MATURITNÍ PRÁCE

Aplikace neuronových sítí (AI)

André Schuhmacher

Vedoucí práce: Tomáš Havrda

Studijní program: IT Technik pro výrobní systémy

Studijní obor: 26-41-M/01

2024

Odevzdáním této maturitní práce na téma Aplikace neuronových sítí (AI) potvrzuji, že jsem ji vypracoval pod vedením vedoucího práce samostatně za použití v práci uvedených pramenů a literatury. Dále potvrzuji, že tato práce nebyla využita k získání jiného nebo stejného titulu.

V Mladé Boleslavi dne 31. 03. 2024

Poděkování

Rád bych poděkoval svému vedoucímu práce, panu Tomášovi Havrdovi, za jeho cenné vedení a odborné rady. Jeho ochota naslouchat mým nápadům byla velkou oporou při tvoření této maturitní práce.

ABSTRAKT

Tato práce se zaměřuje na využití neuronových sítí (AI) k automatizaci procesu počítání bodů ve hře šipky. Cílem je navrhnout a implementovat systém s umělou inteligencí, který by sledoval skóre a usnadnil hru pro všechny hráče. Nejprve je proveden sběr dat a trénink neuronové sítě, která detekuje dopad šipek na terči. Sběr dat zahrnuje anotaci obrázků a jejich přípravu pro trénink. Pro trénink sítě je využita platforma Google Colab, která poskytuje přístup k výkonnému hardwaru zdarma. Implementace zahrnuje integraci kamerového systému, včetně osvětlení a kalibrace kamery pro přesné sledování dopadové plochy. Backendová část aplikace určuje polohu šipek na terči a vyhodnocuje hodnoty hozených šipek. Výsledky jsou ukládány do databáze a zobrazovány na uživatelském rozhraní, které je navrženo pro jednoduché ovládání zejména pomocí mobilních zařízení. Tento systém má potenciál zefektivnit a zjednodušit proces hry, přinášející uživatelům novou úroveň pohodlí a přesnosti ve sledování jejich výkonu.

KLÍČOVÁ SLOVA

Neuronová síť, GUI, automatizace, šipky, umělá inteligence

1 Obsah

Úvod

Tato maturitní práce zkoumá možnosti využití neuronových sítí (AI) pro zefektivnění a zjednodušení procesu počítání bodů ve hře šipky. Cílem práce je navrhnout a implementovat systém s umělou inteligencí, který by dokázal automaticky sledovat skóre a usnadnit tak hru pro všechny účastníky.

V šipkách se snaží jeden nebo více hráčů dosáhnout nulového skóre odečítáním bodů od 501. Toho dosahují házením šipek na terč, rozdělený do segmentů s různou hodnotou. Hra končí v okamžiku, kdy jeden z hráčů dosáhne nulového skóre prostřednictvím hození šipky do vnějšího segmentu s dvojitou hodnotou. Například, pokud zbývá 20 bodů, hráč musí dosáhnout dvojité 10 pro ukončení hry.

Vyhodnocování bodů v této maturitní práci se realizuje prostřednictvím kamery, která je zaměřena na terč. Tato kamera je připojena k počítači, na kterém běží software, který aktivně sleduje a zaznamenává dopad šipek. Tento proces probíhá pomocí sledování změn na povrchu terče a monitorování zvukových úrovní. Jakmile šipka dopadne na terč, začne systém provádět vyhodnocování a přenáší získané hodnoty do databáze. Tyto údaje jsou následně získávány webovým serverem, který je zodpovědný za zobrazení dat na uživatelském rozhraní.

2 Sběr dat a trénink neuronové sítě

Trénink neuronové sítě je jako učení se z vlastních chyb. Síť dostane velké množství dat, například obrázky psaných písmen, objektů nebo třeba i nahrávky řeči. S těmito daty se snaží naučit vzory a souvislosti.

Představte si to tak, že síť dostane obrázek písmene A a zkusí ho napodobit. Porovná svůj výstup se skutečným písmenem A a na základě rozdílu upraví své vnitřní nastavení. Tento proces probíhá pro různé typy dat znovu a znovu.

Během tréninku se síť snaží minimalizovat chybu mezi svými výstupy a skutečnými hodnotami. Je to jako ladění nástroje - čím víc síť trénuje, tím přesnější se stávají její predikce.

Snímání dopadové plochy

Prvním krokem je důkladně zvážit možné metody detekce šipky, která dopadla na kamerovou plochu. Zvolil jsem systém, který využívá mikrofon kamery a samotnou kameru, software čeká na náhlé zvýšení zvukové úrovně z mikrofonu, tedy nějakou abnormalitu od normální zvukové úrovně. Normální zvukovou úrověn v pokoji jsem změřil na -50dB, po hození zhruba 10 šipek jsem dedukoval že zásah šipky způsobí zvýšení o +25dB. Následně fotka se šipkou projde trénovaným modelem který vyhodnotí, zda se na fotce nachází šipka či ne.

```cpp
int paCallback(const void* inputBuffer, void* outputBuffer,
    unsigned long framesPerBuffer,
    const PaStreamCallbackTimeInfo* timeInfo,
    PaStreamCallbackFlags statusFlags,
    void* userData) {


    float* input = (float*)inputBuffer;
    long double power = 0.0;

    // calc power of audio signal
    for (unsigned long i = 0; i < framesPerBuffer; i++) {
        power += static_cast<long double>(input[i]) * static_cast<long double>(input[i]);
    }

    // normalization calculation
    power /= framesPerBuffer;

    // power to decibel conversion
    long double power_dB = 10 * log10l(power);

    currentAudioLevel = power_dB;

    return paContinue;
}
```

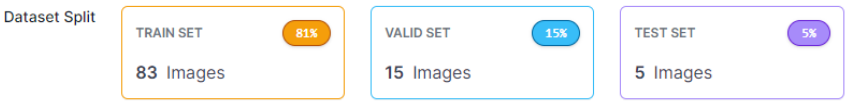
Obr. 1 Ukázka kalkulace decibelové úrovně mikrofonu

2.1 Anotace a příprava tréninkových dat

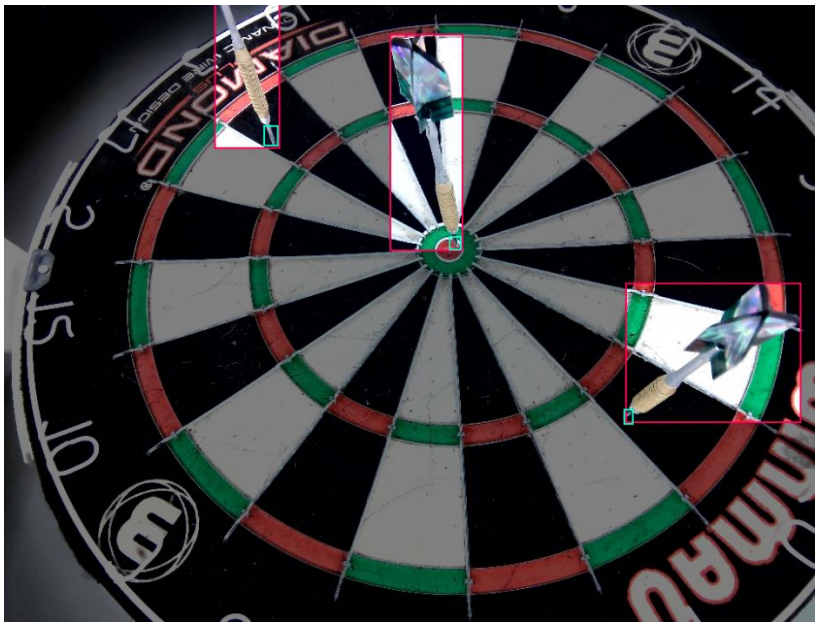
Anotace představuje proces přípravy obrázků pro trénink neuronové sítě. Pro anotaci jsem se rozhodl využít platformu Roboflow, která nabízí jednoduchý přístup k této přípravě. Nahraji obrázky a provedu anotaci kde pomocí obdélníků označím plochu objektů, které má model detekovat, v mém případě jsou to šipky a hroty šipek. Například vytvořím 300 obrázků pro trénink, kde každý obsahuje kombinaci tří šipek vržených různým způsobem do terče. Na každém obrázku musím označit všechny šipky a hroty všech šipek. Poté mi Roboflow sestaví tréninkový balíček, který je složen z 80% obrázků pro trénink, 15% pro validaci a 5% pro testování.

Name	Date modified	Type	Size
test	10.03.2024 15:37	File folder	
train	10.03.2024 15:37	File folder	
valid	10.03.2024 15:37	File folder	
README.dataset.txt	10.03.2024 15:37	Text Source File	1 KB
README.roboflow.txt	10.03.2024 15:37	Text Source File	1 KB
data.yaml	10.03.2024 15:37	Yaml Source File	1 KB

Obr. 2 Ukázka trénovacího balíčku



Obr. 3 Rozložení trénovacího balíčku



Obr. 4 Ukázka anotace

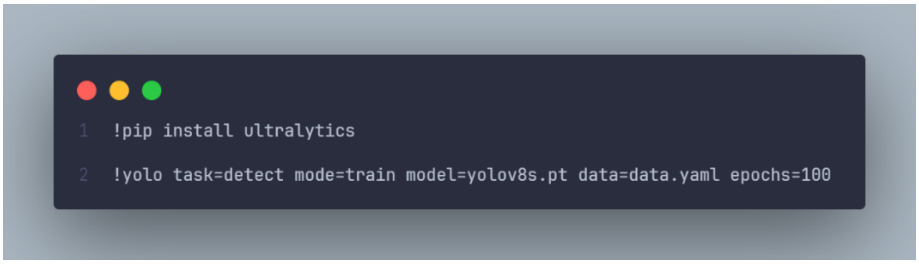
2.2 Naučení neuronové sítě

Učení neuronové sítě je sice relativně jednoduché, avšak náročné zejména na čas a hardware. Proto jsem objevil platformu, která mi umožňuje využít výkonný hardware, který je obvykle drahý a pro většinu lidí, včetně studentů jako jsem já, nedostupný. Tato platforma se nazývá Google Colab, je zdarma dostupná a poskytuje bezplatný přístup např. k grafické kartě NVIDIA Tesla T4 která je určená pro trénování neuronových sítí, nebo k jednotce zpracování tensorů (TPU), která může být až 2/3x rychlejší než GPU. (Trénování s CPU je v dnešní době zastaralé, jelikož je nesmírně pomalé a neefektivní.)

Jako neuronovou síť a model jsem vybral YOLOv8 díky jeho dostupnosti, jednoduchosti a obsáhlé dokumentaci.

Pro trénování neuronové sítě je potřeba stažený základní model YOLOv8, tréninkový balíček vytvořený v předešlé kapitole a nějaký určený počet epoch.

Termín "epocha" odkazuje na jeden kompletní průchod přes tréninkový balíček. Ideálně bych chtěl trénovat přibližně 100 epoch dle potřeby.



Obr. 5 Příkazy na trénování modelu YoloV8

Na obr. 6 můžeme sledovat průběh tréninku, ke každé epoše, máme tyto parametry.

Ztráta obdelníku (box_loss)	Jak dobře umí model najít místo, kde se objekty nacházejí na obrázku. (Čím menší číslo tim lepší)
Ztráta třídy (cls_loss)	Jak dobře umí model klasifikovat co je daný objekt za třídu. (Čím menší číslo tim lepší)
Distribuční fokální ztráta (dfl_loss)	Nevyváženost tříd při trénování. (Čím menší číslo tím lepší)

Tab. 1 Parametry tréninku

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      1/100      2.58G      2.84      4.006      2.259         42        640: 100% 6/6 [00:05<00:00,  1.15it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:01<00:00,  1.35s/it]
                   all         15         82   0.000709     0.0244   0.000757    0.00017

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      2/100      2.57G      2.011      3.637       1.73         28        640: 100% 6/6 [00:00<00:00, 10.09it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  9.03it/s]
                   all         15         82    0.00725      0.305     0.0507     0.0294

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      3/100      2.57G      1.63      2.438      1.367         31        640: 100% 6/6 [00:00<00:00,  7.88it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  2.64it/s]
                   all         15         82    0.00959       0.39       0.15     0.0943

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      4/100      2.57G      1.54      1.961      1.271         18        640: 100% 6/6 [00:00<00:00,  9.66it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  4.84it/s]
                   all         15         82          1     0.0463      0.283      0.173

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      5/100      2.57G      1.368      1.677      1.191         28        640: 100% 6/6 [00:00<00:00,  9.61it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 1/1 [00:00<00:00,  2.91it/s]
                   all         15         82      0.947      0.103      0.346      0.222
```

Obr. 6 Průběh tréninku

3 Implementace a testování

3.1 Integrace kamerového systému

3.1.1 Osvětlení

Setkával jsem se problémy, když dopadla šipka na terč, udělala stín, nebo nebylo vidět na hrot, protože se barva hrotu shodovala a díky tomu model měl problém s predikcí a musel být pokaždé trénován v jiných světelných podmínkách. Proto jsem zvolil metodu osvětlení pomocí světelného kruhu nad terčem, kde tím pádem budou stále stejné světelné podmínky.

„stíny na snímcích s vysokým rozlišením nejen zhoršují radiometrické informace, ale také narušují interpretaci snímku.“ (Luo, Li, & Shen, 2020) [přeloženo z anglického originálu]

Obr. 7 Ukázka terče s osvětlením

3.1.2 Kalibrace

Kamera je umístěna na pravé horní částí kruhu , cca. 20 cm daleko od terče, aby mohla snímat celý terč. Je možnost, že se kamera pohne, takže jsem musel přidat nějakou formu kalibrace. Zvolil jsem metodu perspektivní transformace do stejné roviny jako je rovná fotka terče, na kterou budou předem kalibrované souřadnice pro

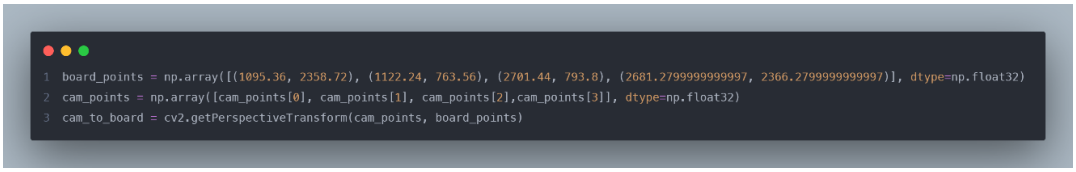
bodové segmenty. Rovná fotka je focená na mobilním telefonu s 4K kamerou. Na fotce jsem zvolil 4 kalibrační body, našel jsem si pixelové hodnoty těchto kalibračních bodů, a uložil je. Na fotce z web-kamery umístěné před terčem zvolím tyto stejné body, a pomocí knihovny OpenCV změním perspektivu, orientaci a roviny této fotky, aby byly stejné jako na fotce z mobilního telefonu.

Obr. 8 Indikace kalibračních bodů

Obr. 9 Fotka před kalibrací

Obr. 10 Fotka po kalibraci

Kalibrační kód je v podstatě velice primitivní, předem určené body z rovné fotky a vybrané body z kamery použijeme jako parametry ve funkci perspektivní transformace. Výsledná proměnná obsahuje data pro kalkulaci pixelových souřadnic ze staré fotky na novou.



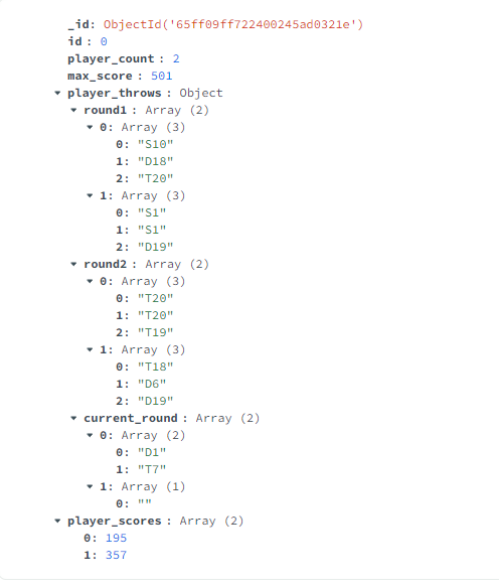
```
board_points = np.array([(1095.36, 2358.72), (1122.24, 763.56), (2701.44, 793.8), (2681.2799999999997, 2366.2799999999997)], dtype=np.float32)
cam_points = np.array([cam_points[0], cam_points[1], cam_points[2],cam_points[3]], dtype=np.float32)
cam_to_board = cv2.getPerspectiveTransform(cam_points, board_points)
```

Obr. 11 Kód pro kalibraci obrazu

3.2 Zprovoznění uživatelského rozhraní

3.2.1 Databáze

Jako komunikační tunel mezi backendem[1] a frontendem[2] jsem zvolil databázový systém MongoDB, je to nerelační dokumentová databáze, jejž úložiště funguje na podobném principu jako systém JSON. Rozhodl jsem se pro zatím relativně jednoduchý systém, který spočívá ve 2 kolekcích, jedna kolekce je aktivní hra, uvnitř se nachází dokument se strukturou, kterou můžeme vidět na obr. 12.



```
_id: ObjectId('65ff09ff722400245ad0321e')
id : 0
player_count : 2
max_score : 501
player_throws : Object
  round1 : Array (2)
    0: Array (3)
      0: "S10"
      1: "D18"
      2: "T20"
    1: Array (3)
      0: "S1"
      1: "S1"
      2: "D19"
  round2 : Array (2)
    0: Array (3)
      0: "T20"
      1: "T20"
      2: "T19"
    1: Array (3)
      0: "T18"
      1: "D6"
      2: "D19"
  current_round : Array (2)
    0: Array (2)
      0: "D1"
      1: "T7"
    1: Array (1)
      0: ""
player_scores : Array (2)
  0: 195
  1: 357
```

Obr. 12 Struktura aktivní hry

[1] Funkční kód, který běží na pozadí, uživatel ho nevidí
[2] Kód, který se stará o vzhled, uživatel ho vidí a interaguje s ním

Po skončení hry se dokument aktivní hry přesune do kolekce s historií her a přidá se mu vlastnost výherce, aby se v budoucnosti uživatel mohl podívat na svoje předešlé hry. Tato kolekce může také sloužit jako základní kámen pro budoucí implementaci statistik, hráčských profilů a vlastně nekonečně jiných možností. Ale prozatím slouží jako sběr dat o hráčských výkonech, či se zhoršil nebo zlepšil, nejvíc trefovaný segment a třeba poměr výher a proher.

3.2.2 Backend

Backendová část softwaru slouží primárně k vyhodnocování hodnot hozených šipek do terče. Nejdříve je potřeba nahrát natrénovaný model, a následně spustit predikci.

```
model = YOLO("best.pt")
results = model.predict(source=im1, save=True, conf=0.4)
```

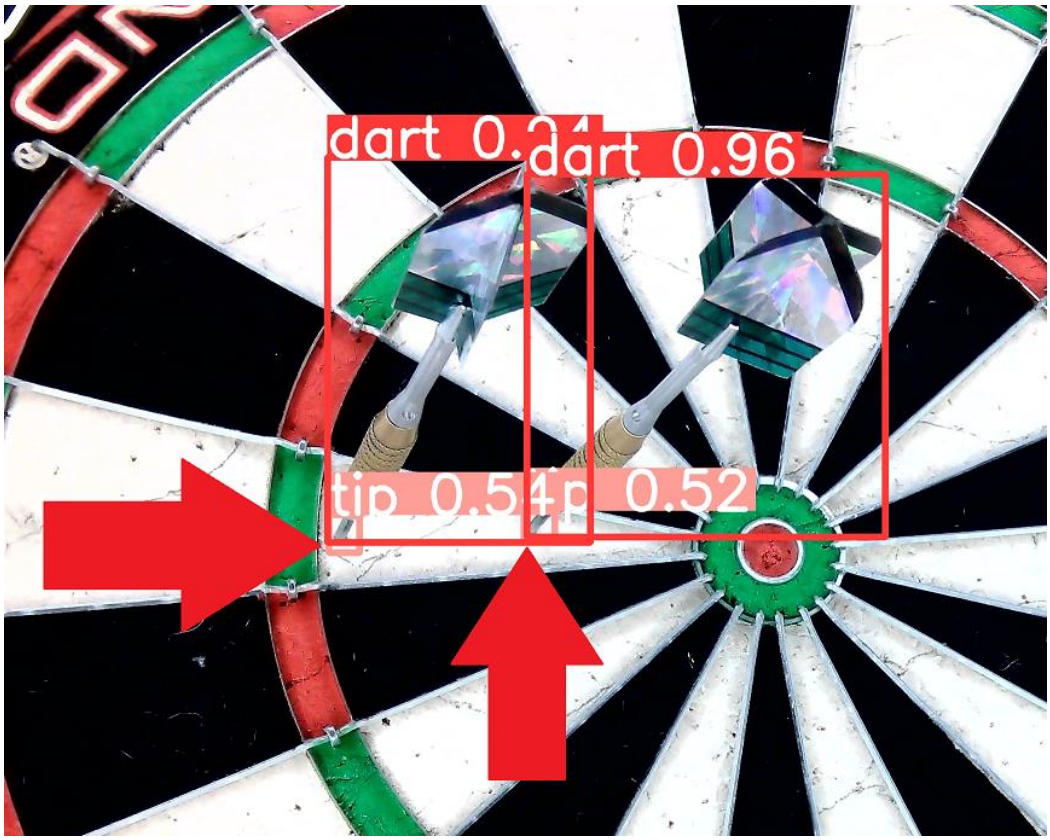
Obr. 13 Zvolení modelu a spuštění predikce

Náš cíl v této části kódu, je nalezení dvojice šipky a hrot dané šipky, abychom byly v další části schopni vypočítat do jakého segmentu hrot šipky přesně dopadl.

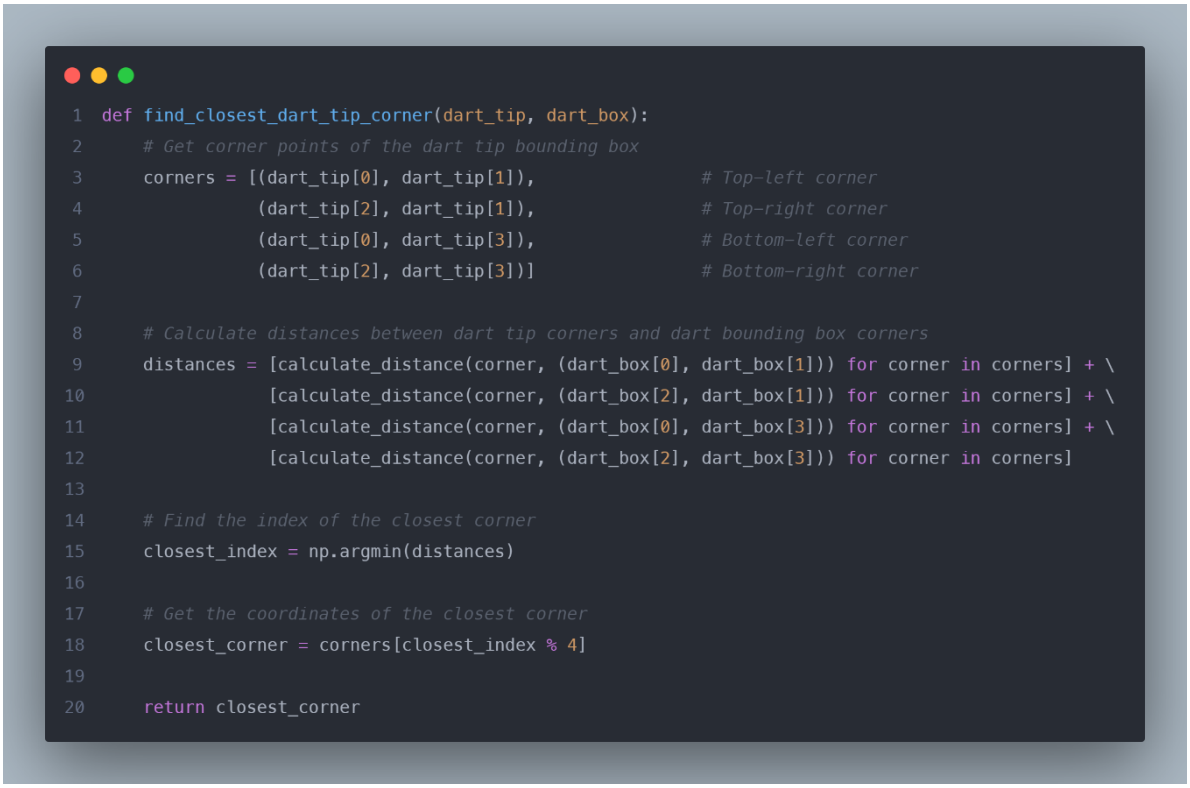
```
def find_dart_tip_pairs(bounding_boxes, classes, min_overlap=0.3):
    pairs = []

    # Get indices of dart tips and darts
    dart_tip_indices = [i for i, cls in enumerate(classes) if cls == 1]
    dart_indices = [i for i, cls in enumerate(classes) if cls == 0]

    # Compare each dart tip with darts
    for dart_tip_index in dart_tip_indices:
        for dart_index in dart_indices:
            overlap_ratio = calculate_overlap(bounding_boxes[dart_tip_index], bounding_boxes[dart_index])
            if overlap_ratio >= min_overlap:
                pairs.append((dart_tip_index, dart_index))

    return pairs
```

Obr. 14 Funkce na nalezení dvojice šipky a hrotem šipky

Po nalezení dvojic, musíme vyhodnotit kde přesně hrot šipky dopadl, učiníme to následujícím způsobem. Víme, že roh obdélníku, který označuje hrot bude vždy nejblíže k rohu obdélníku, který označuje celou šipky, tím pádem můžeme předpokládat že tento roh bude bod kde je hrot zapíchnutý do terče.

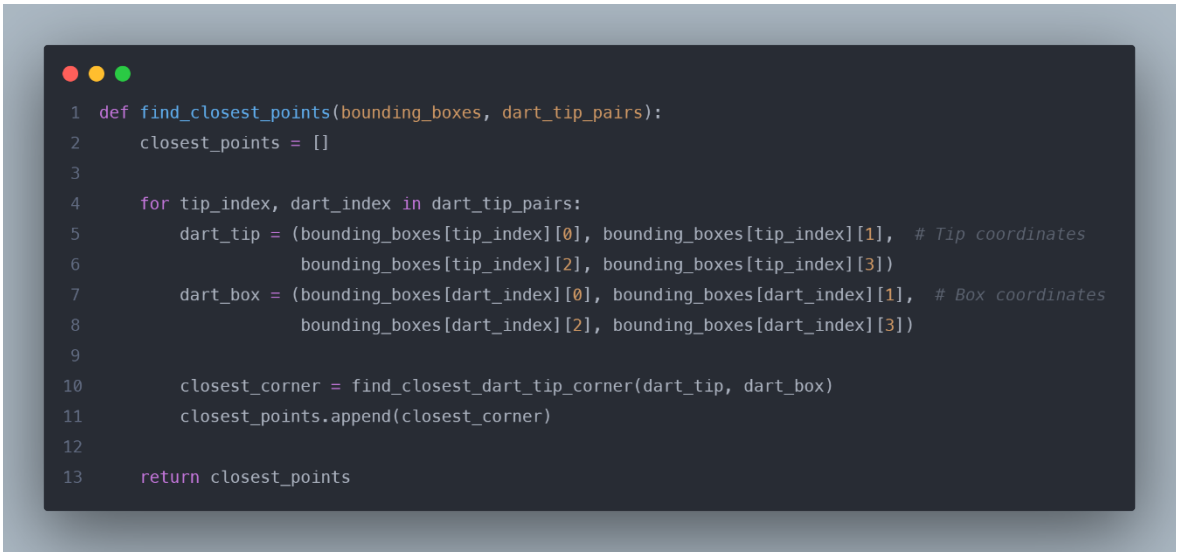


Obr. 15 Ukázka lokace zapíchnutého hrotu



```python
def find_closest_dart_tip_corner(dart_tip, dart_box):
    # Get corner points of the dart tip bounding box
    corners = [(dart_tip[0], dart_tip[1]),               # Top-left corner
               (dart_tip[2], dart_tip[1]),               # Top-right corner
               (dart_tip[0], dart_tip[3]),               # Bottom-left corner
               (dart_tip[2], dart_tip[3])]               # Bottom-right corner

    # Calculate distances between dart tip corners and dart bounding box corners
    distances = [calculate_distance(corner, (dart_box[0], dart_box[1])) for corner in corners] + \
                [calculate_distance(corner, (dart_box[2], dart_box[1])) for corner in corners] + \
                [calculate_distance(corner, (dart_box[0], dart_box[3])) for corner in corners] + \
                [calculate_distance(corner, (dart_box[2], dart_box[3])) for corner in corners]

    # Find the index of the closest corner
    closest_index = np.argmin(distances)

    # Get the coordinates of the closest corner
    closest_corner = corners[closest_index % 4]

    return closest_corner
```

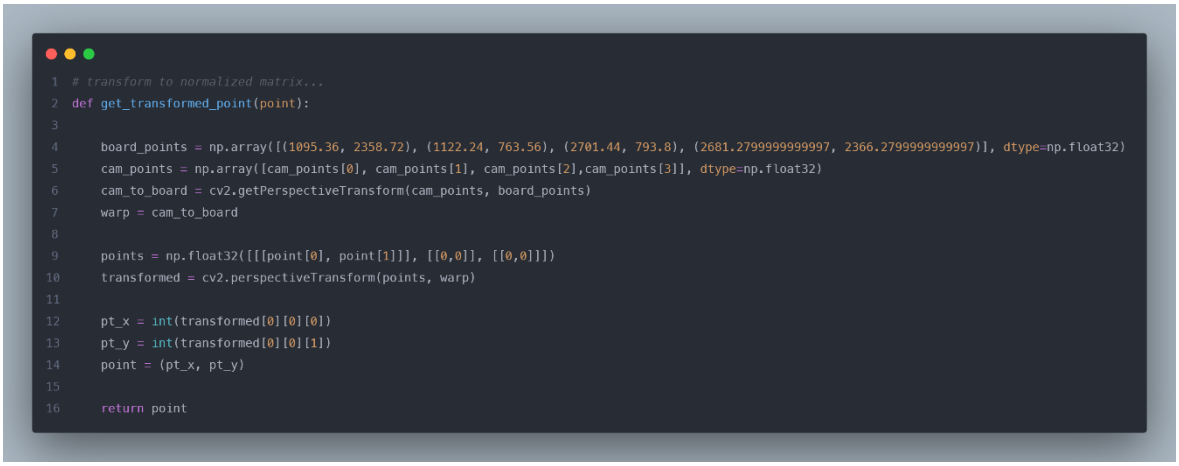
Obr. 16 Funkce pro nalezení nejbližšího rohu obdélníku hrotu k rohu obdélníku šipky



```python
def find_closest_points(bounding_boxes, dart_tip_pairs):
    closest_points = []

    for tip_index, dart_index in dart_tip_pairs:
        dart_tip = (bounding_boxes[tip_index][0], bounding_boxes[tip_index][1],  # Tip coordinates
                    bounding_boxes[tip_index][2], bounding_boxes[tip_index][3])
        dart_box = (bounding_boxes[dart_index][0], bounding_boxes[dart_index][1],  # Box coordinates
                    bounding_boxes[dart_index][2], bounding_boxes[dart_index][3])

        closest_corner = find_closest_dart_tip_corner(dart_tip, dart_box)
        closest_points.append(closest_corner)

    return closest_points
```

Obr. 17 Funkce pro nalezení nejbližších bodů

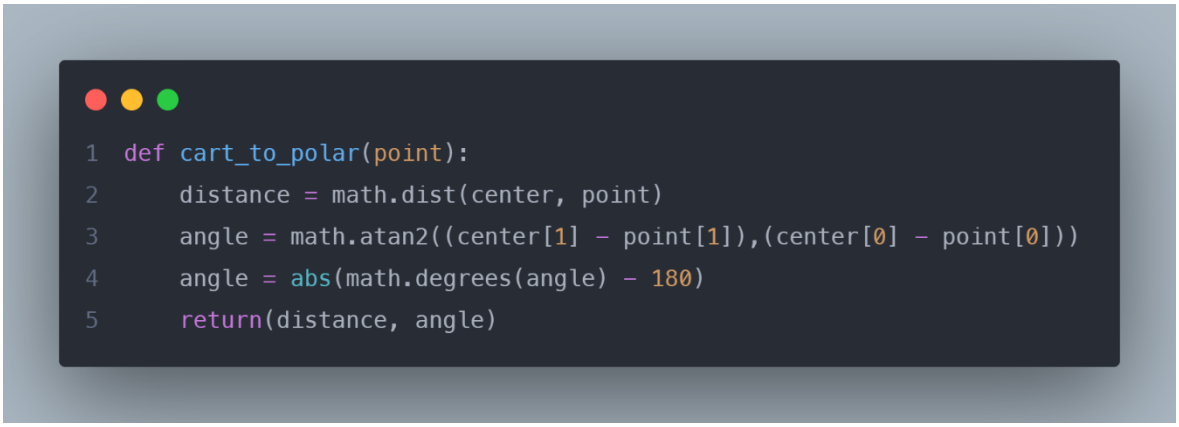
Po nalezení této pixelové souřadnice ji musíme převést na souřadnice které by odpovídali kalibrované fotce, to dokážeme pomocí transformace.



```python
# transform to normalized matrix...
def get_transformed_point(point):

    board_points = np.array([(1095.36, 2358.72), (1122.24, 763.56), (2701.44, 793.8), (2681.2799999999997, 2366.2799999999997)], dtype=np.float32)
    cam_points = np.array([cam_points[0], cam_points[1], cam_points[2],cam_points[3]], dtype=np.float32)
    cam_to_board = cv2.getPerspectiveTransform(cam_points, board_points)
    warp = cam_to_board

    points = np.float32([[[point[0], point[1]]], [[0,0]], [[0,0]]])
    transformed = cv2.perspectiveTransform(points, warp)

    pt_x = int(transformed[0][0][0])
    pt_y = int(transformed[0][0][1])
    point = (pt_x, pt_y)

    return point
```

Obr. 18 Převedení normálových souřadnic na kalibrované

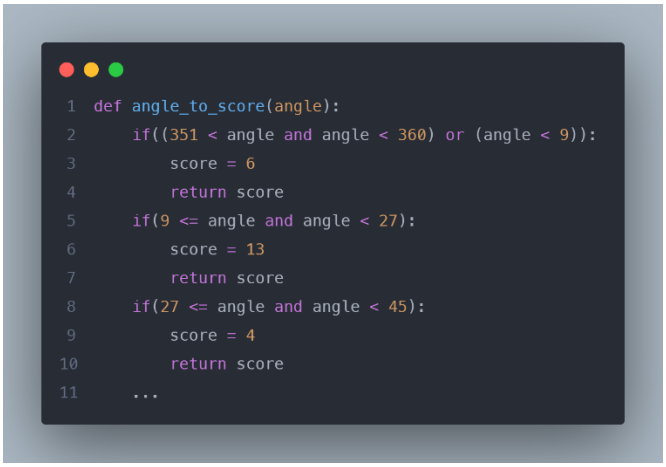
Po získání kalibrovaných souřadnic už nezbývá nic jiného než vypočtení, kolik bodů je na dané souřadnici. K tomu jsem využil převedení kartézské souřadnice na polární. Hlavní rozdíl mezi kartézskými souřadnicemi a polárními je, že kartézské fungují ve dvou, nebo trojrozměrném poli (x,y) nebo (x,y,z) a polární fungují ve dvourozměrném poli, kde polohu bodu určují vzdálenost od počátku souřadnic a úhel, který daný bod svírá s pevnou osou. Na rozdíl od kartézských souřadnic, které vyžadují specifikaci vzdáleností od os až k bodu v každém rozměru, polární souřadnice kombinují vzdálenost a úhel.



```python
def cart_to_polar(point):
    distance = math.dist(center, point)
    angle = math.atan2((center[1] - point[1]),(center[0] - point[0]))
    angle = abs(math.degrees(angle) - 180)
    return(distance, angle)
```

Obr. 19 Funkce k převedení kartézské souřadnice na polární

Rozdělení segmentů využívá pevně zakódované hodnoty, které odpovídají rovné fotce z mobilního telefonu, která se využívá ke kalibraci. Jsou na to dvě funkce, jedna rozděluje segmenty radiálně podle úhlů, a druhá podle vzdálenosti od středu.



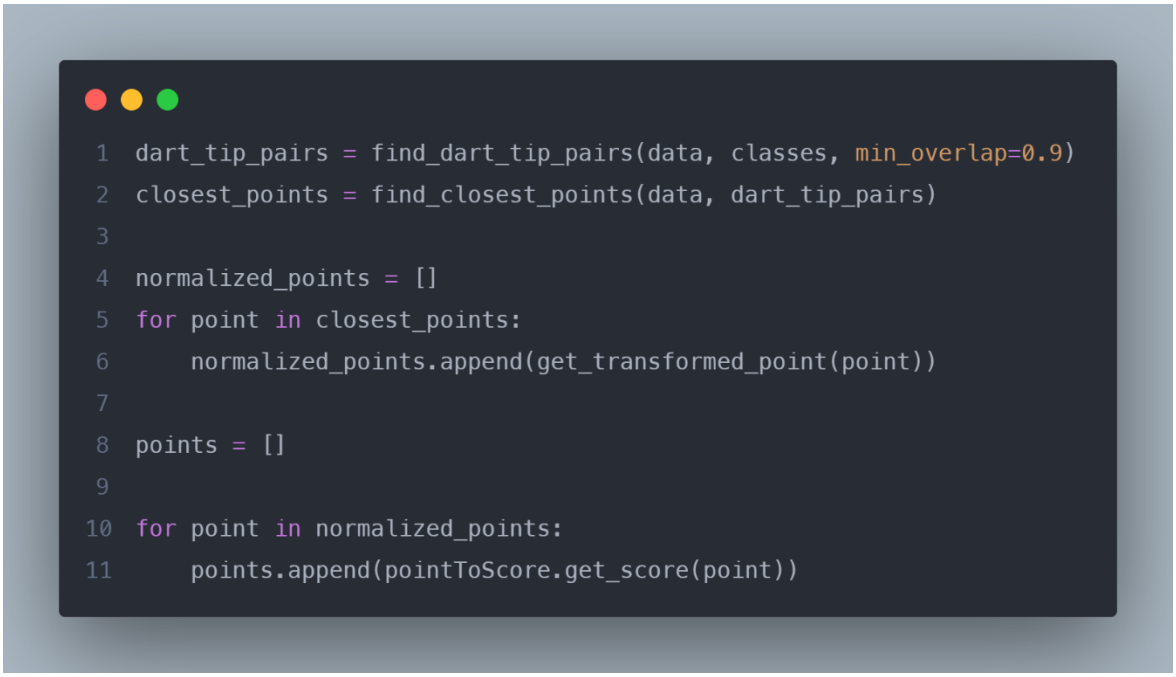
```python
def angle_to_score(angle):
    if((351 < angle and angle < 360) or (angle < 9)):
        score = 6
        return score
    if(9 <= angle and angle < 27):
        score = 13
        return score
    if(27 <= angle and angle < 45):
        score = 4
        return score
    ...
```

```python
def get_score(point):
    polar = cart_to_polar(point)
    print(polar)
    dist = polar[0]
    print(dist)
    wedge = angle_to_score(polar[1])

    if(0<=dist<=44):
        ring = "be_50"
        score =  50
        return(ring, score)

    if(44<dist<=110):
        ring = "be_25"
        score = 25
        return(ring, score)

    if(111<dist<=627):
        ring = "single_1"
        score = wedge
        return(ring, score)
    ...
```

Obr. 20 Funkce pro rozdělení segmentů podle úhlu

Obr. 21 Funkce pro rozdělení segmentů podle vzdálenosti

Nyní už stačí pouze všechno zkombinovat, a výsledné hodnoty se uloží do proměnné points.



```python
dart_tip_pairs = find_dart_tip_pairs(data, classes, min_overlap=0.9)
closest_points = find_closest_points(data, dart_tip_pairs)

normalized_points = []
for point in closest_points:
    normalized_points.append(get_transformed_point(point))

points = []

for point in normalized_points:
    points.append(pointToScore.get_score(point))
```

Obr. 22 Použití funkcí zmíněných v kapitole

3.2.3 Frontend

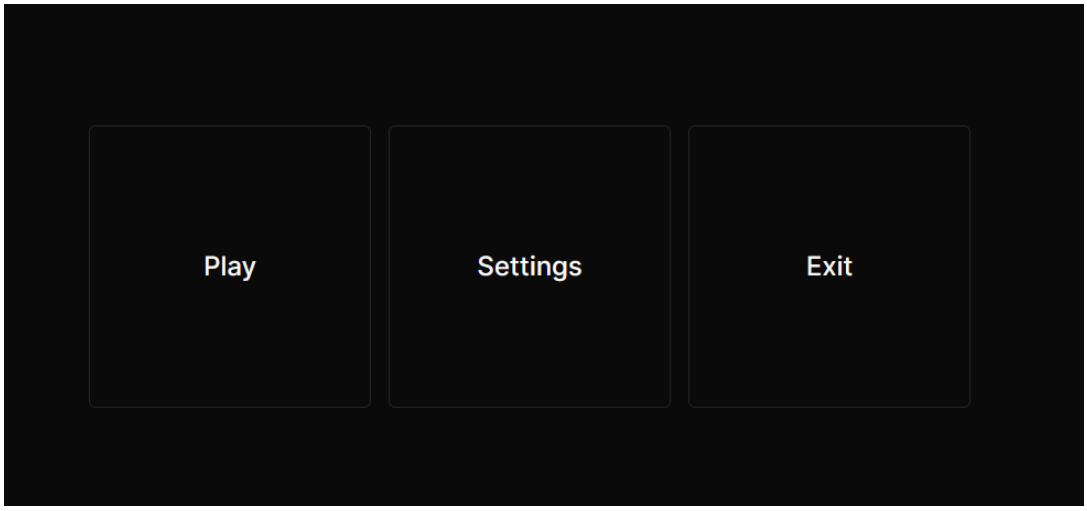
Celý frontend je napsaný v jazyce typescript, a využívá frameworky jako next.js a tailwind, pro design je využívána knihovna shadcn. Jelikož bude hra většinu času ovládána přes mobilní zařízení je potřeba udělat celé uživatelské rozhraní co nejjednodušší.

Domovní stránka

Domovní stránka je napsána velmi jednoduše a obsahuje pouze 3 tlačítka.

```tsx
// homepage... "main menu"
export default function Home() {
  return (
    <div className="overflow-x-hidden overflow-y-auto">
      <div className="flex justify-center items-center space-x-4 h-screen">
        <Button asChild variant="outline" className="h-64 w-64 text-2xl">
          <Link href="/gamemode">Play</Link>
        </Button>
        <Button asChild variant="outline" className="h-64 w-64 text-2xl">
          <Link href="/settings">Settings</Link>
        </Button>
        <Button asChild variant="outline" className="h-64 w-64 text-2xl">
          <Link href="/exit">Exit</Link>
        </Button>
      </div>
    </div>
  );
}
```

Obr. 23 Ukázka kódu domovní stránky



Obr. 24 Ukázka domovní stránky

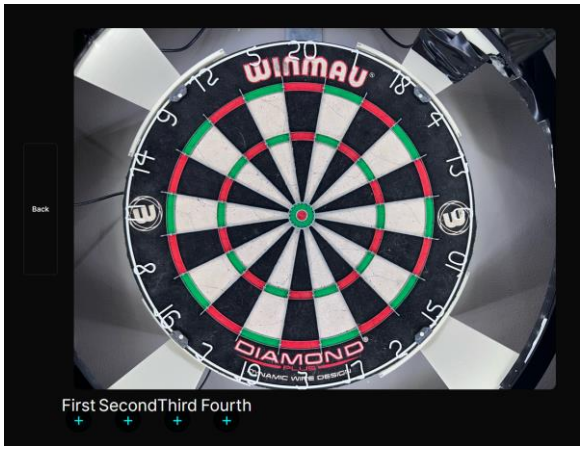
Z domovní stránky se uživatel může přemístit na začátek nové hry, do nastavení nebo vše dohromady zavřít.

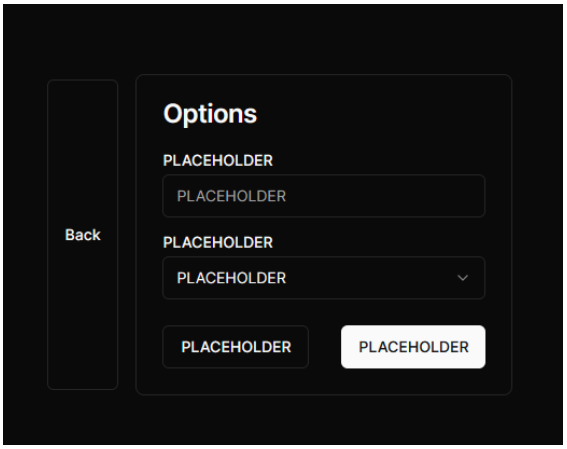
Nastavení

V nastavení si uživatel může zvolit buď menu kalibrace, nebo nastavení.

V menu kalibraci si uživatel zvolí 4 body na terči, které musí být na stejné pozici jako předem určené fotce.

Menu nastavení je zatím prázdné, a časem se do něj dají přidat všelijaké možnosti k zlepšení uživatelnosti.



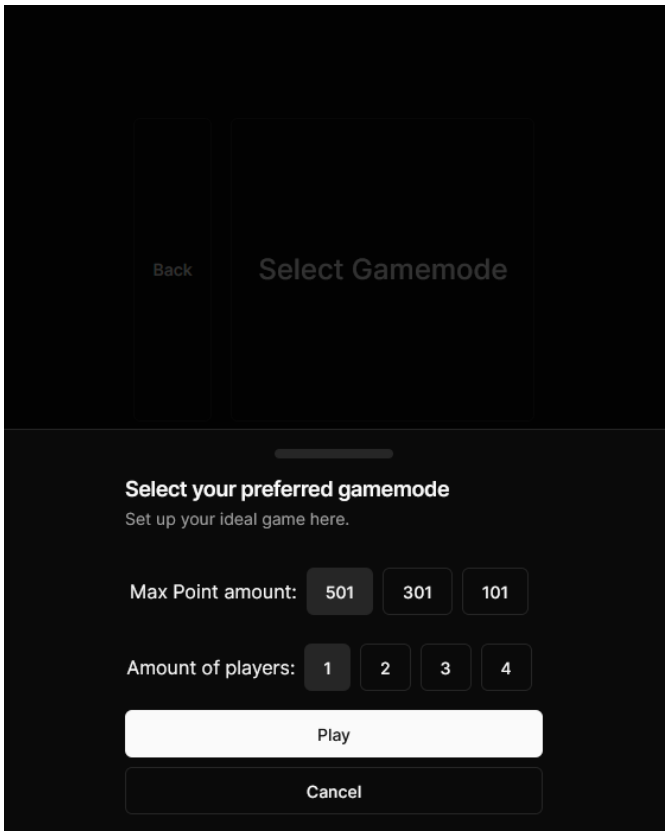


Obr. 25 Ukázka menu kalibrace

Obr. 26 Ukázka menu nastavení

Menu výběru hry

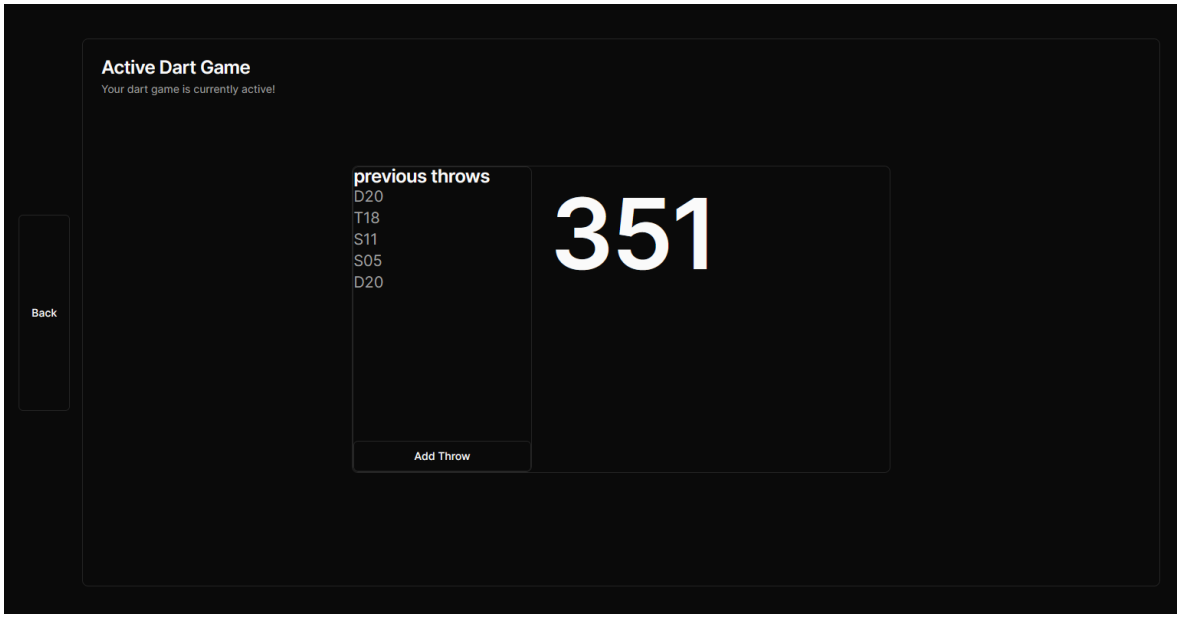
V herním menu si uživatel musí zvolit ze dvou možností, první je kolik je počet bodů od kterého se začne odečítat, a počet hráčů. Jako maximální počet hráčů jsem zvolil 4 jelikož se ve většině případě hraje ve dvou, nebo v týmech po dvou. Toto lze ale kdykoliv změnit pro jakýkoliv počet hráčů, pomocí použití rozbalujícího seznamu, nebo zaškrtávajícího políčka.



Obr. 27 Ukázka výběru hry

Herní menu

Herní menu se skládá ze třech hlavních částí. Celkové skóre, předchozí hody a tlačítko na přidání hodů, pokud se něco nepovede v kalkulaci, nebo v predikci fotky.



Obr. 28 Ukázka herního menu

Získávání dat z databáze

Získávání dat z databáze funguje využitím knihovny mongodb, kdy vytvořím spojení se serverem jako klient, specifikuji databázi a provedu úkon který zrovna potřebuji, například v této ukázce získávám všechny data z aktivní hry, ve formátu json.

```
import { MongoClient } from 'mongodb';
export const dynamic = 'force-dynamic' // defaults to auto

const mongoURI = process.env.MONGODB_URI || "default_uri"

const client = new MongoClient(mongoURI)

export async function GetActiveGame(request: Request) {
  await client.connect();
  const db = client.db(process.env.MONGODB_DB || 'default_db');
  const collection = db.collection('active_game');
  const data = await collection.find({}).toArray();

    return Response.json({ data })
  }

```

Obr. 29 Ukázka API

3.3 Testování funkčnosti

Pro testování funkčnosti aplikace jsem provedl několik testovacích scénářů zahrnujících různé situace a podmínky, aby bylo ověřeno správné chování systému v různých situacích.

Testování různých úhlů kamery

Kamera byla umístěna v různých úhlech a vzdálenostech od terče, aby byla ověřena její ideální pozice.

Zjistil jsem že na pozici kamery ani až tak moc nezáleží, záleží jenom na úhlu, ze kterého kamera sleduje terč, jelikož to nejvíce ovlivňuje, jak kamera vidí na hroty šipek.

Testování různých pozic šipek

Šipky byly házeny na terč z různých vzdáleností a pod různými úhly, včetně situací, kdy byly házeny kousek mimo terč, za sebe, vedle sebe nebo do sebe, aby byla ověřena schopnost systému detekovat a vyhodnocovat takové situace.

Model zvládl ve většině normálních případů přijít na pozici šipky, ale měl největší potíže, když byly šipky za sebou a neviděl na hrot, v 95% případů se model neodvážil tipnout kde by hrot mohl být pokud ho neviděl, ale s dalším trénováním by se toto zlepšilo.

Testování různých osvětlení

Testy byly prováděny za různých světelných podmínek, včetně situací s intenzivním osvětlením nebo stíny.

Testoval jsem se zapnutým a vypnutým světelným kruhem (Obr. 7) a zapnutým nebo vypnutým světlem v místnosti. Přišel jsem na to, že světlo v místnosti má pouze vliv, pokud je světelný kruh vypnutý. S vypnutým světelným kruhem jsem měl vysoké problémy predikovat lokaci šipek, a hlavně jejich hrotu.

Testování opakovaných hodu a stejných fotek

Prověřila se spolehlivost systému při opakovaném hodu šipek, aby bylo zajištěno, že detekce a vyhodnocování bodů jsou konzistentní.

Opakované hody a stejné fotky byly v 99% případů detekovány stejně.

3.4 Zhodnocení výsledků

Použití v domácích podmínkách

Software dokázal že použití v domácích podmínkách je funkční, ale má svoje nedostatky, které by v profesionálním prostředí prostě nemohli být.

Hlavní chyby

Během testování bylo zjištěno, že aplikace má určité nedostatky a chyby. Zejména při různých světelných podmínkách a při házení šipek za sebe nebo vedle sebe docházelo k některým chybám ve vyhodnocování bodů, například že model dokázal detekovat pouze jednu nebo dvě šipky ze tří, nebo naopak místo tří šipek detekoval čtyři, toto ale šlo v softwaru pomocí nějakých faktorů ovlivnit.

Celkově lze říct, že aplikace poskytuje užitečný nástroj pro domácí hráče šipek, avšak je důležité být si vědom možných chyb a nedostatků při použití v různých situacích. Další vylepšení, a především intenzivnější trénování modelu by mohlo přispět ke zlepšení spolehlivosti a přesnosti aplikace v budoucnu.

3.5 Dostupnost

Celý projekt je cenové velmi nenáročný, stačí jakákoliv kamera, ať už webkamera, starý foťák nebo i starý telefon, hlavní je, ať kamera umí pořizovat snímky v minimální kvalitě 1080p, tedy Full HD.

Světelný kruh a držák na kameru si člověk může doma udělat sám, nebo vymodelovat a vytisknout pomocí 3D tiskárny. Bíle LED pásky na vlepení do kruhy nejsou nijak drahé.

Pro interakci s uživatelském rozhraní lze použít jakékoliv zařízení které se dá ovládat a má přístup na internet.

Celkově bych celý projekt odhadl pod 1500 – 2000 KČ, včetně kamery a materiálů pro výrobu.

Závěr

Navržený systém pro automatizaci počítání bodů ve hře šipky s využitím neuronových sítí je ideálním řešením pro použití v domácích podmínkách. V profesionálním prostředí by byla nutností minimálně dvojice kamer pro zajištění vyšší přesnosti a spolehlivosti vyhodnocování. Přestože použití pouze jedné kamery může vést k možným chybám, jako je například chyba v predikci způsobena blízkým umístění šipek, přesněji tedy jejich hrotů. Ale pravděpodobnost těchto chyb pro amatérské uživatele je velmi malá. Pro domácí hráče šipek představuje tento systém efektivní a dostupné řešení, které zvyšuje komfort a přesnost při hře, aniž by vyžadovalo složitou infrastrukturu či vysoké náklady spojené s profesionálním vybavením.

V budoucnu lze tento systém dále rozvíjet a vylepšovat, například přidáním pokročilejších funkcí, rozšířením podpory pro více hráčů nebo integrací dalších herních prvků. Celkově lze konstatovat, že aplikace neuronových sítí pro automatizaci procesu počítání bodů ve hře šipky přináší inovativní řešení, které zlepšuje uživatelský zážitek a otevírá cestu k novým možnostem v oblasti herních aplikací.

Seznam použitých obrázků

Zdroje použitých obrázků

Obr. 1 – Obr. 29 Obrázky autora

Seznam použitých tabulek

Seznam použitých informačních zdrojů

Luo, S., Li, H., & Shen, H. (2020). Deeply supervised convolutional neural network for shadow detection based on a novel aerial shadow imagery dataset. *ScienceDirect*, 443-457.

Seznam příloh

Příloha 1 – Zdrojový kód celého projektu

Příloha 2 – Video ukázka praktické části maturitní práce