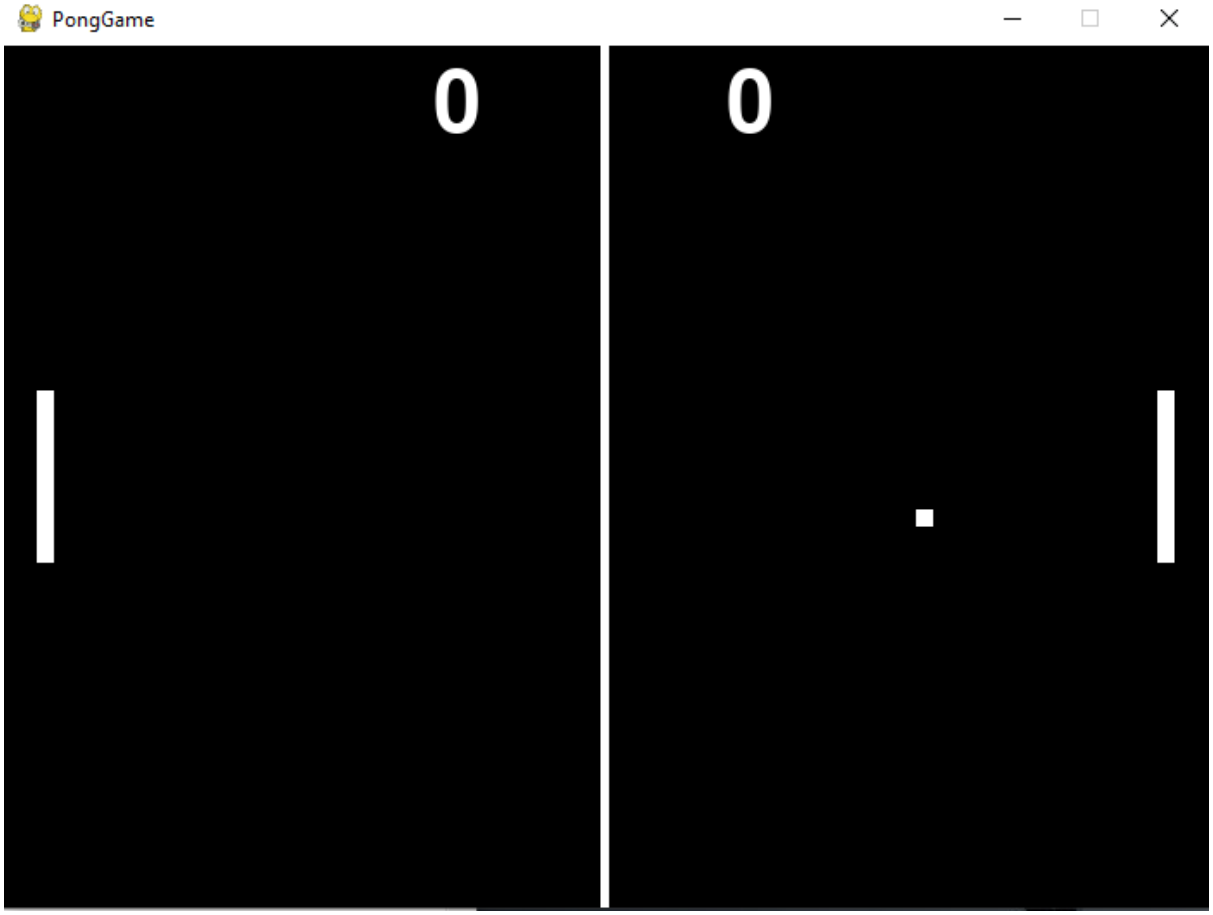
PongGame



Како играти Понг Игрицу:

Понг игтица је игрица која симулира стони тенис. Игрица је намењена за два играча. Прозор је подељен на два једнака дела. На сваком делу се налази по један „играч". Правила игре су слична као и код оригиналне игре. Десни играч постиже поен ако лоптица удари у леви зид. Леви играч постиже поен ако лоптица удати у десни зид. Ако лоптица удари у горњи или доњи зид она се одбија. Сваки играч има „рекет" којим узвраћа лоптицу и који може да се помера горе-доле.

Изворни код за Понг игрицу

main.py

```
1  import pygame
2  from paddle import Paddle
3  from ball import Ball
4  from tkinter import *
5  from tkinter import messagebox
6
7  pygame.init()
8
9  # define some colors
10 BLACK = (0,0,0)
11 WHITE = (255,255,255)
12
13 # open a new window
14 size = (700, 500)
15 screen = pygame.display.set_mode(size)
16 pygame.display.set_caption("PongGame")
17
```

```
18
19 paddleA = Paddle(WHITE, 10, 100)
20 paddleA.rect.x = 20
21 paddleA.rect.y = 200
22
23 paddleB = Paddle(WHITE, 10, 100)
24 paddleB.rect.x = 670
25 paddleB.rect.y = 200
26
27 ball = Ball(WHITE,10,10)
28 ball.rect.x = 345
29 ball.rect.y = 195
30
31 # this will be a list that will contain all the sprites we intend to use in our
game
32 all_sprites_list = pygame.sprite.Group()
33
34 # add the paddle and ball to the list of objects
35 all_sprites_list.add(paddleA)
36 all_sprites_list.add(paddleB)
37 all_sprites_list.add(ball)
38
39 # the loop will carry on until the user exit the game (clicks the close button)
40 carryOn = True
41
42 # the clock will be used to control how fast the screen updates
43 clock = pygame.time.Clock()
44
45 # initialise player scores
46 scoreA = 0
47 scoreB = 0
48
49 # -------- main program -----------
50 Tk().wm_withdraw() #to hide the main window
51 messagebox.showinfo("Play", "Press OK when you want to start, then on a black
surface when the game starts so you can play")
52 while carryOn:
53     # --- main event
54     for event in pygame.event.get(): # user did something
55         if event.type == pygame.QUIT: # if user clicked close
56               carryOn = False # flag that we are done so we exit this loop
57         elif event.type==pygame.KEYDOWN:
58                if event.key==pygame.K_x: # pressing the x key will quit the game
59                     carryOn=False
60
61     # moving the paddles when the use uses the arrow keys (player A) or "W/S" keys
(player B)
62     keys = pygame.key.get_pressed()
63     if keys[pygame.K_w]:
64         paddleA.moveUp(5)
65     if keys[pygame.K_s]:
66         paddleA.moveDown(5)
67     if keys[pygame.K_UP]:
68         paddleB.moveUp(5)
69     if keys[pygame.K_DOWN]:
70         paddleB.moveDown(5)
71
72     # --- game logic
73     all_sprites_list.update()
74
75     # check if the ball is bouncing against any of the 4 walls:
76     if ball.rect.x>=690:
77         scoreA+=1
78         ball.velocity[0] = -ball.velocity[0]
79     if ball.rect.x<=0:
80         scoreB+=1
81         ball.velocity[0] = -ball.velocity[0]
82     if ball.rect.y>490:
```

```
83          ball.velocity[1] = -ball.velocity[1]
84      if ball.rect.y<0:
85          ball.velocity[1] = -ball.velocity[1]
86
87      # detect collisions between the ball and the paddles
88                          if     pygame.sprite.collide_mask(ball,     paddleA)     or
pygame.sprite.collide_mask(ball, paddleB):
89        ball.bounce()
90
91      # --- drawing code
92      # first, clear the screen to black.
93      screen.fill(BLACK)
94      # draw the net
95      pygame.draw.line(screen, WHITE, [349, 0], [349, 500], 5)
96
97      # now let's draw all the sprites in one go (for now we only have 2 sprites)
98      all_sprites_list.draw(screen)
99
100     # display scores:
101     font = pygame.font.Font(None, 74)
102     text = font.render("A: " + str(scoreA), 1, WHITE)
103     screen.blit(text, (230,10))
104     text = font.render("B: " + str(scoreB), 1, WHITE)
105     screen.blit(text, (370,10))
106     if scoreA==30 or scoreB==30:
107         if scoreA>scoreB:
108             Tk().wm_withdraw() #to hide the main window
109             text = "Player A win\nScore: A: {}  B: {} ".format(scoreA, scoreB)
110             messagebox.showinfo("Winner", text)
111         else:
112             Tk().wm_withdraw() #to hide the main window
113             text = "Player A win\nScore: A: {}  B: {} ".format(scoreA, scoreB)
114             messagebox.showinfo("Winner", text)
115         carryOn = False
116
117     # --- go ahead and update the screen with what we've drawn
118     pygame.display.flip()
119
120     # --- limit to 60 frames per second
121     clock.tick(60)
122
123# once we have exited the main program loop we can stop the game engine:
124 pygame.quit()
125
```

paddle.py

```
1 import pygame
2 BLACK = (0,0,0)
3
4 class Paddle(pygame.sprite.Sprite):
5     def __init__(self, color, width, height):
6         super().__init__()
7
8         # set the background color and set it to be transparent
9         self.image = pygame.Surface([width, height])
10        self.image.fill(BLACK)
11        self.image.set_colorkey(BLACK)
12
13        # draw the paddle (a rectangle)
14        pygame.draw.rect(self.image, color, [0, 0, width, height])
15
16        # fetch the rectangle object that has the dimensions of the image
17        self.rect = self.image.get_rect()
18
19    def moveUp(self, pixels):
20        self.rect.y -= pixels
```

```
21          # check that you are not going too far (off the screen)
22          if self.rect.y < 0:
23            self.rect.y = 0
24
25      def moveDown(self, pixels):
26          self.rect.y += pixels
27          # check that you are not going too far (off the screen)
28          if self.rect.y > 400:
29            self.rect.y = 400
```

ball.py

```
1  import pygame
2  from random import randint
3
4  BLACK = (0, 0, 0)
5
6  class Ball(pygame.sprite.Sprite):
7      def __init__(self, color, width, height):
8          super().__init__()
9
10         # set the background color and set it to be transparent
11         self.image = pygame.Surface([width, height])
12         self.image.fill(BLACK)
13         self.image.set_colorkey(BLACK)
14
15         # draw the ball (a rectangle)
16         pygame.draw.rect(self.image, color, [0, 0, width, height])
17
18         self.velocity = [randint(4,8),randint(-8,8)]
19
20         # fetch the rectangle object that has the dimensions of the image
21         self.rect = self.image.get_rect()
22
23     def update(self):
24         self.rect.x += self.velocity[0]
25         self.rect.y += self.velocity[1]
26
27     def bounce(self):
28         self.velocity[0] = -self.velocity[0]
29         self.velocity[1] = randint(-8,8)
```

Увоз и иницијализација PyGame библиотеке

Наша игрица ће започети са два следећа реда кода:

```
1  import pygame
2  pygame.init()
```

Дефинисање боја које ће се користити

Сваку боју коју користимо ћемо прогласити константом. Понг је основна игра и користи само две боје: белу и црну.

```
9  # define some colors
10 BLACK = (0,0,0)
11 WHITE = (255,255,255)
```

Отварање новог прозора

Наша игрица ће се покретати у свом прозору, за који можемо да одредимо наслов, висину и ширину.

```
13 # open a new window
14 size = (700, 500)
15 screen = pygame.display.set_mode(size)
16 pygame.display.set_caption("PongGame")
```

Главни део програма

Главни део програма садржаће три дела:

- Хватање догађаја – Користи се за стално преслушавање корисничких уноса и реаговање на њих. То може бити када корисник користи тастатуру или миш.
- Промена логике игре – Шта се дешава када се игрица игра?
- Освежавање екрана поновним цртањем сцене и sprites[1].

Главни програм користи брзину кадрова да би одлучио колико често програм треба да заврши петљу и освежи екран у секунди. Да бисмо то применили, користићемо објекат clock из PyGame библиотеке.

Главни део ће помоћу тајмера одредити колико ће се пута извршавати у секунди.

```
1 import pygame
2
3 pygame.init()
4
5 # define some colors
6 BLACK = (0,0,0)
7 WHITE = (255,255,255)
8
9 # open a new window
10 size = (700, 500)
11 screen = pygame.display.set_mode(size)
12 pygame.display.set_caption("PongGame")
13
14# the loop will carry on until the user exit the game (clicks the close button)
15 carryOn = True
16
17# the clock will be used to control how fast the screen updates
18 clock = pygame.time.Clock()
19
20# initialise player scores
21 scoreA = 0
22 scoreB = 0
23
24# -------- main program -----------
25 while carryOn:
26    # --- main event
27    for event in pygame.event.get(): # user did something
28        if event.type == pygame.QUIT: # if user clicked close
29              carryOn = False # flag that we are done so we exit this loop
30
31    # --- game logic
32
33    # --- drawing code
34    # first, clear the screen to black.
35    screen.fill(BLACK)
36    # draw the net
37    pygame.draw.line(screen, WHITE, [349, 0], [349, 500], 5)
38
39    # --- go ahead and update the screen with what we've drawn
40    pygame.display.flip()
41
42    # --- limit to 60 frames per second
43    clock.tick(60)
44
45# once we have exited the main program loop we can stop the game engine:
46 pygame.quit()
47
```

[1] Рачунарска графика која се може премештати на екран и на други начин манипулисати њоме као јединственом целином

Креирање класе paddle

Посматрајмо sprite као објекат. Направићемо прву класу paddle и из ње извести објекте `paddleA` и `paddleB`.

Играч А ће моћи да контролише први рекет (`paddleA`) користећи тастере: W за померање на горе и S за померање на доле, док ће играч B контролисати дрги рекет (`paddleB`) користећи стрелице за горе и доле.

За почетак, метода која нам је потребна је `__init__()`. Зове се конструктор. Користи се када се објекат први пут креира за иницирање главних својстава предмета.

```
1 import pygame
2 BLACK = (0,0,0)
3
4 class Paddle(pygame.sprite.Sprite):
5     def __init__(self, color, width, height):
6         super().__init__()
7
8         # set the background color and set it to be transparent
9         self.image = pygame.Surface([width, height])
10        self.image.fill(BLACK)
11        self.image.set_colorkey(BLACK)
12
13        # draw the paddle (a rectangle)
14        pygame.draw.rect(self.image, color, [0, 0, width, height])
15
16        # fetch the rectangle object that has the dimensions of the image
17        self.rect = self.image.get_rect()
```

Касније ћемо овој класи додати још својстава и метода. Али пре него што то урадимо, направићемо прве објекте.

Креирање објеката

Сада када имамо класу можемо да креирамо објекте из ове класе.

У `main.py` на почетку додамо увоз за класу `Paddle`.

```
1.  import pygame
2.  from paddle import Paddle
```

Затим треба да креирамо и поставимо наше sprites у наш главни програм користећи следеће редове кода:

```
19 paddleA = Paddle(WHITE, 10, 100)
20 paddleA.rect.x = 20
21 paddleA.rect.y = 200
22
23 paddleB = Paddle(WHITE, 10, 100)
24 paddleB.rect.x = 670
25 paddleB.rect.y = 200
```

Пошто креирамо игрицу која може да се игра урадићемо још неколико ствари са овим објектима.

```
1 import pygame
2 from paddle import Paddle
3
4 pygame.init()
5
6 # define some colors
7 BLACK = (0,0,0)
```

```
8 WHITE = (255,255,255)
9
10# open a new window
11 size = (700, 500)
12 screen = pygame.display.set_mode(size)
13 pygame.display.set_caption("PongGame")
14
15
16 paddleA = Paddle(WHITE, 10, 100)
17 paddleA.rect.x = 20
18 paddleA.rect.y = 200
19
20 paddleB = Paddle(WHITE, 10, 100)
21 paddleB.rect.x = 670
22 paddleB.rect.y = 200
23
24 ball = Ball(WHITE,10,10)
25 ball.rect.x = 345
26 ball.rect.y = 195
27
28# this will be a list that will contain all the sprites we intend to use in our
game
29 all_sprites_list = pygame.sprite.Group()
30
31# add the paddle and ball to the list of objects
32 all_sprites_list.add(paddleA)
33 all_sprites_list.add(paddleB)
34
35# the loop will carry on until the user exit the game (clicks the close button)
36 carryOn = True
37
38# the clock will be used to control how fast the screen updates
39 clock = pygame.time.Clock()
40
41# -------- main program -----------
42 while carryOn:
43    # --- main event
44    for event in pygame.event.get(): # user did something
45        if event.type == pygame.QUIT: # if user clicked close
46              carryOn = False # flag that we are done so we exit this loop
47        elif event.type==pygame.KEYDOWN:
48                if event.key==pygame.K_x: # pressing the x key will quit the game
49                     carryOn=False
50
51    # --- game logic
52    all_sprites_list.update()
53
54    # --- drawing code
55    # first, clear the screen to black.
56    screen.fill(BLACK)
57    # draw the net
58    pygame.draw.line(screen, WHITE, [349, 0], [349, 500], 5)
59
60    # now let's draw all the sprites in one go (for now we only have 2 sprites)
61    all_sprites_list.draw(screen)
62
63    # --- go ahead and update the screen with what we've drawn
64    pygame.display.flip()
65
66    # --- limit to 60 frames per second
67    clock.tick(60)
68
69# once we have exited the main program loop we can stop the game engine:
70 pygame.quit()
71
```

На 24. линији декларишемо листу која се зове `all_sprites_list` која ће чувати све sprites које ћемо креирати у нашој игрици. За сада имамо два објекта paddleA и paddleB.

Од 22. реда креирамо прве објекте помоћу класе `Paddle`.

Сада када смо креирали наше прве објекте морамо их додати на нашу листу sprites: `all_sprites_list`. То се извршава на 27. и 28. линији.

Додавање метода класи Paddle

У класи Paddle додајмо код на линијама од 19 до 29.

```
19    def moveUp(self, pixels):
20        self.rect.y -= pixels
21        # check that you are not going too far (off the screen)
22        if self.rect.y < 0:
23          self.rect.y = 0
24
25    def moveDown(self, pixels):
26        self.rect.y += pixels
27        # check that you are not going too far (off the screen)
28        if self.rect.y > 400:
29          self.rect.y = 400
```

Као што можемо видети нашој класи смо додали две методе: `moveUp` и `moveDown`. Обе методе примају два аргумента. Први је имплицитни и назива се self. Односи се на тренутни објекат. Други је pixles и односи се на број пиксела који ћемо користити за померање рекета.

Одговори на догађаје – притисак тастера

У главном коду имамо део који даје одговоре на догађаје као што су корисничке интеракције када корисник користи миш или тастатуру.

Па додајмо сада четири event handlers-а, да померамо рекет горе или доле када играчи притисну тастере W или S или стрелице горе или доле. Сваки event handler позваће одговарајућу методу из класе `Paddle`.

Додати event handler су на линијама од 51. до 60.

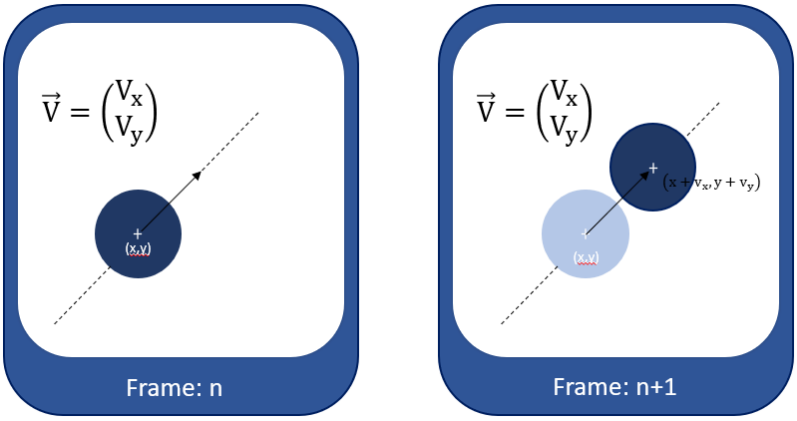
```
51  # moving the paddles when the use uses the arrow keys (player A) or "W/S" keys
(player B)
52    keys = pygame.key.get_pressed()
53    if keys[pygame.K_w]:
54        paddleA.moveUp(5)
55    if keys[pygame.K_s]:
56        paddleA.moveDown(5)
57    if keys[pygame.K_UP]:
58        paddleB.moveUp(5)
59    if keys[pygame.K_DOWN]:
60        paddleB.moveDown(5)
```

Алгоритам одбијања

Да бисмо разумели како применити алогритам одбијања, неопходно је разумети како рачунар контролише путању лоптице на екрану.

Аркадне игре се заснивају на анимацији заснованој на кадру где се екран освежава сваких x милисекунди. Лоптице који се крећу позициониране су помоћу координата (x, y) и имају вектор брзине (Vx, Vy) који одређује делту у пикселима за примену на (x, y) координатама лоптице између два оквира:

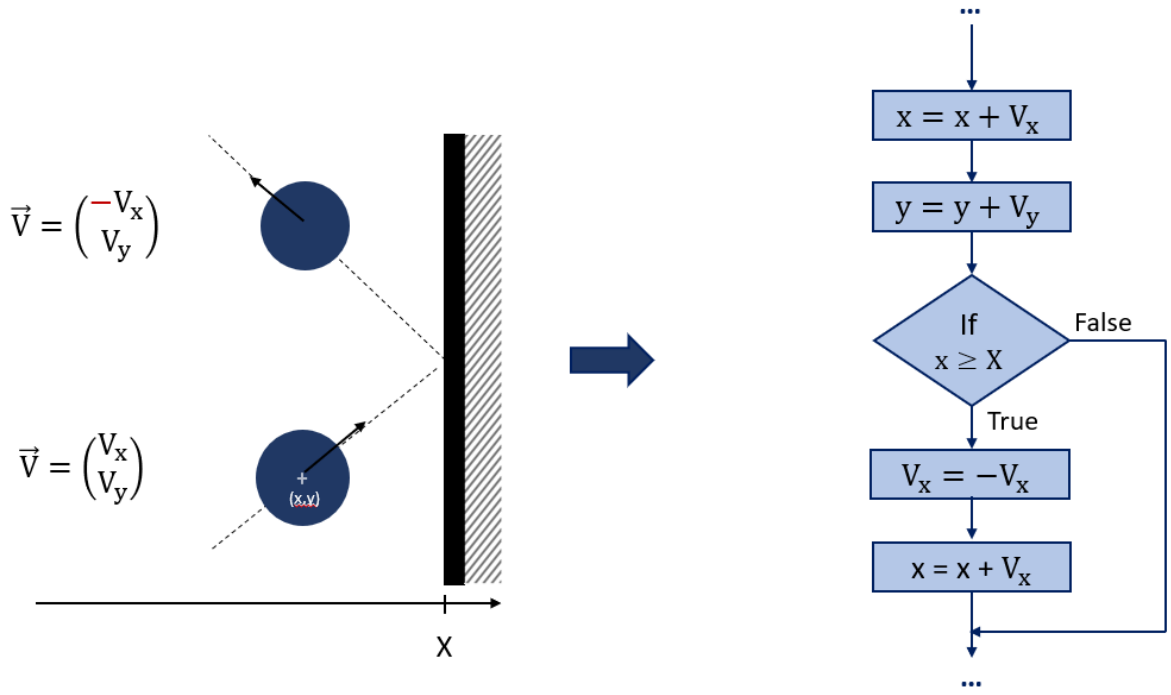
- Оквир n: Координате лоптице: (x, y)
- Оквир n+1: Координате лоптице: (x+Vx, y+Vy)



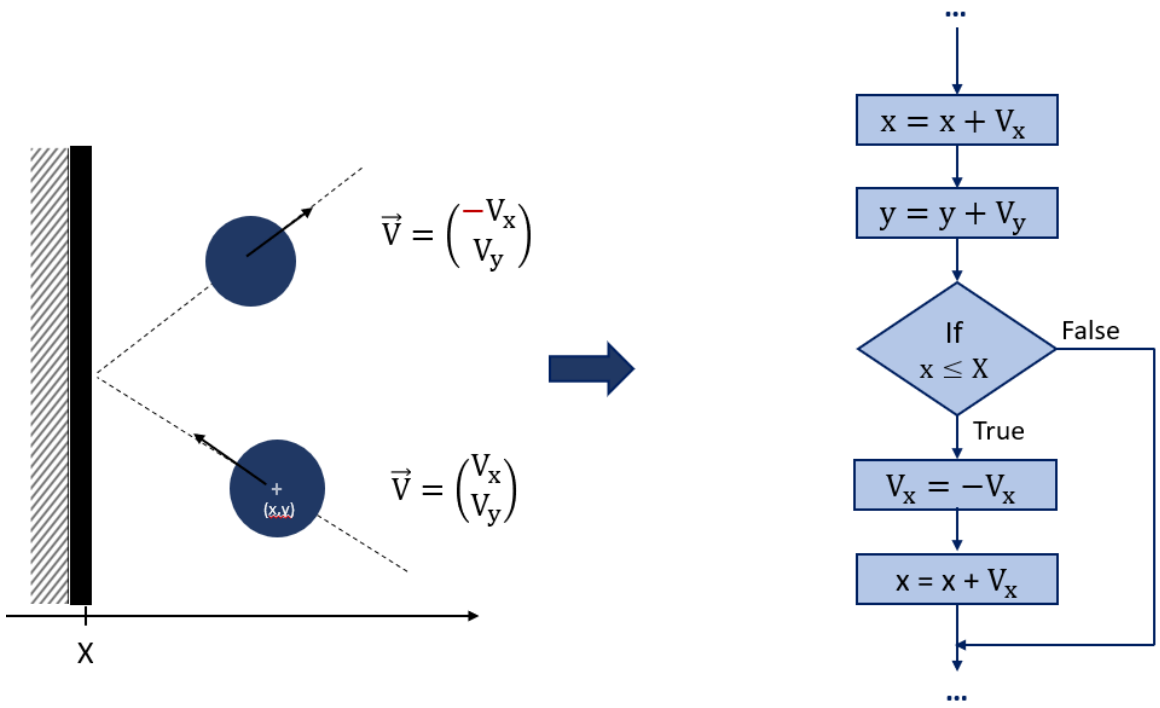
Како се лоптица креће по екрану можда ће требати да се одбије о други објекат или ивицу екрана.

Утицај на вектор брзине када се лоптица одбије према вертикалним и хоризонталним зидовима/ивицама:

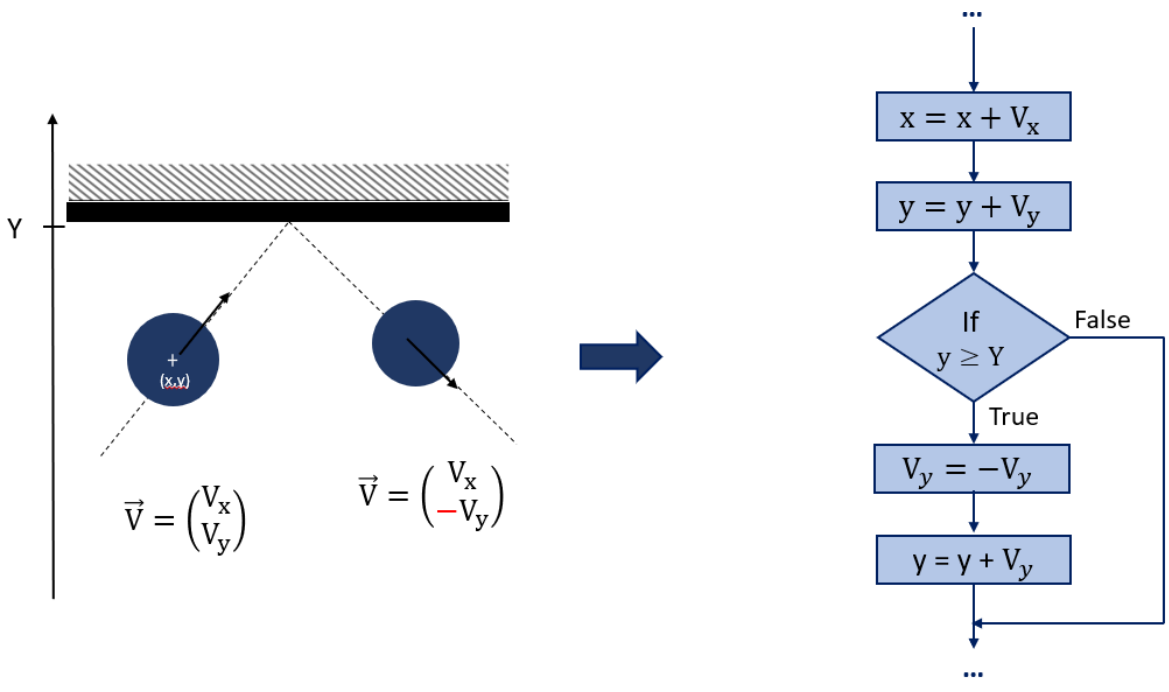
Десни зид



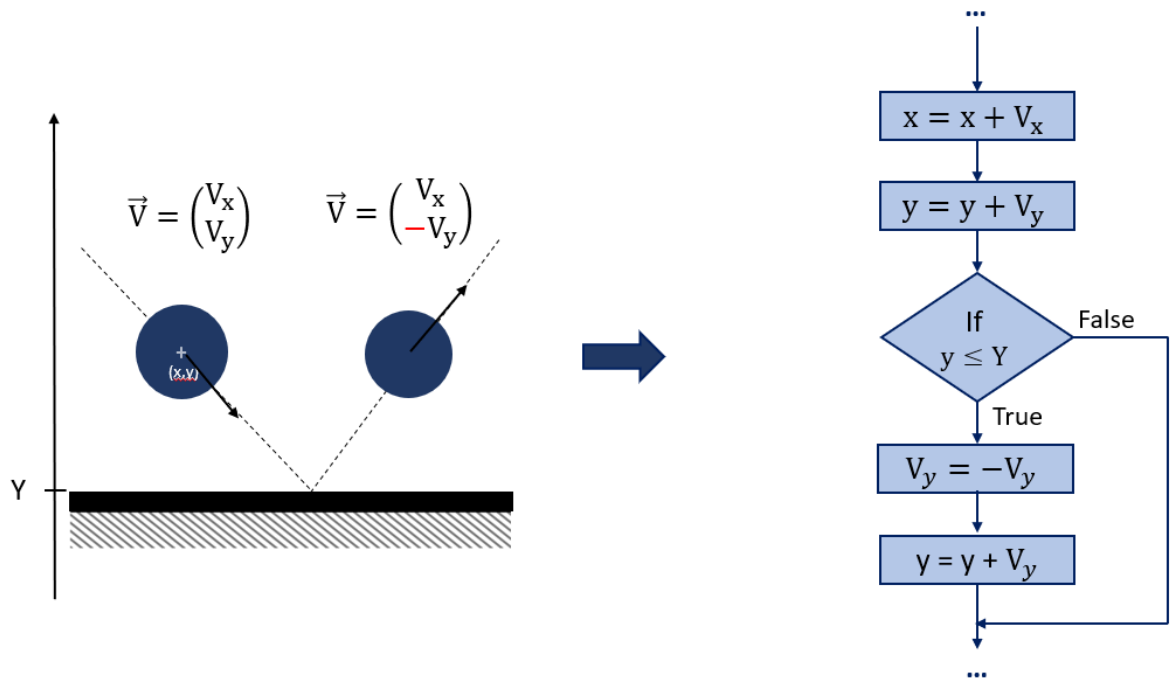
Леви зид



Горњи зид/ивица



Доњи зид/ивица



Класа Ball

Метода `update()` позваће се сваки оквир главног програма. Помера (мења (x, y) координате) лоптице користећи њен вектор брзине.

```
1 import pygame
2 from random import randint
3
4 BLACK = (0, 0, 0)
5
6 class Ball(pygame.sprite.Sprite):
7     def __init__(self, color, width, height):
8         super().__init__()
9
10        # set the background color and set it to be transparent
11        self.image = pygame.Surface([width, height])
12        self.image.fill(BLACK)
13        self.image.set_colorkey(BLACK)
14
15        # draw the ball (a rectangle)
16        pygame.draw.rect(self.image, color, [0, 0, width, height])
17
18        self.velocity = [randint(4,8),randint(-8,8)]
19
20        # fetch the rectangle object that has the dimensions of the image
21        self.rect = self.image.get_rect()
22
23    def update(self):
24        self.rect.x += self.velocity[0]
25        self.rect.y += self.velocity[1]
```

Додавање лоптице игри

У `main.py` увешћемо класу `Ball` – линија 3.

```
3  from ball import Ball
```

Затим ћемо креирати објекат који се зове `ball` користећи класу `Ball` – линије од 28. до 30.

```
28 ball = Ball(WHITE,10,10)
29 ball.rect.x = 345
```

```
30 ball.rect.y = 195
```

Додаћемо овај објекат у групу sprites `all_sprites_list` – линија 38.

```
38 all_sprites_list.add(ball)
```

Детекција судара

Следећи корак је да у нашу игрицу додамо откривање када лоптица удари/сударе се са једним од два рекета. Ако се то догоди, учинићемо да одскочи користећи случајни нови правац.

`ball.py`

Додата метода `bounce()` на 27., 28. и 29. линији.

```
27     def bounce(self):
28         self.velocity[0] = -self.velocity[0]
29         self.velocity[1] = randint(-8,8)
```

`main.py`

У редовима 87 и 88 додали смо код за откривање сударa између лоптице и рекета.

```
87     # detect collisions between the ball and the paddles
88     if pygame.sprite.collide_mask(ball, paddleA) or
pygame.sprite.collide_mask(ball, paddleB):
```

Додавање система за бодовање

Играч А ће постићи поен ако се лоптица одбије о десну бочну ивицу екрана, док ће играч В постићи поен ако се лоптица одбије о леву бочну ивицу екрана. Оба резултата (поени) ће бити приказана на врху екрана.

`main.py`

На линијама 46 и 47 иницијализујемо поене за оба играча на 0.

```
46 scoreA = 0
47 scoreB = 0
```

На линијама од 76 до 80 детектујемо када играч постигне поен.

```
76     if ball.rect.x>=690:
77         scoreA+=1
78         ball.velocity[0] = -ball.velocity[0]
79     if ball.rect.x<=0:
80         scoreB+=1
```

На линијама од 101 до 105 приказујемо резултате на екран.

```
101    font = pygame.font.Font(None, 74)
102    text = font.render("A: " + str(scoreA), 1, WHITE)
103    screen.blit(text, (230,10))
104    text = font.render("B: " + str(scoreB), 1, WHITE)
105    screen.blit(text, (370,10))
```

Додавање pop-up прозора на почетку игре

У `main.py` увешћемо једну библиотеку.

```
4  from tkinter import *
5  from tkinter import messagebox
```

Затим на линији 50 намештамо да се главни прозор `tkinter` библиотеке сакрије.

```
50 Tk().wm_withdraw() #to hide the main window
```

На 51. линији правимо pop-up прозор.

```
51 messagebox.showinfo("Play", "Press OK when you want to start, then on a black surface when the game starts so you can play")
```

Након што један од играча кликне на OK игра почиње, али да би играчи могли да померају свој рекет један од њих мора да кликне на црну позадину.

Додавање pop-up прозора за проглашење победника

У претходном кораку смо увезли библиотеку tkinter и сада је опет користимо.

На линијама од 106. до 115. иде код који одређује који играч је победио и исписује то у pop-up прозору.

```
106     if scoreA==30 or scoreB==30:
107         if scoreA>scoreB:
108             Tk().wm_withdraw() #to hide the main window
109             text = "Player A win\nScore: A: {}  B: {} ".format(scoreA, scoreB)
110             messagebox.showinfo("Winner", text)
111         else:
112             Tk().wm_withdraw() #to hide the main window
113             text = "Player A win\nScore: A: {}  B: {} ".format(scoreA, scoreB)
114             messagebox.showinfo("Winner", text)
115         carryOn = False
```

Резиме

Понг је једна од најранијих аркадних видео игара, коју је Атари први пут објавио 1972. године. То је игра за два играча заснована на стоном тенису. Игра садржи једноставну 2D графику. Састоји се од два рекета која се користе за враћање одскочне лоптице напред - назад по екрану. Резултат се чува у бојевима на врху екрана.