Programmation - TrickShot - 150 points

Kévin DUVERGER

Table des matières

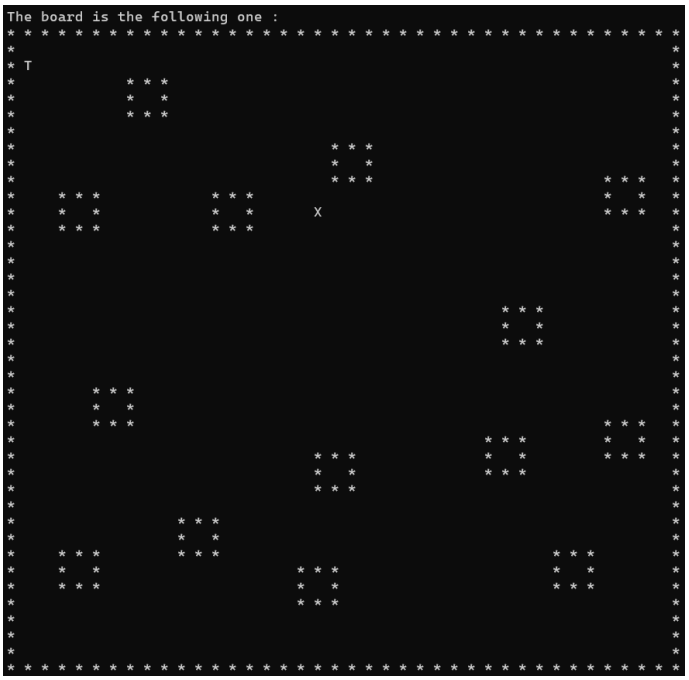
1 Résolution TrickShot :

Nous pouvons commencer en nous connectant au serveur pour voir ce qu'il envoie. La première chose est une présentation du challenge où on nous explique qu'il faut trouver un angle de tir qui permette à `X` de frapper `T` après 9 rebonds (cette valeur ne semble pas changer entre les différentes connexions au serveur) :

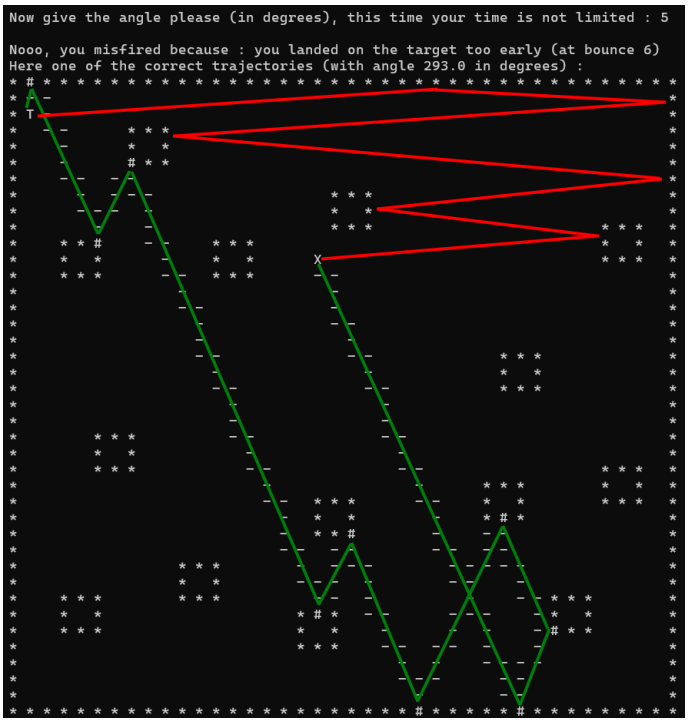
```
PS D:\ALL\Team CRYPTIS\2025\creationCTFInitiation\Programming\TrickShot> ncat 146.59.227.136 23409
libnsock ssl_init_helper(): OpenSSL legacy provider failed to load.

Welcome to TrickShot where we try cool shots !
I will send you a pixelated map, where each cell has size (1, 1) and the top left one is (0, 0) :
        "*" corresponds to a full wall
        " " corresponds to a free space
        "X" corresponds to your tank (at the center of the cell)
        "T" corresponds to the target cell
Your goal is to return an angle at which the tank should fire to hit the target in 9 bounces
The angles start at 0 to the right and are increasing in the anti-clockwise direction
You can consider that the radius of the bullet is 0.05 for your simulation
```

Ensuite nous recevons la carte à utiliser :



Et nous pouvons essayer d'envoyer un angle (probablement faux, c'est la trajectoire en rouge), et le serveur a la gentillesse de nous répondre avec la bonne trajectoire (dessinée en vert) :



Représentation de la carte. La première chose à faire pour résoudre ce challenge est de choisir la représentation de la carte et des différents acteurs (le joueur / la cible) dans le langage de programmation choisi (ici python).
Pour la carte, nous pouvons tout simplement utiliser une liste bi-dimensionnelle de caractères : `'*'` sera un mur, `' '` sera un espace libre, `'X'` sera le joueur et `'T'` sera la cible. La position du joueur et la position de la cible seront également stockées séparément pour ne pas avoir à les rechercher à chaque fois qu'on veut y accéder.
Pour parser la carte, il suffit de splitter l'entrée ligne par ligne, puis de prendre un caractère sur 2 dans chaque ligne (car vous l'avez peut être remarqué il y a un espace entre chaque caractère pour donner un look un peu plus "carré") :

```python
def createBoardFromLineList(stringList) :
    boardSizeY = len(stringList)
    boardSizeX = (len(stringList[0]) + 1) // 2
    board = []
    for y in range(0, boardSizeY) :
        board.append([ ])
        for x in range(0, boardSizeX) :
            board[-1].append(stringList[y][x * 2])
    return board
```

Résolution. Une fois cela fait, l'idée pour résoudre le challenge sera de simuler la trajectoire pour chaque angle entier entre 0 et 360 et prendre le premier qui vérifie la condition de 9 rebonds (nous appellerons p la position du joueur, t la position de la cible et α l'angle en entrée). Pour simuler cette trajectoire, la position de départ de la balle sera $(p.x + 0.5, p.y + 0.5)$ et la direction actuelle sera stockée dans un vecteur initialisé à $(\cos(\alpha), -\sin(\alpha))$ (on a besoin d'un - car l'axe Y est inversé). On va ensuite procéder étape par étape (chaque étape fera avancer la balle de 0.005 unités) et à chaque étape on fera avancer la balle dans la direction stockée dans le vecteur. Si la prochaine position de la balle la fait entrer en collision avec un mur vers le haut / vers le bas il faut inverser sa direction en Y, et si elle la fait entrer en collision avec un murs vers la gauche / vers la droite il faut inverser la direction en X. Voici le code qui ferait cette simulation :

```python
def occupied(x, y, board) :
    return board[y][x] == '*'

def testTrajectory(board, player, target, bounceCount, angle) :
    # Initialisation
    direction = [ math.cos(angle), -math.sin(angle) ]
    currentBounceCount = 0
    currentPosition = [ player[0] + 0.5, player[1] + 0.5 ]
    step = 0.005
    # La boucle
    while True :
        currentPosition = [ currentPosition[0] + step * direction[0], currentPosition[1] + step *
    direction[1] ]
        if [ int(currentPosition[0]), int(currentPosition[1]) ] == target :
            if currentBounceCount == bounceCount :
                return True
            else :
                return False
        # Checking collisions
        oneCollision = False
        if occupied(int(currentPosition[0]), int(currentPosition[1] - 0.05), board) : # The up cell
            if direction[1] < 0 :
                oneCollision = True
                direction[1] *= -1
        if occupied(int(currentPosition[0]), int(currentPosition[1] + 0.05), board) : # The down
    cell
            if direction[1] > 0 :
                oneCollision = True
                direction[1] *= -1
        if occupied(int(currentPosition[0] - 0.05), int(currentPosition[1]), board) : # The left
    cell
            if direction[0] < 0 :
                oneCollision = True
                direction[0] *= -1
        if occupied(int(currentPosition[0] + 0.05), int(currentPosition[1]), board) : # The right
    cell
            if direction[0] > 0 :
                oneCollision = True
                direction[0] *= -1
        # Updating the bounce count
        if oneCollision :
            currentBounceCount += 1
            if currentBounceCount > bounceCount :
                break
    # The result
    return False
```

En plus de tout cela, on va également gérer le nombre de collisions avec les murs et si on arrive sur la cible en le bon nombre on peut retourner que l'on a bien trouvé une trajectoire !

Programme principal. Finalement, il ne reste plus qu'à gérer le programme principal :

```
server = "146.59.227.136"
port = 23409

ma_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    ma_socket.connect((server, port))
except Exception as e:
    print("Problème de connexion", e.args)
    sys.exit(1)

# Getting the board
data = ma_socket.recv(16384).decode("utf-8")
print(data)
lineList = data.split("\n")[11:-1]
board = createBoardFromLineList(lineList)

# Then, finding where the player and the target are
player = [ 0, 0 ]
target = [ 0, 0 ]
for i in range(0, len(board)) :
    for j in range(0, len(board[0])) :
        if board[i][j] == 'T' :
            target = [ j, i ]
        if board[i][j] == 'X' :
            player = [ j, i ]

# Now, we can start finding the angle
firingAngle = 0
for i in range(0, 359) :
    if testTrajectory(board, player, target, 9, math.radians(i)) :
        firingAngle = i
        break

# Sending the result
print("Found firing angle : " + str(firingAngle))
ma_socket.sendall((str(firingAngle) + "\n").encode("utf-8"))
data = ma_socket.recv(8192).decode("utf-8")
print(data)

ma_socket.close()
```