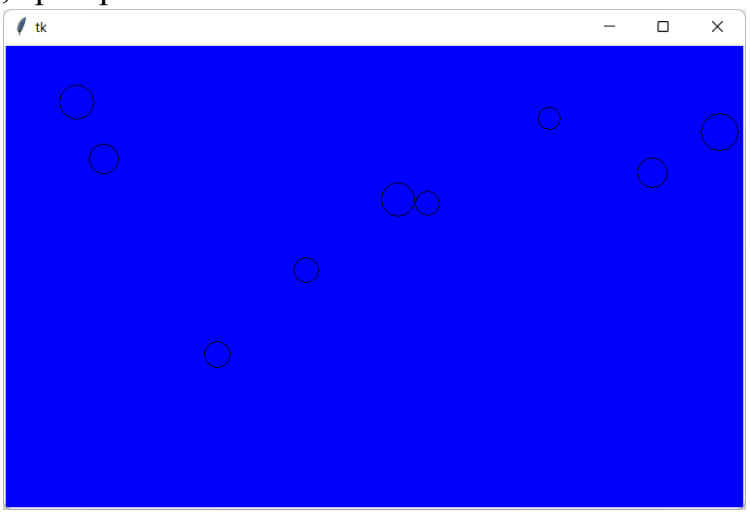


Практичне завдання

Взаємодія об'єктів

Увага! *Під час роботи з комп'ютером дотримуйтеся правил безпеки і санітарно-гігієнічних норм*
Онлайн середовище: ***https://trinket.io/library/trinkets/create?lang=pygame***

Завдання: скласти ігрову програму **Ловець бульбашок**. На ігровому полотні через певні проміжки часу з'являються бульбашки. Гравець ловить бульбашки, клацаючи їх. Якщо кількість бульбашок перевищує допустиму, гра припиняється.



Обладнання: комп'ютер зі встановленим середовищем програмування Python.

Хід роботи

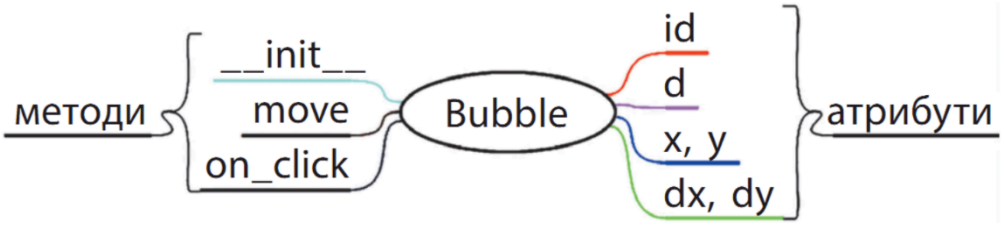
1. Завантажте модулі, що потрібні для реалізації руху об'єктів. Створіть вікно програми і додайте полотно розміром 800 x 500:

```
1  from tkinter import*
2  from random import*
3  from time import*
4  tk = Tk()
5  canvas = Canvas(tk, width = 800, height = 500, bg='blue')
6  canvas.pack()
```

2. Створіть змінні для керування режимом гри:

```
7  maxB=10     # За такої кількості бульбашок на полотні гра зупиниться
8  genT=1000   # Проміжок часу між появою бульбашок
```

Створіть клас **Bub()** – модель бульбашки (рис. 2). Екземпляр цього класу повинен:



- створюватись на полотні **canvas** вікна **tk**;
- мати випадкові координати **x** і **y** (у межах полотна);
- зразу ж після створення починати рух, зміщуючись на кожному кроці на **dx** і **dy** пікселів уздовж відповідних осей;
- відбиватись від країв полотна;
- після клацання – зникати.

Крім того, у класі мають підраховуватися кількості екземплярів-бульбашок, наявних і зловлених.

На кроках 3-5 ми створимо клас **Bubble**, екземпляри якого поводитимуться на полотні **canvas** вікна **tk** саме так, як описано.

3. Конструктор __init__:

```
 9 class Bub():
10     number = 0   # Лічильник бульбашок на полотні
11     clicked = 0  # Лічильник клацнутих бульбашок
12     def __init__(B):
15         B.x = randint(2,755)
16         B.y = randint(2,455)
17         B.dx = randint(-5, 5)
18         B.dy = randint(-5, 5)
19         Bub.number = Bub.number+1
```

```
21         B.id = canvas.create_oval(B.x, B.y, B.x+B.d, B.y+B.d, fill = 'blue')
22         canvas.tag_bind(B.id, '<1>', B.on_click) # Бульбашка має обробник
23         B.move() # клацання і починає рухатись після створення
```

4. Метод **move**, що забезпечує рух і відбивання кульки:

```
24     def move(B):
25         canvas.move(B.id, B.dx, B.dy)
26         tk.update()
27         pos = canvas.coords(B.id)
28         if len(pos)>0: # Якщо кулька ще не зникла...
29             if pos[0]<2 or pos[2]>798: # ...то відбиваємо від країв
30                 B.dx = -1*B.dx
31             if pos[1]<2 or pos[3]>498:
32                 B.dy = -1*B.dy
33             tk.after (100, B.move) # Продовження руху
```

5. Метод-обробник клацання бульбашки **on_click**:

```
def on_click(B,event):
        if Bub.number<maxB:
            canvas.delete(B.id)
            Bub.number= Bub.number-1
            Bub.clicked = Bub.clicked+1
            tk.title(Bub.clicked)
```

6. Створіть функцію **play()**, яка забезпечить створення бульбашок у потрібні моменти, а також вчасне припинення гри:

```
def play():
  if Bub.number<maxB:                    # Якщо бульбашок мало...
      Bub()                              # -то створюємо бульбашку
      tk.after(genT, play)               # ...продовжуємо створювати
  else:
      tk.after_cancel(play)              # Припиняємо створювати
      tk.title('Гру закінчено. Ваш результат:' + str(Bub.clicked))
```

7. В основній програмі запишіть команди виклику функції **play()**:

```
53  play()
54  tk.mainloop()
```

Випробуйте програму. За потреби виправте помилки. Коли програма повністю запрацює, додайте до неї можливості, описані в наступних пунктах.

8. Зараз діаметр всіх бульбашок однаковий. Зробіть так, щоб бульбашка під час створення отримувала випадкове значення діаметра в межах від 20 до 40 пікселів.

```
14          B.d = randint(20, 40)
```

9. По завершенні гри бульбашки, що залишились, продовжують рухатись, але не реагують на клацання. Зробіть так, щоб клацання в цей період спричиняло змінення кольору на жовтий ('Yellow').
10. Кожна бульбашка під час клацання до загального рахунку додає 1 бал. Зробіть так, щоб швидкі бульбашки малого розміру додавали по 2 бали.

```
34    def on_click(B,event):
35        if Bub.number<maxB:
36            canvas.delete(B.id)
37            Bub.number= Bub.number-1
38            if B.d < 30:
39                Bub.clicked = Bub.clicked+2
40            else:
41                Bub.clicked = Bub.clicked+1
42            tk.title(Bub.clicked)
43        canvas.itemconfig(B.id, fill='yellow')
```

11. Бульбашки з'являються через рівні проміжки часу, визначені змінною **genT** (початкове значення 1000 мс). Зробіть так, щоб цей інтервал поступово зменшувався.

```
44  def play():
45      global genT
46      if Bub.number<maxB:             # Якщо бульбашок мало...
47          Bub()                       # -то створюємо бульбашку
48          tk.after(genT, play)        # ...продовжуємо створювати
49          genT-=20
50      else:
51          tk.after_cancel(play) # Припиняємо створювати
52          tk.title('Гру закінчено. Ваш результат: ' + str(Bub.clicked))
```

12. Запропонуйте власну ідею покращення програми і реалізуйте її.

Зробіть висновок: чому для моделювання руху графічних об'єктів доречно використовувати методи ООП.