Jake Cardonick

1/13/25

3. Player Movement

Danger Score: Quantifying Space Creation in the Offensive Zone

The goal of this project is to quantify a player's ability to position themselves optimally to create high-quality scoring opportunities as the recipient of open space. By developing a "danger score," the project evaluates how a player's movement and positioning relative to the net, their teammates, and opposing defenders contribute to the creation of high-danger scoring chances. This approach focuses on understanding the spatial dynamics of hockey, emphasizing how players leverage available space to enhance offensive outcomes. Through a combination of data-driven analysis and hockey-specific context, the project provides a framework to assess a player's effectiveness in exploiting space and positioning themselves in areas most conducive to scoring success. This offers a novel way to measure the value of off-puck movement and its impact on the overall offensive strategy.

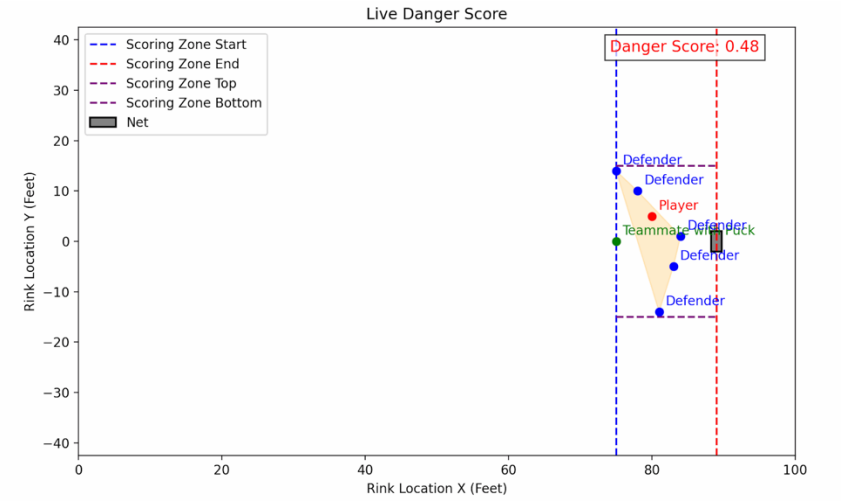
This project evaluates a player's ability to create optimal scoring opportunities by assigning a "danger score" that quantifies how effectively a player generates space and increases the likelihood of scoring as the recipient in a high-danger area. The danger score is calculated using rink coordinates to define strategic areas, specifically focusing on the offensive zone and the high-danger scoring zone near the net. The high-danger scoring zone is defined as the region between x-coordinates 75 and 89 and y-coordinates -15 to 15, aligning with areas where

offensive plays most often result in goals. The calculation of the danger score combines proximity to the net and separation from defenders. A player's position relative to the net is assessed using Euclidean distance, rewarding closer proximity to the goal (but in front of the net). At the same time, the score penalizes players who are closely marked by defenders, with separation from the nearest defender being a key factor. This balance ensures that the danger score accurately reflects a player's ability to capitalize on open space while still accounting for defensive pressure. A player must be within the defined offensive zone and high-danger scoring area, and a teammate must possess the puck within the offensive zone. These conditions ensure that the score reflects live-game situations and excludes circumstances with no offensive potential. Additionally, the methodology explicitly focuses on 5 vs. 5 situations, isolating even-strength play to eliminate biases introduced by power plays or penalty kills. This allows for a consistent evaluation of a player's performance in standard gameplay. The process integrates spatial, positional, and contextual data to produce a comprehensive metric that highlights how well players create scoring chances by effectively using space and positioning themselves as dangerous scoring threats. This approach provides actionable insights into a player's contributions to team offense and their ability to create high-quality opportunities in competitive scenarios.

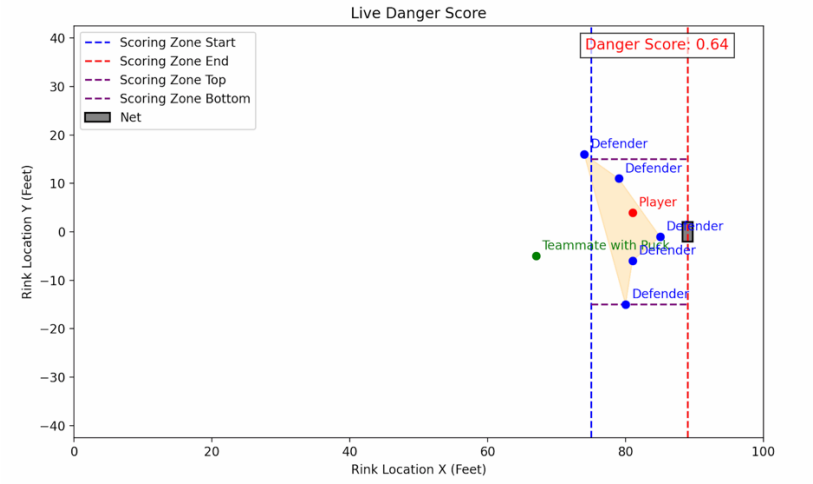
The results from the game played on 2024-11-16 between Team F and Team E demonstrate the application of the danger score model, with Player IDs 20, 89, and 4 achieving the highest cumulative danger scores of 9.82, 9.50, and 9.41 in the first period, respectively. These players exhibited the most effective positioning and ability to create scoring opportunities under the defined conditions of offensive zone play, puck possession, and defender separation.

While these findings provide insights into individual performances during this specific game, the broader significance lies in the robustness of the model itself. By evaluating player movement and scoring opportunities with a danger score grounded in spatial and contextual hockey dynamics, this approach offers a scalable framework for analyzing player contributions across games and teams. The focus remains on refining and validating the methodology to provide meaningful insights into hockey strategy and player performance.

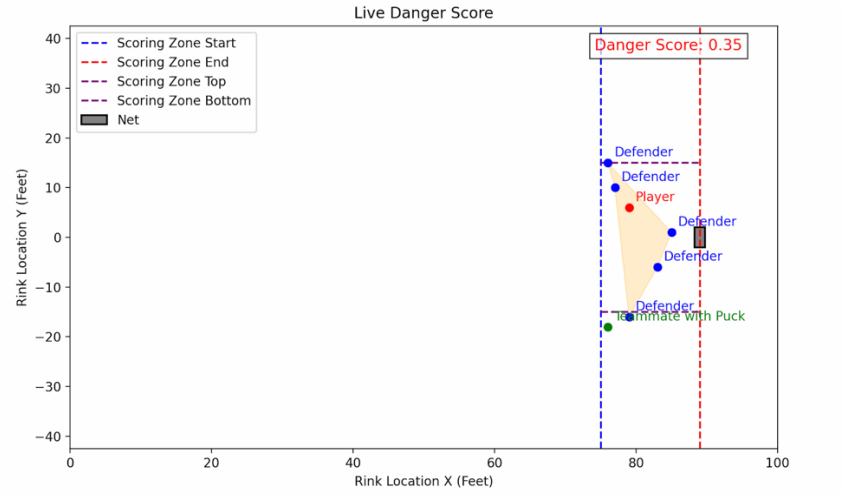
Figures: Visuals of Danger Score Calculation



The "Danger Score" of 0.48 reflects a moderate threat level, considering the proximity of the puck carrier and their teammate to the net, as well as the defensive positioning. While the puck carrier (green) and their teammate (red) are within the scoring zone, the triangular positioning of the defenders creates a barrier that limits high-quality shot opportunities. The score indicates that while a scoring chance exists, the defense's formation and coverage are likely reducing the overall likelihood of success.



The “Danger Zone” player has now gained space from the defenders. Furthermore, he has moved closer to the net. As such, the Danger Score has improved.



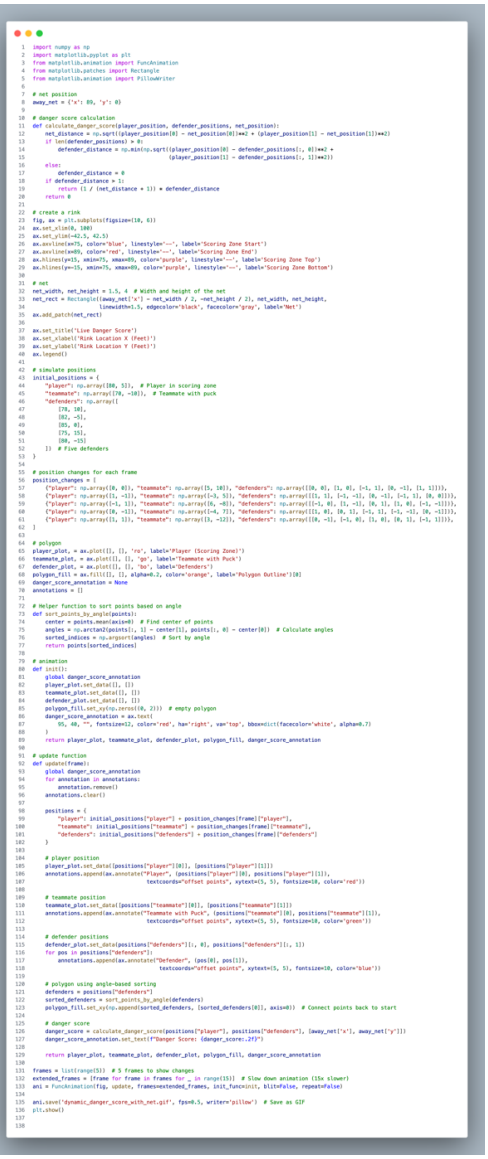
Here, the player loses separation from the defender. His danger score drops back to 0.35.

Future Analysis and Next Steps: Maintaining optimal spacing from the goalie is crucial, as being too close can eliminate passing opportunities and essentially turn the goalie into a defender. Better positioning data could help refine strategies to ensure players stay at a distance where they can capitalize on passing options while still creating pressure. Cross-ice passes remain a key tool for forcing goalie movement and opening up scoring opportunities; with more detailed data, these plays could be optimized by factoring in puck speed, angles, and defender positioning. Teammates without the puck should maintain supportive but not overcrowded spacing, ensuring they are close enough to provide options without limiting playmaking opportunities. Additionally, evaluating and ranking passing lanes could further enhance offensive efficiency by identifying the best opportunities to create high-danger scoring chances. Incorporating analyses of rebound recovery positioning and player synergies could also strengthen the ability to sustain pressure and capitalize on scoring opportunities in the offensive zone. Overall, this project just scratches the surface on how best we can evaluate a player's ability to generate danger and space in the offensive zone.

Code

```python
import os
import pandas as pd
import numpy as np

downloads_folder = os.path.expanduser("~/Downloads")
tracking_file = os.path.join(downloads_folder, "2024-11-16.Team.F.@.Team.E.-.Tracking.csv")
event_file = os.path.join(downloads_folder, "2024-11-16.Team.F.@.Team.E.-.Events.csv")  # Example Event Data file

tracking_data = pd.read_csv(tracking_file, low_memory=False)
event_data = pd.read_csv(event_file, low_memory=False)

event_data = event_data.rename(columns={
    'Clock': 'Game Clock',
    'Home_Team_Skaters': 'Home Team Skaters',
    'Away_Team_Skaters': 'Away Team Skaters'
})

# Net coordinates for both sides
home_net = {'x': 11, 'y': 0}  # Home net on the left
away_net = {'x': 89, 'y': 0}  # Away net on the right

# high-danger scoring zone
def is_in_scoring_zone(x, y):
    return (75 < x) & (x <= 89) & (-15 < y) & (y < 15)

# determine if the player is in the offensive zone
def is_in_offensive_zone(player_x, team, period):
    attacking_right = period % 2 == 1  # Odd periods: Home attacks right, Away attacks left
    if team == "Home":
        return (70 <= player_x <= 100) if attacking_right else (0 <= player_x <= 30)
    elif team == "Away":
        return (0 <= player_x <= 30) if attacking_right else (70 <= player_x <= 100)
    return False

# danger score
def calculate_danger_score(player_position, defender_positions, net_position):
    net_distance = np.sqrt((player_position[0] - net_position[0])**2 + (player_position[1] - net_position[1])**2)

    # separation from nearest defender
    if len(defender_positions) > 0:
        defender_distance = np.min(np.sqrt((player_position[0] - defender_positions[:, 0])**2 +
                                           (player_position[1] - defender_positions[:, 1])**2))
    else:
        defender_distance = 0

    # Calculate danger score
    if defender_distance > 1:
        danger_score = (1 / (net_distance + 1)) * defender_distance
        return max(0, danger_score)  # Ensure score is non-negative
    return 0

tracking_data = tracking_data.merge(
    event_data[['Game Clock', 'Period', 'Home Team Skaters', 'Away Team Skaters']],
    on=['Game Clock', 'Period'],
    how='left'
)

# 5 vs. 5
tracking_data = tracking_data[
    (tracking_data['Home Team Skaters'] == 5) &
    (tracking_data['Away Team Skaters'] == 5)
]

tracking_data['Game Clock Seconds'] = tracking_data['Game Clock'].str.split(':').apply(lambda x: int(x[0]) * 60 + int(x[1]))

player_scores = {}

# process data in segments (my computer is slow)
time_segments = [(0, 300), (300, 600), (600, 900), (900, 1200)]

for start_time, end_time in time_segments:
    valid_tracking_data = tracking_data[
        (tracking_data['Player Id'].apply(lambda x: str(x).isnumeric())) &
        (tracking_data['Game Clock Seconds'] > start_time) &
        (tracking_data['Game Clock Seconds'] <= end_time)
    ].copy()

    period = 1

    # center of the away net
    away_net_center = np.array([away_net['x'], away_net['y']])

    for _, row in valid_tracking_data.iterrows():
        if is_in_scoring_zone(row['Rink Location X (Feet)'], row['Rink Location Y (Feet)']) and \
           is_in_offensive_zone(row['Rink Location X (Feet)'], row['Team'], period):
            player_position = np.array([row['Rink Location X (Feet)'], row['Rink Location Y (Feet)']])

            # identify defenders
            defenders = valid_tracking_data[
                valid_tracking_data['Team'] != row['Team']
            ][['Rink Location X (Feet)', 'Rink Location Y (Feet)']].dropna().values

            # danger score
            danger_score = calculate_danger_score(player_position, defenders, away_net_center)

            # player's total danger score
            player_id = row['Player Id']
            if player_id not in player_scores:
                player_scores[player_id] = 0
            player_scores[player_id] += danger_score

    print(f"Cumulative Danger Scores after {end_time // 60} minutes:")
    print(pd.DataFrame(list(player_scores.items()), columns=['Player Id', 'Total Danger Score']))

player_scores_df = pd.DataFrame(list(player_scores.items()), columns=['Player Id', 'Total Danger Score'])
player_scores_df = player_scores_df.sort_values(by='Total Danger Score', ascending=False)

print("\nFinal Cumulative Danger Scores for Players in the First 20 Minutes:")
print(player_scores_df)
```

Danger Score Calculation



```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from matplotlib.patches import Rectangle
from matplotlib.animation import PillowWriter

# net position
away_net = {'x': 89, 'y': 0}

# danger score calculation
def calculate_danger_score(player_position, defender_positions, net_position):
    net_distance = np.sqrt((player_position[0] - net_position[0])**2 + (player_position[1] - net_position[1])**2)
    if len(defender_positions) > 0:
        defender_distance = np.min(np.sqrt((player_position[0] - defender_positions[:, 0])**2 +
                                           (player_position[1] - defender_positions[:, 1])**2))
    else:
        defender_distance = 0
    if defender_distance > 1:
        return (1 / (net_distance + 1)) * defender_distance
    return 0

# create a rink
fig, ax = plt.subplots(figsize=(10, 6))
ax.set_xlim(0, 100)
ax.set_ylim(-42.5, 42.5)
ax.axvline(x=75, color='blue', linestyle='--', label='Scoring Zone Start')
ax.axvline(x=89, color='red', linestyle='--', label='Scoring Zone End')
ax.hlines(y=15, xmin=75, xmax=89, color='purple', linestyle='--', label='Scoring Zone Top')
ax.hlines(y=-15, xmin=75, xmax=89, color='purple', linestyle='--', label='Scoring Zone Bottom')

# net
net_width, net_height = 1.5, 4  # Width and height of the net
net_rect = Rectangle((away_net['x'] - net_width / 2, -net_height / 2), net_width, net_height,
                     linewidth=1.5, edgecolor='black', facecolor='gray', label='Net')
ax.add_patch(net_rect)

ax.set_title('Live Danger Score')
ax.set_xlabel('Rink Location X (Feet)')
ax.set_ylabel('Rink Location Y (Feet)')
ax.legend()

# simulate positions
initial_positions = {
    "player": np.array([80, 53]),  # Player in scoring zone
    "teammate": np.array([70, -10]),  # Teammate with puck
    "defenders": np.array([
        [78, 10],
        [82, -5],
        [85, 0],
        [75, 15],
        [80, -15]
    ])  # Five defenders
}

# position changes for each frame
position_changes = [
    {"player": np.array([0, 0]), "teammate": np.array([5, 10]), "defenders": np.array([[0, 0], [1, 0], [-1, 1], [0, -1], [1, 1]])},
    {"player": np.array([1, -1]), "teammate": np.array([-1, 5]), "defenders": np.array([[1, 1], [-1, -1], [0, -1], [-1, 1], [0, 0]])},
    {"player": np.array([-1, 1]), "teammate": np.array([0, -8]), "defenders": np.array([[-1, 0], [1, -1], [0, 1], [1, 0], [-1, -1]])},
    {"player": np.array([0, -1]), "teammate": np.array([-4, 7]), "defenders": np.array([[1, 0], [0, 1], [-1, 1], [-1, -1], [0, -1]])},
    {"player": np.array([1, 1]), "teammate": np.array([3, -12]), "defenders": np.array([[0, -1], [-1, 0], [1, 0], [0, 1], [-1, 1]])},
]

# polygon
player_plot, = ax.plot([], [], 'ro', label='Player (Scoring Zone)')
teammate_plot, = ax.plot([], [], 'go', label='Teammate with Puck')
defender_plot, = ax.plot([], [], 'bo', label='Defenders')
polygon_fill = ax.fill([], [], alpha=0.2, color='orange', label='Polygon Outline')[0]
danger_score_annotation = None
annotations = []

# Helper function to sort points based on angle
def sort_points_by_angle(points):
    center = points.mean(axis=0)  # Find center of points
    angles = np.arctan2(points[:, 1] - center[1], points[:, 0] - center[0])  # Calculate angles
    sorted_indices = np.argsort(angles)  # Sort by angle
    return points[sorted_indices]

# animation
def init():
    global danger_score_annotation
    player_plot.set_data([], [])
    teammate_plot.set_data([], [])
    defender_plot.set_data([], [])
    polygon_fill.set_xy(np.zeros((0, 2)))  # empty polygon
    danger_score_annotation = ax.text(
        85, 40, "", fontsize=12, color='red', ha='right', va='top', bbox=dict(facecolor='white', alpha=0.7)
    )
    return player_plot, teammate_plot, defender_plot, polygon_fill, danger_score_annotation

# update function
def update(frame):
    global danger_score_annotation
    for annotation in annotations:
        annotation.remove()
    annotations.clear()

    positions = {
        "player": initial_positions["player"] + position_changes[frame]["player"],
        "teammate": initial_positions["teammate"] + position_changes[frame]["teammate"],
        "defenders": initial_positions["defenders"] + position_changes[frame]["defenders"]
    }

    # player position
    player_plot.set_data([positions["player"][0]], [positions["player"][1]])
    annotations.append(ax.annotate("Player", (positions["player"][0], positions["player"][1]),
                                   textcoords="offset points", xytext=(5, 5), fontsize=10, color='red'))

    # teammate position
    teammate_plot.set_data([positions["teammate"][0]], [positions["teammate"][1]])
    annotations.append(ax.annotate("Teammate with Puck", (positions["teammate"][0], positions["teammate"][1]),
                                   textcoords="offset points", xytext=(5, 5), fontsize=10, color='green'))

    # defender positions
    defender_plot.set_data(positions["defenders"][:, 0], positions["defenders"][:, 1])
    for pos in positions["defenders"]:
        annotations.append(ax.annotate("Defender", (pos[0], pos[1]),
                                       textcoords="offset points", xytext=(5, 5), fontsize=10, color='blue'))

    # polygon using angle-based sorting
    defenders = positions["defenders"]
    sorted_defenders = sort_points_by_angle(defenders)
    polygon_fill.set_xy(np.append(sorted_defenders, [sorted_defenders[0]], axis=0))  # Connect points back to start

    # danger score
    danger_score = calculate_danger_score(positions["player"], positions["defenders"], [away_net['x'], away_net['y']])
    danger_score_annotation.set_text(f"Danger Score: {danger_score:.2f}")

    return player_plot, teammate_plot, defender_plot, polygon_fill, danger_score_annotation

frames = list(range(5))  # 5 frames to show changes
extended_frames = [frame for frame in frames for _ in range(15)]  # Slow down animation (15x slower)
ani = FuncAnimation(fig, update, frames=extended_frames, init_func=init, blit=False, repeat=False)

ani.save('dynamic_danger_score_with_net.gif', fps=0.5, writer='pillow')  # Save as GIF
plt.show()
```

Animation