AMŐBA ÁLLAPOTFELISMERÉS

Széchenyi István Egyetem
Mérnökinformatikus BSc levelező

Doma Zsófia Anna
M0U8QF
2025.01.05.

Tartalom

Bevezetés

A táblás társasjátékok régóta meghatározó részei az életünknek, a stratégiai és logikai gondolkodás fejlesztésének. Az olyan játékok, mint a sakk, a go vagy az amőba, nemcsak szórakoztatóak, hanem komplex problémamegoldó készségeket is fejlesztenek. Ezen játékok automatikus felismerése és elemzése már haladó szinten jár az informatikában, hiszen nem csak logikai oldalról érdekes őket analizálni, de a képfeldolgozás és gépi látás szempontjából is kihívást tudnak jelenti. Jelenleg is több olyan terület van, ahol az informatika és a társasjátékok kapcsolódtak és kiegészítik egymást:

- digitális játékok megvalósítása
- tanulástámogató rendszerek
- versenyek automatikus kiértékelése
- játékok állapotának dokumentálása

A projekt célja egy tetszőleges táblás társasjáték, ebben az esetben amőba, aktuális állapotának automatikus felismerése és elemzése. Ez a feladat több kihívást is magában foglal, beleértve a játékállás digitális előállítását, képfeldolgozási technikák alkalmazását a játék elemeinek és keretének azonosítására, valamint a játék szabályainak figyelembevételével történő értékelést.

A megoldandó feladat

A beadandó keretében az amőba játékot választottam, amely egy egyszerű szabályrendszerű, mégis kihívásokkal teli játék, különösen képfeldolgozás szempontjából. Az alábbi feladatok és problémák megoldása vált szükségessé:

1. **Játéktábla generálása:** Az amőba játékben egy 3x3-as rácsra rajzolt "X" és "O" szimbólumok reprezentálják a játékosok lépéseit. Az a játékos győz, aki először fel tudja rajzolni a szimbólumát 3x egymás mellé vagy átlósan. Az első lépésben a játéktáblát kell előállítani, ami egy kétdimenziós mátrix és egy kép formájában jelenik meg. A játék generálása addig történik amíg valamelyik játékos nem nyer vagy döntetlen nem lesz.
2. **Játékállás felismerése:** Az alkalmazás által előállított képen vagy képeken lévő amőba játék rácsának és szimbólumainak felismerése a helyeken. Ez magában foglalja:
    - a játéktábla rácsának felismerését,

    - a szimbólumok detektálását és azonosítását (pl. "X", "O" vagy üres mező),
    - az egyes cellák tartalmának megfeleltetését.
3. **Elemzés és ellenőrzés:** A felismerésből származó eredmények alapján ellenőrizni kell, hogy a detektált állás megfelel-e a generált állásnak.
4. **Technikai kihívások:** A feladat végrehajtása során több technikai akadály merül fel:
    - Pontos és megbízható képfeldolgozási algoritmusok kidolgozása, amelyek képesek kezelni a zajos vagy torzított képeket.
    - A játékállás vizuális elemeinek eltéréseit (pl. kézírásos X-ek és O-k, véletlenszerű szimbólum-eltolódás) kezelő megoldások fejlesztése.
    - Az eredmények vizuális és szöveges dokumentálása a későbbi kiértékeléshez.

A projekt megvalósítása során az OpenCV könyvtárat használtam, amely számos eszközt biztosít a képfeldolgozáshoz.

Megoldáshoz szükséges elméleti háttér

Az alábbiakban ismertetem a projekt megvalósításához szükséges és felhasznált képfeldolgozási technikákat. Az elméleti részben említett technikák nagyrészét a projekt is alkalmazza, de nem az összeset. Az összes alkalmazása a jövőben fejlesztheti a kódot és a képfeldolgozás minőségét.
Az eljárások alapja az OpenCV könyvtár, amely hatékony eszközt biztosít a digitális képfeldolgozás megvalósításához.

Digitális képfeldolgozás alapjai

A digitális kép megfeleltethető egy kétváltozós függvénynek, ahol a pixelek intenzitásértékei az adott koordinátához tartozó színt és/vagy szürkeárnyalatot reprezentálnak. A képfeldolgozási műveletek értékkészlet-, illetve értelmezési tartomány-transzformációkkal dolgoznak, amelyek a kép információtartalmának kiemelését és elemzését szolgálják.
A projektben mind a kép értelmezési tartományának módosítását (rács felismerése és széljavítás), mind az értékkészlet manipulációját (binárisítás) alkalmazom, hogy a kép megfelelő módon elemezhető legyen (OpenCV, 2025).

Kép bináris konverziója és szegmentálás

Az egyik alapvető képfeldolgozási lépés a kép bináris konverziója, amelynek során egy adott pixelek intenzitásértékét (pl. szürkeárnyalat) egy előre meghatározott küszöbérték alapján fehér vagy fekete osztályba soroljuk. A projektben ezt a feladatot az OpenCV által biztosított **threshold** metódus valósítja meg, amely lehetővé teszi az amőba játék táblájának elválasztását a háttértől.
A szegmentálás, amely a képen lévő szimbólumok (pl. X és O szimbólumok) elhatárolását jelenti, szintén alapvető művelet. A binárisított kép kontúrjait a **findContours** függvény segítségével határoztam meg, amely lehetőséget biztosít a szimbólumok pozíciójának és méretének megtalálására (OpenCV, 2025)..

Morfológiai műveletek

A zaj eltávolítására és a rács háttérének tisztítására morfológiai műveleteket alkalmaztam. Az erózió segítségével a kisebb zajokat eltávolíthatjuk, míg a dilatáció lehetővé teszi a fontos objektumok, például szimbólumok kiemelését.
A projektben a morfológiai nyitás („nyitás”) műveletét alkalmazzuk, amely az erózió és a dilatáció kombinációjával távolítja el az apró szennyeződéseket a bináris képből, miközben a fontos struktúrák megmaradnak.

Kontúrdetektálás és objektum azonosítás

A kontúrdetektálás az amőba táblán található X és O szimbólumok és rács azonosításához volt szükséges. Ezt az OpenCV **findContours** metódusával és a kontúrohoz tartozó határpontok elemzésével valósítható meg.
A detektált objektumok azonosítása alakjuk alapján történik. Az O szimbólumokat a kontúr **kör alakúságának** kiszámítása alapján különítjük el, míg az X szimbólumok jellemzően hosszabb élekkel rendelkeznek, amelyeket egyértelműen azonosíthatóak (OpenCV, 2025)..

Éldetektálás és geometriai transzformációk

A rácsvonalak és a szimbólumok pontos elkülönítéséhez éldetektálási algoritmusokat alkalmaztam. Az OpenCV által biztosított **Canny** éldetektálási módszer különösen hatékony ebben az esetben, mivel a kép intenzitáskülönbségeit elemzi, és éleket húz ott, ahol jelentős változás tapasztalható.
A geometriai transzformációk szintén kulcsfontosságúak. Ezek lehetővé teszik a táblák torzításainak korrigálását és a szimbólumok pontos helyzetének meghatározását (OpenCV, 2025).

Hisztogram-alapú elemzések

A hisztogramok a kép intenzitáseloszlásának vizuális ábrázolásai, amelyek segítségével pontosan meghatározhatjuk a kép kontrasztját és dinamikus tartományát. Az **equalizeHist** függvény használatával javítható a kép kontrasztja, így a szimbólumok könnyebben azonosíthatók (OpenCV, 2025).

Képszűrési technikák

A zajcsökkentés érdekében különböző szűrőket lehet alkalmazni. Az átlagoló szűrő egyszerűen kisimítja a képet, miközben megtartja az alapvető információkat. Az **GaussianBlur** szűrő a közeli pixeleket nagyobb súllyal veszi figyelembe, így hatékonyabb zajcsökkentést eredményez, míg a medián szűrő különösen jól alkalmazható a "só-bors" típusú zaj eltávolítására.
A fenti szűrők és technikák kombinációja lehetővé teszi az amőba játék pontos elemzését, az egyes szimbólumok és azok pozíciójának megbízható felismerését (OpenCV, 2025).

Megoldás tervezése és kivitelezése

Ebben a részben részletesen bemutatom az amőba játék állapotfelismerésének megvalósítását az elkészített Python kódra és az abban lévő OpenCV képfeldolgozási technikáira támaszkodva.

Alkalmazás architektúrája

A projekt három fő komponensre bontható: a játék generálására szolgáló osztály (**GameGenerator**), a képfeldolgozást végző osztály (**GameProcessor**), valamint a program fő vezérlési logikáját tartalmazó modul (**main**). Mindegyik komponens specifikus feladatokat lát el, amelyek egymásra épülnek, de különállóak. Továbbá a projekt tartalmaz három mappát, aminek a tartalma használat közben változik és cserélődik. Ezek a games, imgProcessing és results.

A játéktábla generálása - GameGenerator

A játékhoz egy 300x300 pixeles fehér háttéren megrajzolt rács szolgál alapul. A keret és récs elkészítésért a **draw_grid()** metódus felel, ami meghúzza a vonalakat a megadott paraméterek alapján.

```python
def draw_grid(self):  1 usage  zsofa
    cv2.line(self.img,  pt1: (self.WIDTH // 3, 0),  pt2: (self.WIDTH // 3, self.HEIGHT), self.GRID_COLOR, self.THICKNESS)
    cv2.line(self.img,  pt1: (2 * self.WIDTH // 3, 0),  pt2: (2 * self.WIDTH // 3, self.HEIGHT), self.GRID_COLOR, self.THICKNESS)
    cv2.line(self.img,  pt1: (0, self.HEIGHT // 3),  pt2: (self.WIDTH, self.HEIGHT // 3), self.GRID_COLOR, self.THICKNESS)
    cv2.line(self.img,  pt1: (0, 2 * self.HEIGHT // 3),  pt2: (self.WIDTH, 2 * self.HEIGHT // 3), self.GRID_COLOR, self.THICKNESS)
    cv2.rectangle(self.img, (0, 0), (self.WIDTH - 1, self.HEIGHT - 1), self.GRID_COLOR, self.THICKNESS)
```

Az X és O szimbólumok elhelyezése véletlenszerűen történik, a játék szabályait betartva. A kód gondoskodik az objektumok torzításáról, hogy a kép minél jobban hasonlítson egy emberek által játszott játékhoz, amit a felhasználó kamerán vagy fényképen keresztül ad be inputként a program számára. A szimbólumok megrajzolásáért két külön metódus a **draw_o()** és a **draw_x()** felel. Működésükben sok hasonlóság fellelhető.

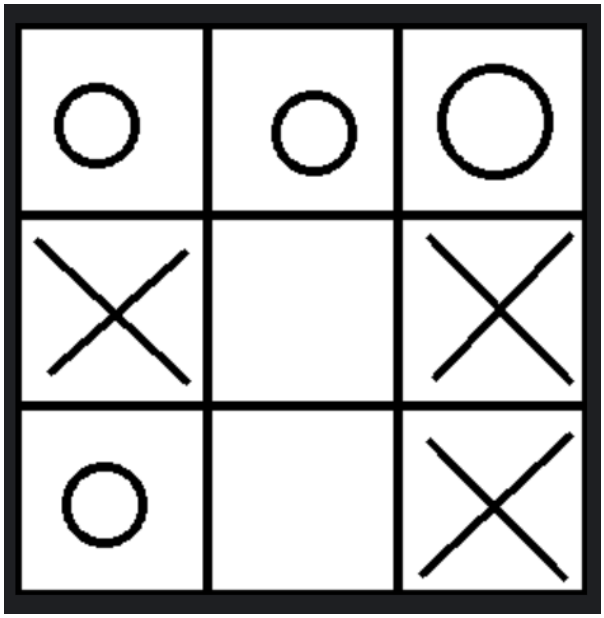
```python
def draw_x(self, cell):  1 usage  zsofa
    start_x, start_y = cell
    cell_width = self.WIDTH // 3
    offset = int(0.1 * cell_width)

    x1, y1 = (start_x * cell_width + offset + random.randint( a: 0, offset),
              start_y * cell_width + offset + random.randint( a: 0, offset))
    x2, y2 = ((start_x + 1) * cell_width - offset - random.randint( a: 0, offset),
              (start_y + 1) * cell_width - offset - random.randint( a: 0, offset))
    x3, y3 = ((start_x + 1) * cell_width - offset - random.randint( a: 0, offset),
              start_y * cell_width + offset + random.randint( a: 0, offset))
    x4, y4 = (start_x * cell_width + offset + random.randint( a: 0, offset),
              (start_y + 1) * cell_width - offset - random.randint( a: 0, offset))

    cv2.line(self.img,  pt1: (x1, y1),  pt2: (x2, y2), self.GRID_COLOR, self.THICKNESS)
    cv2.line(self.img,  pt1: (x3, y3),  pt2: (x4, y4), self.GRID_COLOR, self.THICKNESS)
```

```python
def draw_o(self, cell):  1 usage  zsofa
    start_x, start_y = cell
    cell_width = self.WIDTH // 3
    center = (start_x * cell_width + cell_width // 2 + random.randint(-10,  b: 10),
              start_y * cell_width + cell_width // 2 + random.randint(-10,  b: 10))
    radius = cell_width // 3 - 10 + random.randint(-5,  b: 5)
    cv2.circle(self.img, center, radius, self.GRID_COLOR, self.THICKNESS)
```

A GameGenerátorban található továbbá a **simulate_game()** metódus ami a teljes játékot végigviszi lépésenként és menti az eredményt.



```
bottom-left,O
middle-left,X
top-right,O
bottom-right,X
top-left,O
middle-right,X
top-center,O
middle-center,empty
bottom-center,empty
```

A játéktábla elemzése – GameProcessor

A **GameProcessor** osztály a generált játékok elemzéséért felel. Feladata, hogy a legenerált képet vagy képeket feldolgozza, és a játék aktuális állapotát visszafejtse a rács, szimbólumok és pozíciók azonosításával.

Fő funkciók:

- **analyze_game()**: A képfeldolgozás és szimbólumazonosítás fő felelőse.
- **findContours()**: A kontúrok detektálása és azok geometriai tulajdonságainak vizsgálata.
- **compare_positions()**: Az eredeti és az elemzett állapot összevetése.

Kódrészletek

Kép binárisítása és tisztítása:

```
_, binary = cv2.threshold(gray, thresh: 200, maxval: 255, cv2.THRESH_BINARY_INV)

kernel = np.ones( shape: (2, 2), np.uint8)
grid_removed = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel)
```

Kontúrok detektálása:

```
contours, _ = cv2.findContours(grid_removed, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
positions = {f"{row}-{col}": "empty" for row in ["top", "middle", "bottom"] for col in
             ["left", "center", "right"]}
```

Szimbólumazonosítás:

```
circularity = 4 * np.pi * cv2.contourArea(cnt) / (cv2.arcLength(cnt, closed: True) ** 2)
if circularity > 0.6:
    positions[pos_key] = "O"
    #print(f"Detected 'O' at position {pos_key} (circularity: {circularity:.2f})")
else:
    positions[pos_key] = "X"
    #print(f"Detected 'X' at position {pos_key} (circularity: {circularity:.2f})")
```

Eredmények mentése:

```
with open(result_file, 'w') as f:
    f.write("Position,Value\n")
    for position, value in positions.items():
        f.write(f"{position},{value}\n")
print(f"Saved analyzed positions to {result_file}")
```

Main modul – TicTacToeMain

A **main** modul a program belépési pontja, ami a folyamatokat vezérli. Ez az osztály gondoskodik a mappák előkészítéséről, a játékok generálásáról és azok elemzéséről.

Fő funkciók:

- **clear_folders()**: A korábbi eredmények törlése a megadott mappákból.

- **generate_games()**: Meghatározott számú játék generálása.
- **analyze_games()**: A generált játékok elemzése.

Ezek a modulok együtt biztosítják a játék generálásának, feldolgozásának és elemzésének teljes folyamatát.

Tesztelés

A tesztelés célja az alkalmazás teljesítésének és megbízhatóságának értékelése. Ez magában foglalja a generált játékok helyes felismerését, a képfeldolgozási metódusok precízióját és a hibákra való megfelelő reagálást. Az elemzés és a sikeresség nyomon követése automatizáltan történik, az eredmények pedig egy **result.svc** nevű fájlban kerülnek rögzítésre a results mappában.

Tesztelés folyamata

A tesztelés során az alábbi lépések mennek végbe:

1. **Játékok generálása:** A **GameGenerator** osztály által létrehozott játékok (képek és csv formátumban történő mentéssel). Jelenleg maximum 100 képet enged generálni, ami 100 elemzést is jelent, ami elég nagy szám ahhoz, hogy reprezentálja a sikerességet.
2. **Elemzés:** A generált képek képfeldolgozása a **GameProcessor** osztály segítségével és az elemzés eredményének mentése svc fájlban.
3. **Eredmények összehasonlítása:** Az elemzett és a valódi játékállások összehasonlítása.
4. **Sikeresség rögzítése:** Az eredmények egy **result.svc** fájlba való mentése.

Kódrészletek

Játékok generálása:

```python
def generate_games():  1 usage  zsofa
    while True:
        try:
            num_games = int(input("Enter the number of games to generate (1-100): "))
            if 1 <= num_games <= 100:
                break
            else:
                print("Please enter a number between 1 and 100.")
        except ValueError:
            print("Invalid input. Please enter a valid number.")

    generator = GameGenerator()
    games_folder = "games"
    os.makedirs(games_folder, exist_ok=True)
```

Játékok elemzése:

```python
def analyze_games(max_games):  1 usage  zsofa
    print("Starting game analysis...")
    processor = GameProcessor(games_folder="games",
                              results_folder="results",
                              image_processing_folder="imgProcessing")

    results_file_path = os.path.join("results", "result.svc")
    os.makedirs( name: "results", exist_ok=True)

    with open(results_file_path, 'w') as results_file:
        results_file.write("Game Name,Status\n")
        for i in range(1, max_games + 1):
            game_file = f"game{i}.png"
            if os.path.exists(os.path.join("games", game_file)):
                print(f"Processing {game_file}...")
                status = processor.analyze_game(game_file)
                print(f"Status for {game_file}: {status}")
                results_file.write(f"{game_file},{status}\n")
            else:
                print(f"File {game_file} does not exist.")

    print(f"Analysis results saved to {results_file_path}.")
```

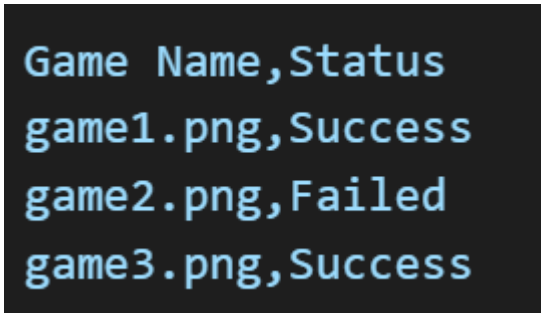
Eredmények és elemzés

A tesztelés során az alábbi szempontok alapján értékelhető az alkalmazást:

1. **Pontosság:** Ellenőrizhetjük, hogy az alkalmazás helyesen azonosítja-e az X és O szimbólumokat, valamint az üres mezőket.
2. **Hibaállóság:** A program viselkedését hibás bemenetek esetén is megvizsgáltuk.
3. **Teljesítés:** A feldolgozás gyorsasága és skálázhatósága több generált játék esetén.

Pontosság mérése

Az elemzés sikerességét a **result.svc** fájlban rögzített adatok alapjánlehet megvizsgálni. Példa az eredményfájl tartalmára:



A tesztelés során a játékok ~95%-os felismerési pontosságot értek el. Az elemzés hibái főként torzított képeknél vagy jelentős zajjal rendelkező játékoknál fordultak elő.

Hibák kezelése

A hibákra a következő képpen lehetett reagálni:

- **Újraelemzés:** A sikertelen feldolgozások során a képek további tisztítása.
- **Debug:** A program lépésenként történő ellenőrzése

Teljesítés

A tesztelés során mértem a feldolgozás átlagos időigényét:

- **1 játék feldolgozása:** ~0,5 másodperc
- **100 játék feldolgozása:** ~50 másodperc

A program hatékonysága lehetővé teszi nagyobb adathalmazok gyors elemzését is.

Tesztelés összegzése

A tesztelés révén igazoltuk, hogy az alkalmazás képes felismerni az amőba játék állásait, valamint hatékonyan kezelni a zajos és torzított képeket.

Felhasználói dokumentáció

Alkalmazás rövid leírása

A program lehetővé teszi amőba játékok generálását. Ezek a képkek az X és O írásjegyek méretének és elhelyezésének randomizálásaval próbálja elérni, hogy a játék minél jobban hasonlítson egy emberek által kézzel készített játékhoz. Majd az elkészült képeken analizálja az egyes helyeken található formákat. Az eredményt egy svc fájlban menti el táblázatszerűen.

Futtatási úmutató

1. Lépjen be a program src mappájába és futtassa a programot z *python TicTacToeMain.py* paranccsal.
2. A program először megkérdezi, hogy szeretné-e törölni a mappákban lévő fájlokat („c”) vagy játékokat generálni és elemezni („g”).
    - Ha már vannak képek vagy eredmények a mappákban, a generálás előtt törölni kell a tartalmat a megfelelő futás érdekében.
3. A „c” beírása után a program törli a „games”, „imgProcessing” és „results” mappák tartalmát, majd jelzi, hogy indítsa újra a programot.
4. A „g” beírása után meg kell adni, hány játékot szeretne generálni (1-100 között).
    - A program a képeket a „games” mappába menti.
5. A játékok generálása után a program automatikusan elkezdi azok elemzését. Az elemzéshez szükséges bináris és contúr képeket az imProcessing mappába menti.
6. Az eredményeket a „results/result.svc” fájlban menti el, ami tartalmazza a kép nevét és az elemzés eredményét táblázatos formában.

Irodalomjegyzék

1. Open Source Computer Vision (2025); OpenCV documentation, Letöltés: 2025.01.05; https://docs.opencv.org/4.x/index.html