

بسمه تعالی

پروژه پایانی برنامه نویسی اسمبلی

درس ساختار و زبان کامپیوتر

دکتر جهانگیر

امیرحسین یگانه دوست 402106767

لینک گیت هاب پروژه:

https://github.com/Yeganeh62722/CSL_project/tree/master

برنامه شماره 1: برنامه ای بنویسید که دست کم قسمت محاسباتی اصلی و زمانبر آن به زبان اسمبلی x86 باشد و بر اساس حداقل سه گزینه مسیر قابل انتخاب توسط کاربر، از یک سوی تصویر، یک توپ رنگی راه راه را به سمت دروازه یا دیوار مقابل بفرستد. مسیرهای قابل انتخاب باید مستقیم زاویه دار، محدب (مثل پرتاب توپ والیبال به سمت مقابل) و سینوسی (!) (عجیب است ولی چرا که نه؟) ... باشد. بنابراین کافی است یک بار برنامه خود را به هر زبان دلخواه سطح بالا بنویسید و بعد، دست کم بخش زمانبر محاسبه مسیر و معادله حرکت توپ را با استفاده از دستورات اسمبلی سریع ممیز شناور و ترجیحاً برداری، پیاده سازی و جایگزین دستورات معادل قبلی در برنامه اولیه سطح بالای خود کنید و بعد، برای ارزیابی و مشاهده سرعت برنامه نویسی اسمبلی این بازی ساده کامپیوتری، سرعت پرتاب توپ را در هر دقیقه در دو حالت فوق مقایسه و گزارش کنید.

برخی کارهای جالب اضافه که میتوانید انجام دهید (و در نمره تاثیر مثبت و حتی امتیازی داشته باشد) میتواند شامل موارد زیر باشد:

- شما در طرف مقابل، به کمک ماوس با برنامه خود بازی کنید و پاسخ متقابل به توپ را (بسته به جهت دریافت و احتمالاً نیروی وارد شده) بدهید و هر واکنش به موقع شما امتیاز مثبت داشته باشد. بعد سرعت بازی (پرتاب کامپیوتری توپ) را کم کم زیاد و زمان واکنش خود را ارزیابی کنید.
 - به توپ رنگارنگ حرکت چرخشی نیز بدهید و رنگهایش در طول حرکت و بسته به جهت ضربه وارد شده به طرف مناسب بچرخد.
 - از یک طرف، برنامه شما توپ را پرتاب کند و در آن سو، یک برنامه هوش مصنوعی با شما مثلاً تنیس یا بدمینتون بازی کند.
- توجه: علاوه بر تحویل حضوری اجرای برنامه، باید گزارشی نیز شامل شرحی از کارکرد برنامه و روش یا الگوریتم به کار رفته ارائه دهید. برنامه هایتان نیز باید به حد کافی دارای توضیح (Comment) دستورات، ثابتهای برنامه و ساخت یافته باشد.

1. بخش اول: بازی پرتاب توپ به زبان پایتون

بازی توصیف شده را به زبان پایتون پیاده میکنیم. بخش های گرافیکی را با استفاده از کتابخانه pygame پیاده کرده و بخش محاسباتی را نیز به زبان پایتون می نویسیم.

بخش محاسباتی:

```
# Update ball position if thrown straight
if ball_thrown_straight:
    ball_x += ball_speed_x
    if ball_x + BALL_RADIUS >= WALL_X:
        ball_speed_x = -ball_speed_x

# Update ball position for curve throw
if ball_thrown_curve:
    ball_x += ball_speed_x
    ball_y += ball_speed_y
    ball_speed_y += 0.07
    if ball_x + BALL_RADIUS >= WALL_X:
        ball_speed_x = -ball_speed_x
        ball_speed_y = 0

# Update ball position for sinusoidal throw
if ball_thrown_sinusoidal:
    ball_x += ball_speed_x
    sinusoidal_angle += 0.1
    ball_y = (GROUND_Y - 54) - 50 * math.sin(sinusoidal_angle)
    if ball_x + BALL_RADIUS >= WALL_X:
        ball_speed_x = -ball_speed_x
```

در این بخش مختصات توپ را با توجه به معادلات فیزیک سینماتیک بسته به نوع حرکت که مستقیم، پرتابی و یا سینوسی باشد محاسبه کرده و در هر لحظه به روز میکنیم. به این صورت توپ در مسیری که میخواهیم حرکت میکند.

Speed: 8.0



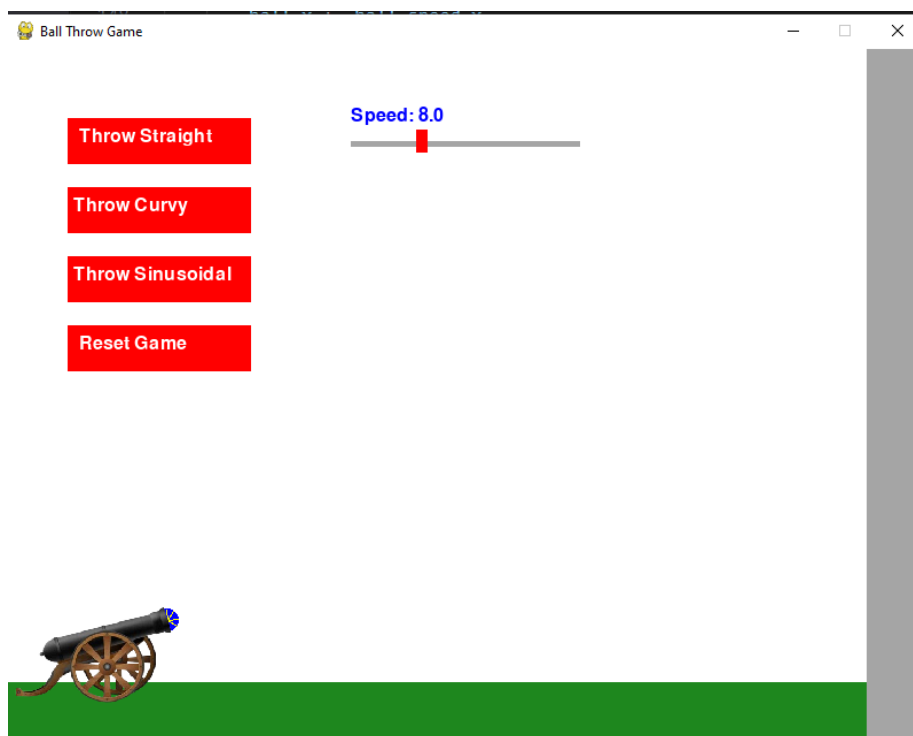
به عنوان یک امکان اضافه در بازی، یک تنظیم کننده وجود دارد که میتوان سرعت حرکت توپ را با آن تنظیم نمود.

همانطور که در توضیحات پروژه مشخص شده، توپ دارای خطوط

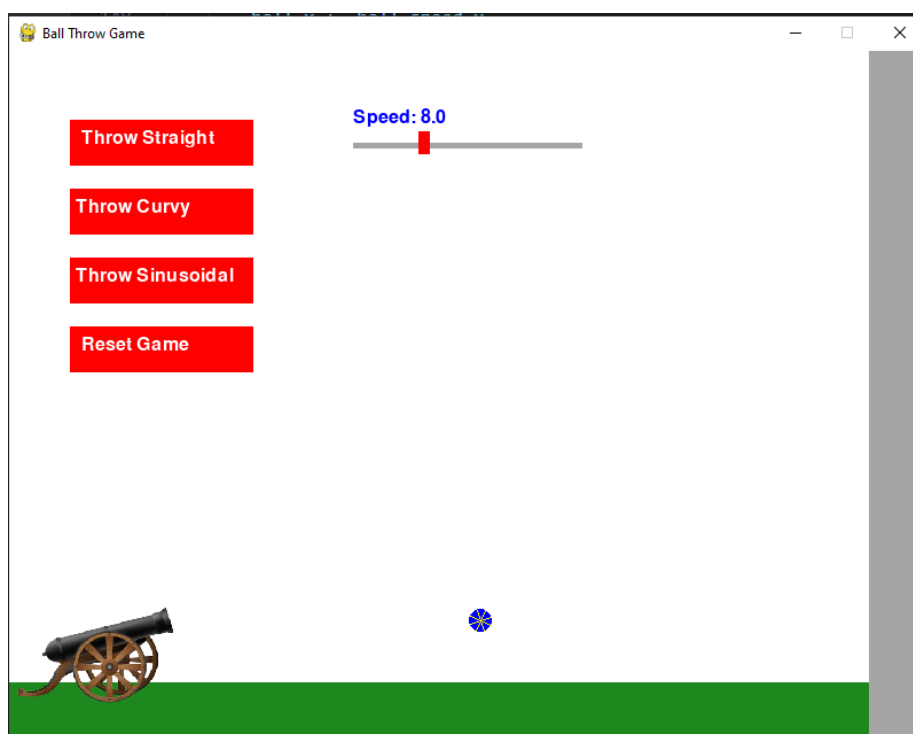


رنگی است که هنگام حرکت در جهت آن می چرخند.

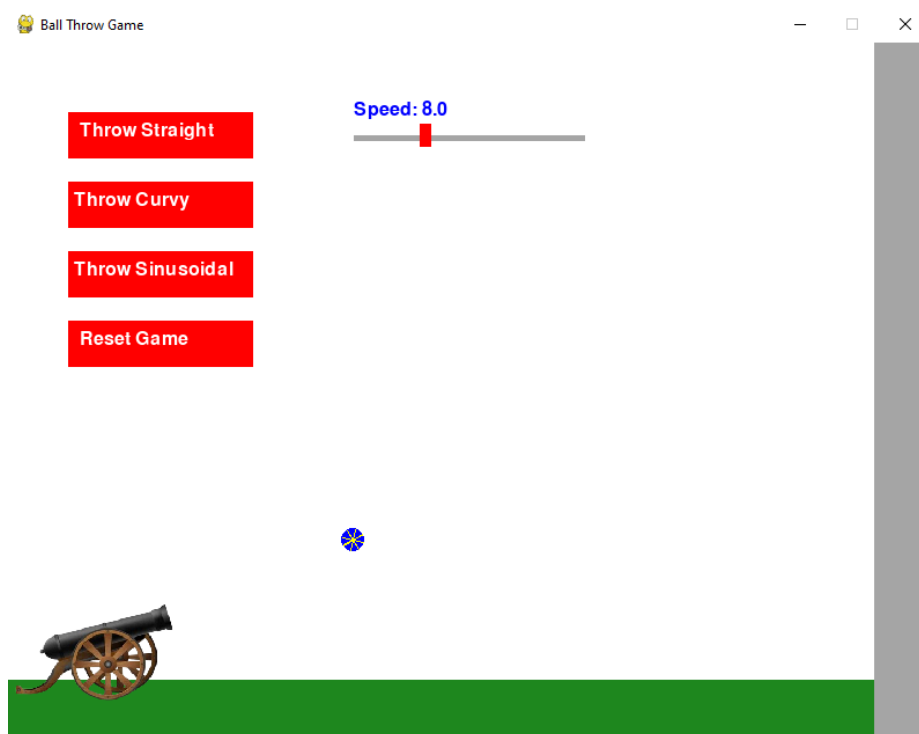
نمایی از صفحه بازی:



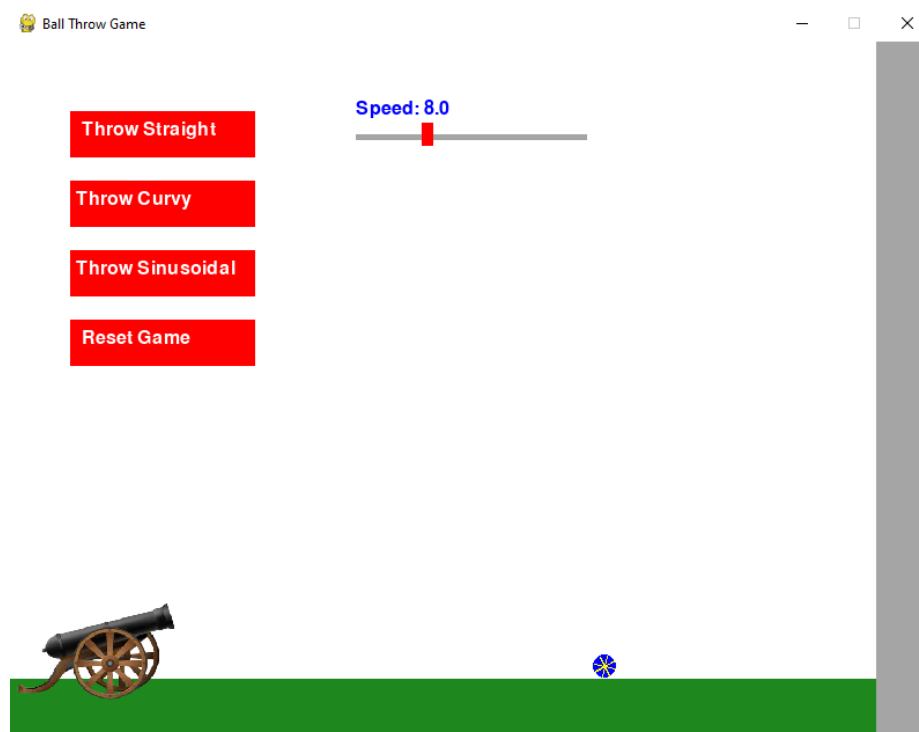
1.1) پرتاب مستقیم



1.2) پرتاب منحنی



1.3) پرتاب سینوسی



2. بخش دوم: پیاده سازی توابع محاسباتی به زبان اسمبلی

در این بخش می خواهیم بخش محاسباتی را به زبان اسمبلی پیاده نموده تا موجب سریعتر شدن اجرای برنامه شود.

در ابتدا کد اسمبلی معادل با بخش محاسباتی هر بخش را پیاده سازی میکنیم.

```
asm straight.s
1  section .text
2  | global update_ball_position
3
4  ; Function: update_ball_position
5  ; Parameters:
6  ;   RCX - pointer to ball_x (int)
7  ;   RDX - pointer to ball_speed_x (int)
8  ; Updates ball_x by adding the value in ball_speed_x
9  update_ball_position:
10
11     ; Load ball_x and ball_speed_x
12     movss xmm0, [rcx]          ; xmm0 = ball_x
13     movss xmm1, [rdx]          ; xmm1 = ball_speed_x
14     addps xmm0, xmm1           ; ball_x += ball_speed_x
15     movss [rcx], xmm0          ; Store updated ball_x
16
17     ret
18
19
```

برای مثال در این قسمت کد جایگزین پرتاب مستقیم پیاده سازی شده است که در آن مقادیر سرعت و مکان X توپ در رجیسترهای برداری لود شده و سپس جمع زده می شوند و نتیجه دوباره در آدرس داده شده ذخیره می شود.

یا در عکس بعدی برای محاسبه حرکت سینوسی از تابع `fsin` بهره می گیریم، بعد از بارگذاری مقادیر ابتدا مقدار مکان X توپ را با سرعت توپ در راستای X جمع زده و نتیجه را بروز میکنیم. سپس زاویه سینوسی توپ را افزایش می دهیم و آن را ذخیره می کنیم.

در انتها برای محاسبه مکان Y توپ از پشته FPU کمک میگیریم و زاویه سینوسی را در آن لود میکنیم. سپس با صدا زدن `fsin` سینوس زاویه را محاسبه کرده و بعد آن را در مقدار ثابت دامنه 50 ضرب میکنیم. در نهایت مقدار عرض از مبدا 700 را در پشته وارد کرده و با صدا زدن `fsub` دو مقدار را از هم کم میکنیم و نهایتاً خروجی را ذخیره میکنیم.

```

update_sinusoidal_ball_position:
    ; Load ball_x and ball_speed_x
    movss xmm0, [rcx]          ; xmm0 = ball_x
    movss xmm1, [r8]           ; xmm1 = ball_speed_x
    addss xmm0, xmm1           ; ball_x += ball_speed_x
    movss [rcx], xmm0          ; Store updated ball_x

    ; Increment sinusoidal angle
    mov dword [rel sin_increment_val], 0x3DCCCCD ; 0.1 as 32-bit float
    movss xmm2, [rel sin_increment_val]
    movss xmm3, [r9]           ; xmm3 = sinusoidal_angle
    addss xmm3, xmm2           ; sinusoidal_angle += 0.1
    movss [r9], xmm3           ; Store updated sinusoidal_angle

    ; Compute ball_y using the formula: ball_y = 700 - 50 * sin(sinusoidal_angle)
    fld dword [r9]             ; Load sinusoidal_angle onto the FPU stack
    fsin                       ; Compute sin(sinusoidal_angle)
    fmul dword [rel sin_amplitude_val] ; Multiply by 50.0
    fld dword [rel base_height] ; Load 700.0
    fsub                        ; 700 - result
    fstp dword [rdx]           ; Store updated ball_y

    ret

```

پس از پیاده سازی اسمبلی با استفاده از دستورات کامپایلرهای nasm و gcc ابتدا object file این کد را به وجود آورده و سپس فایل اجرایی با پسوند dll را درست میکنیم تا از آن در برنامه استفاده کنیم.

```

# Load the assembly
straight_lib = ctypes.CDLL(r'./straight.dll')
curvy_lib = ctypes.CDLL(r'./curvy.dll')
sinusoidal_lib = ctypes.CDLL(r'./sinusoidal.dll')

```

بارگذاری فایل های اجرایی اسمبلی

```
# Update ball position if thrown straight
if ball_thrown_straight:
    # Example game variables
    ball_x = ctypes.c_int(int(ball_x))
    ball_speed_x = ctypes.c_int(int(ball_speed_x))
    straight_lib.update_ball_position.argtypes = [POINTER(ctypes.c_int), POINTER(ctypes.c_int)]
    straight_lib.update_ball_position.restype = None
    # Call the Assembly function
    straight_lib.update_ball_position(byref(ball_x), byref(ball_speed_x))
    ball_x = int(ball_x.value)
    ball_speed_x = int(ball_speed_x.value)
    if ball_x + BALL_RADIUS >= WALL_X:
        ball_speed_x = -ball_speed_x
```

نکته قابل توجه این است که باید پیش از مقدار دهی و پس از گرفتن مقادیر از فایل اسمبلی نوع آنها را تغییر دهیم. یعنی ابتدا آنها را به متغیر های C تبدیل کرده و سپس مقادیر آنها را به متغیر های پایتون برگردانیم.


حال با تعدادی آزمایش پرتاب سرعت برنامه ها را در پرتاب های مختلف میسنجیم.(برای راحتی کار یک تکه کد به برنامه اضافه میکنیم تا با فشردن دکمه M نحوه محاسبات از پایتون به اسمبلی و برعکس تغییر کند.)

Speed: 1.0



Mode: Python

Speed: 1.0



Mode: Assembly

در صفحه بعد می توانید نتایج آزمایش را مشاهده کنید.

A	B	C	D	E	F	G
	Assembly: Sin	Python: Sin	Assembly: Straight	Python Straight	Assembly: Curvy	Python: Curvy
1	1.417043686	1.410205364	1.440274715	1.520180941	0.262314558	0.2759745121
2	1.447938442	1.441894293	1.448228121	1.447213888	0.2609250546	0.271327734
3	1.419920206	1.417390585	1.428885698	1.496795416	0.2856862545	0.2905795574
4	1.423275948	1.420216084	1.575624466	1.468365669	0.2920467854	0.2839267254
5	1.430091619	1.416855812	1.472615004	1.451912403	0.2843124866	0.269872427
6	1.417724371	1.416743994	1.50272131	1.479940653	0.2764122486	0.2753787041
7	1.612066507	1.41593194	1.470282316	1.520629883	0.2830379009	0.2968318462
8	1.413717985	1.415056944	1.490297079	1.501910925	0.2873990536	0.2765638828
9	1.435508013	1.408723593	1.496993065	1.498631716	0.2922482491	0.2753844261
10	1.418591022	1.413929224	1.48703146	1.477479935	0.334505558	0.2756836414
11	1.434874058	1.425475359	1.502276421	1.48184824	0.297970295	0.2835400105
12	1.420279741	1.420038223	1.513324022	1.471811295	0.3018796444	0.2873754501
13	1.415134192	1.450073242	1.514533281	1.498678923	0.2883973122	0.2878251076
14	1.418480635	1.40122056	1.532669544	1.523703575	0.3042421341	0.2778842449
15	1.425642252	1.415529728	1.537036657	1.491385937	0.2861549854	0.2954413891
16	1.419430971	1.424673557	1.517501116	1.467809677	0.2854261398	0.2924308777
17	1.415144682	1.425474882	1.552488327	1.506646872	0.2811977863	0.2817485332
18	1.422874212	1.412443876	1.48900485	1.478043795	0.2968239784	0.2865123749
19	1.428725481	1.483293772	1.525104761	1.496429443	0.2745501995	0.2971076965
20	1.419786215	1.4397614	1.483029604	1.485407829	0.3069181442	0.2744646072
21	1.408757687	1.427026033	1.499116659	1.465194225	0.2858664989	0.2768547535
22	1.425844193	1.554890871	1.482085228	1.478674412	0.2848844528	0.2975978851
23	1.417638063	1.48361516	1.49499321	1.457616091	0.2701153755	0.2755298615
24	1.406979799	1.485029697	1.498279095	1.478664398	0.3017232418	0.2919719219
25	1.421954393	1.477169991	1.511334419	1.501507044	0.2933013439	0.2862164974
26	1.415806532	1.498896122	1.502208233	1.488636494	0.2830882072	0.2825677395
Average	1.428970419	1.43852155	1.498766872	1.485966141	0.2885164573	0.2833304772

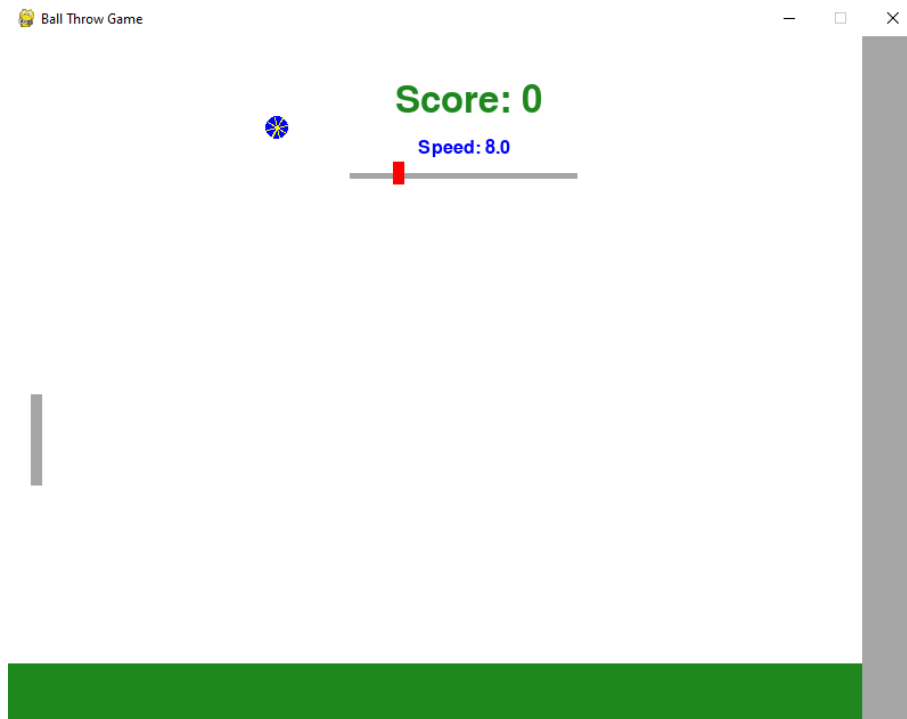
طبق انتظار ما در پرتاب سینوسی کد اسمبلی سریعتر کار میکند و ما توانسته ایم با پیاده سازی بخش محاسبات به زبان اسمبلی برنامه را سریعتر کنیم. اما در بخش پرتاب مستقیم این اتفاق نیفتاده است. نتیجه گیری ما از این اتفاق این است که برای نشان دادن تفاوت سرعت و بهتر بودن اسمبلی از این لحاظ باید محاسبات پیچیده تری را بازنویسی کرد.

دلیل آن این نیست که در محاسبات ساده تر زبان پایتون سریع تر است، بلکه در همه محاسبات زبان اسمبلی باید سریعتر باشد اما هنگام استفاده از زبان اسمبلی در یک بازی به زبان پایتون مقداری از زمان در لود کردن فایل dll و استفاده از آن تلف می شود و این باعث می شود در مجموع تکه کد پایتون سریعتر باشد. برای جبران این اتلاف وقت باید مقدار محاسبات را که در آن زبان اسمبلی سریعتر است بیشتر کرده و مشاهده کنیم تا بازی ما سریع تر می شود.

3. بخش سوم: قسمت امتیازی

3.1 بازی با دیوار

برنامه را بازنویسی می کنیم به طوری که بتوانیم به توپ ضربه بزنیم و توپ پس از برخورد به دیوار به سمت ما بازگردد و دوباره تلاش کنیم تا توپ از صفحه بازی خارج نشود. این کار را با اضافه کردن یک دیوار کوچک به عنوان راکت انجام می دهیم که در جهت محور y ها با حرکت موس جابجا شود.



در بالا یک تنظیم کننده سرعت وجود دارد که میتوان آن را کم و زیاد کرد و سختی بازی را تغییر داد و مهارت پاسخ خود را سنجید. با هر ضربه به توپ یک امتیاز میگیریم در صورتی که توپ از صفحه خارج شود نیز بازی تمام شده و امتیاز و رکورد کلی ما نمایش داده می شود.

Game Over! Press R to Restart

Score: 10

Record: 20

سرعت حرکت توپ با توجه به محل برخورد با راکت در آن لحظه محاسبه می شود.

```
# Ball collision with racket
if (
    ball_x <= RACKET_WIDTH + BALL_RADIUS * 3
    and racket_y < ball_y < racket_y + RACKET_HEIGHT
):
    score += 1

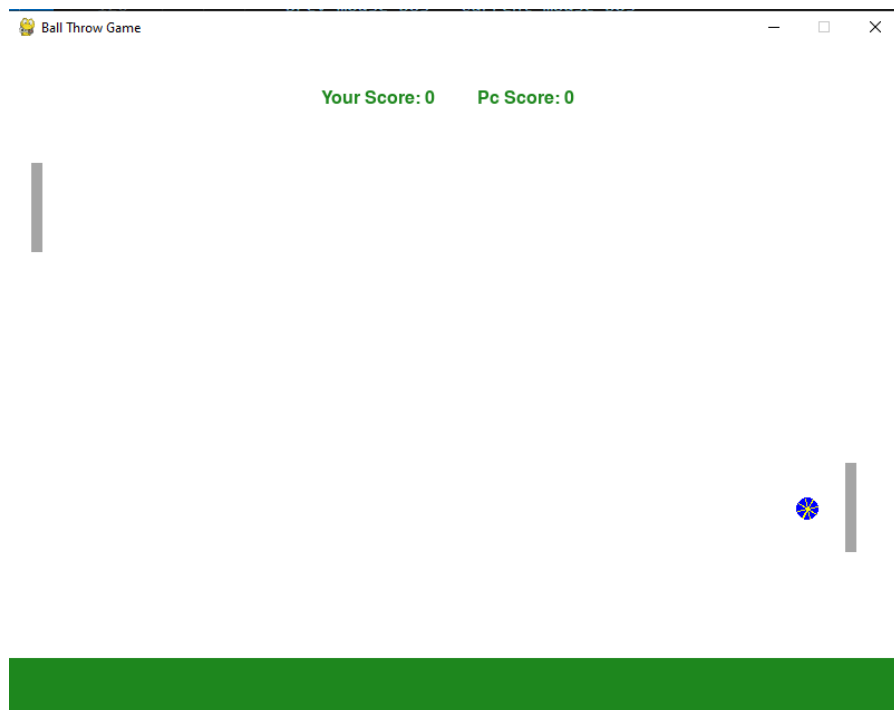
    ball_speed_x *= -1
    rotation_speed *= -1
    hit_pos = (ball_y - (racket_y + RACKET_HEIGHT // 2)) / (RACKET_HEIGHT // 2)

    # Adjust ball speed and Y direction based on hit position
    ball_speed_y += hit_pos * 3 # Increase speed variation
    ball_speed_y = max(min(ball_speed_y, 10), -10) # Cap speed to avoid runaway
```

3.2 بازی با کامپیوتر

در ادامه سعی کردیم بازی را به نحوی طراحی کنیم تا بتوان با کامپیوتر مسابقه داد.

ابتدا یک راکت هم برای کامپیوتر به جای دیوار در آن سمت در نظر گرفتیم، و برچسب هایی که امتیازها را نمایش دهند.



در این پیاده سازی جواب کامپیوتر با سرعتی متغیر خواهد بود که به صورت اتفاقی محاسبه می شود تا جواب دادن ما سخت تر شود.

```
# Ball collision with big wall
if (ball_x >= WIDTH - 50 - BALL_RADIUS and wall_y < ball_y < wall_y + RACKET_HEIGHT):
    ball_speed_x *= -1
    rotation_speed *= -1

# Adjust ball speed and Y direction based on hit position
ball_speed_y = int(random.random() * 30) - 15 # Cap speed to avoid runaway
ball_speed_x = int(random.random() * -12) - 3
```

نحوه پاسخ ما هم به محل برخورد توپ با راکت و سرعت توپ در آن لحظه بستگی دارد. سرعت لحظه ای توپ را به صورت به دست آوردن دیفرانسیل r با توجه به دیفرانسیل x و y در یک بازه زمانی بسیار کوتاه دیفرانسیل t محاسبه می شود.

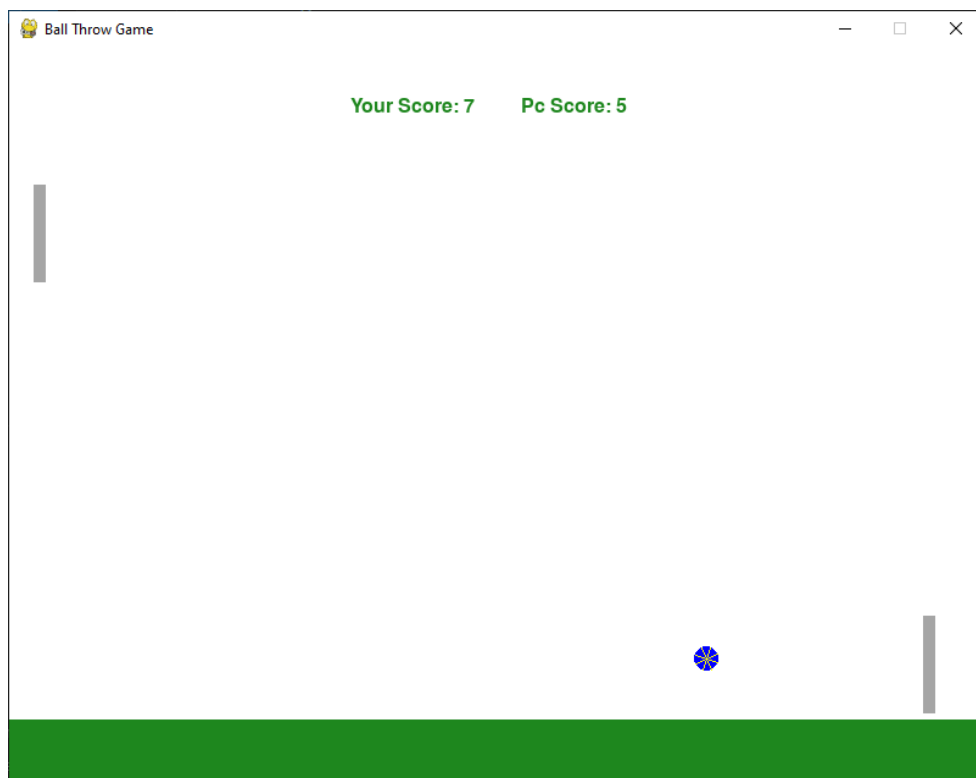
```
# Ball collision with racket
if (ball_x <= RACKET_WIDTH + BALL_RADIUS * 3 and racket_y < ball_y < racket_y + RACKET_HEIGHT):
    ball_speed_x *= -1
    rotation_speed *= -1
    hit_pos = (ball_y - (racket_y + RACKET_HEIGHT // 2)) / (RACKET_HEIGHT // 2)

# Compute mouse speed using the Euclidean distance formula
current_mouse_pos = pygame.mouse.get_pos()
dx = current_mouse_pos[0] - prev_mouse_pos[0]
dy = current_mouse_pos[1] - prev_mouse_pos[1]
mouse_speed = math.sqrt(dx**2 + dy**2)
prev_mouse_pos = current_mouse_pos

# Adjust ball speed and Y direction based on hit position
ball_speed_y += hit_pos * 3 # Increase speed variation
ball_speed_y = max(min(ball_speed_y, 10), -10) # Cap speed to avoid runaway
ball_speed_x = max(5 + mouse_speed//50, ball_speed_x)
```

حرکت راکت ما مانند قبل با مختصات موس است اما حرکت راکت کامپیوتر بسته به سرعت توپ فرق دارد و اگر سرعت توپ زیاد باشد نمیتواند آن را جواب دهد. به همین خاطر امکان امتیاز کسب کردن ما وجود دارد. از طرفی سرعت توپ نیز بستگی به سرعت حرکت دادن راکت توسط ما دارد و ما واقعا می توانیم با ضربه مناسب کامپیوتر را شکست دهیم.

```
# Wall moves with the ball
PC_WALL_SPEED = 5
# Smooth movement toward the ball's position
if ball_y > wall_y + RACKET_HEIGHT // 2:
    wall_y = min(wall_y + PC_WALL_SPEED, HEIGHT - RACKET_HEIGHT)
elif ball_y < wall_y + RACKET_HEIGHT // 2:
    wall_y = max(wall_y - PC_WALL_SPEED, 0)
```



بازی با کامپیوتر

در انتها هرکس به امتیاز 10 برسد برنده بازی می شود.



Game Over! Press R to Restart

PC won 10 - 9