

HALADÓ PROGRAMOZÁS BEDANDÓ

2D játék és perceptron feltanítás

Szklenár Andos Milán

QJVCOT

Kordics Kristóf

ZE27GW

Tartalomjegyzék

Bevezető	2
Irányítás	2
Tanítási adatok generálása.....	3
Pontszam_kalkulator.....	4
Modell feltanítása.....	5
A játék.....	6
Kezdő értékek definiálása	6
Képek és adatok beolvasás	7
Karakterek beolvasása.....	7
Ételek betöltése	8
Pálya képek betöltése	9
Pálya térkép/terv betöltése	10
MI Modell betöltése	11
Az elemek betöltése és értékkadása	11
Alakzatok megrajzolása.....	11
Karakterek rajzolása	11
Pálya rajzolása	12
Élet rajzolása.....	14
Gonosz élet rajzolása	14
Ételek rajzolása	15
Ételek generálása	15
A fő ciklus	16
Étel tesztelése.....	18
Akadály teszt.....	18
Gonosz MI	19
Gonosz élet vesztése	19
Hivatkozások.....	21

Bevezető

Ez a projekt egy 2d játék segítségével bemutatja a perceptron/neurális hálózat működését. Ami 3 részre bontható. [Feltanítási adatok generálása, a modell feltanítása, 2d játék ahol használni lehet a modellt.](#)

Cél: A gonosz elkerülés mivel ha túl közel jön akkor elveszi egy életerőket amit csak étel közelébe lehet megnövelni, ami random keletkezik a pálya egész területén. A gonosz életereje csökkenhető ha sikerül rákattintani.

A projekt futtatásához a következőket kell telepíteni:

Pip install pygame

Pip install pickle

Pip install pandas

Pip install scikit-learn

Irányítás

- W – Előre
- A – Balra
- S – Hátra
- D – Jobbra

Kattintás a gonoszra

Tanítási adatok generálása

Ez az „adatok_gen.py” fájba van megírva. Ami egy csv fájlt fog generálni ahol egy random szituáció generálása és a legjobb lépés kiválasztásával fog történni.

```
with open("adatok.csv", "w", newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['jatekos_dx', 'jatekos_dy', 'kimenet'])

    for i in range(10000):
        gon_x, gon_y = random.uniform(-50, 50), random.uniform(-50, 50)
        kar_x, kar_y = random.uniform(-50, 50), random.uniform(-50, 50)

        legjobb_lepes = pontszam_kalkulator(gon_x, gon_y, kar_x, kar_y)

        # Adatok mentése
        jatekos_dx = kar_x - gon_x
        jatekos_dy = kar_y - gon_y

        writer.writerow([jatekos_dx, jatekos_dy, legjobb_lepes])
```

A ciklusban egy gonosz koordinátát random generálunk (gon_x, gon_y) ami -50 és 50 közötti értéket vehet fel.

A pontszam_kalkulator vissza adja a legjobb lépést (0,1,2,3), és mentjük az adatokat.

Pontszam_kalkulator

```
def pontszam_kalkulator(gon_x, gon_y, kar_x, kar_y):
    lepesek = {
        0: (0, -1), # Fel
        1: (0, 1), # Le
        2: (-1, 0), # Balra
        3: (1, 0) # Jobbra
    }

    legjobb_lepes = -1
    legjobb_pontszam = -float('inf')

    eredeti_tav_jatekos = tavolsag(gon_x, gon_y, kar_x, kar_y)

    #legjobb lpés keresése
    for lepes, (dx, dy) in lepesek.items():
        uj_gon_x, uj_gon_y = gon_x + dx, gon_y + dy

        # Játékos pontszám: A távolság csökkenése a jó
        uj_tav_jatekos = tavolsag(uj_gon_x, uj_gon_y, kar_x, kar_y)
        jatekos_pont = eredeti_tav_jatekos - uj_tav_jatekos

        if jatekos_pont > legjobb_pontszam:
            legjobb_pontszam = jatekos_pont
            legjobb_lepes = lepes
    return legjobb_lepes
```

A for ciklus segítségével iteráljuk a lehetséges lépéseket, amik a *lepesk* könyvtárban vannak, így megkeresve az a lépést ami legközelebb jutott a játékoshoz.

A ciklusban lévő *uj_tav_jatekos* felveszi az adott lépéssel keletkező új távolságot. Majd megkeresve a legkisebb távolságot jelentő lépést, a funkció vissza a legjobb lépést.

Modell feltanítása

```
dataframe = pd.read_csv('adatok.csv')
X = dataframe[['jatekos_dx', 'jatekos_dy']]
Y = dataframe['kimenet'] # Cél (0, 1, 2, vagy 3)

# 2. Adatok szétválasztása
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# 3. Modell létrehozása és tanítása
model = MLPClassifier(hidden_layer_sizes=(4,), max_iter=500, activation='relu', solver='adam')
model.fit(X_train, y_train)

# 4. Kiértékelés
predictions = model.predict(X_test)
print(f"A modell pontossága: {accuracy_score(y_test, predictions) * 100:.2f}%")

# 5. Modell elmentése
with open('gonosz_ai_model.pkl', 'wb') as file:
    pickle.dump(model, file)
print("Modell elmentve: gonosz_ai_model.pkl")
```

A már létrehozott csv file beolvasásával kezdődik a **pandas** csomag segítségével.

A 2. lépésben az adatok szétválasztása történik a **sklearn** beépített funkciója segítségével.

A 3. lépésben létrehozzuk a modellt és fel is tanítjuk, ami 4 neuronos bemenettel rendelkezik, a „relu” aktivációs függvényt használja, és az „adam” nevezetű solver-t.

A 4. lépésben megkapjuk a modell pontosságát.

És a kész modellt kiírjuk egy fájlba.

A játék

Kezdő értékek definiálása

```
pygame.init()

win = pygame.display.set_mode((800, 800))
pygame.display.set_caption("MI Beadandó")

kar_x = 0
kar_y = 0

gon_x = 8
gon_y = 8

elet = 4#maximum 4
gon_elet = 5

mozgas = 50
run = True

bill_ido = time.time()

gonosz_time = time.time()

hp_time = time.time()

etel_ido = time.time()
```

A **pygames** csomagot használva inicializálásra kerül a pygames.

A *kayr_x*, *kary_x* a karakter koordinátái.

A *gon_x*, *gon_y* a gonosz koordinátái.

Az *elet*, a karakter életét tárolja

A *gon_elet*, a gonosz életét tárolja.

A *mozgas*, a mozgás értékét mutatja pixelekben számolva.

A *run* változ segítségével ki lehet lépéni az alkalmazásból.

A *bill_ido* a billentyűzet lenyomot gomb mintavételezéséhez szükséges.

A *gonosz_ido* a gonosz előző lépése óta eltelt időt tartalmazza.

A *hp_time* az előző a gonosz által élet vesztés óta eltelt időt mutatja.

Az *etel_ido* az előző random lefejezett étel óta eltelt időt mutatja.

Képek és adatok beolvasás

Karakterek beolvasása

```
def karakterek_betoltese(path):
    tabla = pygame.image.load(path).convert_alpha()
    tabla_width, tabla_height = tabla.get_size()
    kar_szelesseg = 42
    kar_magassag = 42
    karakterek_ = []
    for y in range(0, tabla_height, kar_magassag):
        for x in range(0, tabla_width, kar_szelesseg):

            rect = pygame.Rect(x, y, kar_szelesseg, kar_magassag)
            image = tabla.subsurface(rect).copy()
            karakterek_.append(pygame.transform.scale(image, (100,100)))
    return karakterek_
```

A funkció egy útvonalat kap paraméterként.

A *table* változó a **pygame** segítségével felveszi a karaktereket táblázatosan tartalmazó képet.(kepek/karakterek.png)

A *kar_szelesseg*, és a *kar_magassag* a karakter képek méretét mutatja pixelben mérve.

Egy egymásba ágyazott for ciklus segítségével végig iteráljuk a *table* szélességét és magasságát a megadott (*kar_szel.*,*kar_mag*) lépés közzel

A *rect* felveszi a karakter négyzetes alakzatát, majd az *image* az alakzat segítségével kivágja a kicsi karakter képét a nagy kép táblázatból.

A *karakterek_* lokális tömbhöz pedig hozzáadjuk a felnagyított karaktert. És a ciklus vissza adja a *karaktere_* tömböt.

Ételek betöltése

```
def etelek_betoltese(path):
    etelek_neve = os.listdir(path)

    eredmeny = []
    for nev in etelek_neve:
        kep = pygame.image.load(f"{path}/{nev}")
        eredmeny.append(pygame.transform.scale(kep, (50,50)))
    return eredmeny
```

A funkció egy útvonalat kap paraméterként.

Az ételek képei külön képekben rendelkeznek a „kepek/etelek” mappában.

Az *etelek_neve* tartalmazza a mappában található összes kép nevét.

Egy for ciklussal iteráljuk a képek neveit. Amivel egyesével beolvassuk a képeket a **pygames** segítségével. Az adott képet pedig tároljuk az *eredmeny*-ek belső tömbben felnagyítva.

Majd a funkció vissza adja az *eredemeny* tömböt.

Pálya képek betöltése

```
def palya_elemk_betoltese(path):
    elem_nevek = os.listdir(path)

    eredmeny = []
    for nev in elem_nevek:
        kep = pygame.image.load(f"{path}/{nev}")

        eredmeny[str(nev).split('.')[0]] = pygame.transform.scale(kep, (50,50))

    return eredmeny
```

A funkció egy útvonalat kap paraméterként.

Az *elem_nevek* fogja tárolni a „kepek/palya” mappában lévő képek neveit.

Egy for ciklussal iteráljuk a képeket.

A **pygames** segítségével betöljtük az adott képet.

Az *eredmeny* nevű lokális könyvtárba tároljuk az adott képet a nevével (file kiterjesztés nélkül) mint kulcs, és maga a felnagyított kép mint érték.

Majd a funkció vissza adja az *eredmeny* könyvtárat.

Pálya térkép/terv betöltése

```
def palya_terv_betoltese(path):
    palya = []

    with open(path, 'r') as file:
        for line in file:

            if(len(line)>2 and line[0]+line[1] != "//"):
                vonal = []
                for char in line:
                    vonal.append(char)
                palya.append(vonal)

    return palya
```

A funkció egy útvonalat kap paraméterként.

A file egy txt file ami a „palya.txt” vagy „palya2.txt”.

Egy for ciklus segítségével iteráljuk a file sorait.

Ha az adott sor hosszabb mint 2 és nem „//” (komment) kezdődik, akkor iteráljuk az adott sor karaktereit és a *vonal* tömbben tároljuk az adott sort karakterenként.

Majd az adott vonal-at tároljuk a *palya* tömbben ami így egy 2dimenziós tömb lesz, reprezentálva a pályát.

MI Modell betöltése

```
with open('gonosz_ai_model.pkl', 'rb') as file:  
    ai_model = pickle.load(file)
```

A **pickle** csomag segítségével beolvassuk a mi modellt.

Az elemek betöltése és értékkadása

```
#kepek meg pálya betöltése  
karakterk = karakterek_betoltese("kepek/karakterek.png")  
palya_elemek = palya_elemek_betoltese("kepek/palya")  
palya = palya_terv_betoltese("palya2.txt")  
elet_kep = [pygame.image.load("kepek/sziv.png").convert_alpha(),pygame.image.load("kepek/ures_sziv.png").convert_alpha(),pygame.image.load("kepek/etelek.png").convert_alpha()]  
etelek = etelek_betoltese("kepek/etelek")
```

Alakzatok megrajzolása

Karakterek rajzolása

```
def karakter Rajz(pos, kari):  
    win.blit(kari, pos)
```

A funkció egy *pos* két elemű tuple és *kari* mint a karakter képe.

A **pygames** beépített blit funkciója megjeleníti a képet.

Pálya rajzolása

```
def palya Rajz(palya_terv, palyaElemek, kar_x,kar_y):
    if(len(palya_terv) > 0):
        elem_x =0
        elem_y = 0

        for y in range(kar_y, len(palya_terv)):
            elem_x = 0
            for x in range(kar_x ,len(palya_terv[y])):
                pos = (elem_x*mozgas,elem_y*mozgas)

                match palya_terv[y][x]:
                    case '#':#Tégla
                        win.blit(palyaElemek["Brick_wall"], pos)

                    case '@':
                        win.blit(palyaElemek["Brick_Wall_Cracked"], pos)
                    case 'H':
                        win.blit(palyaElemek["Wooden_Floor_Horizontal"], pos)
                    case '*':
                        .
                        .
                        .
                    case '5':
                        win.blit(palyaElemek["Grass"], pos)
                        win.blit(palyaElemek["maple"], pos)
                    case '6':
                        win.blit(palyaElemek["Grass"], pos)
                        win.blit(palyaElemek["pine"], pos)
                    case '7':
                        win.blit(palyaElemek["Grass"], pos)
                        win.blit(palyaElemek["sakura"], pos)
                    case '8':
                        win.blit(palyaElemek["Grass"], pos)
                        win.blit(palyaElemek["willow"], pos)
                    elem_x +=1
                elem_y +=1
```

A funkció bemenetként meg kapja a 2d pálya tervet, a pályaelemek könyvtárat, és a karakter pozícióját.

Ez a funkció felelős a mozgás illúziója megteremtéséhez.

Beágyazott for ciklusok segítségével iteráljuk a pálya tervet x és y-nal, a karakter aktuális pozíciójától a pálya végéig. Így amikor a karakter jobbra mozog a ciklus (x=1) akkor a ciklus már az 1 indexű képet jeleníti meg először. Így előre a mozgás hatását.

A pos tuple változóban kiszámoljuk az adott kép ablakon lévő pozícióját. Ahol az elem_x és elem_y a ciklusok futását számolja.

És egy switch segítségével megjelenítjük az adott pályaelem képét az adott pozícióban.

Ami a következő lehet:

- //Komment
- //# - Tégla
- //@ - Törött téglalap
- /* - köves út
- /. - homokos út
- //H - fa padló horizontális
- //W - víz
- //G - fű
- //1 - almafa
- //2 - ciprusfa
- //3 - halott fa
- //4 - gyertya fa
- //5 - juharfa
- //6 - fenyőfa
- //7 - cseresznyefa
- //8 – fűzfa

Élet rajzolása

```
def elet Rajz(kep):
    for i in range(0,4):
        if(i<elet):
            win.blit(kep[0], (i*32,0))
        else:
            win.blit(kep[1], (i*32,0))
```

A funkció egy kép tömböt kap bemenetnek.

A for ciklussal iteráljuk a karakter életét (max 4) és megjelenítjük a megfelelő szívet ami lehet teljes, vagy üres a kép függvényében.

Ebben az esetben ha az *elet* kevesebb mint 4 akkor a fent maradó szívek üresek lesznek.

Ez a képernyő ball felső oldalán lesz megjelenítve.

Gonosz élet rajzolása

```
def gon_elet_Rajz(kep):
    for i in range(0,5):
        if(i<gon_elet):
            win.blit(kep[2], (800-(i*32),0))
        else:
            win.blit(kep[1], (800-(i*32),0))
```

Nagyon hasonlóan történik mint az előző esetben csak a jobb felső sarokba kerülnek a szívek.

Ételek rajzolása

```
def etelek Rajz(le_etelek, kar_x, kar_y):
    for i in range(0, len(le_etelek)):
        x = ((le_etelek[i][0][0]+8)-kar_x)*mozgas
        y = ((le_etelek[i][0][1]+8)-kar_y)*mozgas
        win.blit[le_etelek[i][1], (x,y)]
```

A ciklus egy *le_etelek* mint le rakott ételek tuple tömböt kap, ami tartalmazza a már el helyezett ételeknek képét és pozícióját amit a [etel_generalas](#) állít elő.

Egy for ciklussal iteráljuk az elhelyezett ételeket, és meghatározzuk a képernyőn lévő pozíciójukat a mozgás miatt.

Az x és y az adott étel pozíciója + 8 (16/2 mint a képernyőn látható 16*16 kép mátrix)-karakter pozíció*mozgás

Majd rajzolás.

Ételek generálása

```
def etel_generalas(palya, le_etelek, etel_kepek):
    x = random.randrange(0, len(palya[0]))
    y = random.randrange(0, len(palya))

    le_etelek.append((x,y), etel_kepek[random.randrange(0, len(etel_kepek))]))
```

A funkció a pályát, a le helyezett ételek tuple tömböt és az étel képeket kapja paraméternek.

Az x és y egy random számot (koordinátát) vesz fel a pálya méretein belül.

Majd a lerakott ételek tömbhöz hozzá adjuk egy random képet, a random pozícióhoz.

A fő ciklus

```
while run:
    pygame.time.delay(10)

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

        if event.type == MY_TIMER_EVENT:
            run = False
            pygame.time.set_timer(MY_TIMER_EVENT, 0)

        if event.type == pygame.MOUSEBUTTONDOWN:
            if pygame.mouse.get_pressed()[0]:
                killhim()

    keys = pygame.key.get_pressed()

    if((time.time()-bill_ido) >0.1):

        if keys[pygame.K_a] and akadaly_teszt(palya, max(kar_x-1, 0), kar_y, '#'):
            kar_x -= 1
            kar_x = max(kar_x, 0)

        if keys[pygame.K_d]and akadaly_teszt(palya, max(kar_x+1, 0), kar_y, '#'):
            kar_x += 1

        if keys[pygame.K_w]and akadaly_teszt(palya, kar_x, max(kar_y-1, 0), '#'):
            kar_y -= 1
            kar_y = max(0, kar_y)

        if keys[pygame.K_s]and akadaly_teszt(palya, kar_x, max(kar_y+1, 0), '#'):
            kar_y += 1
        bill_ido = time.time()
```

Egy while ciklus vezérli az eseménykét.

Egy 10ms késleltetés így elérve a 100fps(hz)

Egy for ciklussal iteráljuk a keletkező eseményeket. És ha az adott esemény a kilépés (ablak bezárás) akkor a ciklus megszakad. Ha egér lenyomás keletkezik akkor a [killhim](#) funkció kerül meghívásra.

A keys tárolja a lenyomott billentyűt. Majd ha az előző mozgást irányító billentyű óta eltelt idő nagyobb mint 0.1 másodperc (100ms) akkor megvizsgáljuk a lenyomott billentyűt és az [akadaly_teszt](#) funkció segítségével ellenőrizzük hogy a lépés végrehajtható.

```

if((time.time()-gonosz_time)>0.05):
    gonosz_ai()
    gonosz_time = time.time()

if((time.time()-hp_time)>0.5):
    hp_time = time.time()
    if(kar_x < gon_x + 1 and kar_x > gon_x - 1) and (kar_y < gon_y + 1 and kar_y > gon_y - 1):
        elet = max(0, elet-1)

if((time.time()-etel_ido)>2):
    etel_generalas(palya, lerakott_etelek, etelek)
    etel_ido = time.time()

```

Ha az előző gonosz lépés óta eltelt idő nagyobb mint 0.05 másodperc, akkor a gonosz egy új lépést hajt végre a [gonosz_ai](#)-val.

Ha az előző gonosz által okozott életvesztés óta eltelt idő nagyobb mint 0.5 másodperc, és a gonosz ugyanazon a koordinátán van mint a karakter akkor élet vesztés következik be.

Ha az előző random étel generálása óta eltelt idő nagyobb mint 2 másodperc akkor [etel_generalas](#) segítségével lefejezünk egy új ételt.

```

kaja_teszt(kar_x, kar_y, lerakott_etelek)

win.fill((255, 255, 255))
palya Rajz(palya, palya_elemek, kar_x, kar_y)
karakter Rajz((400 -50, 400-50), karakterek[0])

#gonosz
if gon_elet > 1:
    karakter Rajz(gonosz_mozgatas(kar_x, kar_y, gon_x, gon_y), karakterek[19])

#etelek
etelek Rajz(lerakott_etelek, kar_x, kar_y)

elet Rajz(elet_kep)
gon_elet Rajz(elet_kep)

pygame.display.update()

```

A [kaja_teszt](#) által növeljük a karakter életét ha ugyan ott van mint egy étel.

A `win.fill` funkcióval a képernyőt fehérre töltjük ki.

És a játék elemke rajzolása történik.

Étel tesztelése

```
def kaja_teszt(kar_x,kar_y, le_etelk):
    rmv = None
    global elet
    for i in range(0, len(le_etelk)):
        act = le_etelk[i]
        if(kar_x == act[0][0] and kar_y == act[0][1]):
            elet +=1

            rmv = act
            break
    if(rmv != None):
        le_etelk.remove(rmv)
```

A funkció a karakter pozícióját, és az lehelyezett ételeket kapja meg.

Egy for ciklussal iteráljuk az össze lehelyezett ételt és ha ugyan ott van mint a karakter akkor növeljük a karakter életét. És az *rmv* foglya tárolni a törlendő ételt.

Akadály teszt

```
def akadaly_teszt(palya_terv, kar_x, kar_y, akadaj):
    x = kar_x+8
    y = kar_y+8

    if(y < len(palya_terv)):
        if(x < len(palya_terv[y])):
            if(palya_terv[y][x] == akadaj):
                return False
    return True
```

A funkció a pályát a karakter pozícióját és az akadály karakterét kapja bemenetnek.

Ha a *kar_x* és *y* koordinátán lévő elem ugyan az mint az akadály akkor a funkció Hamis értéket ad vissza.

Gonosz MI

```
def gonosz_ai():
    global gon_x, gon_y

    játékos_dx = kar_x - gon_x
    játékos_dy = kar_y - gon_y

    bemenet = [[játékos_dx, játékos_dy]]
    teszt = pd.DataFrame(bemenet, columns=['jatekos_dx', 'jatekos_dy'])

    josolt_lepes = ai_model.predict(teszt)[0]

    sebesseg = 0.5 # A gonosz sebessége
    if josolt_lepes == 0: # Fel
        gon_y -= sebesseg
    elif josolt_lepes == 1: # Le
        gon_y += sebesseg
    elif josolt_lepes == 2: # Balra
        gon_x -= sebesseg
    elif josolt_lepes == 3: # Jobbra
        gon_x += sebesseg
```

A *játékos_dx* és *dy* felveszi a gonosz és karakter közti távolságot.

A *bemenet* és *teszt* megfelelő tömb helyezi.

A *josolt_lepes* tartalmazza a modell által jóslott lépést.

Majd megtörténik a gonosz mozgatása.

Gonosz élet vesztése

```
def killhim():
    global gon_elet
    mx, my = pygame.mouse.get_pos()
    dx = ((gon_x - kar_x) * 50) + 400
    dy = ((gon_y - kar_y) * 50) + 400
    if (mx < dx + 25 and mx > dx - 25) and (my < dy + 25 and my > dy - 25):
        gon_elet -= 1
        print("bumm hes deads")
    else:
        print("krasne ráno")
```

Ha a kattintáskori egér pozíciója ugyan ott van mint a gonosz a képernyőn, akkor a gonosz étele csökken.

Nyerés/Vesztés

Ha a játékos élete nullára csökken a játék bedob egy game over képernyőt majd kilép. Hasonlóképpen ha a gonosz élete nullára csökken akkor kiírja hogy nyertél majd kilép.

```
if elet == 0 and not timer_started:
    game_over_status = "lose"
    pygame.time.set_timer(MY_TIMER_EVENT, 3000)
    timer_started = True

if gon_elet == 0 and not timer_started:
    game_over_status = "win"
    pygame.time.set_timer(MY_TIMER_EVENT, 3000)
    timer_started = True

if game_over_status == "lose":
    youreloser = pygame.transform.scale(pygame.image.load("kepek/loser.jpeg"), (800,800))
    win.blit(youreloser,(0,0))

if game_over_status == "win":
    yourewinner = pygame.transform.scale(pygame.image.load("kepek/winner.jpeg"), (800,800))
    win.blit(yourewinner,(0,0))
```

Ha bármelyik élete eléri a nullát el indít egy időzítőt ami 3 másodperc múlva elküld egy eventet. Ezt az eventet a fő ciklus az elején elkapja és kilép

```
while run:
    pygame.time.delay(10)

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

        if event.type == MY_TIMER_EVENT:
            run = False
            pygame.time.set_timer(MY_TIMER_EVENT, 0)

        if event.type == pygame.MOUSEBUTTONDOWN:
            if pygame.mouse.get_pressed()[0]:
                killhim()
```

Hivatkozások

pygame: <https://www.pygame.org/docs>

pandas: <https://pandas.pydata.org/docs>

pickle: <https://docs.python.org/3/library/pickle.html>

sklearn: <https://scikit-learn.org/stable>