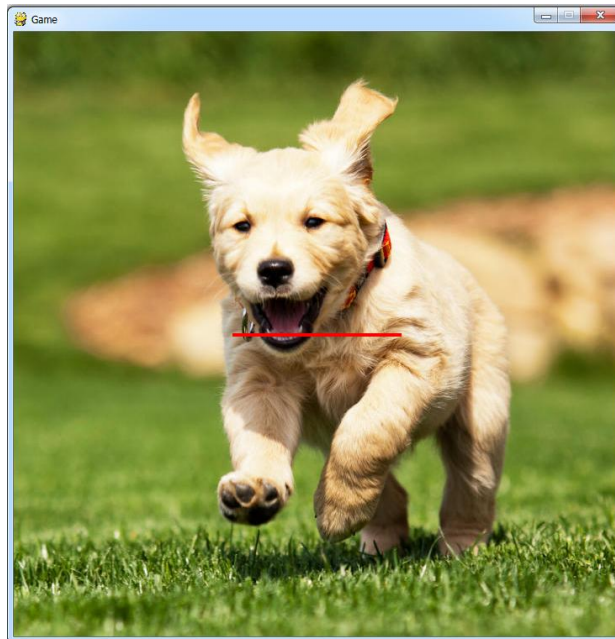
- משטח- אובייקט של `pygame` עליו יצויר הקו
- צבע הקו
- נקודת התחלה
- נקודת סיום
- עובי הקו (ברירת מחדל- 1)

כך:

```
pygame.draw.line(Surface, color, start_pos, end_pos, width=1)
```

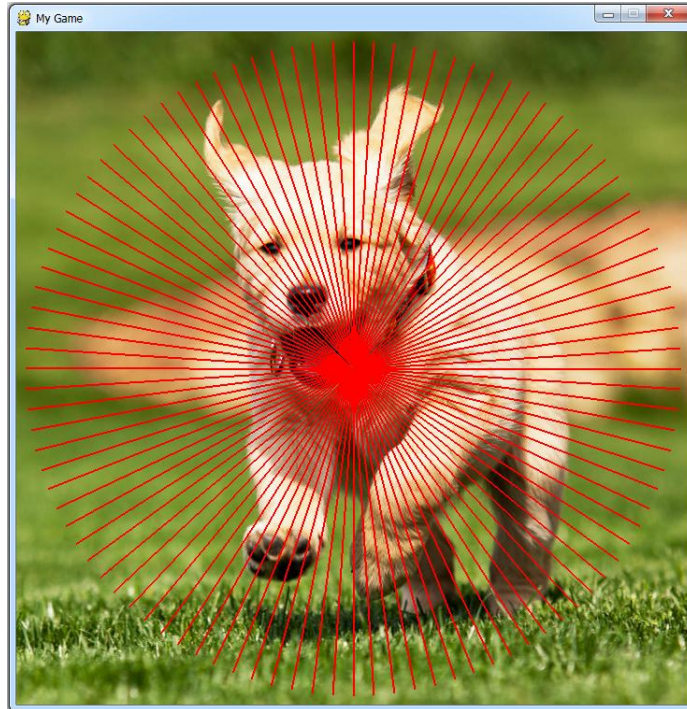
כלומר כדי לצייר קו אדום מנקודה `[260, 360]`, אל נקודה `[460, 360]` ובעובי 2:

```
19 pygame.draw.line(screen, RED, [260, 360], [460, 360], 4)
20 pygame.display.flip()
```



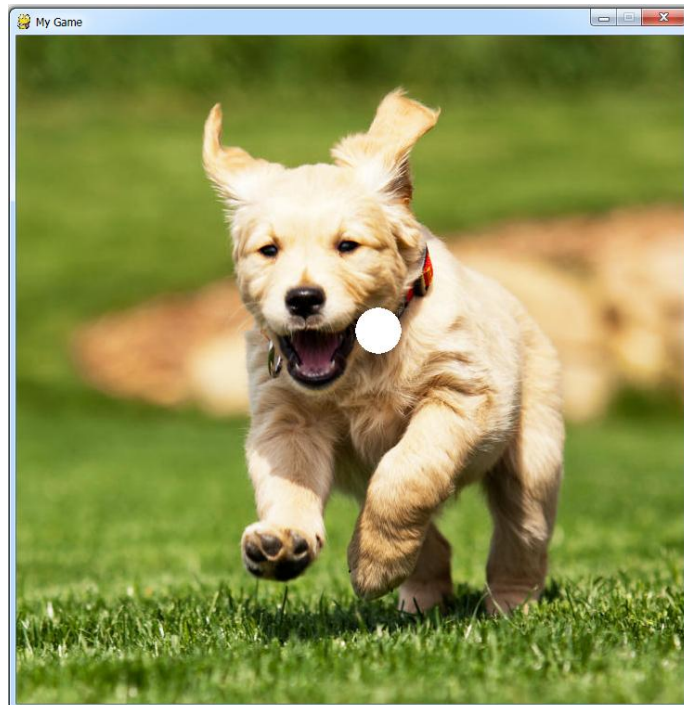
**תרגיל:** צרו בעצמכם את הדוגמה הבאה, המורכבת מ-100 קווים באורך 350 היוצאים ממרכז התמונה. טיפ: השתמשו ב-`import math` כדי לחשב את נקודת הסיום של כל קו, בעזרת הפונקציות `sin` ו-`cos`.





בקישור <https://www.pygame.org/docs/ref/draw.html> תוכלו למצוא הדרכה לציור של צורות נוספות.

**תרגיל:** ציירו עיגול במקום כלשהו על המסך



## תזוזה של גרפיקה

עד עכשיו ציירנו על המסך צורות שונות, מיד נלמד איך להזיז אותן.

מה הרעיון מאחרי צורה שזזה על המסך?

מציירים צורה במקום מסויים. לאחר זמן מה מוחקים אותה ומציירים אותה שוב במקום אחר, ליד המקום הקודם. כך נדמה לצופה שהצורה "זזה".

נתייחס לשלבים השונים:

- מחיקת צורה וציור שלה מחדש: כל מה שעלינו לעשות כדי למחוק צורה הוא פשוט לצייר את תמונת הרקע מעליה. ולצייר צורה מעל תמונת הרקע כבר למדנו.
- המתנה של פרק זמן מוגדר בין המחיקה לציור: על מנת לעשות זאת נזדקק לשעון, טיימר, שיתזמן את פרק הזמן בו יש לחזור על פעולות המחיקה והציור מחדש.

כדי להוסיף שעון, נגדיר:

```
clock = pygame.time.Clock()
```

נוסיף לקבועים שלנו את:

```
REFRESH_RATE = 60
```

כלומר, המסך מתעדכן 60 פעם בשניה.

נבנה מחדש את הלולאה המרכזית של התוכנית:

```
25 finish = False
26 while not finish:
27     for event in pygame.event.get():
28         if event.type == pygame.QUIT:
29             finish = True
30
31     screen.blit(img, (0, 0))
32     pygame.display.flip()
33     clock.tick(REFRESH_RATE)
```

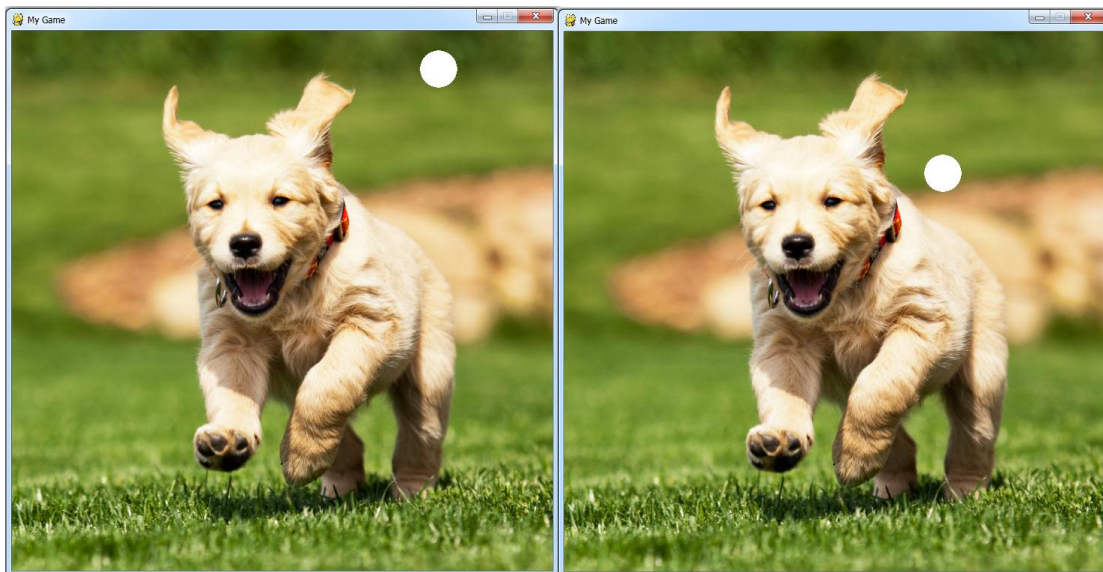
הכנסנו את שורה 31 לתוך הלולאה מכיוון שכעת אנחנו צריכים לצייר את הרקע בכל פעם מחדש (כדי למחוק את הצורה שאנחנו מעוניינים להזיז).

המקום שורה 34 צריך לבוא קטע קוד כלשהו שמזיז את הצורה שלנו. לדוגמה, אפשר להזיז מעט את העיגול בתרגיל הקודם.

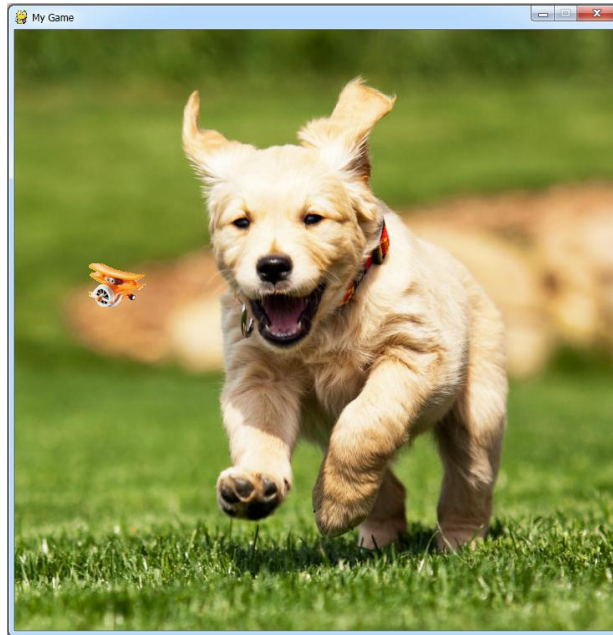
שורה 36 גורמת כפי שראינו לעדכון זיכרון הוידאו של המסך.

שורה 37 מגדירה לשעון שלנו לקצוב פרק זמן לפני הפעם הבאה שלולאת ה-while תרוץ. כאמור בלי קציבת פרק זמן זה, לא נוכל לשלוט במהירות ההתקדמות של הגרפיקה שלנו על המסך.

**תרגיל:** שכללו את התרגיל בו ציירתם עיגול על המסך, כך שהעיגול יתקדם על המסך מצד לצד. במידה והעיגול נוגע בשולי המסך, שנו את וקטור המהירות שלו כאילו שהוא כדור שניתז ממשטח.



## צור Sprite



סיימנו להשתמש עם צורות גאומטריות. כעת נניח על תמונת הרקע שלנו תמונת שחקן קטנה, Sprite, ונזיז אותה ממקום למקום.

בתור תמונת שחקן נבחר קובץ תמונה קטן:



שימו לב שהמטוס נמצא על רקע אחיד. החשיבות של זה מיד תתברר.

כדי ליצור תמונה עם רקע אחיד נפתח את תוכנת הצייר, paint, ונעתיק לתוכה כל תמונה שנרצה. כעת נגדיר צבע רקע באמצעות "עריכת צבעים" אותה הכרנו כבר. חשוב להגדיר צבע שאינו נמצא בתמונה שבחרנו. במקרה זה בחרנו את הצבע הורוד שה- $RGB$  שלו הוא  $(255, 20, 147)$ .

לאחר שסיימנו מומלץ לשמור את התמונה בפורמט  $png$ . זהו פורמט שאינו מעוות את הצבעים בתמונה וכך הצבע שבחרנו בתור רקע יישמר כמו שהוא.

בשלב הבא נטען את ה- $Sprite$  שלנו לתוכנית:

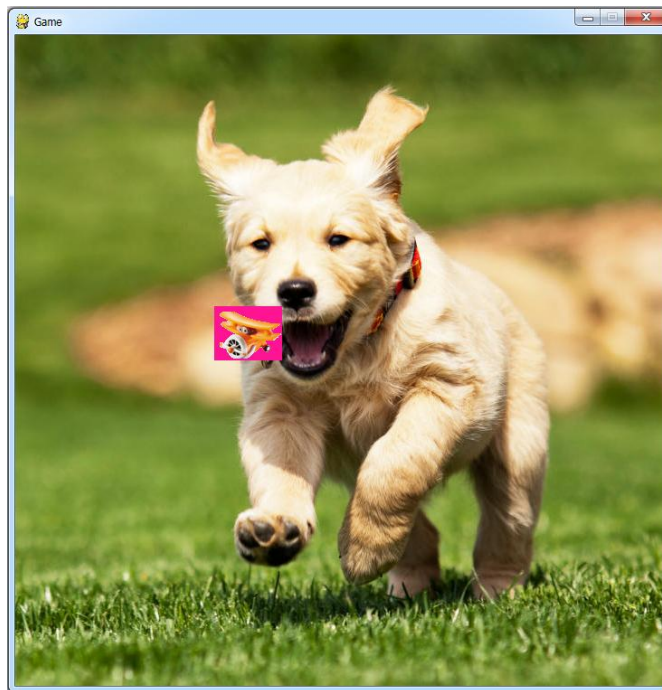
```

17 img = pygame.image.load(IMAGE)
18 screen.blit(img, (0, 0))
19 player_image = pygame.image.load('plane.png').convert()
20 screen.blit(player_image, [220, 300])
21 pygame.display.flip()

```

השורות שנוספו הינן 19 ו-20. בשורה 19 אנחנו טוענים את קובץ התמונה שלנו, ובשורה 20 אנחנו קובעים את המיקום שלו על תמונת הרקע.

אם נריץ את התוכנית כעת, נקבל את התוצאה הבאה:



כאן נכנס לשימוש הרקע שהגדרנו ל-Sprite שלנו.

בתחילת התוכנית נגדיר כקבוע את הצבע שאיתו צבענו את הרקע ה-Sprite שלנו:

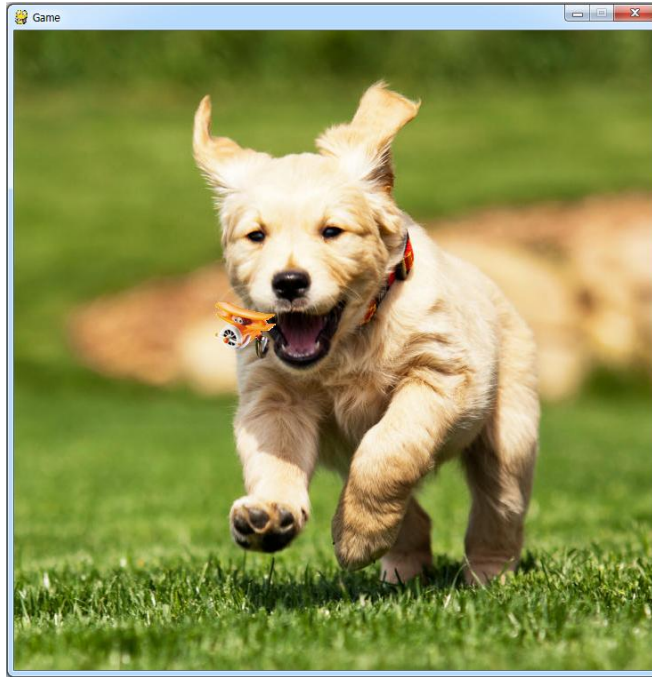
```
PINK = (255, 20, 147)
```

נגדיר לתוכנית שלנו להתעלם מהצבע הזה. כלומר, אם היא מזהה ב-Sprite שלנו פיקסל בצבע הזה, היא תתייחס אליו כשקוף ולא תצבע איתו מעל תמונת הרקע (שימו לב לשורה 21):

```

20 player_image = pygame.image.load('plane.png').convert()
21 player_image.set_colorkey(PINK)
22 screen.blit(player_image, [220, 300])

```



נפלא! כעת אנחנו יכולים להעלות כל תמונה מעל תמונת הרקע שלנו ולהזיז אותה.

## קבלת קלט מהעכבר

על מנת לקבל קלט מהעכבר, אנחנו צריכים לדעת שני דברים:

- מה מיקום העכבר על המסך
- אילו כפתורים לחוצים

לשמחתנו PYGAME מספק לנו את המידע הזה די בקלות.

באמצעות המתודה `pygame.mouse.get_pos()` אנחנו מקבלים את המיקום הנוכחי של העכבר. כל מה שנותר לנו לעשות הוא להודיע למסך שלנו לצייר את ה-Sprite במיקום של העכבר ואז לבצע `flip` למסך.

```
34 while not finish:
35     for event in pygame.event.get():
36         if event.type == pygame.QUIT:
37             finish = True
38
39     screen.blit(img, (0, 0))
40     mouse_point = pygame.mouse.get_pos()
41     screen.blit(player_image, mouse_point)
42
43     pygame.display.flip()
44     clock.tick(REFRESH_RATE)
```

כפי ששמתם לב, האייקון של העכבר מופיע על ה-Sprite שלנו וזה לא נראה מושלם. לכן בתחילת התוכנית נריץ את שורת הקוד הבאה, שתעלים את האייקון של העכבר:

```
pygame.mouse.set_visible(False)
```

כעת נרצה לבצע פעולות שונות על פי לחיצות המשתמש על העכבר.

ראשית נגדיר את לחצני העכבר:

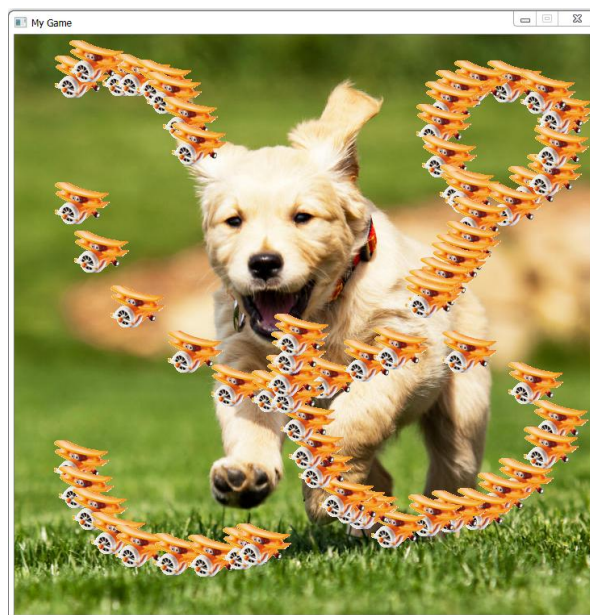
```
17 LEFT = 1
18 SCROLL = 2
19 RIGHT = 3
```

בכל פעם שהמשתמש יקליק על הכפתור השמאלי של העכבר, מיקום העכבר יישמר ברשימה. נוכל להשתמש ברשימת המיקום של העכבר בשביל לעשות כל דבר שנרצה.

```
34 while not finish:
35     for event in pygame.event.get():
36         if event.type == pygame.QUIT:
37             finish = True
38         elif event.type == pygame.MOUSEBUTTONDOWN \
39              and event.button == LEFT:
40             mouse_pos_list.append(pygame.mouse.get_pos())
```

התנאי שהגדרנו מתחיל ב-`event.type`, או בודקים האם מדובר בחיצה על כפתור כלשהו בעכבר. שימו לב שאין צורך להגדיר את הקבוע `MOUSEBUTTONDOWN`, משום שהקבוע הזה מופיע כבר בתוך ה-`event` המוגדרים של `PYGAME`. לאחר שוידאנו שהתרחשה לחיצה על כפתור כלשהו בעכבר, או בודקים איזה כפתור בדיוק נלחץ.

**תרגיל:** כיתבו תוכנית אשר בכל פעם שהעכבר נלחץ בקליק שמאלי, תשאיר על המסך עותק של ה-`Sprite` שלנו. וודאו שניתן להקליק על העכבר ללא הגבלה.





## קבלת קלט מהמקלדת

קבלת קלט מהמקלדת דומה למדי לקבלת קלט מהעכבר. גם כאן אנו מחפשים event ימים בתוך `pygame.event.get()`, אלא שהפעם נחפש ארועים שקשורים למקלדת.

ראשית נבדוק שהיתה לחיצה על המקלדת, וזאת באמצעות ה-`event` שנקרא-  
KEYDOWN

```
if event.type == pygame.KEYDOWN:
```

לאחר מכן נוכל לבדוק אם מקש ספציפי הוקש. כל המקשים במקלדת מתחילים ב-`K_` ואחריו יש את המקש. לדוגמה מקש ה-`a` הוא `K_a`. נוכל לבדוק אם מקש ה-`a` הוקש על ידי:

```
if event.key == pygame.K_a:
```

```
45 while not finish:
46     for event in pygame.event.get():
47         if event.type == pygame.QUIT:
48             finish = True
49         # User pressed a key
50     elif event.type == pygame.KEYDOWN:
51         if event.key == pygame.K_a: # key is 'a'
```

## השמעת צלילים

ראשית ניצור קובץ שמכיל את הצליל שאנחנו רוצים להשמיע.

רוב הצלילים ניתנים למציאה ב-youtube (אפשר לחפש לדוגמה laser beam sound effect).

לאחר מכן נוריד את קובץ ה-mp3 באמצעות אתר כגון:

<http://www.youtube-mp3.org/>

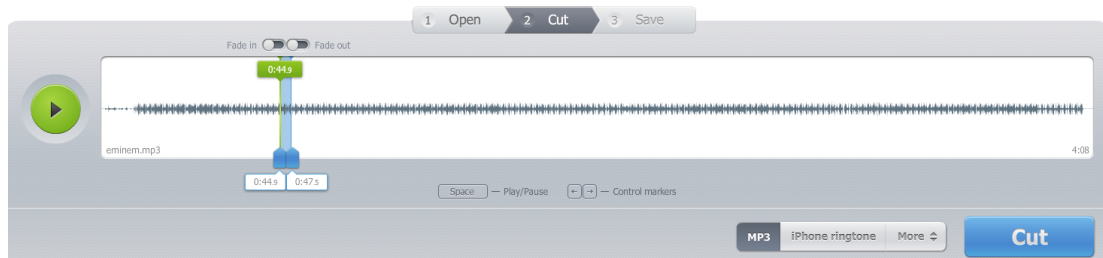
## YouTube mp3

<http://www.youtube.com/watch?v=KMU0tzLwhbE>

Convert Video

כדי לקחת רק קטע קטן מקובץ ה-mp3 שהורדנו נשתמש באתר כגון:

<http://mp3cut.net/>



כעת יש ברשותנו קובץ mp3 עם הצליל המבוקש.

העלאה והשמעה שלו מתבצעות כך:

```
29 SOUND_FILE = "kaboom.mp3"  
30 pygame.mixer.init()  
31 pygame.mixer.music.load(SOUND_FILE)  
32 pygame.mixer.music.play()
```

## תרגיל סיכום ביניים

צרו סקריפט Pygame אשר מבצע את הדברים הבאים:

- העלאה של תמונת רקע
- הזזה של Sprite כלשהו על תמונת הרקע
  - אפשר יהיה להזיז את ה-Sprite ע"י העכבר
  - אפשר יהיה להזיז את ה-Sprite ע"י המקלדת
- לחיצה על קליק שמאלי בעכבר תותיר את הסימן של ה Sprite במיקום שלו על המסך
- לחיצה על מקש ה-space תמחק את כל ה-Sprites שצוירו על הרקע
- לחיצה על קליק עכבר ימני תשמיע אפקט קולי כלשהו

תהנו!

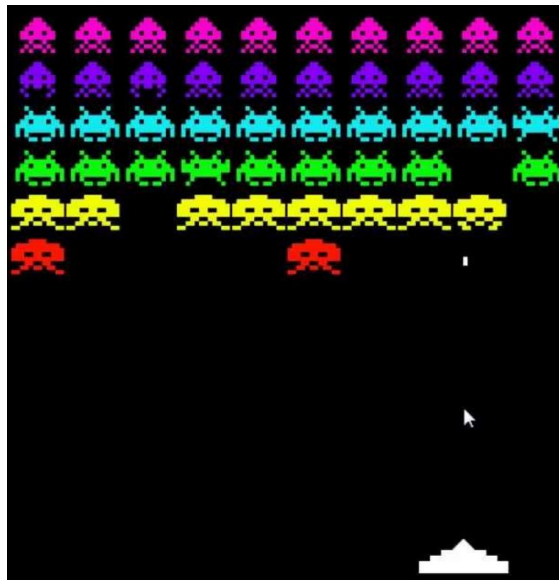
## PYGAME מתקדם - שילוב OOP

### מבוא

לאחר שלמדנו לכתוב OOP בפייתון, נשלב את הידע שלמדנו ב-pygame עם הידע שלנו ב-OOP ליצירת משחקים יפים.

ראשית, מדוע אנחנו זקוקים ל-OOP בשביל לכתוב משחקים בפייתון? הלא אנחנו יודעים כבר להעלות למסך כל צורה או Sprite שאנחנו רוצים ולהזיז אותם?

נכון, אבל הבעיה היא שהשיטה שהשתמשנו בה עד כה טובה בשביל להזיז צורה או שתיים. דמיינו שהמסך שלנו מלא ב-Sprites של Space Invaders שצריך להזיז אותם... נצטרך להגדיר כמה עשרות Sprite, לכל אחד מהם לטעון תמונה, לקבוע מיקום ולנהל את התזוזה שלהם על המסך. אחת הבעיות הגדולות שלנו תהיה לתת שמות למשתנים. נניח שנרצה להגדיר מיקום על המסך לשני space invaders. נצטרך משתנה שישמור את המיקום x של invader1, משתנה למיקום y של invader1, משתנה למיקום x של invader2 ועוד אחד למיקום y של invader2. וזה עוד לפני שהתחלנו לדבר על משתנים למהירויות שלהם (לא תמיד הם נעים באותה מהירות). לא נעים במיוחד לתכנת משחק כזה 😊



בנקודה זו מגיע לעזרתנו ה-OOP. הרעיון הכללי- נגדיר מחלקה שכוללת את כל התכונות של האובייקט שאנחנו צריכים, לדוגמה איך נראה space invader, מה המיקום שלו על המסך ומה המהירות שלו, וכל פעם שנרצה להוסיף invader פשוט ניצור instance חדש של המחלקה שלנו. אז הבה נתחיל.

## הגדרת class

בואו ניתן לכלבלב שלנו כמה כדורים לשחק איתם!

נפתח קובץ חדש, נקרא לו לדוגמה shapes. מאוחר יותר נעשה לו import לתוכנית הראשית שלנו.

נגדיר בתוכו class בשם Ball. חשוב מאד ש-Ball יירש מ-Sprite, כך שנוכל לרשת את כל המתודות המועילות של Sprite, מתודות אשר ישמשו אותנו בהמשך. לא לשכוח לעשות import pygame בתחילת הקובץ.

```
6 class Ball(pygame.sprite.Sprite):
```

קעת נכתוב את ה-constructor שלנו.

```
8 def __init__(self, x, y):
9     super(self.__class__, self).__init__()
10    self.image = pygame.image.load('baseball.png').convert()
11    self.image.set_colorkey(PINK)
12    self.rect = self.image.get_rect()
13    self.rect.x = x
14    self.rect.y = y
15    self.__vx = 3
16    self.__vy = 5
```

בשורה 8 נגדיר שה-Sprite שלנו מקבל מיקום התחלתי בזמן היצירה שלו. נשתמש בו מאוחר יותר לטובת הציור על המסך.

הדבר הראשון שאנחנו צריכים לבצע הוא הריץ את פונקציית האתחול של המחלקה ממנה Ball יורש. כתבנו את האתחול בצורה הכי רחבה שלו- נתנו לפייתון לבדוק בשבילנו מה שם המחלקה באמצעות self.\_\_class\_\_.

שורות 10 ו-11 מוכרות לנו כבר, טעינת הציור.

בשורה 12 מוגדר משתנה בשם rect ששייך לאובייקט שלנו. משתנה זה שומר את המיקום של הכדור שלנו על המסך, והוא מאד חשוב בשביל לדעת מה עושה הכדור שלנו. לדוגמה, באילו אובייקטים אחרים הוא מתנגש. שימו לב לכך שלמרות שאנחנו מציירים על המסך עיגול, הרי ש-rect הוא מרובע. זיכרו שבעיני התוכנית התמונה שלנו איננה עיגול, אלא מרובע שביקשנו להציג צבע אחד ממנו בתור "שקוף". לתוכנית אין דרך לדעת שאחרי שהצגנו את הצבע הורוד בתור שקוף, נותרה צורה של עיגול. הקואורדינטות של rect הן הפינה השמאלית העליונה של הריבוע שלנו, והגודל של rect הוא הגודל של הריבוע.

בשורות 13 ו-14 נקבע את המיקום של הכדור שלנו בתמונה, באמצעות הקואורדינטות שמחזיק rect.

נותר לנו רק לקבוע את מהירות הכדור. שימו לב לכך שהמשתנים של המהירות הם מוסתרים- מתחילים ב- \_\_. היינו יכולים לקבוע מהירות רנדומלית או לקבל את המהירות כפרמטר, משאירים זאת לכם.

### הוספת מתודות accessors ו- mutators שימושיות

בואו נזיז את הכדור שלנו על המסך.

```
18 def update_v(self, vx, vy):
19     self.__vx = vx
20     self.__vy = vy
21
22 def update_loc(self):
23     self.rect.x += self.__vx
24     self.rect.y += self.__vy
25
26 def get_pos(self):
27     return self.rect.x, self.rect.y
28
29 def get_v(self):
30     return self.__vx, self.__vy
```

יצרנו מספר מתודות שתפקידן לאפשר למצוא את מיקום הכדור בכל פעם שנרצה. מיקום הכדור החדש יהיה מיקום הכדור הקודם ועוד מהירות הכדור בכל אחד מהצירים.

אם הכדור שלנו יתנגש בקירות המסך נרצה שהוא יקפוץ חזרה, לכן נצטרך אפשרות לעדכן את המהירות שלו באמצעות מתודה מתאימה. הרי לא נרצה שהתוכנית הראשית תצטרך לעבוד ישירות מול המשתנים המוסתרים של הכדור.

זהו, התבנית של הכדור שלנו מוכנה וכעת נוכל למסור לכלבלב שלנו כמה כדורים שנרצה.

## הגדרת אובייקטים בתוכנית הראשית

בתוכנית הראשית נצטרך לעשות import ל-shapes שיצרנו. כדי שלא נצטרך לכתוב shapes כל פעם, נעשה import באופן הבא:

```
3 from shapes import Ball
```

נגדיר שני כדורים ונצייר אותם על המסך:

```
23 ball1 = Ball(100, 100)
24 ball2 = Ball(200, 200)
25 screen.blit(ball1.image, ball1.get_pos())
26 screen.blit(ball2.image, ball2.get_pos())
27
28 finish = False
29 while not finish:
30     for event in pygame.event.get():
31         if event.type == pygame.QUIT:
32             finish = True
33     clock.tick(REFRESH_RATE)
34     pygame.display.flip()
```

בשורות 23 ו-24 אנחנו יוצרים שני כדורים, לכל אחד מהם יש מיקום התחלתי שאנחנו מעבירים ל-`constructor`. מיקום התחלתי זה יועתק שם ל-`rect`.

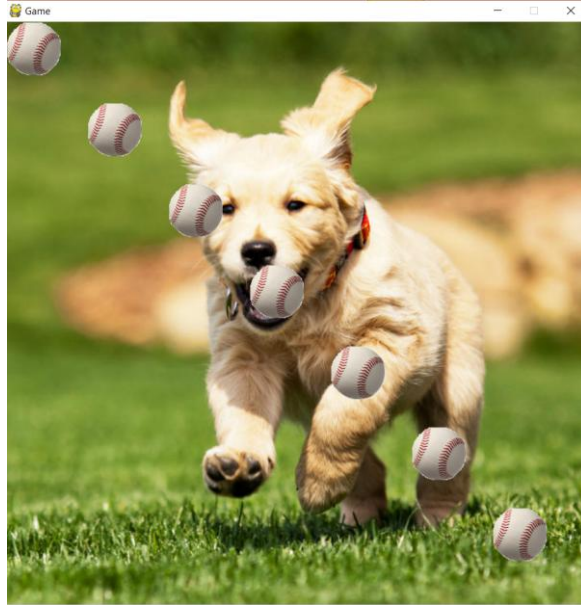
בשורות 25 ו-26 אנחנו מעבירים למסך שלנו את הכדורים. המתודה `blit` מקבלת כזכור את התמונה שאנחנו רוצים להעלות ואת המיקום. התמונה היא `ball.image` ואל המיקום אנחנו ניגשים בעזרת המתודה `get_pos` שיצרנו לשם כך.

שורות 28 והלאה הן הקוד הבסיסי המוכר לנו. נריץ ונקבל כדורים במקומות המתאימים על המסך:



## Sprite.Group()

נהדר, אלא שמה שעשינו כרגע קצת מסורבל. דמיינו שאנחנו רוצים לצייר לא שני כדורים אלא שבעה, כמו בדוגמה הבאה:



האם הגיוני שבשביל להגדיר כמות גדולה של כדורים נצטרך לכתוב שוב ושוב שורות כמו 23 ו-24? לא. האם אין דרך יותר פשוטה להציג את כל הכדורים שלנו, ולא לכתוב שורת קוד נפרדת לכל כדור? ... בוודאי שיש.

```
23 balls_list = pygame.sprite.Group()
24 for i in xrange(7):
25     ball = Ball(i*100, i*100)
26     balls_list.add(ball)
27
28 balls_list.draw(screen)
```

כדי לעבור על כל הכדורים בבת אחת אנחנו צריכים לולאת for, ולולאת for צריכה לעבור על מבנה נתונים שהוא iterable- לדוגמה רשימה. לכן הכל מתחיל בהגדרה של רשימה של כדורים, בשורה 23 אנו מגדירים רשימה ריקה, sprite.Group(). אבל למה אנחנו מגדירים sprite.Group() ולא סתם רשימה ריקה, [ ]? בגלל שלרשימה מסוג sprite.Group() יש כל מיני מתודות שימושיות, שחוסכות לנו עבודה. בקרוב נכיר שתיים מהן:

- מתודה שמדפיסה למסך בבת אחת את כל האובייקטים שברשימה



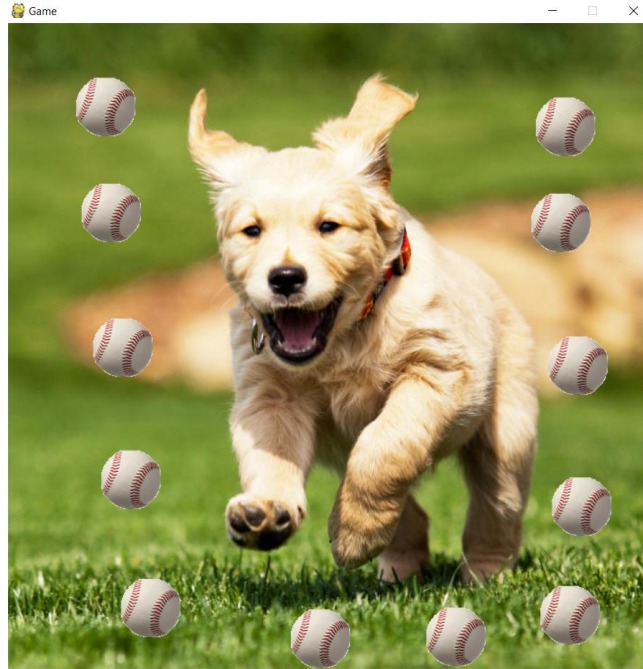
- מתודה שבודקת אם שני יש התנגשויות בין אובייקטים (נמצאים באותו מקום)

בכל איטרציה של הלולאה אנחנו מגדירים כדור חדש, במיקום התחלתי כלשהו, ולאחר מכן מוסיפים אותו לרשימה שלנו באמצעות מתודת add. נפלא, יש לנו כעת רשימה הכוללת 7 כדורים. שימו לב לכך שהעובדה שאנחנו משתמשים בכל איטרציה באותו משתנה בשם ball אינה גורמת למחיקת הכדורים הקודמים: כל משתנה בשם ball הוא מצביע על אובייקט מסוג Ball. בכל פעם שהתוכנית מגיעה לשורה 25, נוצר אובייקט חדש מסוג Ball והמשתנה ball מצביע עליו. בשורה 26 המצביע על האובייקט שנוצר נשמר ברשימה, ולכן אנחנו מסוגלים לגשת אליו גם אחרי ש-ball כבר מצביע על האובייקט הבא.

בשורה 28 אנחנו קוראים למתודה שחוסכת לנו המון עבודה: draw. המתודה הזו פועלת על (`sprite.Group()`) ומאפשרת להעביר למסך בבת אחת את כל האובייקטים שיש ברשימה. זה בדיוק כמו לרוץ בלולאת `for` ולהדפיס כל אובייקט למסך באופן נפרד.

## יצירת אובייקטים חדשים

כעת נרצה שכמות האובייקטים שלנו תהיה ניתנת לקביעה על ידי המשתמש. לדוגמה- שכל הקלקה על לחצן שמאלי בעכבר תיצור כדור חדש על המסך.



ניזכר בקוד שבודק אם היתה הקלקה על הלחצן השמאלי של העכבר – שורה 30 (שממשיכה לתוך שורה 31), ומה היה מיקום העכבר- שורה 32:

```
26 while not finish:
27     for event in pygame.event.get():
28         if event.type == pygame.QUIT:
29             finish = True
30         elif event.type == pygame.MOUSEBUTTONDOWN \
31              and event.button == LEFT:
32             x, y = pygame.mouse.get_pos()
33             ball = Ball(x, y)
34             balls_list.add(ball)
35
36     screen.blit(img, (0, 0))
37     balls_list.draw(screen)
38     pygame.display.flip()
39     clock.tick(REFRESH_RATE)
```

לאחר שמצאנו את מיקום העכבר אנחנו מגדירים ball חדש במיקום זה ומוסיפים אותו לרשימה. את יתר הפקודות כבר הכרנו לפני: הדפסת הרקע (36), הדפסת כל הכדורים (37), עדכון המסך (38) וקציבת זמן עדכון התוכנית (39).

## הזזת האובייקטים

אובייקטים זזים יותר מעניינים מאובייקטים שקבועים במקום. בואו ניתן לכל אובייקט מהירות התחלתית אקראית.

```
30 elif event.type == pygame.MOUSEBUTTONDOWN \
31     and event.button == LEFT:
32     x, y = pygame.mouse.get_pos()
33     ball = Ball(x, y)
34     vx = random.randint(-3, 3)
35     vy = random.randint(-3, 3)
36     ball.update_v(vx, vy)
37     balls_list.add(ball)
```

בשורות 34-36 אנחנו מגרילים מהירות התחלתית ומעדכנים את מהירות הכדור בהתאם. כרגע בכל פעם שנריץ את המתודה `ball.update_loc()` יעודכנו ערכי ה-`rect.x` וה-`rect.y` של הכדור, כפי שקבענו בהתחלה:

```
39 for ball in balls_list:
40     ball.update_loc()
```

זהו, הכדורים שלנו נעים על המסך ☺

...הבעיה היא, שגם כאשר הם מגיעים לקצה המסך הם ממשיכים לנוע ויוצאים ממנו. לכן נוסיף את הקוד הבא, שבודק אם הכדור שלנו נוגע בשולי המסך ואם כן- מעדכן את המהירות שלו כך שהכדור "קופץ" חזרה:

```
39 for ball in balls_list:
40     ball.update_loc()
41     x, y = ball.get_pos()
42     vx, vy = ball.get_v()
43     if x + BALL_SIZE > WIDTH or x < 0:
44         vx *= -1
45     if y + BALL_SIZE > HEIGHT or y < 0:
46         vy *= -1
47     ball.update_v(vx, vy)
```

את שורות הקוד הללו, שבודקות אם הכדור נוגע בשוליים ומעדכנות את המהירות, מומלץ להכניס למתודה של הכדור. לדוגמה `bounce()` שבודקת אם הכדור יוצא מהמסך ואם כן- מקפיצה אותו חזרה. נשאר זאת כתרגיל לכם.

## בדיקת התנגשויות

ברוב משחקי המחשב נרצה לדעת אם שני אובייקטים עולים זה על זה. לדוגמה, אם יריה נוגעת בדמות. או אם הפקמן שלנו נתפס על ידי שדון.

נרצה לשדרג את התוכנית שלנו כך שכל שני כדורים שמתנגשים זה בזה – יימחקו. כעת נראה איך אפשר לזהות בקלות התנגשויות של אובייקטים.

המתודה `spritecollide` משמשת למטרה זו. המתודה מקבלת אובייקט ורשימה של אובייקטים ומחזירה את כל האובייקטים מהרשימה שמתנגשים באובייקט הנבדק. בשורת הקוד הבאה אנו בודקים אם אובייקט מסויים בשם `ball` מתנגש בכדורים ששמורים ב-`balls_list`:

```
balls_hit_list = pygame.sprite.spritecollide(ball, balls_list, False)
```

כיצד המתודה בודקת את ההתנגשויות? באמצעות ה-`rect` של כל אובייקט. כלומר, שני אובייקטים מתנגשים אם ה-`rect` שלהם חופף זה את זה.

בנוסף המתודה מקבלת פרמטר בשם `doKill` ("האם להרוג"), בוליאני. אם הפרמטר הוא `True`, אז כל אובייקט ברשימת האובייקטים שמתנגש באובייקט הנבדק – יימחק.

במקרה שלנו אנחנו לא רוצים למחוק את האובייקט המתנגש. מדוע? מיד נראה.

```
49 new_balls_list.empty()
50 for ball in balls_list:
51     balls_hit_list = pygame.sprite.spritecollide \
52         (ball, balls_list, False)
53     if len(balls_hit_list) == 1:      # ball collides
54                                     # only with itself
55         new_balls_list.add(ball)
56
57 balls_list.empty()
58 for ball in new_balls_list:
59     balls_list.add(ball)
```

בשורה 50 אנחנו מגדירים שאנחנו עוברים על כל הכדורים שבתוך רשימת הכדורים שלנו. כלומר אנחנו בודקים בכמה כדורים בתוך הרשימה מתנגש כל כדור ברשימה. ברור שהתשובה תהיה תמיד לפחות 1- מכיוון שהכדור שבחרנו מהרשימה מתנגש בעצמו. וזו גם הסיבה לכך שאנחנו לא מוחקים את הכדור המתנגש מהרשימה, כי אחרת נישאר עם רשימת כדורים ריקה.

בשורה 53 אנחנו בודקים האם הכדור שלנו התנגש רק בכדור אחד ברשימה (כלומר, בעצמו בלבד). אם כן, זה אומר שהכדור צריך להשאר על המסך. לכן

נשמור אותו ברשימת הכדורים ה"שורדים" `new_balls_list`. שימו לב לכך שזוהי רשימה מסוג `sprite.Group()` ושבשורה 49 אנחנו מרוקנים אותה מכדורים, באמצעות המתודה `.empty`.

כל מה שנותר לנו לעשות הוא להחזיר את הכדורים השורדים לרשימת `balls_list` לפני שנמשיך ונציג אותם על המסך. לשם כך אנחנו מרוקנים את `balls_list` בשורה 57 ולאחר מכן אנחנו מעתיקים כל כדור מתוך רשימת השורדים אל תוך `balls_list`.

שימו לב לכך שאי אפשר לכתוב פשוט `balls_list = new_balls_list`. מה יקרה לדעתכם במקרה זה? תוכלו להעזר בדוגמה הבאה של שתי רשימות שמצביעות על אותו המקום בזיכרון.

```
>>> list1 = [1, 2, 3]
>>> list2 = list1
>>> list1.pop(0)
1
>>> list1.pop(1)
3
>>> list1 = [1, 2, 3]
>>> list2 = list1
>>> list1.pop(0)
1
>>> list1.pop(0)
2
>>> list1.pop(0)
3
>>> list2
[]
>>>
```

לסיכום הנה הקוד המלא של התוכנית שבודקת התנגשויות של כדורים:

```

1 # Pygame example program
2 # Sprite collision detection
3 # Author: Barak Gonen, 2017
4
5 import pygame
6 import random
7 from shapes import Ball
8
9 WIDTH = 720
10 HEIGHT = 720
11 LEFT = 1
12 REFRESH_RATE = 60 # times each second
13 BALL_SIZE = 55 # pixels
14
15 img = pygame.image.load('example.jpg')
16 pygame.init()
17 size = (WIDTH, HEIGHT)
18 screen = pygame.display.set_mode(size)
19 pygame.display.set_caption("Game")
20 screen.blit(img, (0, 0))
21 clock = pygame.time.Clock()
22
23 balls_list = pygame.sprite.Group()
24 new_balls_list = pygame.sprite.Group()
25 finish = False
26 while not finish:
27     for event in pygame.event.get():
28         # quit if window closed
29         if event.type == pygame.QUIT:
30             finish = True
31         # add ball each time user clicks mouse
32         elif event.type == pygame.MOUSEBUTTONDOWN \
33             and event.button == LEFT:
34             x, y = pygame.mouse.get_pos()
35             ball = Ball(x, y)
36             vx = random.randint(-3, 3)
37             vy = random.randint(-3, 3)
38             ball.update_v(vx, vy)
39             balls_list.add(ball)
40
41     # update balls locations, bounce from edges
42     for ball in balls_list:
43         ball.update_loc()
44         x, y = ball.get_pos()
45         vx, vy = ball.get_v()
46         if x + BALL_SIZE > WIDTH or x < 0:

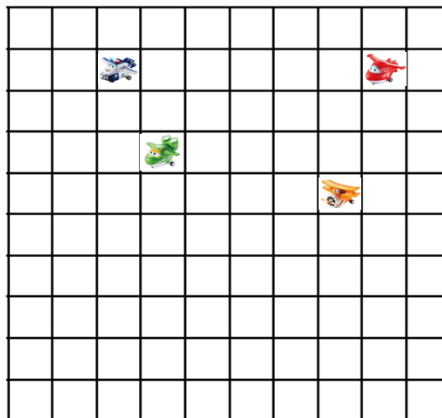
```

```
47         vx *= -1
48     if y + BALL_SIZE > HEIGHT or y < 0:
49         vy *= -1
50     ball.update_v(vx, vy)
51
52     # find which balls collide and should be removed
53     new_balls_list.empty()
54     for ball in balls_list:
55         balls_hit_list = pygame.sprite.spritecollide \
56             (ball, balls_list, False)
57         if len(balls_hit_list) == 1:      # ball collides
58                                           # only with itself
59             new_balls_list.add(ball)
60
61     balls_list.empty()
62     for ball in new_balls_list:
63         balls_list.add(ball)
64
65     # update screen with surviving balls
66     screen.blit(img, (0, 0))
67     balls_list.draw(screen)
68     pygame.display.flip()
69     clock.tick(REFRESH_RATE)
70
71 pygame.quit()
```

## תרגיל מסכם - פיקוח אווירי



אתם יושבים במגדל הפיקוח האווירי ואחראים למנוע התנגשות בין מטוסים במרחב האווירי שלכם. המרחב האווירי הוא מטריצה בגודל 10 משבצות על 10 משבצות (כל משבצת היא בגודל של כמה פיקסלים שתחליטו, לדוגמה 50 על 50 פיקסלים).



למרחב האווירי שלכם נכנסו 4 מטוסים מסוג CrazyPlane שנעים בו אקראית. בכל תור, כל אחד מהמטוסים יכול לזוז לכל משבצת שצמודה למשבצת בה הוא נמצא (כולל באלכסון).

עליכם לכתוב אלגוריתם שמנהל את תנועת המטוסים: בכל תור על האלגוריתם לבחור אם לתת למטוס להמשיך לנוע אקראית או להורות לו לאיזו משבצת צמודה הוא צריך לפנות.

ניקוד: המטרה היא שהאלגוריתם שלכם ישלח כמה שפחות הוראות למטוסים. לכן, כל מטוס שנע למשבצת אקראית מעניק לכם נקודה. כל מטוס שנע למשבצת עקב פקודה שקיבל ממכם, מוריד לכם נקודה. המשחק נגמר כאשר שני מטוסים מתנגשים או לאחר 1000 תורות.

חישוב ניקוד לדוגמה:



ארבעת המטוסים שרדו 1000 תורות, כלומר נעו ביחד 4000 משבצות. בסך הכל המטוסים קיבלו 200 פקודות שינוי מיקום, כלומר הם נעו 3800 צעדים אקראיים (3800 נקודות). סך הכל הרווחתם  $3800 - 200 = 3600$  נקודות.

הצלחתם לגרום למטוסים שלכם לשרוד? יפה מאד! עכשיו שחקו את המשחק עם 10 מטוסים ☺ האם האלגוריתם שלכם עדיין עובד היטב? האם תוכלו לשפר אותו?