Approach Document: GreedyGame (Session Calculation)

1. Open the log file in Notepad++ and perform following steps to align the data in standard JSON format:
   a. Add comma at the beginning of all the lines (except the first line)
   b. Enclose the complete data set in square brackets.
2. Create a table with following schema to dump the data present in log file:

```
CREATE TABLE [dbo].[Test](
    [ai5] [nvarchar](32) NULL,
    [event1] [nvarchar](10) NULL,
    [id] [int] NULL,
    [timestamp] [datetime2](7) NULL
)
```

   We eliminated sdkv, debug and random attributes because they did not contribute to calculation of sessions and their averages. However, they provide us some insights about the users and the devices they use. Dump data into this table.

3. Further create a table with following schema and insert data into new table, using following script.

```
CREATE TABLE [dbo].[Test2]
(
[ai5] UNIQUEIDENTIFIER,
[event1] [nvarchar](10) NULL,
[id] [int] NULL,
[timestamp] [datetime2](7) NULL
)

INSERT INTO [dbo].[Test2]
(
[ai5]
,[event1]
,[id]
,[timestamp]
)
SELECT
CAST(
    SUBSTRING([ai5], 1, 8) + '-' + SUBSTRING([ai5], 9, 4) + '-' + SUBSTRING([ai5], 13, 4) + '-' +
    SUBSTRING([ai5], 17, 4) + '-' + SUBSTRING([ai5], 21, 12)
    AS UNIQUEIDENTIFIER) AS [ai5]
,[event1]
,[id]
,[timestamp]
FROM [dbo].[Test] WITH (NOLOCK)
```

   Storing [ai5] as guid is preferable since, it is only 16 bytes and storing it in varchar or nvarchar would require 32 bytes atleast. Therefore, if the table is large, saving 16 bytes (32 -16) per row is considerable.

4. For each game, device and start time, calculate the minimum stop time. This is done by creating a self-join on the base table, joining them on game-id and device-id respectively, such that the first table contains data for start time only and the second table for stop time only. Also, we need a condition on timestamp such that the stop time is greater than the start time. Summarizing above, we need to find the minimum of the stop time which is just greater than a start time.

```
IF OBJECT_ID(N'tempdb..#MinimumStop', N'U') IS NOT NULL
BEGIN
    DROP TABLE #MinimumStop
END;

SELECT T1.id
    ,T1.ai5
    ,T1.TIMESTAMP AS EventStart
    ,MIN(T2.TIMESTAMP) AS EventStop
INTO #MinimumStop
FROM dbo.Test2 T1
INNER JOIN dbo.Test2 T2
    ON T1.id = T2.id
        AND T1.ai5 = T2.ai5
WHERE T1.TIMESTAMP < T2.TIMESTAMP
    AND T1.EVENT1 = 'ggstart'
    AND T2.event1 = 'ggstop'
GROUP BY T1.id
    ,T1.ai5
    ,T1.TIMESTAMP
ORDER BY T1.id
    ,T1.ai5
    ,T1.TIMESTAMP
```

5. Next, we need to consider the minimum event start for each event stop. This is achieved through CTE that includes the time difference in seconds between start time and stop time. Further, it uses the window function ROW_NUMBER to sort the start times with respect to stop time.

```
IF OBJECT_ID(N'tempdb..#MinimumStart', N'U') IS NOT NULL
BEGIN
    DROP TABLE #MinimumStart
END;

WITH CTE
AS (
    SELECT id
        ,ai5
        ,EventStart
        ,EventStop
        ,DATEDIFF(S, EventStart, EventStop) AS TimeDiff
        ,ROW_NUMBER() OVER (
            PARTITION BY id
            ,ai5
            ,EventStop ORDER BY EventStart ASC
            ) RNStop
    FROM #MinimumStop
    )
SELECT id
    ,ai5
    ,EventStart
    ,EventStop
    ,TimeDiff
INTO #MinimumStart
FROM CTE
WHERE RNStop = 1
```

6. Further we use the LEAD function to get the difference (Consecutive Difference) between current row stop and next row start. Self-join could have been used in place of LEAD function, but LEAD function directly accesses the row at the next physical offset, thereby reducing the overhead of self-join.
   It is good to note the ISNULL statement in the query below, which gives a result of 31, if there is no following row. This means the last row for game-device combination will get a value of 31 (> 30, so will match the criteria of a row that should be an end time in the result set). This step also introduces a ROW NUMBER ordering based on EventStart, which is used and

explained in next step.

```
IF OBJECT_ID(N'tempdb..#ConsecutiveDiff', N'U') IS NOT NULL
BEGIN
    DROP TABLE #ConsecutiveDiff
END;


SELECT ID
    ,ai5
    ,EventStart
    ,EventStop
    ,TimeDiff
    ,ROW_NUMBER() OVER (
        PARTITION BY ID
        ,ai5 ORDER BY EventStart ASC
        ) RNStart
    ,ISNULL(DATEDIFF(ss, EventStop, lead(EventStart) OVER (
                PARTITION BY id
                ,ai5 ORDER BY EventStart
                )), 31) AS CD
INTO #ConsecutiveDiff
FROM #MinimumStart
```

7. Next, as mentioned in the problem statement document, for consecutive time difference less than (or equal to) 30 seconds, the sessions are considered to be one. For this, we need to find the EventStop for which ConsecutiveDifference is > 30.
   The ROW NUMBER introduced in the above query is used to find the next consecutive row based on gameid-device combination.

```sql
IF OBJECT_ID(N'tempdb..#CalcFinalStop', N'U') IS NOT NULL
BEGIN
    DROP TABLE #CalcFinalStop
END;

SELECT t1.id
    ,t1.ai5
    ,t1.EventStart
    ,MIN(MinStop.EventStop) AS EventFinalStop
INTO #CalcFinalStop
FROM #ConsecutiveDiff t1
LEFT JOIN #ConsecutiveDiff t2
    ON t1.id = t2.id
        AND t1.ai5 = t2.ai5
        AND (t1.RNStart - 1) = t2.RNStart
INNER JOIN #ConsecutiveDiff MinStop
    ON t1.id = MinStop.id
        AND t1.ai5 = MinStop.ai5
        AND MinStop.EventStop > t1.EventStart
        AND MinStop.CD > 30
WHERE t1.RNStart = 1
    OR t2.CD > 30
GROUP BY t1.id
    ,t1.ai5
    ,t1.EventStart
ORDER BY t1.id
    ,t1.ai5
    ,t1.EventStart;
```

8. Next, we use a CTE say ***Final,*** which sums the session time of all the sessions combined into one, i.e. all those sessions whose consecutive difference is less than (or equal to) 30. On basis on this, we also calculate the [IsValid] flag which is set to True, only when SUM of a session time exceeds 60 seconds.
   Further this CTE (***Final***) is used to calculate the 'Total Session', 'Valid Session' and 'Valid Session Average'.

```sql
IF OBJECT_ID(N'[tmp].[GameSession]', N'U') IS NOT NULL
BEGIN
    DROP TABLE [tmp].[GameSession]
END;

WITH Final
AS (
    SELECT t6.id
        ,t6.ai5
        ,t6.EventStart
        ,T6.EventFinalStop
        ,SUM(t5.TimeDiff) AS TIME
        ,CASE
            WHEN SUM(t5.TimeDiff) > 60
                THEN 1
            ELSE 0
            END AS IsValid
    FROM #CalcFinalStop T6
    INNER JOIN #ConsecutiveDiff T5
        ON T6.ID = T5.ID
            AND T6.AI5 = T5.AI5
            AND T6.EventFinalStop >= t5.EventStop
    GROUP BY t6.id
        ,t6.ai5
        ,t6.EventStart
        ,T6.EventFinalStop
    )
SELECT ID AS GameID
    ,COUNT(ID) AS 'Total Session'
    ,COUNT(CASE
            WHEN IsValid = 1
                THEN 1
            ELSE NULL
            END) AS 'Valid Session'
    ,ISNULL((
            SUM(CASE
                    WHEN IsValid = 1
                        THEN TIME
                    ELSE NULL
                    END) / COUNT(CASE
                    WHEN IsValid = 1
                        THEN 1
                    ELSE NULL
                    END)
            ), 0) AS 'Session Average'
INTO [tmp].[GameSession]
FROM Final
WHERE TIME > 0
GROUP BY ID
ORDER BY COUNT(ID) DESC
```

9. Select details from final physical table created.

```
22  SELECT [GameID]
23        ,[Total Session]
24        ,[Valid Session]
25        ,[Session Average]
26  FROM [dbo].[GameSession] WITH (NOLOCK)
27
```

100 %

Results | Messages

	GameID	Total Session	Valid Session	Session Average
1	30900473	662	500	375
2	35643983	4	2	308
3	40904844	7	6	320
4	55107008	14032	12007	689
5	59561974	1	1	112
6	10655437	1710	1216	459
7	32950170	151	111	474
8	19049241	80	50	634
9	60374084	20	14	269
10	98010070	6	5	134
11	21554188	19	16	363
12	26962869	1	0	0
13	63960431	4	3	213
14	43346372	1921	1585	592
15	18121481	2534	1533	506
16	67544688	55	28	380
17	10483946	116	85	534