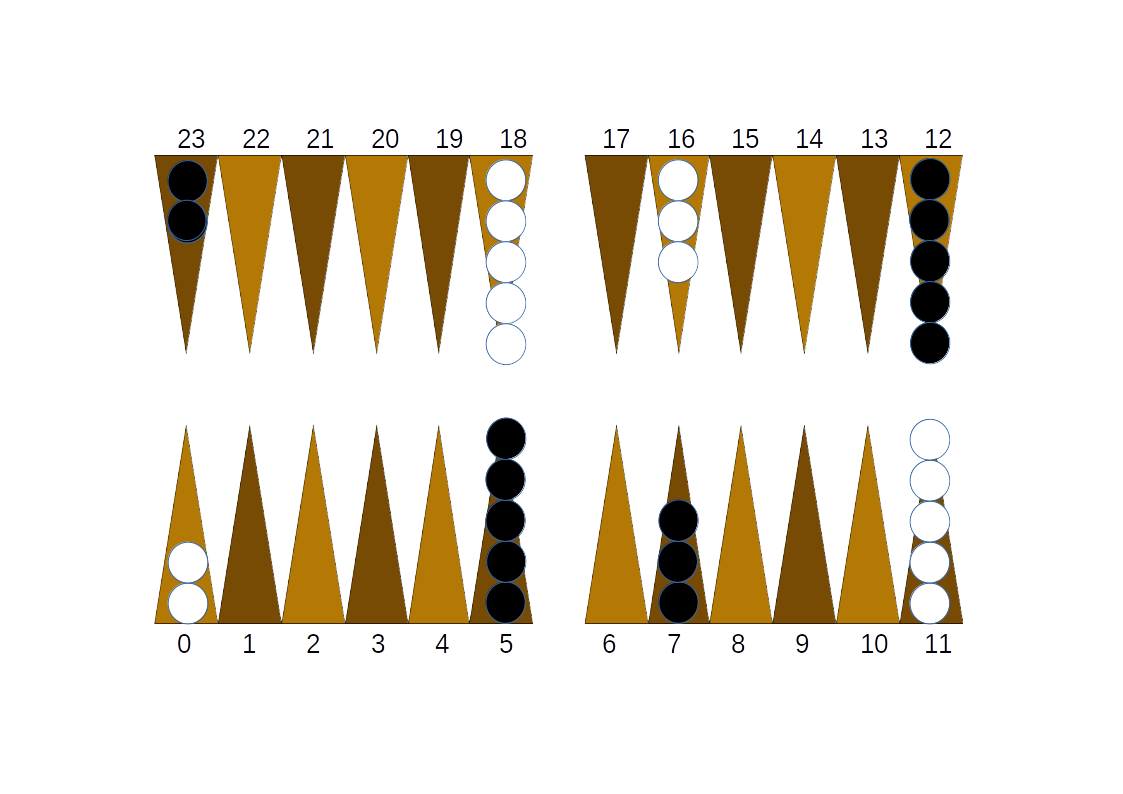
פרוייקט גמר: שש-בש

מגיש: דניאל שקד

**חוקי המשחק**:

המשחק מתנהל על גבי לוח שעליו 24 משולשים, 12 משולשים בחלקו התחתון ו12 משולשים בחלקו העליון. על גבי הלוח מסודרים 15 אבני משחק בצבע לבן ו15 אבני משחק בצבע שחור.

בשלב הראשון כל שחקן בתורו מטיל קובייה. השחקן שהטיל את המספר הגבוה ביותר הוא השחקן שמשחק את התור הראשון.



כל שחקן בתורו מטיל שתי קוביות ומזיז את האבנים שלו על פי כל קובייה בנפרד. השחקן חייב להזיז את האבנים בכל מצב שהלוח מאפשר לו והוא אינו רשאי לנוע אחורה. כדי לנצח במשחק צריך השחקן להכניס את כל האבנים שלו לבית שלו.(לפי הלוח בתמונה המצורפת הלבן צריך להכניס את כל האבנים למשולשים 18-23 והשחור צריך להכניס ל0-5)

כשלאחר מכן מכן השחקן רשאי להוציא אבנים מהלוח לפי הטלת הקוביות. במידה ובמשולש נמצאת רק אבן אחת. משולש זה מוגדר להיות משולש לכידה ובמידה והיריב הזיז אבן למשולש לכידה, אבן זו נאכלת. במידה ולשחקן יש אבן אכולה, כלומר מחוץ ללוח, השחקן אינו רשאי להזיז אבנים על גבי הלוח עד אשר האבן חוזרת למשחק. במידה ולשחקן אין אפשרות להכניס אבן זו, השחקן יוותר על תורו.

**מהלך המשחק**:

כאשר התוכנית מופעלת מוצג על המסך תפריט עם כפתור הפעלה וכפתור יציאה. אם זו פעם ראשונה שתפריט זה מוצג , כאשר לוחצים על כפתור ההפעלה מתחיל תהליך בחירת השחקן שישחק את התור הראשון. לאחר מכן מתחיל שלב המשחק ובו לפי הטלת הקוביות על השחקן ללחוץ על המשולש שבו נמצא האבן שברצונו להזיז ולאחר מכן ללחוץ על המשולש אליו הוא מזיז את האבן. שלב זה חוזר על עצמו עד לשלב הוצאת האבנים מהלוח. בשלב ההוצאה על השחקן ללחוץ פעמיים על המשולש ממנו הוא מוציא את האבן או להזיז את האבן כמו בשלב הראשון.

תהליך בניית המשחק:

בשלב הראשון של בניית המשחק נדרשתי להגדיר היטב את מהלך המשחק ואת כל מרכיביו. שלב זה היה השלב הקשה ביותר מכיוון שנאלצתי לחזור עליו מספר פעמים מכיוון שכל הבאגים שנתקלתי

בהם במהלך בניית המשחק ובדיקתו קרו כתוצאה מפונקציות ומשתנים שלא היו מוגדרים היטב.

שלב זה כלל בתוכו עבודה מול לוח שש-בש אמיתי ותרחישים שיכולים לקרות במשחק

**דוגמאות להגדרת משנים**:

לוח המשחק מוגדר להיות רשימה בגודל 24 המכילה בתוכה מספרים שלמים חיובים ושליליים המייצגים את מספר האבנים על כל משולש. לדוגמא במצב ההתחלתי הלוח מוגדר כ:

initial\_board=[-2,0,0,0,0,5,0,3,0,0,0,-5,5,0,0,0,-3,0,-5,0,0,0,0,2]

ה**קוביות** מוגדרות כרשימה של שני משתנים אקראיים

dices=[die1,die2] or [die1,die2,die1,die2]

**מצב הכלא**(אבני משחק אכולים) מוגדר כרשימה של שני מספרים שלמים גדולים או שווים ל0 כאשר כל אינדקס ברשימה זו שייך לשחקן.

jail=[white\_player\_jail, black\_player\_jail]

כמות אבני המשחק שהשחקנים הוציאו:

out=[white\_player\_out, black\_player\_out]

סטאטוס שחקן מוגדר כרשימה של 5 מספרים כאשר כל אינדקס מייצג מצב מסוים במשחק:

status=[played,jail,phase,length\_of\_dices\_list,skip]

values: played(0-4), jail(0 or 1), phase(0 or 1), length\_of\_dices(0-4), skip(0 or 1)

**צבעים**:

BLACK,WHITE,BROWNISH,GREENISH=(255,255,255),(0,0,0),(155,120,10),(200, 160, 10)

**הגדרת פונקציות עיקריות במשחק**:

מטעמי נוחות רוב הפונקציות אותן הגדרתי מקבלות ערכים ממשתנים גלובליים ומחזירות ערכים למשתנים גלובליים. מכיוון שלוח השש-בש סימטרי, החלטתי שכל הפונקציות מתייחסות ומבצעות את מטרתן עבור שחקן אחד ולכיוון אחד ובסוף כל תור להפוך את הלוח ואת שאר הרשימות. בגלל החלטה זו נדרשה תשומת לב רבה לכיוון המשחק ולשימוש בצבעים והגדרתן בכל תחילת תור.

**ציור הלוח על גבי המסך**:

המסך מורכב מ12 משולשים בצידו התחתון, 12 משולשים בצידו העליו, מלבן שנמצא במרכז המסך ועליו מונחות האבנים האכולות. שורת הוראות לשחקן בחלקו העליון של המסך וכפתור פוס משחק.

תחילה כתבתי פונקציה שמייצרת את הנקודות של הפוליגונים המוצגים על המסך. הפונקציה מקבלת את גודל המסך ומחזירה רשימה של תתי רשימות כאשר כל תת רשימה מייצגת נקודות המרכיבות צורה(משולש או מלבן):

def generate\_polygons():

global size,points

height=size[1]//3

width=size[0]//13

for i in range(0,size[0],width):

if i!=480:

triangle\_down=[[i,size[1]],[(i+width/2),size[1]-height],[i+width,size[1]]]

points.append(triangle\_down)

triangle\_up=[[i,50],[(i+width/2),height],[i+width,50]]

points.append(triangle\_up)

else:

square=[[i,size[1]],[i+width,size[1]],[i+width,50],[i,50]]

points.append(square)

points.append(square)

לאחר מכן נכתבה הפונקציה שמציירת את פוליגונים על המסך:

def draw\_polygons():

global size,points,screen

for i in range(len(points)):

color = BROWNISH if i<12 or i>13 else GREENISH

pygame.draw.polygon(screen,color,points[i])

אחד האתגרים המרכזיים היה לצייר את מצב הלוח שהוגדר כרשימה של מספרים שלמים מכיוון שנדרשת המרה של רשימה זו לפיקסלים במערך דו מימדי על המסך לשם כך הגדרתי לכל אינדקס ערך על גבי הציר האופקי X :

def convert\_idx\_to\_pixels(idx):

screen\_offsets=[40,120,200,280,360,440,600,680,760,840,920,1000]

if idx < 12: pos = screen\_offsets[idx]

elif idx ==50: pos=520

else: pos = screen\_offsets[23-idx]

return pos

לאחר מכן פונקציה שמציירת את האבנים כערימה:

def draw\_pile(posx,amount,rad,color,idx,size,screen):

for i in range(amount):

if idx<12: pygame.draw.circle(screen,color,(posx,size[1]-rad-i\*2\*rad),rad)

else: pygame.draw.circle(screen,color,(posx,50+rad+i\*2\*rad),rad)

פונקציה שמציירת את האבנים שבכלא:

def draw\_jail():

global jail,screen,player

color=WHITE if player else BLACK

rival\_color=BLACK if player else WHITE

rad=30

for i in range(jail[0]):

if jail[0]>0: pygame.draw.circle(screen,rival\_color,(convert\_idx\_to\_pixels(50),500+i\*2\*rad),rad)

for i in range(jail[1]):

if jail[1]>0: pygame.draw.circle(screen,color,(convert\_idx\_to\_pixels(50),400-rad-i\*2\*rad),rad)

פונקציה שמציירת את הלוח:

def draw\_board():

global board,size,screen,jail,player,status,decide\_begin,dices

screen.fill((255, 200, 20))

pause = pygame.image.load('pause.png')

screen.blit(pause, (10,-10))

draw\_polygons()

color=BLACK if player else WHITE

rival\_color=WHITE if player else BLACK

for i in range(len(board)):

draw\_pile(convert\_idx\_to\_pixels(i),np.abs(board[i]),30, color if board[i]>0 else rival\_color, i,size,screen)

draw\_jail()

dice\_img\_list=['dice1.png','dice2.png','dice3.png','dice4.png','dice5.png','dice6.png']

if decide\_begin==0:

if dices[0]!=-1:

dice1 = pygame.image.load(dice\_img\_list[dices[0]-1])

dice1 = pygame.transform.rotate(dice1, angles[0])

screen.blit(dice1, (100,300))

if dices[1]!=-1:

dice2 = pygame.image.load(dice\_img\_list[dices[1]-1])

dice2 = pygame.transform.rotate(dice2, angles[1])

screen.blit(dice2, (500,300))

if len(dices)==4:

if dices[2]!=-1:

dice1 = pygame.image.load(dice\_img\_list[dices[2]-1])

dice1 = pygame.transform.rotate(dice1, angles[0])

screen.blit(dice1, (100,300))

if dices[3]!=-1:

dice2 = pygame.image.load(dice\_img\_list[dices[3]-1])

dice2 = pygame.transform.rotate(dice2, angles[1])

screen.blit(dice2, (500,300))

מכיוון בתחילת המשחק כל שחקן מטיל קובייה אחת נוסף התנאי:

if decide\_begin==0:

כאשר בתחילת המשחק תנאי זה אינו מתקיים ולשם כך נכתבה פונקצייה ייעודית שמציירת את הקוביות בתחילת המשחק:

def draw\_dices(dices2draw,angles2draw):

dice\_img\_list=['dice1.png','dice2.png','dice3.png','dice4.png','dice5.png','dice6.png']

val=100

for i in range(len(dices2draw)):

dice = pygame.image.load(dice\_img\_list[dices2draw[i]-1])

dice = pygame.transform.rotate(dice, angles2draw[i])

screen.blit(dice, (val,300))

val+=400

pygame.display.update()

פונקציית הטלת הקוביות מוגדרת כרשימה של משתנים אקראים בעלי התפלגות אחידה. לשם כך נעזרתי ב**numpy**:

def roll\_dices():

global status,dices,screen,dice1,dice2,dice\_img\_list,dices\_sound,angles,decide\_begin,angles

pygame.mixer.Sound.play(dices\_sound)

sleep(0.3)

die1=np.random.randint(1,7)

die2=np.random.randint(1,7)

if die1==die2: dices=[die1,die1,die1,die1]

else: dices=[die1,die2]

status[3]=len(dices)

angles=[np.random.randint(1,360),np.random.randint(1,360)]

מכיוון שכאשר שחקן מטיל דאבל, הוא רשאי לשחק 4 פעמים בקוביות שקיבל, רשימת הקוביות היא באורך 4 כאשר מתקבל דאבל. בנוסף פונקציה זו מייצרת עוד שני משתנים אקראיים המייצגים את הזוויות של הקוביות כאשר הן מוצגות על גבי המסך.

פונקצייה נוספת בעלת ערך משמעותי בפרוייקט היא הפונקציה:

is\_point\_inside\_shape()

פונקציה זו מקבלת פוליגון ונקודה על גבי המסך וע"י חישוב של 4 נוסחאות טריגונומטריות קובעת האם נקודה נמצאת בתוך הפוליגון:

def is\_point\_inside\_shape(abc, d):

a,b,c=abc[0],abc[1],abc[2]

area = abs(0.5 \* (a[0]\*(b[1]-c[1]) + b[0]\*(c[1]-a[1]) + c[0]\*(a[1]-b[1])))

area1 = abs(0.5 \* (d[0]\*(b[1]-c[1]) + b[0]\*(c[1]-d[1]) + c[0]\*(d[1]-b[1])))

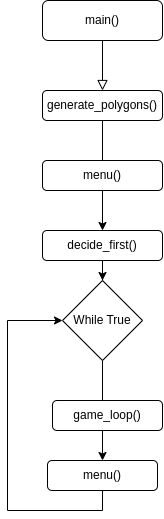
area2 = abs(0.5 \* (a[0]\*(d[1]-c[1]) + d[0]\*(c[1]-a[1]) + c[0]\*(a[1]-d[1])))

area3 = abs(0.5 \* (a[0]\*(b[1]-d[1]) + b[0]\*(d[1]-a[1]) + d[0]\*(a[1]-b[1])))

return area == area1 + area2 + area3

ה**שלב השני** היה הגדרת חוקי המשחק(מנוע המשחק) ויצירת לולאת המשחק . לשם כך נעזרתי בתרשימי זרימה. חלק זה לקח הכי הרבה זמן והיווה את האתגר הכי משמעותי בפרוייקט.

תרשים הזרימה הכללי:



הפונקציות

decide\_first(), game\_loop()

כוללות בתוכן המון פונקציות קטנות שמשתמשות באריטמטיקה ואיטרציות על גבי הרשימות בכדי להגדיר מהלכים תקינים ובעיקר למנוע מהשחקנים לבצע מהלכים לא תקינים .

תרשים הזרימה המפורט גדול מדי בכדי להיכנס בקובץ זה ולכן אצרף אותו בנפרד.

השלב האחרון בבניית הפרוייקט היה יצירת הכפתורים,שילוב הצלילים ובדיקה של המערכת למקרי קיצון.

הוגדרו לשם כך מספר מצבי התחלה:

initial\_board=[-2,0,0,0,0,5,0,3,0,0,0,-5,5,0,0,0,-3,0,-5,0,0,0,0,2]

phase1\_no\_jail=[3,6,4,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,-5,-3,0,-5,-2,0]

all\_block=[2,6,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,-2,-2,-3,-2,-3,-2]

jail\_block=[-2,-6,0,-2,-2,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0,5,3,2]

**סיכום:**

לפרוייקט זה היו מספר תצורות והוא עבר מספר גלגולים. (16 גרסאות שונות שכוללות בתוכן תצורות שונות של הלוח ובחלקן גם שימוש במחלקות) למדתי כיצד פונקציות ומשתנים צריכים להיות מוגדרות היטב על מנת שהמערכת תעבוד בצורה חלקה וכיצד שימוש בתרשימי זרימה יכולים להקל על התהליך.