Simulation Models Pseudo Code/Order of Operations

Model 1: Standard Possession Time with a Markov Chain to transition between events

```
FUNCTION BuildTransitionMatrix(df, state_col):

    1. Sort events so game events are in chronological order
        a. df_sorted ← sort df by [GAME_ID, EVENTNUM]
    2. For each event, record what the next event will be by shifting the state 1 row up
        a. df_sorted.NEXT_STATE ← within each GAME_ID group, shift state_col up by
           one row
    3. Remove rows where NEXT_STATE is missing
        a. df_transitions ← drop rows in df_sorted where NEXT_STATE is null
    4. Count how many times each transition occurs (s0 to s1)
        a. transition_counts ← contingency table of (state_col vs. NEXT_STATE)
    5. Make the counts probabilities by normalizing the row
        a. transition_matrix ← for each row in transition_counts, divide by that row’s sum
    6. Return the transition matrix
```

```
FUNCTION MonteCarloGameSimulation_Markov(game_state, transition_matrix, n_simulations,
avg_possession_time):
    1. initialize home team number of wins at 0
        a. team_A_wins ← 0
    2. Repeat a loop n_simulations times
        a. Pull the score of the game for both teams
        b. If score A is > score B, team A (the home team) wins
        c. Add 1 to team_A_wins
    3. Win percent = team_A_wins/n_simulations *100
```

```
FUNCTION EvaluateMultipleGames(df, transition_matrix, n_games, n_simulations,
avg_possession_time, random_state):
    1. Set random seed for replicability
    2. Initialize empty list
    3. Get all game ids
    4. Calculate length of all game ids
```

```
5. Take a random sample of game ids without replacement
6. For game in game ids
    a. Filter to just that games data
    b. Make sure time remaining is valid and positive
    c. Filter to 4th quarter events with more than 30 seconds left
    d. Pick a random event from that game
    e. Build the game state dictionary, including information on PERIOD,
       TIME_REMAINING, SCOREMARGIN_NUM, current_event, SCORE_A,
       SCORE_B
    f. Simulate forward to get win probability
    g. Pull the actual outcome from the game from play_by_play_df
    h. Compare simulation to reality, indicating a 1 if simulation was correct, a 0 if
       incorrect
    i. Append results to empty list initialized above and convert to dataframe
7. Return dataframe from 6i
```

Model 2: Dynamically Calculated Possession Time with a Markov Chain to transition between events

```
FUNCTION BuildWeightedTeamTransitionMatrix(df, team_id,
state_col='SCOREUPDATEEVENT', simulation_date):
    1. Filter to rows that include TEAM_ID
    2. Parse dates and filter to just dates that happened before the game in question
    3. Make a list of unique game IDs and game dates
    4. Split into groups by number of games since the game in question (5, 10, 15)
    5. Build a transition matrix for each of the last groups created in 4
    6. Weight the transition matrices, 5 games * 0.5, 6-10 games *0.25, 11-15 games * 0.25
    7. Union the matrices states
    8. Combine with weights
    9. Return combined transition matrix
```

```
FUNCTION GetTeamPossessionTimes(df, team_id, simulation_date):
    1. Filter to shooting events (is a 1 or 2 in play_by_play_df)
    2. Filter to only team A's events
    3. Use helper function to compute time remaining
    4. Sort by EVENTNUM
    5. Compute difference in time between possession events, storing them in an array
    6. Return the array of time differences between shots
```

```
FUNCTION GetWeightedTeamPossessionStats(df, team_id, simulation_date):
    1. Copy and parse dates
    2. Find last 25 games IDs for team before this game
    3. Split into the three groups (5, 10, 15 games) - trying to inflate/deflate for hot/cold streaks
    4. Get mean/ standard dev for each of the 3 groups possession times
    5. Collect non-null groups with weights
    6. Just in case this function isn't working on a game, fallback to possession time = 18 (or 20)
    7. Compute weighted mean and standard deviation of possession time, weighting by 0.5 of previous 5 games, 0.25 of games 6-10, 0.25 of games 11-15
    8. Return weighted mean and standard dev possession
```

```
FUNCTION SimulateGameDynamicTeam(game_state, home_stats, visitor_stats, avg_possession_time_fallback=18):
    1. Initialize clock and scores for the game from play_by_play_df
    2. Initialize empty list to store future events in the game
    3. Use helper to sample next event from transition matrix
        a. Take current event, use weighted probability for next event
    4. Draw from normal distribution with weighted mean and standard dev of possession times, using GetWeightedTeamPossessionStats
    5. While time_remaining>0
        a. keep simulating home events
            i. Append to list
        b. Keep simulating away events
            i. Append to list
    6. Calculate score from list
    7. Return scoreA, scoreB, event list
```

```
FUNCTION EvaluateMultipleGamesGeneric(df, n_games = 100, n_simulations = 200, avg_possession_time_fallback = 18, random_state = 42, use_fourth_quarter = TRUE):
    1. Set random seed and initialize empty list
    2. Choose a sample of random games
    3. Loop over each sampled game
        a. Get a list of games events
        b. Get the date and team ids in the game
        c. Build team specific transition matrices and mean/stdev of possession times
        d. Select a starting event
        e. Build the initial game state
        f. Run simulations
        g. get actual game outcomes and indicate the simulation prediction as correct/incorrect
        h. Append the results to the initialized empty list
    4. Return dataframe of results and prediction of each simulated game
```

Model 3: Semi Markov Chain with lognormal possession distributions

```
FUNCTION GetTeamPossessionTimesRaw(df, team_id, simulation_date):
    1. Filter to only made/missed field goal attempts
    2. Filter to just the team’s events
    3. Pull game date and make sure it’s on or before the simulation date
    4. Sort by Game ID, PERIOD, TIME_REMAINING
    5. Loop through each Game ID
        a. Create a list of possession times
    6. Return array of possession times for the team

FUNCTION FitHoldingTimeDistribution(times, dist_names):
    1. If length of times is way too small to make a valid distribution, give an error
    2. Initialize best_ks at infinity
    3. Initialize best parameters and best distribution as none
    4. Loop through all possible dist_names (gamma, weibull, lognorm)
        a. Calculate the KS for the array of times
        b. Store the best KS, best distribution, and best parameters for that distribution
    5. Return the best KS, best distribution, and best parameters for that distribution

FUNCTION GetWeightedTeamHoldingTimeDistribution(df, team_id, simulation_date):
    1. Filter to team’s rows before simulation date
    2. Filter to last 25 game IDs
    3. Define the time horizon groups (5 games, 10 games, 15 games
    4. Collect times with weights
    5. Fit distribution on weighted samples
    6. Return the best KS, distribution, and parameters for that team id and date

FUNCTION SimulateSMPGame(game_state, home_stats, visitor_stats, avg_pos_fallback=18,
min_pos=3):
    1. Compute total time remaining
    2. Initialize scores and game state (last event)
    3. While loop until time runs out
        a. Simulate next event
        b. Change possession as necessary
        c. Calculate score
        d. Run down clock from possession time distribution
    4. Return scoreA, scoreB, event log

FUNCTION EvaluateMultipleGamesSMP(df, n_games=100, n_sim=100, avg_pos_fallback=18,
random_state=42, use_q4=TRUE):
    1. ensure GAME_DATE, SCOREUPDATEEVENT, TIME_REMAINING,
       SCOREMARGIN_NUM exist (compute if needed)
    2. Pull sample game IDs
    3. Loop through game IDs
        a. Determine who is home/away
        b. Build team models
```

   c. Pick start event
      i. If use_4q=TRUE
         1. Then make sure it’s a 4q event with more than 30 sec remaining
   d. Simulate
   e. Capture the simulated prediction and compare it to the actual outcome

4. Return a dataframe with all simulated game outcomes, the real game outcome, and whether or not that prediction was true